UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Thiago Finardi Penteado

Reconhecimento do Alfabeto de Linguagem de Sinais em Tempo Real Através de Redes Neurais Convolucionais Profundas

São Carlos

2022

Thiago Finardi Penteado

Reconhecimento do Alfabeto de Linguagem de Sinais em Tempo Real Através de Redes Neurais Convolucionais Profundas

Monografia apresentada ao Curso de Engenharia de Produção, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro de Produção.

Orientador: Profa. Dra. Maíra Martins da Silva

São Carlos

2022

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Penteado, Thiago Finardi
P419r Reconhecimento do Alfabeto de Linguagem de Sinais em Tempo Real Através de Redes Neurais Convolucionais Profundas / Thiago Finardi Penteado; orientador Maíra Martins da Silva. São Carlos, 2022.

Monografia (Graduação em Engenharia de Produção) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2022.

1. Aprendizado de máquina. 2. Redes Neurais. 3. Redes neurais convolucionais. 4. Machine learning. 5. Deep learning. 6. Linguagem de sinais. 7. Visão computacional. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Candidato: Thiago Finardi Penteado
Título do TCC: Reconhecimento do Alfabeto de Linguagem de Sinais em Tempo Real Através de Redes Neurais Convolucionais Profundas
Data de defesa: 15/12/2022

Comissão Julgadora	**Resultado**
Professora Doutora Maíra Martins da Silva (orientador)	APROVADO
Instituição: EESC **-** SEM	
Professor Associado Fernando César Almada Santos	APROVADO
Instituição: EESC **-** SEP	
Professor Associado Valdir Grassi Junior	APROVADO
Instituição: EESC - SEL	

Presidente da Banca: **Professora Doutora Maíra Martins da Silva**

Este trabalho é dedicado à minha família e amigos.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer à minha família, principalmente ao meu pai e minha mãe, por todo o apoio e incentivo dado a minha formação e as minhas escolhas. Vocês sempre foram e sempre serão minhas maiores inspirações.

Também gostaria de agradecer à República Poltergeist por ser a minha segunda família durante estes cinco anos de graduação que, graças a vocês, foram os melhores da minha vida.

Por fim, agradeço à Profa. Dra. Maíra Martins da Silva pelo tempo dedicado a minha orientação no desenvolvimento deste trabalho.

"Nobody ever figures out what life is all about, and it doesn't matter. Explore the world. Nearly everything is really interesting if you go into it deeply enough."

Richard Feynman

RESUMO

PENTEADO, T. F. **Reconhecimento do Alfabeto de Linguagem de Sinais em Tempo Real Através de Redes Neurais Convolucionais Profundas**. 2022. 60p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2022.

A visão computacional está sendo aplicada nas mais diversas áreas, sendo usada desde em aplicações médicas até em veículos autônomos. Ela tem sido implementada através de vários métodos computacionais diferentes como modelos de aprendizado de máquina tradicionais, redes neurais e redes neurais convolucionais profundas. Neste trabalho, a visão computacional visa a interpretação do alfabeto de linguagem de sinais através de redes convolucionais profundas. Foram avaliadas diferentes arquiteturas de redes neurais convolucionais juntamente com técnicas de melhora de desempenho, como *fine-tuning*, *learning rate decay* e transferência de aprendizado a fim de entender o impacto de cada uma dessas técnicas no resultado e estabelecer um modelo com alto desempenho na tarefa. Como resultado, obteve-se um modelo com alto desempenho na tarefa de interpretação do alfabeto de linguagem de sinais, obtendo uma acurácia de 99.97% em treino e 92.40% em teste, e foi constatado que as técnicas de melhora de desempenho analisadas realmente apresentavam um impacto positivo na acurácia da rede. Por fim, o modelo com melhor desempenho foi incorporado junto a um programa de captação de imagens para a interpretação em tempo real da linguagem de sinais que está disponibilizado no repositório oficial do projeto (GitHub).

Palavras-chave: Aprendizado de máquina. Redes Neurais. Redes neurais convolucionais. *Machine learning*. *Deep learning*. Linguagem de sinais. Visão computacional.

ABSTRACT

PENTEADO, T. F. **Real-time Sign Language Alphabet Interpretation with Deep Convolutional Neural Networks**. 2022. 60p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2022.

Computer vision technology is being applied in the most diverse areas, going from medical applications to autonomous vehicles. It has been implemented through various different methods such as traditional machine learning models, neural networks and deep convolutional neural networks. In this project, the application of computation vision aims to interpret the sign language alphabet through deep convolutional neural networks. Different convolutional neural networks architectures were evaluated along with performance-enhancing techniques such as fine-tuning, learning rate decay and transfer learning in order to understand the impact of each of these techniques and establish a model with high performance in the task. As a result, we achieved a model with high performance in interpreting the sign language alphabet, obtaining an accuracy of 99.97% on the training set and 92.40% on the test set and confirming that the performance improvement techniques analyzed had indeed an positive impact on the accuracy of the neural network. Finally, the model with best performance was incorporated into an image capture program for real-time interpretation of the sign language alphabet and is available on the project's official repository (GitHub).

Keywords: Machine learning. Neural Networks. Convolutional neural networks. Deep learning. Sign Language. Computer vision.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A incorporação da linguagem de sinais no cotidiano é de grande importância nos dias atuais. Além da inclusão dos deficientes auditivos nas mais diversas tarefas, Brereton (2008) mostrou que a incorporação de uma forma de KWS (*Key Word Sign*), que combina gestos manuais com a fala para dar suporte a comunicação, em uma classe de educação pré-escolar causou um aumento na participação e apreciação da diversidade por parte das crianças. A incorporação de tecnologias que facilitem cada vez mais a inclusão da linguagem de sinais nos mais diversos ambientes, seja através de uma tradução direta ou no auxílio ao aprendizado, pode ser de grande ajuda.

A visão computacional é a área de inteligência artificial que visa desenvolver a habilidade de um computador em interpretar o meio à sua volta e retirar informações significativas de imagens, vídeos e outros tipos de dados visuais a fim de tomar decisões ou fazer recomendações (BALLARD,). As aplicações dessa tecnologia estão nas mais diversas áreas, como na medicina, auxiliando os profissionais em diagnósticos, como em mamografias (VYBORNY; GIGER, 1994), exames de lesões cranianas (SILVA et al.,), ou até mesmo em veículos autônomos, como o Mars Curiosity Rover, da NASA .(MARS. . . ,)

Para que um sistema de visão computacional atinja uma alta precisão, é necessário um grande conjunto de dados aliado à duas tecnologias: a aprendizagem de máquina profunda e as redes neurais convolucionais (ou *Convolutional Neural Network* - CNN). Uma CNN aplica filtros nos dados visuais através de operações matemáticas chamadas convoluções a fim de manter uma relação de vizinhança entre um pixel e os pixels vizinhos durante o processamento da rede. Com isso, os filtros conseguem detectar padrões na imagem que serão utilizados como *features* dentro do modelo de aprendizado profundo (GU et al., 2018).

Assim, através dessas tecnologias, é possível realizar diversas tarefas ligadas à classificação de imagens. Dentro deste trabalho, a visão computacional aliada com a utilização de redes neurais convolucionais profundas tem como um de seus objetivos a interpretação de linguagem de sinais em tempo real.

1.1 Objetivos

O trabalho tem como objetivos estipulados:

1. Comparação de diferentes arquiteturas de redes neurais convolucionais.

2. Utilização de diferentes técnicas que visam melhorar o desempenho de uma CNN.

3. Desenvolvimento de um modelo de aprendizado de máquina profundo com alta precisão de classificação em dados do alfabeto de linguagem de sinais.

4. Desenvolvimento de um modelo para aquisição e pré-processamento de imagens em tempo real.

5. Utilização da CNN para a classificação do alfabeto de linguagem de sinais em tempo real.

1.2 Estrutura do trabalho

Primeiramente, o trabalho apresenta uma fundamentação teórica que tem como objetivo de apresentar os conceitos utilizados na monografia, desde o funcionamento dos modelos até as métricas utilizadas para avaliação. Após a revisão da teoria, há uma seção de desenvolvimento que descreve o conjunto de dados utilizados na tarefa, o desenvolvimento de diferentes modelos de aprendizado de máquina juntamente com o estudo de técnicas de otimização, uma apresentação e discussão dos resultados e, por fim, a aplicação do melhor modelo na classificação em tempo real. Após o desenvolvimento, há uma seção de conclusão para avaliar se o trabalho conseguiu atingir todos os objetivos estipulados na seção 1.1.

2 FUNDAMENTAÇÃO TEÓRICA

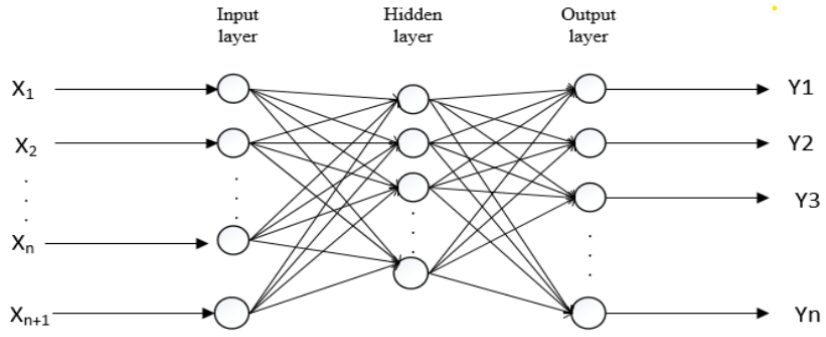
Esse capítulo tem como objetivo a revisão teórica dos conceitos utilizados no trabalho para o embasamento do leitor. Com isso, é descrito nessa seção o conceito de Redes Neurais Artificiais e Redes Neurais Convolucionais, juntamente com o seu funcionamento e técnicas de otimização. Além disso, é abordado conceitos como a transferência de aprendizado, *overfitting*, técnicas de regularização e a descrição das métricas utilizadas no trabalho.

2.1 Redes Neurais Artificiais

Do mesmo modo que em diversas áreas de inteligência artificial, as redes neurais artificiais (RNA) têm sua inspiração baseada na biologia. A representação de uma RNA é comparada com uma maneira simplificada do que se entende do cérebro humano: unidades de processamento, chamadas de neurônios, interconectadas entre si formando uma rede com o objetivo de analisar informações adquiridas (*inputs*).

A Figura 1 apresenta a arquitetura de uma rede neural artificial profunda composta de uma camada de entrada (*input layer*), n camadas de processamento (*hidden layers*) e uma de saída (*output layer*).

Figura 1 – Exemplo de arquitetura de uma Rede Neural Profunda



Fonte: (LI; CHENG, 2019)

Através dos neurônios, que se localizam nas *hidden layers*, é possível, por meio de funções matemáticas baseadas em métodos estatísticos, retirar informações significativas dos *inputs* para a interpretação dos dados. Cada neurônio utilizado em uma RNA, que tem seu modelo demonstrado na Figura 2, de acordo com (HAYKIN, 2001), consiste de:

1. Um conjunto de sinapses com um peso específico para cada uma das entradas, ou seja, um *input* x_j na sinapse j conectada ao neurônio k é multiplicado pelo peso

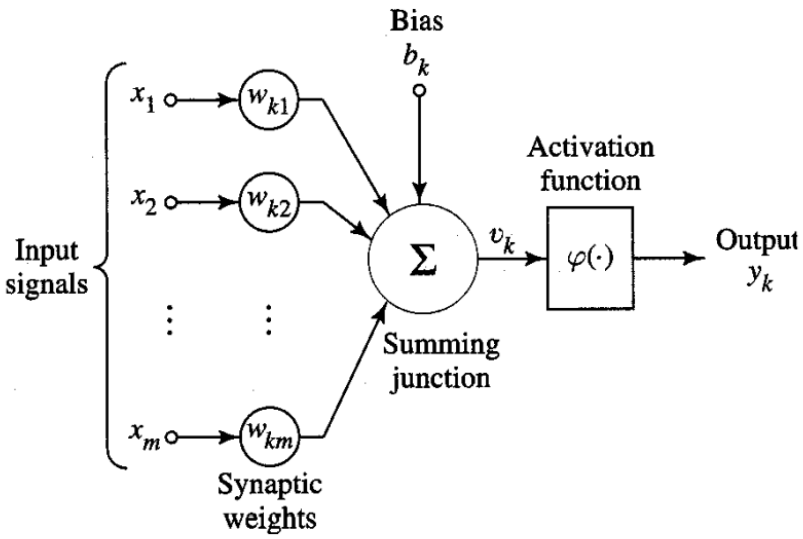
sináptico w_{kj}, podendo assumir tanto valores positivos quanto negativos;

2. Um combinador linear que soma os sinais de entradas ponderados pelos respectivos pesos de cada uma das sinapses do neurônio;

3. Uma função de ativação que interpretará como a combinação linear de entrada no neurônio é passada adiante na rede. A função de ativação limita a faixa de amplitude permitida do sinal de saída a algum valor finito.

