TRABALHO DE GRADUAÇÃO

Detectando Deepfakes em vídeos: Uma abordagem utilizando redes neurais convolucionais residuais

Christian Cruvinel França

Brasília, maio de 2021

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

Detectando Deepfakes em vídeos: Uma abordagem utilizando redes neurais convolucionais residuais

Christian Cruvinel França

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Flávio de Barros Vidal, CIC/UnB _____
Orientador

Prof. Alexandre Ricardo Soares Romariz, ENE/UnB _____
Membro Interno

PCF MsC. Evandro Lorens - INC/PF _____
Membro Externo

Brasília, maio de 2021

FICHA CATALOGRÁFICA

CHRISTIAN, CRUVINEL FRANÇA
Detectando Deepfakes em vídeos: Uma abordagem utilizando redes neurais convolucionais residuais,

[Distrito Federal] 2021.

xix, 87p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2021). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1. Redes Neurais Convolucionais
2. Deepfake
3. Classificação
4. Visão Computacional

I. FT/UnB
II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

FRANÇA, C., (2021). Detectando Deepfakes em vídeos: Uma abordagem utilizando redes neurais convolucionais residuais. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$01, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 87p.

CESSÃO DE DIREITOS

AUTOR: Christian Cruvinel França

TÍTULO DO TRABALHO DE GRADUAÇÃO: Detectando Deepfakes em vídeos: Uma abordagem utilizando redes neurais convolucionais residuais.

GRAU: Engenheiro ANO: 2021

Christian Cruvinel França

Rua 8 Chácara 328 Lote 18 - Setor Habitacional Vicente Pires.

72007-138 Brasília – DF – Brasil.

Dedicatória

Dedico este trabalho a todos aqueles que, em sua trajetória dentro da universidade, se sentiram insuficientes em seus respectivos cursos de graduação. O amadurecimento acadêmico e profissional chega em diferentes épocas para todos, e alguém jamais deve ser julgado por não ter se encontrado ainda. É um caminho diferente para todos nós e devemos respeitá-lo.

Christian Cruvinel França

Agradecimentos

Agradeço a todos que fizeram parte da minha jornada dentro da universidade e fora dela. Foram muitos momentos expressos em conversas, conquistas e questionamentos. Conheci pessoas incríveis desde o início desse trajeto e muitas outras no caminho.

Agradeço aos meus bons amigos de infância que sempre me alegraram em dias ruins e sempre estiveram ao meu lado nas várias facetas da vida nesses últimos 6 anos de faculdade. Em ordem alfabética: Andrei César, Brunno Garcia, Celso Arruda, João Marcos, Wilson Germano, Yuri Pernambuco... Muito obrigado por seu apoio e sua preocupação.

Agradeço também os bons amigos que conheci no meu curso: Carlos Rocha, Danielle Almeida, Estanislau Dantas, Filipe Maia, Guilherme Caetano, Kenneth Lui, Myllena de Almeida, Tiago Galo... Para citar apenas alguns dos nomes, muito obrigado por todo o seu apoio e bons conselhos na minha trajetória acadêmica.

Agradeço também aos meus bons amigos e professores de dança de salão: Carol Andrade e Wesley Andrade, que tive o prazer de conhecer na UnB. Vocês juntos me proporcionaram conhecer um mundo incrível que foi vital para enfrentar os dias difíceis da UnB e tão vital quanto para alegrar ainda mais os dias felizes.

Um agradecimento ao professor Flávio de Barros Vidal por ter sido sempre tão solícito e ter estado sempre presente durante todo o desenvolvimento do meu trabalho de graduação. Sua dedicação com seus alunos orientados é notória em todos os âmbitos. Não houve um único momento durante o desenvolvimento deste trabalho em que não me senti apoiado e que não senti que não poderia contar com o senhor.

Um agradecimento especial à minha namorada, Juliana Félix, que tem sido meu pilar desde o meu quarto ano de faculdade e por estar me proporcionando uma das melhores fases da minha vida. Foram muitos momentos de carinho, compaixão e empatia proporcionados que me deram forças para continuar seguindo em frente rumo a acreditar em futuro glorioso ao seu lado. Cada palavra de apoio e determinação já dita por você para mim será guardada com carinho durante toda minha vida.

Um último agradecimento especial à minha mãe, Marlúcia Souza Cruvinel, cujo apoio tornou toda a minha jornada na UnB possível. Nunca me deixou faltar o que fosse necessário para poder correr atrás dos meus objetivos acadêmicos, e por isso e muito mais que jamais serei capaz de expressar neste texto. Eu te agradeço de todo coração.

Christian Cruvinel França

RESUMO

O grande crescimento de poder computacional nos últimos anos vem acompanhado de uma maior acessibilidade de pessoas comuns à hardware de alto desempenho no quesito de processamento paralelo. A popularização do uso do aprendizado de máquina também abre espaço para essas mesmas pessoas explorarem seus equipamentos neste ramo. Um dos ramos é a criação de *deepfakes*, isto é, vídeos contendo um ou mais rostos humanos trocados cuja troca foi necessariamente realizada por um algoritmo de inteligência artificial, possuindo muitas vezes capacidade suficiente para enganar o olho humano. Isso abre caminho para pessoas de má índole realizarem *deepfakes* em vídeos de indivíduos com significativa importância pública, danificando suas imagens e espalhando desinformações. Com esse fato em mente, torna-se importante a capacidade de identificá-los corretamente. Este trabalho propõe o treinamento de uma rede convolucional residual profunda a fim de detectar a presença de um *deepfake*. Para este fim, inicialmente, foi necessário obter uma base de dados de vídeos com vários exemplos tanto de vídeos apresentando rostos inalterados quanto de vídeos apresentando *deepfakes* para o treinamento da rede. Feito isso, a primeira etapa se caracteriza pela remoção da face do indivíduo sob diferentes instantes de tempo no vídeo e a divisão destas entre faces reais e faces forjadas. Na segunda etapa temos o treinamento de uma rede convolucional residual para a classificação dos rostos extraídos. Temos então na terceira etapa o teste da arquitetura treinada da rede convolucional residual final diretamente na detecção dos vídeos contendo *deepfakes*. Foi demonstrado que uma rede neural convolucional é uma opção viável na tarefa de detecção de *deepfakes*.

Palavras Chave: Redes Neurais Residuais, Redes Neurais Convolucionais, Deepfake, Visão Computacional, Classificação

ABSTRACT

The significant growth in computing power in recent years has been accompanied by greater accessibility of ordinary people to high-performance hardware in parallel processing. The popularization of machine learning also opens space for these same people to explore their equipment in this field. One of the possible applications is the creation of *deepfakes*, that is, videos containing one or more exchanged human faces whose exchange was necessarily carried out by an artificial intelligence algorithm, often having sufficient capacity to deceive the human eye. This opens the way for people of bad character to perform *deepfakes* on individuals of significant public importance, damaging their images and spreading misinformation. With that fact in mind, the ability to correctly identify them becomes important. This work proposes training a deep residual convolutional neural network model to detect the presence of a *deepfake*. For this purpose, initially, it was necessary to obtain a database of videos with several examples, both of videos showing entire faces and videos presenting *deepfakes* for the network's training. That done, the first stage is characterized by the removal of the individual's face under different moments in the video and the division of these between real faces and forged faces. In the second stage, we have the training of a residual convolutional network to classify the extracted faces. In the third stage, we have then tested the final trained architecture for the residual convolutional network directly in the detection of the *deepfake* videos. It was demonstrated that a convolutional neural network is a viable option in detecting *deepfakes*.

Keywords: Residual Neural Networks, Convolutional Neural Networks, Deepfake, Computer Vision, Classification

SUMÁRIO

1 Introdução 1

1.1 Contextualização 1

1.2 Descrição do problema e objetivos 3

1.2.1 Objetivo Geral 3

1.2.2 Objetivos Específicos 3

1.3 Apresentação do manuscrito 3

2 Fundamentos teóricos e Trabalhos Relacionados 4

2.1 Aprendizado Profundo 4

2.1.1 Histórico 4

2.1.2 Principais Definições de Aprendizado Profundo 8

2.2 *DeepFakes* 33

2.2.1 Principais Definições sobre *Deepfakes* 33

2.2.2 Principais Trabalhos Relacionados sobre *Deepfakes* 34

3 Metodologia 37

3.1 Elaboração da Base de Dados 38

3.1.1 *Deepfake Detection Challenge* 38

3.2 Definição da Estratégia de Aprendizagem Profunda 42

3.2.1 Estratégia de Treinamento 42

3.2.2 Descrição do Modelo Utilizado 43

3.2.3 Ajustes do Modelo (Validação Cruzada) 45

3.3 Avaliação dos Resultados 46

3.3.1 Métodos de Validação dos Resultados 47

4 Resultados 49

4.1 Hardware de Treinamento 49

4.2 Base de Dados de Imagens 49

4.2.1 Quantidades e Dimensões 49

4.2.2 Partições para Validação Cruzada 51

4.2.3 Pré-processamento das Imagens 52

4.3 Hiperparâmetros 53

4.4 Treinamento e Validação do Modelo 55

4.4.1 Partição 1 56
4.4.2 Partição 2 58
4.4.3 Partição 3 60
4.4.4 Partição 4 62
4.4.5 Partição 5 64
4.4.6 Conjunto de dados completo 66
4.4.7 Curvas ROC 68
4.5 Análise dos Resultados da Validação Cruzada 68
4.6 Análise dos Resultados do Treinamento no Conjunto Final 70
4.7 Etapa Final de Validação 72
4.8 Tempo de Inferência para um Novo Vídeo 76

5 Conclusões 77
5.1 Perspectivas Futuras 78

REFERÊNCIAS BIBLIOGRÁFICAS 80

Anexos 86

I Programas utilizados 87

LISTA DE FIGURAS

1.1 *Deepfake* de Mark Zuckerberg publicado na rede social *instagram* em julho de 2019. 2

2.1 Um diagrama de Venn mostrando como o aprendizado profundo (Deep learning) pertence ao grupo de aprendizagem de representação (Representation learning), que por sua vez é uma área do aprendizado de máquina (Machine learning). Observa-se também que consiste apenas uma das várias abordagens que se têm disponíveis para a inteligência artifical (AI) [1]. 6

2.2 Uma unidade perceptron simples. As entradas $\boldsymbol{x} = [x_1, x_2, x_3]$ são individualmente multiplicadas pelos seus respectivos pesos $\boldsymbol{\omega} = [w_1, w_2, w_3]^t$ e os resultados são então somados com a multiplicação entre uma entrada unitária e um parâmetro pertinente à própria unidade perceptron chamado *bias* ($\boldsymbol{b} = [b_1]$), passando então por uma função de ativação (aqui representada como uma função sigmoide $\sigma(x)$) final que introduz uma não-linearidade (melhor discutido na Seção 2.1.2.2). O conjunto de parâmetros aprendíveis $\boldsymbol{\omega}$ e $\boldsymbol{b}$ é o conjunto total de parâmetros aprendíveis $\boldsymbol{\theta}$. A entrada *bias*, neste caso, pode ser definida tanto como uma entrada de valor b_1 e peso de valor 1 quanto uma entrada de valor 1 e peso de valor b_1. 8

2.3 Um modelo perceptron multicamadas de 2 camadas, possuindo 2 unidades na primeira camada e 1 unidade na segunda. As entradas $\boldsymbol{x} = [x_1, x_2, x_3]$ são individualmente multiplicadas por pesos associados a cada unidade perceptron dentro da camada. As saídas após as funções de ativação da primeira camada são interpretadas como novas entradas pela camada seguinte. Ao final das operações, a saída $\hat{y}$ é comparada com a classe ou valor correto esperado y utilizando uma função de custo $L(\hat{y}, y)$..... 9

2.4 A **retropropagação**, utilizando a regra da cadeia, propaga os gradientes para as camadas anteriores da rede neural a partir da função de custo $L(y, \hat{y}_{21})$ obtida na saída comparando-se o resultado da rede com o resultado esperado. Com os gradientes, utiliza-se o algoritmo de **descida de gradiente** para atualizar os pesos a cada nova etapa de aprendizado. 12

2.5 Exemplos de (a) *underfitting*, (b) aproximação ideal e (c) *overfitting*. Um modelo de baixa capacidade como na figura (a), sendo linear, não é capaz de aproximar a função o suficiente. Um modelo de grau 4 como o da figura (b) possui capacidade suficiente para aproximar a função quase que com perfeição. Um modelo de grau 15 possui exagerada capacidade para a tarefa e, embora seja capaz de aceitar todos as amostras do treinamento, não é capaz de generalizar corretamente para novar entradas [2]. 13

2.6 Curvas de aprendizado mostrando como valor da função de custo log-verossimilhança negativa muda ao longo do tempo (indicado neste caso como o número de épocas). A curva azul representa o custo no conjunto de dados de treinamento e a curva verde representa o custo no conjunto de dados de teste. Neste exemplo, o modelo é uma rede neural. Observa-se que o valor de custo para o conjunto de treinamento decresce continuamente porém o valor de custo para o conjunto de teste decresce até certo ponto (mais ou menos até a época 25) e então começa a crescer [1]. 19

2.7 *Dropout* treina um conjunto que consiste em todas as sub-redes que podem ser construídas removendo unidades de uma rede de base subjacente. Aqui, começa-se com uma rede base com duas camadas, possuindo duas unidades na primeira camada e uma na segunda. São dezesseis possíveis subconjuntos dessas quatro unidades. Mostramos todas as dezesseis sub-redes que podem ser formadas eliminando diferentes subconjuntos de unidades da rede original. Neste pequeno exemplo, uma grande proporção das redes resultantes não tem unidades de entrada ou nenhum caminho conectando a entrada para a saída. Este problema se torna insignificante para redes com camadas largas, onde a probabilidade de descartar todos os caminhos possíveis de entradas para saídas torna-se menor [1]..... 21

2.8 Um exemplo da operação de convolução em um tensor bidimensional. A saída aqui está restrita apenas às posições a qual o kernel permanece inteiramente dentro da imagem, caracterizando uma convolução "válida" em alguns contextos. As setas indicam como a metade superior esquerda do tensor de saída é formada aplicando o kernel à correspondente região superior esquerda do tensor de entrada [1]..... 22

2.9 Eficiência em detecção de bordas. A imagem na direita foi formada obtendo cada pixel na imagem original e subtraindo o valor do seu pixel vizinho à esquerda. Isso exalta todas as bordas alinhadas verticalmente. Ambas as imagens possuem 280 pixels de altura. A imagem da esquerda possui 320 pixels de largura, enquanto a da direita possui 319 pixels de largura. Essa transformação pode ser descrita por um kernel de convolução contendo dois elementos, e requer $319 \times 280 \times 3 = 267,960$ operações em ponto flutuante (duas multiplicações e uma adição por pixel de saída) para se computar a saída utilizando convolução. Para descrever a exata mesma transformação utilizando multiplicação matricial necessitaria $320 \times 280 \times 319 \times 280$, ou mais ou menos oito bilhões, de entradas na matriz, tornando a convolução quatro bilhões de vezes mais eficiente para representar esta transformação [1]. 23

2.10 Os componentes de uma rede neural convolucional típica. Existem dois conjuntos de terminologias comumente utilizadas. Na terminologia da esquerda, a rede convolucional é vista como um número pequeno de camadas relativamente complexas, com cada camada possuindo vários "estágios". Na terminologia da direita, a rede é vista como um grande número de camadas mais simples. Na terminologia da direita, nem toda camada possui parâmetros [1]..... 24
2.11 Bloco residual. Considerando os parâmetros da primeira camada como W_1, os parâmetros da segunda camada com W_2 e a entrada como x, a saída do bloco residual será dada por $y = W_2 r(W_1 x) + x$, onde r é a função $ReLU$ nesse caso [3]... 26
2.12 Camadas convolucionais da rede proposta em [4], onde (a) corresponde à primeira camada convolucional da rede, (b) corresponde à segunda camada convolucional da rede e (c) corresponde à quarta camada convolucional da rede. Observa-se que a primeira camada se responsabiliza por reconhecer características de mais baixo nível da imagem como arestas. Conforme se adentra a rede, as características reconhecidas começam a se tornar mais complexas. Na camada 4, já se observa a presença de padrões mais específicas da tarefa em questão [4]..... 29
2.13 Matriz de confusão. A quantidade de verdadeiros positivos encontrados é localizada na diagonal superior esquerda, a quantidade de falsos negativos na diagonal direita superior, a quantidade de falsos positivos na diagonal esquerda inferior e a quantidade de verdadeiros negativos na diagonal direita inferior..... 30
2.14 Exemplo de curva *ROC*. Avaliação de três diferentes preditores de epítopos de HIV. A curva tracejada indica uma classificação randômica. Quanto mais para a esquerda a curva do classificador se encontra, melhor é sua capacidade preditiva como um todo. 32
2.15 Exemplos de métodos de manipulação facial. À esquerda tem-se os métodos de manipulação de identidade facial (grupo o qual o *Deepfake* está incluso) e à métodos de manipulação de expressão facial [5]. 33
2.16 A imagem de uma jovem gerada pela StyleGAN, uma rede adversária generativa (GAN). A pessoa nesta foto não existe, mas é gerada por uma inteligência artificial baseada na análise de retratos..... 34
2.17 Arquitetura básica de um Autoencoder. 35
2.18 Comparação de conjuntos de dados de *Deepfakes* [6]. Ambos os eixos estão em escala logarítmica. Os tamanhos dos conjuntos de dados são separados por gerações. Na primeira geração tem-se conjuntos de dados como o *DF-TIMIT* [7]. Na segunda geração são encontrados conjuntos de dados mais famosos como o *CELEB-DF* [8] e o *FF++ DF* [5]. Na terceira geração encontram-se os maiores conjuntos de dados disponíveis atualmente como o *Deep Fake Detection Challenge* [6] com aproximadamente 100 mil vídeos. 36

3.1 Fluxograma de informações das seções presentes no capítulo de metodologia. 37
3.2 Bandeira do *Deepfake Detection Challenge* disponibilizado no site original do desafio na plataforma online *Kaggle*. 38

3.3 Exemplos do conjunto de dados *Deepfake Detection Challenge* onde (a) tem-se o vídeo com o rosto original e (b) o vídeo *Deepfake* equivalente. 38
3.4 Exemplo de estruturação do arquivo "*metadata.json*" para o diretório "*dfdc_train_part_0*". O nome do vídeo dentro do diretório é dado pelo índice da linha. A coluna **label** indica a classe do vídeo (*REAL* ou *FAKE*). A coluna **split** é sempre marcada como *train*. A coluna **original** indica, caso o vídeo seja um *deepfake*, o vídeo original do qual o *deepfake* foi gerado. 39
3.5 Pipeline da estrutura em cascata que inclui os três estágios das redes convolucionais multitarefas em cascata. Em primeiro lugar, as janelas candidatas são produzidas através de uma rede de proposta rápida (P-Net). Depois disso, esses candidatos são refinados na próxima etapa por meio de uma rede de refinamento (R-Net). Na terceira fase, a rede de saída (O-Net) produz a caixa delimitadora final e a posição dos marcadores faciais [9]. 40
3.6 Fluxograma do processo de extração das faces dos vídeos utilizando a MTCNN e posteriormente salvamento do arquivo. 41
3.7 Arquiteturas para o conjunto de dados ImageNet. Os blocos de construção são mostrados entre colchetes (ver também Figura 3.8), com os números de blocos empilhados. A redução da resolução é realizada por conv3_1, conv4_1 e conv5_1 com um *stride* de 2 [3]. 44
3.8 Exemplos de blocos residuais utilizados pelas arquiteturas ResNets para a *ImageNet* [3]. 44
3.9 Processo de realização da validação cruzada. O conjunto de dados de treinamento é separado em N partições, o modelo é treinado em $N-1$ partições e validado na partição restante. O processo se repete para todas as possíveis combinações de $N-1$ partições para treinamento. Pode-se optar inicialmente por separar um conjunto de testes do conjunto total de dados antes de iniciar o processo de validação cruzada. Nesse caso, a validação cruzada é feita apenas para a parte separada para treinamento do conjunto total. Após finalizada a validação cruzada, o modelo é treinado em todo o conjunto de dados e validado uma última vez no conjunto de testes separado inicialmente [2]. 46
3.10 Processo de comparação final realizado para os vídeos do conjunto de testes do diretório *test*. As imagens obtidas pela MTCNN (etapa anterior à todo o treinamento) passam pelo classificador treinado, gerando as predições. As predições são contabilizadas e a classe que estiver mais relativamente presente é então dada como a classe a qual o vídeo pertence. 47

4.1 Exemplos de algumas das imagens extraídas pela MTCNN e suas respectivas classes. 50
4.2 Distribuição aproximada do tamanho das imagens no conjunto de dados obtido após o pré-processamento da MTCNN. 51

4.3 Processo de redimensionamento das imagens durante o treinamento e durante a validação. A imagem original é proporcionalmente redimensionada de forma que a menor dimensão original se torna igual à dimensão desejada. O recorte é então realizado removendo trechos apenas da maior dimensão de forma que a imagem se torne quadrada no final. 52
4.4 Resultados para o teste de faixa da taxa de aprendizado para a (a) partição 1, (b) partição 2, (c) partição 3, (d) partição 4 e (e) partição 5 da validação cruzada. O resultado em (f) representa o teste para o conjunto completo de treinamento..... 54
4.5 Resultados das métricas de custo, acurácia, MCC e área sob a curva ROC durante a etapa de validação do modelo na partição 1 da validação cruzada. A parte superior abrange a etapa com o modelo congelado e a parte inferior abrange a etapa com o modelo descongelado. O tempo de treinamento e validação de cada época é apresentado na coluna **time**. 56
4.6 Matrizes de confusão (a) absoluta e (b) normalizada resultantes para o conjunto de validação da partição 1 da validação cruzada após o treinamento do modelo..... 57
4.7 Exemplos no conjunto de validação da partição 1 da validação cruzada que geraram o maior valor de custo para o modelo treinado. A interpretação é de que esses são os exemplos que deixaram a rede mais "confusa". A legenda acima de cada imagem representa a classe predita pelo modelo, a classe real a qual o modelo pertence, o custo gerado na classificação e a probabilidade que o modelo deu para a classe predita. 57
4.8 Resultados das métricas de custo, acurácia, MCC e área sob a curva ROC durante a etapa de validação do modelo na partição 2 da validação cruzada. A parte superior abrange a etapa com o modelo congelado e a parte inferior abrange a etapa com o modelo descongelado. O tempo de treinamento e validação de cada época é apresentado na coluna **time**. 58
4.9 Matrizes de confusão (a) absoluta e (b) normalizada resultantes para o conjunto de validação da partição 2 da validação cruzada após o treinamento do modelo..... 59
4.10 Exemplos no conjunto de validação da partição 2 da validação cruzada que geraram o maior valor de custo para o modelo treinado. A interpretação é de que esses são os exemplos que deixaram a rede mais "confusa". A legenda acima de cada imagem representa a classe predita pelo modelo, a classe real a qual o modelo pertence, o custo gerado na classificação e a probabilidade que o modelo deu para a classe predita. 59
4.11 Resultados das métricas de custo, acurácia, MCC e área sob a curva ROC durante a etapa de validação do modelo na partição 3 da validação cruzada. A parte superior abrange a etapa com o modelo congelado e a parte inferior abrange a etapa com o modelo descongelado. O tempo de treinamento e validação de cada época é apresentado na coluna **time**. 60
4.12 Matrizes de confusão (a) absoluta e (b) normalizada resultantes para o conjunto de validação da partição 3 da validação cruzada após o treinamento do modelo..... 61

4.13 Exemplos no conjunto de validação da partição 3 da validação cruzada que geraram o maior valor de custo para o modelo treinado. A interpretação é de que esses são os exemplos que deixaram a rede mais "confusa". A legenda acima de cada imagem representa a classe predita pelo modelo, a classe real a qual o modelo pertence, o custo gerado na classificação e a probabilidade que o modelo deu para a classe predita. 61
4.14 Resultados das métricas de custo, acurácia, MCC e área sob a curva ROC durante a etapa de validação do modelo na partição 4 da validação cruzada. A parte superior abrange a etapa com o modelo congelado e a parte inferior abrange a etapa com o modelo descongelado. O tempo de treinamento e validação de cada época é apresentado na coluna **time**. 62
4.15 Matrizes de confusão (a) absoluta e (b) normalizada resultantes para o conjunto de validação da partição 4 da validação cruzada após o treinamento do modelo..... 63
4.16 Exemplos no conjunto de validação da partição 4 da validação cruzada que geraram o maior valor de custo para o modelo treinado. A interpretação é de que esses são os exemplos que deixaram a rede mais "confusa". A legenda acima de cada imagem representa a classe predita pelo modelo, a classe real a qual o modelo pertence, o custo gerado na classificação e a probabilidade que o modelo deu para a classe predita. 63
4.17 Resultados das métricas de custo, acurácia, MCC e área sob a curva ROC durante a etapa de validação do modelo na partição 5 da validação cruzada. A parte superior abrange a etapa com o modelo congelado e a parte inferior abrange a etapa com o modelo descongelado. O tempo de treinamento e validação de cada época é apresentado na coluna **time**. 64
4.18 Matrizes de confusão (a) absoluta e (b) normalizada resultantes para o conjunto de validação da partição 5 da validação cruzada após o treinamento do modelo..... 65
4.19 Exemplos no conjunto de validação da partição 5 da validação cruzada que geraram o maior valor de custo para o modelo treinado. A interpretação é de que esses são os exemplos que deixaram a rede mais "confusa". A legenda acima de cada imagem representa a classe predita pelo modelo, a classe real a qual o modelo pertence, o custo gerado na classificação e a probabilidade que o modelo deu para a classe predita. 65
4.20 Resultados das métricas de custo, acurácia, MCC e área sob a curva ROC durante a etapa de validação do modelo utilizando o conjunto de dados completo. A parte superior abrange a etapa com o modelo congelado e a parte inferior abrange a etapa com o modelo descongelado. O tempo de treinamento e validação de cada época é apresentado na coluna **time**. 66
4.21 Matrizes de confusão (a) absoluta e (b) normalizada resultantes para o conjunto de teste da após o treinamento do modelo no conjunto de treinamento completo. 67
4.22 Exemplos no conjunto de teste do conjunto de treinamento completo que geraram o maior valor de custo para o modelo treinado. A interpretação é de que esses são os exemplos que deixaram a rede mais "confusa". A legenda acima de cada imagem representa a classe predita pelo modelo, a classe real a qual o modelo pertence, o custo gerado na classificação e a probabilidade que o modelo deu para a classe predita. 67

4.23 Curvas ROC resultantes para os modelos da (a) partição 1, (b) partição 2, (c) partição 3, (d) partição 4 e (e) partição 5 da validação cruzada estratificada e (f) do modelo obtido do conjunto de dados completo. 68
4.24 Estimativas de densidade por kernel gaussiano das distribuições das probabilidades resultantes do modelo para as classes *REAL* e *FAKE*..... 71
4.25 Estimativas de densidade por kernel gaussiano das distribuições das probabilidades resultantes do modelo para as classes *REAL* e *FAKE* com ampliação no intervalo de 0 a 1 no eixo de densidade. 72
4.26 Matrizes de confusão (a) absoluta e (b) normalizada resultantes para a inferência do modelo nos agrupamentos de imagens específicos de cada um dos 17.606 vídeo do diretório *test*, onde a classe predita para o vídeo segue o processo descrito na Figura 3.10. Resultados para $\rho = 1$ 73
4.27 Diferentes valores de ρ versus as taxas de verdadeiro positivo e verdadeiro negativo do modelo inferido nos agrupamentos de imagens do diretório *test*..... 74
4.28 Matrizes de confusão (a) absoluta e (b) normalizada resultantes para a inferência do modelo nos agrupamentos de imagens específicos de cada um dos 17.606 vídeo do diretório *test*, onde a classe predita para o vídeo segue o processo descrito na Figura 3.10. Resultados para $\rho = 2,75$..... 75

LISTA DE TABELAS

4.1 Configurações da máquina *host* utilizada para o treinamento do modelo de aprendizado profundo. 49
4.2 Quantidade de imagens geradas por diretório por classe após o pré-processamento utilizando a MTCNN. 50
4.3 Distribuição da quantidade de imagens por cada uma das 5 partições da validação cruzada. 51
4.4 Valores de média e desvio padrão originais utilizados nas imagens da ImageNET..... 53
4.5 Hiperparâmetros utilizados para o treinamento do modelo Resnet18. 53
4.6 Tabela de médias e desvios padrões de algumas da métricas utilizadas na validação do modelo para as 5 partições da validação cruzada. 68
4.7 Tabela de resultados obtidos no conjunto de treinamento e teste final, resultados esperados para cada uma das métricas pela validação cruzada e diferença entre o valor obtido para a métrica e o valor esperado..... 70
4.8 Tempos de processamento de uma solução end-to-end para 10 vídeos diferentes do diretório "*dfdc_train_part_0*". Todos os vídeos em questão apresentam 10 segundos de duração. As extrações das faces foram realizadas de 30 em 30 quadros (aproximadamente de 1 em 1 segundo). As faces foram extraídas sequencialmente pela MTCNN e, após completo o processo, o modelo realizava as predições sequencialmente em cada uma das imagens. 76

LISTA DE SÍMBOLOS

Símbolos Gregos

γ	Fração do Vetor de Atualização da Iteração Passada do SGD
ω	Peso Arbitrário Interno de um Modelo
ρ	Coeficiente de Liberdade
σ	Função de Ativação Sigmoide
θ	Parâmetro Arbitrário Interno de um Modelo
e	Número de Euler ou Constante de Euler

Siglas

$\hat{y}$	Predição Arbitrária de um Modelo
$\tilde{L}$	Função de Custo Regularizada
b	Viés Arbitrário de um Modelo - *bias*
fn	Falsos Negativos - *False Negatives*
fp	Falsos Positivos - *False Positives*
tn	Verdadeiros Negativos - *True Negatives*
tp	Verdadeiros Positivos - *True Positives*
v	Vetor de Atualização do SGD
W	Matriz de Pesos de um Modelo
x	Entrada Arbitrária de um Modelo
CNN	Rede Neural Convolucional - Convolutional Neural Network
CPU	Unidade Central de Processamento - *Central Processing Unit*
DFDC	*Deepfake Detection Challenge*
EER	Taxa de Erro Igual - *Equal Error Rate*

FPR Taxa de Falsos Positivos - *False Positive Rate*

GAN Rede Adversária Generativa - *Generative Neural Network*

GPU Unidade de Processamento Gráfico - *Graphics Processing Unit*

LSTM Memória Longa de Curto Prazo - *Long Short-term Memory*

MCC Coeficiente de Correlação de Matthews - *Matthews Correlation Coefficient*

MTCNN Rede convolucional multitarefa em cascata - *Multi-task cascaded convolutional network*

NAG Descida Acelerada do Gradiente de Nesterov - *Nesterov Accelerated Gradient*

RAM Memória de Acesso Aleatório - *Random Access Memory*

ReLU Retificadora Linear - *Rectified Linear Unit*

RNA Rede Neural Artificial

RNN Rede Neural Recorrente - *Recurrent Neural Network*

ROC Característica de Operação do Receptor - *Receiver Operating Characteristic*

SGD Descida de Gradiente Estocástico - *Stochastic Gradient Descent*

TFA Taxa de Falsa Aceitação

TFA Taxa de Falsa Rejeição

TPR Taxa de Verdadeiros Positivos - *True Positive Rate*

Capítulo 1

Introdução

1.1 Contextualização

A revolução trazida pela internet impactou as mais diversas áreas do conhecimento humano. Em praticamente tudo que fazemos, utilizamos a internet. Essa ferramenta revolucionou toda a forma como a sociedade humana se comunica. Antes da internet, se um indivíduo quisesse acompanhar as notícias mais recentes a respeito de acontecimentos no mundo, seria necessário se deslocar até a banca mais próxima e comprar um jornal com as reportagens do que aconteceu no último dia. Hoje, alguns cliques são mais do que suficientes para obter as informações a respeito dos acontecimentos mais recentes do último minuto [10].

As facilidades na transmissão de informação providas pela internet são também observadas na transmissão de desinformação. Uma técnica que atualmente está na liderança da desinformação baseada em vídeos são os chamados *deepfakes* [11]. *Deepfake* constitui um vídeo onde um indivíduo tem seu rosto trocado necessariamente por algum algoritmo de inteligência artificial [6].

Em junho de 2019, um usuário da rede social *instagram* publicou um vídeo do atual CEO da empresa *Facebook*, Mark Zuckerberg, comentando sobre como ele manipula bilhões de dados roubados. O vídeo, na verdade, se trata de um *deepfake* produzido por dois artistas chamados Bill Posters e Daniel Howe. Estes trabalharam em conjunto com uma agência da área de publicidade chamada *Canny*. Os algoritmos utilizados fizeram uso de imagens reais de Mark Zuckerberg em diversas aparições públicas. Essas imagens foram então alteradas e posteriormente combinadas ao rostode um ator, que foi responsável por citar as palavras mencionadas no vídeo. O vídeo utilizou também a logo da CBS. um grande canal de TV norte-americana, de forma a transparecer mais credibilidade. A arquitetura de inteligência artificial foi treinada durante horas e fez uso de diversos vídeos reais mais curtos de Zuckerberg do ano de 2017 [12]. Desde o início da publicação, os autores deixaram claro que se tratava de um vídeo *deepfake*, todavia, nem sempre é o caso.

O processo de criação de *deepfakes* é capaz de gerar substituições realistas, o que se torna um problema quando o alvo é uma figura política ou pública de impacto ou uma celebridade. Evidências anedóticas sugerem que a perspectiva de produção em massa e difusão de deepfakes por atores maliciosos pode representar o desafio mais sério até agora para a autenticidade do

Figura 1.1: *Deepfake* de Mark Zuckerberg publicado na rede social *instagram* em julho de 2019.

discurso político online [11]. As imagens têm um poder de persuasão mais forte do que o texto e os cidadãos têm defesas comparativamente fracas contra enganos visuais dessa natureza [13, 14].

O que torna o *deepfake* particularmente perigoso nos tempos modernos é a fácil acessibilidade dos usuários a alto poder computacional combinado com recursos grátis disponíveis na internet de como realizar tais forjações. Por se tratar do produto de um algoritmo de aprendizado de máquina profundo, os *deepfakes* necessitam de elevada carga computacional para serem produzidos [15]. O poder computacional necessário para lidar com essas cargas é amplamente acessível nos tempos modernos. Temos placas de vídeo (GPUs) com preços acessíveis atualmente e que possuem poder de computação paralela suficiente para tornar o processo de criação de *deepfakes* viável em muitas máquinas. Serviços de computação em nuvem também se tornaram mais acessíveis e oferecem a usuários poderosas máquinas virtuais com GPUs facilmente acessíveis e em alguns casos até mesmo grátis como na plataforma de programação da *Google* nomeada *Google Colab* ou na plataforma de ciência de dados *Kaggle* [16]. Aliado a esses fatores, a internet tornou as informações necessárias para a criação de *deepfakes* públicas e de fácil acesso. Tutoriais completos e ferramentas que abstraem a maior parte do conhecimento específico necessário para a sua criação já são facilmente encontrados na internet, como é o caso do *DeepFaceLab* [17].

A capacidade de se detectar *deepfakes* é de vital importância para garantir a imagem e a integridade das pessoas, assim como combater a desinformação que os tempos atuais vivem.

1.2 Descrição do problema e objetivos

1.2.1 Objetivo Geral

Elaborar solução utilizando redes neurais convolucionais profundas para detecção de *deepfakes* a partir de imagens contidas em vídeos.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho consistem em treinar uma rede neural convolucional para detectar *deepfakes* em vídeos. Este objetivo é dividido nas seguintes etapas:

- Definir uma base de dados ampla e representativa suficiente para o treinamento da rede.
- Escolher uma arquitetura de rede neural convolucional a ser utilizada para detectar *deepfakes* em vídeos.
- Realizar o ajuste do modelo de treinamento e dos hiperparâmetros a serem utilizados.
- Analisar os resultados obtidos utilizando diversas métricas de desempenho para o classificador escolhido.
- Verificar a viabilidade da rede neural convolucional treinada como ferramenta no combate à *deepfakes* utilizando os resultados obtidos anteriormente.

