

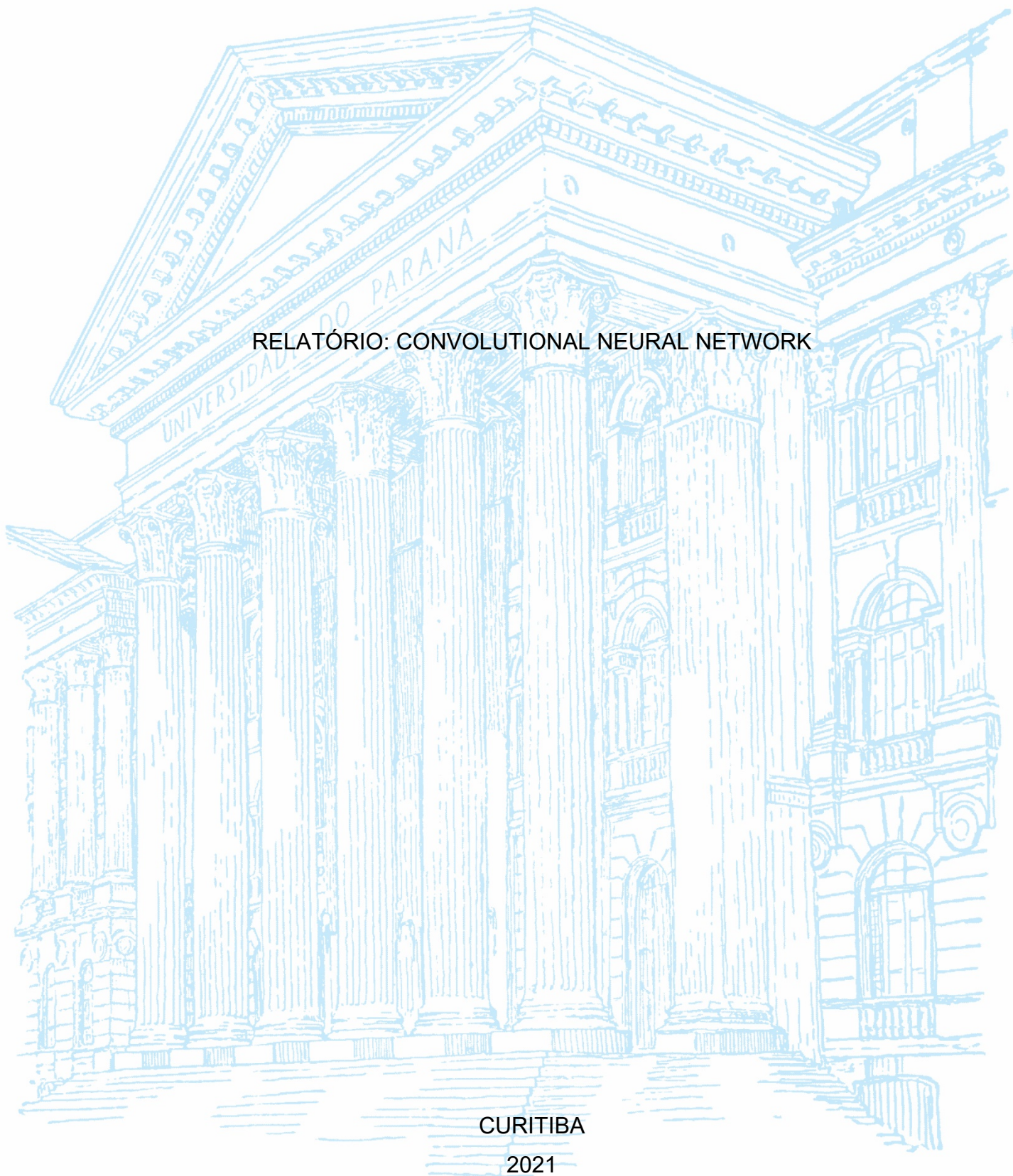
UNIVERSIDADE FEDERAL DO PARANÁ

MATEUS FELIPE DE CÁSSIO FERREIRA

RELATÓRIO: CONVOLUTIONAL NEURAL NETWORK

CURITIBA

2021



MATEUS FELIPE DE CÁSSIO FERREIRA

RELATÓRIO: CONVOLUTIONAL NEURAL NETWORK

Relatório apresentado como requisito parcial à conclusão da disciplina CI171 – Aprendizagem de Máquina, no Curso de Bacharelado em Informática Biomédica, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Orientador: Prof. Dr. Luiz Eduardo Soares de Oliveira

CURITIBA

2021

SUMÁRIO

1 INTRODUÇÃO	3
2 METODOLOGIA	4
2.1 LENET 5.....	4
2.2 CNN IMPLEMENTADA	5
2.3 <i>DATA AUGMENTATION</i>	7
3 RESULTADOS.....	8
3.1 LENET 5.....	8
3.2 MATT CNN.....	10
3.3 EXTRAÇÃO DE CARACTERÍSTICAS	12
3.4 DATA AUGMENTATION	13
4 CONCLUSÃO	16

1 INTRODUÇÃO

Este relatório busca apresentar os resultados obtidos referentes ao terceiro Laboratório da disciplina de CI171 – Aprendizagem de Máquina, que consiste em implementar duas *Convolutional Neural Network* (CNN), funções de *Data Augmentation* e desenvolver o conceito de *Transfer Learning* / *Fine-Tuning*.

