PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Alexandre Neukirchen

DETECÇÃO DE PONTOS-CHAVE FACIAIS

Caxias do Sul
2020

Alexandre Neukirchen

DETECÇÃO DE PONTOS-CHAVE FACIAIS

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Caxias do Sul
2020

LISTA DE ABREVIATURAS E SIGLAS

API	Interface de programação de aplicações
CSV	Valores Separados por vírgula (Comma Separated Values)
CNN	Redes Neurais Convolucionais (Convolutional Neural Networks)
DARPA	Agência de Projetos de Pesquisa Avançada de Defesa
FBI	Federal Bureau of Investigation
FERET	Tecnologia de Reconhecimento de Face
NaN	Não é um número (Not a number)
NFL	National Football League
NIST	Instituto Nacional de Padrões e Tecnologia

GLOSSÁRIO

Apple	Empresa multinacional norte-americana
Super Bowl	Jogo final do campeonato da NFL
Facebook	Mídia social e rede social virtual
FaceFirst	Empresa do segmento de IA
iPhone X	Smart phone da Aple
GIMP	Software de edição de imagens
Inteiro	Tipo de dado numérico
Kaggle	Comunidade online de ciência de dados
Keras	Biblioteca open-source
MasterCard	Empresa norte-americana do setor de pagamentos
MasterCard Identity Check	Sistema de pagamento via reconhecimento facial
Matplotlib	Biblioteca escrita em Python
Ndarray	Objeto tipo matriz multidimensional
Pandas	Biblioteca escrita em Python
Photoshop	Software de edição de imagens
Python	Linguagem de programação
Síndrome de DiGeorge	Doença do tipo congênita
String	Tipo de dado texto
Tensorflow	Biblioteca open-source

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1. Introdução

1.1. Contextualização

A detecção de pontos chaves faciais está associada ao reconhecimento de padrões, o qual é um campo que envolve áreas como matemática, estatística e computação. Não é algo recente, em 1959, BLEDSOE e BROWNING escreveram um artigo sobre reconhecimento de padrões alfanuméricos, no qual ele aborda um método básico para identificar padrões únicos. Em 1979, TIEGER e GANZ, realizam um experimento relacionado a reconhecimento de imagens, onde é aplicado uma técnica chamada de máscara de grades. Na década de 90, a DARPA (Agência de Projetos de Pesquisa Avançada de Defesa) e o NIST (Instituto Nacional de Padrões e Tecnologia), ambos dos Estados Unidos, lançaram o programa chamado FERET (Tecnologia de Reconhecimento de Face).

No Super Bowl de 2002, a polícia utilizou pela primeira vez um sistema de reconhecimento facial em massa, com o objetivo de encontrar criminosos. Apesar de o sistema detectar algumas pessoas corretamente, o teste foi considerado um fracasso pela polícia, devido ao grande número de falsos positivos. Em 2009 foi criado nos Estado Unidos um banco de dados forense que permitiu aos policiais acessar fotos do departamento de segurança rodoviária e veículos motorizados. Em 2010 o Facebook começou a implementar a funcionalidade de reconhecimento facial, com o objetivo de identificar e marcar as pessoas que apareciam nas fotos. No início de 2014, a empresa FaceFirst desenvolveu uma plataforma com diversos serviços na área de inteligência artificial. Em 2017 a Apple lançou o telefone iPhone X, este telefone trouxe consigo a funcionalidade de desbloqueio do aparelho via reconhecimento facial.

Embora a ideia de reconhecimento facial e de padrões não ser algo recente, somente na última década ambas se tornaram viáveis por que hoje temos grande quantidade de dados, fácil acesso a esses dados, e baixo custo de processamento desses dados. A soma desses fatores vem contribuindo para o desenvolvimento de soluções na área do reconhecimento facial, bem como no problema de detecção de pontos chaves faciais, uma vez que a detecção desses pontos chaves é o alicerce para o reconhecimento facial, e será o enfoque deste trabalho.

1.2. O problema proposto

O problema é um desafio do Kaggle denominado Detecção dos Pontos Chaves Faciais (em inglês *Facial Keypoints Detection*), a detecção destes pontos é um fator importante quando tratamos do problema de reconhecimento facial, dessa forma o reconhecimento facial e a detecção dos pontos chaves faciais estão conectados intrinsecamente. O reconhecimento facial vem ganhando notoriedade devido a enorme gama de soluções em que pode ser aplicado. Dentre essas soluções podemos citar:

- Segurança

  O FBI possui um sistema que utiliza o reconhecimento facial para identificar suspeitos e foragidos. Eles possuem um banco de dados com fotos das carteiras de motoristas, e de criminosos, onde esse sistema realiza o cruzamento com imagens de sistemas de segurança em tempo real com o intuito de notificar entidades policiais.

- Pessoas Desaparecidas e Tráfico de Pessoas

  Em aeroportos, portos, lojas de varejo e estabelecimentos públicos de grandes centros em países desenvolvidos, existem sistemas conectados internacionalmente buscando identificar e sinalizar pessoas desaparecidas e sequestradas.

- Saúde

  O reconhecimento facial é utilizado pelo Instituo de Pesquisa do Genoma Humano, para diagnosticar uma doença rara chamada de Síndrome de DiGeorge, ajudando nesse diagnóstico em 96% dos casos.

- Sistemas de Pagamentos

  A empresa MasterCard lançou o serviço MasterCard Identity Check, no qual basicamente os usuários utilizam a câmera de seu telefone para efetuar um pagamento. Para confirmar seu pagamento basta apenas piscar os olhos.

Neste trabalho, os dados analisados para resolver o problema, foi disponibilizado pelo Kaggle em forma de arquivos CSV. Foram disponibilizados quatro arquivos, de treino, teste, arquivo de referência para criação do arquivo de submissão e o arquivo no formato padrão de envio para submeter na plataforma, com o propósito de avaliar os resultados obtidos.

O objetivo principal deste estudo, assim como descrito no Kaggle, é prever as posições dos pontos chaves em imagens de faces de pessoas. Para resolver esse problema será criado um modelo de deep learning, após ele será treinado e testado com imagens aleatórias, como saída ele retornará os pontos chaves faciais encontrados. Este desafio é antigo, o mesmo foi lançado em 2017, dessa forma, os dados disponibilizados são deste ano ou anterior a ele. Na descrição do conjunto de dados não existe informações sobre a origem, tão pouco sobre o período e a forma em que os mesmos foram coletados. Apenas foram disponibilizados para serem salvos e explorados.

2. Coleta de Dados

Os dados para o desenvolvimento da solução do problema foram obtidos através da plataforma Kaggle. As imagens fornecidas estão no formato 96x96 pixeis. Foram disponibilizados quatro arquivos no formato CSV como descrito abaixo:

- **train.csv**: Lista com 7049 imagens para treinar o algoritmo. Contem trinta e uma colunas, as primeiras trintam são às coordenadas (x, y) para quinze pontos chaves faciais, onde cada ponto chave é representado por duas colunas. A última coluna contem uma lista com os pixeis de cada imagem. Este arquivo é serve para treinar o modelo.
- **test.csv**: Lista com 1783 imagens para testar o algoritmo. Contem duas colunas, ImageId que é o identificador da imagem, e a coluna Image a qual possui uma lista de pixeis de cada imagem a ser avaliada. Este arquivo serve para testar o modelo.
- **IdLookupTable.csv**: Lista com 27124 registros. Cada linha contêm as seguintes colunas, RowId que é o identificador da linha, ImageId que é o

identificador da imagem, FeatureName que pode ser qualquer uma das colunas do arquivo train.csv, exceto a coluna Image e a coluna Location que é o campo que precisa ser previsto. Este arquivo serve como referência para gerar o arquivo de envio para plataforma Kaggle.

- **SampleSubmission.csv**: Lista com 27124 pontos a serem previstos. Cada linha contêm duas colunas, RowId que é o identificador da linha e Location que é o campo que precisa ser previsto. Este arquivo serve como exemplo de arquivo de envio para plataforma Kaggle.