Além disso, o modelo descrito na Figura 2 inclui um *bias*, denotado por b_k que tem o efeito de aumentar ou diminuir o argumento da função de ativação.

Figura 2 – Modelo não-linear de um neurônio



Fonte: (HAYKIN, 2001).

Assim, temos que um neurônio k é descrito matematicamente pelas equações:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.1}$$

e

$$y_k = \varphi(u_k + b_k) \tag{2.2}$$

onde:

$x_1, x_2, ..., x_m$ são os *inputs* do neurônio;

$w_{k1}, w_{k2}, ..., w_{km}$ são os pesos sinápticos do neurônio k;

u_k é a combinação linear baseada nos *inputs* e seus respectivos pesos;

b_k é o *bias* e

φ é a função de ativação e

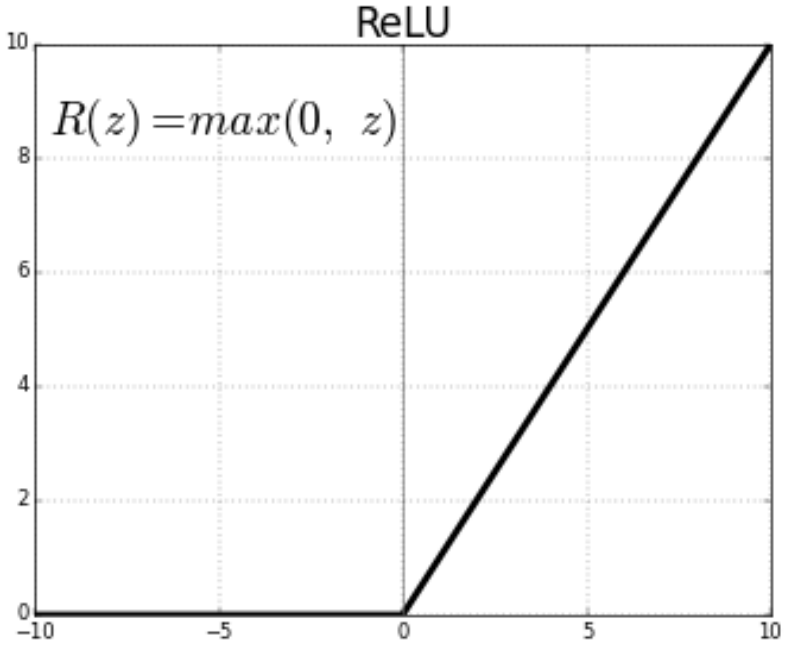
y_k é o *output* do neurônio

2.1.1 Funções de ativação utilizadas na tarefa

Duas funções de ativação foram usadas no trabalho: a *Rectified linear unit* (ReLU) e a *Softmax*. A função de ativação ReLU, que é representada matematicamente pela equação (2.3), retorna 0 para todos os valores negativos mantendo a parte positiva com os próprios valores (SHARMA; SHARMA; ATHAIYA, 2017). Com isso, ela retorna valores no intervalo [0, ∞[, sendo representado pelo gráfico descrito na Figura 3.

$$ReLU(z) = max(0, z) \tag{2.3}$$

Figura 3 – Função de ativação ReLU



Fonte: Sarkar, Kanchan. 2018

A função de ativação *softmax*, que é representada matematicamente pela equação 2.4, é usada para problemas de classificação com múltiplas classes, retornando uma probabilidade para cada uma das classes da tarefa. Normalmente ela é usada na última camada da rede neural, onde o número de neurônios será o mesmo que possíveis classes.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2.4}$$

2.1.2 Retropropagação e função de perda

Uma rede neural nada mais é do que um conjunto de neurônios interconectados entre si, em que cada neurônio recebe valores de entrada que são multiplicados pelos seus respectivos pesos sinápticos e, após uma combinação linear descrita pela equação 2.1, são processados através de uma função de ativação, gerando assim o *output* para a próxima camada.

Com isso, a maneira com que uma rede neural aprende é através de uma técnica chamada retropropagação. Esta técnica tem como finalidade ajustar os pesos sinápticos de toda a rede, comparando as saídas atuais da rede neural com as saídas corretas. Essa diferença entre o *output* gerado pela rede e o *output* real é calculada pela função de perda, que nos informa o nível de precisão da rede neural comparando as previsões da rede com as reais.

Após fazer uma previsão, o algoritmo ajusta cada um dos pesos sinápticos a fim de minimizar a função de perda, uma vez que quanto menor a função de perda de uma rede, mais precisa ela é. Assim, a cada ciclo de propagação, o algoritmo ajusta seus pesos através da retropropagação a fim de diminuir, cada vez mais, a sua função de perda.

Resumindo, a retropropagação nada mais é do que encontrar os melhores pesos sinápticos para toda a rede a fim de obter um *output* mais preciso e, consequentemente, diminuir sua perda. A função de perda que foi utilizada neste trabalho é a entropia cruzada que é descrita matematicamente pela equação 2.5. (ASTION; WILDING, 1992)

$$L(y, \hat{y}) = -\sum_{j=1}^{C} y_j \log \hat{y}_j \tag{2.5}$$

onde:

C é o número de classes;

y_j é a classe real que o modelo está tentando prever e

$\hat{y_j}$ é o valor probabilístico previsto pela rede para y_j.

Enquanto a função de perda é calculada para um único exemplo, a função de custo é soma das perdas de todos os exemplos contidos em cada ciclo de propagação e retropropagação.

2.1.3 Otimizadores

A rede neural procura através da retropropagação, ao ajustar cada um dos pesos, reduzir a função de custo a fim de obter uma saída mais precisa. A função dos otimizadores é ajudar a minimizar a função de custo mais rapidamente. Neste trabalho, o otimizador utilizado foi o Adam, porém, para explicá-lo, é abordado os conceitos de gradiente

descendente, *momentum* e do algoritmo RMSProp.

2.1.3.1 Gradiente Descendente

O gradiente descendente é um dos algoritmos de maior sucesso na tarefa de minimizar a função de custo. O método consiste em encontrar, de forma iterativa, os valores dos parâmetros, ou pesos, que minimizam a função de custo em questão, como exemplificado na equação 2.6.

$$\min_{\theta_0,\theta_1} J(\theta_0, \theta_1) \tag{2.6}$$

Para isso, o método do gradiente descendente atualiza seus parâmetros tomando ajustes a cada interação proporcionais ao inverso do gradiente da função de custo, ou seja, no sentido oposto das derivadas parciais da função em relação aos seus pesos, como descrito na equação 2.7 (RAO, 2019).

$$\theta_{t+1} = \theta_t - \alpha_t \cdot \nabla J\theta_t \tag{2.7}$$

onde:

α_t é o tamanho do ajuste, chamado de *learning rate*;

$\nabla J\theta_t$ é o gradiente da função de custo na iteração t

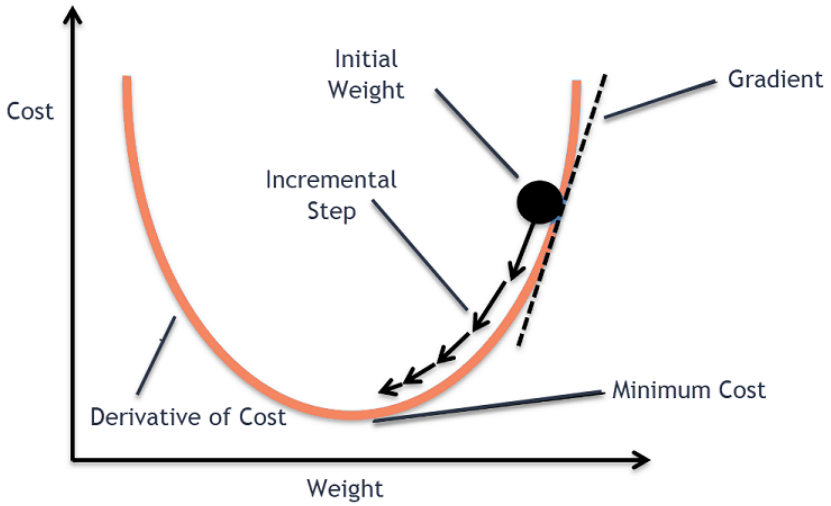
Ao se tomar um sentido contrário ao gradiente, o algoritmo vai tomando passos em direção ao mínimo da função de custo. A Figura 4 representa o método do gradiente descendente poucas dimensões para fins explicativos, porém, para problemas com maiores dimensões, como é o caso deste trabalho, a intuição continua a mesma.

É importante se destacar que, no caso do gradiente descendente explicado acima, a função de custo é calculada baseada em todos os exemplos do conjunto de dados, ou seja, a cada passo do gradiente descendente é utilizado toda a amostra de treino (FERREIRA, ; ANDRYCHOWICZ et al., 2016).

O gradiente descendente trata-se de um algoritmo útil para matrizes bem condicionadas, porém frequentemente traz problemas como um grande custo computacional, pois para cada atualização é necessário passar por todo conjunto de treino, e fraquezas em relação a observações redundantes.

Por esse motivo, neste trabalho foi utilizado o Gradiente Descendente *Mini-batch*, no qual cada passo do algoritmo utiliza-se de uma amostra do conjunto de treino, o qual chamamos de batch, e não dele todo, o que gera uma velocidade maior em se atualizar os parâmetros. O tamanho da amostra do conjunto utilizado é chamado de *batch size*.

Figura 4 – Representação do gradiente descendente.



Fonte: Clairvoyant.

2.1.3.2 *Momentum*

O conceito de *Momentum* é usado para acelerar o algoritmo do gradiente descendente ao utilizar uma média exponencialmente ponderada dos gradientes da rede. O uso dessa média faz com que o algoritmo convirja para o mínimo de uma maneira mais rápida e suave. (RUDER, 2016). A atualização dos pesos pelo algoritmo de *Momentum* pode ser descrita matematicamente pelas equações a seguir:

$$m_t = \beta m_{t-1} + (1 - \beta) \cdot \nabla J\theta_t \tag{2.8}$$

$$\theta_{t+1} = \theta_t - \ \alpha_t \cdot m_t \tag{2.9}$$

onde:

m_t é o agregado dos gradientes no tempo t;

m_{t-1} é o agregado dos gradientes no tempo t - 1;

$\nabla J\theta_t$ é o gradiente da função de custo na iteração t ;

β é o parâmetro da média móvel e

α_t é o tamanho do ajuste, chamado de *learning rate*.