1.3 Apresentação do manuscrito

Este manuscrito consiste dos seguintes capítulos:

- Capítulo 2 - Fundamentos teóricos: neste capítulo, será realizada uma revisão bibliográfica de assuntos relacionados ao trabalho e será oferecido o contexto teórico do aprendizado profundo e as principais definições necessárias para o entendimento e a validação da pesquisa;
- Capítulo 3 - Metodologia: neste capítulo, serão definidas as técnicas de pesquisa utilizadas, as ferramentas utilizadas, listados os procedimentos experimentais, definidas as propostas as arquiteturas das redes neurais, definido o procedimento de treinamento da arquitetura escolhida e definido o processo de validação e testes do modelo escolhido;
- Capítulo 4 - Resultados e Discussões: neste capítulo, serão mostrados e discutidos os resultados quantitativos e qualitativos, que foram a criação de uma base de dados de imagens a partir de uma base de dados de vídeos, o treinamento da arquitetura no conjunto de imagens gerado e a validação da arquitetura treinada nos conjuntos de validação, teste e nos vídeos em si;
- Capítulo 5 - Conclusões: contém um resumo do trabalho feito e as considerações finais.

Capítulo 2

Fundamentos teóricos e Trabalhos Relacionados

2.1 Aprendizado Profundo

2.1.1 Histórico

O termo aprendizado profundo ou aprendizado de máquina profundo tem seus alicerces no campo da Inteligência Artificial [1].

A inteligência artificial hoje se caracteriza como um campo de pesquisa e aplicações práticas de elevado sucesso, repleto de tópicos ativos na área de pesquisa. Em seu surgimento, o campo lidou com problemas que em sua essência eram intelectualmente difíceis para seres humanos porém relativamente fáceis para computadores, resolvendo-os de forma bem direta. Esses problemas podiam ser descritos por uma lista de regras formais e matemáticas. Ironicamente, o verdadeiro desafio da inteligência artificial está na solução de problemas que em sua essência são fáceis para seres humanos realizarem, porém difíceis de seres humanos descreverem formalmente. Problemas esses que os seres humanos são naturalmente capazes de resolverem de forma intuitiva, isto é, de forma automática sem a necessidade de investir muito pensamento naquilo, como por exemplo reconhecer objetos específicos em imagens ou perceber sentimento em uma sentença falada ou escrita [1].

Várias soluções foram propostas ao longo dos anos para esses problemas mais intuitivos, e definitivamente uma das de maior sucesso hoje é a de aprendizado profundo. Essa solução envolve permitir computadores aprenderem através de experiência e entender o mundo em termos de uma hierarquia do que podem ser interpretados como conceitos, onde cada conceito é definido pela sua relação com conceitos mais simples. A abordagem de adquirir conhecimento por experiência traz consigo um importante fator: remove a necessidade de operadores humanos especificarem formalmente todo o conhecimento envolvido no aprendizado do computador. A ideia de hierarquia de conceitos permite o computador construir uma cadeia de conceitos complexa em cima de conceitos relativamente mais simples. Se fôssemos construir uma estrutura mostrando como cada

um desses conceitos se encadeia uns após os outros, essa estrutura seria profunda, com várias camadas. Por essa razão, a abordagem em questão para inteligência artificial recebe o nome de aprendizado de máquina **profundo** [1].

Algumas das tarefas mais difíceis e mentalmente desafiadoras para seres humanos se provam algumas das mais simples para um computador. Há tempos os computadores já são capazes de vencer até o melhor jogador humano de xadrez, porém só recentemente eles se mostraram capazes de apresentar uma performance similar a de seres humanos ordinários como reconhecer objetos ou fala. O dia-a-dia de uma pessoa envolve uma quantidade imensa de conhecimento a respeito do mundo. Muito desse conhecimento é intuitivo e subjetivo, portanto difícil de se articular de uma maneira formal. Dessa forma, para que computadores se comportem de uma maneira considerada inteligente, estes têm de apresentar a capacidade de capturar essa mesma ideia de conhecimento. Daqui, surge um dos principais desafios da inteligência artificial: como inserir esse conhecimento profundamente informacional em um computador [1].

Várias tentativas de se programar diretamente conhecimento sobre o mundo em linguagens mais formais para uma máquina foram realizadas, como por exemplo o sistema Cyc [18]. Essa é uma abordagem de inteligência artificial conhecida como abordagem baseada em conhecimento e caracteriza-se por uma abordagem notoriamente complexa e pesada de ser executada. Nenhum dos projetos realizados nesses termos levou a um grande sucesso. As dificuldades enfrentadas por sistemas baseados em programação direta de conhecimento sugerem que uma abordagem relevante para um sistema de inteligência artificial consiste na capacidade de adquirir seu próprio conhecimento, sendo capaz por si só de identificar e extrair padrões de dados. Essa capacidade veio a ser conhecida como **aprendizado de máquina** [1].

Com o aprendizado de máquina, os computadores se tornaram capazes de enfrentar problemas que envolviam conhecimento do mundo real e, dessa forma, tomar decisões que pareciam subjetivas. Algoritmos simples de aprendizado de máquina como a **regressão logística** podem ser utilizados para recomendar ou não um parto por cesariana [19]. Algoritmos tradicionais simples como o mencionado dependem fortemente da **representação** dos dados que lhe são entregues e isto pode representar um problema. Se uma regressão logística fosse receber uma varredura por um equipamento de ressonância magnética, em vez do relatório formalizado do médico, ela não saberia o que fazer. Pixels individuais de uma varredura de ressonância magnética possuem correlação praticamente nula com as complicações que podem ocorrer durante um parto [1].

Para muitas tarefas, o conhecimento dos atributos que devem ser extraídos pode se mostrar diversas vezes uma tarefa relativamente difícil. Isto torna também difícil solucionar problemas utilizando aprendizado de máquina tradicional. Uma solução para esse problema consiste em utilizar o aprendizado de máquina para aprender não apenas o mapeamento da representação para uma saída mas também a própria representação. Esta abordagem vem a ser conhecida como **aprendizagem de representação**. Representações que foram aprendidas frequentemente resultam em uma performance superior em comparação com a que se pode obter com representações montadas a mão. Um algoritmo de aprendizagem de representação pode descobrir um bom conjunto de atributos para uma tarefa simples em minutos, ou horas a meses para tarefas mais complexas [1].

Quando se estão projetando atributos ou algoritmos para aprender atributos, o objetivo geralmente consiste em separar fatores de variação que explicam os dados observados. Extrair atributos tão abstratos e de alto nível assim de dados puros pode se mostrar uma tarefa bem difícil. Alguns desses fatores de variação, como o sotaque de um orador ou a claridade do sol em uma imagem, podem ser identificados apenas utilizando um conhecimento relativamente sofisticado dos dados [1].

O **aprendizado profundo** resolve esse problema introduzindo a estrutura de hierarquia mencionada anteriormente, onde representações são expressas em termos de outras representações mais simples, permitindo o computador montar conceitos complexos a partir de conceitos mais simples. A ideia de aprender a correta representação dos dados provê uma das perspectivas do aprendizado profundo. Outra perspectiva possível é a de que a profundidade permite o computador aprender um programa cuja organização se dá em várias etapas. Embora o aprendizado profundo possa ser considerado de forma segura o estudo de modelos que envolvem uma quantidade maior que funções aprendidas ou conceitos aprendidos do que as dos modelos tradicionais de aprendizado de máquina, não há um consenso da profundidade mínima que um modelo deve ter para ser considerado "profundo"[1]. A idéia fundamental da organização da estrutura hierárquica é apresentada na Figura 2.1.

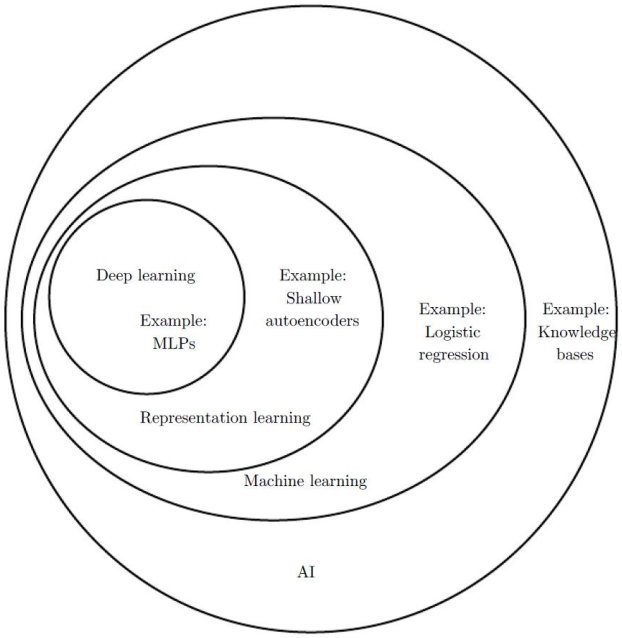


Figura 2.1: Um diagrama de Venn mostrando como o aprendizado profundo (Deep learning) pertence ao grupo de aprendizagem de representação (Representation learning), que por sua vez é uma área do aprendizado de máquina (Machine learning). Observa-se também que consiste apenas uma das várias abordagens que se têm disponíveis para a inteligência artifical (AI) [1].

O aprendizado profundo aparenta ser uma tecnologia nova, pertinente à última década, todavia isso não é verdade. Suas primeiras aparições datam da década de 1940 [1]. Seu surgimento e propagação se deu inicialmente por nomes diferentes, sendo apenas recentemente chamado de "aprendizado profundo". Em sua trajetória já assumiu nomes como **cibernéticos** e **conexio-**

nismo. Vários algoritmos que possuíam o embasamento do aprendizado profundo foram inicialmente propostos a partir do aprendizado biológico, isto é, modelos de como o aprendizado acontece no cérebro. Devido a tal aspecto, um dos nomes que o aprendizado profundo recebeu é o de **redes neurais artificiais** (RNAs). Embora a inspiração tenha vindo do campo da neurociência e de como neurônios se conectam no cérebro, os modelos de aprendizado profundo não são projetados para serem cópias realísticas de funções biológicas. Sabemos hoje que neurônios reais apresentam funções bem diferentes das empregadas nos modelos de aprendizado profundo. O aprendizado profundo moderno tira inspiração de vários campos de fundamentos de matemática aplicada tais como álgebra linear, probabilidade, teoria da informação e otimização numérica [1].

Os algoritmos no campo do aprendizado profundo hoje são relativamente similares aos da década de 1980, com algumas mudanças para simplificar o processo de treinamento para arquiteturas muito profundas. Uma das grandes vantagens que os tempos modernos proporcionam é prover esses algoritmos com os recursos que eles de fato precisam para obterem sucesso. Isso vem da atual e contínua digitalização da sociedade: quanto mais as pessoas passam o dia a dia em um computador, mais as suas ações são gravadas. Como os computadores estão cada vez mais conectados, se torna mais fácil centralizar esses dados e agregá-los em um conjunto de dados separado suficientemente grande e útil para alimentar os algoritmos de aprendizado profundo hoje. Os primeiros conjuntos de dados utilizados no campo de aprendizado de máquina possuíam apenas algumas centenas de exemplos como o conjunto de dados Iris [20, 21] enquanto os mais recentes como o conjunto de dados WMT 2014 Inglês para Francês [22] contém dezenas de milhões de exemplos.

Um fator determinante para o sucesso do aprendizado profundo hoje vem também do fato que a disponibilidade de recursos computacionais hoje é significativamente maior, o que permite o treinamento de modelos muito maiores dos que aquelas propostos na década de 1980. O aumento no tamanho dos modelos ao longo do tempo devido à disponibilidade de processadores mais rápidos, o advento de GPUs de uso geral, conectividade à rede mais rápida e melhor infraestrutura de software para computação distribuída, é uma das tendências mais importantes na história do aprendizado profundo [1]. Com o passar do anos, os modelos de aprendizado de máquina se tornaram maiores e mais eficazes em resolver tarefas, o que consequentemente aumentou a complexidade das tarefas que eles são capazes de resolver.

Avanços no aprendizado profundo hoje também foram significativamente facilitados uma vez que poderosas bibliotecas de software se tornaram mais acessíveis. Bibliotecas como o TensorFlow [23] e o Pytorch [24] possuem suporte para vários sistemas operacionais e sistemas computacionais em nuvem com acesso a múltiplas GPUs, possibilitando um foco maior na pesquisa em si e não na infraestrutura para poder realizá-la.

Com isso, o aprendizado profundo pode ser resumido como uma abordagem para a inteligência artificial inspirada fortemente nos mecanismos de funcionamento do cérebro humano, possuindo fortes fundamentos nas áreas de estatística e matemática aplicada. A grande popularização dessa abordagem se deu recentemente em decorrência de computadores mais poderosos, conjuntos de dados significativamente maiores e técnicas mais avançadas para treinamento de arquiteturas

profundas [1].

2.1.2 Principais Definições de Aprendizado Profundo

2.1.2.1 Perceptron Multicamadas

De acordo com [1], perceptrons multicamadas, também conhecidos como redes neurais ou redes neurais profundas a depender da quantidade de camadas presentes, consistem os exemplos mais típicos de modelos de aprendizado profundo. O objetivo destes modelos é aproximar alguma função f^*. Tendo um classificador como exemplo, a função $\hat{\boldsymbol{y}} = f^*(\boldsymbol{x})$ mapeia uma entrada $\boldsymbol{x}$ qualquer para uma categoria $\hat{\boldsymbol{y}}$ pré-definida. Em uma descrição mais específica, uma rede neural define um mapeamento $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta})$ e aprende um conjunto de valores numéricos chamados de parâmetros ou pesos da rede (definidos por $\boldsymbol{\theta}$) que resultam na melhor aproximação da função.

O nome perceptron multicamadas surge do modelo original perceptron [25]. A sequência dessas unidades na vertical dá origem ao tamanho da camada e a sequência dessas unidades na horizontal dá origem ao número de camadas. A informação flui sempre de forma sequencial, uma unidade após a outra na horizontal. O perceptron simples está detalhado na Figura 2.2 e um exemplo do perceptron multicamadas está detalhado na Figura 2.3. Unidades individuais do perceptron são comumente chamados de **neurônios** da rede.

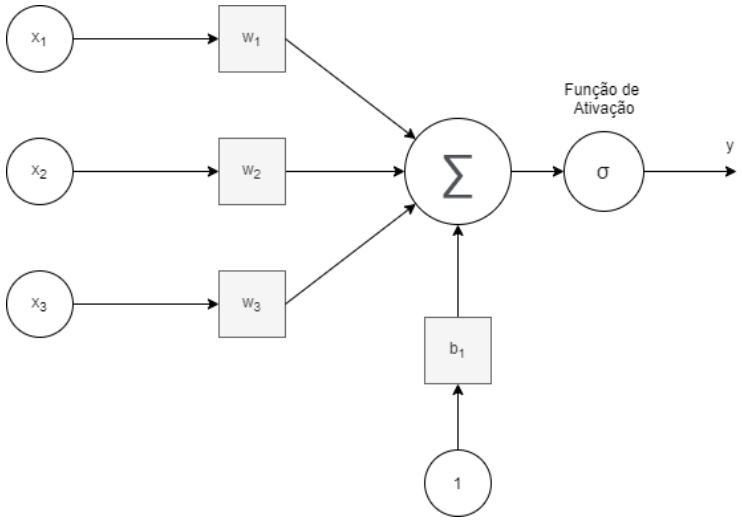


Figura 2.2: Uma unidade perceptron simples. As entradas $\boldsymbol{x} = [x_1, x_2, x_3]$ são individualmente multiplicadas pelos seus respectivos pesos $\boldsymbol{\omega} = [w_1, w_2, w_3]^t$ e os resultados são então somados com a multiplicação entre uma entrada unitária e um parâmetro pertinente à própria unidade perceptron chamado *bias* ($\boldsymbol{b} = [b_1]$), passando então por uma função de ativação (aqui representada como uma função sigmoide $\sigma(x)$) final que introduz uma não-linearidade (melhor discutido na Seção 2.1.2.2). O conjunto de parâmetros aprendíveis $\boldsymbol{\omega}$ e $\boldsymbol{b}$ é o conjunto total de parâmetros aprendíveis $\boldsymbol{\theta}$. A entrada *bias*, neste caso, pode ser definida tanto como uma entrada de valor b_1 e peso de valor 1 quanto uma entrada de valor 1 e peso de valor b_1.

É possível observar pela Figura 2.3 que a saída do perceptron multicamadas pode ser obtida

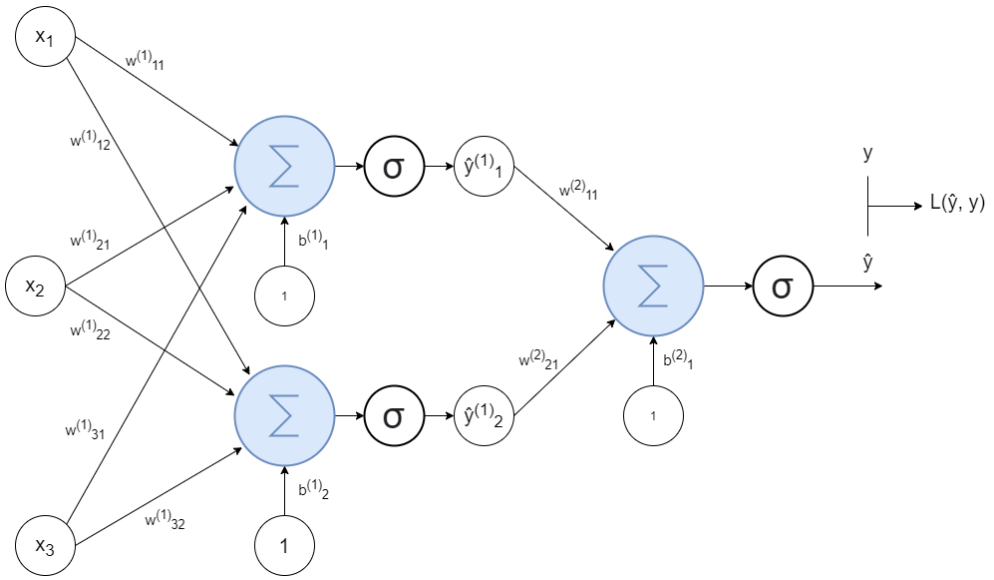


Figura 2.3: Um modelo perceptron multicamadas de 2 camadas, possuindo 2 unidades na primeira camada e 1 unidade na segunda. As entradas $\boldsymbol{x} = [x_1, x_2, x_3]$ são individualmente multiplicadas por pesos associados a cada unidade perceptron dentro da camada. As saídas após as funções de ativação da primeira camada são interpretadas como novas entradas pela camada seguinte. Ao final das operações, a saída $\hat{y}$ é comparada com a classe ou valor correto esperado y utilizando uma função de custo $L(\hat{y}, y)$.

realizando um conjunto de equações matriciais. A Equação 2.1 mostra a saída da primeira camada e a Equação 2.2 a saída da segunda camada. As matrizes dos pesos $w_{ij}^{(k)}$, onde i corresponde à entrada x_i, j corresponde à unidade perceptron a qual a entrada esta conectada e k corresponde à camada, crescem verticalmente com a quantidade de entradas da camada anterior e crescem horizontalmente com a quantidade de unidades perceptron na camada atual. Os vetores de *bias* $b_j^{(k)}$ seguem a mesma notação para k e j. As saídas das camadas internas são dadas por $\hat{y}_i^{(k)}$ onde i corresponde à saída da unidade perceptron dada como j na camada anterior. A saída final do modelo é dada apenas por $\hat{y}$.

$$\begin{bmatrix} \hat{y}_1^{(1)} & \hat{y}_2^{(1)} \end{bmatrix} = \sigma(\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} \end{bmatrix} + \begin{bmatrix} b_1^{(1)} & b_2^{(1)} \end{bmatrix}), \tag{2.1}$$

$$\hat{y} = \sigma(\begin{bmatrix} y_1^{(1)} & y_2^{(1)} \end{bmatrix} \begin{bmatrix} w_{11}^{(2)} \\ w_{21}^{(2)} \end{bmatrix} + \begin{bmatrix} b_1^{(2)} \end{bmatrix}). \tag{2.2}$$

Desta forma, considerando a matriz de pesos de uma determinada camada como $W^{(k)}$, o vetor de *bias* de uma determinada camada como $\boldsymbol{b^{(k)}}$, onde k corresponde à camada atual, a entrada o vetor $\boldsymbol{x} = [x_1, x_2, x_3]$ e a saída final o valor $\hat{y}$, temos que as Equações 2.1 e 2.2 se tornam:

$$\hat{y} = \sigma(\sigma(\boldsymbol{x}W^{(1)} + \boldsymbol{b}^{(1)})W^{(2)} + \boldsymbol{b}^{(2)}). \tag{2.3}$$

O perceptron multicamadas da Figura 2.3 possui um total de 11 parâmetros, isto é, os pesos que a rede é capaz de ajustar conforme treina utilizando um algoritmo de aprendizado. Eles correspondem às matrizes $W^{(k)}$ e os vetores $\boldsymbol{b}^{(k)}$. A inserção de mais uma entrada acarretaria em um acréscimo de mais 3 conexões na primeira camada. A inserção de uma nova unidade perceptron na primeira camada acarretaria em um acréscimo de mais 4 conexões na primeira camada (pesos mais o *bias*). Observa-se como isso pode escalonar rápido. Um simples modelo de 576 entradas (correspondente aos pixels de uma imagem pequena de dimensões 24x24), com apenas 3 camadas e cada camada com 20 unidades perceptron, teria 12380 parâmetros aprendíveis.

As saídas de um modelo perceptron não produzem resultados que podem ser diretamente utilizados por métricas de performance (Seção 2.1.2.15). Para tarefas de classificação, por exemplo, a saída passa através de uma função *softmax* definida na Equação 2.4 a seguir:

$$softmax(\hat{\boldsymbol{y}})_i = \frac{e^{\hat{y}_i}}{\sum_{j=1}^{K} e^{\hat{y}_j}}, \tag{2.4}$$

onde $\hat{\boldsymbol{y}}$ é o vetor de saída do perceptron multicamadas e K é a quantidade de unidades perceptron na saída (coincide com a quantidade de classes). Basicamente, a função *softmax* aplica uma exponencial em cada elemento $\hat{y}_j$ do vetor de saída $\hat{\boldsymbol{y}}$ e normaliza esses valores dividindo cada resultado pela soma de todas as exponenciais. Essa normalização garante que a soma dos componentes do vetor de saída $softmax(\hat{\boldsymbol{y}})$ é 1. Dessa forma, é possível agora interpretar a saída como as probabilidades referentes a cada classe do problema em questão. Geralmente assume-se que a classe que produziu a maior probabilidade após a aplicação da função *softmax* é a classe a qual a entrada pertence.

2.1.2.2 Funções de Ativação

Funções de ativação permitem a introdução de não linearidades no modelo. Na Figura 2.3, a função de ativação exemplo foi uma sigmoide definida como $\sigma(x) = \frac{1}{1+e^{-x}}$. A ausência de uma não linearidade em um modelo como o da Figura 2.3 resultaria em um modelo linear independente da quantidade de neurônios ou da quantidade de camadas da rede. Com essas não linearidades introduzidas, obtemos um modelo matemático capaz de aproximar qualquer função contínua com uma precisão arbitrária de acordo com o *teorema da aproximação universal* [26]. Desta forma, redes neurais são aproximadoras universais.

Algumas funções de ativação comumente utilizadas no campo do aprendizado profundo são citadas a seguir:

- Sigmoide: $\sigma = \frac{1}{1+e^{-x}}$
- Tangente Hiperbólica: $tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
- Retificadora Linear: $ReLU(x) = max(0, x)$
- Swish: $s(x) = \frac{x}{1+e^{-x}}$

- Softplus: $softplus(x) = ln(1 + e^x)$
- Mish: $m(x) = x * tanh(softplus(x))$

Dos exemplos citados, temos destaque para a função $ReLU$ que domina o campo do aprendizado profundo recente, sendo uma das funções mais amplamente utilizadas em modelos considerados estado da arte. Funções mais recentes como a $Swish$ [27] e a $Mish$ [28] apresentam desempenho tão bom quanto ou superior à $ReLU$ em várias aplicações.

2.1.2.3 Funções de Custo - *Loss Functions*

Um aspecto importante para o projeto de redes neurais profundas é a escolha de uma função de custo. Funções de custo $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ são funções específicas utilizadas para quantificar o quão próximas as saídas $\hat{\boldsymbol{y}}$ de um modelo de aprendizado profundo estão ou não das saídas desejadas $\boldsymbol{y}$ durante o treinamento. Um aspecto importante dessas funções é que elas devem ser diferenciáveis (melhor discutido na Seção 2.1.2.4). São apresentadas a seguir 2 funções de custo comumente utilizadas no ramo do aprendizado de máquina **supervisionado**, isto é, no aprendizado em que as saídas desejadas são conhecidas:

- Entropia Cruzada: É a função de custo dominante para tarefas de classificação atualmente. Consiste na média da probabilidade logarítmica negativa dos N elementos classificados, onde $p(y_x)$ é a probabilidade (0 ou 1) associada à x-ésima classe do n-ésimo exemplo conhecido e $\hat{y}_x$ é a probabilidade predita (intervalo entre 0.0 e 1.0) pelo modelo para a x-ésima classe (considerando X classes ao todo) do n-ésimo exemplo conhecido. Sua definição está na Equação 2.5 a seguir:

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = CE_L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{x=1}^{X} p(y_x) \cdot log(\hat{y}_x). \tag{2.5}$$

- Erro Quadrático Médio: Função de custo dominante para diversas tarefas de regressão. Consiste na subtração entre o valor esperado y_i para o n-ésimo exemplo e o valor predito $\hat{y}_i$ para o n-ésimo exemplo. É definido pela Equação 2.6 a seguir:

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = MSE_L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2. \tag{2.6}$$

2.1.2.4 Aprendizado por Gradiente

Redes neurais são geralmente treinadas utilizando otimizadores baseados em gradiente que trabalham de forma iterativa, levando o valor da função de custo para um valor bem baixo. A técnica de utilizar o gradiente para o aprendizado para minimizar uma função diferenciável associada é chamada de **Descida de Gradiente**. As redes neurais utilizam esse algoritmo através da **Retropropagação**, onde, como o próprio nome indica, o gradiente é propagado da saída da rede para as camadas anteriores.

A função de custo $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ é uma função das predições $\hat{\boldsymbol{y}}$ e das classes $\boldsymbol{y}$. Como $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ é diferenciável, o gradiente da função de custo ∇L no ponto $(\hat{\boldsymbol{y}}, \boldsymbol{y})$ será um vetor cujas componentes são as derivadas parciais de L no ponto $(\hat{\boldsymbol{y}}, \boldsymbol{y})$. O vetor gradiente pode ser interpretado como a direção e a taxa de crescimento mais rápida da função de custo L, isto é, o gradiente indica em que direção os parâmetros $\boldsymbol{\theta}$ devem seguir para maximizar a função L. Como o interesse é minimizar, utiliza-se o negativo do gradiente para atualizar os parâmetros $\boldsymbol{\theta}$ da rede a cada etapa do treinamento. Logo, o novo peso $\omega^{(k)}_{ij_n}$ e novo *bias* $b^{(k)}_{j_n}$ serão dados por:

$$\omega^{(k)}_{ij_n} = \omega^{(k)}_{ij} - \alpha \frac{\partial L}{\partial \omega^{(k)}_{ij}}, \tag{2.7}$$

$$b^{(k)}_{j_n} = b^{(k)}_{j} - \alpha \frac{\partial L}{\partial b^{(k)}_{j}}, \tag{2.8}$$

onde α é a **taxa de aprendizado** (Seção 2.1.2.6). Para se obter $\frac{\partial L}{\partial \omega^{(k)}_{ij}}$, basta recorrer à regra da cadeia e observar que $\frac{\partial L}{\partial \omega^{(k)}_{ij}} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\dots} \times \dots \times \frac{\dots}{\partial \omega^{(k)}_{ij}}$. A Figura 2.4 apresenta o exemplo da Figura 2.3 desta vez com a retropropagação gerando os novos pesos $\omega^{(k)}_{ij_n}$ e bias $b^{(k)}_{j_n}$. Na figura se encontram presentes as equações de atualização apenas para os pesos $\omega^{(1)}_{11}$ e $\omega^{(1)}_{12}$ e o *bias* $b^{(1)}_{1}$, porém a lógica é semelhante para todos os outros parâmetros do modelo.

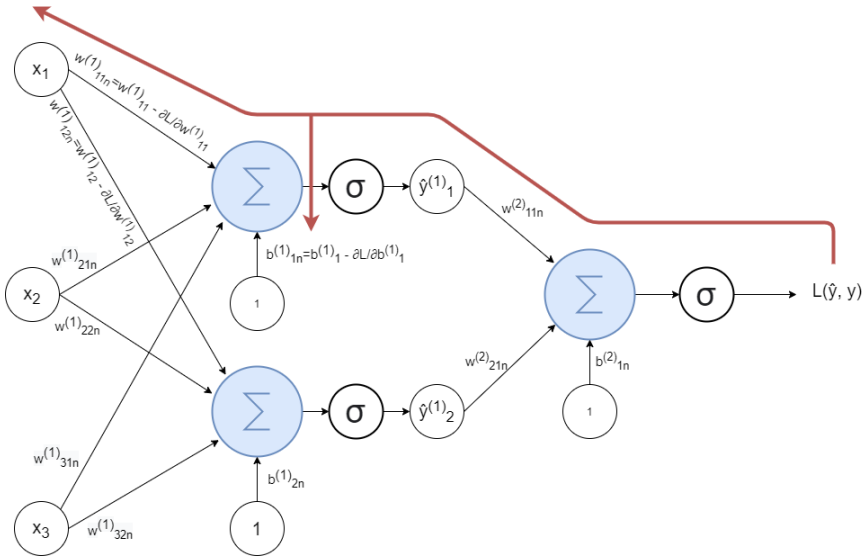


Figura 2.4: A **retropropagação**, utilizando a regra da cadeia, propaga os gradientes para as camadas anteriores da rede neural a partir da função de custo $L(y, \hat{y}_{21})$ obtida na saída comparando-se o resultado da rede com o resultado esperado. Com os gradientes, utiliza-se o algoritmo de **descida de gradiente** para atualizar os pesos a cada nova etapa de aprendizado.

2.1.2.5 *Overfitting* e *Underfitting*

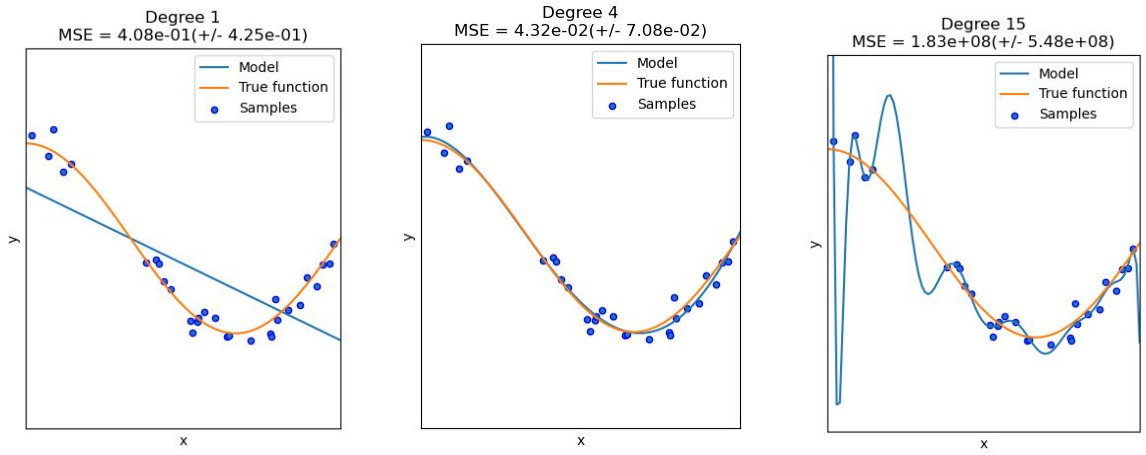
Um desafio central em aprendizado de máquina é a necessidade de o algoritmo obter boa performance em um conjunto de dados novo, não visto ainda, e não apenas naquele em que o

modelo foi treinado. Essa habilidade se refere à capacidade de **generalização** do modelo.

Além do **erro de treinamento** ser baixo (erro obtido durante o processo de treinamento), deseja-se que o **erro de teste** ou **erro de generalização** seja também baixo. Usualmente, o erro de generalização de um modelo é estimado baseando-se no seu desempenho em um **conjunto de teste**. O conjunto de teste pode ser derivado de uma porcentagem do conjunto de dados total disponível, como por exemplo reservar 20% do conjunto total para teste e os 80% restantes para treinamento.

Quando ocorre do modelo não obter um erro de treinamento baixo o suficiente quanto se é desejável, por quaisquer fatores que sejam, tem-se o subajuste (*underfitting*). O sobreajuste (*overfitting*) ocorre quando a diferença entre o erro de treinamento e o erro de teste é significativamente alta.

Modelos de aprendizado de máquina apresentam melhor desempenho geralmente quando sua **capacidade** é apropriada para a verdadeira complexidade de uma tarefa empregada e para a quantidade de dados de treinamento que são providos. A Figura 2.5 apresenta um exemplo disso, onde são comparados preditores polinomiais de graus 1, 4 e 15, respectivamente, na tarefa de aproximação de uma parte da função cosseno.



(a) (b) (c)

Figura 2.5: Exemplos de (a) *underfitting*, (b) aproximação ideal e (c) *overfitting*. Um modelo de baixa capacidade como na figura (a), sendo linear, não é capaz de aproximar a função o suficiente. Um modelo de grau 4 como o da figura (b) possui capacidade suficiente para aproximar a função quase que com perfeição. Um modelo de grau 15 possui exagerada capacidade para a tarefa e, embora seja capaz de aceitar todos as amostras do treinamento, não é capaz de generalizar corretamente para novar entradas [2].

2.1.2.6 Taxa de Aprendizado - *Learning Rate*

A taxa de aprendizado se refere à constante numérica que é multiplicada pelos gradientes obtidos no algoritmo de retropropagação para se atualizar os parâmetros aprendíveis da rede. Na

equação de atualização de um peso arbitrário $\omega_{ijk_n} = \omega_{ijk} - \alpha \frac{\partial L}{\partial \omega_{ijk}}$, a taxa de aprendizado é dada pelo parâmetro α. Valores típicos utilizados para α estão entre 10^{-1} e 10^{-5}, dependendo da tarefa e do modelo utilizado.

Diferentes algoritmos de otimização (ou simplesmente "otimizadores"), como será visto na Seção 2.1.2.9, apresentam diferentes formas de adaptar dinamicamente a taxa de aprendizado de forma a melhorar a performance do modelo durante o treinamento.

A taxa de aprendizado constitui o **hiperparâmetro** (Seção 2.1.2.12) mais importante de se definir para o treinamento de um modelo de aprendizado profundo.

2.1.2.7 Lote - *Batch*

Em aprendizado profundo, o termo lote ou *batch* se refere à quantidade de amostras enviadas ao modelo durante uma única iteração. Uma das principais vantagens de se treinar o modelo em tamanhos de lote maiores que 1 é que é possível tirar proveito do paralelismo computacional fornecido pelas GPUs recentes, onde várias amostras podem ser processadas simultaneamente de forma a acelerar drasticamente a velocidade com a qual o modelo é treinado.

O tamanho do lote é um hiperparâmetro ajustável. Há diversos trabalhos na literatura que estudam metodologias para dimensionar corretamente o tamanho do lote para o treinamento de um modelo para classificação de imagens. Para efeitos de regularização, tamanhos de lote pequenos já foram recomendados [29]. Outros demonstraram que, no caso do conjunto de dados CIFAR-10 [30], um tamanho ótimo para o lote está na ordem de 80 imagens [31]. Outros artigos recentes utilizam a razão entre a taxa de aprendizado e o tamanho do lote como guia para o treinamento do modelo [32] [33].

Para muitas aplicações práticas, o ideal torna-se apenas obter o melhor desempenho possível minimizando o tempo computacional necessário. Diferente do hiperparâmetro taxa de aprendizado, o tamanho do lote afeta significativamente o tempo de treinamento do modelo. *Leslie N. Smith* explora isso em [34] e propõe que, juntamente à política de *taxa de aprendizado cíclico* [35], valores elevados para o tamanho de lote possam ser utilizados de forma a minimizar o tempo de treinamento e obter elevada acurácia do modelo.

2.1.2.8 Épocas

Épocas dizem respeito ao número de passagens do conjunto inteiro de dados de treinamento pelo modelo. Se o conjunto de dados de treinamento possui N amostras e está organizado em lotes de tamanhos iguais b, então uma época será contada após $\frac{N}{b}$ iterações do modelo. A quantidade de épocas é um hiperparâmetro ajustável e deve ser dimensionado de acordo com a tarefa específica.

2.1.2.9 Algoritmos de Otimização

Como foi mencionado na Seção 2.1.2.6, existem diferentes algoritmos de otimização propostos ao longo dos anos para solucionar diversos problemas que podem vir a surgir no treinamento de modelos de aprendizado profundo.