Tabela 1. Arquivo de treino – train.csv

Nome da coluna/campo	Descrição	Tipo
left_eye_center_x	Coordenada x do centro do olho esquerdo	float64
left_eye_center_y	Coordenada x do centro do olho esquerdo	float64
right_eye_center_x	Coordenada x do centro do olho direito	float64
right_eye_center_y	Coordenada y do centro do olho direito	float64
left_eye_inner_corner_x	Coordenada x do canto interno do olho esquerdo	float64
left_eye_inner_corner_y	Coordenada y do canto interno do olho esquerdo	float64
left_eye_outer_corner_x	Coordenada x do canto externo do olho esquerdo	float64
left_eye_outer_corner_y	Coordenada y do canto externo do olho esquerdo	float64
right_eye_inner_corner_x	Coordenada x do canto interno do olho direito	float64
right_eye_inner_corner_y	Coordenada y do canto interno do olho direito	float64
right_eye_outer_corner_x	Coordenada x do canto externo do olho direito	float64
right_eye_outer_corner_y	Coordenada y do canto externo do olho direito	float64
left_eyebrow_inner_end_x	Coordenada x da extremidade interna da sobrancelha esquerda	float64
left_eyebrow_inner_end_y	Coordenada y da extremidade interna da sobrancelha esquerda	float64
left_eyebrow_outer_end_x	Coordenada x da extremidade externa da sobrancelha esquerda	float64
left_eyebrow_outer_end_y	Coordenada y da extremidade externa da sobrancelha esquerda	float64
right_eyebrow_inner_end_x	Coordenada x da extremidade interna da	float64

	sobrancelha direita	
right_eyebrow_inner_end_y	Coordenada y da extremidade interna da sobrancelha direita	float64
right_eyebrow_outer_end_x	Coordenada x da extremidade externa da sobrancelha direita	float64
right_eyebrow_outer_end_y	Coordenada y da extremidade externa da sobrancelha direita	float64
nose_tip_x	Coordenada x da ponta do nariz	float64
nose_tip_y	Coordenada y da ponta do nariz	float64
mouth_left_corner_x	Coordenada x do canto esquerdo da boca	float64
mouth_left_corner_y	Coordenada y do canto esquerdo da boca	float64
mouth_right_corner_x	Coordenada x do canto direito da boca	float64
mouth_right_corner_y	Coordenada y do canto direito da boca	float64
mouth_center_top_lip_x	Coordenada x do lábio superior do centro da boca	float64
mouth_center_top_lip_y	Coordenada y do lábio superior do centro da boca	float64
mouth_center_bottom_lip_x	Coordenada x do lábio inferior do centro da boca	float64
mouth_center_bottom_lip_y	Coordenada y do lábio inferior do centro da boca	float64
Image	Dados binários da imagem	object

Tabela 2. Arquivo de teste – test.csv

Nome da coluna/campo	Descrição	Tipo
ImageId	Identificador da imagem	int64
Image	Dados binários da imagem	object

Tabela 3. Arquivo de referência – IdLookupTable.csv

Nome da coluna/campo	Descrição	Tipo
RowId	Identificador da linha	int64
ImageId	Identificador da imagem	int64
FeatureName	Nome da feature	object
Location	Campo a ser previsto	float64

Tabela 4. Arquivo de exemplo – SampleSubmission.csv

Nome da coluna/campo	Descrição	Tipo
RowId	Identificador da linha	int64
Location	Campo a ser previsto	float64

3. Processamento/Tratamento de Dados

A partir deste tópico inicia a utilização do código fonte do projeto, todo o desenvolvimento da solução deste projeto foi realizado utilizando a linguagem de programação Python 3.6. Este tópico de processamento e tratamento de dados ficou dividido em quatro partes, a leitura das informações dos arquivos, listagem dos dados, verificação de dados ausentes e conversão de tipo de dados.

3.1 Leitura dos arquivos

Neste tópico foram utilizadas as bibliotecas Pandas e Matplotlib. Onde a primeira é específica para manipulação e análise de dados e a segunda é utilizada para a análise e visualização de dados. Ambas foram importadas no script da seguinte forma:

```
import pandas as pd
from matplotlib import pyplot as plt
```

Para carregar os dados e transformá-los em datasets foi utilizado a função *read_csv* da biblioteca Pandas como descrito abaixo, cada linha criou um dataset com os respectivos nomes *IdLookupTable, SampleSubmission, training* e *test*:

```
IdLookupTable = pd.read_csv('IdLookupTable.csv')
SampleSubmission = pd.read_csv('SampleSubmission.csv')
training = pd.read_csv('training.csv')
test = pd.read_csv('test.csv')
```

Para verificar informações como o número de registros, o número de colunas e o tipo de dado que cada coluna contem foi utilizado a função *info* como descrito abaixo.

```
training.info()
```

O resumo das informações dos quatro arquivos estão descritos na seção dois coleta de dados, nas tabelas 1, 2, 3 e 4. Utilizando a função *info* em cada dataset, foi possível gerar a seguinte tabela que resume o total de registros encontrados em cada dataset:

Tabela 5. Total de registros em cada dataset

Dataset	Total de registros
IdLookupTable	27124
SampleSubmission	27124
test	1783
training	7049

Até o presente momento estamos analisando todos os datasets. Porém, a partir desta etapa focaremos nossa atenção aos dados de treino, uma vez que esses dados serão utilizados para treinar nosso modelo. Na etapa de treino não será utilizada a feature *Image*, devido a esse fato foi removida essa feature do dataset utilizando a função *drop*, o que resulta em um novo dataset chamado *training_no_image*:

```
training_no_image = training.drop(['Image'], axis=1)
```

3.2 Listagem dos dados

Para visualizar o conteúdo de cada dataset foi utilizado a função *head* e *tail* passando como parâmetro o número de registros que queremos visualizar, dessa forma *head* retorna os registros superiores e *tail* os registros inferiores. Para tentar compreender como os dados estão dispostos e visualizá-los foi executado as funções head e tail:

```
training_no_image.head(5)
```

```
training_no_image.tail(5)
```

Figura 1 – Retorno das funções head e tail, visualizando 5 registros.

	left_eye_center_x	left_eye_center_y	right_eye_center_x	right_eye_center_y	left_eye_inner_corner_x	left
0	66.033564	39.002274	30.227008	36.421678	59.582075	
1	64.332936	34.970077	29.949277	33.448715	58.856170	
2	65.057053	34.909642	30.903789	34.909642	59.412000	
3	65.225739	37.261774	32.023096	37.261774	60.003339	
4	66.725301	39.621261	32.244810	38.042032	58.565890	

5 rows × 30 columns

	left_eye_center_x	left_eye_center_y	right_eye_center_x	right_eye_center_y	left_eye_inner_corner_x
7044	67.402546	31.842551	29.746749	38.632942	NaN
7045	66.134400	38.365501	30.478626	39.950198	NaN
7046	66.690732	36.845221	31.666420	39.685042	NaN
7047	70.965082	39.853666	30.543285	40.772339	NaN
7048	66.938311	43.424510	31.096059	39.528604	NaN

5 rows × 30 columns

3.3 Validação de dados faltantes

No retorno da função *tail*, verificando a feature *left_eye_inner_corner_x*, ficou constatado a existência de dados faltantes (NaN), o que é um problema para o processo de treino do modelo. Para tentar compreender o contexto geral em que está esse problema, precisamos verificar todo o dataset. Para realizar essa verificação foi utilizado as funções *isnull*, *any* e *value_counts*. A junção delas em uma única instrução ficou da seguinte forma:

```
training_no_image.isnull().any().value_counts()
```

O código acima gerou a seguinte saída:

```
True     28
False     3
dtype: int64
```

Apenas três features não tem dados faltantes, e vinte e oito features com dados faltantes. Para resolver esse problema podemos prosseguir de duas formas, imputar os dados ou remover esses registros do dataset. Partindo do princípio que o tempo de processamento diminui conforme reduz o volume de dados, os dados foram removidos com o intuito de otimizar o tempo de treino. A etapa de remoção de dados faltantes se caracteriza pela utilização da função *dropna*:

```
training_no_image = training_no_image.dropna()
```

No dataset de treino tínhamos um total de 7049 registros, após a remoção dos dados faltantes o dataset ficou com um total de 2140, sendo 4909 registros removidos.

