Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

RECONHECIMENTO FACIAL NO DIA A DIA

LUCAS GRECHI LEME

ORIENTADOR: PROF. DR. RUBENS BARBOSA FILHO

DOURADOS/MS
2022

RECONHECIMENTO FACIAL NO DIA A DIA

LUCAS GRECHI LEME

Este exemplar corresponde ao trabalho de conclusão de curso da disciplina Projeto Final de Curso do aluno Lucas Grechi Leme, que será submetido a avaliação da banca examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação.

ORIENTADOR: PROF. DR. RUBENS BARBOSA FILHO

DOURADOS/MS
2022

L567r Leme, Lucas Grechi
Reconhecimento facial no dia a dia / Lucas Grechi Leme. – Dourados, MS: UEMS, 2022.
52 p.

Monografia (Graduação) – Ciência da Computação – Universidade Estadual de Mato Grosso do Sul, 2022.
Orientador: Prof. Dr. Rubens Barbosa Filho.

1. Inteligência artificial 2. Reconhecimento facial 3. Aprendizagem de máquina I. Barbosa Filho, Rubens. II. Título

CDD 23. ed. - 006.3

Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

RECONHECIMENTO FACIAL NO DIA A DIA

LUCAS GRECHI LEME

Outubro de 2022

Banca Examinadora:

Prof. Dr. Rubens Barbosa Filho (Orientador)
Área de Computação - UEMS

Prof. Dr. Osvaldo Vargas Jaques
Área de Computação - UEMS

Prof. MSc. Alfred Foster Junior
Área de Computação - UEMS

Dedico este trabalho a minha avó Neca,
que foi tão importante para mim, quanto
para esta universidade que me acolheu.

AGRADECIMENTOS

Agradeço a minha mãe, meu pai e meu padrasto, que me apoiaram durante toda minha vida e permitiram que eu me tornasse a pessoa que sou hoje.

Agradeço também ao professor Rubens, meu orientador, que sempre se fez disposto a ajudar nos momentos em que eu falhava.

Gostaria de agradecer a Deus por fazer tudo isso possível.

E por fim, agradecer a mim mesmo por não ter desistido.

RESUMO

Inteligência artificial é uma área de estudo que evoluiu significativamente desde sua invenção em meados dos anos 50 do século XX pelo professor Alan Mathison Turing. Hoje é possível encontrar aplicações desta tecnologia em diferentes situações do cotidiano. O aspecto da inteligência artificial que foi escolhido para análise neste trabalho, refere-se ao reconhecimento facial, utilizando aprendizagem de máquina. Diante desta escolha, o objetivo geral do estudo compreendeu desenvolver um algoritmo capaz de ler milhares de imagens de indivíduos diferentes e depois de receber o treinamento correto, reconhecer a quem pertence cada rosto encontrado em cada imagem. A linguagem de programação utilizada foi Python juntamente com a biblioteca Tensorflow e a tecnologia CUDA. De forma complementar, investigou-se o estado da arte no que se refere ao reconhecimento facial e comparou-se os resultados dos trabalhos pesquisados, com o intuito de entender os atuais limites que podem ser alcançados e visualizar a evolução obtida ao longo dos anos.

Palavras-chave: Inteligência artificial. Visão de computador. CUDA. Softmax. Aprendizagem de máquina. Tensorflow.

SUMÁRIO

1. Introdução

Reconhecimento facial não é um conceito novo, nem mesmo quando é um computador fazendo o reconhecimento. Os primeiros relatos de interesse em desenvolvimento de um sistema de reconhecimento facial automático e computadorizado começaram a surgir no início dos anos 1960, com um engenheiro americano chamado Woodrow Wilson Bledsoe, como descrito por Pandya *et* Cols (2013). Hoje, com quase 60 anos de desenvolvimento e evolução, o conceito de reconhecimento facial está bem definido, com um enorme leque de complexidade e características.

Um sistema de reconhecimento facial pode ser desenvolvido de várias maneiras. Para esse trabalho escolheu-se utilizar a técnica de aprendizado de máquina para o treinamento do sistema, pois trata-se de um método possível de acompanhar a evolução do treinamento algorítmico de forma direta, ou seja, fazendo intervenções quando necessário em tempo de execução. A biblioteca que foi utilizada para fazer o treinamento foi a *TensorFlow*[1], destaca-se que esta biblioteca é *open source* e fornece uma quantidade grande de material de estudo gratuitamente on-line.

Como a intenção desse trabalho é utilizar inteligência artificial (IA), mais especificamente *machine learning* (ML), escolheu-se utilizar um *subset* de ML muito popular na área de reconhecimento facial chamado de *convolutional neural networks* (CNNs). CNN é um tipo de rede neural comumente utilizado para trabalhos envolvendo visão de computador, como, por exemplo, reconhecimento de imagens. CNNs são mecanismos poderosos e requerem muito poder computacional para realizar os treinamentos necessários por um sistema de ML, por essa razão, o processamento do sistema foi realizado em GPU(Graphic Processing Unit) (WANG *et* COLS, 2018).

Outra parte essencial do desenvolvimento de um sistema de reconhecimento facial é o cálculo da camada de perda. A camada de perda é a função responsável por fazer a discriminação das diferentes faces, registrando pontos de referência em cada classe e comparando-os com outros, até encontrar uma face que esteja dentro dos parâmetros aceitos. O método para o desenvolvimento da camada de perda escolhido para ser utilizado nesse projeto foi o *Softmax*, originalmente creditado a John S. Bridle (GAO *et* COLS, 2018). Esse método foi escolhido por ser a base de todos os outros métodos desenvolvidos posteriormente e requer menor poder computacional, sacrificando um pouco da qualidade dos resultados finais. A fórmula para essa e outras formas de cálculos da camada de perda estão disponíveis na seção 2.

Assim como muitas outras tecnologias, o reconhecimento facial se torna mais presente em nossas vidas a cada dia que passa. Desde segurança e defesa, com câmeras que estão sempre alertas às pessoas procuradas pela justiça ou com histórico de causadoras de problemas (BENNETT, 2001), até marketing (WU *et* COLS, 2015), pois, é possível utilizar reconhecimento

[1] Fonte: https://www.tensorflow.org . Acesso em : 15 ago, 2021.

facial para identificar idade, gênero e outras características que possam ajudar a criar propagandas mais personalizadas para cada cliente.

Na vida pessoal, a tecnologia de reconhecimento facial já começou a ser inserida em objetos como celulares, e não é mais necessário utilizar uma senha de números ou padrões, proporcionando acesso às suas funcionalidades apenas para aqueles que o aparelho reconhecer como seu dono e é apenas uma questão de tempo até que essa facilidade se expanda para outros meios, como computadores ou até mesmo substituindo as chaves para as casas (SYAFEEZA *et* COLS, 2020).

Desta forma, o objetivo geral deste estudo foi desenvolver e implementar um algoritmo que seja capaz de realizar a função de reconhecimento facial, de tal maneira que seja aplicável às situações do dia-a-dia. Para alcançar essa meta, realizou-se um estudo do funcionamento de softwares de reconhecimento facial. Foram analisados diferentes métodos para o desenvolvimento desse sistema e realizados testes investigativos. Fizemos também uma análise dos dados obtidos pelos trabalhos analisados para que seja possível obter uma linha de evolução da tecnologia nos últimos anos.

Levando em consideração a importância e o impacto que essa tecnologia já tem, esse trabalho pode contribuir com pesquisas futuras na área de reconhecimento facial, *ML* e *Internet of things* (IoT). Quando se trata de tecnologia, a procura por mais informação é constante e com isso em mente, este estudo justifica-se, pois amplia o conhecimento disponível para análise.

O trabalho está dividido em seções. Na seção 2 discutiu-se sobre o estado da arte, falou-se sobre o desenvolvimento do assunto ao longo dos últimos anos, quais os métodos mais populares e para qual propósito eles foram desenvolvidos, assim como as características atuais dos modelos de reconhecimento facial. Na seção 3 apresentou-se as ferramentas e equipamentos que foram utilizados no desenvolvimento deste trabalho, desde linguagens de programação escolhidas, bibliotecas, *Applications Programming Interface* (APIs) utilizadas e características do equipamento que fizeram parte do processo de treinamento e aplicação do sistema. Em seguida, na seção 4, seguimos com a descrição da metodologia utilizada para o desenvolvimento do *software*, na qual explicou-se como foi o processo de criação do sistema, além de descrever os resultados obtidos, e comparação entre os sistemas estudados. Finalmente, na seção 5, são apresentadas as conclusões alcançadas durante o desenvolvimento e testes do sistema.

2. Estado da arte

Reconhecimento facial é uma tecnologia com grande potencial de aplicação tanto comercialmente quanto cientificamente, a qual desperta a atenção dos pesquisadores e desenvolvedores para novos avanços. Com o avanço de técnicas como aprendizagem de máquina e reconhecimento facial profundo. tornou-se possível acelerar a forma como o computador discrimina faces umas das outras, como descrito por Parkhi et Cols (2015), fazendo com que o uso dessas tecnologias se torne algo cada vez mais presente no dia a dia. Na revisão teórica a seguir, são apresentados os conceitos de: reconhecimento facial profundo e aprendizagem de máquina, CNN e camada de perda.

2.1 Reconhecimento Facial e aprendizagem de máquina

Mais comumente conhecido por seu nome em inglês, *deep face recognition* é uma ferramenta que apareceu nos últimos anos e possibilitou um avanço imprescindível na área de reconhecimento facial. Parkhi *et* Cols (2015) explicam que, anteriormente ao desenvolvimento dessa ferramenta, as bases de dados para criação de *softwares* de treino de reconhecimento facial eram muito limitadas em relação ao tamanho que elas eram capazes de alcançar. Em 2015, por exemplo, o *Google* desenvolveu um sistema de reconhecimento facial utilizando uma base de dados com mais de duzentos milhões de imagens (SCHROFF *et* COLS, 2015). Até então, uma base de dados desse tamanho era inimaginável para outros pesquisadores.

Outro aspecto importante referente à inteligência artificial, que tem relação com reconhecimento facial, é a ML. A ML é a capacidade de fazer com que o computador aprenda através de experiência. Fazendo uso de cálculos matemáticos e estatística, a ML consegue fazer a leitura de informações que lhe são dadas, transformando-as nos mais diversos tipos de informações. Hoje, aprendizagem de máquina é um dos campos mais estudados da inteligência artificial e devido ao seu alto desenvolvimento e popularidade, tornou-se uma das ferramentas mais aplicadas no dia a dia. Abraham (2005) descreve como essa tecnologia se infiltrou nos mais diversos aspectos da vida humana, como, por exemplo, filtros de *e-mails* e cupons de desconto.