2.1.3.3 *RMSProp*

Assim como o algoritmo de *momentum*, o RMSProp é uma adaptação do gradiente descendente a fim de convergir para o mínimo de uma maneira mais rápida e suave (HUK, 2020). Ele é descrito matematicamente pelas expressões a seguir:

$$v_t = \beta v_{t-1} + (1 - \beta) \cdot [\nabla J\theta_t]^2 \tag{2.10}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha_t}{\sqrt{(v_t + \epsilon)}} \cdot \nabla J\theta_t \tag{2.11}$$

onde:

v_t é o momento de segunda ordem no tempo t;

v_{t-1} é o momento de segunda ordem no tempo t - 1;

β é o parâmetro da média móvel;

$\nabla J\theta_t$ é o gradiente da função de custo na iteração t ;

α_t é o tamanho do ajuste, chamado de *learning rate* e

ϵ é uma constante positiva de valor muito pequeno para evitar uma divisão por 0.

2.1.3.4 Adam

O Adam junta os dois métodos descritos anteriormente para construir um gradiente descendente mais otimizado. O algoritmo consegue uma oscilação mínima quando atinge o mínimo global enquanto dá passos maiores para não estagnar em mínimos locais (KINGMA; BA,). O algoritmo de otimização Adam pode ser descrito matematicamente pelas equações a seguir, que utilizam das equações 2.8 e 2.10 para calcular m_t e v_t respectivamente.

$$\hat{m}_t = \frac{m_t}{(1 - \beta_1^t)} \tag{2.12}$$

$$\hat{v}_t = \frac{v_t}{(1 - \beta_2^t)} \tag{2.13}$$

$$\theta_{t+1} = \theta_t - \hat{m}_t \cdot (\frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon}) \tag{2.14}$$

onde:

β_1^t é o parâmetro para o momentum de primeira ordem no tempo t;

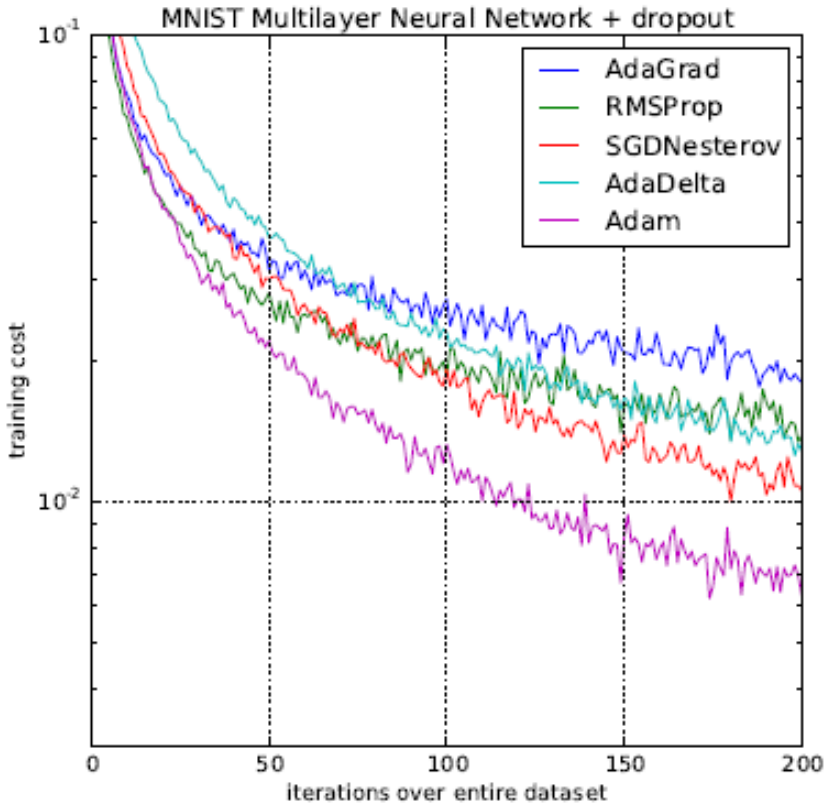
β_2^t é o parâmetro para o momentum de segunda ordem no tempo t;

α_t é o tamanho do ajuste, chamado de *learning rate*;

ϵ é uma constante positiva de valor muito pequeno para evitar uma divisão por 0.

A Figura 5 mostra a comparação entre diferentes otimizadores no treino no MNIST *dataset*. Com isso, pode-se observar um melhor desempenho do otimizador Adam em relação aos outros.

Figura 5 – Comparação do desempenho de diferentes otimizadores no MNIST *dataset*



Fonte: (KINGMA; BA, 2014)

2.1.3.5 *Batch normalization*

As escolhas relacionadas ao pré-processamento de dados geralmente fazem uma grande diferença no resultado final dos modelos. Com isso, a normalização dos dados de entrada na rede é uma prática muito comum, sendo normalmente feita para que a entrada tenha média zero e variância um, deixando os parâmetros, a priori, em uma escala semelhante.

Porém, durante o treinamento, as variáveis podem assumir valores com magnitudes amplamente variáveis durante a passagem pelas *hidden layers*, prejudicando assim a convergência da rede. Para atacar esse problema, os inventores do *Batch normalization* propuseram normalizar as entradas das camadas intermediárias do modelo (IOFFE; SZEGEDY, 2015).

As entradas, a cada iteração de treino, quando atingem a camada do *batch normalization* (BN) são normalizadas subtraindo a média do *batch* por seu desvio padrão, como explicitado nas equações a seguir. Com isso, é possível acelerar consistentemente a convergência da rede.

$$\hat{\mu_B} = \frac{1}{B}\sum_{i=1}^{B} x_i \tag{2.15}$$

$$\hat{\sigma_B}^2 = \frac{1}{B}\sum_{i=1}^{B}(x_i - \mu_B)^2 \tag{2.16}$$

$$BN(x) = \gamma \cdot \frac{x - \hat{\mu_B}}{\hat{\sigma_B}} + \beta \tag{2.17}$$

onde:

B é o tamanho do *batch*;

γ é o parâmetro de escala e

β é o parâmetro de deslocamento.

2.1.3.6 *Learning rate decay*

Com o objetivo de se chegar ao mínimo global da função de custo, o gradiente descendente vai tomando passos em direção à essa região, como representado na Figura 4. Um dos parâmetros do gradiente descendente é o *learning rate*, representado por α na equação 2.7, que altera o tamanho do ajuste feito a cada *epoch* (passagem por todos os exemplos do conjunto de dados).

Dependendo do valor do *learning rate*, o modelo pode ter complicações em se direcionar para o mínimo global da função, como demonstrado na Figura 6. Com isso, o *learning rate decay* é um algoritmo de otimização que visa atacar esse problema. (YOU et al., 2019)

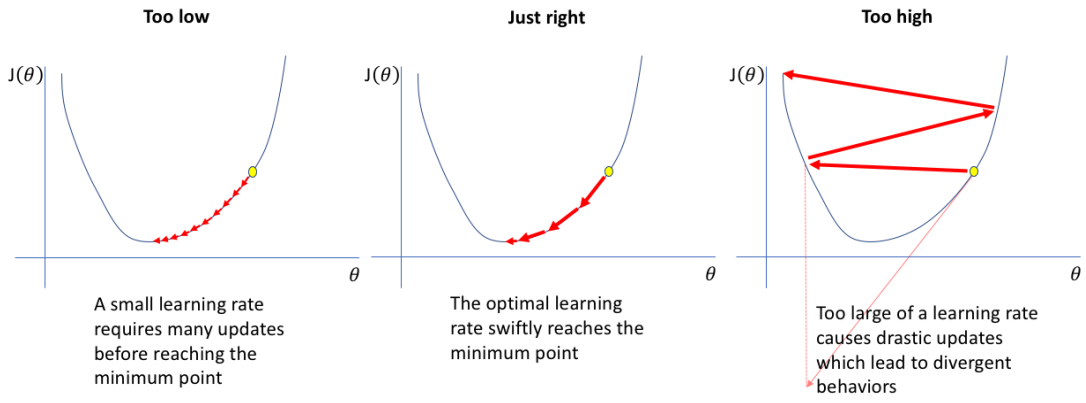
Após um certo número pré-estabelecido de *epochs* sem ganho de desempenho, o *learning rate* é corrigido por uma taxa pré-estabelecida que diminui seu valor para que o algoritmo consiga convergir para o mínimo global da função de uma maneira mais suave.

2.1.4 *Overfitting*

Overfitting é um dos problemas centrais na área de aprendizado de máquina. Ele consiste na incorporação de ruídos no aprendizado de uma rede, ou seja, quando um modelo aprende detalhes do seu conjunto de dados de treinamento que não estão presentes no restante como um todo. Com isso, ao tentar aplicar a rede em dados nunca antes vistos, o modelo perde sua capacidade de generalizar o aprendizado.

Para identificar esse problema, é reservado parte dos dados de treinamento para a validação. Assim, os dados de validação são dados nunca antes visto pelo modelo e o desempenho na classificação destes serve de métrica para medir o *overfitting*. Caso haja uma diferença considerável entre o desempenho da rede nos dados de treino comparada

Figura 6 – Efeito de diferentes *learning rates* na performance da rede



Jordan, Jeremy. 2018.

aos dados de validação, o modelo provavelmente está com um problema de *overfitting.* (CHOLLET, 2021; PROVOST; FAWCETT, 2013)

Quanto mais complexo um modelo, mais ele tende a sofrer deste problema. Com isso, o ideal é buscar um equilíbrio entre modelos mais simples e flexíveis, que generalizam melhor, com redes mais complexas que buscam uma melhor precisão. Além disso, técnicas de regularização, como as regularizações L1 e L2, buscam diminuir o efeito do *overfitting.*

2.2 Redes Neurais Convolucionais

As Redes Neurais Convolucionais, ou *Convolutional Neural Networks* (CNNs), são uma classe de rede neural artificial focada no processamento e análise de imagens digitais. Uma imagem nada mais é do que uma matriz de valores de pixels que pode ser achatada em um vetor. Assim, seria possível, após essa conversão, alimentar uma Rede Neural Artificial, como descrito na seção anterior, com esse vetor para aprender os parâmetros para a interpretação desta imagem.

Porém, uma rede neural tradicional não consegue desempenhar com muita precisão a tarefa de análise de imagens digitais. Este problema se dá principalmente por conta do custo computacional necessário para analisar uma imagem digital. Um dos conjuntos de dados de imagens mais famosos, o MNIST *dataset*, é adequado para a maioria das Redes Neurais Artificiais por conta da sua pequena resolução, sendo apenas de 28 x 28. Com isso, um neurônio na primeira *hidden layer* terá apenas 784 pesos, uma vez que as imagens são em escala cinza, o que ainda é gerenciável para a RNA.

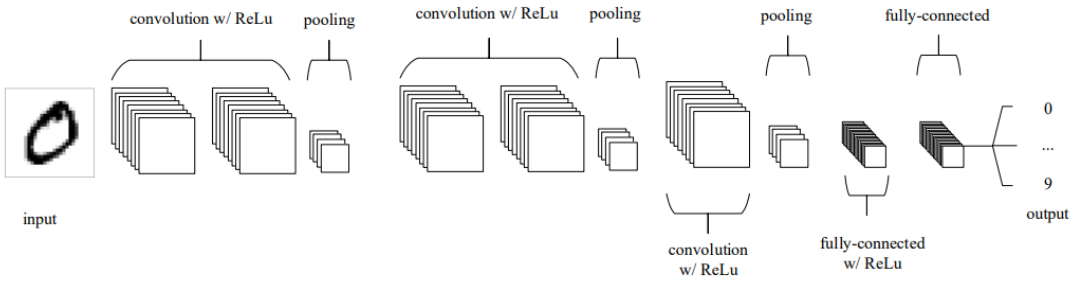
Entretanto, com uma imagem com uma resolução um pouco maior, como uma imagem colorida (RGB) de 64 x 64, o número de pesos aumenta substancialmente para 12288. Com isso, o custo computacional e a complexidade de rede tem que aumentar muito para a interpretação de imagens.

