Reconhecimento Facial Utilizando Redes Neurais Convolucionais Para Auxiliar na Automatização de Processos no Campus Santo Antônio de Jesus do IFBA

Daniel Almeida Caitano[1], Orientador: Ikaro Campos de Araújo[2]

[1] Instituto Federal de Educação, Ciência e Tecnologia da Bahia, Campus Santo Antônio de Jesus
danielcaitano9@gmail.com
[2] Instituto Federal de Educação, Ciência e Tecnologia da Bahia, Campus Santo Antônio de Jesus
ikarocampos75@gmail.com

Resumo. O objetivo deste trabalho é desenvolver um processo de reconhecimento facial para automatizar tarefas no IFBA, campus Santo Antônio de Jesus, a fim de melhorar e facilitar a rotina diária na universidade. Para realizar este objetivo foram utilizadas redes neurais convolucionais. Com a coleta de imagens a partir de *datasets* extraídos do site *kaggle*. Estas imagens foram treinadas e analisadas com o intuito de gerar uma rede neural com uma boa taxa de precisão. Com o resultado obtido, se espera que a rede possa ser aprimorada e utilizada na automatização de tarefas na instituição.

1 Introdução

Segundo BURNS et al. (2022) a Inteligência Artificial (IA) é a simulação de processos de inteligência humana por máquinas, especialmente sistemas de computador. Aplicações específicas de IA incluem sistemas especialistas, processamento de linguagem natural, reconhecimento de fala e visão de máquina. Graças a Inteligência Artificial é possível conseguir uma análise mais precisa de dados complexos, aperfeiçoamento de processos e ganho de uma maior produtividade.

O Aprendizado de Máquina *(machine learning)* é um dos seguimentos da Inteligência Artificial em que as máquinas executam seus trabalhos por conta própria. Utilizando, para isso, algoritmos que ajudam na tomada de decisão e na interpretação de dados. Com isso, as máquinas não realizam apenas funções específicas.

Isso se difere dos sistemas tradicionais, pois as novas técnicas de *Machine Learning* não precisam seguir necessariamente regras, nem sequências lógicas preestabelecidas de forma escrita, que determinam o fluxo do *software*. Mas sim, aprendem e produzem conhecimento a partir de dados que estão disponíveis, definindo e especificando as regras a seguir.

A Aprendizagem Profunda (*Deep Learning*) de acordo com RUSK (2016) é uma poderosa forma de *machine learning* que permite que computadores resolvam problemas como reconhecimento de fala e imagem. A *Deep Learning* realiza seu

aprendizado através de redes neurais para facilitar suas operações e procedimentos.

Com a ajuda do *Deep Learning* e o seu funcionamento utilizando várias camadas para processar os dados, e sua característica de não linearidade da rede, é possível que os programadores consigam representações que tenham uma expressiva taxa de acerto e que se adaptem às diversas variações dos dados apresentados.

A motivação deste trabalho é auxiliar o desenvolvimento de futuros processos envolvendo IA, no Instituto Federal da Bahia (IFBA) campus Santo Antônio de Jesus, e mostrar que um software inicial de reconhecimento facial ajudaria a facilitar e agilizar logins, entradas, presenças e demais tarefas no instituto. Com um sistema de reconhecimento facial podemos ter um sistema de presença automática, identificação dos alunos para conceder acessos e/ou permissões, controle de entrada e saída no campus entre outras utilidades. Segundo Silva Filho (2022), o uso da Inteligência Artificial permite a obtenção de uma maior produtividade e agilidade de processos.

Sendo assim, este trabalho tem como objetivo construir uma Rede neural Convolucional (CNN) para reconhecimento facial.