Quando se trata de reconhecimento facial, a ML fornece uma variedade de opções que permitem reconhecer uma determinada face. Algumas das ferramentas mais populares são:

- *Scale-invariant feature transform* (SIFT), que foca na busca de imagens em outras imagens como, por exemplo, uma imagem de um galho em um parque, independente de zoom ou grau de rotação da mesma;
- *Random sample consensus* (RANSAC), que se especializa em encontrar imagens relacionadas ao mesmo local ou objeto, fazendo a fusão das mesmas e criando uma nova imagem de melhor qualidade, como, por exemplo, a fusão de uma imagem tirada por satélite com imagens menores tiradas por *drones*, fornecendo mais detalhamento.

Além dos exemplos acima, Singh (2019) explora o assunto abordando a forma como é feita a implementação e como são utilizados esses e outros algoritmos de processamento de imagens, incluindo os modelos que foram utilizados neste estudo, redes neurais e CNN, as quais serão melhor explicados na próxima seção.

2.2 Convolutional Neural Network

Redes neurais artificiais são um modelo de resolução de problemas desenvolvido com o intuito de encontrar uma maneira mais fácil, rápida e eficiente de solucionar problemas relacionados com reconhecimento de padrões, previsões, controle de dados e eficiência. Muitas vezes inspirada em redes neurais biológicas, redes neurais artificiais tem como objetivo adquirir algumas das melhores características da versão em que se baseia, como a habilidade de aprender, capacidade de generalizar, adaptabilidade e tolerância a falhas, como descrito por Jain et Cols (1996). Uma rede neural artificial normalmente pode ser dividida em 3 camadas: a camada de entrada, a camada de saída e, entre elas, a camada escondida, ou *hidden* (ABRAHAM, 2005). Nesta camada *hidden* os valores e cálculos desejados são processados e resolvidos. No caso de reconhecimento facial feito com CNN, essa camada é chamada de camada de perda.

CNN se diferencia de redes neurais artificiais normais de algumas maneiras, a principal é referente ao tamanho e quantidade dos dados com os quais ela é capaz de trabalhar. Como descrito por Krizhevsky *et* Cols (2012), CNNs são capazes de trabalhar com milhões de objetos simultaneamente e apresenta uma maior facilidade de treinamento, sacrificando apenas uma pequena porcentagem de sua performance para alcançar esses resultados. Tudo isso, porque possuem uma menor quantidade de conexões e parâmetros.

Porém, apesar de ser uma ferramenta extremamente poderosa, as CNNs também são muito complicadas de se trabalhar. A quantidade de variáveis que são utilizadas nos cálculos junto com o volume de neurônios envolvidos na realização dessas operações eleva a complexidade dessas redes neurais. Gu *et* Cols (2017) explicam que, semelhante a redes neurais artificiais normais, CNN possuem, no geral, três camadas: *convolutional,* que é responsável por aprender as características apresentadas nas imagens de entrada; *pooling*, responsável por reduzir invariância de dados na movimentação entre um camada e outra; e, por fim, *fully-connected layer*, que performa os raciocínios finais do sistema.

Gu *et* Cols (2017) aprofundam o raciocínio, explicando que a *convolutional layer* é composta por uma convolução de kernels, que são usados para computar os diferentes mapas de características. Mais especificamente, cada neurônio de um mapa de características é conectado a uma região de neurônios vizinhos em uma camada anterior, essa região de neurônios é conhecida como área receptiva de um neurônio na camada anterior. O mapa de características completo é alcançado através do uso de uma série de diferentes kernels nos cálculos. Gu *et* Cols (2017) fornecem o seguinte cálculo matemático para descobrir o valor de uma característica $Z^{L}_{I,J,K}$, onde (I, J) representam a posição da variável, K representa o mapa de características onde ela é encontrada e L a camada na qual esse mapa se encontra:

$$Z^{L}_{I,J,K} = W^{L}_{K} X^{L}_{I,J} + B^{L}_{K} \quad (1)$$

Nessa formula W^{L}_{K} representa o vetor de peso (*weight*), $X^{L}_{I,J}$ é o valor de entrada na posição (I, J) e B^{L}_{K} é o termo de tendência (*bias*).

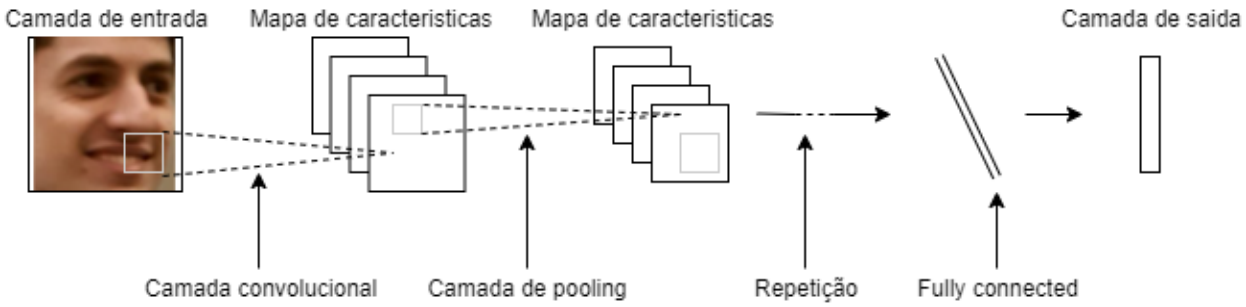
A camada de *pooling* tem como objetivo diminuir a invariância de dados na movimentação entre duas camadas, através da redução do tamanho da resolução dos mapas de características, normalmente, encontrando-se entre duas camadas convolucionais. Cada mapa de características de uma camada de *pooling* está conectado a seu mapa correspondente na camada convolucional anterior.

De acordo com Wang *et* Cols (2012), existem duas principais maneiras de implementar camadas de *pooling*. Primeiro, a camada de *pooling* pega um mapa de características da camada convolucional anterior, e o divide em blocos de tamanho m X n. Em seguida, a função de *pooling* é passada por cada bloco. Esse processo retorna um mapa para cada bloco, com dimensões m X n. Nesse ponto é possível escolher qual modo de *pooling* se deseja utilizar, *max polling* ou *average pooling.* No caso de *max pooling*, o valor de retorno de cada bloco é o valor mais alto do grupo, enquanto no *average pooling* o valor de retorno se torna a média dos valores do bloco.

Seguindo várias iterações de camadas de convolução e *pooling*, é possível ter uma ou mais *fully-connected layers*. Para conseguir realizar o raciocínio de alto nível desejado dessa camada, a camada utiliza todos os neurônios da camada anterior a ela e os conecta com todos os neurônios da camada atual, gerando informações semânticas globais.

Por fim, temos a última camada de um CNN, a camada de saída. Para *softwares* de classificação, como *softwares* de reconhecimento facial, uma operação como softmax é utilizada, mas existem outros métodos que também podem ser utilizados para diferentes tipos de problemas ou modelos de classificação específicos. Um modelo da estrutura de um CNN pode ser visto na figura 1.

Figura 1 - modelo de estrutura de CNN



Fonte: o autor, adaptado de GU *et* Cols (2017)

Em anos recentes, avanços tecnológicos como placas gráficas mais poderosas, permitiram aos pesquisadores aprofundarem os conceitos e aplicações utilizando CNNs no estudo das redes neurais. Gu *et* Cols (2017) descreveram em detalhes as diferentes partes relacionadas com a CNN, sendo: a camada de perda; otimização; rápido processamento; aplicação e normalização.

2.3 Camada de perda

Essa área de estudo se tornou muito popular entre os anos de 2016 e 2021, criando uma alta variedade de estudos desenvolvidos por diversas pessoas. A maneira como elas lidam com a função de perda, por exemplo, possui um leque de alternativas para se escolher. Softmax é a forma mais simples onde as pessoas se baseiam para projetar suas próprias funções de perda. Usando Softmax como base, Weiyang Liu *et* Cols (2016) desenvolveram o que eles chamaram de *large-margin softmax (L-Softmax),* com foco em encorajar discriminação entre diferente classes e compactação de informações dentro de cada classe. O cálculo do Softmax é dada pela seguinte fórmula:

$$L = \frac{1}{N}\sum_{i}^{N} - log\left(\frac{e^{f_{yi}}}{\sum_{j} e^{f_j}}\right) \quad (2)$$

onde f_j representa a j-ésima classe, N é o número de informações de treino.

Com as alterações feitas na fórmula original, a equipe de Weiyang Liu (2016) chegou na seguinte fórmula, onde W_{yi} representa a yi-ésima coluna da camada W, x_i representa os valores de entrada na camada W, $\psi\left(\theta_{yi}\right)$ é o ângulo de diferença entre o vetor W_{yi} e x_i, onde $\psi(\theta) = cos(m\theta)$, se $0 <= \theta <= \pi/m$ ou $\psi(\theta) = D(\theta)$, se $\pi/m < \theta <= \pi$, onde m é um valor próximo da margem de classificação e D(θ) é uma função monotonicamente decrescente, onde $D(\pi/m) = \cos(\pi/m)$:

$$L_i = -\log\left(\frac{e^{||W_{yi}||\,||x_i||\,\psi(\Theta_{yi})}}{e^{||W_{yi}||\,||x_i||\,\psi(\Theta_{yi})} + \sum_{j \neq y} e^{||W_j||\,||x_i||\,cos(\Theta_j)}}\right) \quad (3)$$

Um ano depois, Weiyang Liu *et* Cols (2017) desenvolveram um outro método para calcular a camada de perda, o *angular softmax(A-Softmax)*, que introduz a ideia de ensinar ao CNN como reconhecer características angularmente discriminativas, com o propósito de resolver o problema, pois as características faciais possuem uma maior distância máxima dentro de cada classe do que a distância mínima entre outras classes. A fórmula alcançada para o cálculo de A-Softmax foi:

$$L_A = \frac{1}{N}\sum_i^N -\log\left(\frac{e^{||x_i||\,cos(\Theta_{yi,i})}}{e^{||x_i||cos(\Theta_{yi,i})} + \sum_j e^{||x_i||\,cos(\Theta_{j,i})}}\right) \quad (4)$$

Em 2018, baseando-se nos trabalhos de Weiyang Liu *et* Cols (2016) e Weiyang Liu *et* Cols (2017), Feng Wang *et* Cols (2018) desenvolveram, com o intuito de criar um sistema mais intuitivo e interpretável do que os anteriores, o *additive margin Softmax (AM-Softmax)*, enfatizando a normalização de características faciais. Através de testes, concluíram que o AM-Softmax possui, em média, uma performance melhor do que os dois projetos anteriormente desenvolvidos. O cálculo para função de perda do AM-Softmax, em que s é um parâmetro de escalabilidade e controle da magnitude do raio, e C é a classe em questão, é dado por:

$$L_{AM} = -\frac{1}{N}\sum_{i=1}^{N} \log\left(\frac{e^{s\cdot\left(W_{yi}^T f_i - m\right)}}{e^{s\cdot\left(W_{yi}^T f_i - m\right)} + \sum_{j=1, j\neq yi}^{c} e^{s\cdot\left(W_j^T fi\right)}}\right) \quad (5)$$