Tabela 6. Total de registros por etapa

Dataset: training	Registros
Total inicial	7049
Total removidos	4909
Total final para treino	2140

3.4 Conversão de dados

Ao tentar visualizar a imagem na posição 0, ocorreu um erro de atributo não existente, o código assume que na posição 0 da feature Image tenha um *array* e não uma *string,* código utilizado para visualizar a imagem:

```
plt.imshow(training['Image'][0].reshape(96, 96), cmap='gray')
plt.axis('off')
plt.show()
```

Retorno com o erro:

```
AttributeError: 'str' object has no attribute 'reshape'
```

Executando a verificação do tipo de dado da feature *Image*, foi constatado que ela está como *string*, conforme mostra abaixo:

```
type(training['Image'][0])
```

Retorno:

```
str
```

Para corrigir esse problema os dados foram convertidos em um *ndarray* de dimensão 96x96. Para realizar isso foi executado a função *apply* que aplica em toda a feature *Image* a conversão de string para inteiros e redimensiona esse array para 96x96 posições:

```
training['Image'] = training['Image'].apply(lambda x:
np.fromstring(x, dtype=int, sep=' ').reshape((96,96)))
```

4. Análise e Exploração dos Dados

O dataset de treino fornecido traz a imagem bem como os pontos chaves faciais da própria imagem. Para uma visão preliminar apenas da imagem foi executado o seguinte:

Figura 2 – Visualizando uma imagem aleatória do dataset de treino.

```
plt.imshow(training['Image'][random.randint(0, 2139)].reshape(96,96), cmap='gray')
plt.axis('off')
plt.show()
```

Figura 3 – Visualização de uma imagem aleatória do dataset de treino.

Após verificar essa imagem foi gerado um novo script para visualizar três imagens e um determinado ponto chave facial. Foram gerados diferentes scripts para cada figura, no entanto a diferença do código de um script para outro é mínima, basicamente o que muda é a posição das imagens e dos pontos chaves faciais. No script a primeira linha cria o objeto do tipo *Figure*, passando o parâmetro *figsize* informando a largura e a altura da imagem em polegadas. A segunda linha ajusta os espaçamentos entre as imagens. Após tem a função *add_subplot* que cria uma grade com uma linha e três colunas para dispor as imagens, após foi criado um objeto chamado *red_patch* da classe *patches.Patch* da biblioteca *matplotlib* para atribuir textos e valores nas legendas, a função *imgshow* carrega a imagem na posição i, após é adicionado a legenda, e em seguida a função *plot*, adiciona os pontos chaves faciais na imagem, e na última linha é chamado a função *show* para visualizar as imagens, os pontos e as legendas.

Figura 4 – Script para visualizar três imagens e os pontos-chave faciais.

```
fig = plt.figure(figsize=(20, 20))
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.02, wspace=0.02)

for i in range(3):
    fig.add_subplot(1, 3, i + 1, xticks=[], yticks=[])
    red_patch = mpatches.Patch(color='red', label='left_eye_center   x='
                       + str(round(training.loc[i][0], 2))
                       +'   y=' + str(round(training.loc[i][1], 2)))
    plt.imshow(training['Image'][i], cmap='gray')
    plt.legend(handles=[red_patch], fontsize='xx-large')
    plt.plot(training.loc[i][0], training.loc[i][1], 'ro')
plt.show()
```

Figura 5 – Imagens com o ponto chave no centro do olho direito.

Figura 6 – Imagens com o ponto chave na ponta do nariz.

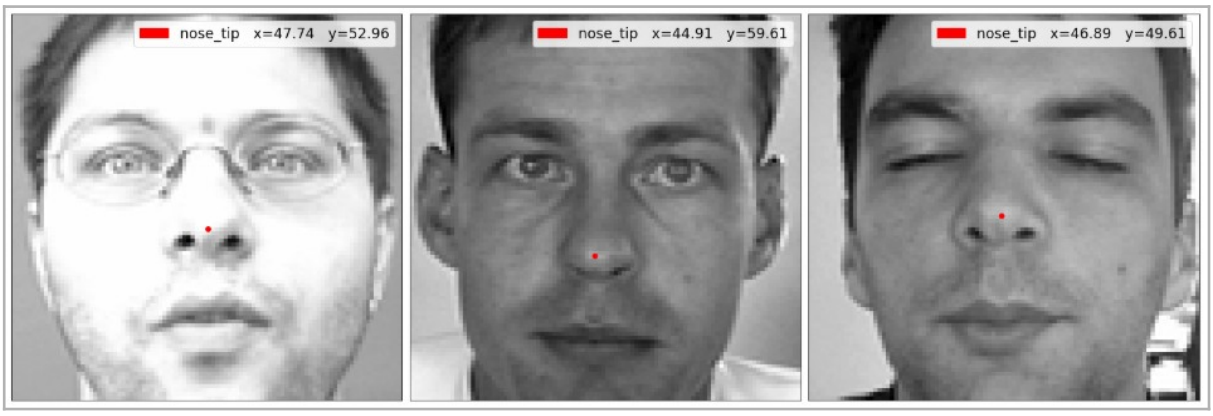
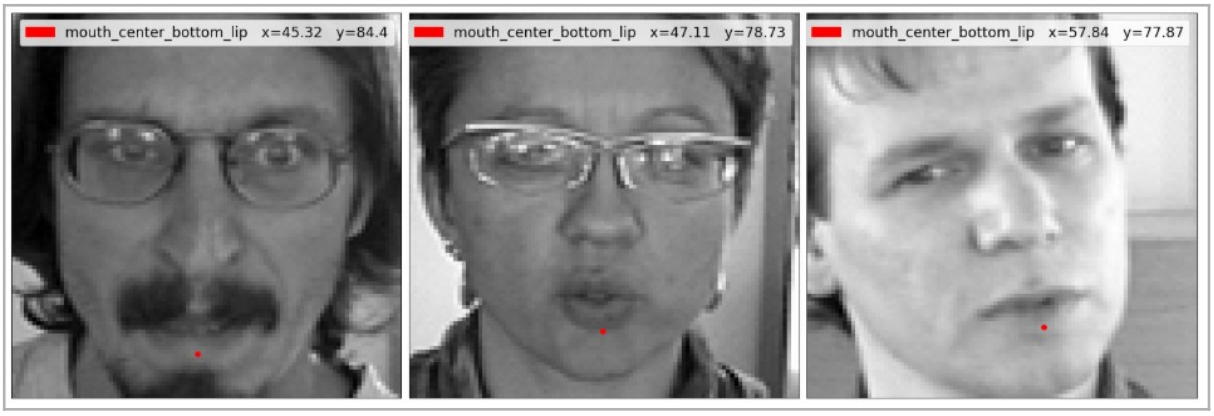
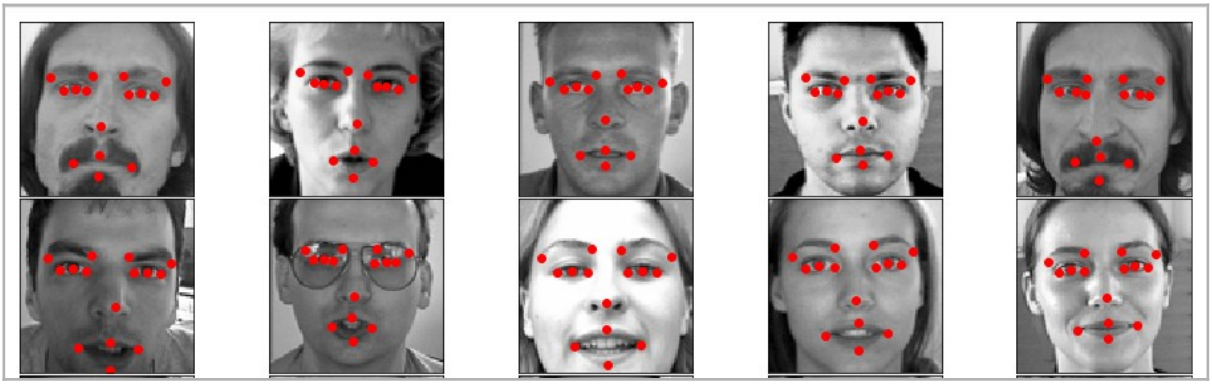


Figura 7 – Imagens com o ponto chave lábio inferior.