2 Fundamentação teórica

O trabalho de reconhecimento facial para detecção de emoções utilizando redes neurais convolucionais com *tensorflow* de FERREIRA (2021) é uma contribuição relevante, considerando que, tem como objetivo realizar o reconhecimento através do treinamento de uma rede neural utilizando a biblioteca *TensorFlow.js* e o módulo *Keras*. A biblioteca *"Tensorflow"* provê acesso às funções de manipulação e treinamento de uma rede neural. Já o módulo *Keras*, permite a criação dos modelos e o uso da base de dados.

Após importar as imagens para a CNN os valores de entrada dos dados são normalizados para números que variam entre 0 e 1 e a saída de classificação também possui variação entre 0 e 1. As funções de ativação já normalizam os dados para esse intervalo. Em seguida, os dados são separados para o treinamento, validação e teste com a biblioteca *sklearn* utilizando o método *train_test_split().* As camadas da rede foram configuradas com o *keras* utilizando os seguintes atributos:

- Número de filtros: que define a quantidade de filtros de saída na convolução;
- Tamanho da janela: que especifica a altura e largura da janela de convolução 2D;
- Função de ativação: função de ativação a ser usada;
- Dimensões de entrada: dimensões dos dados nas entradas;
- Formato da entrada: ordem das dimensões nas entradas;
- Regularização do kernel: função *regularizer* aplicada ao vetor de polarização.

Estes parâmetros tiveram seus valores variados em quatro modelos de CNN diferentes, conforme as imagens a seguir.

```
# Número de filtros da rede neural convolucional
num_features = 64
# Quantidade de classes de resposta da rede neural
num_labels = 7
# Quantidade de registros a serem verificados antes de fazer a troca dos pesos da rede neural
batch_size = 64
# Número máximo de épocas que esse algoritmo irá executar
epochs = 100
# Altura e largura das imagens
width, height = 48, 48
```

Fig. 1. Parâmetros utilizados pelo primeiro modelo da rede neural.

```
model = Sequential()

model.add(Conv2D(num_features, kernel_size=(3,3), activation='relu',
                 input_shape=(width, height, 1), data_format = 'channels_last',
                 kernel_regularizer = l2(0.01)))
model.add(Conv2D(num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(2*2*2*num_features, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(2*2*num_features, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(2*num_features, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(num_labels, activation = 'softmax'))
```

Fig. 2. Definição do primeiro modelo da rede neural utilizando o *Keras API.*

```
# Número de filtros da rede neural convolucional
num_features = 64
# Quantidade de classes de resposta da rede neural
num_labels = 7
# Quantidade de registros a serem verificados antes de fazer a troca dos pesos da rede neural
batch_size = 64
# Número máximo de épocas que esse algoritmo irá executar
epochs = 100
# Altura e largura das imagens
width, height = 48, 48
```

Fig. 3. Parâmetros utilizados pelo segundo modelo da rede neural.

```
model = Sequential()

model.add(Conv2D(num_features, kernel_size=(3,3), padding = 'same', kernel_initializer="he_normal",
                 input_shape = (width, height, 1)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(num_features, kernel_size=(3,3), padding = "same", kernel_initializer="he_normal",
                 input_shape = (width, height, 1)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(2*num_features, kernel_size=(3,3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(2*num_features, kernel_size=(3,3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(2*2*num_features, kernel_size=(3,3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*num_features, kernel_size=(3,3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(2*2*2*num_features, kernel_size=(3,3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*2*num_features, kernel_size=(3,3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(2*num_features, kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(2*num_features, kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(num_labels, kernel_initializer="he_normal"))
model.add(Activation("softmax"))
```

Fig. 4. Definição do segundo modelo da rede neural utilizando o *Keras API.*

```
# Número de filtros da rede neural convolucional
num_features = 32
# Quantidade de classes de resposta da rede neural
num_labels = 7
# Quantidade de registros a serem verificados antes de fazer a troca dos pesos da rede neural
batch_size = 16
# Número máximo de épocas que esse algoritmo irá executar
epochs = 100
# Altura e largura das imagens
width, height = 48, 48
```