Todas as alternativas para Softmax apresentadas até esse ponto foram criadas com o mesmo método em mente, **maximizar a variância dentro de cada classe e minimizar a variância entre diferentes classes**. Porém Hao Wang *et* Cols (2018) propuseram um método novo para calcular discriminação, chamado de *large margin cosine loss*(LMCL), conhecido como *CosFace*, que faz alterações a maneira como Softmax calcula a perda, reformulando-o com o uso de cossenos para o cálculo de espaço angular e normalizando características e pesos dos vetores utilizados, para remover possíveis variações radiais. A fórmula para o LMCL foi definida como:

$$L_{LMCL} = \frac{1}{N}\sum_i^N -\log\left(\frac{e^{s\cdot\left(cos\left(\Theta_{yi,i}\right) - m\right)}}{e^{s\cdot\left(cos\left(\Theta_{yi,i}\right) - m\right)} + \sum_{j\neq yi} e^{s\cdot\left(cos\left(\Theta_{j,i}\right)\right)}}\right) \quad (6)$$

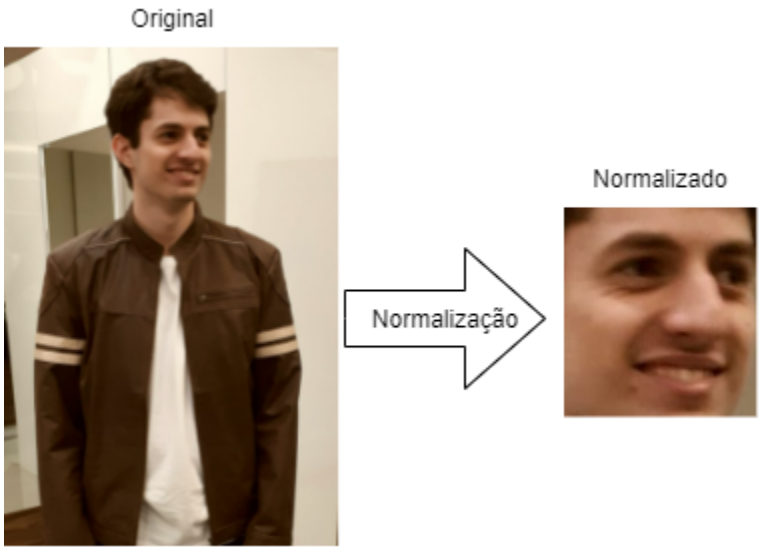
Jiankang Deng *et* Cols (2019), em 2019, escolheram analisar o problema de perda por outro ângulo. Para alcançar uma capacidade discriminatória ainda maior do que as previamente obtidas, uma nova dimensão teve que ser adicionada ao cálculo. *Additive Angular Margin Loss(ArcFace)*, como veio a ser chamada, faz o uso de *Geodesic Distance(GDis)* para calcular a camada de perda e conseguir aumentar o poder discriminativo do modelo de reconhecimento facial. A fórmula alcançada por Jiankang foi a seguinte:

$$L_{Arc} = -\frac{1}{N}\sum_{i=1}^{N} log\left(\frac{e^{s\cdot(cos(\Theta_{yi}))}}{e^{s\cdot(cos(\Theta_{yi}))} + \sum_{j=1, j\neq yi}^{C} e^{s\cdot(cos(\Theta_{j}))}}\right) \quad (7)$$

2.4 Normalização

Normalização é uma das técnicas mais utilizadas quando se está trabalhando com os mais diferentes tipos de dados. Sua função é a de preparar os dados que serão utilizados em um sistema. Em estatística, o ato de normalizar os valores trabalhados se faz para que seja possível trabalhar com todos os valores em um mesmo gráfico. De modo similar, quando se normaliza os dados em um sistema de ML, isso é feito com o objetivo de facilitar o processo de treinamento de um dado sistema, com o intuito de acelerar o processo. A normalização pode tomar diferentes formas dependendo do sistema em que ela está trabalhando. No caso de um sistema de reconhecimento facial utilizando CNN, a normalização é feita de modo que as imagens que são envolvidas na fase de treino do sistema, antes de serem computadas, são modificadas para que todas as imagens tenham o mesmo tamanho e mostrem apenas o rosto da pessoa em questão. A figura 2, representa o resultado do processo de normalização.

Figura 2 - resultado do processo de normalização



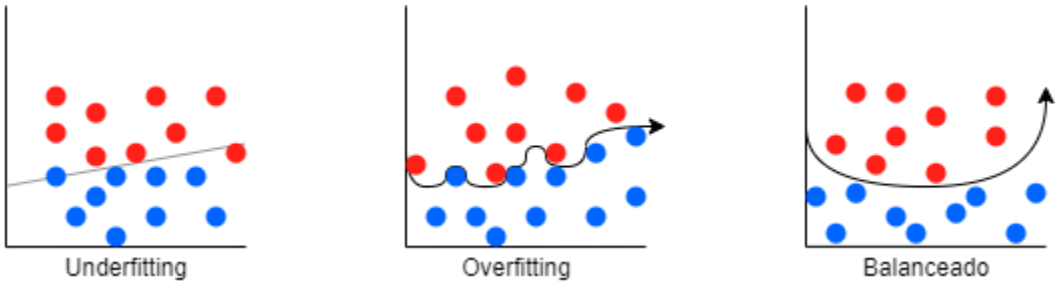
Fonte: o autor adaptado de SHAN DU E RABAD WARD (2005)

Esse método não é a única forma de normalização que existe quando se trata de reconhecimento facial, ou trabalho com imagens no geral. Shan Du e Rabab Ward (2005) descrevem um modelo de normalização envolvendo trabalho com iluminação das imagens. Adini *et* Cols (1997) demonstraram como a alteração da iluminação de uma foto pode causar variações aos detalhes da imagem muitas vezes maiores do que a alteração do indivíduo que esteja na imagem Por essa razão, a normalização de atributos com luz, ângulo e formato da imagem é de extrema importância quando se trabalha com o aprendizado de máquina para uma CNN.

Normalização é uma técnica poderosa capaz de alterar drasticamente os valores de treino de um sistema e por consequência, os resultados. Devido a isso é preciso cautela quando se trabalha com ela, pois uma normalização mal feita pode gerar muitos problemas para o sistema. Zhi Kou *et* Cols (2020) descrevem o *overfitting* e *underfitting* como um dos problemas mais comuns que podem ser causados por normalizações.

Overfitting e *underfitting* são termos que descrevem problemas referentes à ML, representando, respectivamente, o problema de que a rede neural se torna muito dependente das características aprendidas, não sendo capaz de generalizar os dados que a foram alimentados, causando um número menor do que o esperado de faces reconhecidas. Já o *underfitting* seria quando a rede neural não é muito bem treinada, fazendo com que o sistema retorne uma quantidade elevada de falso-positivos. A figura 3 mostra uma representação gráfica dos problemas de *overfitting* e *underfitting* quando comparados com um sistema que foi corretamente balanceado.

Figura 3 - representação gráfica dos problemas de *overfitting* e *underfitting*



Fonte: o autor, a partir de JABBAR *et* Cols (2015)

Para alcançar um resultado mais desejado e o mais balanceado possível, existe uma variedade de técnicas que foram desenvolvidas ao longo dos anos para tratarem diversos cenários diferentes como, por exemplo, o *Batch Normalization,* desenvolvido por Ioffe e Szegedy (2015), que tem o propósito de acelerar o processo de treinamento de uma rede neural. O problema abordado pelo *Batch Normalization* é o de que, quando se faz o treinamento de uma rede neural, a distribuição dos dados de entrada muda de camada para camada, juntamente com os parâmetros das camadas anteriores, diminuindo a velocidade do processo de treino.

O sistema de normalização criado por Ioffe e Szegedy(2015) fornece uma solução para esses problemas na forma de uma arquitetura que inclui normalização como uma parte integral do sistema, possibilitando o acontecimento de normalização dos dados para cada grupo, ou

batch, que está sendo usado para o treino no momento. Isso permitiu que fosse alcançado uma velocidade de aprendizado do sistema muito maior e em alguns casos, a eliminação de *Dropout,* que por sua vez, é uma técnica desenvolvida por Srivastava *et* Cols (2014) justamente com o propósito de tratar casos de *overfitting* através da medida de derrubar unidades e suas conexões, da rede neural, prevenindo que as unidades co-adaptem muito. A desvantagem de se usar *Dropout* é que essa técnica faz com que a fase de treinamento se torne muito longa, chegando a durar de duas a três vezes mais do que outras formas de treino.

Mas *Batch Normalization* não é a única alternativa para normalização disponível. Baseando-se nos trabalhos de Ioffe e Szegedy, Cooijmans *et* Cols (2017) desenvolveram o *Recurrent Batch Normalization,* que tem como objetivo levar os benefícios de *batch normalization* para *recurrent neural networks.* Eles alcançaram isso porque viram a possibilidade de aplicar normalização em *batch* não apenas nas transformações de entrada para *hidden,* como originalmente proposto por Ioffe e Szegedy, mas também nas transformações de *hidden* para *hidden.* Cooijmans et Cols (2017) concluíram que a utilização de *batch normalization* em *recurrent neural networks* é não apenas possível, mas, com uma inicialização adequada, pode ser ainda mais eficiente do que outros métodos testados até então.

Outro método possível de se usar para fazer a normalização é o *group normalization,* desenvolvido por Wu e He (2018), com o propósito de fornecer uma alternativa mais viável do que *batch normalization* quando se está trabalhando com grupos de tamanhos menores. Eles chegaram à conclusão que o sistema pode ter uma performance até 10% melhor do que o modelo original de Ioffe e Szegedy, em condições ideais. Porém, devido a popularidade alcançada pelo modelo *batch,* muitos sistemas atuais são modelados para trabalhar com método de Ioffe e Szegedy, criando a necessidade de remodelagem do sistema que queira utilizar o *group normalization.*

Em 2017, Ulyanov *et* Cols (2017), baseando-se no trabalho de Ioffe e Szegedy(2015), desenvolveram um novo método de normalização chamado de *instance normalization,* esse método foi criado com o propósito específico de acelerar o processo de *fast stylization* que eles mesmos desenvolveram em 2016. O processo de *fast stylization* é basicamente ensinar uma rede neural a observar uma imagem x, observar outra imagem y e redesenhar a imagem x no estilo da imagem y. Originalmente Ulyanov et Cols (2016) utilizaram o modelo *batch normalization* de Ioffe e Szegedy(2015), mas, se depararam com um problema na performance do programa. Para contornar esse desafio eles decidiram substituir o método de normalização pelo *instance normalization* e descobriram que é possível obter um enorme melhoramento em redes neurais usadas para geração de imagens.

3. Ferramentas

3.1 Linguagem Python

A linguagem de script Python é uma linguagem de alto nível de propósito geral. Sendo de alto nível, Python possui um grande poder computacional, além de fornecer um código mais amigável e legível do que outras linguagens de nível médio, como C ou C++. Python também é uma linguagem orientada a objetos, porém, diferentemente de Java ou Ruby, inclui características de linguagens funcionais como Lisp e Haskell (CHUN, 2006).