O método de atualização dos parâmetros explorado na Seção 2.1.2.4 é chamado de **Descida de Gradiente Estocástico** (ou SGD), onde a taxa de aprendizado α é constante durante todo treinamento e o tamanho de lote é igual a 1. Quando a atualização é feita utilizando a média do custo das amostras de um lote, o método recebe o nome de **Descida de Gradiente Estocástico em mini-Lotes** (*Mini-batch Stochastic Gradient Descent*) [36]. Esse método apresenta alguns problemas:

- Escolher um valor para a taxa de aprendizado pode ser difícil. Uma taxa de aprendizado muito pequena leva a uma convergência do modelo significativamente lenta, enquanto uma taxa muito alta pode impedir a convergência e causar a oscilação da função de custo ou até mesmo a divergência [36].
- A taxa de aprendizado se aplica igualmente à todos os parâmetros. Se os dados em questão forem esparsos e os recursos possuírem frequências muito diferentes, podemos não querer atualizar todos eles na mesma extensão, mas realizar uma atualização maior para recursos que ocorrem raramente [36].
- Minimizar a função de custo para tarefas complexas muitas vezes é minimizar uma função fortemente não convexa. Um problema que surge dessa propriedade é muitas vezes a rede neural ficar "presa"em um mínimo local ou um ponto de "descanso", onde uma dimensão curva positivamente e outra curva negativamente. O gradiente no entorno desse ponto é próximo de zero, tornando notoriamente difícil para o SGD escapar [36].

O algoritmo do SGD, pela Equação 2.8, dará prioridade a uma forte atualização na direção em que $\frac{\partial L}{\partial \omega_{ijk}}$ for maior. Em "ravinas", regiões em que a superfície da função de custo se curva com muito mais intensidade em uma dimensão do que na outra, isso resulta em uma oscilação desnecessária dos parâmetros antes de progredirem para a região de mínimo. **Momento** [37] é uma técnica que corrige esse problema, acelerando a convergência do algoritmo SGD na direção relevante do gradiente e amenizando essas oscilações. Ele realiza isso adicionando uma fração γ do vetor de atualização da iteração passada $v^{(k)}_{ij,t-1}$ ao atual vetor de atualização $v^{(k)}_{ij,t}$ (Equações 2.11 e 2.12) para um peso $\omega^{(k)}_{ij,t}$ ou *bias* $b^{(k)}_{j,t}$ qualquer. Com $\gamma \leq 1$, um valor típico para γ é 0.9 [36]. As Equações 2.11 e 2.12 abaixo mostram como seria a atualização de um parâmetro $\omega^{(k)}_{ij,t}$ qualquer:

$$v^{(k)}_{ij,t} = \gamma v^{(k)}_{ij,t-1} + \frac{\partial L}{\partial \omega^{(k)}_{ij,t}}, \tag{2.9}$$

$$\omega^{(k)}_{ij,t} = \omega^{(k)}_{ij,t-1} - v^{(k)}_{ij,t}. \tag{2.10}$$

onde o termo v é frequentemente chamado "velocidade". O algoritmo de atualização utilizando momento frequentemente fornece uma convergência mais rápida do modelo. Uma analogia co-

mumente feita é a de uma bola descendo uma colina. Conforme a bola desce, ela adquire mais momento, se tornando cada vez mais rápida, seguindo em direção à base da colina, que aqui seria um mínimo da função da custo. Todavia, uma bola descendo uma colina sem qualquer controle não é satisfatório. É preciso ter noção de que se deve desacelerar antes que a colina comece a subir novamente em algum ponto [36].

A **Descida Acelerada do Gradiente de Nesterov** (*Nesterov Accelerated Gradient (NAG)*) [38] é uma forma de dar ao termo do momento essa presciência. Tendo conhecimento dos parâmetros atuais $\boldsymbol{\theta}$ e sabendo que a próxima atualização levará em conta a última feita através do termo $\boldsymbol{\theta}_t - \gamma \boldsymbol{v}_{t-1}$ presente na combinação das Equações 2.11 e 2.12, é possível ter uma ideia aproximada da próxima posição dos parâmetros. Agora é possível calcular o gradiente em relação não aos parâmetros $\boldsymbol{\theta}$ do modelo mas em relação à posição futura aproximada $\boldsymbol{\theta}_t - \gamma \boldsymbol{v}_{t-1}$ dos parâmetros [36].

$$v_{ij,t}^{(k)} = \gamma v_{ij,t-1}^{(k)} + \frac{\partial L}{\partial(\omega_{ij,t}^{(k)} - \gamma v_{ij,t-1}^{(k)})}, \tag{2.11}$$

$$\omega_{ij,t}^{(k)} = \omega_{ij,t-1}^{(k)} - v_{ij,t}^{(k)}. \tag{2.12}$$

Essa atualização antecipada impede que o gradiente avance com muita velocidade, resultando em uma crescente responsividade do modelo, o que aumentou significativamente a performance das RNAs em várias tarefas [39].

No aprendizado profundo moderno, existem uma gama de algoritmos de otimização diferentes disponíveis. Esses algoritmos comumente recebem apenas o nome de **otimizadores**. Cada um implica técnicas diferentes para atacar um determinado problema na otimização dos modelos de aprendizado profundo nas diversas áreas de aplicação possíveis. Alguns exemplos clássicos e amplamente utilizados de otimizadores e seus princípios são citados a seguir:

- *Adagrad* [40]: Adapta a taxa de aprendizado para cada parâmetro individualmente, realizando pequenas atualizações para parâmetros associados com recursos altamente frequentes e grandes atualizações para parâmetros associados com recursos mais raros.
- *Adadelta* [41]: Uma extensão do *Adagrad* que procura reduzir sua "agressividade", diminuindo a taxa de aprendizado monotonicamente, restringindo o acúmulo de gradientes anteriores para um valor máximo.
- *Adam* [42]: Em adição em manter uma média exponencial decadente dos gradientes quadráticos passados do modelo para atualização dos parâmetros, também mantém uma média exponencial decadente dos momentos passados.

Dentre os citados, o *Adam* é um dos mais amplamente utilizados atualmente. É um algoritmo estável que devolve boa velocidade de convergência durante o treinamento em tempos menores quando comparado a outros algoritmos. Otimizadores novos são propostos todos os anos, e alguns mais recentes já demonstram superar o *Adam* em vasta maioria das aplicações, como por exemplo o *AdaBelief* [43].

2.1.2.10 Regularização

Como foi discutido na Seção 2.1.2.5, um problema central no campo de aprendizado profundo é como fazer um modelo se sair bem não apenas no conjunto de dados de treinamento, mas também no de teste. O conjunto de estratégias no campo do aprendizado de máquina que são explicitamente projetadas para reduzir o erro de teste, possivelmente às custas de aumentar o erro de treinamento, se chama **regularização**. Existem várias estratégias de regularização disponíveis atualmente e o desenvolvimento de novas é um dos grandes tópicos de pesquisa na área de aprendizado de máquina [1].

Uma estratégia frequentemente utilizada é a **penalidade de parâmetros** (*Parameter Norm Penalties*). Trata-se de limitar a capacidade de um modelo adicionando um parâmetro de penalidade $\Omega(\boldsymbol{\theta})$ à função de custo $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$. A função de custo regularizada é denotada como $\tilde{L}$:

$$\tilde{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \eta\Omega(\boldsymbol{\theta}), \tag{2.13}$$

onde $\eta \in [0, \infty)$ é um hiperparâmetro que pesa a contribuição do parâmetro de penalidade $\Omega(\boldsymbol{\theta})$. Quando o algoritmo minimizar a função $\tilde{L}$ irá minimizar tanto a função de custo L quando alguma medida do tamanho dos parâmetros $\boldsymbol{\theta}$. Diferentes escolhas para Ω resultam em diferentes soluções preferenciais [1].

No aspecto de redes neurais, tipicamente se escolhe utilizar o parâmetro de penalidade Ω apenas para os pesos $\boldsymbol{\omega}$, mantendo os *bias* sem regularização. Os *bias* geralmente requerem menos dados que os pesos para serem ajustados corretamente. Cada peso especifica como duas variáveis interagem. Ajustar os pesos requer observar ambas as variáveis em uma variedade de condições. Os *bias* controlam apenas uma única variável. Isso significa que não se introduz muita variância mantendo os *bias* sem regularização. Dessa forma, utiliza-se o vetor $\boldsymbol{\omega}$ para indicar todos os pesos que devem ser afetados pela penalidade, enquanto o vetor $\boldsymbol{\theta}$ denota todos os parâmetros, incluindo ambos $\boldsymbol{\omega}$ e os parâmetros não regularizados [1].

No campo da regularização por penalidade de parâmetros, as duas estratégias mais comumente utilizadas no aprendizado de máquina prático são listadas e explicadas a seguir:

- Regularização L^2: Também conhecida como **decaimento de pesos**, esta regularização aproxima os pesos da origem adicionando um termo $\Omega(\boldsymbol{\theta}) = \frac{1}{2}||\boldsymbol{\omega}||_2^2 = \sum_i \omega_i^2$ à função de custo L. Para entender o comportamento do decaimento de pesos torna-se útil analisar o seu gradiente. Considerando que $\hat{y}$ é uma função dos parâmetros $\boldsymbol{\theta}$ e da entrada $\boldsymbol{x}$, podemos reescrever $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ como $L(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y})$. Para simplificar, assume-se a ausência do parâmetro de *bias*, portanto $L(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y})$ é apenas $L(\boldsymbol{\omega}, \boldsymbol{x}, \boldsymbol{y})$. O modelo terá então a seguinte função de custo $\tilde{L}$:

  $$\tilde{L}(\boldsymbol{\omega}, \boldsymbol{x}, \boldsymbol{y}) = L(\boldsymbol{\omega}, \boldsymbol{x}, \boldsymbol{y}) + \frac{\eta}{2}\boldsymbol{\omega}^T\boldsymbol{\omega}, \tag{2.14}$$

  com o gradiente a seguir

  $$\nabla_{\boldsymbol{\omega}}\tilde{L}(\boldsymbol{\omega}, \boldsymbol{x}, \boldsymbol{y}) = \nabla_{\boldsymbol{\omega}}L(\boldsymbol{\omega}, \boldsymbol{x}, \boldsymbol{y}) + \eta\boldsymbol{\omega}. \tag{2.15}$$

Ao se realizar uma atualização dos pesos, a equação do SGD é realizada:

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \alpha(\nabla_{\boldsymbol{\omega}} L(\boldsymbol{\omega}, \boldsymbol{x}, \boldsymbol{y}) + \eta\boldsymbol{\omega}), \tag{2.16}$$

que escrita de outra forma, neste caso, se torna

$$\boldsymbol{\omega} \leftarrow (1 - \alpha\eta)\boldsymbol{\omega} - \alpha\nabla_{\boldsymbol{\omega}} L(\boldsymbol{\omega}, \boldsymbol{x}, \boldsymbol{y}). \tag{2.17}$$

Dessa forma, observa-se que a adição do termo de decaimento de pesos acarreta na modificação da regra de aprendizado, encolhendo o vetor de pesos de forma multiplicativa por um fator constante a cada iteração imediatamente antes de realizar a atualização pelo gradiente. A ideia de penalizar os parâmetros do modelo vem do fato que um modelo com parâmetros menores tende a ser mais simples, logo, penalizar o tamanho dos parâmetros da rede é uma solução para se reduzir a complexidade do modelo. Essa estratégia é útil para evitar o sobreajuste [1].

- Regularização L^1: A regularização L^1 provê como parâmetro de penalidade a soma absoluta dos pesos, isto é, $\Omega(\boldsymbol{\theta}) = ||\boldsymbol{\omega}||_1 = \sum_i |\omega_i|$. De forma semelhante à feita para a regularização L^2, a função de custo modificada $\tilde{L}$ e o gradiente estão representados a seguir:

$$\tilde{L}(\boldsymbol{\omega}, \boldsymbol{x}, \boldsymbol{y}) = L(\boldsymbol{\omega}, \boldsymbol{x}, \boldsymbol{y}) + \eta||\boldsymbol{\omega}||_1, \tag{2.18}$$

$$\nabla_{\boldsymbol{\omega}}\tilde{L}(\boldsymbol{\omega}, \boldsymbol{x}, \boldsymbol{y}) = \nabla_{\boldsymbol{\omega}} L(\boldsymbol{\omega}, \boldsymbol{x}, \boldsymbol{y}) + \eta sign(\boldsymbol{\omega}), \tag{2.19}$$

onde $sign(\boldsymbol{\omega})$ extrai apenas o sinal de todos os pesos do modelo. Essa regularização resulta em pesos bem mais **esparsos**, isto é, vários pesos com valores nulos. Essa propriedade torna a regularização L^1 útil para **seleção de características** (*feature selection*). Seleção de características simplifica um modelo de aprendizado de máquina escolhendo qual subamostra das características disponíveis devem ser utilizadas. Um exemplo desse uso é o modelo LASSO [44, 1].

A melhor forma de fazer um modelo de aprendizado de máquina generalizar melhor é treinar em mais dados. Na prática, claro, a quantidade de dados disponíveis é limitada. Uma forma de contornar esse problema é criar dados falsos e adicioná-los ao conjunto de dados. Essa técnica de chama **aumento do conjunto de dados** (*dataset augmentation*) [1].

Essa estratégia é mais fácil de ser aplicada para tarefas de classificação. Um classificador precisa receber uma entrada arbitrária $\boldsymbol{x}$ complicada, com alta dimensionalidade e resumir como uma única categoria y. Isso significa que a principal tarefa que um classificador enfrenta é ser invariante a uma variedade de transformações. É possível gerar novos pares $(\boldsymbol{x}, y)$ facilmente apenas transformando as entradas $\boldsymbol{x}$ no nosso conjunto de treinamento. Um exemplo é um modelo que classifica se uma imagem contém um carro ou uma moto. Se uma imagem contém um carro, caso a imagem seja espelhada horizontalmente, continuará contendo um carro. O mesmo é válido para o caso da moto. Caso o brilho da imagem mude um pouco, uma parte da imagem seja removida (de forma que ainda seja plenamente possível identificar o veículo) ou a imagem seja ampliada, a classificação do veículo continua válida. Observa-se através desse exemplo que a

técnica de aumento do conjunto de dados é válida para tarefas de classificação no âmbito da visão computacional e de fato é uma das técnicas de regularização mais utilizadas [1].

Essa abordagem, todavia, não é aplicável em várias outras tarefas. Por exemplo, é difícil gerar novos dados falsos para uma tarefa de estimação de densidade a não ser que o problema de estimar a densidade já tenha sido resolvido [1].

Ao se aplicar a técnica de aumento do conjunto de dados é necessário tomar certas precauções. Certas transformações podem mudar a classe correta da amostra. Por exemplo, um sistema de reconhecimento de caracteres necessita reconhecer as diferenças entre as letras "b"e "d"e também a diferença entre os números "6"e "9". Logo, espelhamento horizontal e vertical não são transformações apropriadas para essa tarefa [1].

Uma técnica também frequentemente utilizada é a de **interrupção precoce**. Quando se treina modelos complexos com capacidade representacional suficiente para sobreajustar a tarefa, comumente observa-se que o erro de treinamento cai continuamente ao longo do tempo, porém o erro de teste começa a aumentar após algum tempo [1]. Esse comportamento é bem exemplificado na Figura 2.6.

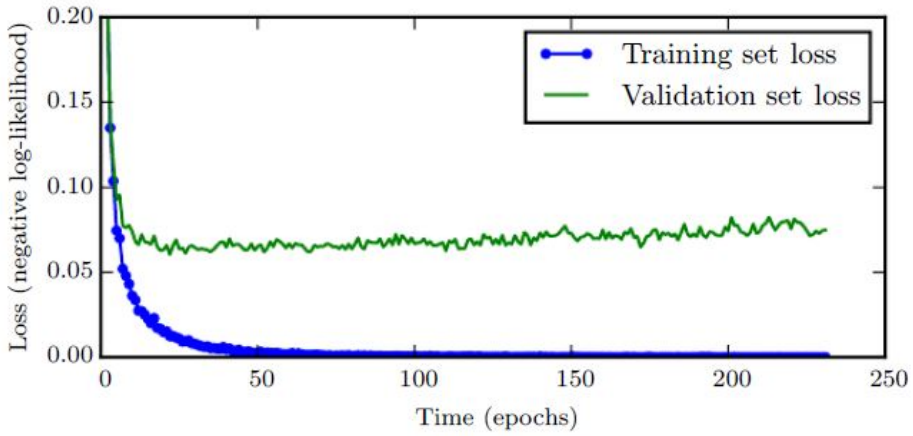


Figura 2.6: Curvas de aprendizado mostrando como valor da função de custo log-verossimilhança negativa muda ao longo do tempo (indicado neste caso como o número de épocas). A curva azul representa o custo no conjunto de dados de treinamento e a curva verde representa o custo no conjunto de dados de teste. Neste exemplo, o modelo é uma rede neural. Observa-se que o valor de custo para o conjunto de treinamento decresce continuamente porém o valor de custo para o conjunto de teste decresce até certo ponto (mais ou menos até a época 25) e então começa a crescer [1].

Observando esse comportamento, uma estratégia que se torna útil para se extrair a melhor fase do modelo (momento que o modelo apresentou menor custo para o conjunto de testes) é a interrupção do treinamento quando se perceber que o erro de teste não está mais caindo, ou treinar o modelo durante todo o período especificado e, ao final do treinamento, retornar apenas os parâmetros de quando o modelo apresentou o melhor desempenho ao invés de interromper completamente o treinamento [1].

Outra técnica de regularização disponível e que é considerada computacionalmente barata e poderosa é o ***Dropout*** [45]. *Dropout* consiste em anular unidades da rede de forma aleatória com certa probabilidade, de forma que suas saídas se tornem zero. Isso pode ser visto como o treinamento de várias sub-redes que podem ser formadas removendo-se unidades de uma rede base, como ilustrado na Figura 2.7. Nas redes neurais modernas, é possível remover uma unidade de forma efetiva multiplicando sua saída por zero. Esse procedimento talvez necessite ser adaptado para modelos como por exemplo redes de função de base radial, que utilizam a diferença entre o estado da unidade e algum valor de referência. O *Dropout* é apresentado como uma saída multiplicada por zero a fim de simplicidade, porém pode ser trivialmente adaptado para trabalhar com outras operações que removem uma unidade da rede [1].

Foi demonstrado em [45] que a técnica de *Dropout* é mais efetiva que outras técnicas computacionalmente baratas de regularização, como decaimento de pesos. O *Dropout* pode também ser combinado com outros métodos de regularização de forma a entregar um desempenho ainda melhor.

Como mencionado, uma das vantagens é que o *Dropout* é computacionalmente barato. A utilização de *Dropout* durante treinamento requer apenas $O(n)$ computações por exemplo por atualização para gerar n número binários aleatórios e multiplicá-los pelo estado. Dependendo da implementação, pode também requerer apenas $O(n)$ de memória para armazenar esses números binários até o estágio de retropropagação [1].

2.1.2.11 Redes Convolucionais

Redes convolucionais [46], também conhecidas como **redes neurais convolucionais** ou CNNs , são um tipo especializado de redes neurais para processamento de dados que possuem uma topologia tipo grade bem conhecida. Um exemplo seria dados de séries temporais, que podem ser pensados como uma estrutura 1-D (unidimensional) de grade tendo suas amostras obtidas em intervalos de tempo regulares, e dados de uma imagem, que podem ser pensados como uma grade 2-D (bidimensional) de pixels. Redes neurais convolucionais tem sido frequentemente bem sucedidas em aplicações práticas. O nome "redes neurais convolucionais" indica que a rede aplica uma operação matemática chamada **convolução**. Convolução é apenas um tipo específico de operação linear. Dessa forma, temos que redes neurais convolucionais são apenas redes neurais que utilizam convoluções no lugar de multiplicações matriciais genéricas em pelo menos uma de suas camadas [1].

A pesquisa na área de redes neurais convolucionais prossegue tão rapidamente que uma nova melhor arquitetura para dada tarefa é anunciada a cada poucas semanas ou meses, tornando rapidamente obsoleta a descrição de uma "melhor arquitetura" [1].

A operação de convolução é tipicamente denotada com um asterisco:

$$s(t) = (x * w)(t). \tag{2.20}$$

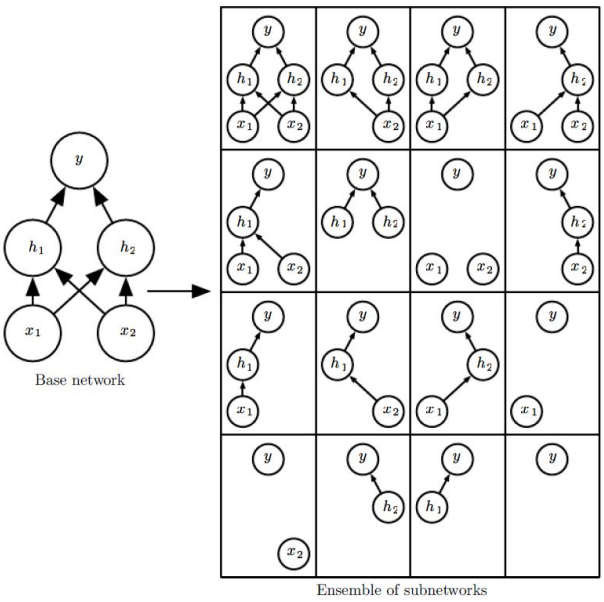


Figura 2.7: *Dropout* treina um conjunto que consiste em todas as sub-redes que podem ser construídas removendo unidades de uma rede de base subjacente. Aqui, começa-se com uma rede base com duas camadas, possuindo duas unidades na primeira camada e uma na segunda. São dezesseis possíveis subconjuntos dessas quatro unidades. Mostramos todas as dezesseis sub-redes que podem ser formadas eliminando diferentes subconjuntos de unidades da rede original. Neste pequeno exemplo, uma grande proporção das redes resultantes não tem unidades de entrada ou nenhum caminho conectando a entrada para a saída. Este problema se torna insignificante para redes com camadas largas, onde a probabilidade de descartar todos os caminhos possíveis de entradas para saídas torna-se menor [1].

Na terminologia de redes neurais convolucionais, o primeiro argumento (no exemplo da Equação 2.20, a função x) para a convolução é frequentemente denotada como **entrada**, e o segundo argumento (no exemplo da Equação 2.20, a função w) é o **kernel**. A saída frequentemente é chamada de **mapa de features** (*feature map*).

Nas aplicações de aprendizado de máquina, a entrada geralmente é um vetor de dados multidimensional, e o kernel é um vetor multidimensional de parâmetros que são adaptados pelo algoritmo de aprendizado. A designação dada a esses vetores multidimensionais é **tensores**.

A Figura 2.8 mostra um exemplo da operação de convolução aplicada a um tensor bidimensional.

O uso de convoluções alavanca três importantes ideias que podem ajudar um sistema de aprendizado de máquina: **interações esparsas**, **compartilhamento de parâmetros** e **representações equivariantes**.

Redes neurais tradicionais utilizam multiplicação matricial com uma matriz de parâmetros com cada parâmetro separado descrevendo a interação entre cada unidade de entrada e cada unidade de saída. Isso significa que cada unidade de saída interage com todas as unidades de entrada. Redes

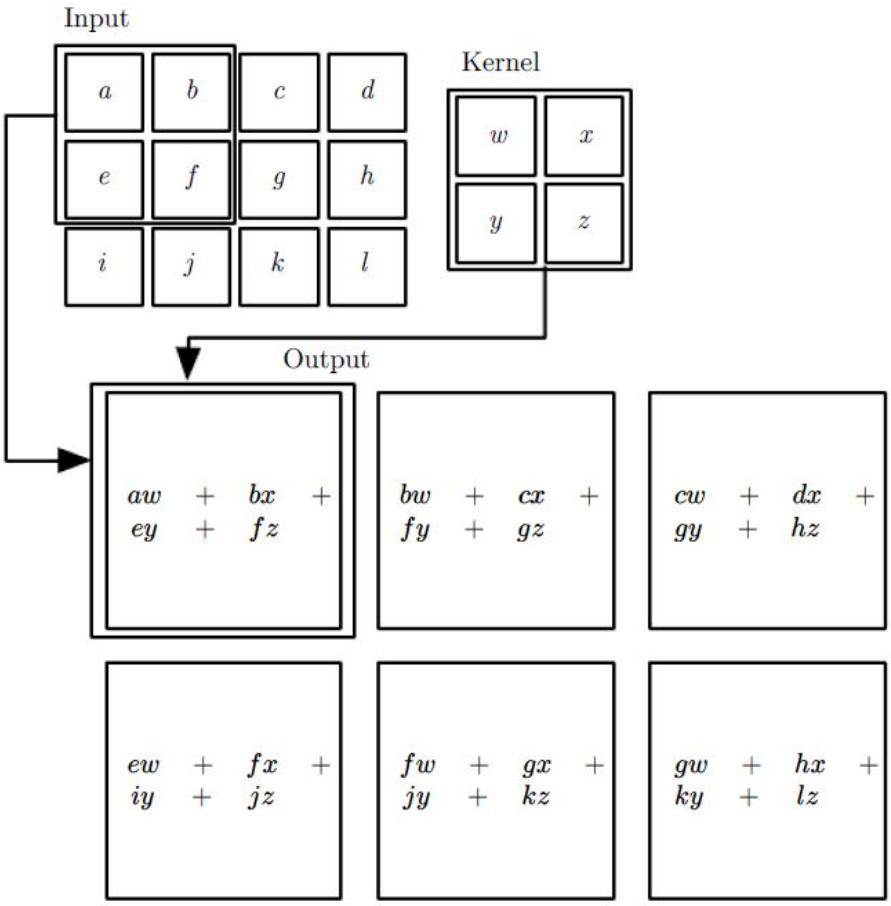


Figura 2.8: Um exemplo da operação de convolução em um tensor bidimensional. A saída aqui está restrita apenas às posições a qual o kernel permanece inteiramente dentro da imagem, caracterizando uma convolução "válida" em alguns contextos. As setas indicam como a metade superior esquerda do tensor de saída é formada aplicando o kernel à correspondente região superior esquerda do tensor de entrada [1].

convolucionais, no entanto, possuem **interações esparsas**. Isso é realizado tornando o kernel menor que a entrada. Por exemplo, no processamento de imagem, uma imagem de entrada pode conter milhões de pixels, porém é possível detectar pequenas características bem significativas tais como arestas com kernels que ocupam apenas algumas dezenas ou centenas de pixels. Isso significa que é possível armazenar menos parâmetros, o que reduz os requisitos de memória do modelo e melhora sua eficiência estatística. Também significa que computar a saída requer menos operações.

Compartilhamento de parâmetros se refere à utilização do mesmo parâmetro por mais de uma função dentro do modelo. Em uma rede neural tradicional, cada elemento da matriz de pesos é utilizado exatamente uma vez ao computar a saída de uma camada. Em uma rede convolucional, cada membro do kernel é utilizado em todas as posições da entrada (exceto talvez nos pixels nas bordas da imagem, dependendo das decisões de projeto relativas ao tratamento das bordas). O compartilhamento de parâmetros utilizado pela operação de convolução significa que, ao invés de

aprender um conjunto de parâmetros diferentes para cada localização, é aprendido apenas um conjunto.

Figura 2.9: Eficiência em detecção de bordas. A imagem na direita foi formada obtendo cada pixel na imagem original e subtraindo o valor do seu pixel vizinho à esquerda. Isso exalta todas as bordas alinhadas verticalmente. Ambas as imagens possuem 280 pixels de altura. A imagem da esquerda possui 320 pixels de largura, enquanto a da direita possui 319 pixels de largura. Essa transformação pode ser descrita por um kernel de convolução contendo dois elementos, e requer $319 \times 280 \times 3 = 267,960$ operações em ponto flutuante (duas multiplicações e uma adição por pixel de saída) para se computar a saída utilizando convolução. Para descrever a exata mesma transformação utilizando multiplicação matricial necessitaria $320 \times 280 \times 319 \times 280$, ou mais ou menos oito bilhões, de entradas na matriz, tornando a convolução quatro bilhões de vezes mais eficiente para representar esta transformação [1].

Um exemplo dos dois princípios mencionados anteriormente em ação se encontra na Figura 2.9.

No caso da convolução, a forma particular do compartilhamento de parâmetros fornece à camada a propriedade de **equivariância** à translação. Dizer que uma função é equivariante significa que se a entrada mudar, a saída muda na mesma forma. No caso de imagens, convoluções criam mapas bidimensionais onde certas características aparecem na entrada. Se o objeto se mover na entrada, sua representação irá se mover a mesma quantidade na saída. Vale ressaltar que convoluções não são naturalmente equivariantes à toda e qualquer transformação, como por exemplo mudança de escala e rotação de uma imagem. Outros mecanismos são necessários para lidar com esses tipos de transformação.

Uma camada típica de uma rede neural consiste na verdade de três estágios (Figura 2.11. No primeiro estágio, a camada realiza várias convoluções em paralelo para produzir um conjunto de ativações lineares. No segundo estágio, cada ativação linear passa por uma função de ativação não linear, como discutido na Seção 2.1.2.2. No terceiro estágio, é utilizada uma função de *pooling* para modificar a saída da camada posteriormente.

Uma função de *pooling* substitui a saída de uma rede em uma localidade específica por uma síntese estatística das saídas próximas. Por exemplo, a função de ***max pooling*** [47] retorna o valor máximo em uma vizinhança retangular. Outras funções de *pooling* populares incluem realizar a média de uma vizinhança retangular, a normalização L^2 de uma vizinhança retangular, ou a

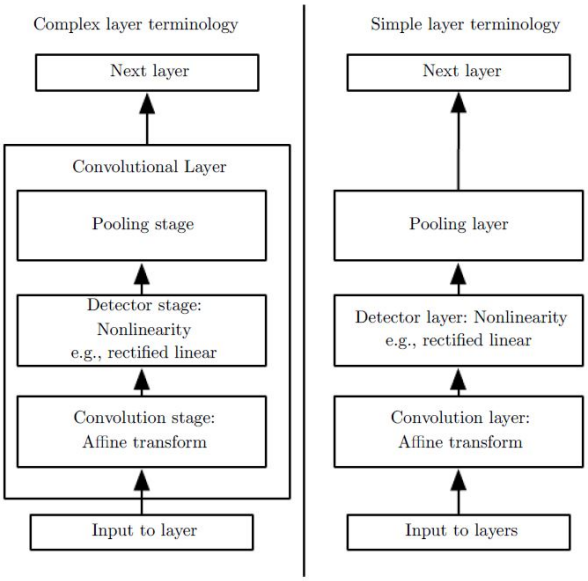


Figura 2.10: Os componentes de uma rede neural convolucional típica. Existem dois conjuntos de terminologias comumente utilizadas. Na terminologia da esquerda, a rede convolucional é vista como um número pequeno de camadas relativamente complexas, com cada camada possuindo vários "estágios". Na terminologia da direita, a rede é vista como um grande número de camadas mais simples. Na terminologia da direita, nem toda camada possui parâmetros [1].

média ponderada da vizinhança retangular baseada na distância do pixel central. Em todo caso, o *pooling* ajuda a realizar a representação aproximadamente **invariante** a pequenas translações da entrada. Invariante a translações significa que se a entrada sofrer uma translação por uma pequena quantidade, os valores de máximo das saídas das funções de *pooling* não se alteram. Invariância a translações locais se torna uma propriedade útil se o interesse maior for a presença ou não de uma característica específica e não onde exatamente ela está.

2.1.2.12 Hiperparâmetros

Hiperparâmetros são parâmetros determinados fora do escopo de aprendizado do algoritmo. Isto significa que o usuário deve definir manualmente esses parâmetros antes de iniciar o processo de treinamento de um modelo.

Alguns exemplos de hiperparâmetros são listados abaixo:

- Taxa de aprendizado α (Seção 2.1.2.6): Valores típicos estão no intervalo $10^{-1} \geq \alpha \leq 10^{-5}$.
- Tamanho do lote B (Seção 2.1.2.7): Valores típicos estão no intervalo $10 \geq B \leq 1000$.
- Número de épocas (Seção 2.1.2.8): Pode variar por ordens de magnitude a depender da tarefa. Valores típicos estão no intervalo de 10 a 1000.
- Momento γ (Seção 2.1.2.9): Valores típicos estão no intervalo $0 \geq \gamma < 1.0$.

- Regularização L^2 ou decaimento de pesos η (Seção 2.1.2.10): Valores típicos estão no intervalo $0.0 \geq \eta < 1.0$
- Tamanho da imagem: No caso de modelos que trabalhem com imagem, em alguns casos, é necessário informar o tamanho das imagens, seja o tamanho original ou o tamanho de pré processamento desejado. Dimensões típicas (considerando imagens quadradas) estão no intervalo de 24x24 pixels a 512x512 pixels.

2.1.2.13 Arquiteturas

No campo do aprendizado de máquina, mais especificamente no aprendizado profundo, o termo **arquitetura** é utilizado para se referir à planta de um modelo específico. O modelo com os parâmetros treinados em si não caracteriza a arquitetura, mas sim suas informações quanto ao número de camadas, tipos de camadas, blocos, quantidade de conexões, unidades ocultas, etc. É possível utilizar uma mesma arquitetura para atacar problemas diferentes, por exemplo, uma arquitetura treinada para classificação de imagens pode ser utilizada também para detecção de objetos e vice-versa.

Um exemplo de arquitetura no campo da visão computacional que representou um importante marco para a tarefa de classificação de imagens foi a **ResNet** [3] (muitas arquiteturas recebem um nome arbitrário antes do sufixo "Net" de forma a identificá-las mais facilmente por praticantes e pesquisadores da área). Uma forma comum de representar a arquitetura e sua profundidade é adicionar um sufixo numérico indicando o número de camadas, por exemplo, arquiteturas comuns para a ResNet incluem a **ResNet18**, a **ResNet34** e a **ResNet50**, indicando a arquitetura original da ResNet com 18, 34 e 50 camadas, respectivamente.

A ResNet alterou o bloco de convolução introduzido da Seção 2.1.2.11 para um **bloco residual** (Figura 2.11). Em contraste ao bloco de convolução, o bloco residual introduz uma conexão resíduo, permitindo que as entradas se propaguem da entrada para a saída do bloco diretamente. Esse formato permite o treinamento de arquiteturas bem mais profundas sem a forte interferência do **problema da dissipação do gradiente**, onde sucessivas multiplicação numéricas dos gradientes pela regra da cadeia para camadas mais profundas da rede acarretam em valores numéricos relativamente baixos de atualização para os parâmetros das camadas iniciais do modelo. Modelos montados sob essa arquitetura também são mais fáceis de otimizar.

Arquiteturas novas são propostas frequentemente, e diferentes arquiteturas são encontradas para diferentes áreas de pesquisa do aprendizado profundo. Abaixo são listadas, por área de pesquisa, algumas arquiteturas utilizadas no campo específico. Os exemplos estão restritos à arquiteturas apenas do campo de aprendizado profundo.

- Visão Computacional
  - **EfficientNet** [48]
  - **Vision Transformer** [49]
  - **Faster R-CNN** [50]

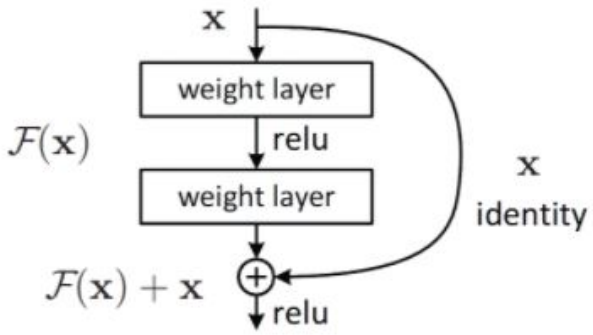


Figura 2.11: Bloco residual. Considerando os parâmetros da primeira camada como W_1, os parâmetros da segunda camada com W_2 e a entrada como x, a saída do bloco residual será dada por $y = W_2 r(W_1 x) + x$, onde r é a função $ReLU$ nesse caso [3].