Uma CNN é um algoritmo de aprendizado profundo que atribui filtros relevantes capazes de capturar com sucesso as dependências espaciais em uma imagem. Através dessa arquitetura específica das redes neurais convolucionais, é possível retirar da imagem de entrada informações importantes, ou *features*, através dos filtros enquanto se reduz o número de parâmetros envolvidos. Assim, a rede pode ser treinada para entender com mais precisão a sofisticação de uma imagem. (GU et al., 2018)

2.2.1 Arquitetura de uma Rede Neural Convolucional

Uma Rede Neural Convolucional é composta por três principais tipos de camadas: a camada convolucional (*convolutional layer*), a camada de *pooling* (*pooling layer*) e a camada totalmente conectada (*fully connected layer*). A Figura 7 representa um exemplo de arquitetura de uma rede neural convolucional.

Figura 7 – Representação da arquitetura de uma rede neural convolucional



Fonte: (O'SHEA; NASH, 2015)

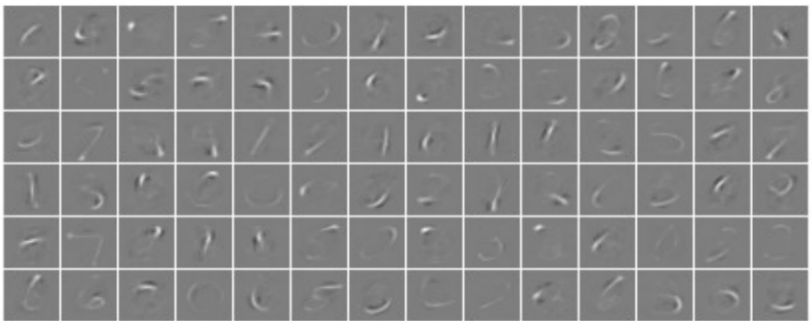
2.2.1.1 Camada convolucional

A camada convolucional, como o nome diz, está intimamente ligada às CNNs. Essas camadas têm como foco o aprendizado dos filtros que serão usados para retirar *features* das imagens. A Figura 8 mostra as saídas da primeira camada convolucional de uma CNN treinada no MNIST *dataset* de dígitos escritos manualmente. A rede conseguiu com sucesso, a partir dos filtros aplicados, extrair características, mesmo que simples, únicas para dígitos específicos.

Os filtros normalmente são pequenos em relação a sua dimensão espacial, porém, são aplicados ao longo de toda a dimensionalidade da imagem. Quando o dado chega em uma camada convolucional, a camada faz operações matemáticas chamadas convoluções para produzir um mapa de ativação 2D.

A Figura 9 mostra a representação de uma *convolutional layer* onde é aplicado um filtro em uma amostra da imagem. Com isso, o valor de saída é baseado na convolução entre o filtro e a amostra em questão, como na Figura 10.

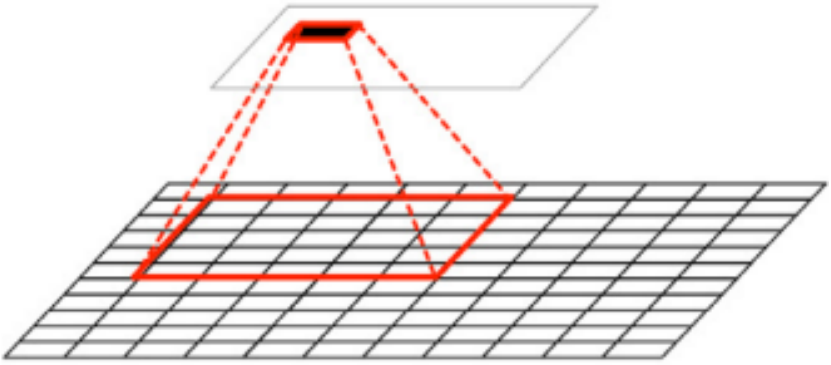
Figura 8 – *Outputs* da primeira camada convolucional de uma CNN



Fonte: (O'SHEA; NASH, 2015)

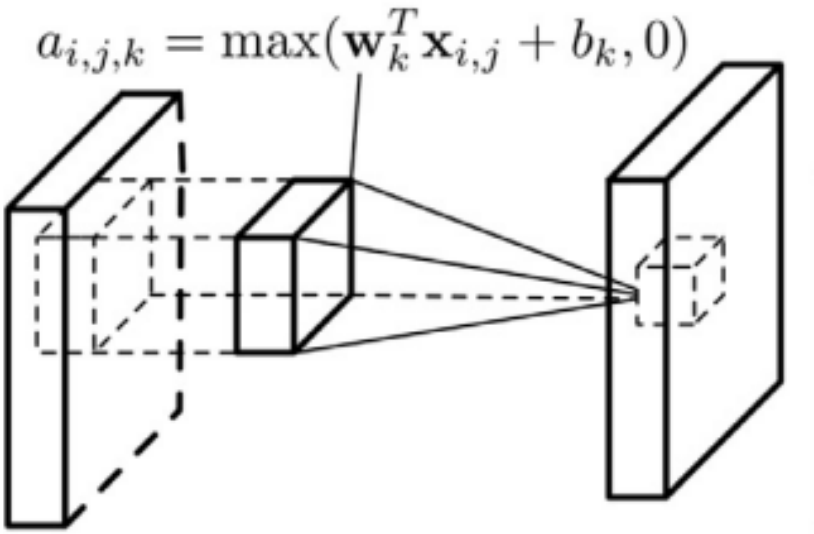
Cada filtro será correspondente à um padrão de identificação, como por exemplo cantos e linhas verticais e horizontais. Assim, é possível identificar esses padrões pelas operações de convolução.

Figura 9 – Representação visual de uma convolução.



Fonte: (GU et al., 2018)

Figura 10 – Representação da operação de convolução.



Fonte: (GU et al., 2018)

As camadas convolucionais são capazes de aprender através da otimização do *output* do modelo pela retropropagação. Para isso, elas buscam a otimização de hiperparâmetros do modelo: a profundidade, o *stride* e o *padding*.

A profundidade é o volume da saída produzida pela camada convolucional, ou seja, o número de filtros aplicados. Ao se reduzir esse hiperparâmetro, temos uma diminuição considerável no número total de parâmetros do modelo, reduzindo sua complexidade, porém há uma perda significativa na habilidade de reconhecimento do modelo.

Stride é o hiperparâmetro que modifica a quantidade de movimento do filtro dentro da imagem de entrada, ou seja, o tamanho de seus passos. Por exemplo, se o *stride* é definido como um, o filtro se move um pixel por vez, caso o *stride* seja definido como dois, o filtro move dois pixels por vez. Isso afeta diretamente o tamanho da saída da camada convolucional.

Padding é uma técnica que tem como objetivo adicionar pixels, normalmente com valor zero, na borda da imagem para controlar a dimensionalidade do *output* da camada. Com isso, o valor estabelecido decide quantos pixels a mais serão adicionados ao redor da imagem de entrada. Neste trabalho foi utilizado o *same padding*, que consiste em, para um *stride* de um, é estabelecido um valor para de *padding* para que a dimensão de entrada seja igual a de saída.

A dimensão de saída de uma convolução para uma imagem de dimensões Win x Hin, com um filtro K x K, *stride* S e *padding* P é dada pelas equações a seguir.

$$Wout = \frac{W_{in} - K + 2P}{S} + 1 \tag{2.18}$$

$$Hout = \frac{H_{in} - K + 2P}{S} + 1 \tag{2.19}$$

$$Output_{dim} = (Wout, Hout) \tag{2.20}$$

2.2.1.2 Camada de *Pooling*

A *pooling layer* tem como objetivo reduzir a dimensionalidade da imagem gradualmente e, consequentemente, reduzir o número de parâmetros do modelo e seu custo computacional.

Uma operação de *pooling* é semelhante ao que se observa na Figura 9, porém, ao invés do processo de convolução, a camada reduz a dimensionalidade baseada em uma média dos pixels que estão na amostra (*average pooling*) ou, como é mais comum nos dias atuais, através do valor máximo dos pixels dentro da amostra (*max-pooling*).

Uma das operações de *pooling* mais utilizadas seria com filtros de dimensão 2 x 2 e *stride* de 2. Com esses parâmetros de filtro e *stride*, a camada de *pooling* reduz o tamanho da imagem para 25% de sua dimensão original.

2.2.1.3 Camada totalmente conectada

A camada totalmente conectada contém neurônios que estão diretamente conectados à neurônios nas duas camadas adjacentes, similarmente a maneira com que os neurônios são arranjados em uma rede neural artificial tradicional.

2.2.2 ResNets

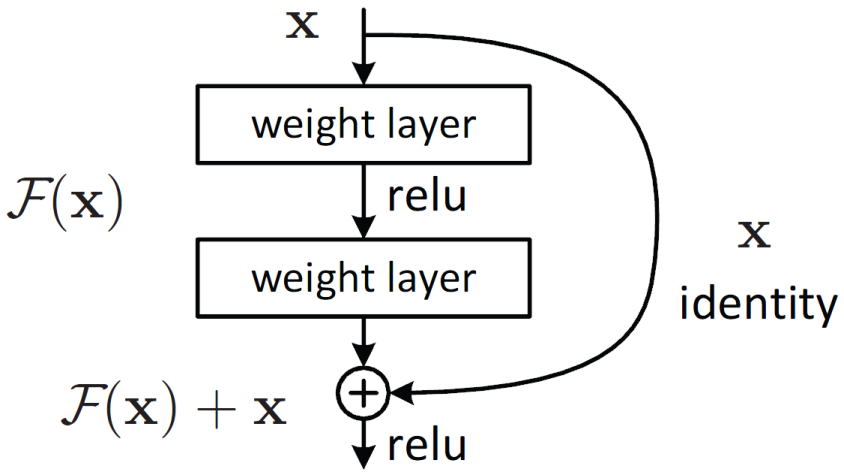
Diante do grande número de variáveis que podem mudar na arquitetura de uma rede neural convolucional, algumas arquiteturas que se destacaram em certas tarefas são usadas em grande escala. As ResNets são um desses casos. Dentro da área de aprendizagem profunda, houve um investimento em redes mais profundas para aumentar a precisão dos modelos. Porém, durante o treinamento dessas redes foi observado que, ao se adicionar muitas camadas à rede neural o valor da acurácia estagnava ou até diminuía.

Isso acontece por conta do *vanishing gradient problem*. Este problema se dá por conta que, a medida em que mais camadas com alguns tipos de função de ativação são adicionadas no modelo, o gradiente da função de custo, que é usado pela retropropagação para ajustar os pesos e adquirir uma precisão maior, se aproxima de zero nas camadas iniciais, prejudicando assim o aprendizado e gerando uma imprecisão em toda a rede.

Com isso, as *Residuals Neural Networks*, ou ResNets, foram arquitetadas para resolver o problema do *vanishing gradient*. A sua diferença está em um novo tipo de conexão entre as camadas chamadas de Identidade.