Após verificar essa imagem foi gerado um novo script para visualizar as trinta e cinco primeiras imagens com seus pontos chaves faciais.

Figura 8 – Visualização das imagens de treino junto com os pontos chaves faciais.



O objetivo dessa etapa de análise exploratória dos dados é visualizar as imagens bem como explorar o posicionamento dos pontos chaves sobre cada face. Com a visualização é possível verificar que algumas faces são da mesma pessoa, com pequenas variações de expressão, essas variações se resume a imagens de faces sorrindo outras com a face normal. É possível perceber que uma foto da mesma pessoa com uma pequena alteração na face é o suficiente para alterar as coordenas (x,y) dos pontos chaves faciais. Tendo em vista todo esses indicadores, pode-se então prosseguir com a criação do modelo.

5. Criação do Modelo de Deep Learning

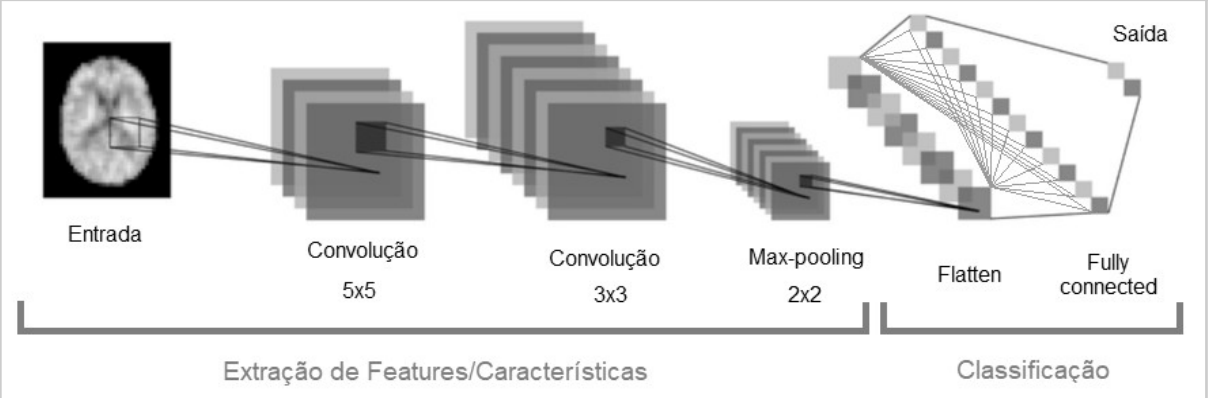
Para criar o modelo foi utilizado a API de deep learning Keras versão 2.3.1 em conjunto com a plataforma TensorFlow versão 2.0.0. O enfoque deste trabalho é a detecção de pontos chaves faciais, o qual é um subproblema da área de reconhecimento facial. A técnica amplamente utilizada para tratamento e reconhecimento de imagens é conhecida como rede neural convolucional, ou CNN, do inglês *Convolutional Neural Network*. Esse tópico será dividido em duas partes,

primeiro será abordado como funciona o aprendizado de máquina em uma rede neural convolucional, e o segundo a criação, treinamento e validação do modelo.

5.1 Redes Neurais Convolucionais

Uma CNN é uma arquitetura de aprendizado profundo que divide os dados para tentar extrair padrões. São muito utilizadas no processamento de fala e compreensão da linguagem natural utilizando camadas de uma dimensão com convoluções temporais, classificação e segmentação de imagens com camadas de duas dimensões com convoluções espaciais, e processamento de vídeos com camadas de três dimensões com convoluções volumétricas. Esse tipo de rede possui três partes principais, a camada convolucional, a camada de pooling e a camada densa (*convolutional layer, pooling layer* e *fully connected layer)*. A primeira camada é responsável por extrair as características, a segunda camada reduz a dimensão da rede, e a última camada é totalmente conectada, ou seja, seus neurônios possuem conexão com cada neurônio da camada anterior.

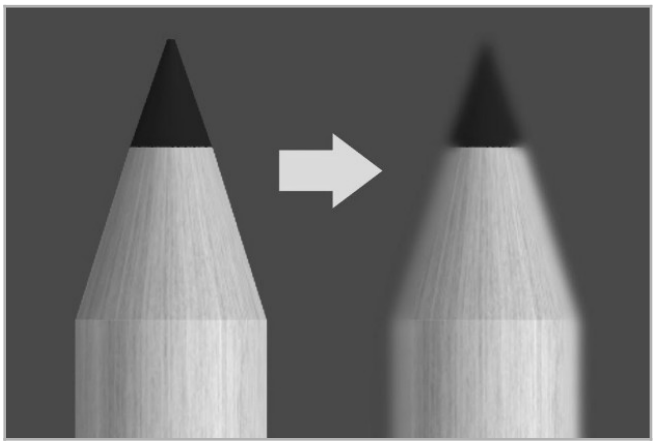
Figura 9 – Arquitetura básica de uma rede neural convolucional.



5.1.1 Camada Convolucional

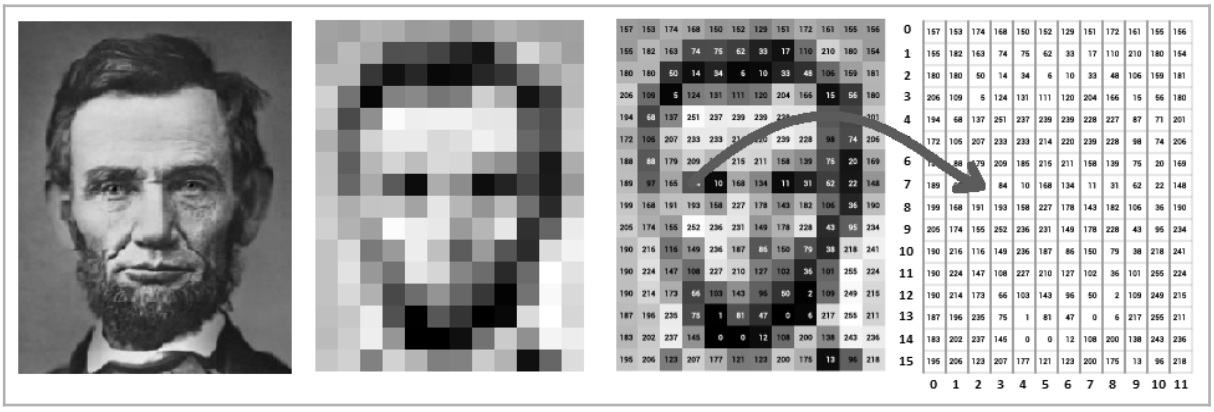
Muitos softwares utilizam a técnica da CNN, como por exemplo o Photoshop ou GIMP aplicam filtros de suavização, desfoque e detecção de bordas, esses filtros aplicam convoluções na imagem para realizar algum tipo de alteação. O software realiza alguma operação matemática com os pixeis da imagem e redesenha essa nova imagem na tela fazendo o efeito, como mostra a figura 10 abaixo.

Figura 10 – Imagem aplicando efeito de suavização de arestas.



Para que seja possível executar algum tipo de filtro, o software aplica convoluções sobre a imagem, o primeiro passo é converter a imagem em uma matriz de píxeis. Qualquer imagem possui uma dimensão específica de largura e altura, podendo ser 16x16, 96x64, 1200x960, etc. Uma forma de simplificar é criar a matriz com a mesma dimensão da imagem. No exemplo abaixo como mostra a figura 11 a matriz ficou com 12x16, onde cada posição x e y assume um valor da escala de cinza que varia de 0 a 255, quanto mais escuro menor, mais claro maior.