  - **YOLOv4** [51]
  - **EfficientDet** [52]

- Processamento de Linguagem Natural
  - **SHA-RNN** [53]
  - **Transformer** [54]
  - **BERT** [55]
  - **GPT-3** [56]

- Dados Tabulares
  - **TabNet** [57]

- Previsão de Série Temporal
  - **Temporal Fusion Transformer** [58]

2.1.2.14 Transferência de Aprendizado - *Transfer Learning*

No cenário clássico de aprendizado de máquina supervisionado, se é necessário treinar um modelo para alguma tarefa e domínio A, assume-se que os dados rotulados para a tarefa e domínio em questão são fornecidos. Considera-se a tarefa como o objetivo que o modelo deseja atingir, como por exemplo reconhecer objetos em uma imagem, e o domínio como a fonte de onde os dados estão vindo. O modelo A é então treinado no conjunto de dados e é esperado que ele desempenhe bem em novos dados da mesma tarefa e domínio. Em outra ocasião, quando se tem dados para uma outra tarefa ou domínio B, torna-se necessária a utilização de dados rotulados novamente para que seja possível treinar um novo modelo B para que seja esperado que seu desempenho para o novo conjunto de dados seja bom.

Quando não se têm dados rotulados suficientes para a tarefa ou domínio o qual deseja-se treinar um modelo resiliente, o paradigma do aprendizado de máquina tradicional quebra. Se é desejado treinar um modelo para detectar carros em imagens noturnas, uma solução seria aplicar um modelo que foi treinado em um domínio similar, por exemplo detectar carros durante o dia. No entanto, na prática, frequentemente ocorre a deterioração ou o colapso na performance do modelo uma vez que este herdou o viés dos seus dados de treinamento originais e não sabe como generalizar para o novo domínio. Outro problema seria a reutilização de classes. Um modelo treinado para detectar marcas de carros específicos durante o dia não aceitaria quaisquer outras marcas que fossem necessárias também serem detectadas durante o período noturno.

A técnica de **transferência de aprendizado** (*transfer learning*) permite lidar com esses cenários aproveitando os dados rotulados já existentes de alguma tarefa de um domínio relacionada. A abstração de "conhecimento"do modelo que foi obtida durante o treinamento na tarefa original no conjunto de dados original é armazenada e posteriormente aplicada para outro problema específico relacionado. Na prática, deseja-se transferir a maior quantidade possível de conhecimento da fonte para o problema de interesse em questão.

A importância da transferência de aprendizado surge de uma dicotomia atual do uso do aprendizado de máquina na indústria: de um lado, ao longo dos últimos anos, a habilidade de se treinar modelos cada vez mais precisos se tornou viável. Chegou-se um ponto que, para muitas tarefas, modelos estado da arte atingiram um nível de desempenho tão alto que não é mais um obstáculo para usuários. Do outro lado, esses modelos bem sucedidos são imensamente dependentes de volumes de dados gigantescos. Para algumas tarefas e domínios, esses dados já estão disponíveis pois foram meticulosamente colhidos ao longo de muitos e muitos anos, como é o caso do conjunto de dados público do ImageNet [59], mas grandes conjuntos de dados geralmente são proprietários ou significativamente caros de serem obtidos, como é o caso em muitos conjuntos de dados voltados para fala.

Existem muitas formas de aplicar a transferência de aprendizado e conceitos mais específicos de como deve ser feito depende muitas vezes da área de conhecimento em que se deseja aplicar. Aplicações fora do campo da visão computacional estão fora do escopo desse trabalho e uma leitura recomendada se encontra em [60].

No campo da visão computacional, mais especificamente no reconhecimento de imagens, a transferência de aprendizado é realizada utilizando CNNs pré-treinadas. Mais especificamente utilizando a parte de extração de *features* da rede (caracterizada pelas camadas convolucionais).

Já foi demonstrado que camadas convolucionais inferiores tendem a capturar características de baixo nível da imagem [4], como arestas (Figura 2.12). Conforme se aprofunda nas camadas convolucionais da rede, características cada vez mais complexas começam a aparecer, como partes corporais de animais e rodas de carros.

As camadas finais da rede geralmente compreendem camadas totalmente conectadas, que nada mais são que perceptrons multicamadas 2.1.2.1. Essas camadas geralmente são associadas à captura de informações relevantes a tarefas específicas, por exemplo, a camada final totalmente conectada da ResNet18 indicaria quais *features* são relevantes para classificar uma imagem em uma

de 1000 categorias de objetos, considerando a arquitetura original proposta para o problema da ImageNet.

No entanto, embora saber que possuir bigodes, patas e pelos é necessário para identificar o animal como um gato, isso não ajuda na tarefa de identificar novos objetos ou em outras tarefas comuns no campo de visão computacional. O que ajuda, no entanto, são as representações que capturam informação geral de como uma imagem é composta e quais combinações de arestas e formatos ela contém. Como mencionado anteriormente, essa informação está contida nas camadas finais da rede.

Para uma nova tarefa então pode-se simplesmente utilizar as camadas de *features* da CNN estado da arte pré-treinada na ImageNet (exemplo) e treinar o novo modelo nessas *features* extraídas. Na prática, ou mantém-se os parâmetros pré-treinados fixos ou ajustam-se os parâmetros com um pequeno valor de taxa de aprendizado de forma a garantir que o conhecimento adquirido anteriormente não seja perdido.

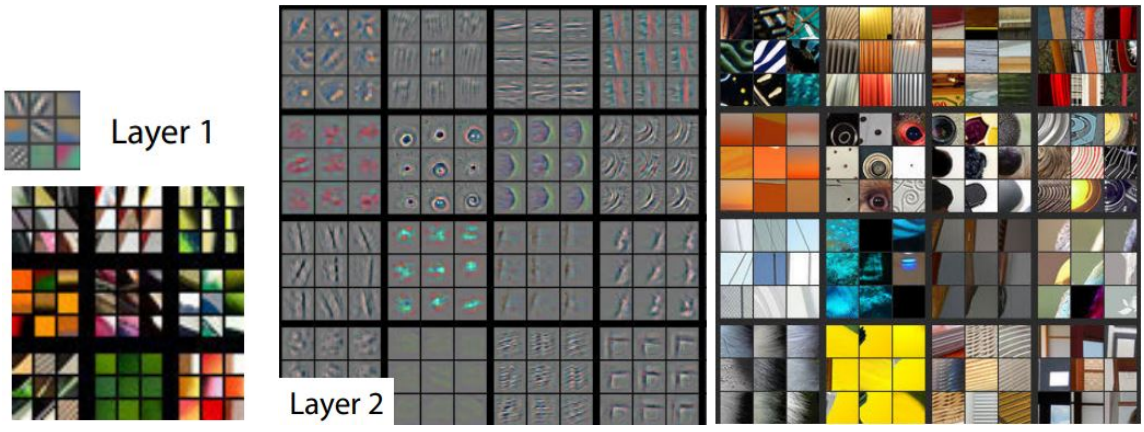
2.1.2.15 Métricas de Performance

Determinar os objetivos, em termos de que métrica de erro utilizar, é um primeiro passo necessário uma vez que a métrica de erro irá guiar todas as futuras ações. Também é necessário ter uma ideia do nível de performance desejado.

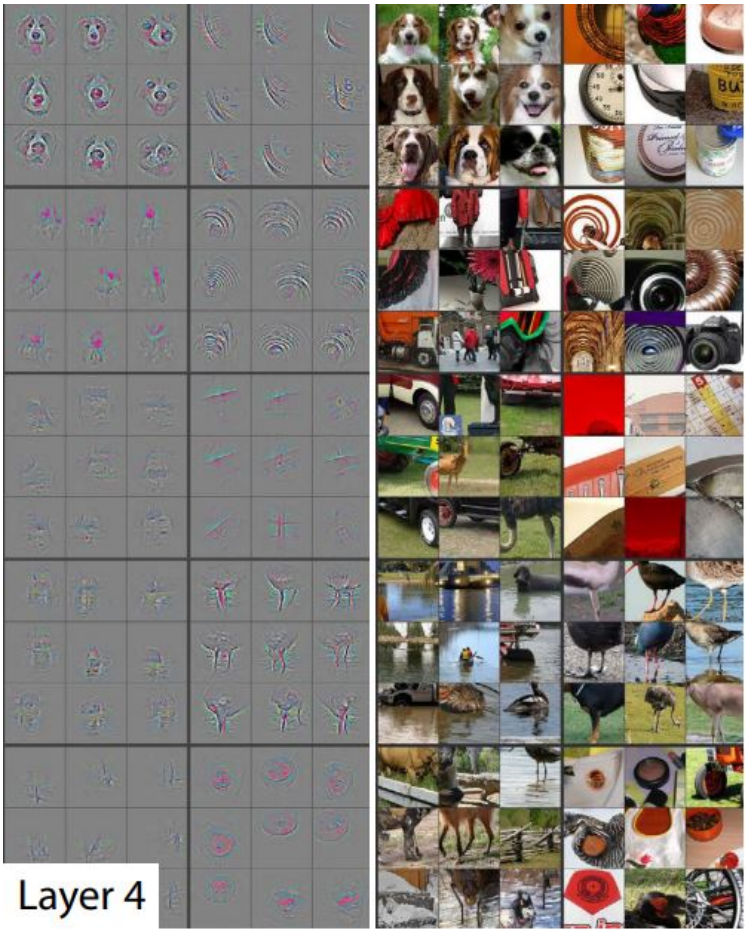
Como se determinar um nível razoável de performance a se esperar de um modelo? Tipicamente, em condições acadêmicas, existem estimativas da taxa de erro que é atingível baseando-se em resultados previamente publicados. Em condições do mundo real, existe uma ideia da taxa de erro necessária para uma aplicação ser considerada segura, possuir bom custo-benefício e apelar para os consumidores. Uma vez determinada uma taxa de erro realística a ser alcançada, as decisões de projeto serão guiadas a fim de atingir essa taxa de erro.

Outra importante consideração além da métrica de performance é a escolha de que métrica utilizar. Várias métricas de performance diferentes podem ser utilizadas para mensurar a efetividade de uma aplicação completa que inclui componentes de aprendizado de máquina. Essas métricas de performance geralmente são diferentes das funções de custo utilizadas para treinar o modelo.

Por exemplo, para tarefas de classificação, classificação com entradas faltantes, e transcrição, é normal fazer uso da métrica de **acurácia** (*accuracy*). Acurácia é apenas a proporção de exemplos classificados corretamente pelo modelo. Considerando um conjunto total de exemplos $\mathbf{x}$, um conjunto total de rótulos $\mathbf{y}$ e um conjunto total de predições $\hat{\mathbf{y}}$ feitas por um classificador binário, se o elemento y_i for a classe de interesse (classe positiva) e o modelo classificou o exemplo x_i corretamente como $\hat{y}_i = y_i$, então é dito que ocorreu um **verdadeiro positivo**. Se o elemento y_i não for a classe de interesse (classe negativa) e o modelo classificou o exemplo x_i corretamente como $\hat{y}_i = y_i$, então é dito que ocorreu um **verdadeiro negativo**. Se o elemento y_i for a classe de interesse (classe positiva) e o modelo classificou o exemplo x_i incorretamente, ou seja, $\hat{y}_i$ pertencendo à classe negativa, então é dito que ocorreu um **falso negativo**. Se o elemento y_i não



(a) (b)



(c)

Figura 2.12: Camadas convolucionais da rede proposta em [4], onde (a) corresponde à primeira camada convolucional da rede, (b) corresponde à segunda camada convolucional da rede e (c) corresponde à quarta camada convolucional da rede. Observa-se que a primeira camada se responsabiliza por reconhecer características de mais baixo nível da imagem como arestas. Conforme se adentra a rede, as características reconhecidas começam a se tornar mais complexas. Na camada 4, já se observa a presença de padrões mais específicas da tarefa em questão [4].

for a classe de interesse (classe negativa) e o modelo classificou o exemplo x_i incorretamente, ou seja, $\hat{y}_i$ pertencendo à classe positiva, então é dito que ocorreu um **falso positivo**. Considerando a quantidade total de verdadeiros positivos como tp, a quantidade total de verdadeiros negativos como tn, a quantidade total de falsos negativos como fn e a quantidade total de falsos positivos como fp, a acurácia pode ser definida pela Equação 2.21 a seguir:

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn}. \tag{2.21}$$

A partir das taxas descritas anteriormente, surge uma métrica útil para entender o desempenho de um modelo de classificação binária, que é a **matriz de confusão**. Caracteriza-se por uma matriz de dimensões 2x2 onde as métricas tp, tn, fp e fn se encontram. A Figura 2.13 mostra a disposição da matriz. A vantagem da matriz de confusão é a sua simples descrição visual geral dos resultados do modelo no problema em questão.

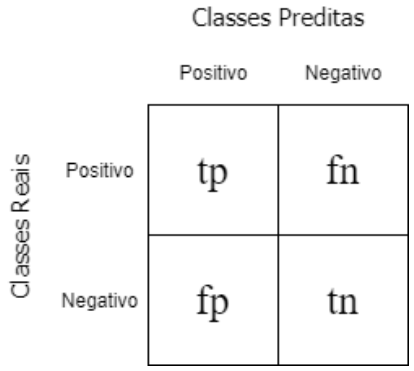


Figura 2.13: Matriz de confusão. A quantidade de verdadeiros positivos encontrados é localizada na diagonal superior esquerda, a quantidade de falsos negativos na diagonal direita superior, a quantidade de falsos positivos na diagonal esquerda inferior e a quantidade de verdadeiros negativos na diagonal direita inferior.

Embora a acurácia seja utilizada em muitas aplicações, ela sozinha pode não compor um bom indicativo de performance do modelo. Frequentemente é mais custoso realizar um tipo de erro do que outro. Por exemplo, um sistema que detecta a presença de câncer pulmonar pode cometer dois tipos de erros: incorretamente classificar uma pessoa doente como saudável ou incorretamente classificar uma pessoa saudável como doente. É mais perigoso classificar incorretamente uma pessoa doente como saudável do que incorretamente classificar uma pessoa saudável como doente. Ao invés de mensurar a acurácia de um modelo detector de câncer, torna-se melhor mensurar alguma outra métrica, onde o custo de julgar um paciente com câncer como saudável seja muito maior do que julgar um paciente saudável como apresentando câncer.

Às vezes, é de interesse treinar um classificador binário que se destina a detectar algum tipo de evento raro. Por exemplo, talvez seja necessário projetar um teste médico para uma doença rara. Supõe-se que apenas uma pessoa a cada um milhão possua essa doença. É possível atingir facilmente 99.9999% de acurácia na tarefa em questão codificando um classificador que simplesmente sempre reporte que a doença está ausente. Claramente, a acurácia é uma maneira ineficaz

de caracterizar a performance de um sistema desses. Uma forma de corrigir esse problema seja utilizando a **precisão** (*Precision*) ou a **sensibilidade** (*Recall*). Precisão (Equação 2.22) é a fração das detecções reportadas pelo modelo que de fato estão corretas, enquanto a sensibilidade (Equação 2.23) é a fração de eventos verdadeiros que foram detectados. Um detector que informa que ninguém apresenta a doença atingiria uma precisão perfeita, porém uma sensibilidade nula. Um detector que informa que todos apresentam a doença atingiria sensibilidade perfeita, porém uma precisão igual à porcentagem de pessoas que apresentam a doença (0.0001% no exemplo dado anteriormente).

$$Precision = \frac{tp}{tp + fp}, \tag{2.22}$$

$$Recall = \frac{tp}{tp + fn}. \tag{2.23}$$

Muitas vezes deseja-se resumir a performance do classificador com um único número. Para isso, precisão e sensibilidade podem ser combinados em uma nova métrica chamada **F-score** ou **F1-score**, dado pela Equação 2.24 abaixo:

$$F = \frac{2 \times Precision \times Recall}{Precision + Recall}. \tag{2.24}$$

Todavia, mesmo a acurácia e o F1-score sendo amplamente utilizadas na estatística, ambas podem fornecer interpretações errôneas, uma vez que não são considerados os tamanhos das quatro classes da matriz de confusão na computação final da métrica [61]. Supondo, por exemplo, que um conjunto de testes desbalanceado de 100 elementos possua 95 elementos da classe positiva e apenas 5 elementos da classe negativa. Supondo um modelo que sempre prevê positivo para qualquer exemplo e que quem o modelou não está ciente deste problema. Desta forma, para o modelo e a tarefa em questão especificados, as categorias da matriz de confusão seriam $tp = 95$, $fp = 5$, $tn = 0$ e $fn = 0$. Esses valores levam a um valor de acurácia de 95% e a um valor de F1-score de 97.44%. Ao se observar esses resultados otimistas, o responsável estaria satisfeito e acreditaria que o modelo de aprendizado de máquina está realizando um ótimo trabalho. Claramente essa não é a verdade.

Para evitar essas ilusões errôneas, existe outra métrica de performance disponível para explorar, o **coeficiente de correlação de Matthews** (MCC) [62], descrito pela Equação 2.25. Por considerar a proporção de cada classe da matriz de confusão na fórmula, seu valor será alto apenas se o classificador se sair bem em ambas as classes positivas e negativas. O valor retornado está no intervalo $[-1, +1]$. Um coeficiente de $+1$ indica uma predição perfeita de ambas as classes, um coeficiente de 0 indica um desempenho equivalente a predições feitas de forma aleatória e um coeficiente de -1 indica uma discordância completa entre as classes reais e as classes preditas. Se qualquer uma das quatro somas no denominador forem zero, o denominador pode arbitrariamente ser definido para o valor 1. Isso irá resultar em um MCC de zero, corrigindo o caso específico.

$$MCC = \frac{tp \times tn - fp \times fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}}. \tag{2.25}$$

Supondo agora que um modelo na mesma tarefa anterior acertou 1 único exemplo da classe negativa e errou 5 exemplos da classe positiva, totalizando $tp = 90$, $fp = 4$, $tn = 1$ e $fn = 5$. Isso produziria uma acurácia de 91% e um F1-score de 95.24%. Porém, ao se calcular o MCC, obtém-se o valor de 0.14, indicando que o algoritmo está desempenhando próximo do aleatório.

Uma maneira útil para avaliar um modelo de forma qualitativa em uma tarefa de classificação binária é utilizar a **Curva de Característica de Operação do Receptor** (Curva ROC). A curva ROC é uma forma de avaliar a habilidade diagnóstica de um classificador binário conforme seu limiar de discriminação é variado. A curva é criada plotando a **taxa de verdadeiros positivos** (TPR) com a **taxa de falsos positivos** (FPR) do modelo em vários limiares de discriminação diferentes. A taxa de verdadeiros positivos nada mais é que a sensibilidade apresentada na Equação 2.23. A taxa de falsos negativos é obtida pela Equação 2.26 e descreve a fração de eventos falsos que não foram detectados. A Figura 2.14 mostra um exemplo da curva ROC.

$$FPR = \frac{fp}{fp + tn}. \tag{2.26}$$

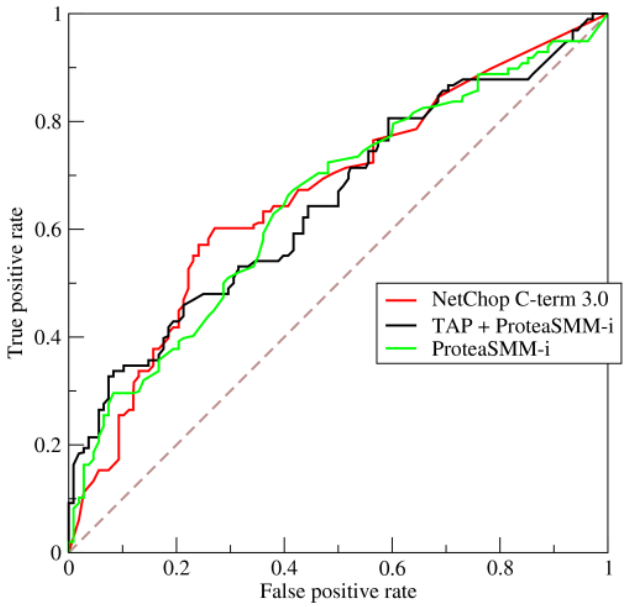


Figura 2.14: Exemplo de curva ROC. Avaliação de três diferentes preditores de epítopos de HIV. A curva tracejada indica uma classificação randômica. Quanto mais para a esquerda a curva do classificador se encontra, melhor é sua capacidade preditiva como um todo.

Uma maneira simples de quantificar a curva ROC em um único número é utilizar a **área sob a curva** ROC ($AUC - ROC$). Como o próprio nome consta, caracteriza-se pela área sob a curva ROC. Um preditor perfeito apresentará uma área sob a curva de valor 1. Um preditor randômico apresentará área sob a curva de valor 0.5. Um preditor que erre todas as classificações apresentará área sob a curva de valor 0.

2.2 *DeepFakes*

2.2.1 Principais Definições sobre *Deepfakes*

O termo *Deepfake* possui muitas definições. Assumindo a definição em [6], *Deepfake* consiste em um vídeo contendo um ou mais rostos trocados cuja troca foi necessariamente realizada por um algoritmo de rede neural. Rostos são de especial interesse para os métodos atuais de manipulação por várias razões: primeiramente, a reconstrução e o rastreamento de faces humanas é um campo bem examinado da visão computacional [63], o que é a fundação para essas abordagens de edição. Em segundo, rostos desempenham um papel central na comunicação humana, uma vez que a feição de uma pessoa pode enfatizar uma mensagem ou até transmitir uma mensagem por si só [64].

Métodos de manipulação facial atuais podem ser divididos em duas categorias: manipulação de expressão facial e manipulação de identidade facial [5]. Uma das técnicas de manipulação de expressão facial mais proeminentes é o método em [65] chamado *Face2Face*. Ele permite a transferência de expressões faciais de uma pessoa para outra em tempo real utilizando apenas hardware comum. Trabalhos como [66] tornam possível animar a face de uma pessoa a partir de uma sequência de áudio como entrada.

Manipulação de identidade facial é a segunda categoria de falsificações faciais. Ao invés de mudar apenas a expressão, esses métodos trocam o rosto de uma pessoa com o rosto de outra pessoa. Essa categoria é chamada de troca de rostos. Tornou-se popular com aplicativos a nível de consumidor amplamente difundidos, como Snapchat. *Deepfakes* também realizam troca de rostos, porém com aprendizado profundo. Enquanto trocas de rostos baseadas em técnicas gráficas computacionais simples podem rodar em tempo real, *Deepfakes* precisam ser treinados para cada par de vídeos, o que é uma tarefa que demanda bastante tempo (de horas até semanas) [5, 15].

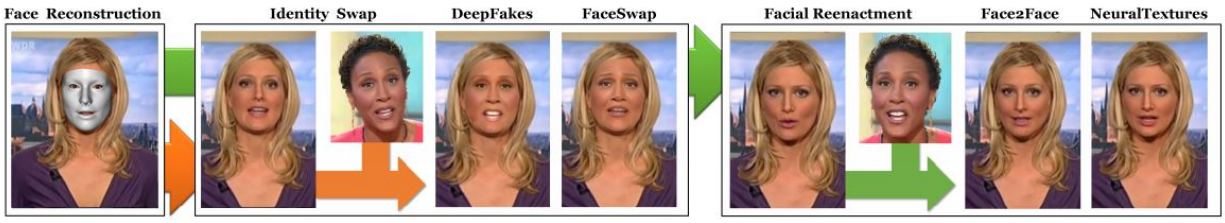


Figura 2.15: Exemplos de métodos de manipulação facial. À esquerda tem-se os métodos de manipulação de identidade facial (grupo o qual o *Deepfake* está incluso) e à métodos de manipulação de expressão facial [5].

Produzir um *Deepfake* não requer hardware especializado além de uma GPU de nível comercial simples, e vários pacotes de software para criação de *Deepfakes* estão disponíveis para o público. A combinação desses fatores fez com que a popularidade da técnica explodisse, tanto em termos de produção de vídeos de paródias para entretenimento quanto no uso de ataques direcionados contra indivíduos ou instituições [67].

A produção de *Deepfakes* se dá principalmente por uma de duas arquiteturas base: uma **Rede Adversária Geradora** (*Generative Neural Network* ou *GAN*) [68] ou um **Autoencoder** [69].

Uma GAN consiste em duas redes batalhando entre si, onde uma é a geradora e a outra a discriminatória. O objetivo da rede geradora é produzir imagens realistas o suficiente para enganar a rede discriminatória, e o objetivo da rede discriminatória é corretamente identificar imagens forjadas pela rede geradora. O processo de treinamento se dá dessa forma, com as duas redes treinando simultaneamente. O resultado final é muitas vezes uma rede com excelente capacidade de geração de imagens realistas, como exemplo a Figura 2.16.

Figura 2.16: A imagem de uma jovem gerada pela StyleGAN, uma rede adversária generativa (GAN). A pessoa nesta foto não existe, mas é gerada por uma inteligência artificial baseada na análise de retratos.

Um autoencoder constitui uma arquitetura de rede neural bem simples, utilizada com o objetivo de aprender uma representação de um conjunto de dados. Ela é constituída essencialmente por duas partes: um **encoder** e um **decoder** (Figura 2.17). O encoder é responsável por reduzir a dimensionalidade da entrada, enquanto o decoder é capaz de realizar a reconstrução dessa entrada.

2.2.2 Principais Trabalhos Relacionados sobre *Deepfakes*

Ao mesmo passo que as técnicas de manipulação facial cresceram em qualidade na criação de *Deepfakes*, as técnicas de detecção dessas manipulações cresceram igualmente. Várias técnicas mais simples se mostraram efetivas na tarefa de detecção, sem realizar abordagens utilizando aprendizado profundo.

As técnicas abordadas por MATERN et al. [70] mostram eficiência na tarefa de detecção de manipulação facial utilizando detecção de artefatos visuais simples como coloração inconsistente entre os olhos, falta de reflexões luminosas e geometria faltante nos rostos gerados. No estudo de YANG et al. [71], é demonstrado que um *Deepfake* muitas vezes apresenta inconsistência na pose de cabeça gerada e isso pode ser explorado na detecção. A abordagem realizada por AGARWAL et al. [72] faz uso de uma técnica forense que modela expressões faciais e movimentos que caracterizam o padrão de fala de um indivíduo e demonstra que *Deepfakes* violam essas correlações.

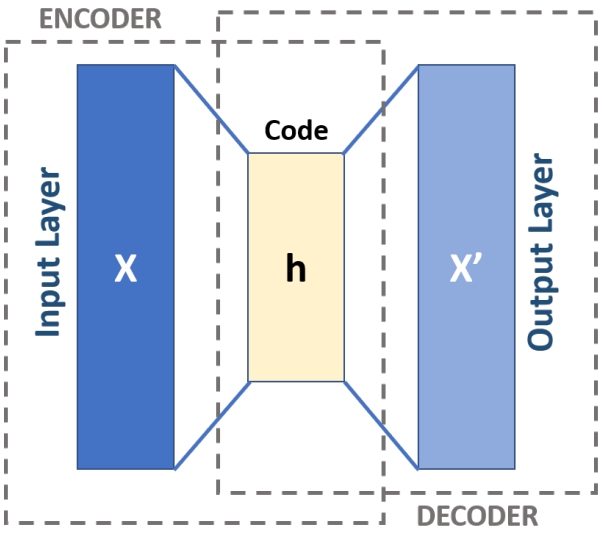


Figura 2.17: Arquitetura básica de um Autoencoder.

Técnicas utilizando aprendizado profundo se mostram o estado da arte atualmente na área de detecção de *Deepfakes*. A técnica utilizada por AFCHAR et al. [73] explora o uso de duas redes convolucionais simples para detecção das manipulações faciais tanto de *Deepfakes* quanto da técnica de *Face2Face*, obtendo ótimos resultados nos testes. No estudo de SABIR et al. [74], uma rede neural convolucional é combinada com uma estrutura recorrente capaz de tirar proveito do fator temporal dos vídeos para aumentar a eficácia de detecção dos *Deepfakes*. Utilizando o mecanismo de **atenção** [75] em redes neurais convolucionais, a técnica de DANG et al. [76] demonstra melhora na detecção de rostos forjados.

Uma importante consideração deve ser dada aos conjuntos de dados disponíveis para a tarefa de detecção de *Deepfakes*, uma vez que grande parte das técnicas que utilizam aprendizado profundo a fazem utilizando técnicas de aprendizado supervisionado, necessitando de exemplos para o treinamento dos modelos. Os conjuntos de dados públicos aumentaram significativamente de tamanho nos últimos anos. A Figura 2.18 mostra um gráfico comparativo do tamanho de vários conjuntos de dados no âmbito de *Deepfakes*.

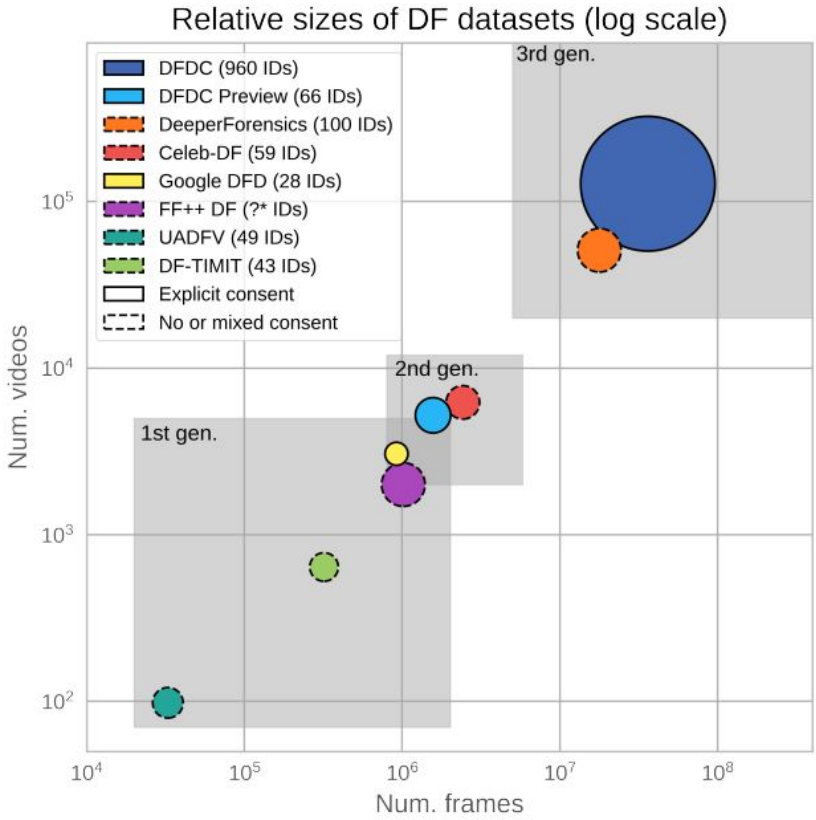


Figura 2.18: Comparação de conjuntos de dados de *Deepfakes* [6]. Ambos os eixos estão em escala logarítmica. Os tamanhos dos conjuntos de dados são separados por gerações. Na primeira geração tem-se conjuntos de dados como o *DF-TIMIT* [7]. Na segunda geração são encontrados conjuntos de dados mais famosos como o *CELEB-DF* [8] e o *FF++ DF* [5]. Na terceira geração encontram-se os maiores conjuntos de dados disponíveis atualmente como o *Deep Fake Detection Challenge* [6] com aproximadamente 100 mil vídeos.

Capítulo 3

Metodologia

Neste capítulo, inicialmente aborda-se a escolha da base de dados (Seção 3.1). É discutida sua estruturação e quais elementos a compõe, bem como as modificações necessárias nela para que a metodologia de treinamento do modelo seja possível (Seção 3.1.1). A definição de toda a estratégia para a aprendizagem profunda é definida na Seção 3.2. Nela, é discutida a estratégia de treinamento adotada (Seção 3.2.1), onde as ferramentas envolvidas no processo são especificadas e quais motivações as levaram a serem escolhidas. Também é relatado um fluxo básico desde o pré-processamento do conjunto de dados necessário para tornar a estratégia de treinamento viável até a validação da qualidade do modelo utilizando métricas de validação e performance. O modelo é descrito em melhor detalhes na Seção 3.2.2, além de serem explícitas as motivações que levaram à sua escolha. É descrita uma importante etapa de ajustes do modelo escolhido na Seção 3.2.3. Tem-se então a descrição da avaliação final dos resultados na Seção 3.3, onde os métodos expostos por fim na Seção 3.3.1 são utilizados. O fluxograma da Figura 3.1 destaca de forma visual a organização das seções.

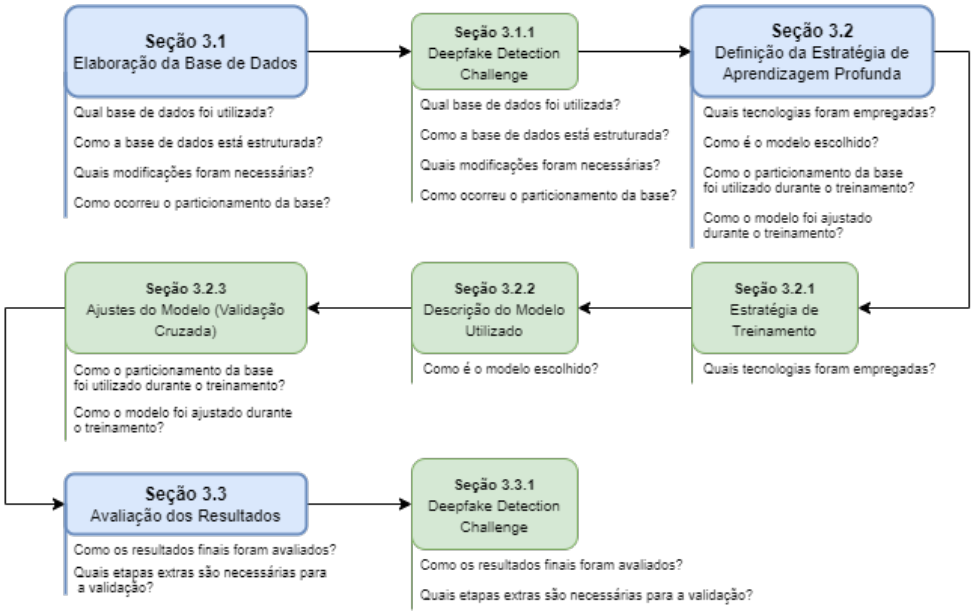


Figura 3.1: Fluxograma de informações das seções presentes no capítulo de metodologia.

3.1 Elaboração da Base de Dados

Como o trabalho consiste na utilização de modelos de aprendizagem profunda para a detecção de *Deepfakes*, a abordagem de aprendizado supervisionado foi escolhida. Para tal feito, se faz necessária a utilização de um amplo conjunto de dados para o treinamento do modelo.

3.1.1 *Deepfake Detection Challenge*

O *Deepfake Detection Challenge* [6] ou apenas DFDC foi a base de dados escolhida para o desenvolvimento do trabalho em questão. Disponibilizada em dezembro de 2019, consiste atualmente no maior conjunto de dados público de vídeos com rostos trocados, contando com aproximadamente 100 mil vídeos originados de 3426 atores pagos. As trocas foram realizadas utilizando diversas técnicas como GANs (Seção 2.2) e métodos de não aprendizado. O conjunto de dados foi feito público através de uma competição no site *Kaggle* [77], que se encerrou oficialmente no dia 23 de abril de 2020. O conjunto de dados completo ocupa aproximadamente $470GB$ de espaço físico em memória.

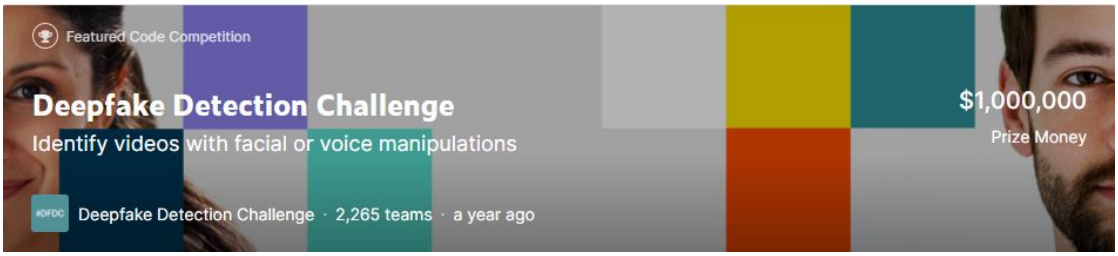


Figura 3.2: Bandeira do *Deepfake Detection Challenge* disponibilizado no site original do desafio na plataforma online *Kaggle*.