Fig. 5. Parâmetros utilizados pelo terceiro modelo da rede neural.

```
model = Sequential()

model.add(Conv2D(num_features, kernel_size=(3,3), activation='relu',
                 input_shape=(width, height, 1), data_format = 'channels_last',
                 kernel_regularizer = l2(0.01)))
model.add(Conv2D(num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))
model.add(Conv2D(2*2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(2*2*2*num_features, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2*2*num_features, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2*num_features, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_features, activation='relu'))
model.add(Dropout(0.5))


model.add(Dense(num_labels, activation = 'softmax'))

adam = optimizers.Adam(lr = 0.001)

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

Fig. 6. Definição do terceiro modelo da rede neural utilizando o *Keras API.*

```
# Quantidade de classes de resposta da rede neural
num_labels = 7
# Quantidade de registros a serem verificados antes de fazer a troca dos pesos da rede neural
batch_size = 256
# Número máximo de épocas que esse algoritmo irá executar
epochs = 100
# Altura e largura das imagens
width, height = 48, 48
```

Fig. 7. Parâmetros utilizados pelo quarto modelo da rede neural.

```
num_labels = 7
batch_size = 256
epochs = 100
width, height = 48, 48

model = Sequential()

model.add(Conv2D(20, kernel_size=(3,3), padding='same', activation='relu', input_shape=(width, height, 1)))
model.add(Conv2D(30, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(40, kernel_size=(3,3), padding='same', activation='relu'))
model.add(Conv2D(50, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(60, kernel_size=(3,3), padding='same', activation='relu'))
model.add(Conv2D(70, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(80, kernel_size=(3,3), padding='same', activation='relu'))
model.add(Conv2D(90, kernel_size=(3,3), padding='same', activation='relu'))

model.add(Flatten())

model.add(Dense(1000, activation='relu'))
model.add(Dense(512, activation='relu'))

model.add(Dense(num_labels, activation='softmax'))