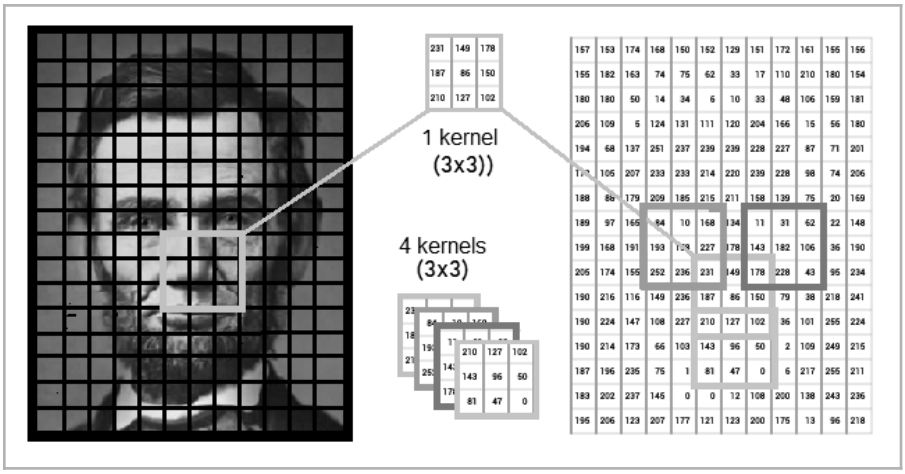
Figura 11 – Convertendo uma imagem em uma matriz.



Podemos afirmar que uma convolução é uma operação matemática entre duas funções (f e g) que produz uma terceira função h. O primeiro passo de uma convolução é criar os filtros também chamados de kernels, esses kernels são pequenas matrizes criadas a partir da própria imagem, (dessa forma, quando

treinamos um modelo como, por exemplo, imagens de árvores, ele irá somente conseguir identificar árvores, pois ele vai obter os kernels que identificará folhas, galhos, troncos e assim por diante) e durante esse processo de execução da CNN, os melhores kernels tornam-se as features do modelo.

Figura 12 – Criação de 4 filtros com tamanho 3x3.

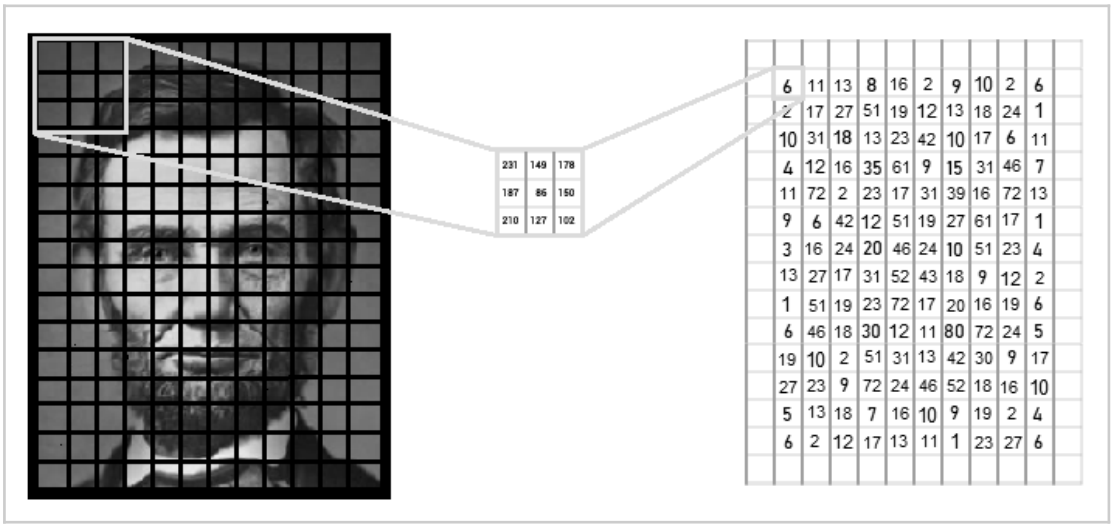


Um único modelo pode conter centenas senão milhares de kernels, a seleção deles é feita de forma aleatória, mas podemos definir algumas configurações, como qual será o tamanho do kernel, 1x1, 3x3, 5x5, assim por diante, quanto maior o nível de detalhe da imagem a ser tratada menor precisa ser o kernel, a busca pelo kernel que traz a melhor precisão pode variar em cada tipo de aplicação. Essas configurações como o tamanho do kernel entre outras que veremos são comumente chamadas de hiper-parâmetros.

5.1.1.1 Stride (Passo)

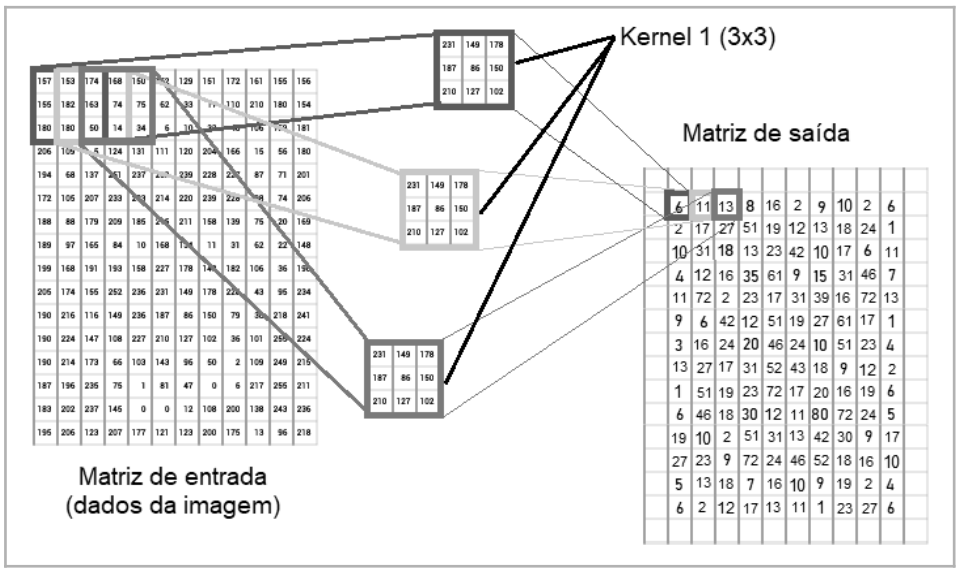
O stride ou passo, por exemplo, é outro hiper-parâmetro. Após a definição dos kernels, cada um é sobreposto na matriz iniciando na posição 0,0 deslizando da esquerda pra direita e de cima pra baixo, essa ação de deslizar caracteriza o passo ou *stride*. Em cada ponto de parada da matriz é executado alguma operação matemática entre os valores do kernel e os valores da imagem, (sendo que esses valores são os valores dos pixeis da imagem) gerando o terceiro valor. Conforme aumentamos o número do stride diminui o tamanho da nossa matriz resultante. Na figura 13 temos o primeiro passo da convolução em um kernel 3x3.

Figura 13 – Primeiro passo da convolução em um kernel 3x3.



Na figura 14 abaixo temos os três primeiros passos da convolução, o mesmo kernel desliza realizando um passo e para cada posição gera um valor, esse valor é adicionado em uma nova matriz de saída que será utilizada posteriormente.

Figura 14 – Execução de três passos no processo de convolução.

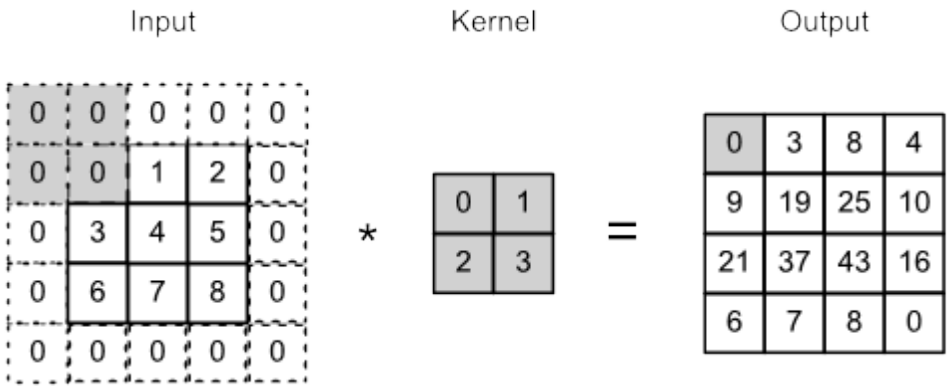


5.1.1.2 Padding (Preenchimento)

Quando os filtros deslizam sobre a matriz tanto no início como no fim da matriz temos as bordas, e nem sempre o tamanho do kernel junto com o passo encaixa exatamente com a largura ou altura da matriz. Esse problema pode ser corrigido usando o padding. Quando não queremos que o kernel ultrapasse a borda

da matriz configuramos o padding como *valid*, e quando queremos que ele ultrapasse as bordas da matriz configuramos o padding como *same*. No exemplo da figura 15 temos uma matriz 3x3, e um kernel 2x2, e stride igual a um, onde foi ajustado o padding como *same*, fazendo com que seja adicionado uma borda com zeros no contorno de toda a matriz.