O conjunto de dados completo foi obtido através do *Kaggle*, onde 50 arquivos arquivos compactados no formato zip foram obtidos. Cada arquivo apresenta um dos diretórios do conjunto de dados. Desta forma, o conjunto completo apresenta 50 diretórios. Os nomes dos diretórios

(a) (b)

Figura 3.3: Exemplos do conjunto de dados *Deepfake Detection Challenge* onde (a) tem-se o vídeo com o rosto original e (b) o vídeo *Deepfake* equivalente.

estão no formato "*dfdc_train_part_**N***" onde ***N*** é um inteiro que vai de 0 a 49. Cada diretório contém, em média, 2000 vídeos, com aproximadamente 80% dos vídeos dentro de cada diretório pertencendo à classe *FAKE*, ou seja, aproximadamente 80% dos vídeos dentro de cada diretório são um *Deepfake*. As classes de cada vídeo, isto é, a classificação do vídeo como real ou falso, estão agregadas, por diretório, em um arquivo no formato *JSON* de nome "*metadata.json*". A Figura 3.4 abaixo mostra a de estruturação do arquivo para o diretório *dfdc_train_part_0*. Os vídeos estão no formato *.mp*4 e possuem seus nomes como uma combinação única de letras. Todos possuem 29.97 como taxa de quadros por segundo.

	label	split	original
owxbbpjpch.mp4	FAKE	train	wynotylpnm.mp4
vpmyeepbep.mp4	REAL	train	NaN
fzvpbrzssi.mp4	REAL	train	NaN
htorvhbcae.mp4	FAKE	train	wclvkepakb.mp4
fckxaqjbxk.mp4	FAKE	train	vpmyeepbep.mp4
...	...	...	...
hectzcuzdv.mp4	FAKE	train	vtunvalyji.mp4
yxkbjxmtzr.mp4	FAKE	train	sttnfyptum.mp4
wkdnagybtb.mp4	FAKE	train	jytrvwlewz.mp4
fonrexmbzz.mp4	FAKE	train	fufcmupzen.mp4
etychryvty.mp4	FAKE	train	uqtqhiqymz.mp4

1334 rows × 3 columns

Figura 3.4: Exemplo de estruturação do arquivo "*metadata.json*" para o diretório "*dfdc_train_part_0*". O nome do vídeo dentro do diretório é dado pelo índice da linha. A coluna **label** indica a classe do vídeo (*REAL* ou *FAKE*). A coluna **split** é sempre marcada como *train*. A coluna **original** indica, caso o vídeo seja um *deepfake*, o vídeo original do qual o *deepfake* foi gerado.

Para a utilização do modelo escolhido (discutido na Seção 3.2.2), é necessária a utilização de imagens, logo o conjunto de dados original da forma apresentada não pode ser utilizado para treinar o modelo.

Com esse fator em mente, optou-se pela extração da face do indivíduo dos vídeos. Para isso, utilizou-se uma arquitetura (Seção 2.1.2.13) em redes neurais convolucionais chamada **Rede convolucional multitarefa em cascata** (*Multi-task cascaded convolutional network* ou *MTCNN*) [9]. A Figura 3.5 apresenta o funcionamento da arquitetura para detecção de rostos. O principal objetivo da arquitetura é detectar a presença de um rosto humano em uma imagem e, com isso, delimitar suas fronteiras (caixa delimitadora) de forma que o rosto fique em destaque do restante da imagem. A rede também é capaz de extrair os marcadores faciais (posição dos olhos, nariz e boca) caso seja necessário, todavia, essa funcionalidade não foi utilizada.

Com um método de extração de rostos definido, tornou-se necessário definir a partição de teste e treinamento das faces extraídas, seguindo as boas práticas citadas na Seção 2.1.2.5. Nesta etapa,

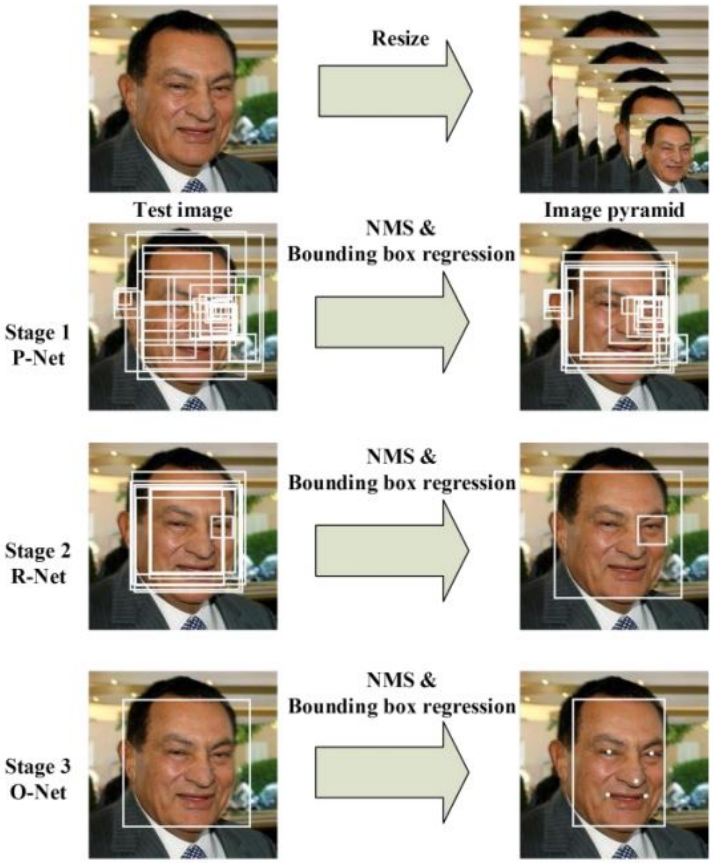


Figura 3.5: Pipeline da estrutura em cascata que inclui os três estágios das redes convolucionais multitarefas em cascata. Em primeiro lugar, as janelas candidatas são produzidas através de uma rede de proposta rápida (P-Net). Depois disso, esses candidatos são refinados na próxima etapa por meio de uma rede de refinamento (R-Net). Na terceira fase, a rede de saída (O-Net) produz a caixa delimitadora final e a posição dos marcadores faciais [9].

o arquivo "*metadata.json*" foi alterado individualmente para todas os diretórios do conjunto de dados. Para cada "*metadata.json*", definiu-se que 80% dos vídeos categorizados como falsos e 80% dos vídeos categorizados como verdadeiros seriam utilizados para o treinamento, e 20% dos vídeos categorizados como falsos e 20% dos vídeos categorizados como verdadeiros seriam utilizados para o treinamento. A partição foi feita de maneira aleatória. Esse processo foi realizado para todos os arquivos "*metadata.json*" em todos os diretórios "*dfdc_train_part_**N***".

O caminho dos arquivos extraídos foi definido então como "***root**/Kaggle Faces Dataset/**split**/**label**/**folder nome índice** .jpg*", onde:

- **root**: Diretório raíz do projeto.
- **split**: A partição do vídeo para treinamento ou teste. Neste caso, *train* ou *test*.
- **label**: A classe do vídeo em questão. Neste caso, *FAKE* ou *REAL*.
- **folder**: O diretório o qual o vídeo original pertencia.

- **nome**: O nome do vídeo.
- **índice**: O índice do rosto atual extraído dado como um inteiro positivo incremental.

A Figura 3.6 mostra o processo de extração da face e salvamento do arquivo para um vídeo de nome *ajeegjgyzk.mp4* em um diretório *dfdc_train_part_0*. Com uma estruturação dessas torna-se possível saber exatamente de qual vídeo e de qual diretório uma imagem se originou.

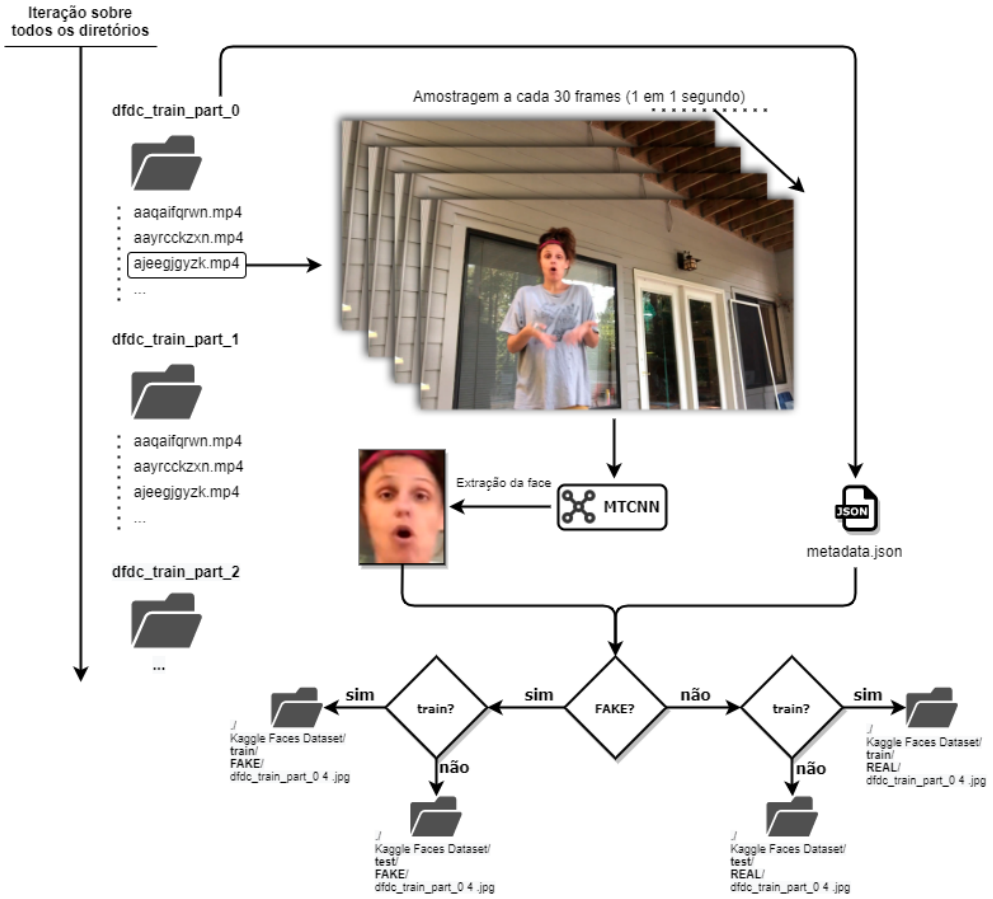


Figura 3.6: Fluxograma do processo de extração das faces dos vídeos utilizando a MTCNN e posteriormente salvamento do arquivo.

Uma importante decisão de projeto foi a definição da frequência de amostragem. Uma amostragem a cada um quadro do vídeo, considerando todos os vídeos com aproximadamente 10 segundos de duração e um rosto detectável em todos eles, produziria um conjunto de dados com aproximadamente 30 milhões de imagens. Uma quantidade de imagens dessa proporção acarretaria em uma elevada carga computacional, carga essa que não se justifica com o hardware disponível no momento da execução deste trabalho. Portanto, o valor de uma amostragem a cada 30 quadros (1 em 1 segundo) foi a abordagem escolhida.

3.2 Definição da Estratégia de Aprendizagem Profunda

Nesta seção serão apresentados os critérios e estratégias, estes usados para definição do modelo de aprendizagem profunda para detectar *Deepfakes* em um vídeo.

3.2.1 Estratégia de Treinamento

O treinamento foi realizado fazendo uso de diversas ferramentas presentes na linguagem de programação *Python*. Uma dessas ferramentas é o *framework* de desenvolvimento **Pytorch** [24] para aprendizado de máquina. O Pytorch é uma poderosa e otimizada biblioteca de tensores para aprendizado de máquina profundo que faz uso de GPUs para acelerar suas computações.

Existem diversas bibliotecas construídas utilizando o Pytorch como base, uma vez que se caracteriza por um *framework* mais baixo nível no contexto de aprendizado profundo. Uma dessas bibliotecas é a **Fastai** [78]. Caracteriza-se por uma biblioteca que fornece componentes de alto nível de forma rápida e fácil, tornando possível obter resultados de última geração no domínio do aprendizado profundo. Ao mesmo tempo, ela fornece componentes de baixo nível que podem ser misturados e combinados para construir novas abordagens. Desta forma, ela visa realizar ambas as coisas sem comprometer substancialmente a facilidade de seu uso, sua flexibilidade ou seu desempenho. Isso é possível graças a uma arquitetura cuidadosamente construída em camadas que expressa padrões fundamentais comuns de muitas técnicas de aprendizagem profunda e processamento de dados em termos de abstrações desacopladas. Essas abstrações podem ser expressa de forma concisa e clara, aproveitando o dinamismo da linguagem Python subjacente e a flexibilidade da biblioteca PyTorch. A biblioteca já é amplamente utilizada em pesquisas, na indústria e no ensino.

A biblioteca Fastai foi então a escolhida para a implementação do treinamento. Primeiro, separou-se de forma aleatória sem repetição o conjunto de imagens do diretório */train* de forma que uma parcela fosse utilizada para o treinamento e outra parcela para a validação do modelo. As imagens de cada iteração eram então carregadas em memória, onde eram convertidas em tensores, normalizadas para que seus valores ficassem no intervalo $[0, 1]$ e, posteriormente, concatenadas para realizar cada passagem pelo modelo. A função de custo foi balanceada de forma a penalizar o modelo por errar exemplos mais raros do conjunto de dados. A cada época de treinamento completa realizada sobre o conjunto de dados, as métricas de entropia cruzada (Equação 2.5) foram obtidas tanto para o treinamento quanto para as imagens separadas para validação. As métricas da área sob a curva *ROC* e do *MCC* também foram geradas porém apenas para o conjunto de validação. O modelo foi treinado de forma que a métrica *MCC* fosse minimizada tanto quanto possível.

Para o treinamento, foi utilizada uma política específica de treinamento introduzida inicialmente em [35] e estendida em [79]. Consiste na política da taxa de aprendizado cíclica (Seção 2.1.2.7), mais especificamente na política da taxa de aprendizado de um ciclo. Ela consiste em utilizar uma taxa de aprendizado base e uma taxa de aprendizado máxima e, durante todo o treinamento, variar a taxa de aprendizado do modelo em 2 etapas de mesmo comprimento: a taxa

de aprendizado se inicia em um valor mínimo, aumenta até um valor máximo e retorna a outro valor mínimo, tudo em um único ciclo e utilizando alguma função para o processo de aumento e diminuição, como uma simples função linear ou uma função cosseno. O autor demonstra que essa política funciona bem com valores elevados para o tamanho de lote e também para a taxa de aprendizado, conseguindo realizar o treinamento de modelos uma ordem de magnitude mais rápido que com métodos clássicos, um fenômeno que ele apelidou de super-convergência.

A taxa de aprendizado máxima do modelo, considerando a política de taxa de aprendizado de um ciclo, foi encontrada utilizando o **teste de faixa da taxa de aprendizado** [35]. Ela consiste em rodar o modelo no conjunto de dados por várias épocas, aumento o valor da taxa de aprendizado linearmente entre um valor mínimo e um valor máximo. Em seguida, é plotada a acurácia ou o custo do modelo versus a taxa de aprendizado. Tipicamente se escolhe um valor de taxa de aprendizado que esteja em uma região de elevado aumento de acurácia ou elevado declive de custo.

Todo o processo de treinamento do modelo foi realizado em **meia precisão** (ou precisão em ponto flutuante 16 bits). Treinamento em meia precisão é uma técnica utilizada para reduzir o uso de memória de uma GPU e o uso de memória RAM durante o treinamento de um modelo de aprendizado profundo. Geralmente uma rede neural realiza todas as suas computações em precisão única (ou precisão em ponto flutuante 32 bits). Utilizar apenas meia precisão, por definição, acaba por ocupar metade da memória RAM , o que permite na teoria dobrar o tamanho do modelo e dobrar o tamanho do lote de imagens, permitindo um treinamento mais rápido.

Como foi utilizada a estratégia de transferência de aprendizado (Seção 2.1.2.14), a abordagem padrão de se substituir a última camada do modelo classificador (discutido melhor na Seção 3.2.2), "congelar" as camadas anteriores da rede (tornar seus parâmetros "congelados", isto é, impedir que sejam atualizados pelo algoritmo de gradiente descendente) e treinar essa última camada por uma quantidade fixa de épocas foi escolhida. Após isso, "descongelar" todas as camadas da rede e continuar o treinamento por mais uma quantidade fixa de épocas.

3.2.2 Descrição do Modelo Utilizado

A abordagem de aprendizado supervisionado utilizando aprendizado profundo leva à escolha de um modelo de visão computacional. Como discutido na Seção 2.1.2.11, redes convolucionais se tornaram estado da arte em diversas tarefas de visão computacional, isso inclui classificação de imagens. Com isso em mente, o trabalho focou no treinamento da arquitetura **ResNet18** para a classificação das imagens obtidas do conjunto de dados do *Deep Fake Detection Challenge*. As vantagens da arquitetura da ResNet18 em relação à redes convolucionais tradicionais se justificam na Seção 2.1.2.13. A Figura 3.7 apresenta as arquiteturas originais de diversas versões das ResNets. O *stride* mencionado refere-se à passada do *kernel* de convolução, ou seja, o *kernel* irá se mover pela imagem pulando uma quantidade equivalente ao *stride* de cada vez.

A arquitetura original apresenta uma última camada composta por um perceptron multicamadas (Seção 2.1.2.1) cujas saídas, após aplicada a função *softmax*, resultam nas probabilidades das

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				

Figura 3.7: Arquiteturas para o conjunto de dados ImageNet. Os blocos de construção são mostrados entre colchetes (ver também Figura 3.8), com os números de blocos empilhados. A redução da resolução é realizada por conv3_1, conv4_1 e conv5_1 com um *stride* de 2 [3].

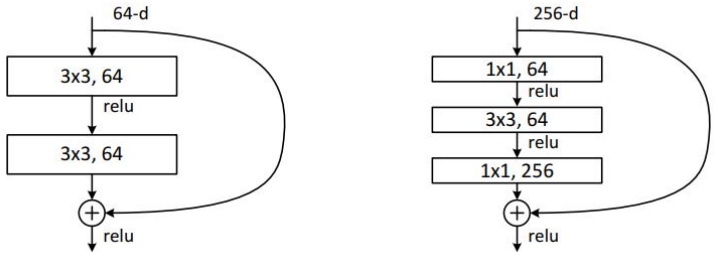


Figura 3.8: Exemplos de blocos residuais utilizados pelas arquiteturas ResNets para a *ImageNet* [3].

1000 classes do conjunto de dados *ImageNet*. Todavia, não é possível utilizá-la nessa estrutura original, uma vez que o problema deste trabalho consiste na classificação de apenas duas classes (*FAKE* e *REAL*).

Uma vez que o conjunto de dados original tenha passado pelo pré-processamento descrito na Seção 3.1, pode-se utilizar a estratégia discutida na Seção 2.1.2.14, onde se remove a última camada que corresponde à camada classificadora e a substitui por outra camada perceptron multicamadas, abordagem essa conhecida como *transfer learning*, permitindo utilizar boa parte das informações já adquiridas pela rede no treinamento anterior. Isso garante que, após a aplicação da função *softmax*, as probabilidades de cada uma das classes estarão disponíveis após uma imagem ser fornecida como entrada e a rede ter realizado as computações necessárias. Essa nova camada é inicializada com parâmetros aleatórios, obtidos de uma distribuição normal com média 0 e desvio padrão 1, sendo treinada pelo algoritmo de gradiente descendente (Seção 2.1.2.4) e função de custo entropia cruzada (Seção 2.1.2.3).

3.2.3 Ajustes do Modelo (Validação Cruzada)

Durante a Seção 2.1.2.5, foi discutida a necessidade de se separar o conjunto de dados em um conjunto de treinamento e um conjunto de teste. Porém, não existe uma forma de garantir que o conjunto de validação seja representativo do conjunto de dados original. Como a divisão é feita, na maioria das vezes, de forma aleatória, é possível que a parcela de validação esteja composta na maioria por imagens relativamente mais fáceis ou mais difíceis de serem preditas. O primeiro caso, após o treinamento do modelo, pode levar à crença de que o modelo está generalizando muito bem para novas imagens. O segundo caso, de forma semelhante, pode levar à crença de que o modelo é insuficiente para a tarefa em questão. Uma forma eficiente de atestar a qualidade e a capacidade de generalização de um modelo de aprendizado de máquina, sem as desvantagens mencionadas anteriormente, é a utilização de uma técnica conhecida como **validação cruzada**.

A validação cruzada é uma técnica que permite o modelo experienciar todo o conjunto de dados fazendo uso de diferentes partições para treinamento e validação. O conjunto de dados completo é separado em N partições (também chamadas de *folds*), com cada partição dada por $P_1, P_2, ..., P_N$. Inicialmente, as partições P_2 até P_N são utilizadas para treinamento e a partição P_1 é reservada para validação. Após o treinamento e a validação, o modelo é zerado e se inicia um novo processo de treinamento, onde agora as partições $P_1, P_3, ..., P_N$ são utilizadas para treinamento e a partição P_2 é reservada para validação. Esse processo se repete até que todas as N partições tenham sido utilizadas para validação uma única vez. As métricas finais de validação de cada uma das N partições são utilizadas para obter uma média e um desvio padrão dos valores. Estes são capazes de fornecer mais informação a respeito da qualidade geral do modelo para a tarefa e conjunto de dados em questão. A Figura 3.9 exemplifica esse processo.

A técnica de validação cruzada não é uma técnica para seleção de modelos e sim uma técnica para validação qualitativa de modelos ou de instâncias de modelos, isto é, verificar se algum modelo específico com hiperparâmetros específicos é adequado para uma determinada tarefa. A validação cruzada é útil na exploração de diferentes hiperparâmetros (Seção 2.1.2.12) a fim de se obter a melhor combinação que minimize o erro do modelo no conjunto de dados em questão.

Por apresentar tamanhas vantagens citadas, a técnica de validação cruzada foi escolhida para este trabalho. O conjunto completo dentro do diretório *train* foi particionado de forma que cada partição mantivesse a mesma proporção de imagens das duas classes que o conjunto de dados original possui. Quando a proporção de classes em todas as partições da validação cruzada é mantida, a técnica passa a ser uma **validação cruzada estratificada**. Bibliotecas em *Python* como *Scikit Learn* [2] provêm funções para a realização desta técnica de maneira simples. No caso, a própria biblioteca foi a escolhida para realizar as partições.

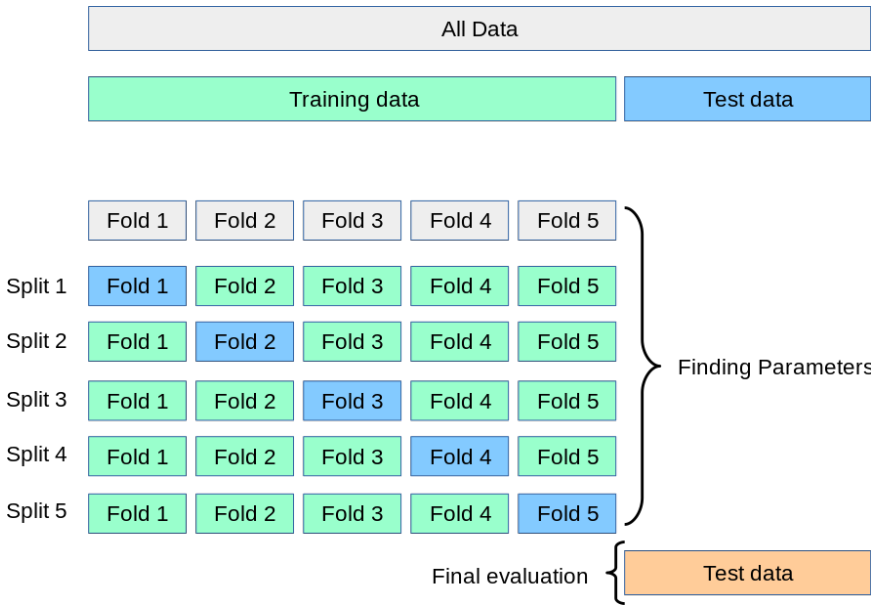


Figura 3.9: Processo de realização da validação cruzada. O conjunto de dados de treinamento é separado em N partições, o modelo é treinado em $N-1$ partições e validado na partição restante. O processo se repete para todas as possíveis combinações de $N-1$ partições para treinamento. Pode-se optar inicialmente por separar um conjunto de testes do conjunto total de dados antes de iniciar o processo de validação cruzada. Nesse caso, a validação cruzada é feita apenas para a parte separada para treinamento do conjunto total. Após finalizada a validação cruzada, o modelo é treinado em todo o conjunto de dados e validado uma última vez no conjunto de testes separado inicialmente [2].

3.3 Avaliação dos Resultados

Os resultados são avaliados para todos as partições da validação cruzada estratificada. Para cada uma delas, as métricas de validação dos resultados (Seção 3.3.1) são geradas e as médias e os desvios padrões são calculados, de forma que se obtenha a capacidade preditiva do modelo. Em seguida, o classificador é treinado em todas as imagens do diretório de treinamento *train* e as métricas finais para o modelo são obtidas para o conjunto de teste do diretório *test*.

Uma etapa adicional final se faz necessária para concluir a qualidade do modelo na detecção de *deepfakes* em vídeos: as métricas obtidas para o conjunto de teste refletem apenas a qualidade de classificação de imagens de rostos individuais. Todavia, um vídeo é composto por múltiplas imagens seguidas. Desta forma, para classificar um vídeo como *FAKE* ou *REAL*, é necessário utilizar todas as imagens do vídeo em questão e só então obter uma predição para ele. Com isso em mente, as imagens no conjunto de teste foram segmentadas, separadas por título de vídeo. As imagens agrupadas por título passam pelo classificador uma a uma, gerando as predições de cada imagem. As predições de cada classe são então contabilizadas. A quantidade de imagens falsas é comparada com uma fração ρ da quantidade de imagens verdadeiras e a atribuição é feita então para a classe com o maior resultado. ρ é um coeficiente para a taxa de liberdade desejada. $\rho < 1.0$ garante mais conservacionismo e $\rho > 1.0$ garante mais liberdade. $\rho = 1.0$ garante o mesmo peso

para ambas as quantidades. A Figura 3.10 mostra esse processo.

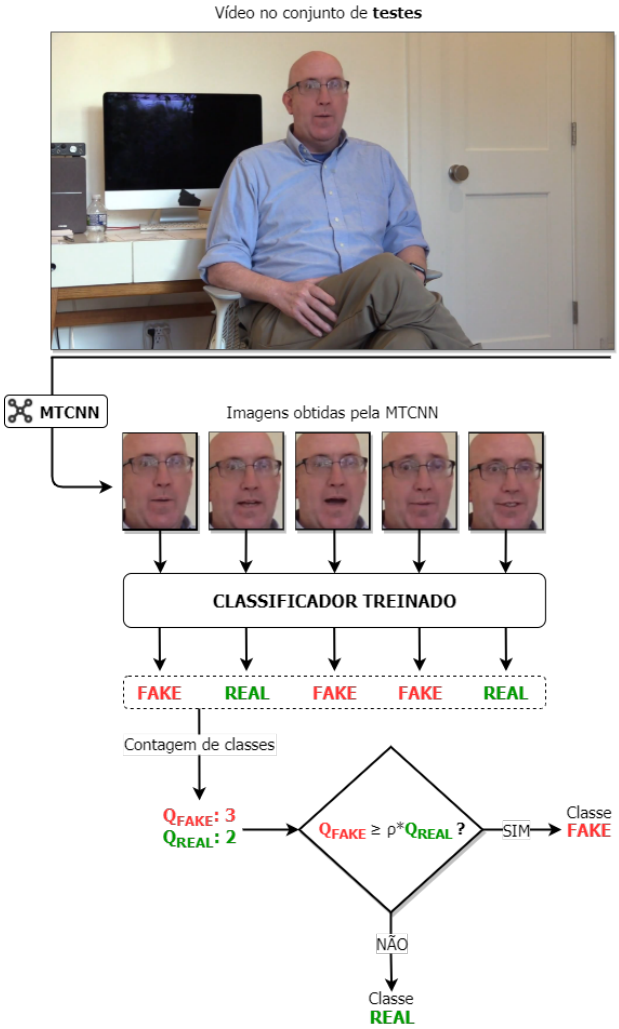


Figura 3.10: Processo de comparação final realizado para os vídeos do conjunto de testes do diretório *test*. As imagens obtidas pela MTCNN (etapa anterior à todo o treinamento) passam pelo classificador treinado, gerando as predições. As predições são contabilizadas e a classe que estiver mais relativamente presente é então dada como a classe a qual o vídeo pertence.

3.3.1 Métodos de Validação dos Resultados

Por se tratar de uma classificação binária (*FAKE* ou *REAL*), existem várias possíveis métricas de performance disponíveis para serem utilizadas. É importante ressaltar que, uma vez que *deepfakes* representam um grande perigo, como discutido na Seção 2.2, detectá-los sempre que possível se torna vital. Isso significa, em métricas, que a sensibilidade do modelo (Seção 2.1.2.15) para novas imagens deve ser maximizada, mesmo às custas de um aumento na taxa de falsos positivos. É menos custoso julgar um rosto *REAL* como *FAKE* do que julgar um rosto *FAKE* como *REAL*.

Para o trabalho, foram utilizadas três métricas principais:

- **Entropia Cruzada** (Equação 2.5): Fornece uma média a respeito do quão longe as pro-

babilidades computadas pelo classificador estão das classes reais. Importante ressaltar que esse valor, idealmente, deve ser o menor possível, porém pode ocorrer de ele aumentar e as métricas seguintes continuarem a melhorar.

- **Acurácia** (Equação 2.21): Útil para se ter uma ideia geral dos resultados do classificador. Como discutido na Seção 2.1.2.15, não é interessante de ser utilizada sozinha em algunas casos.
- **Coeficiente de Correlação de Matthews** (Equação 2.25): Provê uma informação melhor quanto aos erros e acertos do classificador no evento de um conjunto de dados desbalanceado.
- **Matriz de Confusão (Figura 2.13)**: Oferece uma excelente maneira de atestar visualmente as taxas de acerto do modelo classificador.
- **Curva ROC**: Métrica que permite uma percepção visual da qualidades das predições do classificador em relação ao nível de certeza das predições.
- **Área sob a curva ROC**: Métrica útil para saber numericamente a qualidade do classificador em relação ao nível de certeza das predições.

Capítulo 4

Resultados

4.1 Hardware de Treinamento

Todo o treinamento do modelo Resnet18 com o conjunto de dados descrito na Seção 3.1.1 foi realizado utilizando um computador pessoal. As configurações da máquina em questão podem ser conferidas na tabela 4.1 abaixo.

	Modelo
Sistema Operacional	Microsoft Windows 10 Home
Placa de Vídeo (GPU)	NVIDIA GeForce RTX 2070 8GB GDDR6
Memória RAM	2 x HyperX Fury 8GB 2666MHz DDR4
Processador	Intel Core i7-9700K Coffee Lake 3.6GHz
Unidade de Armazenamento	3 x SSD Kingston A400, 480GB SATA

Tabela 4.1: Configurações da máquina *host* utilizada para o treinamento do modelo de aprendizado profundo.

Destaca-se como informação vital para o treinamento a memória da GPU disponível (8GB).

4.2 Base de Dados de Imagens

4.2.1 Quantidades e Dimensões

Como discutido na Seção 3.1.1, foi utilizada a rede MTCNN para realizar a extração de todos os rostos identificados em todos os vídeos do conjunto de dados DFDC. A amostragem definida para a inferência da rede foi de 30 em 30 frames (aproximadamente de 1 em 1 segundo) para cada vídeo em cada um dos 50 diretórios. A tabela 4.2 apresenta a quantidade de imagens de rostos e cada classe obtidas para cada uma das partições *train* e *test*.

Observa-se a presença superior de imagens pertencentes à classe *FAKE*, caracterizando um conjunto de dados desbalanceado. Aproximadamente 82% de todas as imagens do conjunto de

	train		test	
Classe	**FAKE**	**REAL**	**FAKE**	**REAL**
Quantidade de Imagens	787.990	164.950	174.662	36.601
Porcentagem do Total de Imagens	67,7%	14,2%	15,0%	3,1%

Tabela 4.2: Quantidade de imagens geradas por diretório por classe após o pré-processamento utilizando a MTCNN.

dados gerado pertencem à classe *FAKE* e apenas 18% pertencem à classe real. A Figura 4.1 apresenta alguns exemplos dos rostos extraídos pela MTCNN e suas respectivas classes.

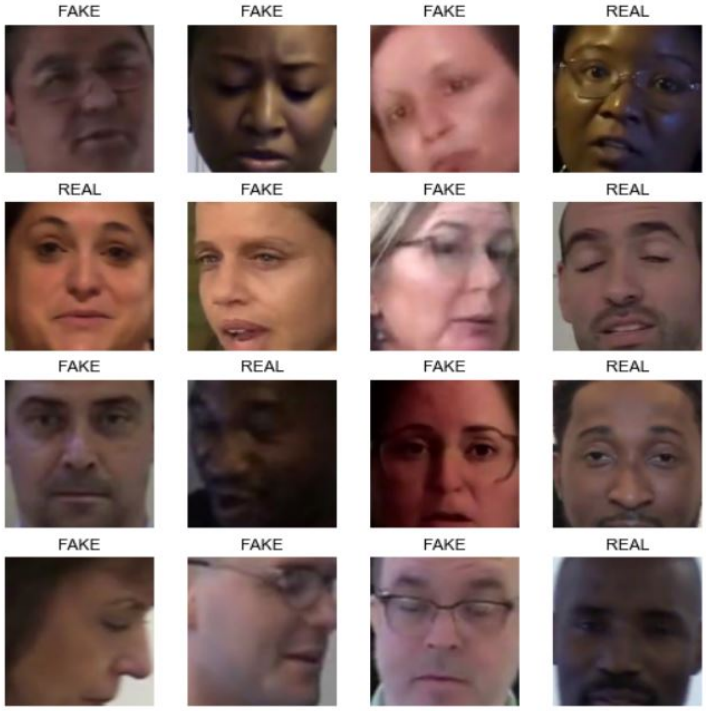


Figura 4.1: Exemplos de algumas das imagens extraídas pela MTCNN e suas respectivas classes.

Pode-se obter uma estimativa da distribuição dos tamanhos obtidos para as imagens. Obtendo, para cada uma das imagens, as dimensões de altura e largura e extraindo a raiz quadrada da multiplicação desses dois valores obtém-se as dimensões da imagem caso esta fosse quadrada. Esta é uma análise útil pois as imagens dos rostos possuem aproximadamente dimensões quadradas e as imagens enviadas ao modelo necessariamente devem possuir também dimensões quadradas. A Figura 4.2 abaixo apresenta a distribuição aproximada do tamanho das imagens utilizando a raiz quadrada da multiplicação das dimensões de altura e largura das imagens. Observa-se que a maioria das imagens se encontra abaixo das dimensões de 400x400 pixels.

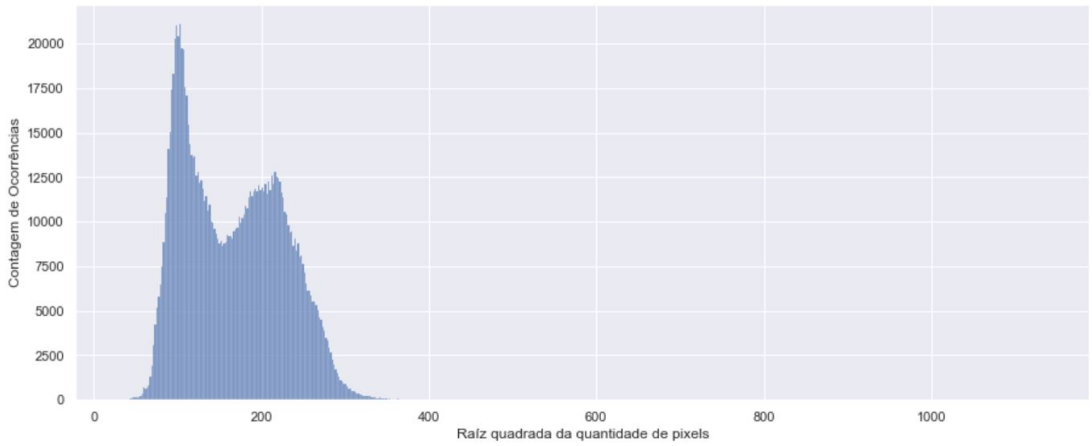


Figura 4.2: Distribuição aproximada do tamanho das imagens no conjunto de dados obtido após o pré-processamento da MTCNN.