Uma das maiores vantagens de trabalhar com Python é a existência de muitas funções e bibliotecas prontas para o uso, facilitando o desenvolvimento de várias e diferentes funcionalidades, como, por exemplo, a biblioteca TensorFlow[2] que foi utilizada nesse projeto. Além disso, a popularidade da linguagem Python, número 1 na lista das 10 linguagens de programação para se aprender em 2021 da Northeastern University (EASTWOOD, 2020), proporciona um enorme volume de conteúdo desenvolvido por outros usuários, disponíveis na internet livre de custos, auxiliando programadores de todos os níveis de experiência.

O motivo da escolha da linguagem Python para esse projeto se deu principalmente pelos motivos listados acima. O fato de ser uma linguagem de alto nível propicia um desenvolvimento mais fácil do código, a popularidade fornece um volume considerável de material gratuito na internet para análise, comparação e aprendizado. Além disso, a linguagem possui acesso a bibliotecas e APIs que há anos já vem sendo desenvolvidas para os mais diversos motivos e que muitas dessas comportam as necessidades que existem quando se desenvolve um projeto como esse.

3.2 TensorFlow

TensorFlow é uma biblioteca completamente *open source* feita para o desenvolvimento de ML, com ênfase em *deep learning,* utilizada, principalmente, para realizar treinamento de modelos com redes neurais. Essa biblioteca é equipada para lidar com programas de todos os tamanhos, sendo capaz de trabalhar em conjunto com diversas APIs diferentes, proporcionando uma poderosa ferramenta no estudo e desenvolvimento de qualquer projeto relacionado à aprendizagem de máquina (TENSORFLOW, 2021). Trata-se de uma ferramenta flexível, sendo possível utilizá-la em JavaScript ou Python; sendo esta a linguagem escolhida para este projeto. Com essa biblioteca é possível utilizar a GPU da máquina para acelerar a realização dos treinos. O site oficial da plataforma oferece tutoriais de como utilizá-la adequadamente e uma documentação extensa, além de uma comunidade de desenvolvedores que também pode oferecer auxílio.

[2] Fonte: https://www.tensorflow.org . Acesso em : 15 ago, 2021.

Foi escolhido usar o TensorFlow como biblioteca para esse projeto pois, como descrito anteriormente, ela é completamente gratuita, diferentemente de outras alternativas como por exemplo, Keras, que apesar de ser outra ferramenta muito poderosa é limitada em sua versão gratuita. Outro motivo para a escolha dessa biblioteca é a extensividade do material de apoio disponível online, que proporciona um aprendizado acelerado do uso e detalhes da ferramenta.

3.3 Matplotlib

O Matplotlib é uma ferramenta utilizada para criação e edição de imagens em tempo real para uso em Python. Uma biblioteca capaz de criar visualizações estáticas, animadas e interativas. Utilizando essa ferramenta é possível demonstrar os pontos faciais importantes que são gravados pelo programa para reconhecer e comparar as faces. Matplotlib é patrocinado pela NumFOCUS, uma companhia sem fins lucrativos americana, e por isso é open source, disponibilizando uma extensa documentação on-line, que é fácil de aprender.

Apesar de existir alternativas para essa biblioteca, como por exemplo o CV2, Matplotlib é uma das ferramentas mais utilizadas e conhecidas da linguagem Python. Possui uma alta flexibilidade das coisas que é capaz de fazer, o que gerou um grande número de seguidores na internet. Isso faz com que o Matplotlib possua um alto volume de material gratuitamente disponível, além de ter uma excelente sincronia com Python, sendo esta a razão pela qual foi escolhida para ser utilizada nesse projeto.

3.4 Características do Computador

Para os testes realizados neste projeto será utilizado um computador pessoal, com o processamento sendo realizado em uma GeForce 1060 com 6GB e, sistema operacional Windows 10 Pro de 64-bit.

3.5 Instalação do Windows, CUDA, Tensorflow e CUDNN

3.5.1 Windows

A instalação do Windows é igual a qualquer outro sistema operacional. O primeiro passo é fazer o download do sistema operacional (https://www.microsoft.com/pt-br/software-download/windows10). O próximo passo é criar um *bootable USB* onde o sistema operacional será inserido, isso pode ser feito em qualquer dispositivo USB, contanto que ele possua, ao menos, 8GB de memória. É importante lembrar que qualquer outros itens salvos dentro do USB serão deletados no processo. Com o *bootable USB* criado, agora é necessário reiniciar a máquina com o USB inserido e escolher a opção para bootar pelo USB, em seguida, basta seguir as instruções do instalador e escolher as preferenciais.

O Windows foi o SO escolhido para a realização deste trabalho por vários motivos, incluindo facilidade de acesso de se trabalhar com o programa. Porém a principal razão da escolha desse sistema operacional ao invés de outro foi para a instalação das outras ferramentas que também foram utilizadas. Em particular a ferramenta CUDA, que vamos explicar em seguida, junto com a ferramenta CuDNN, são bem específicas em relação a quais versões são compatíveis umas com as outras. O Windows fornece a oportunidade de se trabalhar com o aplicativos como anaconda navigator e GeForce Experience, que auxiliam na instalação das versões e drivers corretos.

3.5.2 CUDA

CUDA *toolkit* é uma ferramenta desenvolvida pela Nvidia com o objetivo de possibilitar que o processamento de informação seja realizado pela placa de vídeo do computador ao invés do processador. Antes de começar o processo de instalação do CUDA é importante ter em mente que não são todas as placas de vídeo que são capazes de utilizar essa ferramenta Por ser uma ferramenta desenvolvida pela Nvidia, apenas placas de vídeo da mesma são capazes de utilizá-la Além disso, de todas as placas de vídeo produzidas pela Nvidia, só algumas tem a tecnologia necessária para o uso. Uma lista com todas as placas de vídeo com capacidade de utilizar CUDA está presente no seguinte link: (https://developer.nvidia.com/cuda-gpus).

Depois de consultar o *link* e verificar se a placa de vídeo instalada é compatível com o CUDA, pode-se baixar o CUDA do próprio site da Nvidia, no link: (https://developer.nvidia.com/cuda-toolkit-archive). Lá haverá todas as versões disponíveis para download, é importante lembrar qual versão será escolhida, pois é preciso que haja compatibilidade entre as versões do CUDA, CuDNN e dos drives da placa de vídeo. Também é importante ter em mente que o CUDA é apenas compatível com os drivers oficiais da Nvidia.

Caso deseje instalar o CUDA no Linux é possível fazer isso através do comando:

$ sudo apt install nvidia-cuda-toolkit

Depois de instalado, é possível verificar a instalação e a versão instalada utilizando o comando:

$ nvcc -V

3.5.3 CuDNN

CuDNN é uma sigla que significa CUDA *Deep Neural Network*. Ela é uma biblioteca acelerada por GPU do computador feita especificamente para trabalhar com CUDA em redes neurais profundas. Implementada para trabalhar com rotinas específicas como convolução, *pooling*, normalização e camadas de ativação, fica claro que essa é uma biblioteca altamente necessária para um projeto como este. CuDNN pode ser baixada no site da própria Nvidia, no link: (https://developer.nvidia.com/cudnn).

4. Metodologia

A primeira parte da metodologia que foi utilizada nesse projeto se deu na forma de estudo e análise do estado da arte, com o intuito de apreender e entender como este assunto está sendo tratado por aqueles profissionais que se destacam nos estudos deste tópico. Foram ainda estudadas diferentes maneiras de encarar os problemas e dificuldades encontradas quando se trabalha com esses conceitos, bem como as diferentes maneiras de utilizar as ferramentas para se alcançar o resultado final, qual a melhor maneira de preparar o programa para usar a menor quantidade de tempo e recursos da máquina. Para esses tópicos foi utilizado o livro "*Core Python Programming*. (CHUN, 2006)", esse livro foi escolhido pois apresenta uma didática simples e direta para o aprendizado inicial da linguagem de programação. Para os tópicos mais voltados especificamente para o estudo de reconhecimento facial, aprendizagem de máquina e redes neurais, foram lidos diversos artigos e periódicos citados nas referências.

A parte prática do projeto se dividiu em três partes. A primeira parte foi o processamento das imagens, nesta parte todas as imagens que serão utilizadas pelo sistema, tanto para o treino quanto para os testes, passam pelo processo de normalização, uniformizando o tamanho das imagens para 250 X 250 pixels, centralizando os rostos e marcando os 5 pontos faciais que servirão como pontos de comparação, sendo dois pontos nos olhos, um no nariz e dois pontos na boca, um em cada canto. Essa técnica foi desenvolvida por Yi Sun(2013), em seu trabalho sobre detecção facial.

Para realizar essa tarefa usaremos funções disponíveis no python através das bibliotecas cv2, numpy e matplotlib. Cada uma dessas bibliotecas e funções serão explicadas em maior detalhe quando discutirmos o código do programa, mas envolve uma sequência de funções que são usadas para transformar as imagens como nós vemos em um array de números que o computador é capaz de ler e compreender.

Com as imagens preparadas para o uso, podemos dar início a segunda fase, a de treinamento. Nessa primeira etapa, o computador é apresentado a uma lista de imagens com suas classes já definidas. A máquina vai então ler todas essas imagens, aprendendo quais características definem cada classe, e com isso aprenderá a apartar diferentes faces. Essa fase é a que possui a maior quantidade de imagens. Para garantir um resultado adequado, as imagens da base de dados utilizada são separadas em treinamento e teste. A fase de treinamento possui em torno de 80% da quantidade total de imagens disponíveis em cada classe, e a fase de teste com os 20% restantes.

Posteriormente, voltou-se para a parte de treinamento do programa. Para o treinamento, utilizou-se a base de dados CASIA - WebFace[3], essa base de dados foi escolhida pela sua

[3] Fonte: https://paperswithcode.com/dataset/casia-webface#:~:text=The%20CASIA-WebFace%20dataset%20is,Mask%20using%20Multi-scale%20GANs . Acesso em 15 ago, 2021

amplitude, possuindo quase 500.000 imagens de mais de 10.000 pessoas, além do fato de ser completamente gratuita. A fase de teste, na qual novas imagens são testadas pelo programa, que irá realizar o cálculo da distância entre as características que foram aprendidas durante a fase de treino. Esse valor representa a pontuação de similaridade, seguida da identificação facial acontece através de cálculos e comparações envolvendo as pontuações de similaridade e limites de diferença.

Além de comparar o desempenho do sistema a partir das alterações, de forma a descobrir o modo no qual ele teria melhor performance, também foi feita a comparação dos resultados obtidos por outros pesquisadores com o propósito de descobrir qual dos modelos possui os melhores resultados. Os modelos comparados foram os listados na seção 2.3, onde descrevemos as diferenças matemáticas em cada um deles. O resultado dessas comparações será descrito na seção 5.2.