model.summary()
```

Fig. 8. Definição do quarto modelo da rede neural utilizando o *Keras API.*

FERREIRA (2021) notou que os modelos possuem um rápido grau de aprendizagem. A validação da acurácia foi feita comparando os valores da emoção retratada na imagem, e as predições do modelo. Os acertos foram somados e depois divididos pela quantidade de imagens presentes na base de dados de teste que é composta por 3589 imagens. O modelo 1 teve acurácia de 61,38%, O modelo 2 teve acurácia de 63,80%, o modelo 3 teve 59,77% e o 4 teve 57,62%.

DAMIÃO et al. (2021) no seu trabalho de Sistema de Controle de Acesso Utilizando Reconhecimento Facial, considerou as vantagens de uma CNN, e restringiu-se a soluções que utilizavam a linguagem Javascript. Foi escolhida a *Face-API.js*, uma *API - Application Programming Interface* (Interface de

Programação de Aplicações), em *Javascript* para reconhecimento facial implementada a partir do núcleo do *Tensorflow.js*.

Para a realização da detecção facial, DAMIÃO et al. (2021) utiliza um modelo *Tiny Face Detector*. O foco da sua rede é mantê-la performática utilizando poucos recursos computacionais, para poder ser utilizada em diferentes plataformas. Com relação a descrição do rosto ele utilizou um vetor de 128 posições. A *Face-API.js*, tem uma arquitetura parecida com a *ResNet-34* para programar uma CNN similar à biblioteca *dlib* quando utilizada para reconhecimento facial. Segundo DAMIÃO et al. (2021) esta rede consegue 99.38% de precisão no LFW, *Labeled Faces in the World* - Rostos Rotulados em Ambientes Não-Controlados, este é um conhecido repositório para estudo de reconhecimento facial.

COSTA (2022), no seu trabalho de reconhecimento de pessoas com e sem máscara usando redes neurais convolucionais. Utilizou um *dataset* contendo 19.544 imagens, das quais 6.832 imagens correspondiam a pessoas com máscara e 12.712 imagens correspondiam a pessoas sem máscara. As imagens foram redimensionadas para 224 x 224 pixels.

A biblioteca utilizada foi a *torchvision* do *PyTorch* que já trazia redes neurais pré-treinadas. Foram usados 6 modelos de CNNs no trabalho: *Alexnet, VGG16, Squeezenet, Mobilenet V2, Mobilenet V3 Large e Small*. Estes modelos foram selecionados para fazerem a transferência de aprendizado (*transfer learning*), permitindo assim, um excelente desempenho para classificar conjuntos de imagens com um grande número de classes.

Segundo COSTA (2022) a avaliação dos modelos foi usando 5.000 imagens das 19.544 para fazer o treinamento da rede, 5.000 para a validação e o restante para teste. A rede *Alexnet* teve acurácia de 97,63%, *VGG16* teve 98,76%, *Squeezenet* teve 98,04%, *Mobilenet V2* teve 96,87%, *Mobilenet V3 Large e Small* tiveram 96,43% e 95,75% respectivamente.

3 Metodologia

Este trabalho utilizou uma abordagem de processamento de imagens e reconhecimento de padrões para detectar e conseguir realizar a classificação das imagens.

O código foi desenvolvido em *javascript* utilizando a biblioteca *ml5.js* que é uma interface *open source* de alto nível que fornece acesso a algoritmos e modelos de aprendizado de máquina no navegador, com base no *TensorFlow.js.* Também foi utilizada a biblioteca *p5.js* que possui um conjunto de funcionalidades para trabalhar com objetos HTML5 para texto, vídeo, imagem, webcam e som, além de desenhos com o canvas.

O projeto foi iniciado utilizando a biblioteca Javascript *ml5.js* para a criação de uma rede neural convolucional para fazer a classificação das imagens. As imagens precisaram ser redimensionadas para todas possuírem o tamanho de entrada da rede e esta tarefa foi realizada utilizando o método de redimensionamento de imagens do *p5.js.*