4.2.2 Partições para Validação Cruzada

Como foi descrito na Seção 3.2.3, foi realizada a técnica de validação cruzada estratificada de forma a obter a melhor estimativa qualitativa possível das métricas reais do modelo no conjunto de dados disponível. Foi escolhido realizar a validação cruzada utilizando 5 partições, o que resulta em 5 treinamentos de diferentes instâncias do mesmo modelo. A tabela 4.3 apresenta o quantitativo de imagens em cada uma das partições obtidas e suas respectivas porcentagens como um todo dentro da partição. Observa-se como a validação cruzada estratificada procura preservar a porcentagem de amostras de cada classe em cada uma das partições.

	treinamento		**validação**	
	FAKE	**REAL**	**FAKE**	**REAL**
Quantidade Partição 1	630.392	131.960	157.598	32.990
Porcentagem Partição 1	66,15%	13,85%	16,54%	3,46%
Quantidade Partição 2	630.392	131.960	157.598	32.990
Porcentagem Partição 2	66,15%	13,85%	16,54%	3,46%
Quantidade Partição 3	630.392	131.960	157.598	32.990
Porcentagem Partição 3	66,15%	13,85%	16,54%	3,46%
Quantidade Partição 4	630.392	131.960	157.598	32.990
Porcentagem Partição 4	66,15%	13,85%	16,54%	3,46%
Quantidade Partição 5	630.392	131.960	157.598	32.990
Porcentagem Partição 5	66,15%	13,85%	16,54%	3,46%

Tabela 4.3: Distribuição da quantidade de imagens por cada uma das 5 partições da validação cruzada.

4.2.3 Pré-processamento das Imagens

As imagens, antes de serem convertidas para tensores, foram redimensionadas para o tamanho de 224x224 pixels. A escolha do tamanho levou em conta um ponto vital no processo de treinamento de um modelo de aprendizado profundo que é o tempo de treinamento. Aumentar o tamanho da imagem significa aumentar seu espaço ocupado em memória por uma razão quadrática. Considerando que um pixel ocupe em memória um byte para uma imagem preto e branco, uma imagem com dimensões 200x200 pixels ocuparia 4×10^4 bytes, porém uma imagem de dimensões 400x400 pixels não ocuparia $4 * 2 \times 10^4$ mas sim $4^2 \times 10^4$ bytes. Isso significa que é possível inserir quatro imagens de dimensões 200x200 pixels no mesmo espaço que uma única imagem de dimensões 400x400 pixels ocupa. Esse fato se reflete diretamente na quantidade de imagens que podem ser armazenadas diretamente na memória de uma GPU e processadas de forma paralela durante uma iteração do treinamento da rede. Uma vez que o modelo Resnet18 original foi treinada em imagens de dimensões 224x224 pixels, é também considerada uma boa prática manter as dimensões originais das imagens que foram utilizadas para treinar o modelo original, favorecendo a sua convergência.

O redimensionamento da imagem foi realizado utilizando um recorte da região central da face. Inicialmente, a imagem é proporcionalmente redimensionada de forma que a menor dimensão dela se torne do tamanho da dimensão desejada. O recorte final é então realizado removendo trechos apenas da maior dimensão de forma que a imagem se torne quadrada. Quanto em etapa de treinamento, a região de recorte é aleatória, podendo qualquer porção da imagem original ser enviada à rede. Esta é uma técnica de *dataset augmentation* (Seção 2.1.2.10), uma vez que cada imagem aleatoriamente recortada acaba por ser uma nova imagem do conjunto de dados. Em etapa de validação, o recorte é sempre realizado na região central.

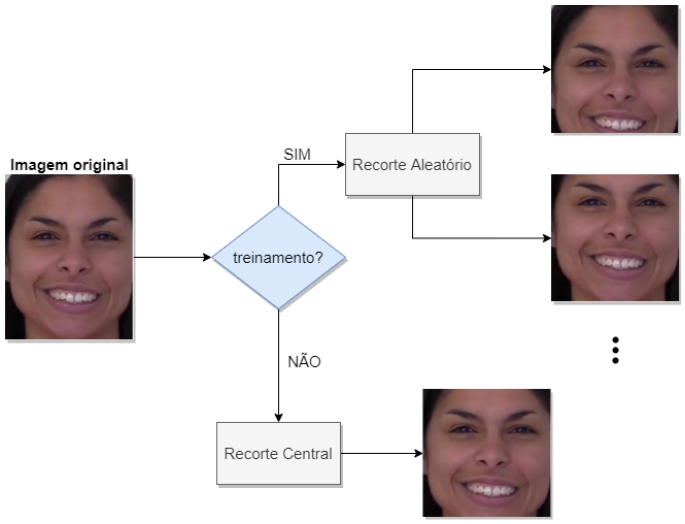


Figura 4.3: Processo de redimensionamento das imagens durante o treinamento e durante a validação. A imagem original é proporcionalmente redimensionada de forma que a menor dimensão original se torna igual à dimensão desejada. O recorte é então realizado removendo trechos apenas da maior dimensão de forma que a imagem se torne quadrada no final.

Após o redimensionamento e o recorte das imagens, estas então são transformadas em tensores com valores no intervalo $[0, 1]$. Este processo é feito dividindo todos os pixels da imagem (valores no intervalo $[0, 255]$) de cada um dos 3 canais (RGB) pelo valor máximo de 255. Uma vez que a rede original foi treinada no conjunto de dados ImageNET, é considerada uma boa prática normalizar as imagens da mesma forma como as imagens no conjunto de dados original foram normalizadas. Desta forma, após a obtenção dos tensores no intervalo de $[0, 1]$, as imagens em um mesmo lote de treinamento foram todas normalizadas utilizando as equações 4.1 até 4.3 abaixo:

$$Pm_c = \frac{1}{224 \times 224} \sum_{i=1}^{224 \times 224} P_{c,i}, \tag{4.1}$$

$$Pstd_c = \sqrt{\frac{\sum_{i=1}^{224 \times 224} (P_{c,i} - Pm_c)^2}{224 \times 224}}, \tag{4.2}$$

$$Pn_{c,i} = \frac{Pn_{c,i} - Pm_c}{Pstd_c}, \tag{4.3}$$

onde Pm_c é a média dos valores do canal c de um tensor, $Pstd_c$ é o desvio padrão dos valores do canal c de um tensor e $Pn_{c,i}$ é o i-ésimo novo valor do canal c de um tensor. A tabela 4.4 apresenta os valores originais de média e desvio padrão utilizados pela ImageNet.

Canal	**Vermelho**	**Verde**	**Azul**
Média (Pm_c)	0.485	0.456	0.406
Desvio Padrão ($Pstd_c$)	0.229	0.224	0.225

Tabela 4.4: Valores de média e desvio padrão originais utilizados nas imagens da ImageNET.

Com isso, conclui-se a etapa de pré-processamento das imagens. A partir de agora, as imagens podem fluir pelo modelo sem problemas.

4.3 Hiperparâmetros

A tabela 4.5 a seguir apresenta os hiperparâmetros utilizados durante o treinamento do modelo. Os hiperparâmetros foram os mesmos para todas as partições da validação cruzada estratificada.

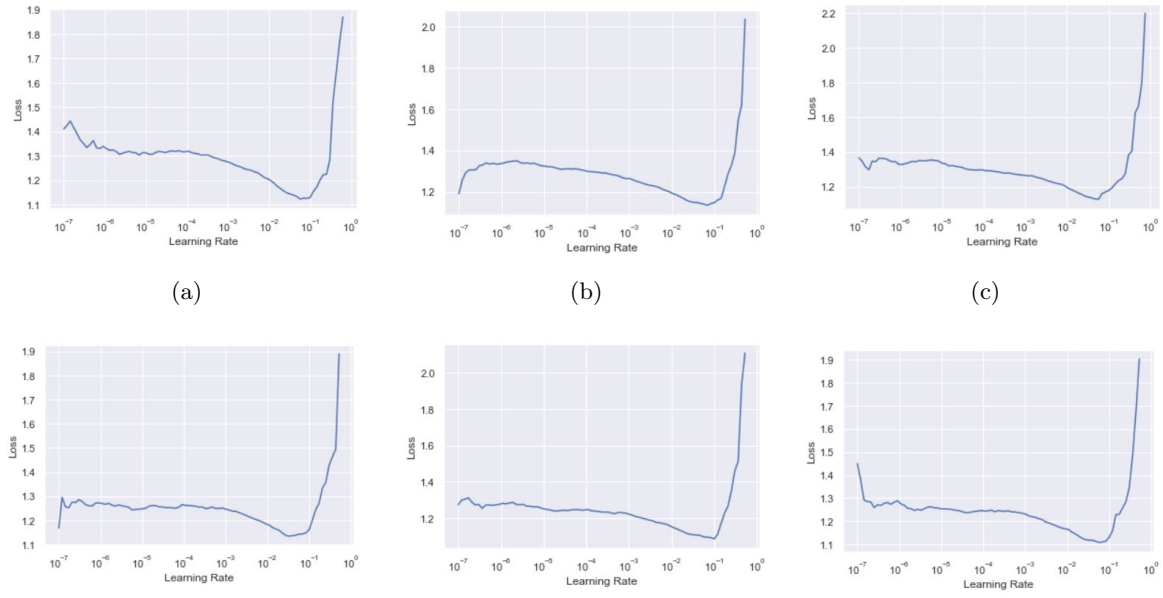
	Valor
Épocas Congeladas	1
Épocas Descongeladas	10
Taxa de Aprendizado Máxima	0.01
Tamanho do Lote	352
Peso da Classe *FAKE*	1.00
Peso da Classe *REAL*	2.7581

Tabela 4.5: Hiperparâmetros utilizados para o treinamento do modelo Resnet18.

Os pesos de cada classe foram utilizados na função de entropia cruzada de forma a penalizar o modelo com mais intensidade caso errasse uma imagem da classe *REAL*. Isso foi feito pois o conjunto de dados está desbalanceado e a probabilidade de o modelo ver imagens da classe real é menor que ver imagens da classe *FAKE*. Para o cálculo do peso da classe *REAL*, foi obtido o número de exemplos da classe *FAKE* e o valor foi dividido pelo número de exemplos da classe *REAL* e posteriormente o resultado foi dividido por 1.73. O valor da divisão final por 1.73 se mostrou, de forma empírica, mais estável durante o processo de treinamento.

Como foi discutido na Seção 3.2, utilizou-se a política da taxa de aprendizado de um ciclo para o treinamento, onde o valor da taxa de aprendizado é variado de um valor mínimo até um valor máximo e depois retornado até outro valor mínimo. A biblioteca Fastai já fornece por padrão os valores de mínimo da taxa de aprendizado dado um valor máximo, bem como o ajuste de curva utilizado para realizar a transição. Ela realiza o aumento do valor da taxa de aprendizado partindo do valor máximo escolhido dividido por 25, alcançando o valor máximo escolhido e por fim diminuindo até o valor máximo escolhido dividido por 10000. Logo, o valor mínimo inicial é 0.0004 e o valor mínimo final é 10^{-6}.

O valor máximo da taxa de aprendizado foi escolhido como descrito na Seção 3.2, isto é, realizando um teste de faixa da taxa de aprendizado e escolhendo um valor que, na plotagem da taxa de aprendizado versus o custo, estivesse em uma região de elevado declive de custo. Para todas as partições da validação cruzada e para o conjunto completo de treinamento, a região de elevado declive de custo convergiu no mesmo valor. A Figura 4.4 abaixo apresenta as plotagens dos testes realizados.



(a) (b) (c)

(d) (e) (f)

Figura 4.4: Resultados para o teste de faixa da taxa de aprendizado para a (a) partição 1, (b) partição 2, (c) partição 3, (d) partição 4 e (e) partição 5 da validação cruzada. O resultado em (f) representa o teste para o conjunto completo de treinamento.

Durante o treinamento, também foram inseridas duas chamadas especiais: uma para parar o treinamento do modelo caso este estivesse perdendo a qualidade (medida através do acompanhamento do valor da métrica MCC) e não melhorasse durante o período de 2 épocas e outra para que o modelo fosse salvo ao final de cada época caso esta tivesse apresentado uma métrica MCC melhor que a da época anterior.

4.4 Treinamento e Validação do Modelo

As figuras nas Seções 4.4.1, 4.4.2, 4.4.3, 4.4.4 e 4.4.5 a seguir apresentam, em sequência, para cada uma das partições da validação cruzada estratificada e para o conjunto final completo de treinamento:

1. o processo de treinamento do modelo na partição em questão, onde são explicitadas o número da época, as métricas mencionadas na Seção 3.3.1 para a fase de validação da época em questão, o tempo total de treinamento e validação da época em questão e os gráficos do custo gerado pelo modelo na fase de treinamento e na fase de validação (Figuras 4.5, 4.8, 4.11, 4.14, 4.17 e 4.20).

2. as matrizes de confusão produzidas pela fase de validação final após o treinamento completo do modelo (Figuras 4.6, 4.9, 4.12, 4.15, 4.18 e 4.21).

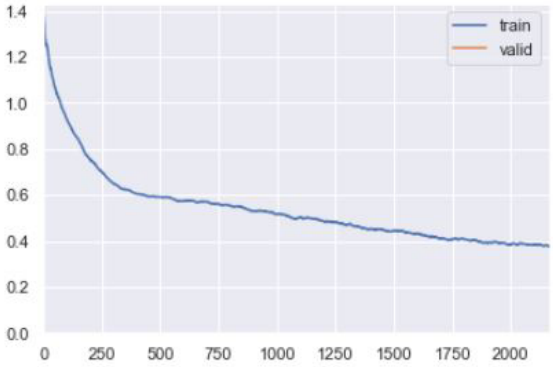
3. alguns exemplos de imagens que provocaram o maior custo do modelo na hora da validação, isto é, que deixaram o modelo mais "confuso"(Figuras 4.7, 4.10, 4.13, 4.16, 4.19 e 4.22).

A Seção 4.4.7 apresenta, por fim, as curvas ROC obtidas para todas as instâncias dos modelos da validação cruzada estratificada e para o modelo final do conjunto de treinamento completo (Figura 4.23).

4.4.1 Partição 1

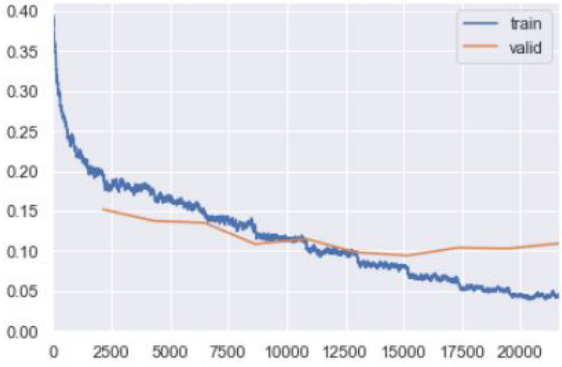
epoch	train_loss	valid_loss	accuracy	matthews_corrcoef	roc_auc_score	time
0	0.377719	0.281932	0.883534	0.608935	0.920233	54:11

```
Better model found at epoch 0 with matthews_corrcoef value: 0.6089347709828594.
```



epoch	train_loss	valid_loss	accuracy	matthews_corrcoef	roc_auc_score	time
0	0.196007	0.152010	0.938632	0.789907	0.977348	51:31
1	0.172332	0.137691	0.945547	0.810704	0.979986	55:04
2	0.147561	0.135278	0.943800	0.807296	0.980614	53:39
3	0.130908	0.108684	0.956797	0.847662	0.986706	52:35
4	0.112199	0.115198	0.950941	0.838657	0.989246	52:58
5	0.098009	0.098203	0.960632	0.864596	0.990592	51:12
6	0.079250	0.094263	0.962936	0.872989	0.991811	51:07
7	0.063523	0.104070	0.959887	0.865385	0.991952	51:11
8	0.053773	0.103138	0.965176	0.879262	0.991521	50:10
9	0.044985	0.109534	0.965711	0.880623	0.991238	50:29

```
Better model found at epoch 0 with matthews_corrcoef value: 0.7899067179945869.
```




```
Better model found at epoch 1 with matthews_corrcoef value: 0.8107039177254557.
Better model found at epoch 3 with matthews_corrcoef value: 0.8476618890955913.
Better model found at epoch 5 with matthews_corrcoef value: 0.8645955495281441.
Better model found at epoch 6 with matthews_corrcoef value: 0.8729894360034502.
Better model found at epoch 8 with matthews_corrcoef value: 0.8792617334600477.
Better model found at epoch 9 with matthews_corrcoef value: 0.8806232319437073.
```

Figura 4.5: Resultados das métricas de custo, acurácia, MCC e área sob a curva ROC durante a etapa de validação do modelo na partição 1 da validação cruzada. A parte superior abrange a etapa com o modelo congelado e a parte inferior abrange a etapa com o modelo descongelado. O tempo de treinamento e validação de cada época é apresentado na coluna **time**.

(a) (b)

Figura 4.6: Matrizes de confusão (a) absoluta e (b) normalizada resultantes para o conjunto de validação da partição 1 da validação cruzada após o treinamento do modelo.

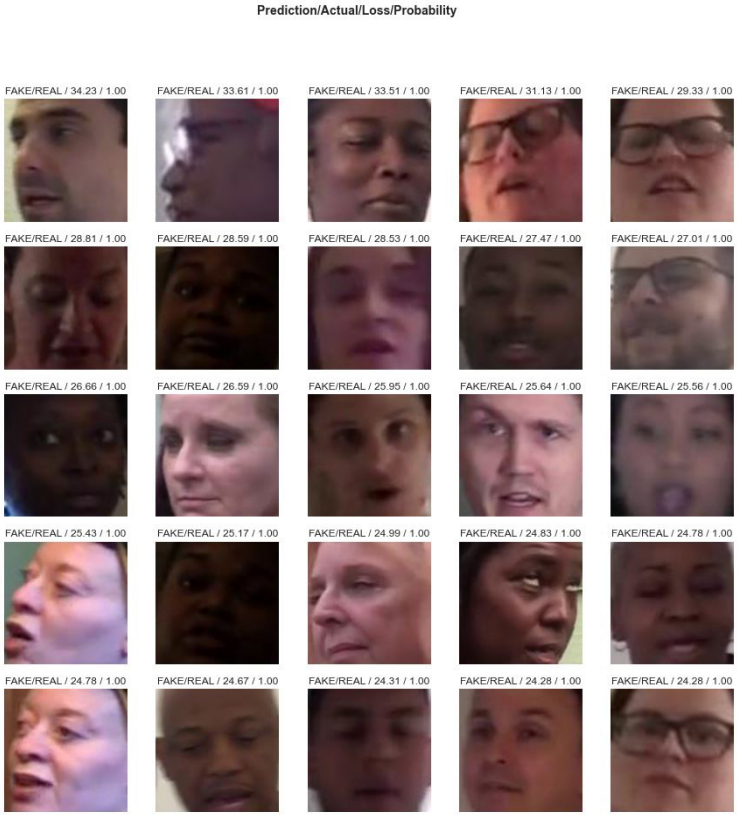
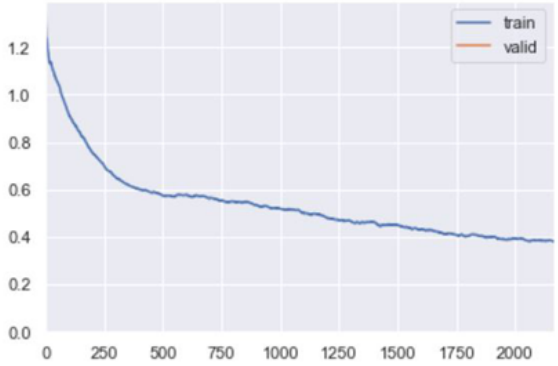


Figura 4.7: Exemplos no conjunto de validação da partição 1 da validação cruzada que geraram o maior valor de custo para o modelo treinado. A interpretação é de que esses são os exemplos que deixaram a rede mais "confusa". A legenda acima de cada imagem representa a classe predita pelo modelo, a classe real a qual o modelo pertence, o custo gerado na classificação e a probabilidade que o modelo deu para a classe predita.

4.4.2 Partição 2

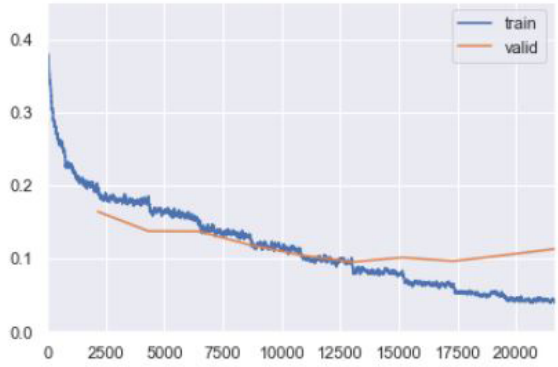
epoch	train_loss	valid_loss	accuracy	matthews_corrcoef	roc_auc_score	time
0	0.381505	0.264074	0.895948	0.623256	0.921672	56:40

```
Better model found at epoch 0 with matthews_corrcoef value: 0.6232562589309956.
```



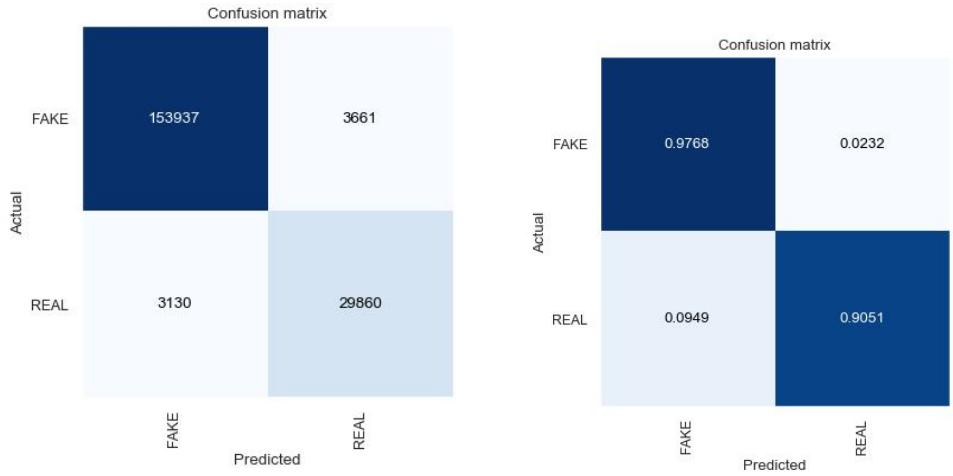
epoch	train_loss	valid_loss	accuracy	matthews_corrcoef	roc_auc_score	time
0	0.196096	0.164229	0.932031	0.775442	0.975516	50:55
1	0.182135	0.137373	0.946067	0.815329	0.982282	50:27
2	0.152024	0.137022	0.941733	0.806043	0.982592	49:31
3	0.129410	0.118513	0.951534	0.835609	0.986611	50:24
4	0.109563	0.104661	0.957353	0.853936	0.989381	52:54
5	0.100167	0.095351	0.960265	0.863625	0.990710	56:01
6	0.081855	0.101366	0.958019	0.859645	0.991526	52:07
7	0.064853	0.096472	0.962899	0.872240	0.991644	51:52
8	0.048239	0.104472	0.964368	0.876358	0.991335	51:11
9	0.040464	0.113154	0.963707	0.874313	0.991010	51:01

```
Better model found at epoch 0 with matthews_corrcoef value: 0.7754420241972589.
```




```
Better model found at epoch 1 with matthews_corrcoef value: 0.815328536942094.
Better model found at epoch 3 with matthews_corrcoef value: 0.835609058354621.
Better model found at epoch 4 with matthews_corrcoef value: 0.8539362983014827.
Better model found at epoch 5 with matthews_corrcoef value: 0.863624654653984.
Better model found at epoch 7 with matthews_corrcoef value: 0.8722399877154617.
Better model found at epoch 8 with matthews_corrcoef value: 0.8763575559706996.
```

Figura 4.8: Resultados das métricas de custo, acurácia, MCC e área sob a curva ROC durante a etapa de validação do modelo na partição 2 da validação cruzada. A parte superior abrange a etapa com o modelo congelado e a parte inferior abrange a etapa com o modelo descongelado. O tempo de treinamento e validação de cada época é apresentado na coluna **time**.



(a) (b)

Figura 4.9: Matrizes de confusão (a) absoluta e (b) normalizada resultantes para o conjunto de validação da partição 2 da validação cruzada após o treinamento do modelo.

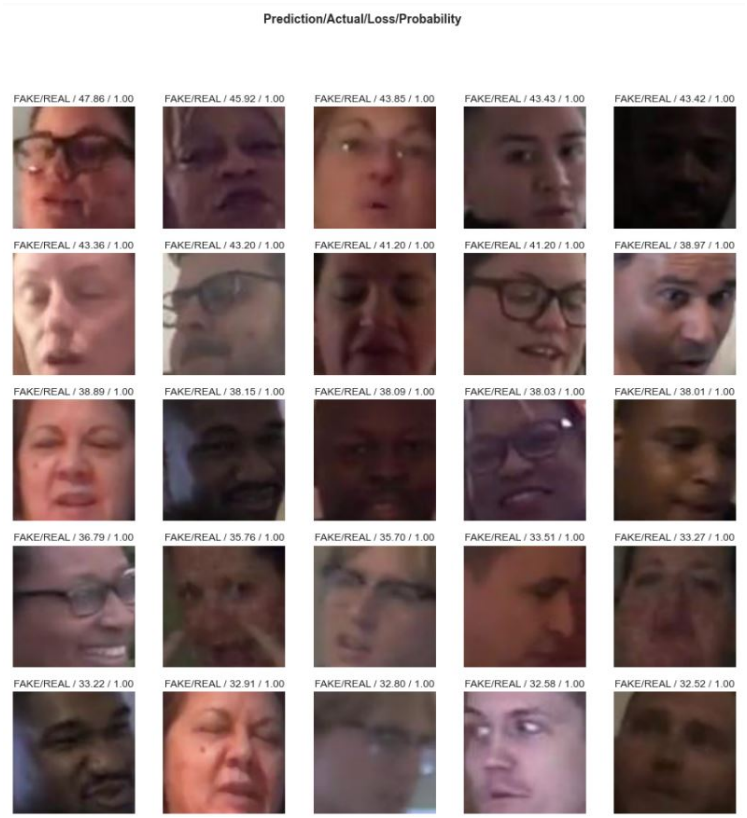
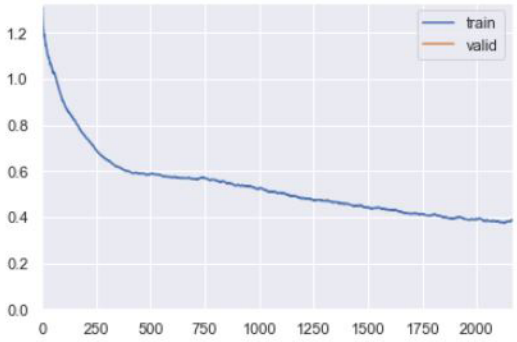


Figura 4.10: Exemplos no conjunto de validação da partição 2 da validação cruzada que geraram o maior valor de custo para o modelo treinado. A interpretação é de que esses são os exemplos que deixaram a rede mais "confusa". A legenda acima de cada imagem representa a classe predita pelo modelo, a classe real a qual o modelo pertence, o custo gerado na classificação e a probabilidade que o modelo deu para a classe predita.

4.4.3 Partição 3

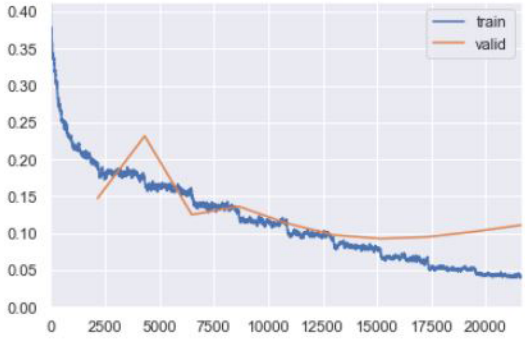
epoch	train_loss	valid_loss	accuracy	matthews_corrcoef	roc_auc_score	time
0	0.388233	0.268780	0.890843	0.618169	0.921314	57:36

```
Better model found at epoch 0 with matthews_corrcoef value: 0.6181686805156662.
```



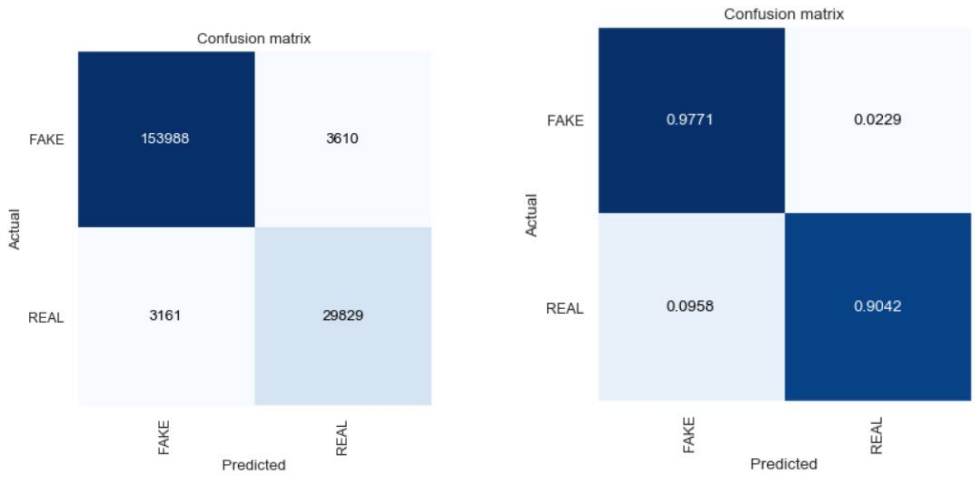
epoch	train_loss	valid_loss	accuracy	matthews_corrcoef	roc_auc_score	time
0	0.193571	0.147344	0.938999	0.787100	0.975637	52:25
1	0.172458	0.231728	0.908714	0.726317	0.971939	53:33
2	0.155394	0.125230	0.948318	0.816029	0.981721	52:09
3	0.128542	0.136220	0.941287	0.815201	0.988279	52:28
4	0.118819	0.113797	0.952883	0.839857	0.987426	53:49
5	0.094786	0.097702	0.959326	0.861785	0.990956	51:47
6	0.079148	0.092728	0.960134	0.864883	0.991649	52:08
7	0.068067	0.095047	0.962600	0.871782	0.991763	51:25
8	0.051656	0.102294	0.964116	0.875789	0.991447	52:17
9	0.041667	0.111116	0.964473	0.876590	0.991186	55:07

```
Better model found at epoch 0 with matthews_corrcoef value: 0.7870996221630469.
```




```
Better model found at epoch 2 with matthews_corrcoef value: 0.8160293402521481.
Better model found at epoch 4 with matthews_corrcoef value: 0.8398565619398023.
Better model found at epoch 5 with matthews_corrcoef value: 0.8617850129591854.
Better model found at epoch 6 with matthews_corrcoef value: 0.8648832653742421.
Better model found at epoch 7 with matthews_corrcoef value: 0.8717823024709855.
Better model found at epoch 8 with matthews_corrcoef value: 0.8757886033425081.
Better model found at epoch 9 with matthews_corrcoef value: 0.8765896668965436.
```

Figura 4.11: Resultados das métricas de custo, acurácia, MCC e área sob a curva ROC durante a etapa de validação do modelo na partição 3 da validação cruzada. A parte superior abrange a etapa com o modelo congelado e a parte inferior abrange a etapa com o modelo descongelado. O tempo de treinamento e validação de cada época é apresentado na coluna **time**.



(a) (b)

Figura 4.12: Matrizes de confusão (a) absoluta e (b) normalizada resultantes para o conjunto de validação da partição 3 da validação cruzada após o treinamento do modelo.

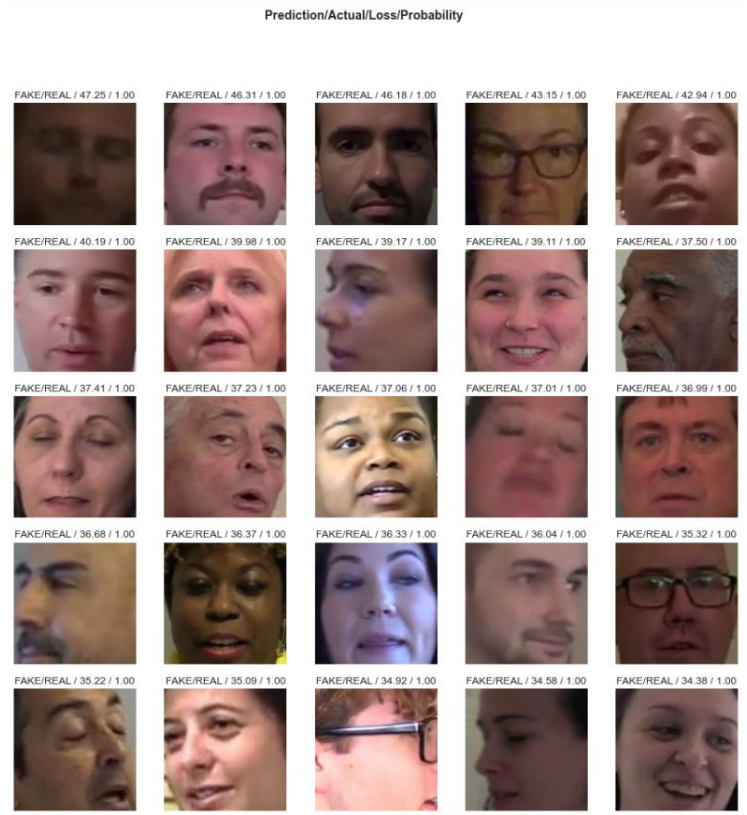
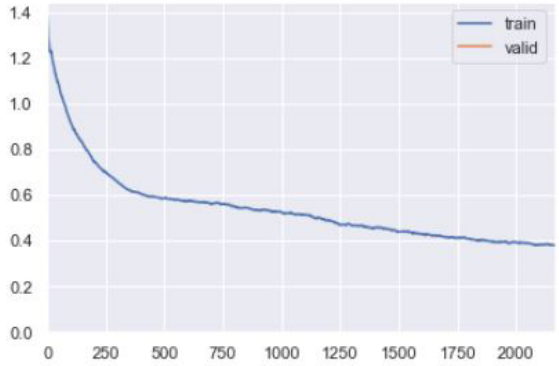


Figura 4.13: Exemplos no conjunto de validação da partição 3 da validação cruzada que geraram o maior valor de custo para o modelo treinado. A interpretação é de que esses são os exemplos que deixaram a rede mais "confusa". A legenda acima de cada imagem representa a classe predita pelo modelo, a classe real a qual o modelo pertence, o custo gerado na classificação e a probabilidade que o modelo deu para a classe predita.

4.4.4 Partição 4

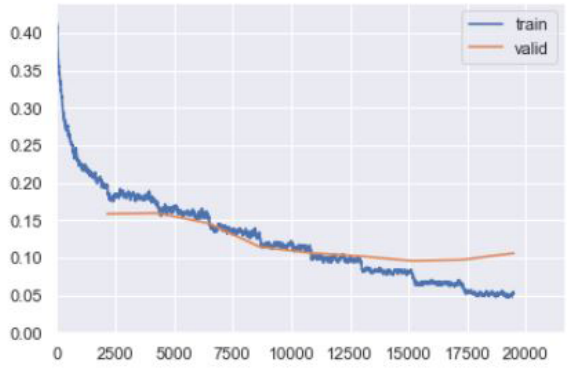
epoch	train_loss	valid_loss	accuracy	matthews_corrcoef	roc_auc_score	time
0	0.377285	0.275351	0.888141	0.618855	0.921359	50:32

```
Better model found at epoch 0 with matthews_corrcoef value: 0.618855344831957.
```



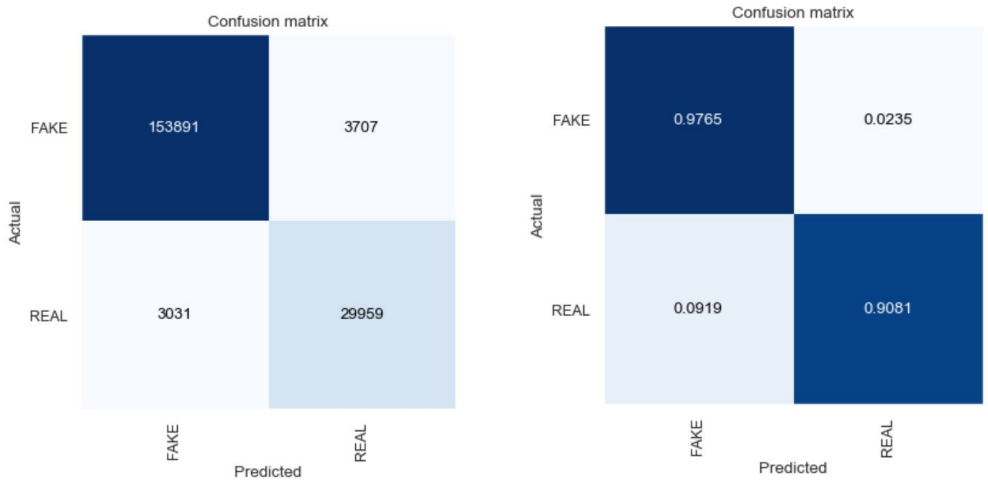
epoch	train_loss	valid_loss	accuracy	matthews_corrcoef	roc_auc_score	time
0	0.196563	0.158586	0.937273	0.791354	0.979061	52:13
1	0.171954	0.159622	0.935542	0.770113	0.971580	51:22
2	0.154652	0.145838	0.943910	0.809936	0.982529	49:18
3	0.131786	0.114528	0.953733	0.840420	0.986564	50:31
4	0.110514	0.106358	0.954514	0.846985	0.989354	50:24
5	0.093185	0.101710	0.957904	0.856359	0.990049	50:11
6	0.078517	0.095904	0.962285	0.870415	0.991246	50:28
7	0.064159	0.097347	0.964646	0.877560	0.991651	50:21
8	0.052153	0.105927	0.963838	0.874834	0.991236	51:07
9	0.042190	0.115868	0.963387	0.873767	0.991053	55:11

```
Better model found at epoch 0 with matthews_corrcoef value: 0.7913542515112569.
```




```
Better model found at epoch 2 with matthews_corrcoef value: 0.8099359968458094.
Better model found at epoch 3 with matthews_corrcoef value: 0.8404202670912934.
Better model found at epoch 4 with matthews_corrcoef value: 0.8469848075776131.
Better model found at epoch 5 with matthews_corrcoef value: 0.8563593106043276.
Better model found at epoch 6 with matthews_corrcoef value: 0.8704154130388204.
Better model found at epoch 7 with matthews_corrcoef value: 0.8775596603431078.
No improvement since epoch 7: early stopping
```

Figura 4.14: Resultados das métricas de custo, acurácia, MCC e área sob a curva ROC durante a etapa de validação do modelo na partição 4 da validação cruzada. A parte superior abrange a etapa com o modelo congelado e a parte inferior abrange a etapa com o modelo descongelado. O tempo de treinamento e validação de cada época é apresentado na coluna **time**.