Após a finalização da implementação e dos testes do sistema, foi feita a escrita do trabalho, apresentando o tema com uma breve história de seu surgimento, como se desenvolveu ao longo dos anos, o impacto que teve na sociedade até os dias de hoje e possíveis impactos que terá no futuro. Além disso, apresentaram-se informações sobre as partes que dão suporte a um sistema como esse, demonstrando o estado da arte das áreas relacionadas com o projeto.

5. Desenvolvimento e Resultados

Este item do trabalho detalhou o programa desenvolvido, descrevendo a funcionalidade de cada linha do código. A seguir inicia-se a apresentação do código:

```
import os
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import cv2
import matplotlib.pyplot as plt
from matplotlib.image import imread
```

Nas duas primeiras células do código realizamos as importações das bibliotecas mais básicas que serão utilizadas. A biblioteca “os” é utilizada para que o programa possa acessar os arquivos salvos na máquina. “Pandas” é uma biblioteca usada para leitura de dados e criação e organização de datasets. “Numpy” é a biblioteca padrão do python para trabalho com arrays. A biblioteca “seaborn” é utilizada para criação e visualização de gráficos e imagens. “Cv2” é uma biblioteca com funções de edição de imagem. A biblioteca “matplotlib.pyplot” permite que o python trabalhe com imagens e gráficos. “Imread” é a biblioteca que permite a visualização de imagens no código.

```
data_dir = 'C://Users//Lucas//Desktop//tcc_test'
```

```
test_path = data_dir+'//test'
train_path = data_dir+'//train'
```

Logo em seguida, definimos as variáveis que representam o caminho dos arquivos no computador que serão usados, indicando o local da pasta que contém as imagens que serão usadas para o treino do sistema e as imagens para os testes.

```
1  def create_dataset(data_dir) :
2      img_data_array=[]
3      class_name=[]
4
5      for dir1 in os.listdir(data_dir):
6          for filename in os.listdir(data_dir + '//' + dir1):
7              image_path = data_dir + '//' + dir1 + '//' + filename
8              image = cv2.imread(image_path)
9              image = np.array(image)
10             image = image.astype('float32')
11             image /= 255
12             img_data_array.append(image)
13             class_name.append(dir1)
14     return img_data_array, class_name
```

A primeira função definida no código é a "create_dataset". Essa função prepara as imagens da base de dados para serem lidas pelo programa. Ela recebe como argumento apenas o diretório para onde as imagens estão salvas, em seguida cria dois *arrays*, um para guardar as imagens e outro para guardar a classe de cada imagem, que será o label que permite o computador diferenciar os objetos.

No primeiro laço da função, na linha 5, o programa vai acessar cada classe disponível no diretório. Dentro de cada uma dessas classes, o programa passa por outro laço, linha 6, acessando todas as imagens do diretório. Para cada imagem, o programa salva o caminho para a imagem, lê a imagem na linha 8, a transforma em um array na linha 9 e a traduz para o formato "float32" na linha 10. Como todas as imagens são coloridas e no formato RGB cada pixel tem um valor de 0 a 255 que define a cor. Para simplificar os valores eles são normalizados, dividindo-os por 255, linha 11 para que todos fiquem entre 0 e 1. Por fim, para cada imagem analisada, o programa salva os valores em um array e a classe de cada imagem em outro, retornando esses arrays para o chamado da função.

```
x_train, y_train = create_dataset(train_path)
```

```
x_test, y_test = create_dataset(test_path)
```

Em seguida temos as chamadas da função "create_dataset". A função recebe como parâmetro o path para os arquivos com as imagens e retorna os arrays e labels das imagens, com x recebendo os números e y os labels.

```
target_dict = {k: v for v, k in enumerate(np.unique(y_train))}
```

```
target_dict_test = {k: v for v, k in enumerate(np.unique(y_test))}
```

O próximo passo é redefinir os *labels* das classes. Para isso usamos a função da biblioteca numpy "unique", que retorna todas as possibilidades diferentes para os *labels*, e para cada um

retornado, atribuímos um valor aritmético. Um exemplo pode ser visto a seguir, onde atribuímos ao label ‘0000439’ o valor 0, para o ‘0004266’ o valor 1 e assim por diante.

```
target_dict
```

```
{'0000439': 0, '0004266': 1, '0010736': 2, '0515116': 3, '0519456': 4}
```

```
target_val = [target_dict[y_train[i]] for i in range(len(y_train))]
```

```
target_val_test = [target_dict_test[y_test[i]] for i in range(len(y_test))]
```

Com os novos labels definidos podemos agora atribuir as imagens a eles. Para isso simplesmente se passa por cada imagem, ligando-a com seu respectivo novo *label*.

```
from tensorflow.keras.utils import to_categorical
```

```
y_cat_train = to_categorical(target_val, 5)
```

```
np.unique(target_val)
```

```
array([0, 1, 2, 3, 4])
```

```
y_cat_train.shape
```

```
(2847, 5)
```

```
y_cat_train[1000]
```

```
array([0., 1., 0., 0., 0.], dtype=float32)
```

```
y_cat_test = to_categorical(target_val_test, 5)
```

```
y_cat_test.shape
```

```
(714, 5)
```

A última coisa que precisa ser feita com os *labels* é transformá-lo em um formato que pode ser utilizado pelo algoritmo. Para isso usamos uma função da biblioteca tensorflow chamada “to_categorical” que recebe a lista e o número de classes que ela possui e retorna um array com todas as classes já definidas.

```
x_train = np.array(x_train)
```

```
x_train.shape
```

```
(2847, 250, 250, 3)
```

```
x_test = np.array(x_test)
```

```
x_test.shape
```

```
(714, 250, 250, 3)
```

Agora que os labels estão organizados corretamente podemos trabalhar com a organização das imagens. A primeira coisa que fazemos é reorganizar o array x_train e x_test. No momento, dentro de cada um desses arrays, estão todos os pixels de cada uma das imagens organizados em várias listas, mas depois de usarmos a função np.array, transformamos todas essas listas em um único array contendo todas as informações. Podemos ver também o resultado disso quando olhamos para o formato dos arrays, que agora tem quatro números, o primeiro representa a quantidade de imagens no array, o segundo e o terceiro são o tamanho da imagem e por fim o último número indica que as imagens estão no formato de cor RGB.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Activation, Dropout
```

Com a nossa base de dados pronta para ser usada, podemos agora importar o restante das bibliotecas do tensorflow que iremos usar para a criação do modelo. Essas bibliotecas são a Sequential, Dense, Conv2D, MaxPooling2D, Flatten, Activation e Dropout. A função Sequential é a primeira a ser utilizada, ela indica que a rede neural que está sendo criada será formada por uma sequência de camadas neurais. Dense é outra função de redes neurais que será utilizada, o propósito dela é informar que essa é uma camada profunda, significando que todos os neurônios da camada estão conectados com todos os neurônios da anterior. Conv2D é a função que gera a camada convolucional da CNN, essa camada recebe a camada anterior como entrada e gera uma camada melhor como saída. A função de MaxPooling2D caminha passo a passo pelo espaço das camadas e para cada espaço avaliado pela função, ela retorna o valor máximo de cada espaço. Flatten é utilizado para transformar inputs de várias dimensões em um de uma única dimensão. A função Activation são entidades matemáticas que simulam os neurônios de uma rede neural biológica. A última função, Dropout, tem a função de derrubar uma quantidade de neurônios aleatórios com o propósito de evitar o *overfitting*.

```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(4,4),input_shape=(250,250,3),activation='relu',))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(filters=64, kernel_size=(4,4),input_shape=(250,250,3),activation='relu',))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(filters=64, kernel_size=(4,4),input_shape=(250,250,3),activation='relu',))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dropout(0.25))

model.add(Dense(5))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Como mencionado anteriormente, Sequential é a primeira função chamada, com ela damos início ao nosso modelo de treinamento. Como estamos criando uma CNN, a primeira camada que criamos é utilizando a função Conv2D. A função Conv2D possui vários parâmetros, o primeiro parâmetro é filters, ele indica a dimensionalidade da saída dessa camada. O segundo parâmetro é o kernel_size, que pode receber ou uma tupla ou um valor inteiro, esse valor representa o tamanho da janela de análise da camada. O próximo parâmetro é o input_shape, esse parâmetro indica o tamanho das imagens que estarão sendo analisadas pela camada. É importante que esse valor seja o mesmo indicado anteriormente no código. Por último, activation, pode ser utilizado tanto como um parâmetro, como é o caso aqui, quanto sua própria camada, como veremos em seguida. Para essas camadas iniciais usaremos a função relu para activation. Relu significa rectified linear unit e retorna uma tupla com o valor máximo da camada.

A próxima parte da camada é a criação de uma MaxPooling2D. Essa função recebe como parâmetro apenas a variável pool_size, que pode ser um valor inteiro ou uma tupla, do mesmo modo que o parâmetro kernel_size na função Conv2D. Essa tupla indica o tamanho do espaço que será analisado por vez, nesse caso é uma janela de tamanho 2 por 2. Como a altura e a largura da janela tem o mesmo tamanho poderíamos ter escrito pool_size=2 e teríamos obtido o mesmo resultado.

Essas duas funções juntas formam a primeira camada de nossa CNN. Com isso repetimos as duas funções para criarmos outras camadas e dar profundidade para nossa rede. Todos os parâmetros possuem os mesmo valores, com exceção do valor de *filters*, que podemos aumentar para gerarmos mais neurônios.

Com as principais camadas da rede neural criadas, o próximo passo é agrupar as informações para o tamanho desejado pelo programa. A camada de flatten irá transformar a saída das camadas anteriores de modo que esses dados caibam no batch, ou lote, definido.

A camada de dropout, como definido anteriormente, é uma forma de se evitar o overfitting durante o treinamento. Com ela podemos selecionar aleatoriamente uma quantidade

de neurônios da rede para serem derrubados, esses neurônios são selecionados de maneira aleatória a cada repetição do algoritmo. A quantidade a ser derrubada pode ser escolhida através do parâmetro enviado, nesse caso serão 25% dos neurônios.

A próxima camada é a dense. Ela recebe o número de classes da base de dados como parâmetro e pode ser chamada mais de uma vez. É importante garantir que o valor esteja correto quando essa camada é chamada pela última vez, pois se não for condizente com a quantidade de classes presentes, o programa não poderá ser rodado.

Por fim, temos o método *compile*. O propósito desse método é o de configurar o modelo para o treinamento. Ele recebe como parâmetro loss, que é uma instância que define como a perda será calculada, e optimizer, são os dois requisitos para podermos usar o método compile. O optimizer que estamos usando nesse caso é o adam, pois é o modelo padrão do Tensorflow. O último parâmetro do método é metrics, que define qual métrica o sistema vai analisar conforme o processo é realizado, a métrica escolhida foi *accuracy*, mas uma alternativa para ela seria loss.