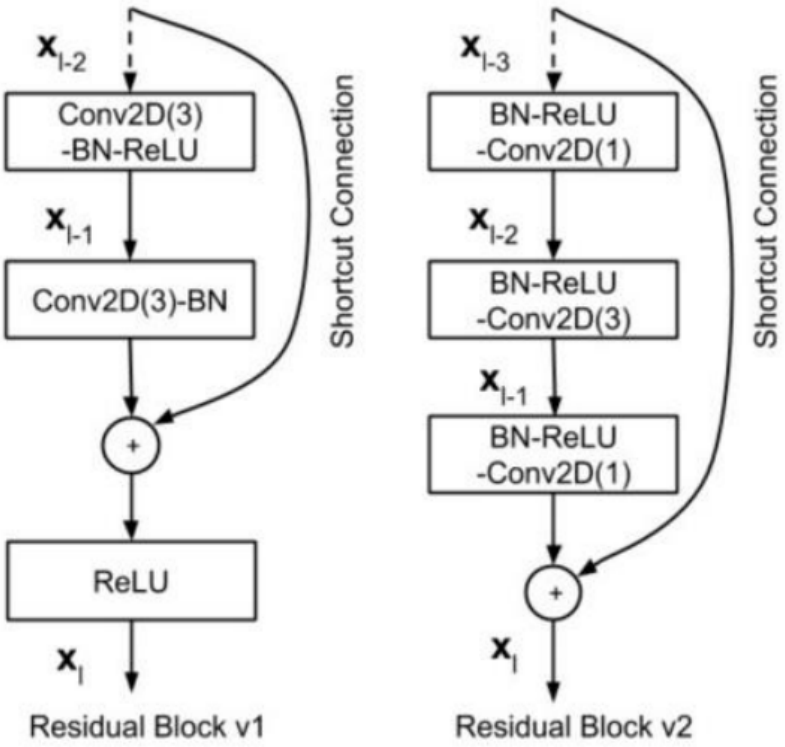
Ao invés de aprender por um caminho direto entre entrada e saída de uma camada, as redes residuais fornecem conexões residuais ao longo de suas camadas, como exemplificado na Figura 11. A conexão residual adiciona diretamente o valor de entrada do bloco, descrito na figura por x, ao final do bloco, resultando em $F(x) + x$, em que $F(x)$ simboliza o *outputs* das camadas não lineares. Como essa conexão não passa pelas funções de ativação, não há uma degradação do gradiente através das derivadas parciais e, com isso, reduzir o impacto do *gradient vanishing* nas camadas iniciais.

Figura 11 – Representação de um bloco residual



Fonte: (HE et al., 2016)

Figura 12 – Comparação entre o bloco residual da ResNetV1 e ResNetV2



Fonte: (HIEU; HIEN, 2020)

A ResNetV2 é uma versão mais nova do modelo, em que há um novo arranjo das camadas no bloco residual, como mostrado na Figura 12. A Figura 13 detalha diversas variações da arquitetura das redes residuais, como a ResNet50 e Resnet101, em que o número final representa o número de camadas.

Neste trabalho foi utilizada a rede ResNet50V2. Por fim, é importante destacar que as ResNets apresentam uma grande precisão no conjunto de dados de imagem *ImageNet*.

Figura 13 – Detalhamento das variações das arquiteturas das redes residuais.

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				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Fonte: (HE et al., 2016)

2.3 Transferência de Aprendizado

Modelos de aprendizado profundo necessitam de uma grande quantidade de dados para conseguir aprender os seus parâmetros. Entretanto, em algumas áreas, há uma dificuldade em conseguir o número de dados necessários para isso.

Assim, para superar esse problema, é utilizado da técnica de transferência de aprendizado, que visa transmitir o conhecimento de modelos pré-treinados de um problema com um banco de dados considerável para outros problemas similares.

Um exemplo de um modelo que é bastante utilizado em transferência de aprendizado é a ResNet. Essa rede é treinada para extrair as características necessárias para a classificação das classes na base de dados *ImageNet*, que contém 14 milhões de imagens. A partir desta rede pré-treinada, que tem seus pesos congelados, impedindo assim a alteração desses parâmetros, são criados novos modelos que introduzem novas camadas a esta rede pré-treinada.

A partir disso, apenas as camadas introduzidas pelo novo modelo são treinadas a fim de se adequar a nova tarefa. Além dessa possibilidade, pode-se descongelar as últimas camadas convolucionais da rede pré-treinada, a fim de otimizar ainda mais a extração de *features* significativas. Esse processo é chamado de *Fine-tuning*. (IESB; LIMA, ; TAN et al., 2018)

2.4 Métodos de regularização

A regularização tem como objetivo diminuir o efeito do *overfitting*. Existem algumas maneiras para se atacar o problema, como aumentar a quantidade de dados de treinamento ou reduzir o tamanho da rede, mesmo sabendo que estas têm maior potencial de obterem um desempenho melhor. Porém, em momentos em que a coleta de mais dados se mostra inviável e deseja-se manter o tamanho da rede fixa, são introduzidas as técnicas de regularização.

A regularização tem como efeito desejado a preferência da rede de aprender pequenos pesos sinápticos, em que pesos grandes só serão incorporados se melhorarem substancialmente os resultados.

Isso pode ser feito através de uma penalização na função de custo, como explicitado na equação 2.21, que demonstra a entropia cruzada regularizada, em que a primeira parte da equação é a entropia cruzada demonstrada na equação 2.5 e a segunda parte da equação é a penalidade da regularização L2, também chamada de *Ridge regression*.

$$L(y, \hat{y}) = -\sum_{j=1}^{C} y_j \log \hat{y_j} + \lambda \sum_{i=1}^{N} w_i^2 \tag{2.21}$$

onde:

C é o número de classes;

y_j é a classe real que o modelo está tentando prever;

$\hat{y_j}$ é o valor probabilístico previsto pela rede para y_j;

λ é o fator de regularização;

N é o número de pesos da rede e

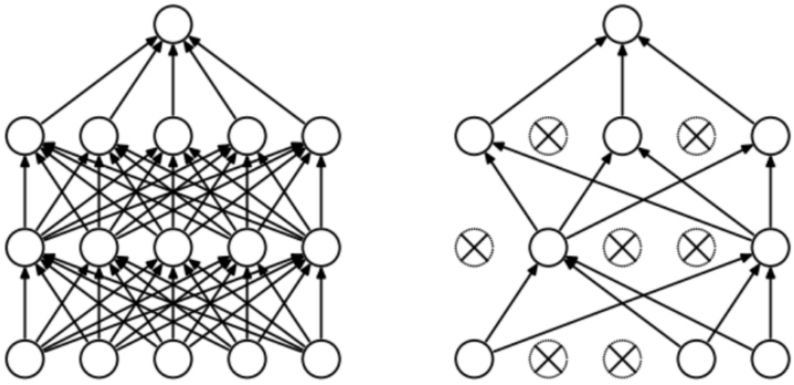
w_i sendo cada um dos pesos.

Ao se colocar uma penalidade, dificulta-se que uma rede regularizada aprenda os efeitos do ruído do conjunto de dados de treino, gerando uma tendência para o aprendizado de pesos pequenos que conseguem generalizar melhor. Um modelo com pesos grandes, por outro lado, fica sensível a pequenas alterações nos dados de entrada. Duas técnicas de regularização foram utilizadas neste trabalho: o *dropout* e a *data augmentation*.

2.4.1 *Dropout*

O *Dropout* tem uma abordagem diferente para a regularização quando comparado a métodos que modificam a função de custo, como no exemplo da regularização L2 utilizada na equação 2.21. O *Dropout*, ao invés de alterar a função de custo, modifica a própria rede. O processo elimina aleatoriamente e temporariamente, com uma probabilidade estabelecida p, neurônios das camadas ocultas, como pode ser observado na Figura 14.

Figura 14 – Representação da técnica de *dropout*



Fonte: (ÖZGÜR; NAR, 2020).

A cada iteração da rede os neurônios que foram eliminados são reinstaurados e novos são eliminados para a próxima iteração. Com isso, os neurônios são treinados à não confiar na presença de outros neurônios em particular. Assim, a rede não fica dependendo de algum neurônio em específico, sendo forçada a aprender recursos mais robustos que geram uma melhor generalização e, consequentemente, um menor *overfitting*.

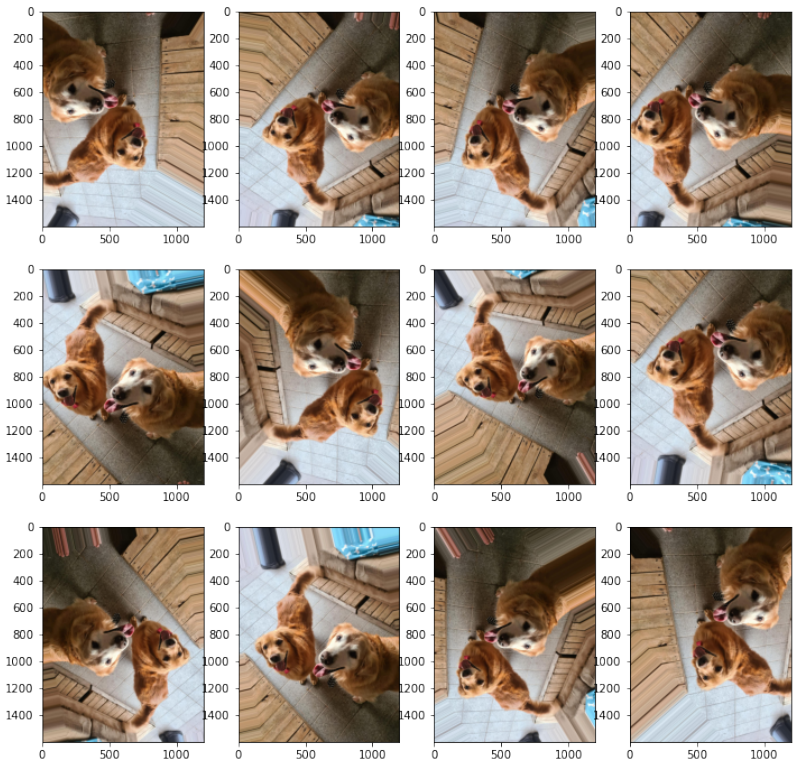
2.4.2 *Data augmentation*

Data augmentation é um método de regularização que visa gerar novos dados de imagens de forma artificial baseados no seu conjunto original. Através de diversas técnicas de modificação de imagens, como reflexão horizontal e vertical, zoom, variação de brilho e saturação, é possível a partir de um conjunto de dados de treino pequeno gerar novas imagens com pequenas modificações que deixam a rede mais robusta e com um maior conjunto para treinar e aprender os seus parâmetros. A Figura 16 mostra as imagens geradas pela técnica de *data augmentation* na imagem original demonstrada na Figura 15.

Figura 15 – Imagem original

Fonte: Elaboração própria.

Figura 16 – Imagens geradas pela técnica de *Data augmentation*



Fonte: Elaboração própria.

2.5 Métricas de avaliação

As métricas de avaliação em uma tarefa de classificação tem como objetivo analisar o desempenho do modelo acerca de uma tarefa. Para este trabalho, foram utilizadas quatro métricas: acurácia, precisão, *recall* e *F1-score*. Além disso, é explicado o conceito de matriz de confusão. (GRANDINI; BAGLI; VISANI, 2020)

Uma classificação feita pela rede pode ser alocada dentro de quatro classes:

1. Verdadeiro positivo (VP): quando a rede classifica a classe como positiva e a classe é realmente positiva;
2. Verdadeiro negativo (VN): quando a rede classifica a classe como negativa e a classe é realmente negativa;
3. Falso positivo (FP): quando a rede classifica a classe como positiva e a classe na verdade é negativa e
4. Falso negativo (FN): quando a rede classifica a classe como negativa e a classe na verdade é positiva.

2.5.1 Acurácia

Acurácia é uma métrica de avaliação que é calculada pela razão entre o número de classificações corretas e o número de classificações totais.

$$Acurácia = \frac{VP + VN}{VP + VN + FP + FN} \tag{2.22}$$

2.5.2 Precisão

Precisão é uma métrica de avaliação que é calculada pela razão entre o número de verdadeiros positivos e a soma de todos os valores que são realmente positivos.

Como parâmetro da função, foi utilizado nesse trabalho uma média "macro", que calcula a métrica para cada classe e realiza uma média simples. Como os dados têm um conjunto como número igual de exemplos para cada classe, não há problema em relação a desbalanceamento de quantidade.

$$Precisão = \frac{VP}{VP + FP} \tag{2.23}$$

2.5.3 *Recall*

Recall é uma métrica de avaliação que tem como objetivo avaliar a capacidade do modelo de detectar com sucesso resultados dados como positivos. É calculada pela razão

entre o número de verdadeiros positivos e a soma entre o número de verdadeiros positivos e o número de falsos negativos.

$$Recall = \frac{VP}{VP + FN} \tag{2.24}$$

2.5.4 *F1-Score*

F1-Score é uma métrica de avaliação que é calculada pela média harmônica com base na precisão e no *recall*. Como parâmetro utilizado da média "macro".