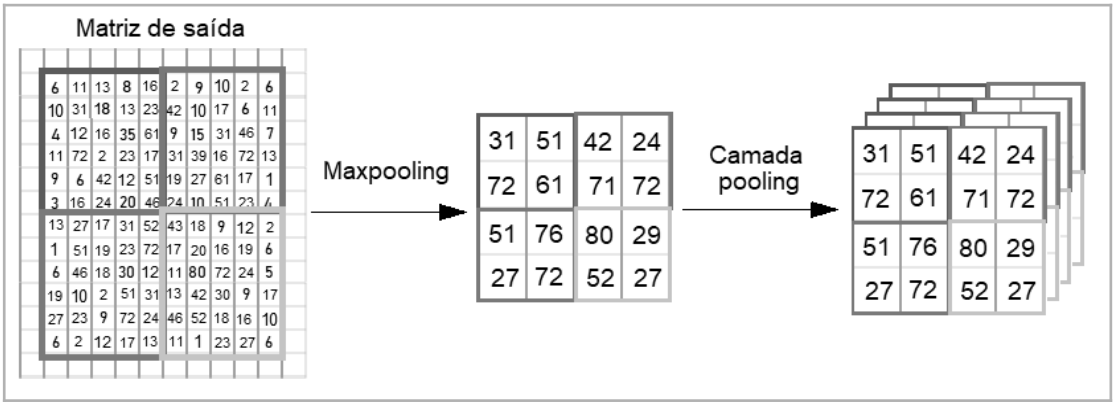
Figura 15 – Convolução de um kernel 2x2 em uma matriz 3x3 e padding igual a same.



5.1.2 Camada Pooling

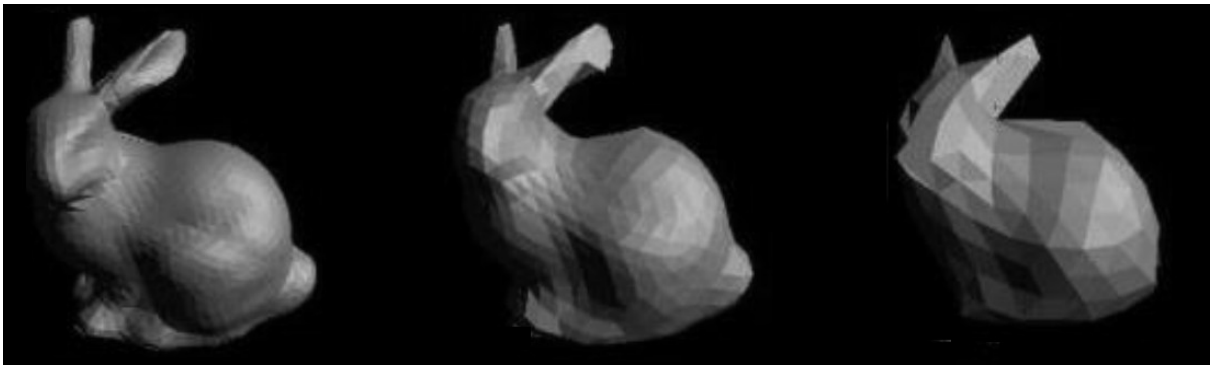
Após gerado a matriz de saída é iniciado o processo de pooling, o resultado desse processo é a geração das features maps. O processo consiste em aplicar funções em alguns subconjuntos de valores com o propósito de diminuir o volume de dados sem perder a uniformidade e variabilidade desses dados. A função aplicada na etapa de pooling tem algumas variações, como pegar o valor máximo, mínimo ou a média (*max*, *min* e *average*), cada tipo de problema é resolvido com um tipo de função específico.

Figura 16 – Processo de max-pooling gerando uma fatia das features maps.



O ponto relevante é que o pooling realiza operações que resultam na redução da dimensionalidade da matriz original, criando uma nova matriz com dimensões menores, porém, essa matriz menor mantêm as características principais da matriz original. De forma geral o processo de pooling tem dois objetivos, diminuir a quantidade de informação e manter as principais características da imagem como mostra a figura 17.

Figura 17 – Imagem aplicada por duas etapas de pooling.



5.1.3 Flatten

Todo esse processo de definição dos kernels, criação da matriz de saída e com ela as feature maps, pode ser realizado repetidas vezes até que satisfaça a resolução do problema. Após a última camada de pooling é criada uma única camada chamada *Flatten*, essa camada é um vetor com todas as *features* da última camada de pooling, ela é quem fornece as *features* para a camada totalmente conectada ou *fully connected layer* (sigla em inglês FCL).

5.1.4 Fully Connected Layer

Essa camada é uma rede totalmente conectada, com a camada inicial, podendo ter várias camadas intermediárias, e a camada de saída. Na camada inicial cada neurônio conecta com todas as features maps armazenadas no *Flatten*, e na saída cria uma conexão para cada neurônio da camada seguinte, e na última camada possui o número de neurônios de saída igual ao número de classes que queremos prever.

Na camada densa é onde ocorre a previsão, se a imagem de entrada pertence a alguma classe hipotética X ou Y. Os neurônios da penúltima camada avaliam as features atribuindo um valor entre 0 e 1 para cada feature, um valor de 0.9 significa 90% de certeza que essa feature é de uma classe específica. Após muitas iterações os neurônios descobrem que quando certas features são ativadas, então provavelmente a imagem pertence a classe X ou Y.

5.1.4.1 Função de Ativação

A função de ativação é responsável por definir a ativação de saída do neurônio em termos do seu nível de ativação interna. A camada totalmente conectada pode ser do tipo *Multi-Layer Perceptron*, que utiliza funções de ativação para classificar as imagens, existem vários tipos de funções de ativação como tangente hiperbólica, sigmóide e ReLU. A função popularmente utilizada para classificar e que apresenta bons resultados é a função ReLU ou *Rectified Linear Unit* (Função Linear Retificada). Após a rede treinada com algumas milhares de imagens, ela poderá afirmar com grande grau de certeza a qual classe pertence a imagem de entrada, em outras palavras a rede informa que a imagem de entrada tem 95% de chances de ser da classe X e 5% de chance de ser da classe Y.

5. 2 Modelo para classificar pontos chaves faciais

O Keras disponibiliza um conjunto de ferramentas para trabalhar com redes neurais convolucionais, ele oferece dois tipos de modelos, o Sequential API e o Functional API. O tipo *Sequential* permite criar uma rede neural camada por camada, porém é limitado, ele não permite criar modelos que compartilham camadas ou tenham várias entradas ou saídas, enquanto que o *Functional* é customizável, permitindo camadas compartilhadas e múltiplas camadas de entrada e saída. O problema proposto neste trabalho não necessita do compartilhamento entre camadas ou de múltiplas camadas de entrada e saída, portanto foram gerados os modelos utilizando *Sequential API*. Com o propósito de comparar modelos entre si, foram criados cinco modelos, cada um com alguma pequena alteração (mudança de hiper-parâmetro) em sua configuração, porém, essa pequena alteração fez com que cada modelo apesentasse diferentes resultados.