(a) (b)

Figura 4.15: Matrizes de confusão (a) absoluta e (b) normalizada resultantes para o conjunto de validação da partição 4 da validação cruzada após o treinamento do modelo.

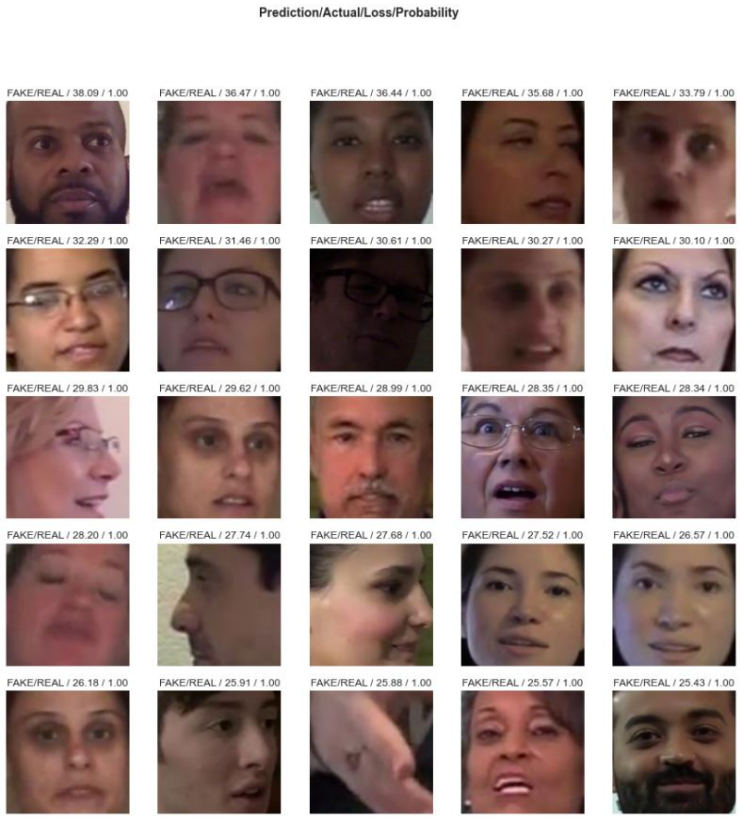
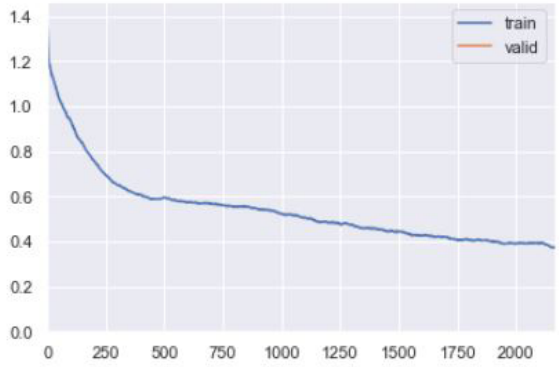


Figura 4.16: Exemplos no conjunto de validação da partição 4 da validação cruzada que geraram o maior valor de custo para o modelo treinado. A interpretação é de que esses são os exemplos que deixaram a rede mais "confusa". A legenda acima de cada imagem representa a classe predita pelo modelo, a classe real a qual o modelo pertence, o custo gerado na classificação e a probabilidade que o modelo deu para a classe predita.

4.4.5 Partição 5

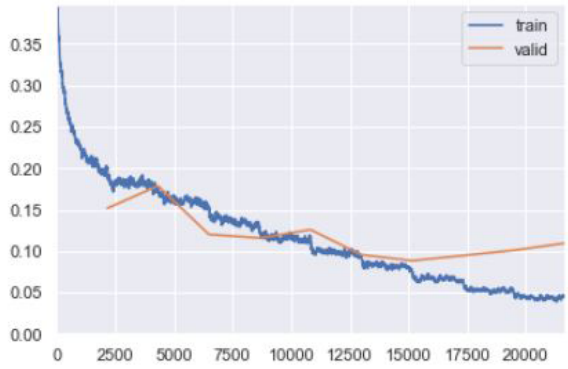
epoch	train_loss	valid_loss	accuracy	matthews_corrcoef	roc_auc_score	time
0	0.373470	0.305773	0.864577	0.598049	0.924189	59:32

```
Better model found at epoch 0 with matthews_corrcoef value: 0.5980492544982019.
```



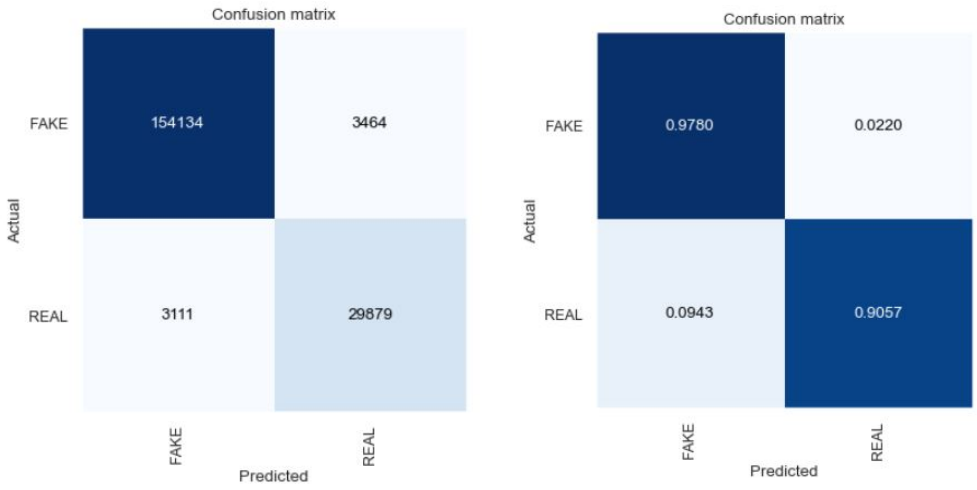
epoch	train_loss	valid_loss	accuracy	matthews_corrcoef	roc_auc_score	time
0	0.199542	0.151575	0.938186	0.783214	0.974597	49:29
1	0.174598	0.178660	0.930526	0.767384	0.974294	56:10
2	0.153183	0.120049	0.951177	0.827448	0.983415	51:32
3	0.128307	0.115638	0.950957	0.835419	0.987360	54:07
4	0.116074	0.125715	0.948197	0.831174	0.988791	50:57
5	0.093548	0.095441	0.962364	0.869369	0.990491	50:26
6	0.075409	0.088320	0.964641	0.876807	0.991638	51:34
7	0.063297	0.094463	0.964326	0.877293	0.992032	53:59
8	0.050174	0.100896	0.965501	0.880014	0.991637	53:10
9	0.044193	0.109374	0.964998	0.878543	0.991368	54:16

```
Better model found at epoch 0 with matthews_corrcoef value: 0.7832136918822512.
```




```
Better model found at epoch 2 with matthews_corrcoef value: 0.8274475342468308.
Better model found at epoch 3 with matthews_corrcoef value: 0.8354187598622488.
Better model found at epoch 5 with matthews_corrcoef value: 0.8693692217374868.
Better model found at epoch 6 with matthews_corrcoef value: 0.8768068325755505.
Better model found at epoch 7 with matthews_corrcoef value: 0.87729326445668.
Better model found at epoch 8 with matthews_corrcoef value: 0.8800144551536717.
```

Figura 4.17: Resultados das métricas de custo, acurácia, MCC e área sob a curva ROC durante a etapa de validação do modelo na partição 5 da validação cruzada. A parte superior abrange a etapa com o modelo congelado e a parte inferior abrange a etapa com o modelo descongelado. O tempo de treinamento e validação de cada época é apresentado na coluna **time**.



(a) (b)

Figura 4.18: Matrizes de confusão (a) absoluta e (b) normalizada resultantes para o conjunto de validação da partição 5 da validação cruzada após o treinamento do modelo.

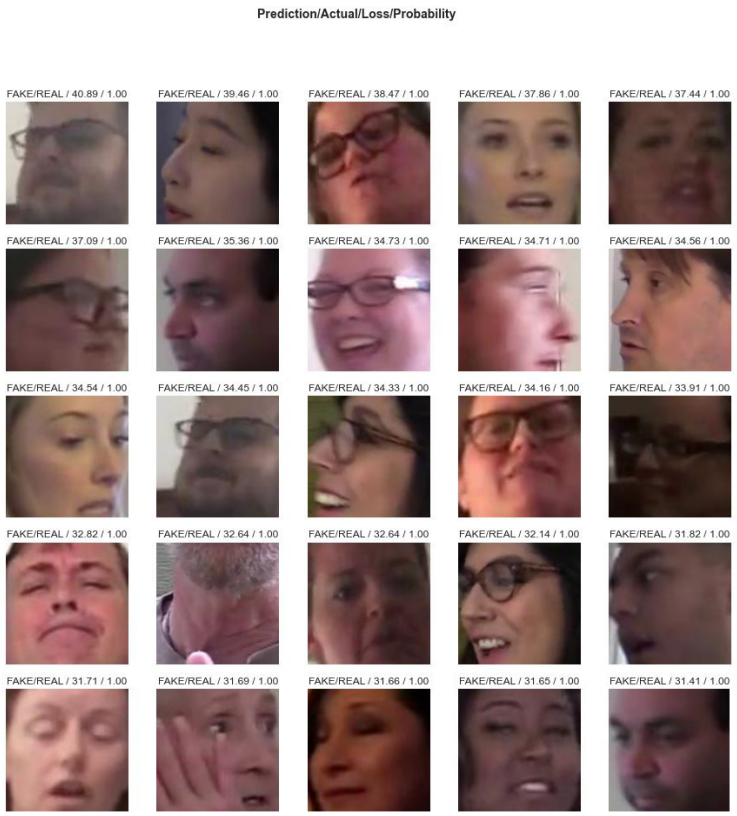
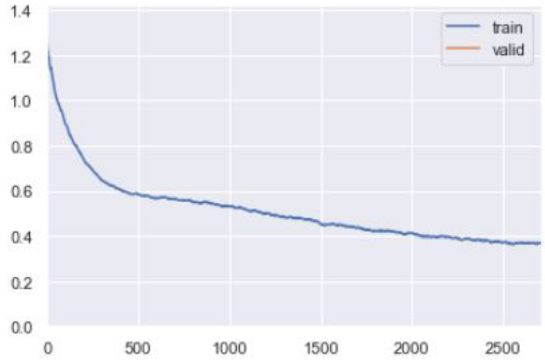


Figura 4.19: Exemplos no conjunto de validação da partição 5 da validação cruzada que geraram o maior valor de custo para o modelo treinado. A interpretação é de que esses são os exemplos que deixaram a rede mais "confusa". A legenda acima de cada imagem representa a classe predita pelo modelo, a classe real a qual o modelo pertence, o custo gerado na classificação e a probabilidade que o modelo deu para a classe predita.

4.4.6 Conjunto de dados completo

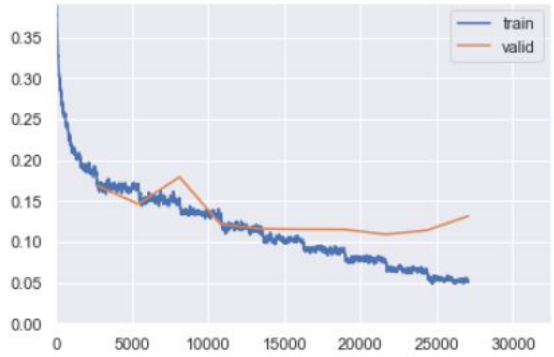
epoch	train_loss	valid_loss	accuracy	matthews_corrcoef	roc_auc_score	time
0	0.368598	0.285544	0.881943	0.619524	0.924868	1:00:06

```
Better model found at epoch 0 with matthews_corrcoef value: 0.6195242695164843.
```



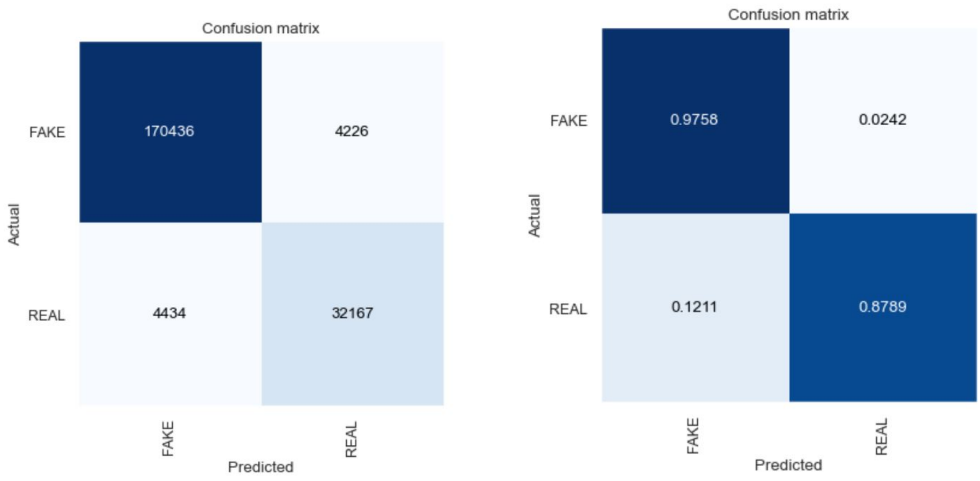
epoch	train_loss	valid_loss	accuracy	matthews_corrcoef	roc_auc_score	time
0	0.176387	0.169006	0.928492	0.767816	0.975823	1:01:54
1	0.170924	0.146314	0.940406	0.787317	0.975032	1:02:17
2	0.152264	0.179419	0.922907	0.758357	0.976717	1:06:36
3	0.135315	0.121325	0.950668	0.830383	0.984993	1:04:34
4	0.113547	0.115973	0.952046	0.836592	0.987060	1:02:47
5	0.098323	0.115466	0.951156	0.835754	0.987942	1:03:10
6	0.091150	0.115230	0.954043	0.842678	0.987590	1:06:12
7	0.075169	0.109170	0.958497	0.855916	0.988523	1:09:18
8	0.066593	0.114210	0.957877	0.854555	0.988913	1:12:14
9	0.051655	0.131567	0.955482	0.847214	0.988046	1:12:49
10	0.041378	0.152793	0.959008	0.856589	0.986869	1:09:59

```
Better model found at epoch 0 with matthews_corrcoef value: 0.7678162026924191.
```




```
Better model found at epoch 1 with matthews_corrcoef value: 0.78731666755654.
Better model found at epoch 3 with matthews_corrcoef value: 0.8303832449113355.
Better model found at epoch 4 with matthews_corrcoef value: 0.8365921583670158.
Better model found at epoch 6 with matthews_corrcoef value: 0.8426778868464095.
Better model found at epoch 7 with matthews_corrcoef value: 0.8559155556981736.
Better model found at epoch 10 with matthews_corrcoef value: 0.8565894507267037.
No improvement since epoch 7: early stopping
```

Figura 4.20: Resultados das métricas de custo, acurácia, MCC e área sob a curva ROC durante a etapa de validação do modelo utilizando o conjunto de dados completo. A parte superior abrange a etapa com o modelo congelado e a parte inferior abrange a etapa com o modelo descongelado. O tempo de treinamento e validação de cada época é apresentado na coluna **time**.



(a) (b)

Figura 4.21: Matrizes de confusão (a) absoluta e (b) normalizada resultantes para o conjunto de teste da após o treinamento do modelo no conjunto de treinamento completo.

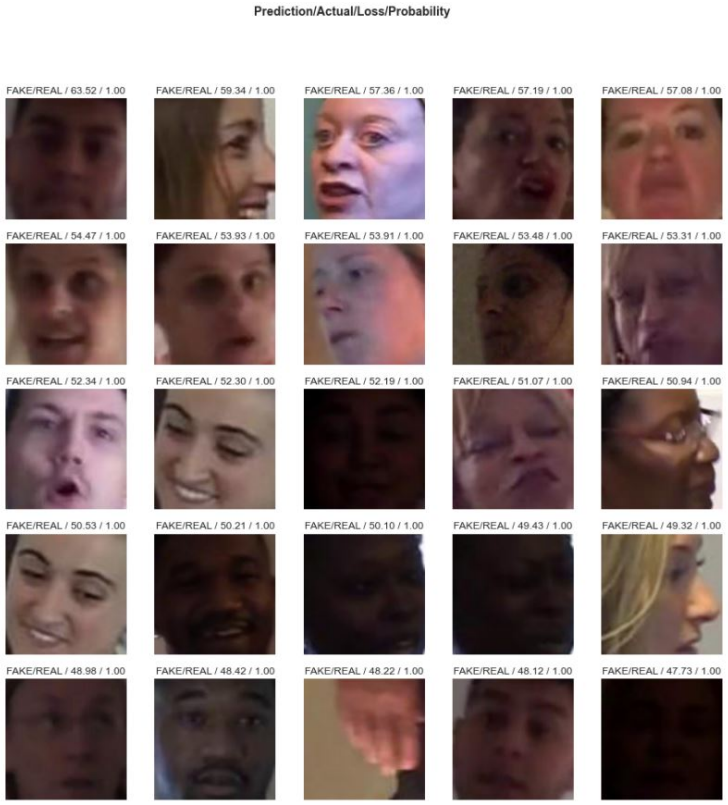


Figura 4.22: Exemplos no conjunto de teste do conjunto de treinamento completo que geraram o maior valor de custo para o modelo treinado. A interpretação é de que esses são os exemplos que deixaram a rede mais "confusa". A legenda acima de cada imagem representa a classe predita pelo modelo, a classe real a qual o modelo pertence, o custo gerado na classificação e a probabilidade que o modelo deu para a classe predita.

4.4.7 Curvas ROC

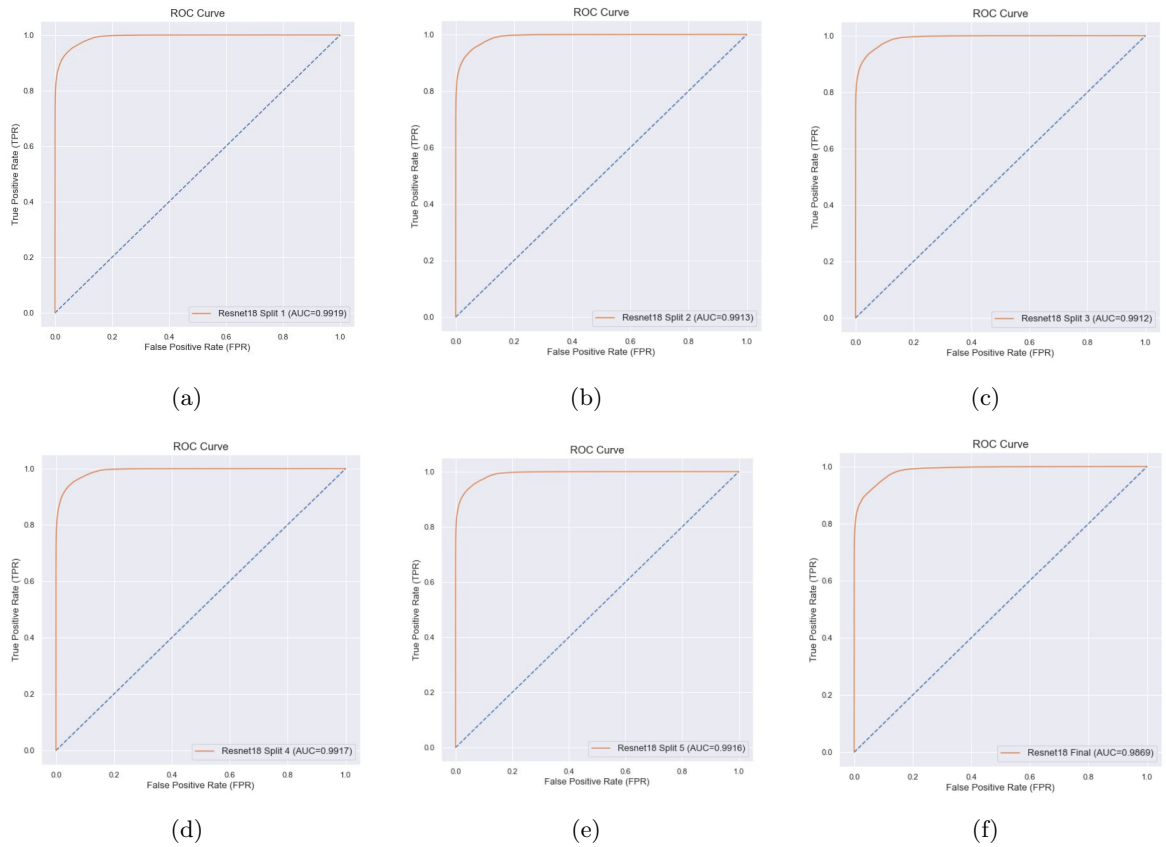


Figura 4.23: Curvas ROC resultantes para os modelos da (a) partição 1, (b) partição 2, (c) partição 3, (d) partição 4 e (e) partição 5 da validação cruzada estratificada e (f) do modelo obtido do conjunto de dados completo.

4.5 Análise dos Resultados da Validação Cruzada

A tabela 4.6 a seguir apresenta a média e o desvio padrão obtido para as métricas de acurácia, MCC, área sob a curva ROC, taxa de verdadeiros positivos *tvp* (sensibilidade) e taxa de verdadeiros negativos *tvn* (especificidade).

	Média	**Desvio Padrão**
Acurácia	96,49%	± 0,05%
MCC	0,8782	± 0,0018
Área sob a curva ROC	0,9914	± 0,0002
tvp	97,75%	± 0,09%
tvn	90,56%	± 0,13%

Tabela 4.6: Tabela de médias e desvios padrões de algumas da métricas utilizadas na validação do modelo para as 5 partições da validação cruzada.

Observa-se que, para todos as métricas, o desvio padrão foi relativamente baixo. Isso significa que as partições obtidas pela validação cruzada estratificada são representativas, não existindo em nenhuma delas a forte presença de algum viés. Significa também que o modelo foi capaz de generalizar o que aprendeu no treinamento de forma consistente durante todas as etapas.

O modelo, em todos os casos da validação cruzada, foi capaz de manter um valor de MCC acima de 0,87, acertando em média 97,75% dos rostos falsos e 90,56% dos rostos reais. Isso significa que o modelo acerta, aproximadamente, 44 a cada 45 rostos falsos e 9 a cada 10 rostos reais. Além de serem valores relativamente bons, o modelo atingiu um certo resultado desejado de conservadorismo. Ele acerta a maioria dos rostos falsos às custas de alguns rostos reais.

Observando a área sob a curva ROC (Figura 4.23) dos 5 modelos gerados pela validação cruzada, temos um valor superior a 0,99 em todos os casos. Todos os 5 gráficos apresentam um resultado significativamente semelhante e satisfatório do ponto de vista de capacidade de separação de classes do modelo. É importante ressaltar que um modelo com um bom valor para a área sob a curva ROC não necessariamente é um modelo confiante. Um modelo confiante atribui altos valores de probabilidade em suas predições. Um modelo com um alto valor para a área sob a curva ROC apresenta alta capacidade de separabilidade dos dados. Supondo um classificador para uma tarefa de classificação binária que sempre acerte a classe de qualquer novo exemplo apresentado porém sempre acerte a classe com probabilidade de 50,1%, uma vez que é considerado pertencente à classe caso a probabilidade para uma das classes esteja acima de 50%. Este classificador receberá uma pontuação perfeita da área sob a curva ROC, porém apresentará pouca confiança em todas as suas predições. Essa distribuição de probabilidades é melhor estudada no modelo final (Seção 4.6).

As Figuras 4.7, 4.10, 4.13, 4.16 e 4.19 apresentam algumas imagens que geraram a maior confusão no classificador. Observa-se em todos os casos que a confusão foi gerada por uma imagem real que foi incorretamente classificada como falsa. Isso mostra o conservadorismo do modelo e de fato é um comportamento benéfico pela proposta do que o modelo deveria ser. O modelo apresentou confusão para os casos opostos também, onde um rosto falso foi predito como real, porém estes não geraram custos tão significativos como o outro caso.

Alguns aspectos das imagens podem ter levado o classificador a incorretamente classificar os exemplos, como por exemplo o desfoque de movimento, fenômeno que pode gerar inconsistências no rosto suficientes para o modelo acreditar que se trata de um *deepfake*. Uma vez que o modelo não possui a informação temporal do vídeo original do qual o rosto se originou, não existem informações suficientes para ele compreender com clareza a diferença de uma distorção por desfoque de movimento e uma distorção por uma manipulação artificial do rosto. Outro aspecto vital vem do fato que algumas técnicas de geração de *deepfakes*, assim como a MTCNN, não são imediatamente capazes de encontrar todos os rostos em um vídeo para realizar a substituição facial. A técnica utilizada de recuperar imagens de rostos estáticas pode acabar por recuperar momentos em que o rosto não sofreu substituição por uma falha ou imprecisão do algoritmo de *deepfakes*. Esses casos ainda receberam a classe *FAKE*, pois esta é definida pelo vídeo original e não pelos quadros individuais do vídeo.

Observa-se a presença também de imagens incorretas no grupo de classificações incorretas, isto é, imagens que não são rostos humanos. Esse é um tipo de ruído introduzido pela incerteza da rede MTCNN utilizada para extração de rostos, que identificou áreas incorretas do vídeo como rostos humanos, inserindo-as nas imagens de saída no processo de pré-processamento do conjunto de dados original. É possível observar esse ruído na Figura 4.16, na imagem da última fileira, terceira coluna. A imagem em questão é uma mão, e não um rosto. Outro exemplo presente está na Figura 4.19, na imagem da penúltima fileira, segunda coluna. A imagem em questão trata-se do pescoço do indivíduo, e não seu rosto. É de se esperar que uma rede que esteja aguardando necessariamente rostos humanos em todo o seu processo de treinamento, quando se deparar com imagens desse tipo, acredite que os rostos sofreram uma severa manipulação.

As Figuras 4.5, 4.8, 4.11, 4.14 e 4.17 comprovam a importância dos detalhes discutidos na Seção 2.1.2.5. Todos os casos apresentaram início de um *overfitting* refletido no aumento do custo de validação (*valid_loss*) conforme o treinamento se estendia por mais épocas. Se destaca então a importância da técnica de regularização utilizada, a interrupção do treinamento caso as métricas específicas de performance não melhorassem por uma quantidade pré definida de épocas. Desta forma é possível salvar e manter o melhor estado do classificador durante o treinamento.

4.6 Análise dos Resultados do Treinamento no Conjunto Final

A tabela 4.7 abaixo indica as métricas obtidas pela etapa de treinamento no conjunto de dados completo.

	Obtido	**Esperado**	**Diferença**
Acurácia	95,85%	96,49%	-0,64%
MCC	0,8559	0,8782	-0,0223
Área sob a curva ROC	0,9885	0,9914	-0,003
$\boldsymbol{tp_n}$	97,58%	97,75%	-0,17%
$\boldsymbol{tn_n}$	87,89%	90,56%	-2,67%

Tabela 4.7: Tabela de resultados obtidos no conjunto de treinamento e teste final, resultados esperados para cada uma das métricas pela validação cruzada e diferença entre o valor obtido para a métrica e o valor esperado.

Observa-se uma degradação de todas as métricas em relação às métricas esperadas pela validação cruzada. A degradação é relativamente baixa, porém existente. Algumas razões possíveis são listadas a seguir:

- Viés (*bias*): O conjunto de dados DFDC não apresenta apenas *deepfakes*, mas também manipulações vocais. Essas representam uma parcela bem inferior da quantidade de vídeos do conjunto de dados, porém existem. Existe a possibilidade de alguns desses casos terem convergido em maior quantidade para o conjunto de testes durante a amostragem aleatória.

Isso caracteriza um viés específico. O modelo foi treinado apenas em imagens de faces humanas, não possuindo informação sonora do vídeo o qual a imagem que ele foi treinado pertence.

- Hiperparâmetros: Como estamos lidando com uma quantidade 25% maior de dados para treinamento que anteriormente durante a validação cruzada, os hiperparâmetros do modelo talvez deixem de ser os mais apropriados para alcançar a convergência no mesmo número de épocas que anteriormente. Um exemplo é o tamanho de lote. Foram utilizadas 352 imagens em lote durante a validação cruzada e esse número não se alterou quando foi utilizado o conjunto completo de treinamento. Este fato pode ter atrapalhado a convergência do modelo.

Observa-se que a curva ROC para o modelo final (Figura 4.23) se assemelha às dos outros modelos da validação cruzada, com uma pequena perda de performance que é possível constatar visualmente pela "compressão" da curva na porção esquerda superior da figura. Para se tomar conhecimento da confiança do modelo em suas predições, é necessário observar a distribuição das suas probabilidades para cada uma das classes. A Figura 4.24 a seguir apresenta as estimativas de densidade por *kernel* das distribuições das probabilidades atribuídas aos itens das classes *FAKE* e *REAL* dos itens de testes do conjunto de treinamento completo. A estimativa de densidade por *kernel* é apenas um histograma normalizado que suaviza a distribuição utilizando um kernel gaussiano.

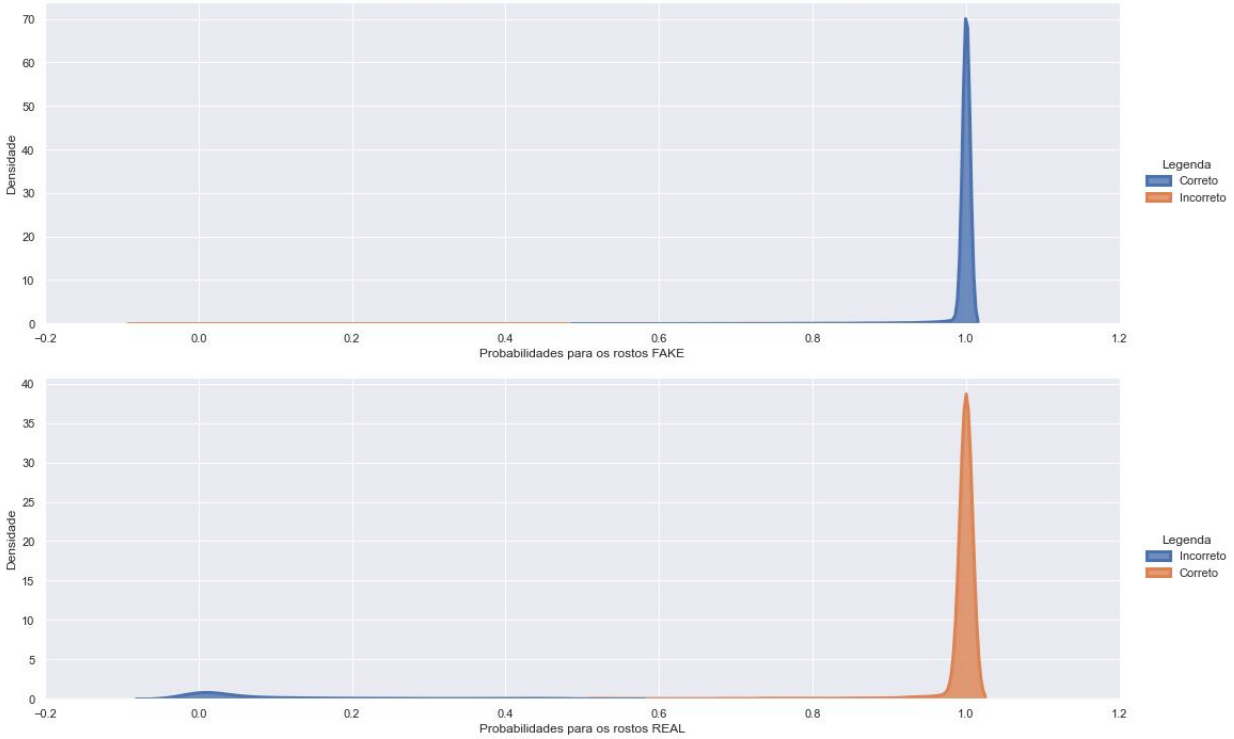


Figura 4.24: Estimativas de densidade por kernel gaussiano das distribuições das probabilidades resultantes do modelo para as classes *REAL* e *FAKE*.

Pela distribuição, é possível observar que o modelo é significativamente confiante em suas predições. O elevado pico no valor de probabilidade próximo de 1 indica que a maioria das imagens foram preditas com probabilidade quase máxima para suas respectivas classes corretas. Observa-se que, no caso das classes de rostos reais, existe um pequeno pico na região de 0. A Figura 4.25 abaixo apresenta os mesmos gráficos da Figura 4.24 anterior porém ampliada no intervalo de 0 a 1 no eixo de densidade.

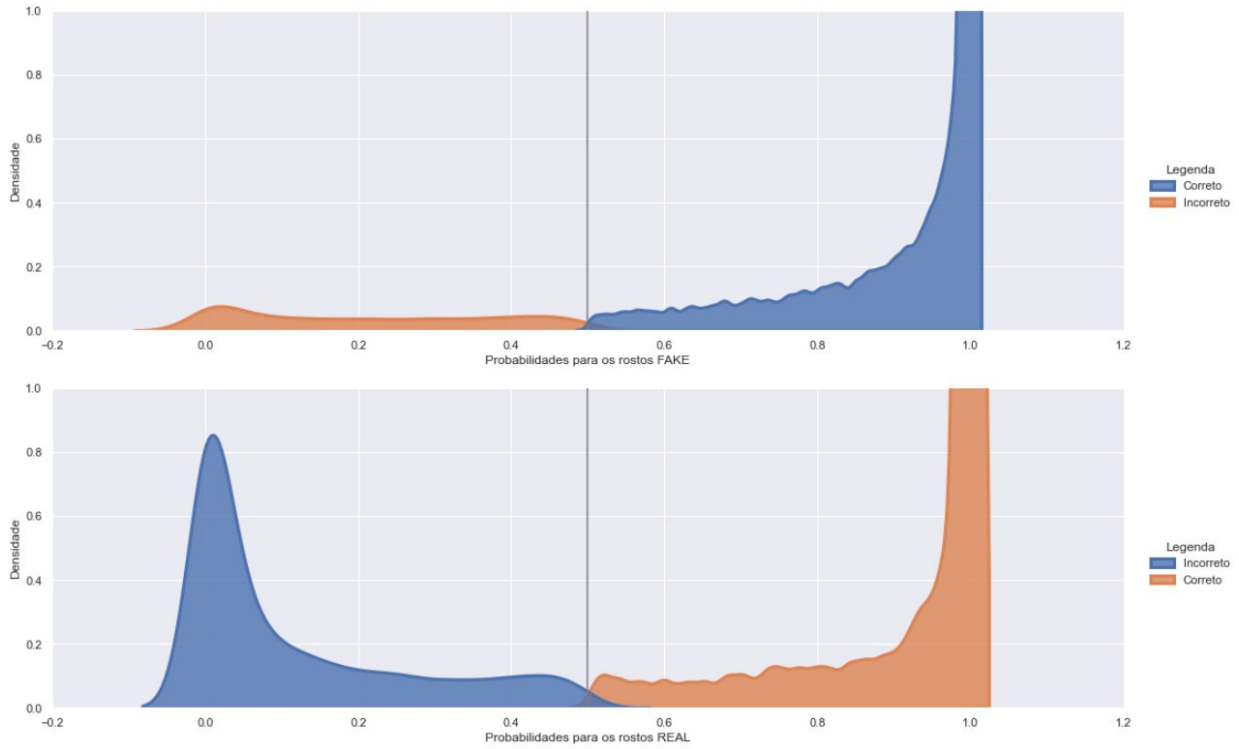


Figura 4.25: Estimativas de densidade por kernel gaussiano das distribuições das probabilidades resultantes do modelo para as classes *REAL* e *FAKE* com ampliação no intervalo de 0 a 1 no eixo de densidade.