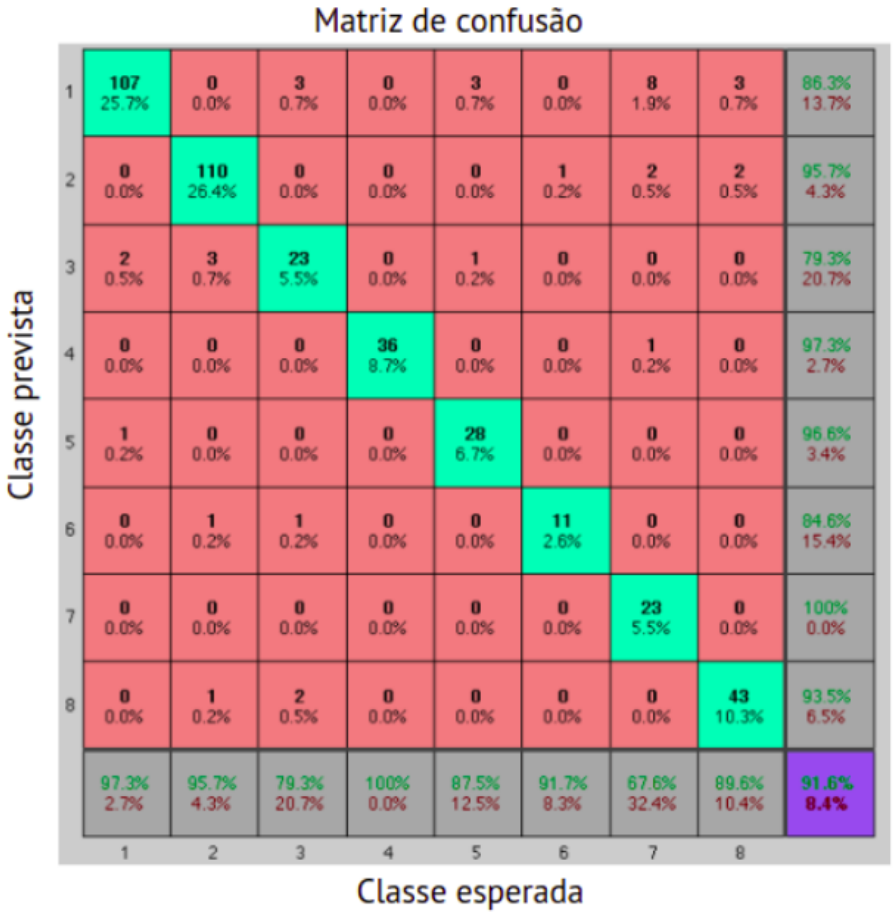
$$F1 = 2 \cdot \frac{precisão \cdot recall}{precisão + recall} \tag{2.25}$$

2.5.5 Matriz de confusão

A matriz de confusão é uma tabela que indica os falsos positivos, falsos negativos, verdadeiros positivos e verdadeiros negativos de cada uma das classes com o objetivo de fornecer uma melhor interpretação dos resultados do modelo.

As linhas da matriz representam as classes verdadeiras e as colunas representam as classes previstas pelo modelo. Assim a matriz mostra o número de vezes em que o modelo previu corretamente cada classe em relação ao número de vezes em que a classe foi realmente observada. Com isso, quanto mais desempenho o modelo tiver, mais próxima de uma matriz diagonal é a matriz de confusão.

Figura 17 – Exemplo de matriz de confusão



Fonte: (GIANESI; ALUíSIO, 2021)

3 DESENVOLVIMENTO

Este trabalho teve como objetivo a criação de um modelo de aprendizado de máquina baseado em redes neurais convolucionais que apresentasse uma alta precisão na interpretação do alfabeto de linguagem de sinais.

Para isso, foi comparado diferentes arquiteturas de redes neurais convolucionais juntamente com o uso de técnicas que visam otimizar os resultados das redes no conjunto de dados utilizado para treino e teste.

Este capítulo tem como objetivo explicitar o processo de criação dos modelos analisados juntamente com a descrição dos dados utilizados e a discussão de seus resultados. Além disso, a rede com melhor desempenho foi incorporada em um modelo de captura de imagem por vídeo para classificação em tempo real do alfabeto de linguagem de sinais. O capítulo é dividido nas seguintes seções:

1. Descrição do conjunto de dados utilizado para treinamento e validação da tarefa;
2. Desenvolvimento de diferentes arquiteturas de redes neurais convolucionais juntamente com o estudo sobre técnicas de otimização destacadas no Capítulo 2;
3. Apresentação e discussão dos resultados e
4. Aplicação da rede em tempo real.

3.1 Descrição dos dados

O *dataset* utilizado para a tarefa contém 78 mil imagens de gestos representando 26 classes, onde cada uma tem 3000 exemplos, relacionadas ao alfabeto de linguagem de sinais, disponibilizado na plataforma Kaggle (ASL Alphabet Dataset), e foi feito por Akash Nagaraj. É importante destacar que originalmente os dados continham 84 mil imagens com 28 classes, porém, como a letra J e Z no alfabeto de linguagem de sinais exige movimento, as duas classes foram descartadas nesta primeira análise.

Como demonstrado na Figura 18, temos 26 diferentes classes para a classificação do modelo: A, B, C, D, E, F, G, H, I, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, *nothing* e *space*, onde as primeiras 24 classes designam uma letra do alfabeto enquanto a classe *nothing* representa um fundo vazio e a *space* a representação do espaço.

Outro ponto a ser destacado é que os dados não seguem um padrão de linguagem, como o *American Sign Language* (ASL), como um todo. Grande parte das letras são representadas seguindo o ASL, porém as letras M e N seguem o padrão da Linguagem

Brasileira de Sinais (LIBRAS) e as letras D, F e T seguem o padrão da Linguagem Italiana de Sinais (LIS).

Dentro do conjunto de 78 mil imagens, 80% dos exemplos (62400 imagens) foi direcionado para o conjunto de treino enquanto os 20% (15600 imagens) foi direcionado para o conjunto de teste. As imagens utilizadas tiveram seu tamanho ajustado para 128x128x3.

Figura 18 – Classes dos dados utilizados

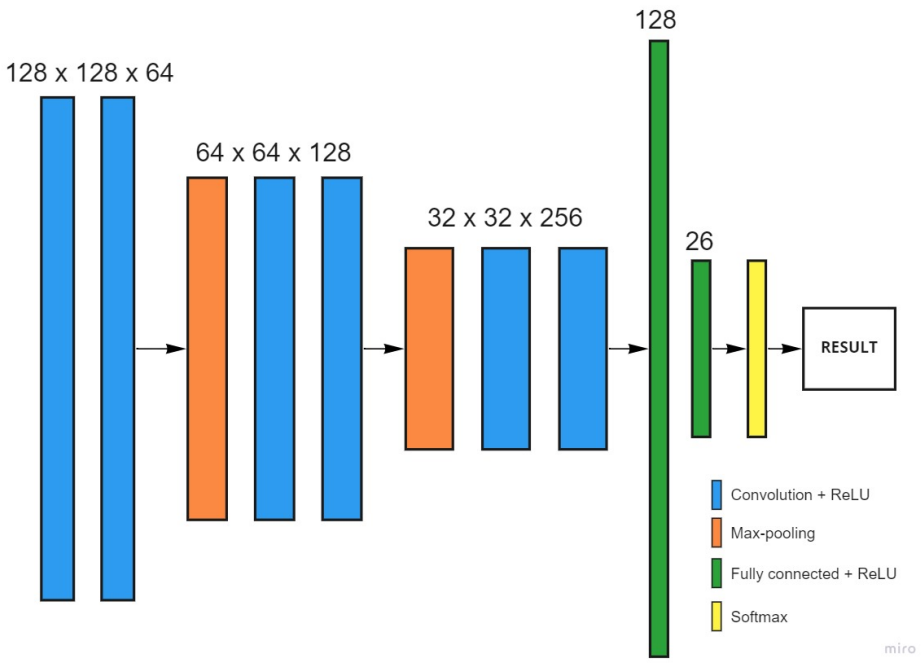
Fonte: Dados do ASL Alphabet Dataset (3.1) e elaboração própria

3.2 Desenvolvimento dos modelos

Neste trabalho foram analisados quatro diferentes modelos de redes neurais convolucionais. Dentro desses quatro, duas arquiteturas em especial foram aplicadas e, os outros dois modelos, foram uma variação de uma delas. Essa seção tem como objetivo demonstrar todas as arquiteturas utilizadas.

A primeira arquitetura, nomeada neste trabalho de C6P2FC2, é baseada no artigo (LI et al., 2020) e pode ser observada na representação exibida na Figura 19. A arquitetura, quando comparada com a do artigo citado, foi simplificada por conta de falta de poder computacional.

Figura 19 – Arquitetura da rede C6P2FC2 utilizada

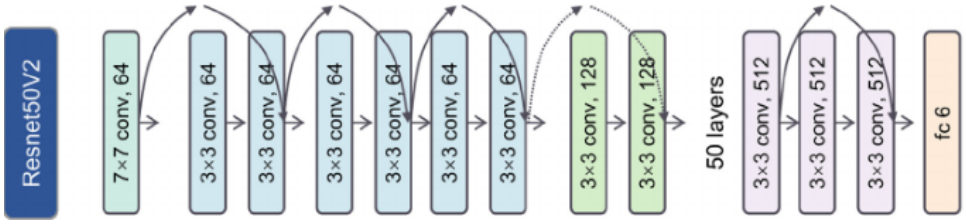


Fonte: Elaboração própria

Essa rede inclui 6 camadas convolucionais, 2 camadas de *max-pooling* e 2 camadas totalmente conectadas. O tamanho de cada filtro de convolução é de 3 x 3 e utiliza da ReLU como função de ativação e a camada de *max-pooling* tem filtros de 2 x 2 com *stride* 2. Com isso, obteve-se um número de parâmetros treináveis de 21.628.890. Além disso, foi utilizado do otimizador Adam (2.1.3.4) para o algoritmo de gradiente descendente e a entropia cruzada como função de perda.

Os outros três modelos foram treinados com base na transferência de aprendizado, conceito explicado na seção 2.3, do modelo ResNet50V2 (2.2.2), que tem sua arquitetura demonstrada na Figura 20. A ResNet50V2 se divide em cinco blocos, demarcados pelas conexões residuais na imagem. Esse modelo foi escolhido para ser utilizado através da transferência de aprendizado por conta dos seus bons resultados no conjunto de dados *ImageNet*.