3.1 Dataset

O *dataset* (conjunto de dados) foi construído com conjuntos de dados de pessoas famosas da história (escritores, atletas, estrelas pop etc.) do site *Kaggle*. Para este estudo foi usado imagens de 30 pessoas e cada pessoa contém cerca de 33 imagens, o que dá um total aproximado de 990 fotos. A organização das imagens foram categorizadas por pastas, cada pessoa possui sua própria pasta com suas fotos. Essas imagens foram escavadas usando o pacote *python icrawler* pelos

criadores dos *datasets*. As imagens possuem tamanhos diferentes, por isso precisaram ser redimensionadas para todas terem o mesmo tamanho.

O dataset utilizado nesse treinamento são caracterizados pelo conjunto de imagens de pessoas famosas Fig. 9 , 10 e 11.

Fig. 9. Foto presente em um dos *datasets* da apresentadora norte-americana Oprah Winfrey.

Fig. 10. Foto presente em um dos *datasets* da atriz norte-americana Meryl Streep.

Fig. 11. Foto presente em um dos *datasets* do ator norte-americano Chris Evans.

3.2 Arquitetura do Sistema

O código da CNN segue a estrutura típica de um processamento digital de imagens com sistema de reconhecimento de padrões utilizando o *ml5.js* e consiste em cinco etapas:
Etapa 1: Carregar dados ou criar alguns dados (*data augmentation*);
Etapa 2: Definir as opções de rede neural e inicializá-la;
Etapa 3: Normalizar dados à rede neural;
Etapa 4: Treinar a rede neural;
Etapa 5: Usar o modelo treinado para fazer uma classificação.

3.2.1 Carregar dados ou criar alguns dados: Nesta etapa, 25 imagens de cada pessoa foram carregadas com ajuda do *p5.js* com o método de carregar imagem*(loadImage*) que permite que as fotos sejam carregadas de um caminho e se crie um objeto *Image* a partir dela.

Após serem carregadas, as imagens foram redimensionadas, pois possuíam tamanhos diferentes. Para fazer isso foi usado o método de redimensionamento do *p5.js (resize)* que permite redimensionar a imagem para uma nova largura e altura. As novas dimensões de largura e altura de imagem foram 96 pixels. O redimensionamento das imagens ajuda a diminuir o número de pixels da foto e com isso reduzir o tempo de treinamento da rede neural.

3.2.2 Definir as opções de rede neural e inicializá-la: Nesta etapa, a rede foi inicializada utilizando o método de rede neural do *ml5.js (neuralNetwork*). Alguns parâmetros precisaram ser passados*,* como *task* que define se é uma tarefa de regressão (*regression*) ou classificação (*classification*). O ml5.js também possui uma opção de classificação de imagem (*imageClassification*). *ImageClassfication* foi a opção usada. Outro parâmetro utilizado foi o *input* no qual especificamos a largura(*width*) e altura(*height*) das imagens e seus canais (*channels*) foi utilizado um *width* e um *height* de 96 pixels e 4 *channels* – RGBA (*Red, Blue, Green and Alpha*).

A rede foi configurada com base na arquitetura *resnet* conforme a fig. 12:

Model Summary

Layer Name	Output Shape	# Of Params	Trainable
conv2d_Conv2D1	[batch,48,48,64]	12,608	true
batch_normalization_BatchNormalization1	[batch,48,48,64]	256	true
activation_Activation1	[batch,48,48,64]	0	true
max_pooling2d_MaxPooling2D1	[batch,24,24,64]	0	true
conv2d_Conv2D2	[batch,24,24,64]	36,928	true
batch_normalization_BatchNormalization2	[batch,24,24,64]	256	true
activation_Activation2	[batch,24,24,64]	0	true
conv2d_Conv2D3	[batch,24,24,64]	36,928	true
batch_normalization_BatchNormalization3	[batch,24,24,64]	256	true
activation_Activation3	[batch,24,24,64]	0	true