O objetivo deste relatório é o de analisar o desempenho de duas Redes Neurais Convolucionais a partir de diferentes parâmetros para a compilação e treinamento do modelo. Ainda, pretende-se realizar os mesmos testes para uma base de treinamento com e sem as funções de *Data Augmentation*.

Além disso, esse laboratório visa repetir os experimentos com duas redes pré-treinadas da ImageNet, que serão utilizadas como extratores de características para realizar a classificação da base de teste em um outro classificador.

2 METODOLOGIA

Este relatório utilizou, como base de dados, imagens de meses do ano manuscritos em português. Uma vez que estamos trabalhando com imagens, uma rede neural do tipo CNN é uma metodologia usualmente utilizada para a classificação desse tipo de problema. Assim, foi utilizado 1.578 exemplos para treinamento do modelo e 401 exemplos para o teste.

No entanto, sabe-se que problemas que envolvam imagens necessitam de uma vasta quantidade de imagens na base de treinamento, visto que a CNN possui pesos que devem ser ajustados durante a fase de treinamento do modelo. Nesse sentido, uma das técnicas utilizadas para contornar esse problema é o chamado *Data Augmentation*, uma técnica de regularização que consiste em aumentar os exemplos de uma base de treinamento ao adicionar ruídos, rotações, translações e mudanças de escala, por exemplo. Essa técnica permite, assim, a redução de erros de generalização.

2.1 LENET 5

A LeNet foi uma CNN proposta no ano de 1998 por Yann LeCun, Leon Bottou, Yoshua Bengio, e Patrick Haffner. A estrutura dessa CNN conta com três camadas convolucionais, duas camadas de agregação e duas camadas totalmente conectadas, totalizando sete camadas ao total. A Figura 1 apresenta a forma como a LeNet 5 foi implementada para o problema proposto. O parâmetro *input_shape* contém uma lista com os valores (32, 32, 1). Além disso, na última camada, o parâmetro do número de classes (*num_classes*) é igual a doze (doze meses do ano).

A Figura 2, por outro lado, apresenta a arquitetura da LeNet 5 que foi implementada utilizando a biblioteca “Keras” para redes neurais.

FIGURA 1 – IMPLEMENTAÇÃO DA LENET 5

```

def LeNet_5():
    ## Create CNN model
    model = Sequential()
    model.add(Conv2D(6, kernel_size=5, strides=1, activation='tanh', input_shape=input_shape, padding='valid')) #C1
    model.add(AveragePooling2D(pool_size=2, strides=2, padding='valid')) #S2
    model.add(Conv2D(16, kernel_size=5, strides=1, activation='tanh', padding='valid')) #C3
    model.add(AveragePooling2D(pool_size=2, strides=2, padding='valid')) #S4
    model.add(Flatten()) #Flatten
    model.add(Dense(120, activation='tanh')) #C5
    model.add(Dense(84, activation='tanh')) #F6
    model.add(Dense(num_classes, activation='softmax')) #Output Layer

    ## Print CNN layers
    print ('Network structure -----')

    model.summary()

    print ('-----')

    model.compile(metrics=['accuracy'], loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.SGD
(learning_rate=learning_rate))

    return model

```

FONTE: O autor (2021).

FIGURA 2 – ARQUITETURA DA LENET 5

Layer (type)	Output Shape	Param #
conv2d_35 (Conv2D)	(None, 28, 28, 6)	156
average_pooling2d_31 (Averag	(None, 14, 14, 6)	0
conv2d_36 (Conv2D)	(None, 10, 10, 16)	2416
average_pooling2d_32 (Averag	(None, 5, 5, 16)	0
flatten_17 (Flatten)	(None, 400)	0
dense_51 (Dense)	(None, 120)	48120
dense_52 (Dense)	(None, 84)	10164
dense_53 (Dense)	(None, 12)	1020
Total params: 61,876		
Trainable params: 61,876		
Non-trainable params: 0		

FONTE: O autor (2021).