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
early_stop = EarlyStopping(monitor='val_loss', patience=1)
```

```
batchsize = 8
```

As duas últimas coisas que faltam antes de realizarmos o treinamento são o EarlyStopping e o *batch_size*. Essas duas ferramentas servem o propósito de acelerar o processo de treinamento. EarlyStopping é uma função do keras que serve para parar o processo de treinamento quando ele chega ao ponto em que mais treinamento não é mais beneficial. A função possui dois parâmetros, monitor, que estabelece qual métrica será observada e patience, que define quantas iterações o sistema aguarda antes de parar o treinamento. *Batch_size* agiliza o processo dividindo a base de dados em lotes, fazendo com que o computador possa utilizar menos memória durante os cálculos.

```
model.fit(x_train,y_cat_train,batch_size=batchsize,epochs=15,validation_data=(x_test,y_cat_test),callbacks=[early_stop])
```

Utilizando o modelo criado podemos começar o treinamento com a função fit. Essa função recebe como parâmetro primeiro a base de dados que será utilizada para treinar, a variavel *x_train* passa as imagens e *y_cat_train* os labels que foram preparados anteriormente. Em seguida passamos o batch_size, que define quantas imagens deverão ser pegas por vez. *Epochs* define a quantidade de iterações que o treinamento terá, como definimos epochs como 15 isso quer dizer que o sistema passará por todas as imagens 15 vezes, cada vez com neurônios diferentes graças a função dropout. *Validation_data* define as imagens e labels que serão utilizados para testar os resultados do treinamento. Por fim, o callbacks recebe o *early_stop* que foi definido anteriormente para saber se é necessário parar o treinamento antes dos 15 epochs serem concluídos.

Essa parte do algoritmo é a que requer o maior esforço e tempo para a máquina realizar todos os cálculos, podendo levar de minutos a horas, dependendo da capacidade computacional e da quantidade de imagens sendo utilizadas.

```
losses = pd.DataFrame(model.history.history)
```

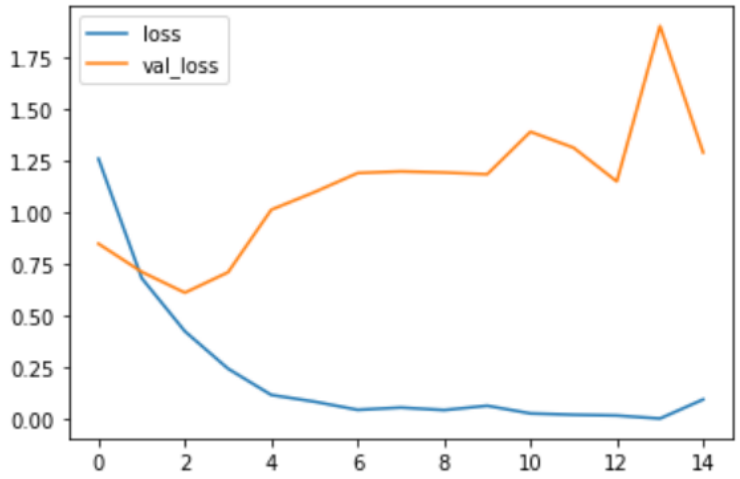
Com o treinamento realizado podemos usar a biblioteca pandas para transformar os dados desse treino para gerar um data frame para visualização dessas informações. Desta maneira é possível gerar gráficos e tabelas representando a taxa de aprendizado, a perda e a acurácia do sistema como um todo.

```
losses
```

	loss	accuracy	val_loss	val_accuracy
0	1.259216	0.486126	0.848144	0.680672
1	0.681898	0.756937	0.712179	0.746499
2	0.424749	0.849315	0.611402	0.788515
3	0.244027	0.913593	0.710290	0.788515
4	0.116974	0.958553	1.012604	0.784314
5	0.084610	0.972251	1.097718	0.798319
6	0.045005	0.982086	1.190037	0.787115
7	0.056113	0.980330	1.197886	0.767507
8	0.043864	0.985248	1.192546	0.792717
9	0.064646	0.978574	1.184265	0.794118
10	0.027645	0.992624	1.389654	0.803922
11	0.021054	0.992624	1.312618	0.798319
12	0.017554	0.994029	1.149408	0.816527
13	0.002792	0.999297	1.901209	0.803922
14	0.095307	0.971549	1.289240	0.805322

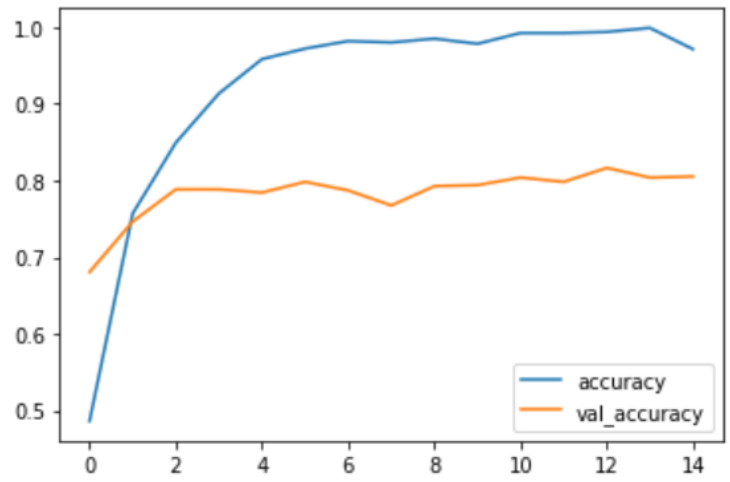
```
losses[['loss','val_loss']].plot()
```

```
<AxesSubplot:>
```




```
losses[['accuracy','val_accuracy']].plot()
```

```
<AxesSubplot:>
```



Com a tabela e os gráficos acima, podemos visualizar a taxa de aprendizagem do programa. Para essa visualização existem quatro principais variáveis, *loss*, *val_loss, accuracy e val_accuracy*. *Loss* e *accuracy* representam os valores de perda e acurácia do sistema quando ele trabalha com as imagens de treino, podemos ver no segundo gráfico que depois do treinamento o sistema tem quase 100% de acerto nas imagens usadas para treinar. *Val_loss* e *val_accuracy* representam os valores das imagens de teste que o sistema não tinha estudado ainda. É possível visualizar uma queda esperada na capacidade discriminativa do programa, porém a taxa de acerto se mantém em torno de 80%.

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
classes_x = model.predict(x_test)
predictions = np.argmax(classes_x, axis=1)
```

```
print(classification_report(target_val_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.93      0.73      0.82       143
           1       0.82      0.86      0.84       161
           2       0.79      0.76      0.78       131
           3       0.68      0.84      0.75       130
           4       0.85      0.83      0.84       149

    accuracy                           0.81       714
   macro avg       0.81      0.80      0.80       714
weighted avg       0.82      0.81      0.81       714
```

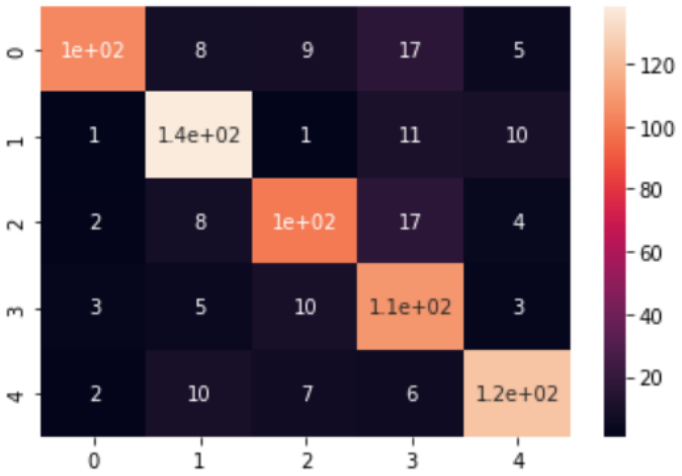
Se quisermos obter ainda mais informações sobre os resultados do sistema, podemos utilizar as bibliotecas classifcation_report e *confusion_matrix*. *Classification_report* posibilita a criação de uma tabela mais detalhada dos resultados, apresentando ao usuário não só a acurácia final, mas também a taxa de acerto de cada classe do sistema. Na tabela acima podemos visualizar que a classe 0 é a classe com a maior taxa de acerto, enquanto a classe 3 possui a menor.

```
confusion_matrix(target_val_test,predictions)
```

```
array([[104,   8,   9,  17,   5],
       [  1, 138,   1,  11,  10],
       [  2,   8, 100,  17,   4],
       [  3,   5,  10, 109,   3],
       [  2,  10,   7,   6, 124]], dtype=int64)
```

```
sns.heatmap(confusion_matrix(target_val_test,predictions),annot=True)
```

```
<AxesSubplot:>
```



A *confusion_matrix*, como o nome sugere, retorna uma matriz de confusão, isso quer dizer uma matriz que indica onde o sistema falhou em distinguir as classes corretamente. Podemos ver através dessa matriz que os locais onde houve o maior número de erros foram entre as classes 3 e 0 e as classes 3 e 2, alinhando com o que vimos anteriormente no *classification_report*.

```
img = x_test[713]
```

```
img_class = model.predict(img.reshape(1,250,250,3))
my_img_class = np.argmax(img_class,axis=1)
```

```
my_img_class
```

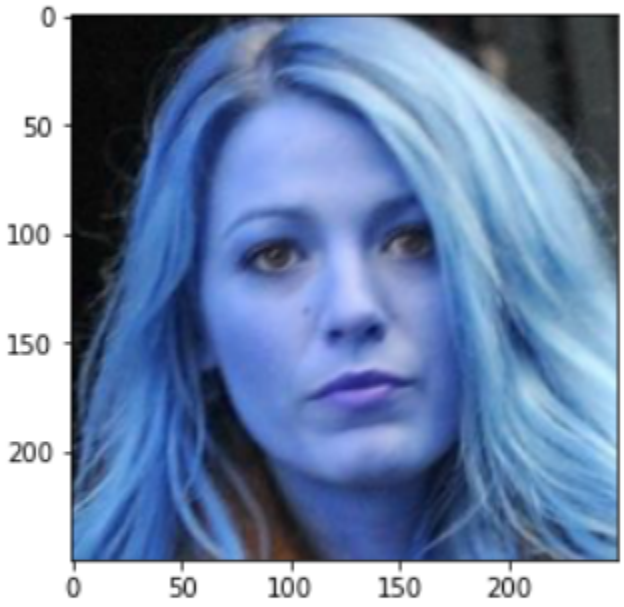
```
array([4], dtype=int64)
```

Também é possível, caso desejado, fazer a análise de uma única imagem. Pegando uma imagem qualquer do sistema, podemos passá-la pelo modelo criado. O modelo então irá retornar a classe na qual ela possui a maior probabilidade de pertencer. Podemos ver no código acima que passamos a imagem de teste de número 713. Depois de ser analisada pelo programa, o sistema retorna corretamente que a imagem pertence a classe 4.

```
img = x_test[438]
```

```
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x29118d8d978>
```




```
img_class = model.predict(img.reshape(1,250,250,3))
my_img_class = np.argmax(img_class,axis=1)
```

```
my_img_class
```

```
array([3], dtype=int64)
```

5.1 Alterações no código

Uma série de alterações foram feitas ao código implementado para descobrir se alguma delas causaria um melhoramento considerável. A primeira delas foi um aumento da quantidade de camadas convolucionais na rede neural e a quantidade de neurônios. No código original temos três camadas convolucionais: uma com 32 neurônios, outra com 64, e por fim mais uma com 64.