conv2d_Conv2D4	[batch,24,24,64]	36,928	true
batch_normalization_BatchNormalization4	[batch,24,24,64]	256	true
activation_Activation4	[batch,24,24,64]	0	true
conv2d_Conv2D5	[batch,24,24,64]	36,928	true
batch_normalization_BatchNormalization5	[batch,24,24,64]	256	true
conv2d_Conv2D6	[batch,24,24,256]	295,168	
batch_normalization_BatchNormalization6	[batch,24,24,256]	1,024	
activation_Activation6	[batch,24,24,256]	0	
conv2d_Conv2D7	[batch,24,24,256]	590,080	
batch_normalization_BatchNormalization7	[batch,24,24,256]	1,024	
activation_Activation7	[batch,24,24,256]	0	
conv2d_Conv2D8	[batch,24,24,512]	1,180,160	
batch_normalization_BatchNormalization8	[batch,24,24,512]	2,048	
activation_Activation8	[batch,24,24,512]	0	
conv2d_Conv2D9	[batch,24,24,512]	2,359,808	
batch_normalization_BatchNormalization9	[batch,24,24,512]	2,048	
activation_Activation9	[batch,24,24,512]	0	
global_average_pooling2d_GlobalAveragePooling2D1	[batch,512]	0	
dense_Dense1	[batch,30]	15,390	

Fig. 12. Resumo do modelo gerado pelo ml5.js

O modelo *resnet* possuí como características:

- *Residual Connections* para tornar o modelo pelo menos tão bom quanto as redes mais rasas, mas também com potencial de se tornar superior. Nas redes neurais, os dados fluem por cada camada sequencialmente: a saída de

uma camada é a entrada para a próxima camada. A conexão residual fornece outro caminho para os dados alcançarem as últimas partes da rede neural, ignorando algumas camadas;

- *batch normalization* é aplicado após cada camada convolucional para melhorar a estabilidade, velocidade e desempenho do processo de treinamento.

3.2.3 Normalizar dados à rede neural: Nesta etapa adicionamos os dados para a rede neural e foi utilizado o método de adicionar dados (*addData*) do *ml5.js*. Após isso, os dados precisaram ser normalizados. Para fazer isto foi utilizado o método *normalizeData* do *ml5.js*, assim os pixels das imagens foram convertidos para valores entre 0 e 1.

A normalização dos dados é uma técnica que ajuda a resolver o problema do intervalo das variáveis que pode acabar sendo muito diverso. Se utilizar a escala padrão da imagem pode acabar sendo colocado mais pesos nas variáveis com maior variação. O objetivo da normalização é garantir que os dados estejam o mais próximo possível da mesma escala.

3.2.4 Treinar a rede neural: Nesta etapa a rede neural convolucional teve seu treinamento utilizando o método *train* do *ml5.js.* O parâmetro *epochs* teve de ser usado para saber o número de épocas de treinamento, o número de 400 épocas foi escolhido, o tamanho do batch (batch size) foi utilizado o número 75. Ao finalizar o treinamento o *ml5.js* retorna um resumo do modelo usado na Fig 12 e a performance do treinamento na Fig. 13.

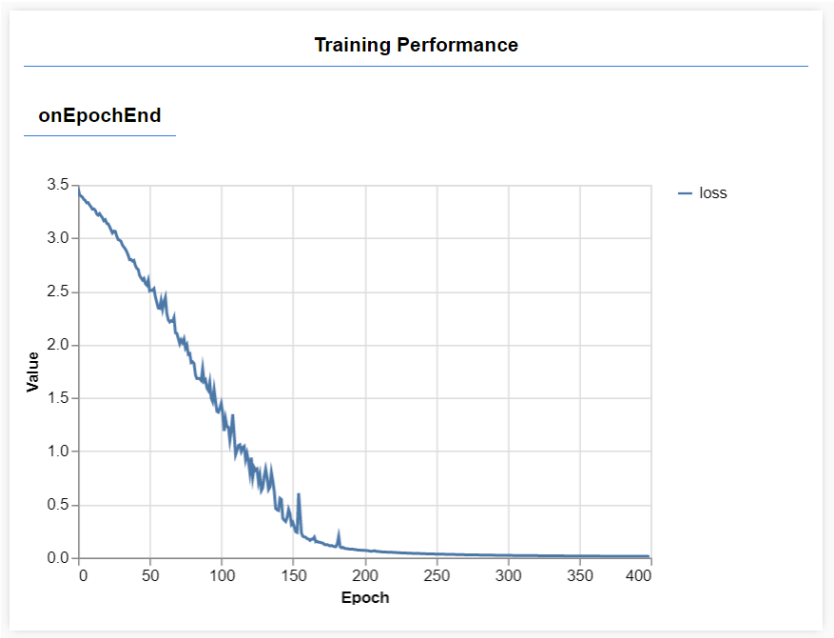

Fig. 13. Performance do treinamento gerado pelo ml5.js

Após o modulo ser treinado foi utilizado o método *save* do *ml5.js* que permite salvar os dados que acabaram de ser treinados.

3.2.5 Usar o modelo treinado para fazer uma classificação: Primeiramente é

importante ressaltar que alguns fatores relevantes mostraram serem impactantes na acurácia e precisão do resultado de classificação da rede neural, como uso de acessórios (óculos, bonés, toucas, chapéus), cortes de cabelo e cores de cabelo diferentes em cada foto, imagens com e sem barba além de formato da barba, fotos com anos de diferença (o que pode ocasionar aumento de peso corporal, envelhecimento), qualidade da imagem e distância da imagem.