Para a criar o modelo o primeiro passo é criar uma variável do tipo *Sequential* chamada model, como segue o código abaixo:

```
model = Sequential()
```

Após a criação desta variável foi adicionado a ela as camadas, fazendo a construção da rede neural. Para criar a primeira camada da rede foi adicionado uma camada do tipo *Convolution2D*, passando por parâmetro o número de neurônios 32, o número de filtros da convolução 3x3, o tipo do padding como *same*, usa bias como *false* e a dimensão das imagens. O código abaixo cria a primeira camada da rede.

```
model.add(Convolution2D(32, (3,3), padding='same', use_bias=False, input_shape=(96,96,1)))
```

Somente na primeira camada é necessário informar o parâmetro *input_shape*, ele informa qual será a dimensão das imagens do dataset. Após a camada Convolution2D, foi adicionada uma camada de ativação do tipo LeakReLU com inclinação de 0.1, em seguida foi adicionada a camada BatchNoralization para normalizar alguma variação de escala, após foi adicionado uma camada do tipo MaxPool2D com tamanho 2x2, abaixo o código desenvolvido.

```
model.add(LeakyReLU(alpha = 0.1))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
```

Após adicionar as camadas acima citadas foram adicionadas mais camadas com a mesma estrutura porém com maior número de neurônios. Para finalizar a rede foi adicionado a camada Flatten, em seguida uma camada densa com 512 neurônios com a função de ativação ReLU, e para finalizar a camada de saída (densa) com 30 neurônios, onde cada um desses neurônios representa um ponto chave facial que estamos tentando prever para a imagem de entrada.

```
model.add(Flatten())
model.add(Dense(512, activation='relu'))
```

```
model.add(Dense(30))
```

Como citado no início deste tópico foi criado sete modelos para realizar um teste comparando qual modelo teve mais precisão e menos erro. Foi criado sete tabelas que descrevem a configuração das camadas dos modelos. Nas camadas Convolution2D a configuração fica por exemplo 32, 3x3, False, 96x96, onde temos respectivamente o número de filtros, tamanho do kernel, se usa bias (True ou False), e no caso da primeira camada é informado o input_shape. Na camada de LeakReLU é informado o alpha por exemplo 0.1. Na camada MaxPool2D é informado o pool_size, por exemplo 2x2. Na camada Dense é informado o número de neurônios, e na camada Dropout é informado o percentual de dados a ser descartado.

Tabela 5. Modelo 1

Nome da Camada	Configuração da Camada
Convolution2D	32, 3x3, False, 96x96
LeakReLu	0.1
BatchNormalization	-
Convolution2D	32, 3x3, False
LeakReLu	0.1
BatchNormalization	-
MaxPool2D	2x2
Convolution2D	64, 3x3, False
LeakReLu	0.1
BatchNormalization	-
Convolution2D	64, 3x3, False
LeakReLu	0.1
BatchNormalization	-
MaxPool2D	2x2
Convolution2D	128, 3x3, False
LeakReLu	0.1
BatchNormalization	-
Convolution2D	128, 3x3, False
LeakReLu	0.1
BatchNormalization	-
MaxPool2D	2x2
Convolution2D	256, 3x3, False
LeakReLu	0.1
BatchNormalization	-
Convolution2D	256, 3x3, False
LeakReLu	0.1
BatchNormalization	-
MaxPool2D	2x2
Convolution2D	512, 3x3, False
LeakReLu	0.1
BatchNormalization	-
Convolution2D	512, 3x3, False
LeakReLu	0.1
BatchNormalization	-
MaxPool2D	2x2
Flatten	-
Dense	512, relu
Dropout	0.1
Dense	30

Tabela 6. Modelo 2

Nome da Camada	Configuração da Camada
Convolution2D	32, 5x5, False, 96x96
LeakReLu	0.1
BatchNormalization	-
Convolution2D	32, 5x5, False
LeakReLu	0.1
BatchNormalization	-
MaxPool2D	2x2
Convolution2D	64, 5x5, False
LeakReLu	0.1
BatchNormalization	-
Convolution2D	64, 5x5, False
LeakReLu	0.1
BatchNormalization	-
MaxPool2D	2x2
Convolution2D	128, 5x5, False
LeakReLu	0.1
BatchNormalization	-
Convolution2D	128, 5x5, False
LeakReLu	0.1
BatchNormalization	-
MaxPool2D	2x2
Convolution2D	256, 5x5, False
LeakReLu	0.1
BatchNormalization	-
Convolution2D	256, 5x5, False
LeakReLu	0.1
BatchNormalization	-
MaxPool2D	2x2
Convolution2D	512, 5x5, False
LeakReLu	0.1
BatchNormalization	-
Convolution2D	512, 5x5, False
LeakReLu	0.1
BatchNormalization	-
MaxPool2D	2x2
Flatten	-
Dense	512, relu
Dropout	0.1
Dense	30

Tabela 7. Modelo 3

Nome da Camada	Configuração da Camada
Convolution2D	32, 7x7, False, 96x96
LeakReLu	0.1
BatchNormalization	-
Convolution2D	32, 7x7, False
LeakReLu	0.1
BatchNormalization	-
MaxPool2D	2x2
Convolution2D	64, 7x7, False
LeakReLu	0.1
BatchNormalization	-
Convolution2D	64, 7x7, False
LeakReLu	0.1
BatchNormalization	-
MaxPool2D	2x2
Convolution2D	128, 7x7, False
LeakReLu	0.1
BatchNormalization	-
Convolution2D	128, 7x7, False
LeakReLu	0.1
BatchNormalization	-
MaxPool2D	2x2
Convolution2D	256, 7x7, False
LeakReLu	0.1
BatchNormalization	-
Convolution2D	256, 7x7, False
LeakReLu	0.1
BatchNormalization	-
MaxPool2D	2x2
Convolution2D	512, 7x7, False
LeakReLu	0.1
BatchNormalization	-
Convolution2D	512, 7x7, False
LeakReLu	0.1
BatchNormalization	-
MaxPool2D	2x2
Flatten	-
Dense	512, relu
Dropout	0.1
Dense	30

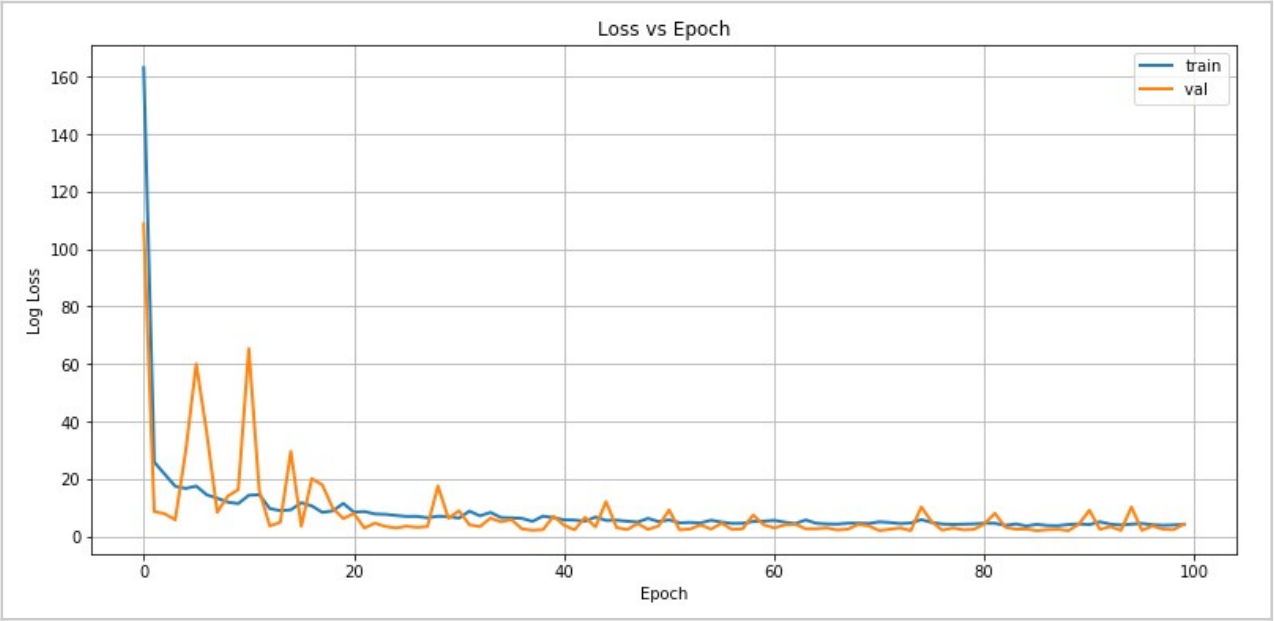
Foi elaborado três modelos iniciais, onde a única diferença entre eles é o tamanho do kernel, sendo o modelo 1 com o kernel 3x3, modelo 2 com 5x5 e o modelo 3 com 7x7. Após a elaboração os três modelos foram treinados e avaliados quanto a sua performance analisando o tempo de execução, o valor de *loss* e o valor de erro médio padrão.