A alteração pode ser vista na imagem a seguir:

```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(4,4),input_shape=(250,250,3),activation='relu',))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(filters=64, kernel_size=(4,4),input_shape=(250,250,3),activation='relu',))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(filters=128, kernel_size=(4,4),input_shape=(250,250,3),activation='relu',))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(filters=256, kernel_size=(4,4),input_shape=(250,250,3),activation='relu',))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dropout(0.25))

model.add(Dense(5))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

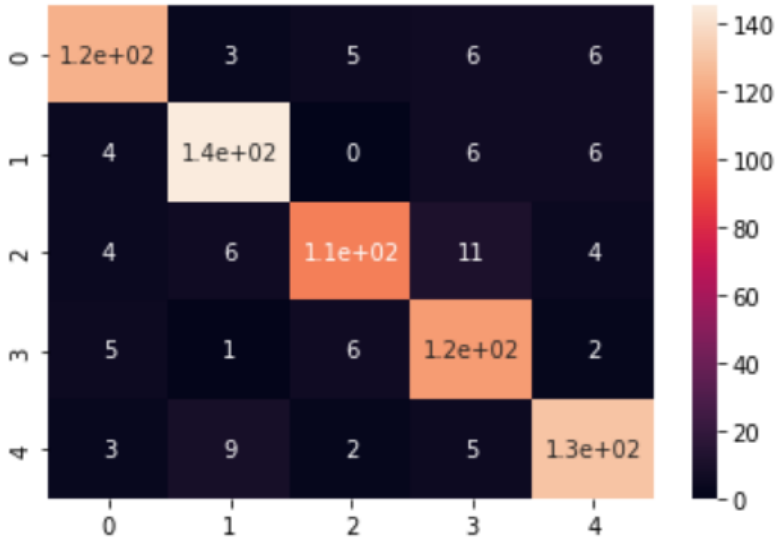
Pode-se ver que agora temos quatro camadas convolucionais, dessa vez uma com 32 neurônios, outra com 64, uma terceira camada com 128 e uma quarta e última camada com 256.

```
confusion_matrix(target_val_test,predictions)
```

```
array([[123,   3,   5,   6,   6],
       [  4, 145,   0,   6,   6],
       [  4,   6, 106,  11,   4],
       [  5,   1,   6, 116,   2],
       [  3,   9,   2,   5, 130]], dtype=int64)
```

```
sns.heatmap(confusion_matrix(target_val_test,predictions),annot=True)
```

```
<AxesSubplot:>
```



Podemos ver na matriz de confusão acima que um aumento na quantidade de camadas e neurônios envolvidos nos cálculos podem retornar resultados mais corretos do que os anteriormente obtidos. É possível ver esse resultado claramente se compararmos a quantidade de imagens que foram definidas corretamente pelo sistema na classe 0, onde originalmente o sistema classificou corretamente 104 das imagens, e no novo sistema, com mais camadas e neurônios, esse valor subiu para 123. Esse melhoramento também pode ser visto nas outras classes.

Outra alteração que foi testada foi a remoção da ferramenta de *dropout* do sistema, podendo ser visualizada na próxima página. Como discutido anteriormente neste trabalho, dropout é uma ferramenta que derruba, de maneira aleatória, uma certa quantidade de neurônios para evitar o *overfitting*. Quando removemos essa ferramenta podemos ver o efeito dessa remoção.

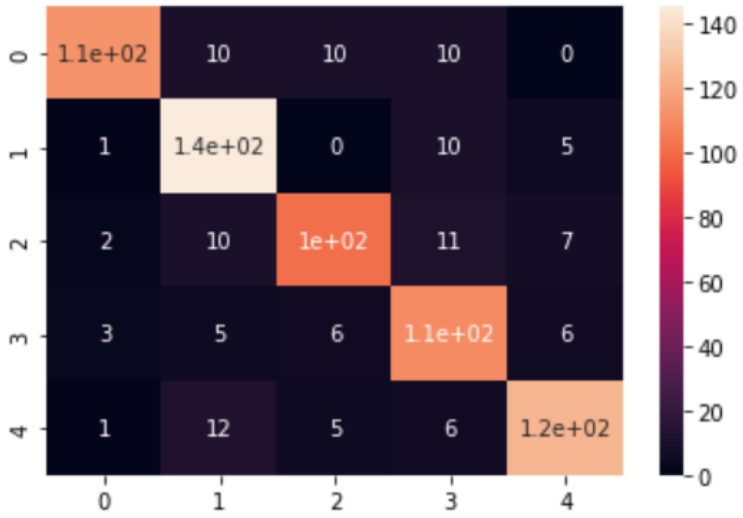
```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(4,4),input_shape=(250,250,3),activation='relu',))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(filters=64, kernel_size=(4,4),input_shape=(250,250,3),activation='relu',))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(filters=64, kernel_size=(4,4),input_shape=(250,250,3),activation='relu',))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

#model.add(Dropout(0.25))

model.add(Dense(5))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
confusion_matrix(target_val_test,predictions)
```

```
array([[113,  10,  10,  10,   0],
       [  1, 145,   0,  10,   5],
       [  2,  10, 101,  11,   7],
       [  3,   5,   6, 110,   6],
       [  1,  12,   5,   6, 125]], dtype=int64)
```