Nesta etapa, em primeiro momento, as imagens que foram usadas nos passos anteriores para treinar a rede neural convolucional foram usadas como *input* para verificar como a rede classificava as imagens. Após os testes a rede retornou uma média de confiança (*confidence*) de aproximadamente 99%. Em seguida foram usadas 8 imagens, que não foram usadas para fazer o treinamento da rede neural, de cada pessoa para avaliar a classificação da rede, desta vez a rede retornou uma média de confiança de 72,37%.

3.3 Considerações éticas

Os *datasets* utilizados possuem licença de domínio público (CC0) que significa "*no rights reserved*" e foram obtidos através do site *kaggle*. Esta licença é uma condição jurídica na qual uma obra não possui direito autoral, não podendo ter restrição de uso. A obra fica em domínio público, ou seja, se torna livre e gratuita.

4 Conclusão

Este trabalho desenvolveu uma rede neural convolucional baseada em reconhecimento de imagens, utilizando para isso técnicas de *machine learning* e *deep learning* através da biblioteca *ml5.js*.

O algoritmo foi desenvolvido de acordo com as expectativas iniciais, permitindo o reconhecimento de pessoas utilizando uma rede neural convolucional. Graças a média de confiança de 72,37%, é possível dizer que o projeto já apresenta grau inicial de desenvolvimento que permite a realização de testes com estudantes, e com isso verificar sua utilização e desempenho, para assim conseguir aprimorar a rede neural, como também identificar possíveis novas funcionalidades.

Apesar de o código ser construído utilizando o *ml5.js*, a rede construída pode ser replicada, através do modelo apresentado no trabalho, em outras bibliotecas ou linguagens de programação.

O código da CNN se encontra no repositório RECONHECIMENTO FACIAL (2023). Com o resultado, espera-se que a rede possa ser aprimorada e utilizada para automatizar algumas tarefas no IFBA campus Santo Antônio de Jesus, a fim de melhorar e, facilitar as atividades no campus. Essa automatização trará mais simplicidade para a rotina de movimentação na universidade. Com ela, será possível obter identificação dos alunos, e assim realizar o registro de presença na sala de aula, empréstimos de livros da biblioteca, controle de entrada e saída no campus, entre outras utilidades que vão permitir conceder acessos e/ou permissões.

Referências

1. BANERJEE, Sourav. Indian Actor Images Dataset. https://www.kaggle.com/datasets. Acessado: 05-11-2022.
2. BURNS, Ed et al. What is artificial intelligence (AI)? Massachusetts: TechTarget Business Technology, 2022.
3. COSTA, Lucas Z. et al. Reconhecimento de Pessoas com e sem Máscara Usando Redes Neurais Convolucionais. Lages Instituto Federal de Santa Catarina, 2022.

4. DAMIÃO, Pedro Henrique Ferreira da Costa et al. RECOGSYS - Sistema de Controle de Acesso Utilizando Reconhecimento Facial. Juiz de Fora: IF Sudeste MG, 2021.
5. FERREIRA, Michael Henrique Souza. Reconhecimento Facial Para Detecção De Emoções Utilizando Redes Neurais Convolucionais Com Tensorflow. Goiânia: Universidade Católica de Goiás, 2021.
6. IFBA. RECONHECIMENTO FACIAL. Santo Antônio de Jesus: https://github.com/ifba-saj-ads-tcc/Reconhecimento-Facial-
7. RAWAT, Kunal. 4 Actors Classification Dataset (for beginners). https://www.kaggle.com/datasets. Acessado: 05-11-2022.
8. RAWAT, Kunal. Famous Iconic Women. https://www.kaggle.com/datasets. Acessado: 05-11-2022.
9. RUSK, Nicole. METHODS TO WATCH | SPECIAL FEATURE. vol. 13, no.1. Nature America, Inc., 2016.
10. SILVA FILHO, Darlan de Castro. Reconhecimento de Caracteres Utilizando Redes Neurais Convolucionais para Auxiliar nas Correções do Sistema Multiprova. Natal: Epidemiol. Universidade Federal do Rio Grande do Norte, 2022.