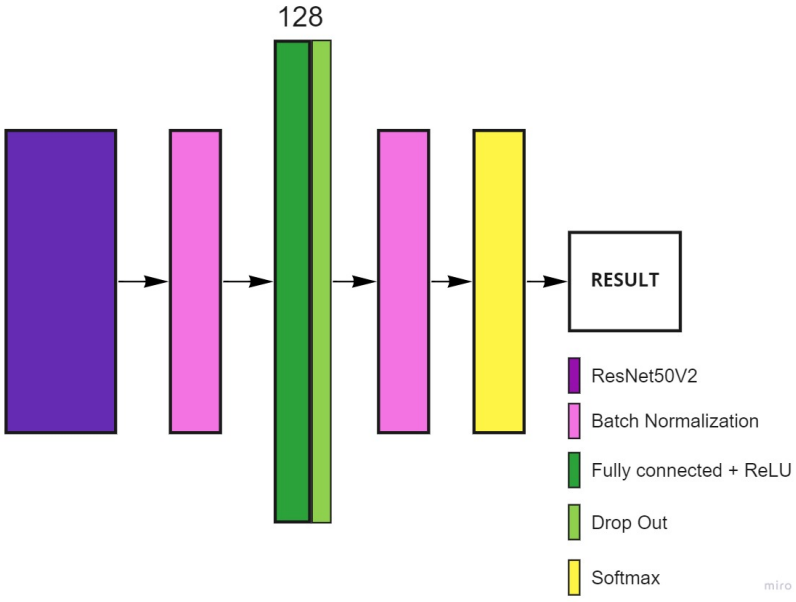
Figura 20 – Arquitetura da ResNet50V2



Fonte: (LIAO et al., 2021)

A arquitetura final da rede pode ser observada na Figura 21. Utilizando-se da ResNet50V2, foi incorporado ao final do modelo, antes da camada de ativação *softmax*, duas camadas de *Batch Normalization* separadas por uma camada totalmente conectada com 128 neurônios com um *Drop Out* de 50%. Além disso, foi utilizado do otimizador Adam (2.1.3.4) para o algoritmo de gradiente descendente e a entropia cruzada como função de perda.

Figura 21 – Arquitetura da rede gerada por transferência de aprendizado utilizada



Fonte: Elaboração própria

Os três últimos modelos foram baseados na arquitetura da Figura 21. O primeiro deles, nomeado de RNNOFT (*ResNet with no fine tuning*), congela todos os parâmetros da rede residual e apenas treina os que foram incorporados posteriormente a ResNet50V2, resultando em 4.263.578 parâmetros treináveis.

O segundo e o terceiro modelo, nomeados de RNFT (*ResNet fine tunned*) e RNFTLRD (*ResNet fine tunned with learning rate decay*) respectivamente, descongelam o último bloco da rede residual, ficando assim com 19.234.458 parâmetros treináveis.

A diferença entre o RNFT e o RNFTLRD é que o último incorpora um *Learning Rate Decay* (2.1.3.6), com fator de correção de 0.5 a cada três *epochs* sem ganho na acurácia do modelo, para que o algoritmo de gradiente descendente vá em direção ao mínimo da função de custo de uma maneira mais sauve.

3.3 Apresentação dos resultados e discussão

Essa seção busca apresentar os resultados de cada um dos modelos em relação às métricas de avaliação descritas na seção 2.5. Todos os modelos foram treinados em um total de 20 *epochs* com um *batch size* de 128. As Tabelas 1 e 2 mostram os resultados de cada uma das redes e seu tempo de treinamento.

Tabela 1 – Resultados de teste e treino das redes trabalhadas

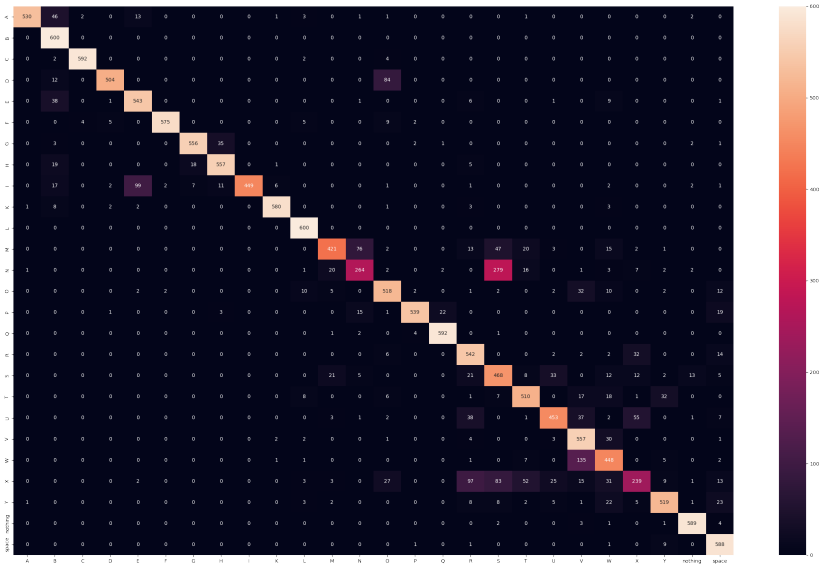
	Accuracy	Precision	Recall	F1-Score
C6P2FC2	Treino: 0.9948	Treino: 0.9948	Treino: 0.9948	Treino: 0.9948
	Teste: 0.8547	Teste: 0.8638	Teste: 0.8547	Teste: 0.8521
RNNOFT	Treino: 0.9941	Treino: 0.9941	Treino: 0.9941	Treino: 0.9941
	Teste: 0.8155	Teste: 0.8277	Teste: 0.8155	Teste: 0.8132
RNFT	Treino: 0.9989	Treino: 0.9989	Treino: 0.9989	Treino: 0.9989
	Teste: 0.9187	Teste: 0.9238	Teste: 0.9187	Teste: 0.9170
RNFTLRD	Treino: 0.9997	Treino: 0.9997	Treino: 0.9997	Treino: 0.9997
	Teste: 0.9240	Teste: 0.9350	Teste: 0.9240	Teste: 0.9219

Tabela 2 – Tempo de treinamento para 20 *epochs*

	Tempo de treinamento	Média por *epoch*
C6P2FC2	26h58min	1h21min
RNNOFT	6h08min	18min23s
RNFT	11h44min	35min13s
RFNTLRD	11h45min	35min16s

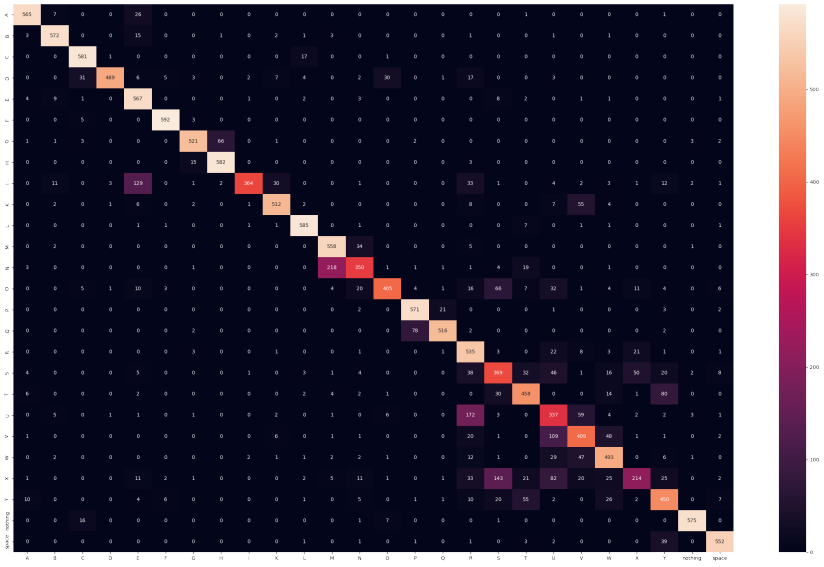
As Figuras 22, 23, 24 e 25 mostram a matriz de confusão, apresentada na seção 2.5.5, de cada um dos modelos para o conjunto de treino. Os eixos estão organizados da mesma maneira que o exemplo da Figura 17, contendo 26 classes em cada eixo.

Figura 22 – Matrix de confusão da rede C6P2FC2



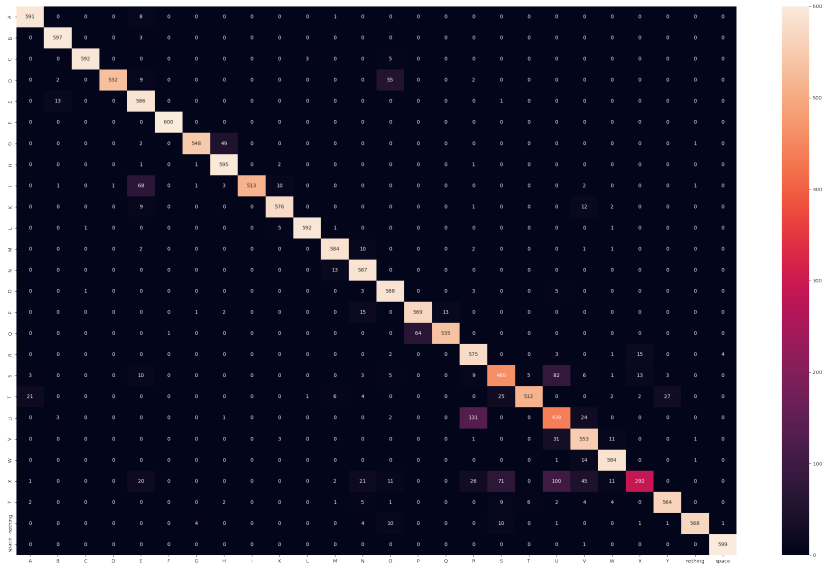
Fonte: Elaboração própria

Figura 23 – Matrix de confusão da rede RNNOFT



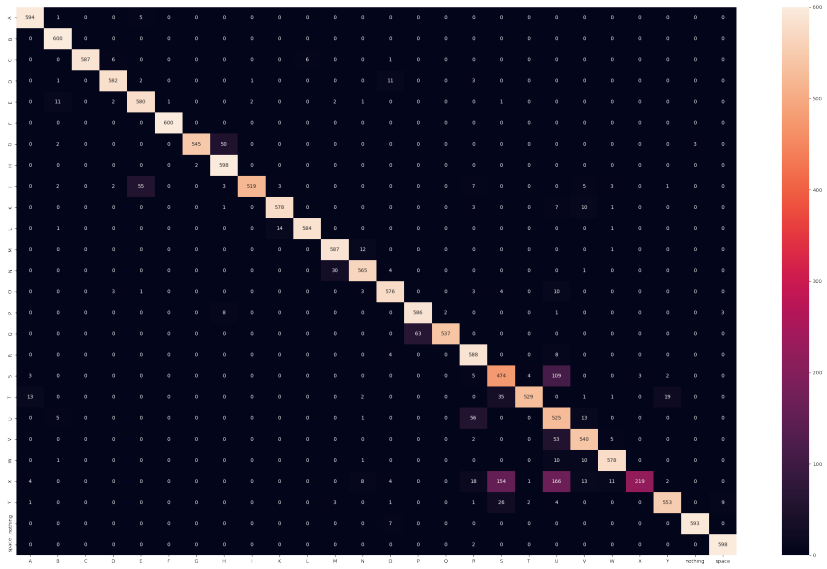
Fonte: Elaboração própria

Figura 24 – Matrix de confusão da rede RNFT



Fonte: Elaboração própria

Figura 25 – Matrix de confusão da rede RNFTLRD



Fonte: Elaboração própria

Como se pode observar através dos resultados da Tabela 1, o modelo com melhor desempenho, tanto no treino quanto no teste, é o RNFTLRD. Ao se comparar ele com o segundo melhor, o RNFT, que contém a mesma arquitetura e número de parâmetros, é observado o impacto positivo da técnica de *learning rate decay*, demonstrada na Seção 2.1.3.6, sem acréscimo de tempo de treinamento, como mostrado na Tabela 2.