6. Apresentação dos Resultados

Após a criação, treino e validação dos modelos foi elaborado uma tabela comparativa com os modelos e os resultados de precisão e de perda.

Modelo	Perda	Acurácia
Modelo 1	4,29	1,66
Modelo 2	3,87	1,48
Modelo 3	14,39	3,09
Modelo 4	2,28	1,09
Modelo 5	2,92	1,23

Figura 18 – Modelo 1, gráfico de perda por época.



7. Links

https://github.com/AlexandreNeuk/Facial-Keypoint-Detection

REFERÊNCIAS

BLEDSOE, Wilson Woodrow. **Pattern Recognition and Reading by Machine.** Disponível em: <https://dl.acm.org/doi/10.1145/1460299.1460326/> Acesso em: 15 fev. 2020.

DARPA. **Defense Advanced Research Projects Agency**, 2020. Disponível em: <https://www.darpa.mil/>. Acesso em: 16 fev. 2020.

Deeplearningbook. **Capítulo 10 – As 10 Principais Arquiteturas de Redes Neurais**, 2020. Disponível em:<http://deeplearningbook.com.br/as-10-principais-arquiteturas-de-redes-neurais/>. Acesso em: 03 fev. 2020.

Facefirst. **FaceFirst**, 2020. Disponível em:<https://www.facefirst.com/>. Acesso em: 16 fev. 2020.

Kaggle. **Deep Learning**. 2020. Disponível em: <https://www.kaggle.com/learn/deep-learning />. Acesso em: 15 fev. 2020.

Kaggle Desafio. **Facial Keypoints Detection**. 2020. Disponível em:<https://www.kaggle.com/c/facial-keypoints-detection/>. Acesso em: 21 jan. 2020.

Kaggle. **Intermediate Machine Learning**. 2020. Disponível em: <https://www.kaggle.com/learn/intermediate-machine-learning/>. Acesso em: 20 jan. 2020.

Kaggle. **Intro to Machine Learning**. 2020. Disponível em: <https://www.kaggle.com/learn/intro-to-machine-learning/>. Acesso em: 10 jan. 2020.

Kaggle. **Kaggle**, 2019. Disponível em:<https://www.kaggle.com/>. Acesso em: 19 set. 2019.

Kaggle. **Python**. 2020. Disponível em: <https://www.kaggle.com/learn/python/>. Acesso em: 20 jan. 2020.

Kaggle. **Rectified Linear Units (ReLU) in Deep Learning**. 2020. Disponível em: <https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning/>. Acesso em: 15 fev. 2020.

Keras Documentation. **Advanced Activations Layers**. 2020. Disponível em:<https://keras.io/layers/advanced-activations/>.Acesso em: 10 fev. 2020.

Keras Documentation. **Convolutional Layers**. 2020. Disponível em:<https://keras.io/layers/convolutional/>.Acesso em: 10 fev. 2020.

Keras Documentation. **Core Layers**. 2020. Disponível em: <https://keras.io/layers/core/>. Acesso em: 12 fev. 2020.

Keras Documentation. **Getting started with the Keras Sequential model**. 2020. Disponível em:<https://keras.io/getting-started/sequential-model-guide/>.Acesso em: 09 fev. 2020.

Keras Documentation. **Normalization Layers**. 2020. Disponível em:<https://keras.io/layers/normalization/>.Acesso em: 11 fev. 2020.

Keras Documentation. **Pooling Layers**. 2020. Disponível em:<https://keras.io/layers/pooling/>.Acesso em: 12 fev. 2020.

Keras Documentation. **Sequential**. 2020. Disponível em:<https://keras.io/models/sequential/>.Acesso em: 11 fev. 2020.

Mastercard Identity Check. **Mastercard Identity Check**, 2020. Disponível em:<https://pt.wikipedia.org/wiki/Sistema_de_reconhecimento_facial/>. Acesso em: 18 fev. 2020.

MEDIUM. **Activation Functions: Sigmoid, ReLU, Leaky ReLU and Softmax basics for Neural Networks and Deep Learning.** 2020. Disponível em:<https://medium.com/tensorflow/standardizing-on-keras-guidance-on-high-level-apis-in-tensorflow-2-0-bad2b04c819a/>.Acesso em: 15 fev. 2020.

MEDIUM. **Glossary of Deep Learning: Activation Function.** 2020. Disponível em:<https://medium.com/deeper-learning/glossary-of-deep-learning-activation-function-20262327a907/>.Acesso em: 15 fev. 2020.

MEDIUM. **Standardizing on Keras: Guidance on High-level APIs in TensorFlow 2.0.**2020. Disponível em:<https://medium.com/tensorflow/standardizing-on-keras-guidance-on-high-level-apis-in-tensorflow-2-0-bad2b04c819a/>.Acesso em: 15 fev. 2020.

NIST. **National Institute of Standards and Technology.** Disponível em: <https://www.nist.gov /> Acesso em: 15 fev. 2020.

TANZ, Leo. **Recognition of faces in the presence of two-dimensional sinusoidal masks.** Disponível em: <https://link.springer.com/content/pdf/10.3758/BF03208310.pdf/>. Acesso em: 15 jan. 2020.

Towardsdatascience. **Introduction to Data Preprocessing in Machine Learning**. 2020. Disponível em:<https://towardsdatascience.com/introduction-to-data-preprocessing-in-machine-learning-a9fa83a5dc9d/>. Acesso em: 14 fev. 2020.

WIKIPEDIA. **Defense Advanced Research Projects Agency**. 2020. Disponível em: <https://en.wikipedia.org/wiki/DARPA />. Acesso em: 16 fev. 2020.

WIKIPEDIA. **Facial Recognition Sysrtem**, 2020. Disponível em:<https://en.wikipedia.org/wiki/Facial_recognition_system/>. Acesso em: 16 fev. 2020.

WIKIPEDIA. **Facial Recognition Technology**, 2020. Disponível em:<https://en.wikipedia.org/wiki/FERET_(facial_recognition_technology)/>. Acesso em: 16 fev. 2020.

WIKIPEDIA. **National Institute of Standards and Technology**, 2020. Disponível em:<https://en.wikipedia.org/wiki/National_Institute_of_Standards_and_Technology/>. Acesso em: 16 fev. 2020.

WIKIPEDIA. **Perceptron multicamadas**, 2020. Disponível em:<https://pt.wikipedia.org/wiki/Perceptron_multicamadas/>. Acesso em: 22 jan. 2020.

WIKIPEDIA. **Sistema de reconhecimento facial**, 2020. Disponível em:<https://pt.wikipedia.org/wiki/Sistema_de_reconhecimento_facial/>. Acesso em: 16 fev. 2020.

WIKIPEDIA. **Woody Bledsoe.** 2020. Disponível em: <https://en.wikipedia.org/wiki/Woody_Bledsoe/>. Acesso em: 15 fev. 2020.

YOUTUBE. **Deep learning – Redes Neurais Convolucionais**, 2020. Disponível em:<https://www.youtube.com/watch?v=DXnyuUZcAAI/>. Acesso em: 05 fev. 2020.

YOUTUBE. **Redes neurais convolucionais – Parte 1 - Fundamentos**, 2020. Disponível em:<https://www.youtube.com/watch?v=n4rmrZg1_58/>. Acesso em: 26 fev. 2020.

YOUTUBE. **Redes Neurais profundas Convolucionais - Parte II - Arquiteturas Modernas**. 2020. Disponível em:<https://www.youtube.com/watch?v=0XUrLfQXzcw&t=1939s/>. Acesso em: 26 fev. 2020.