Observa-se que, quando o modelo erra a classe *REAL*, uma parcela significativa dos erros é predita com extrema confiança para a classe oposta. Isso pode ser observado pelo pico em 0 do gráfico inferior. Isso significa que existem imagens reais no conjunto de dados de teste que o modelo apresentou elevada confiança de que são falsas. Essas imagens podem estar no espectro daquelas discutidas na Seção 4.5, isto é, imagens em desfoque de movimento ou imagens ruidosas (mãos, pescoços) que foram extraídas de um vídeo cuja classe era *REAL* e que, do ponto de vista de imagens estáticas, aparentam terem sofrido manipulações.

4.7 Etapa Final de Validação

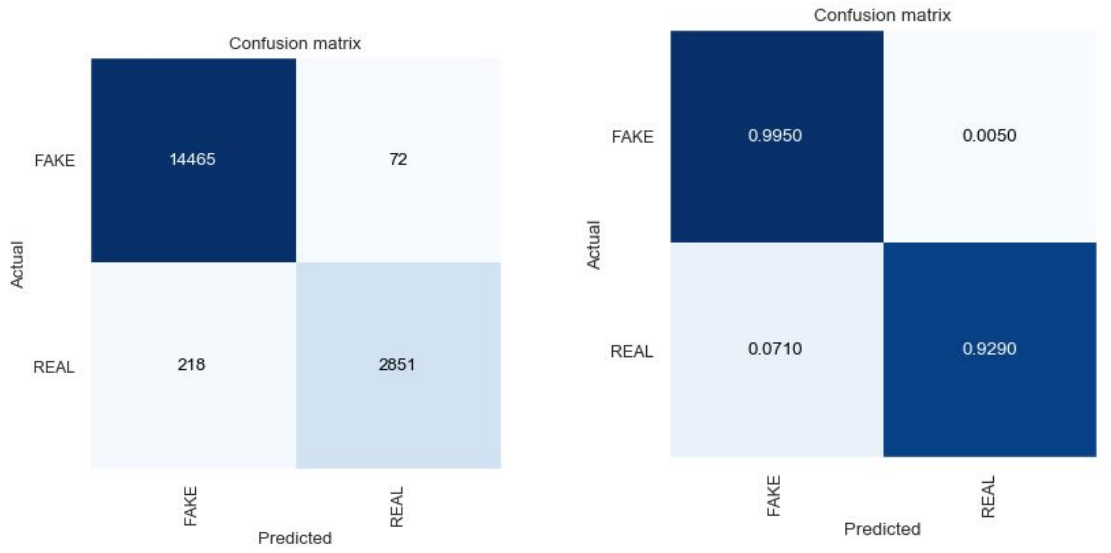
Como foi mencionado na Seção 3.3, a proposta de classificação de vídeos faz uso de uma etapa adicional. Até agora os resultados foram para imagens independentes do conjunto de imagens de rostos extraídos pela MTCNN. A predição por vídeo segue o processo descrito na Figura 3.10,

onde a classe que um vídeo recebe é uma função da quantidade de imagens preditas falsas e reais. A equação 4.4 abaixo explicita isso:

$$\text{Classe do vídeo} = \begin{cases} FAKE, & \text{se } Q_{FAKE} \geq \rho \times Q_{REAL} \\ REAL, & \text{caso contrário} \end{cases}, \tag{4.4}$$

onde Q_{FAKE} é a quantidade de imagens preditas $FAKE$ e Q_{REAL} a quantidade de imagens preditas $REAL$ para um vídeo específico.

As imagens presentes no diretório *train* foram então agrupadas pelo nome do vídeo as quais se originaram. Isso é possível utilizando o nome das imagens, que seguem os padrões descritos na Seção 3.1.1. Com as imagens agrupadas por vídeo, realizou-se a inferência do mesmo modelo obtido do conjunto de treinamento completo nos agrupamentos de imagens de cada vídeo, e para cada um deles a quantidade de cada classe obtida foi contabilizada. São ao total 17.606 vídeos resultantes no diretório *test*. A Figura 4.26 apresenta as matrizes de confusão original e normalizada resultantes desse processo utilizando $\rho = 1$.



(a) (b)

Figura 4.26: Matrizes de confusão (a) absoluta e (b) normalizada resultantes para a inferência do modelo nos agrupamentos de imagens específicos de cada um dos 17.606 vídeo do diretório *test*, onde a classe predita para o vídeo segue o processo descrito na Figura 3.10. Resultados para $\rho = 1$

Temos que, dos 14.537 vídeos falsos, o modelo errou apenas 72, e dos 3.069 vídeos reais, o modelo errou 218, representando uma taxa de acerto de mais de 92% em ambos os casos. Era de se esperar um aumento nesse caso pois utilizando o processo descrito na Figura 3.10 estamos fornecendo um meio para que erros de imagens individuais possam ser suprimidos por uma taxa geral maior de acertos das outras imagens de um mesmo vídeo.

A fim de se entender o impacto do coeficiente ρ nessas métricas, o valor de ρ foi variado no

intervalo $[0, 25, 4]$ com passadas de $0, 25$, resultando em uma avaliação para 16 diferentes valores de ρ. A Figura 4.27 a seguir apresenta o resultado desses 16 diferentes valores de ρ versus a taxa de falsa rejeição (TFR) e a taxa de falsa aceitação (TFA) de um mesmo modelo no conjunto de imagens agrupadas por vídeo do diretório *test*. Neste caso, a taxa de falsa rejeição simboliza a parcela de rostos reais que foram incorretamente classificados como falsos e a taxa de falsa aceitação simboliza a parcela de rostos falsos que foram incorretamente classificados como reais. Os valores para TFA e TFA podem ser obtidos pelas equações 4.5 e 4.6 abaixo:

$$TFR = \frac{fp}{tn + fp}, \tag{4.5}$$

$$TFA = \frac{fn}{tp + fn}. \tag{4.6}$$

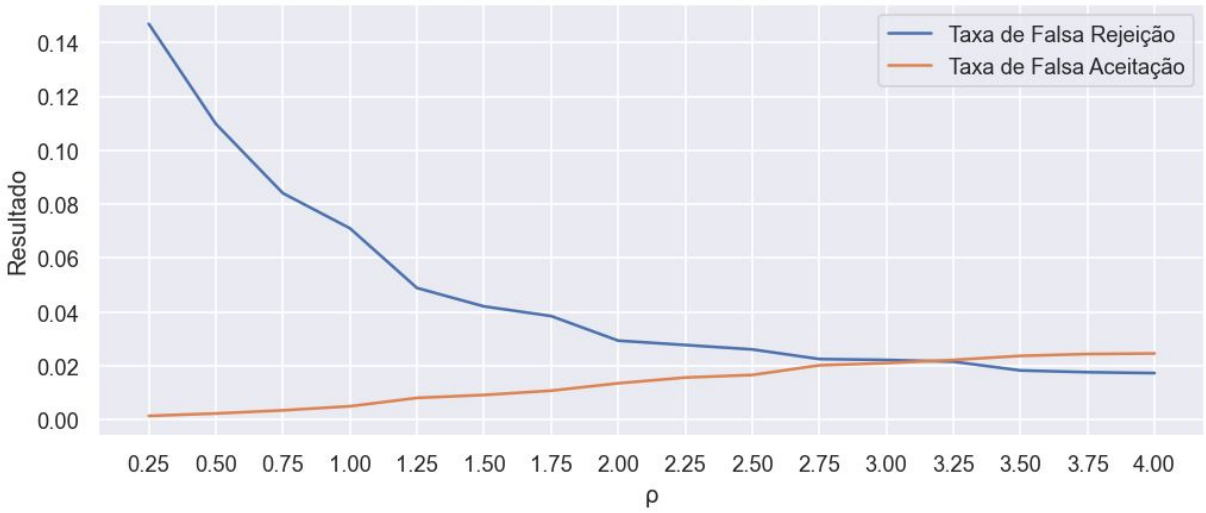
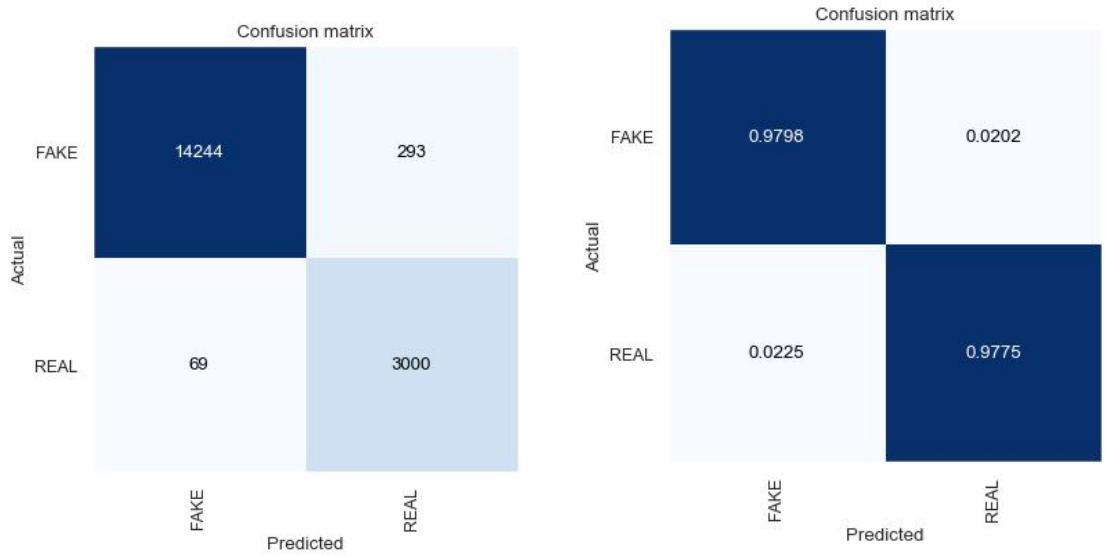


Figura 4.27: Diferentes valores de ρ versus as taxas de verdadeiro positivo e verdadeiro negativo do modelo inferido nos agrupamentos de imagens do diretório *test*.

Observa-se um comportamento interessante. Valores maiores de ρ não comprometem significativamente a taxa de falsa aceitação porém diminuem consideravelmente a taxa de falsa rejeição. Eventualmente, existe um valor para ρ que a taxa de falsa aceitação, inicialmente menor, supera o valor da taxa de falsa rejeição. Esse ponto de cruzamento é onde ocorre uma taxa de erro igual ou EER (*equal error rate*) do modelo. A Figura 4.28 a seguir apresenta as matrizes de confusão original e normalizada para o valor de $\rho = 2, 75$ que, de acordo com a Figura 4.27, é visualmente um local onde as taxas de falsa aceitação e falsa rejeição estão próximas.



(a) (b)

Figura 4.28: Matrizes de confusão (a) absoluta e (b) normalizada resultantes para a inferência do modelo nos agrupamentos de imagens específicos de cada um dos 17.606 vídeo do diretório *test*, onde a classe predita para o vídeo segue o processo descrito na Figura 3.10. Resultados para $\rho = 2,75$.

Embora a proposta do modelo seja o conservadorismo, a perda ínfima de performance na taxa de verdadeiros positivos possa ser justificada pelo aumento de produtividade no acerto de vídeos reais com maior frequência pelo modelo. Em outras ocasiões o modelo talvez não necessite ser tão conservador e a melhor proposta passa a ser o melhor equilíbrio entre as partes. Para o valor de $\rho = 2,75$, temos um modelo que acerta ambas as classes com mais de 97% de chances.

Um valor de $\rho = 2,75$ significa que para um vídeo ser considerado *FAKE* é necessário que a quantidade de imagens preditas como *REAL* nele seja no mínimo $2,75$ vezes menor que a quantidade de imagens falsas. Para um vídeo de 10 segundos, com amostragem de 30 em 30 frames (aproximadamente de 1 em 1 segundo), serão aproximadamente 10 imagens por vídeo. Dessas 10 imagens, a faixa de valor possível para que o vídeo seja considerado *REAL* é $Q_{REAL} \geq 3$, ou seja, 3 imagens ou mais preditas como *REAL* já são suficientes para que o vídeo seja considerado *REAL*. A rápida queda na taxa de falsa rejeição sem um forte prejuízo na taxa de falsa aceitação significa que, no geral, o peso por encontrar rostos reais é maior do que encontrar rostos falsos. Logo, encontrar rostos reais por si só é um forte indício de que o vídeo possa provavelmente ser real e não um *deepfake* (considerando 10 imagens por vídeo como é aproximadamente o caso para o conjunto de dados utilizado).

É importante destacar que para a utilização do método de averiguação utilizado em um vídeo exterior ao conjunto de dados é necessário que apenas o trecho do vídeo o qual se acredita ser um *deepfake* deve ser fornecido ao modelo. Uma vez que o processo de detecção é realizado em toda a extensão do vídeo, inserir trechos do vídeo que contenham rostos reais (supondo o

conhecimento prévio) pode criar uma quantidade elevada de detecções de classes reais, o que pode levar a quantidade de detecções de classe *FAKE* ser negligenciada completamente.

4.8 Tempo de Inferência para um Novo Vídeo

O tempo de inferência para um novo vídeo caracteriza-se pelo tempo necessário aproximado que o programa leva para executar desde a etapa de extração dos rostos de um vídeo até a etapa de predição das imagens pelo modelo, contagem dos resultados e geração do valor final utilizando a equação 4.4. Isso é popularmente conhecido como uma solução *end-to-end*, onde uma solução completa para o problema inicial proposto é fornecida. Neste caso, um vídeo contendo ou não um *deepfake* entra na solução e o resultado final é apenas a predição do vídeo como *FAKE* ou *REAL*.

Aqui não há interesse em saber se o modelo acerta ou não a classe correta do vídeo, mas sim o tempo que o processo completo leva. Como as predições estarão sendo feitas individualmente por vídeo e não há processo de treinamento, o uso da GPU não se faz significativamente necessário, portanto, os tempos calculados nessa etapa se referem às computações realizadas exclusivamente na CPU (checar tabela 4.1 para conhecimento do hardware utilizado).

A tabela 4.8 a seguir apresenta os tempos de processamentos da solução *end-to-end* para 10 vídeos diferentes do diretório "*dfdc_train_part_0*".

Nome do Vídeo	Quantidade de Faces Extraídas	Tempo de Processamento Total (s)	Tempo Médio de Processamento por Face (ms)
aaqaifqrwn.mp4	10	2,826	282,6
aayrffkzxn.mp4	10	2,963	296,3
abhggqdift.mp4	10	2,853	285,3
acagallncj.mp4	10	2,735	273,5
acdkfksyev.mp4	10	2,844	284,4
achdeirhym.mp4	10	2,913	291,3
acryukkddn.mp4	10	2,876	287,6
adftmlvzfs.mp4	10	3,060	306,0
aegvtnwqoe.mp4	10	2,887	288,7
aejvkfbtxs.mp4	10	2,780	278,0
Média	10	2,874	287,4
Desvio Padrão	0	0,087	8,7

Tabela 4.8: Tempos de processamento de uma solução end-to-end para 10 vídeos diferentes do diretório "*dfdc_train_part_0*". Todos os vídeos em questão apresentam 10 segundos de duração. As extrações das faces foram realizadas de 30 em 30 quadros (aproximadamente de 1 em 1 segundo). As faces foram extraídas sequencialmente pela MTCNN e, após completo o processo, o modelo realizava as predições sequencialmente em cada uma das imagens.

Os tempos aqui indicam o processo de extração de faces feito sequencialmente pela MTCNN e posteriormente as predições feitas sequencialmente pelo modelo treinado. Observa-se que o tempo médio de processamento de um vídeo de 10 segundos extraindo as faces a cada 30 quadros (aproximadamente a cada 1 segundo) e realizando a predição para todas as faces extraídas é de 2,874 segundos. Este é um tempo de inferência viável para esta solução ser utilizada em escala atualmente uma vez que a presença de *deepfakes* não é tão recorrente. Para um possível ganho de performance é recomendada a utilização de uma GPU.

Capítulo 5

Conclusões

Neste trabalho foi explorada a detecção de *deepfakes* em vídeos utilizando redes neurais convolucionais profundas, o que no final se mostrou uma solução viável para o problema em questão e o conjunto de dados utilizado.

O conjunto de dados original *Deepfake Detection Challenge*, com 118.145 vídeos, foi processado por uma rede convolucional multitarefas em cascata, extraindo os rostos detectados de todos os indivíduos de um vídeo de 30 em 30 quadros. A geração do novo conjunto de dados de faces, com 1.164.203 imagens, se deu de forma consistente na maioria dos vídeos disponíveis, apresentando baixa presença de ruídos, isto é, imagens que não são rostos, no conjunto de imagens final obtido. Foi possível observar a influência de algumas dessas imagens de ruído nas predições finais realizadas, gerando um alto custo (confusão) para o modelo, mas que por estarem em baixa presença não impactaram significativamente o processo de treinamento.

A estratégia de treinamento do modelo utilizando transferência de aprendizado, treinamento em meia precisão e valores elevados para o tamanho de lote e taxa de aprendizado se mostraram eficientes em conjunto com a política de um ciclo para a taxa de aprendizado. A utilização do teste de faixa da taxa de aprendizado também foi uma estratégia excelente para a escolha da taxa de aprendizado inicial máxima do modelo, convergindo em um valor semelhante para todos os casos.

A separação do conjunto completo de imagens em 5 partições para validação cruzada estratificada foi realizado com sucesso. O treinamento completo de cada partição levou em média 9 horas, resultando em aproximadamente 45 horas de treinamento para a etapa completa de validação cruzada. O treinamento demonstrou que o conjunto de dados possui pouca variabilidade e o modelo Resnet18 escolhido apresentou satisfatório desempenho durante todo o processo, apresentando uma acurácia média de 96,49% e com uma média de 0,8782 para o coeficiente de correlação de matthews, indicando que o modelo foi capaz de balancear relativamente bem as taxas de verdadeiros positivos e verdeiros negativos. Os modelos da validação cruzada produziram uma área sob a curva ROC de média 0,9914, o que é considerada uma excelente capacidade de separação de classes do modelo. Todas as métricas apresentaram desvio padrão significativamente baixo, comprovando a representatividade das diferentes partições da validação cruzada.

O treinamento final no conjunto de imagens separado inicialmente para testes ocorreu nos conformes porém apresentou uma degradação pequena das métricas em relação às esperadas pela validação cruzada, onde algumas possíveis causas para tal comportamento podem estar no viés do conjunto de dados original por não apresentar apenas rostos trocados mas também a voz dos indivíduos trocada e também na otimização dos hiperparâmetros uma vez que o conjunto de treinamento se tornou maior e os hiperparâmetros não foram ajustados de acordo. Embora as métricas tenham se degradado, a degradação foi mínima, mantendo ainda uma acurácia média de 95,85% e um valor médio para o coeficiente de correlação de matthews de 0,8559.

A etapa de testes finais, onde se consideravam todas as imagens de rostos extraídos de um único vídeo como ferramenta para se classificar o vídeo em si, ocorreu como o previsto, produzindo ótimos resultados. O ajuste do coeficiente de liberdade ρ que controla o nível de influência da quantidade de classes reais levou a um valor que prioriza uma peso maior para os rostos preditos como reais. O valor, que se encontra em torno de $2,75$, mostrou que é possível manter uma taxa de erro igual entre as classes e obter mais de 97% de acerto em cada uma das classes diferentes de vídeos.

Com a conclusão deste trabalho, obtém-se então uma rede neural convolucional treinada em um amplo conjunto de vídeos reais e vídeos *deepfakes* que, aplicada utilizando uma estratégia específica, é capaz de classificar aproximadamente 49 a cada 50 vídeos de forma correta. Demonstrou-se por fim que o tempo de inferência de uma solução *end-to-end* onde a entrada é um vídeo e a saída é a classe predita para aquele vídeo viabiliza a utilização desta solução em escala.

5.1 Perspectivas Futuras

O processo realizado neste trabalho é apenas uma de muitas abordagens disponíveis na detecção de *deepfakes*. Existem diversas melhorias e modificações possíveis de serem feitas.

Os hiperparâmetros utilizados não foram tão bem explorados quanto poderiam, uma vez que a carga computacional do problema inviabilizou muita exploração na área. Otimização de hiperparâmetros em aprendizado de máquina profundo é um campo amplo e continuamente estudado, com diversas propostas disponíveis e possíveis de serem aplicadas neste trabalho para solução de problemas como convergência do modelo, redução de tempo de treinamento. Uma interessante abordagem seria explorar diferentes parâmetros para regularização. A Seção 4.6, mais especificamente na Figura 4.25, observa-se que o modelo tende a apresentar elevada confiança para a classe incorreta quando um erro de classificação ocorre. Regularização é exatamente uma das abordagens utilizadas para reduzir a confiança do modelo e permiti-lo trabalhar em intervalos de menor confiança, que acabam sendo interessantes para abordagens posteriores que utilizem alguma filtragem.

Alguns dos problemas relatados durante a etapa de análise dos resultados (Capítulo 4) foram decorrentes de ruídos resultantes da utilização da MTCNN para obter o conjunto de dados de imagens. Como mais de 100 mil vídeos estão presentes no conjunto original do DFDC, recortar manualmente os rostos se mostra uma tarefa árdua dificilmente alcançável sem acesso a muitos

recursos humanos. A estratégia de utilização da MTCNN se mostrou consistente porém se faz necessária uma segunda etapa de correção de erros, onde um segundo algoritmo diferente da MTCNN possa filtrar as imagens obtidas por ela de forma a comprovar ou não a existência de um rosto na imagem. Acredita-se que a extração desses ruídos será benéfico para todo o processo de treinamento do modelo.

Foi também averiguado que o algoritmo de *deepfakes*, em alguns casos, não é capaz de substituir todos os rostos encontrados no vídeo por um rosto forjado, acarretando momentos do vídeo em que o rosto permanece inalterado e a MTCNN o extrai com a classe de *deepfake*, pois esta vem diretamente do vídeo e não da imagem em si. Uma segunda etapa de filtragem aqui também seria benéfica para o modelo.

Observando a fundo o conjunto de dados DFDC, mais especificamente os vídeos dentro dele que constituem *deepfakes*, observa-se que vários rostos apresentam inconsistências facilmente perceptíveis por um humano. Outros, no entanto, são melhores trabalhados e apresentam maior dificuldade de serem percebidos. O estudo não faz discriminação quanto à qualidade do *deepfake* em si. Acredita-se ser um grande acréscimo o estudo do quão bem a rede é capaz de detectar *deepfakes* dados como “bem produzidos“ ou “mal produzidos“, uma vez que um perigo nato deles esteja na qualidade de produção do rosto forjado.

Como foi mencionado no Capítulo 4, *deepfakes* são uma classe de detecção que carrega informação no aspecto temporal. Casos em que a extração do rosto de um *deepfake* ocorreu durante um quadro em que o algoritmo de *deepfake* não havia alterado o rosto ainda receberão a classe *FAKE*, pois esta vem do vídeo original e não dos quadros individuais do vídeo. Isso será interpretado no conjunto de dados como um ruído. Um algoritmo que seja treinado no aspecto temporal dos vídeos pode vir a aprender esses padrões e utilizá-los de forma benéfica, entendendo inconsistências entre quadros seguidos de um *deepfake*. Acredita-se que a utilização de arquiteturas ou unidades que processem informações sequenciais possa ser um grande acréscimo a este estudo. Arquiteturas como o *transformer* [54] são recentes e aproveitam o processamento paralelo das GPUs modernas sem as desvantagens que arquiteturas recorrentes como as RNNs e as redes LSTMs possuem em sua necessidade de processamento sequencial.

Por último, é importante ressaltar que métodos que utilizam aprendizado profundo muitas vezes apresentem o problema de interpretabilidade. Muitos dos modelos pertencentes a esse conjunto são retratados como "caixas pretas", onde não entende-se como o modelo chega ao resultado final, apenas aceita-se que seu resultado é acurado e preciso. Felizmente, as CNNs possuem entradas (imagens) que são visualmente interpretáveis por humanos, então existem várias técnicas para entender como funcionam, o que aprendem e por que funcionam de uma determinada maneira. Neste trabalho, não aprofundou-se na tarefa de interpretação do modelo final, isto é, entender que características ou que regiões da figura provocaram ativações mais significativas do modelo. O trabalho em [80] apresenta um bom ponto de partida para estudos futuros nessa área.

Finalizando, externa-se a esperança de que este trabalho possa cumprir sua missão e seus objetivos, contribuindo para um mundo onde a segurança e a privacidade das pessoas seja sempre uma prioridade de manutenção e de melhorias.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011.

[3] HE, K. et al. *Deep Residual Learning for Image Recognition*. 2015.

[4] ZEILER, M. D.; FERGUS, R. *Visualizing and Understanding Convolutional Networks*. 2013.

[5] RöSSLER, A. et al. *FaceForensics++: Learning to Detect Manipulated Facial Images*. 2019.

[6] DOLHANSKY, B. et al. The deepfake detection challenge dataset. *ArXiv*, abs/2006.07397, 2020.

[7] KORSHUNOV, P.; MARCEL, S. *DeepFakes: a New Threat to Face Recognition? Assessment and Detection*. 2018.

[8] LI, Y. et al. Celeb-df: A large-scale challenging dataset for deepfake forensics. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 3204–3213, 2020.

[9] ZHANG, K. et al. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, v. 23, n. 10, p. 1499–1503, Oct 2016. ISSN 1070-9908.

[10] CASTELLS, M. et al. *Change: 19 Key Essays on How Internet Is Changing Our Lives*. [S.l.]: Turner House Publications, 2014. ISBN 8415832451.

[11] VACCARI, C.; CHADWICK, A. Deepfakes and disinformation: Exploring the impact of synthetic political video on deception, uncertainty, and trust in news. *Social Media + Society*, v. 6, n. 1, p. 2056305120903408, 2020. Disponível em: <https://doi.org/10.1177/2056305120903408>.

[12] RIBEIRO, T. *Mark Zuckerberg aparece em vídeo manipulado com técnica deepfake*. `https://www.showmetech.com.br/mark-zuckerberg-aparece-em-video-manipulado-com-tecnica-deepfake/`. Acessado: 2021-05-01.

[13] NEWMAN, E. et al. Truthiness and falsiness of trivia claims depend on judgmental contexts. *Journal of experimental psychology. Learning, memory, and cognition*, v. 41, 03 2015.

[14] STENBERG, G. Conceptual and perceptual factors in the picture superiority effect. *European Journal of Cognitive Psychology*, v. 18, p. 813–847, 11 2006.

[15] DSOUZA, J. *What is a GPU and do you need one in Deep Learning?* `https://towardsdatascience.com/what-is-a-gpu-and-do-you-need-one-in-deep-learning-718b9597aa0d`. Acessado: 2021-05-01.

[16] RISDAL, M. *How Kaggle Makes GPUs Accessible to 5 Million Data Scientists.* `https://developer.nvidia.com/blog/how-kaggle-makes-gpus-accessible-to-5-million-data-scientists/`. Acessado: 2021-05-01.

[17] PEROV, I. et al. *DeepFaceLab: A simple, flexible and extensible face swapping framework.* 2020.

[18] LENAT, D. B.; GUHA, R. V. *Building Large Knowledge-Based Systems; Representation and Inference in the Cyc Project.* 1st. ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201517523.

[19] MOR-YOSEF, S. et al. Ranking the risk factors for cesarean: Logistic regression analysis of a nationwide study. *Obstetrics Gynecology*, v. 75, p. 944–947, 1990.

[20] FISHER, R. A. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, v. 7, n. 2, p. 179–188, 1936. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-1809.1936.tb02137.x>.

[21] ANDERSON, E. The species problem in iris. *Annals of the Missouri Botanical Garden*, St. Louis :Missouri Botanical Garden Press,1914-, v. 23, p. 457–509, 1936. Disponível em: <https://www.biodiversitylibrary.org/part/4079>.

[22] BOJAR, O. et al. Findings of the 2014 workshop on statistical machine translation. In: *Proceedings of the ninth workshop on statistical machine translation.* [S.l.: s.n.], 2014. p. 12–58.

[23] ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org/>.

[24] PASZKE, A. et al. Pytorch: An imperative style, high-performance deep learning library. In: WALLACH, H. et al. (Ed.). *Advances in Neural Information Processing Systems 32.* Curran Associates, Inc., 2019. p. 8024–8035. Disponível em: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.

[25] ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958.

[26] HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, v. 2, n. 5, p. 359–366, 1989. ISSN 0893-6080. Disponível em: <https://www.sciencedirect.com/science/article/pii/0893608089900208>.

[27] RAMACHANDRAN, P.; ZOPH, B.; LE, Q. V. Swish: a self-gated activation function. *arXiv: Neural and Evolutionary Computing*, 2017.

[28] MISRA, D. Mish: A self regularized non-monotonic neural activation function. *arXiv preprint arXiv:1908.08681*, CoRR, v. 4, 2019.

[29] WILSON, D.; MARTINEZ, T. R. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, v. 16, n. 10, p. 1429–1451, 2003. ISSN 0893-6080. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0893608003001382>.

[30] KRIZHEVSKY, A. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

[31] SMITH, S.; LE, Q. Understanding generalization and stochastic gradient descent. 10 2017.

[32] JASTRZęBSKI, S. et al. *Residual Connections Encourage Iterative Inference*. 2018.

[33] XING, C. et al. *A Walk with SGD*. 2018.

[34] SMITH, L. N. *A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay*. 2018.

[35] SMITH, L. N. *Cyclical Learning Rates for Training Neural Networks*. 2017.

[36] RUDER, S. *An overview of gradient descent optimization algorithms*. 2017.

[37] QIAN, N. On the momentum term in gradient descent learning algorithms. *Neural Networks*, v. 12, n. 1, p. 145–151, 1999. ISSN 0893-6080. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0893608098001166>.

[38] NESTEROV, Y. A method for unconstrained convex minimization problem with the rate of convergence o(1/k^2).*In* : .[*S.l.* : *s.n.*], 1983.

[39] BENGIO, Y.; BOULANGER-LEWANDOWSKI, N.; PASCANU, R. *Advances in Optimizing Recurrent Networks*. 2012.

[40] DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, v. 12, n. 7, 2011.

[41] ZEILER, M. D. *ADADELTA: An Adaptive Learning Rate Method*. 2012.

[42] KINGMA, D. P.; BA, J. *Adam: A Method for Stochastic Optimization*. 2017.

[43] ZHUANG, J. et al. *AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients*. 2020.

[44] TIBSHIRANI, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, [Royal Statistical Society, Wiley], v. 58, n. 1, p. 267–288, 1996. ISSN 00359246. Disponível em: <http://www.jstor.org/stable/2346178>.

[45] SRIVASTAVA, N. et al. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, v. 15, n. 56, p. 1929–1958, 2014. Disponível em: <http://jmlr.org/papers/v15/srivastava14a.html>.

[46] Le Cun, Y. et al. Handwritten digit recognition: applications of neural network chips and automatic learning. *IEEE Communications Magazine*, v. 27, n. 11, p. 41–46, 1989.

[47] ZHOU, Y.; CHELLAPPA, R. Computation of optical flow using a neural network. *IEEE 1988 International Conference on Neural Networks*, p. 71–78 vol.2, 1988.

[48] TAN, M.; LE, Q. V. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020.

[49] DOSOVITSKIY, A. et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020.

[50] REN, S. et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016.

[51] BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020.

[52] TAN, M.; PANG, R.; LE, Q. V. *EfficientDet: Scalable and Efficient Object Detection*. 2020.

[53] MERITY, S. *Single Headed Attention RNN: Stop Thinking With Your Head*. 2019.

[54] VASWANI, A. et al. *Attention Is All You Need*. 2017.

[55] DEVLIN, J. et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019.

[56] BROWN, T. B. et al. *Language Models are Few-Shot Learners*. 2020.

[57] ARIK, S. O.; PFISTER, T. *TabNet: Attentive Interpretable Tabular Learning*. 2020.

[58] LIM, B. et al. *Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting*. 2020.

[59] Deng, J. et al. Imagenet: A large-scale hierarchical image database. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2009. p. 248–255.

[60] RUDER, S. *Transfer Learning - Machine Learning's Next Frontier*. 2017. `http://ruder.io/transfer-learning/`.

[61] CHICCO, D.; JURMAN, G. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, v. 21, 01 2020.

[62] MATTHEWS, B. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, v. 405, n. 2, p. 442–451, 1975. ISSN 0005-2795. Disponível em: <https://www.sciencedirect.com/science/article/pii/0005279575901099>.

[63] ZOLLHÖFER, M. et al. State of the art on monocular 3d face reconstruction, tracking, and applications. *Computer Graphics Forum*, v. 37, 2018.

[64] FRITH, C. Role of facial expressions in social interactions. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, v. 364, p. 3453–8, 12 2009.

[65] THIES, J. et al. Face2face: Real-time face capture and reenactment of rgb videos. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 62, n. 1, p. 96–104, dez. 2018. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/3292039>.

[66] SUWAJANAKORN, S.; SEITZ, S. M.; KEMELMACHER-SHLIZERMAN, I. Synthesizing obama: Learning lip sync from audio. *ACM Trans. Graph.*, Association for Computing Machinery, New York, NY, USA, v. 36, n. 4, jul. 2017. ISSN 0730-0301. Disponível em: <https://doi.org/10.1145/3072959.3073640>.

[67] FLORIDI, L. Artificial intelligence, deepfakes and a future of ectypes. *Philosophy Technology*, v. 31, 08 2018.

[68] GOODFELLOW, I. J. et al. *Generative Adversarial Networks*. 2014.

[69] HINTON, G. E.; SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. *Science*, American Association for the Advancement of Science, v. 313, n. 5786, p. 504–507, 2006. ISSN 0036-8075. Disponível em: <https://science.sciencemag.org/content/313/5786/504>.

[70] Matern, F.; Riess, C.; Stamminger, M. Exploiting visual artifacts to expose deepfakes and face manipulations. In: *2019 IEEE Winter Applications of Computer Vision Workshops (WACVW)*. [S.l.: s.n.], 2019. p. 83–92.

[71] YANG, X.; LI, Y.; LYU, S. *Exposing Deep Fakes Using Inconsistent Head Poses*. 2018.

[72] AGARWAL, S. et al. Protecting world leaders against deep fakes. In: *CVPR Workshops*. [S.l.: s.n.], 2019.

[73] AFCHAR, D. et al. *MesoNet: a Compact Facial Video Forgery Detection Network*. 2018.

[74] SABIR, E. et al. *Recurrent Convolutional Strategies for Face Manipulation Detection in Videos*. 2019.

[75] BAHDANAU, D.; CHO, K.; BENGIO, Y. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016.

[76] DANG, H. et al. *On the Detection of Digital Face Manipulation*. 2020.

[77] DEEPFAKE Detection Challenge | Kaggle. `https://www.kaggle.com/c/deepfake-detection-challenge`. (Accessed on 03/26/2021).

[78] HOWARD, J.; GUGGER, S. Fastai: A layered api for deep learning. *Information*, MDPI AG, v. 11, n. 2, p. 108, Feb 2020. ISSN 2078-2489. Disponível em: <http://dx.doi.org/10.3390/info11020108>.

[79] SMITH, L. N.; TOPIN, N. *Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates*. 2018.

[80] ZEILER, M. D.; FERGUS, R. *Visualizing and Understanding Convolutional Networks*. 2013.

ANEXOS

I. PROGRAMAS UTILIZADOS

Todos os programas utilizados podem ser encontrados no repositório oficial do trabalho no link abaixo:

`https://github.com/ChristianCFranca/unb-tg-deepfakerecognition`