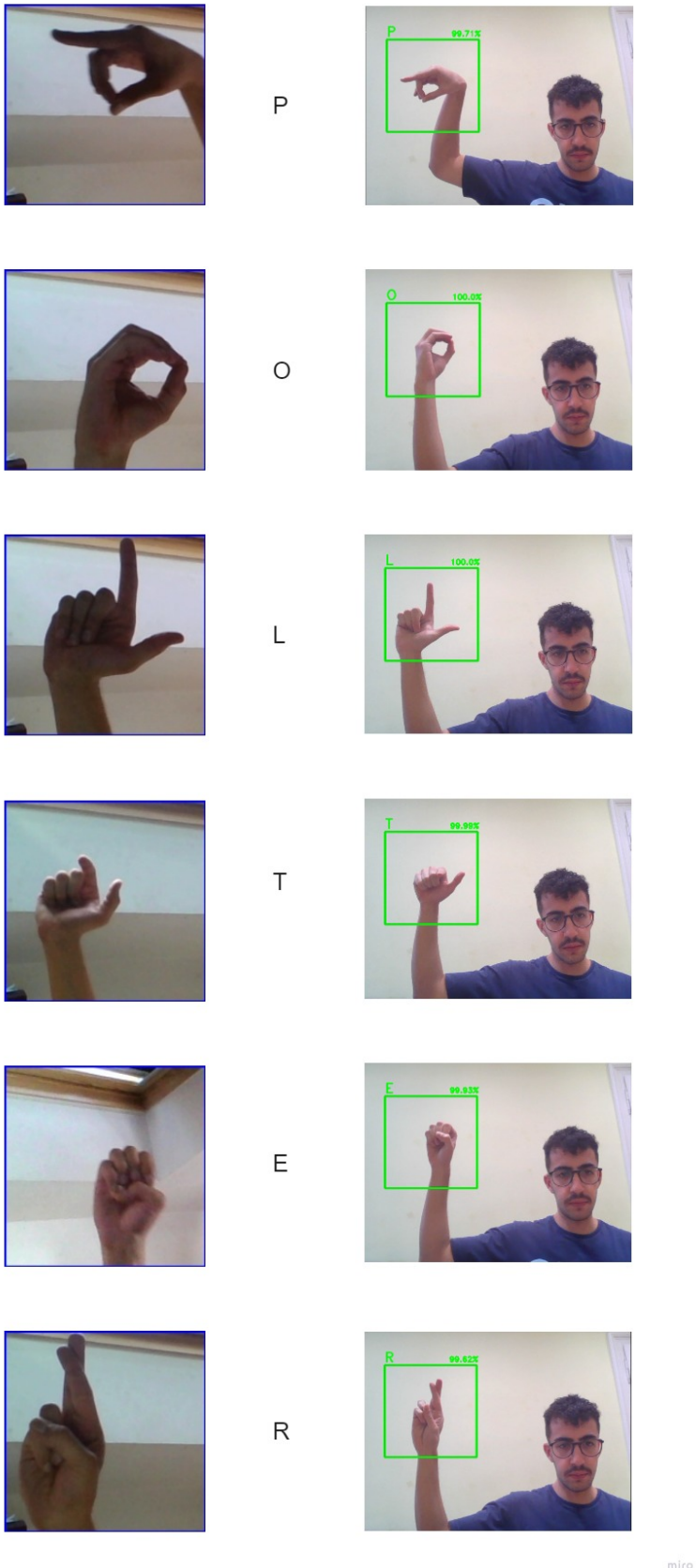
Outro ponto a ser observado é o ganho que se obtém ao realizar a transferência de aprendizado, descrita na subseção 2.3. Ao se comparar os resultados dos modelos C6P2FC2 e o RNNOFT nas Tabelas 1 e 2, é observado que o modelo RNNOFT, que utiliza da transferência de aprendizado, consegue obter uma acurácia próxima do modelo C6P2FC2, com uma perda de 4,59% no teste, com apenas 22,74% do tempo de treino.

Além disso, ao se comparar os resultados dos modelos RNNOFT e RNFT, é possível observar o ganho de acurácia ao realizar a técnica de *fine tuning* de uma maneira mais profunda, uma vez que foi descongelado o último bloco da ResNet e não apenas as últimas camadas adicionadas, como descrito na subseção 3.2. Colocando os modelos RNNOFT e RNFT lado a lado, observa-se que, mesmo com um aumento no tempo de treinamento, uma melhora na acurácia em teste de 12.65%.

3.4 Aplicação da rede em tempo real

O modelo utilizado para aplicação da rede em tempo real foi o RNFTLRD, uma vez que demonstrou melhor resultado segundo a Tabela 1. As imagens capturadas por vídeo em tempo real foram realizadas através da ferramenta OpenCV (BRADSKI, 2000), que possibilita a captura de um *frame* retangular dentro da imagem. A partir desse *frame*, a rede neural consegue estimar a sua classificação, retornando sua classe e o nível de confiança da classificação. A Figura 26 demonstra a classificação em tempo real juntamente com a imagem da respectiva letra no conjunto de treino. Um ponto que foi entendido empiricamente com a aplicação da rede em tempo real foi que, como esperado, o modelo interpreta a linguagem com uma acurácia maior quando não há objetivos no fundo do *frame*.

Figura 26 – Interpretação em tempo real



Dados do ASL Alphabet Dataset (3.1) e elaboração própria

4 CONCLUSÃO

Neste trabalho, foram comparadas quatro diferentes arquiteturas de redes neurais convolucionais no conjunto de imagens do alfabeto de linguagem de sinais do *dataset* ASL Alphabet Dataset, juntamente com o uso de técnicas que visam a melhora do desempenho da rede neural na tarefa de reconhecimento de imagens, como *fine-tuning*, *learning rate decay* e transferência de aprendizado.

Ao se analisar os dados explicitados nas Tabelas 1 e 2, é possível observar o poder da transferência de aprendizado, tanto em questão de desempenho quanto em tempo de treinamento, ao se comparar os resultados dos três modelos que utilizam da técnica com o modelo C6P2FC2. Quando se é comparado o modelo C6P2FC2 com o RNNOFT (*ResNet with no fine tuning*) observa-se que o segundo, mesmo com todos os parâmetros da rede residual congelados, atinge um resultado próximo do primeiro modelo com um tempo de treinamento 77.26% menor.

A técnica de *fine-tuning* se mostrou muito eficiente. Ao se comparar o modelo RNNOFT (*ResNet with no fine tuning*) com o RNFT (*ResNet fine tunned*) observa-se que, com o descongelamento do último bloco residual da rede, o desempenho em treino foi de 0.9941 para 0.9989 enquanto em teste a melhora foi ainda maior, levando a acurácia de 0.8155 para 0.9187. Essa melhora de desempenho que acompanha o aumento de parâmetros treináveis, 19.234.458 contra 4.263.578, gera, por outro lado, um aumento no tempo de treinamento do modelo de 6h08min para 11h44min.

Por fim, foi incorporado ao modelo a técnica de *learning rate decay* que gerou um aumento na desempenho do modelo RNFTLRD (*ResNet fine tunned with learning rate decay*) quando comparado ao modelo RNFT com apenas um aumento mínimo de tempo de treinamento. A acurácia do modelo foi de 0.9989 para 0.9997 em treino e de 0.9187 para 0.9240 em teste com apenas um minuto a mais de tempo de treinamento.

Após o desenvolvimento e análise dos resultados, o modelo com melhor desempenho, RFFTLRD, foi incorporado a um modelo de aquisição de imagens para a interpretação em tempo real do alfabeto de linguagem de sinais. Todo o desenvolvimento dos modelos juntamente com o programa para interpretação em tempo real podem ser vistos no repositório oficial do projeto no GitHub.

O projeto conseguiu atingir todos os objetivos estipulados na Seção 1.1 e chegou em um modelo de aprendizado de máquina profundo com alta precisão na tarefa de classificação de imagens do alfabeto de linguagem de sinais juntamente com a aplicação em tempo real.

REFERÊNCIAS

ANDRYCHOWICZ, M. et al. Learning to learn by gradient descent by gradient descent. **Advances in neural information processing systems**, v. 29, 2016.

ASTION, M. L.; WILDING, P. The application of backpropagation neural networks to problems in pathology and laboratory medicine. **Archives of pathology & laboratory medicine**, v. 116, n. 10, p. 995–1001, 1992.

BALLARD, D. H. Ballard d. and brown cm 1982 computer vision. **Image**, v. 17, p. 2–2.

BRADSKI, G. The opencv library. **Dr. Dobb's Journal: Software Tools for the Professional Programmer**, Miller Freeman Inc., v. 25, n. 11, p. 120–123, 2000.

BRERETON, A. Sign language use and the appreciation of diversity in hearing classrooms. **Early Years**, Taylor & Francis, v. 28, n. 3, p. 311–324, 2008.

CHOLLET, F. **Deep learning with Python**. [S.l.]: Simon and Schuster, 2021.

FERREIRA, E. V. Gradiente descendente.

GIANESI, B. H. d. C.; ALUíSIO, S. M. **Classificação de gênero via análise de áudio utilizando métodos de aprendizado de máquina tradicionais**. 2021.

GRANDINI, M.; BAGLI, E.; VISANI, G. Metrics for multi-class classification: an overview. **arXiv preprint arXiv:2008.05756**, 2020.

GU, J. et al. Recent advances in convolutional neural networks. **Pattern recognition**, Elsevier, v. 77, p. 354–377, 2018.

HAYKIN, S. **Redes neurais: princípios e prática**. [S.l.]: Bookman Editora, 2001.

HE, K. et al. Deep residual learning for image recognition. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 770–778.

HIEU, N. V.; HIEN, N. L. H. Recognition of plant species using deep convolutional feature extraction. **Int J Emerg Technol**, 2020.

HUK, M. Stochastic optimization of contextual neural networks with rmsprop. In: SPRINGER. **Asian Conference on Intelligent Information and Database Systems**. [S.l.], 2020. p. 343–352.

IESB, C. U.; LIMA, R. T. M. D. O. Arquitetura baseada em redes neurais convolucionais para a detecção automática da covid-19 usando imagens de raio x.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: PMLR. **International conference on machine learning**. [S.l.], 2015. p. 448–456.

KINGMA, D.; BA, J. Dp kingma and j. ba, adam: A method for stochastic optimization. **arXiv preprint arxiv:1412.6980**.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014.

LI, D.; CHENG, X. Application of bp neural network in network fault diagnosis. In: IOP PUBLISHING. **IOP Conference Series: Materials Science and Engineering**. [S.l.], 2019. v. 631, n. 5, p. 052039.

LI, W. et al. Integrating google earth imagery with landsat data to improve 30-m resolution land cover mapping. **Remote Sensing of Environment**, Elsevier, v. 237, p. 111563, 2020.

LIAO, W. et al. Deep transfer learning and time-frequency characteristics-based identification method for structural seismic response. **Frontiers in Built Environment**, v. 7, 02 2021.

MARS Curiosity Rover. <https://mars.nasa.gov/msl/home/>. Accessed: 2022-07-12.

O'SHEA, K.; NASH, R. An introduction to convolutional neural networks. **arXiv preprint arXiv:1511.08458**, 2015.

ÖZGÜR, A.; NAR, F. Effect of dropout layer on classical regression problems. In: IEEE. **2020 28th Signal Processing and Communications Applications Conference (SIU)**. [S.l.], 2020. p. 1–4.

PROVOST, F.; FAWCETT, T. **Data Science for Business: What you need to know about data mining and data-analytic thinking**. [S.l.]: "O'Reilly Media, Inc.", 2013.

RAO, S. S. **Engineering optimization: theory and practice**. [S.l.]: John Wiley & Sons, 2019.

RUDER, S. An overview of gradient descent optimization algorithms. **arXiv preprint arXiv:1609.04747**, 2016.

SHARMA, S.; SHARMA, S.; ATHAIYA, A. Activation functions in neural networks. **towards data science**, v. 6, n. 12, p. 310–316, 2017.

SILVA, E. da et al. Sistema especialista de visão computacional para diagnóstico de fraturas cranianas em imagens de tomografia computadorizada.

TAN, C. et al. A survey on deep transfer learning. In: SPRINGER. **International conference on artificial neural networks**. [S.l.], 2018. p. 270–279.

VYBORNY, C. J.; GIGER, M. L. Computer vision and artificial intelligence in mammography. **AJR. American journal of roentgenology**, Am Roentgen Ray Soc, v. 162, n. 3, p. 699–708, 1994.

YOU, K. et al. How does learning rate decay help modern neural networks? **arXiv preprint arXiv:1908.01878**, 2019.