```
sns.heatmap(confusion_matrix(target_val_test,predictions),annot=True)
```

```
<AxesSubplot:>
```



Apesar da quantidade de acerto estar similar a do modelo original, deve-se chamar atenção para os valores dos erros. Notamos que dessa vez os erros estão mais concentrados do que nos casos anteriores. As classes 0, 1 e 3 possuem um foco maior dos erros do que as classes 2 e 4. Isso é um resultado do sistema sofrendo de *overfitting*, criando alta dependência das características aprendidas durante a fase de treino, gerando um maior número de falsos positivos nos resultados.

5.2 Resultados Comparativos com a Literatura

Nesta seção, vamos apresentar e explicar os resultados encontrados pelos pesquisadores dos outros trabalhos apresentados na seção 2.3 em seus respectivos trabalhos relacionados ao desenvolvimento de suas técnicas, a L-Softmax e AM-Softmax. Os valores obtidos nos trabalhos referenciados foram alcançados rodando os programas desenvolvidos pelos pesquisadores em cima da base de dados LFW[4]. Essa base possui mais de 12 mil imagens, pertencentes a mais de 5700 pessoas.

Modelo	Softmax	L-Softmax (2016)	AM-Softmax (2018)	SphereFace (2017)	CosFace (2018)	ArcFace (2019)
Precisão (%)	96,53	98,71	99,10	99,42	99,73	99,83

Na tabela acima pode-se ver os resultados obtidos pelos pesquisadores. Como cada um deles fez os próprios testes e compararam os resultados com a função base do *Softmax*, apresentamos o obtido por Weiyang, pois seu trabalho sobre L-Softmax foi o primeiro a ser feito e é referenciado em todos os outros. Também é possível notar que, com exceção da função *SphereFace*, desenvolvido por Weiyang, existe um crescimento estável, ao longo dos anos, da precisão alcançada pelos novos sistemas.

[4] Fonte: http://vis-www.cs.umass.edu/lfw/ Acesso em 26 de set de 2022.

6. Conclusão

Devido a rápida evolução das tecnologias que compõem visão de computadores e discriminação de imagens, novas possibilidades de implementação de reconhecimento facial estão se apresentando a cada dia. Com computadores cada vez mais potentes e técnicas de desenvolvimento cada vez mais eficientes, a utilização de novas tecnologias é uma realidade constante em nossas vidas. Com a aplicação de modelos de reconhecimento facial modernos em atividades diárias, temos a possibilidade de criar novas ferramentas que causarão um impacto real e positivo.

Neste trabalho foi apresentado um projeto que é capaz de realizar o reconhecimento e discriminação facial, utilizando técnicas de aprendizagem de máquina e redes neurais, realizando o treinamento e testes mostrando uma taxa de acerto de mais de 80%.

A utilização de tabelas e gráficos na seção 5 facilitou a compreensão dos dados adquiridos ao longo do processamento do sistema, oferecendo mais clareza nas informações. Permitindo assim compreender que apesar de existir uma quantidade considerável de alternativas no momento da criação do modelo da rede neural, a discrepância no resultado final pode variar de tamanho, desde uma diferença gritante até uma quase desconsiderável.

Durante o desenvolvimento desse projeto houve vários desafios que precisaram ser superados: o processo para obter uma compreensão didática do tema; o desenvolvimento do código que levasse até o resultado desejado e o preparo da máquina para que ela pudesse trabalhar com esse código. Essa última parte apresentou um desafio diferenciado, com várias abordagens possíveis sendo utilizadas para alcançar o resultado final, mas ao longo do desenvolvimento desse trabalho todos esses desafios foram superados.

Uma possível continuação desta pesquisa para desenvolvimento futuro, seria a implementação de um algoritmo mais especializado, como um dos que mostramos nesse projeto, desenvolvidos pelos autores referenciados ao longo do trabalho. Outra alternativa para evolução, no futuro, seria a utilização de uma máquina mais moderna e avançada, com um poder computacional mais desenvolvido do que a que estava disponível. A placa de vídeo utilizada nos testes foi, como mencionado anteriormente, uma rtx 1060 da Nvidia, que possui 2560 núcleos CUDA, enquanto placas mais modernas como a rtx 3090, lançada em 2020, que possui 10496 núcleos CUDA. Em setembro de 2022 a Nvidia revelou a mais recente série de placas de vídeo e a mais poderosa dessa série, a rtx 4090, terá mais de 16 mil núcleos[5]. A utilização de peças mais capazes, tais como estas listadas, permitiria o sistema trabalhar com um maior número de imagens, camadas, neurônios e cálculos matemáticos, o que pode levar não apenas a processamentos mais rápidos mas, também, a resultados mais precisos.

[5] Fonte: https://www.tecmundo.com.br/voxel/248342-comparativo-rtx-4090-rtx-4080-veja-diferencas-gpus-nvidia.htm Acesso em : 26 set 2022.

7. Referencias

LIU, W.; WEN, Y.; YU, Z.; YANG, M. **Large-Margin Softmax Loss for Convolutional Neural Network,** ICML, v. 2, n. 3, p. 7, 2016. Disponível em <http://proceedings.mlr.press/v48/liud16.pdf> Acesso em 01 jun, 2021.

LIU, W.; WEN, Y.; YU, Z.; LI, M.; RAJ, B.; SONG, L. **SphereFace: Deep Hypersphere Embedding for Face Recognition,** Proceedings of the IEEE conference on computer vision and pattern recognition, p. 212-220, 2017. Disponivel em <https://openaccess.thecvf.com/content_cvpr_2017/papers/Liu_SphereFace_Deep_Hypersphere_CVPR_2017_paper.pdf> Acesso em 01 jun, 2021.

WANG, F.; LIU, W.; LIU, H.; CHENG, J. **Additive margin softmax for face verification,** IEEE Signal Processing Letters 25.7, p. 926-930, 2018. Disponível em <https://arxiv.org/pdf/1801.05599.pdf> Acesso em 01 jun, 2021.

WANG, H.; WANG, Y.; ZHOU, Z.; JI, X.; GONG, D.; ZHOU, J.; LI, Z.; LIU, W. **CosFace: Large Margin Cosine Loss for Deep Face Recognition,** CosFace: Large Margin Cosine Loss for Deep Face Recognition, p. 5265-5274, 2018. Disponível em <https://openaccess.thecvf.com/content_cvpr_2018/papers/Wang_CosFace_Large_Margin_CVPR_2018_paper.pdf> Acesso em 19 set, 2021.

DENG, J.; GUO, J.; XUE, N.; ZEFEIRIOU, S. **ArcFace: Additive Angular Margin Loss for Deep Face Recognition,** Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, p. 4690-4699, 2019. Disponivel em <https://openaccess.thecvf.com/content_CVPR_2019/papers/Deng_ArcFace_Additive_Angular_Margin_Loss_for_Deep_Face_Recognition_CVPR_2019_paper.pdf> Acesso em 01 jun, 2021

PANDYA, J. M.; RATHOD, D.; JADAV, J. J.; **A Survey of Face Recognition approach,** International Journal of Engineering Research and Applications, p. 632-635, 2013. Disponivel em <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.415.2082&rep=rep1&type=pdf> Acesso em 26 set, 2021.

SYAFEEZA, A. R.; ALIF, M. K. M. F.; ATHIRAH, Y. N.; JAAFAR, A. S.; NORIHAN, A. H.; SELAHA, M. S. **IoT based facial recognition door access control home security system using raspberry pi,** International Journal of Power Electronics and Drive Systems 11.1, p. 417, 2020. Disponivel em <https://pdfs.semanticscholar.org/5518/8e2dd3dc25dc554365011f258fd1827a1a64.pdf> Acesso em 27 set, 2021.

BENNETT, K. A. **Can Facial Recognition Technology Be Used to Fight the New Way against Terrorism: Examining the Constitutionality of Facial Recognition Surveilance Systems,** NCJL & Tech, p. 151, 2001. Disponível em <https://scholarship.law.unc.edu/cgi/viewcontent.cgi?article=1018&context=ncjolt> Acesso em 26 set, 2021.

WU, C.; ZENG, Y.; SHIH. M. **Enhancing retailer marketing with an facial recognition integrated recommender system,** IEEE International Conference on Consumer Electronics-Taiwan, p. 25-26, 2015. Disponível em <https://ieeexplore.ieee.org/abstract/document/7216881> Acesso em 25 set, 2021.

CHUN, W. J. **Core Python Programming,** Second Edition. Pearson Education India, 2006. Disponivel em <http://14.99.188.242:8080/jspui/bitstream/123456789/12621/1/Core%20Python%20Programming.pdf> Acesso em 29 set, 2021.

EASTWOOD, B. **The 10 Most Popular Programming Languages to Learn in 2021**, 2020. Disponível em: <https://www.northeastern.edu/graduate/blog/most-popular-programming-languages/> Acesso em 28 set, 2021.

TENSORFLOW, **tensorflow homepage,** 2021. Disponível em: <https://www.tensorflow.org> Acesso em 15 ago, 2021.

PARKHI, O. M.; VEDALDI, A.; ZISSERMAN, A. **Deep Face Recognition,** 2015. Disponivel em <http://cis.csuohio.edu/~sschung/CIS660/DeepFaceRecognition_parkhi15.pdf> Acesso em 28 set, 2021.

JAIN, A. K.; JIANCHANG, M. **Artificial neural networks: A tutorial,** Computer, p. 31-44, 1996. Disponivel em <http://metalab.uniten.edu.my/~abdrahim/mitm613/Jain1996_ANN%20-%20A%20Tutorial.pdf > Acesso em 26 set, 2021.

ABRAHAM, A. Artificial Neural Networks. In: SYDENHAM P. H.; THORN, R. **Handbook of Measuring System Design,** Nova York, 2005, p. 901-908. Disponível em: <http://wsc10.softcomputing.net/ann_chapter.pdf> Acesso em: 21 set, 2021.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. **ImageNet Classification with Deep Convolutional Neural Networks,** Advances in neural information processing systems, p. 1097-1105, 2012. Disponivel em

<https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf> Acesso em 20 set, 2021

GU, J.; WANG, Z.; KUEN, J.; MA, L.; SHAHROUDY, A.; SHUAI, B.; LIU, T.; WANG, X.; WANG, L.; WANG, G.; CAI, J.; CHEN T. **Recent Advances in Convolutional Neural Networks,** Pattern Recognition, v. 77, p 354-377, 2018. Disponível em <https://arxiv.org/pdf/1512.07108.pdf> Acesso em 01 out, 2021.

SINGH, H. **Practical Machine Learning and Image Processing,** Apress, Allahabad, Uttar Pradesh, India, 2016. Disponivel em <https://hoclaptrinhdanang.com/downloads/pdf/ai/Practical%20Machine%20Learning%20and%20Image%20Processing.pdf> Acesso em 01 out, 2021.

DU, S; WARD, R. **WAVELET-BASED ILLUMINATION NORMALIZATION FOR FACE RECOGNITION,** IEEE International Conference on Image Processing 2005, v. 2, 2005. Disponível em <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.1559&rep=rep1&type=pdf> Acesso em 11 out, 2021.

ADINI, Y.; MOSES, Y.; ULLMAN, S. **Face recognition: The problem of compensating for changes in illumination direction,** IEEE Transaction on Pattern Analysis and Machine Intelligence, v. 19, p. 721-732, 1997. Disponível em <https://ieeexplore.ieee.org/abstract/document/598229> Acesso em 11 out, 2021.

KOU, Z.; YOU, K.; LONG, M.; WANG, J. **STOCHASTIC NORMALIZATION,** Advances in Neural Information Processing Systems 33, 2020. Disponível em <http://ise.thss.tsinghua.edu.cn/~mlong/doc/stochastic-normalization-nips20.pdf> Acesso em 12 out, 2021.

IOFFE, S.; SZEGEDY, C. **Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,** International Conference on Machine Learning, p. 448-456, 2015. Disponível em <http://proceedings.mlr.press/v37/ioffe15.pdf> Acesso em 12 out, 2021.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. **Dropout: A Simple Way to Prevent Neural Networks from Overfitting,** The Journal of machine learning research, v. 15, p. 1929-1958, 2014. Disponível em <https://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf?utm_campaign=buffer&utm_content=buffer79b43&utm_medium=social&utm_source=twitter.com> Acesso em 12 out, 2021.

COOIJMANS, T.; BALLAS, N.; LAURENT, C.; GULÇEHERE, Ç.; COURVILLE, A. **RECURRENT BATCH NORMALIZATION,** arXiv preprint arXiv:1603.09025, 2017. Disponível em <https://arxiv.org/pdf/1603.09025.pdf> Acesso em 12 out, 2021.

WU, Y.; HE, K. **Group Normalization,** Proceedings of the European conference on computer vision (ECCV), p. 3-19, 2018. Disponível em <https://openaccess.thecvf.com/content_ECCV_2018/papers/Yuxin_Wu_Group_Normalization_ECCV_2018_paper.pdf> Acesso em 12 out, 2021.

ULYANOV, D.; VEDALDI, A.; LEMPITSKY, V. **Instance Normalization: The Missing Ingredient for Fast Stylization,** arXiv preprint arXiv:1607.08022, 2017. Disponível em <https://arxiv.org/pdf/1607.08022.pdf> Acesso em 12 out, 2021.

ULYANOV, D.; VEDALDI, A.; LEMPITSKY, V. **Texture Networks: Feed-forward Synthesis of Textures and Stylized Images,** ICML, v. 1, n. 2, 2016. Disponível em <http://proceedings.mlr.press/v48/ulyanov16.pdf> Acesso em 12 out, 2021.

WANG, T.; WU, D.; COATES, A.; NG, A. **END-TO-END TEXT RECOGNITION WITH CONVOLUTIONAL NEURAL NETWORKS,** Proceedings of the 21st international conference on pattern recognition, 2012. Disponível em <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.664.6212&rep=rep1&type=pdf> Acesso em 13 out, 2021.

SCHROFF, F.; KALENICHENKO, D.; PHILBIN, J. **FaceNet: A Unified Embedding for Face Recognition and Clustering,** Proceedings of the IEEE conference on computer vision and pattern recognition, p. 815-823, 2015. Disponível em <https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Schroff_FaceNet_A_Unified_2015_CVPR_paper.pdf> Acesso em 14 out, 2021.

JABBAR, H.; KHAN, Z. **METHODS TO AVOID OVERFITTING AND UNDER-FITTING IN SUPERVISED MACHINE LEARNING (COMPARATIVE STUDY),** Computer Science, Communication and Instrumentation Devices, p. 163-172, 2015. Disponível em <https://d1wqtxts1xzle7.cloudfront.net/37902828/017-with-cover-page-v2.pdf?Expires=1634433653&Signature=fis~lIVN9eGYZ7grIqNS2mo9K7o8KRt870NWk14AvAP8cQq8K9jqEqRFeIYUC-Iq1vxsoRIisFCTu6qhkSV9n0-n5m4A7c70CMckCQxrAFXngB9hePYd9stJMYxdSV8FKhlMomnuWQUBvQEPB35AcDl~I9S10wgD1iDT0QrOExU50W8K24ks6R8vMJKiljQmPv2QeSmEfCpHiQ97PH6iOgYAODm-QTALitVnZguWrdA~rhH9mOpdEgTp4oteQcxr-LrI0rht5i3Or8f

X2iXFK5zGMyrrB7aQdBx25ChqxS3l-WhVybiIBPJJmpk7ey3RCiEPY9kfASO4AN6Rsxe4kg_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA> Acesso em 15 out, 2021.

SUN, Y.; WANG, X.; TANG, X. **DEEP CONVOLUTIONAL NETWORK CASCADE FOR FACIAL POINT DETECTION,** In Proceedings of the IEEE conference on computer vision and pattern recognition, p. 3476-3483. 2013. Disponível em <https://openaccess.thecvf.com/content_cvpr_2013/papers/Sun_Deep_Convolutional_Network_2013_CVPR_paper.pdf> Acesso em 28 ago, 2022.

GAO, B.; PAVEL, L. **ON THE PROPERTIES OF THE SOFTMAX FUNCTION WITH APPLICATION IN GAME THEORY AND REINFORCEMENT LEARNING,** arXiv preprint arXiv:1704.00805. Disponível em <https://arxiv.org/pdf/1704.00805> Acesso em 24, set, 2022.