2.2 CNN IMPLEMENTADA

A estrutura da CNN implementada, a MATT CNN, conta com três camadas convolucionais, duas camadas de agregação, duas camadas totalmente conectadas, e duas camadas de normalização, totalizando nove camadas ao total. A Figura 3

apresenta a forma como a MATT CNN foi implementada para o problema proposto. O parâmetro *input_shape* contém uma lista com os valores (32, 32, 1).

A Figura 4, por outro lado, apresenta a arquitetura CNN implementada.

FIGURA 3 – IMPLEMENTAÇÃO DA MATT CNN

```
def MATT_CNN():
    ## Create CNN model
    model = Sequential()
    model.add(Conv2D(16, kernel_size=5, strides=1, activation='sigmoid', input_shape=input_shape, padding='valid'))
    model.add(LayerNormalization())
    model.add(MaxPooling2D(pool_size=2, strides=2, padding='valid'))
    model.add(Conv2D(32, kernel_size=3, strides=1, activation='sigmoid', padding='valid'))
    model.add(LayerNormalization())
    model.add(AveragePooling2D(pool_size=4, strides=4, padding='valid'))
    model.add(Flatten()) #Flatten
    model.add(Dense(120, activation='tanh'))
    model.add(Dense(24, activation='tanh'))
    model.add(Dense(num_classes, activation='softmax')) #Output Layer

    ## Print CNN layers
    print ('Network structure -----')

    model.summary()

    print ('-----')

    model.compile(metrics=['accuracy'], loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.SGD
    (learning_rate=learning_rate))

    return model
```

FONTE: O autor (2021).

FIGURA 4 – ARQUITETURA DA MATT CNN

Layer (type)	Output Shape	Param #
conv2d_148 (Conv2D)	(None, 28, 28, 16)	416
layer_normalization_59 (Layer Normalization)	(None, 28, 28, 16)	32
max_pooling2d_46 (MaxPooling2D)	(None, 14, 14, 16)	0
conv2d_149 (Conv2D)	(None, 12, 12, 32)	4640
layer_normalization_60 (Layer Normalization)	(None, 12, 12, 32)	64
average_pooling2d_96 (AveragePooling2D)	(None, 3, 3, 32)	0
flatten_70 (Flatten)	(None, 288)	0
dense_188 (Dense)	(None, 120)	34680
dense_189 (Dense)	(None, 24)	2904
dense_190 (Dense)	(None, 12)	300

FONTE: O autor (2021).

2.3 DATA AUGMENTATION

A Figura 5 apresenta uma parte da função que foi construída para realizar o processo de *Data Augmentation*. Nessa função foi feito um “flip” vertical e horizontal e três rotações (nos ângulos de 5, 10 e 15) para cada imagem do conjunto de treinamento.

FIGURA 5 – IMPLEMENTAÇÃO DA FUNÇÃO DE DATA AUGMENTATION

```
save_file = open(save_train_file, 'w')

train_paths.remove('') # Remove empty lines
train_paths.sort()

for image_path in train_paths:
    path, label = image_path.split(' ')
    path = drive_path + 'data/' + path

    trash, name, extension = path.split('.')
    image_name = name.split('/')[2]

    #abertura da imagem
    image = Image.open(path)

    # TRANSLAÇÃO
    horizontal_image = image.transpose(Image.FLIP_LEFT_RIGHT)
    vertical_image = image.transpose(Image.FLIP_TOP_BOTTOM)

    # ROTAÇÃO
    rot_1 = image.rotate(5)
    rot_2 = image.rotate(10)
    rot_3 = image.rotate(15)

    image.save(f'./data_aug/{image_name}.jpg', format="JPEG")

    horizontal_image.save(f'./data_aug/{image_name}_horizontal.jpg', format="JPEG")
    vertical_image.save(f'./data_aug/{image_name}_vertical.jpg', format="JPEG")

    rot_1.save(f'./data_aug/{image_name}_rot_1.jpg', format="JPEG")
    rot_2.save(f'./data_aug/{image_name}_rot_2.jpg', format="JPEG")
    rot_3.save(f'./data_aug/{image_name}_rot_3.jpg', format="JPEG")
```

FONTE: O autor (2021).

3 RESULTADOS

3.1 LENET 5

A Tabela 1 apresenta os resultados obtidos para diferentes situações em que foram alterados os parâmetros *batch_size*, *n_epochs* e *learning_rate*. Nota-se que o melhor resultado, considerando apenas a medida da acurácia, foi a SIT09 (ou SIT10). No entanto, a Figura 6 apresenta as medidas de acurácia e os erros cometidos pelo modelo durante o treino e validação. Nota-se que o valor do erro cometido pelo modelo durante a validação é muito maior do que o erro do modelo durante a fase de treinamento. Essa situação indica que o modelo entrou em *overfitting* e, apesar de ter o melhor desempenho no valor da acurácia, na prática, o modelo não consegue generalizar tão bem as entradas quando é preciso fazer a validação.

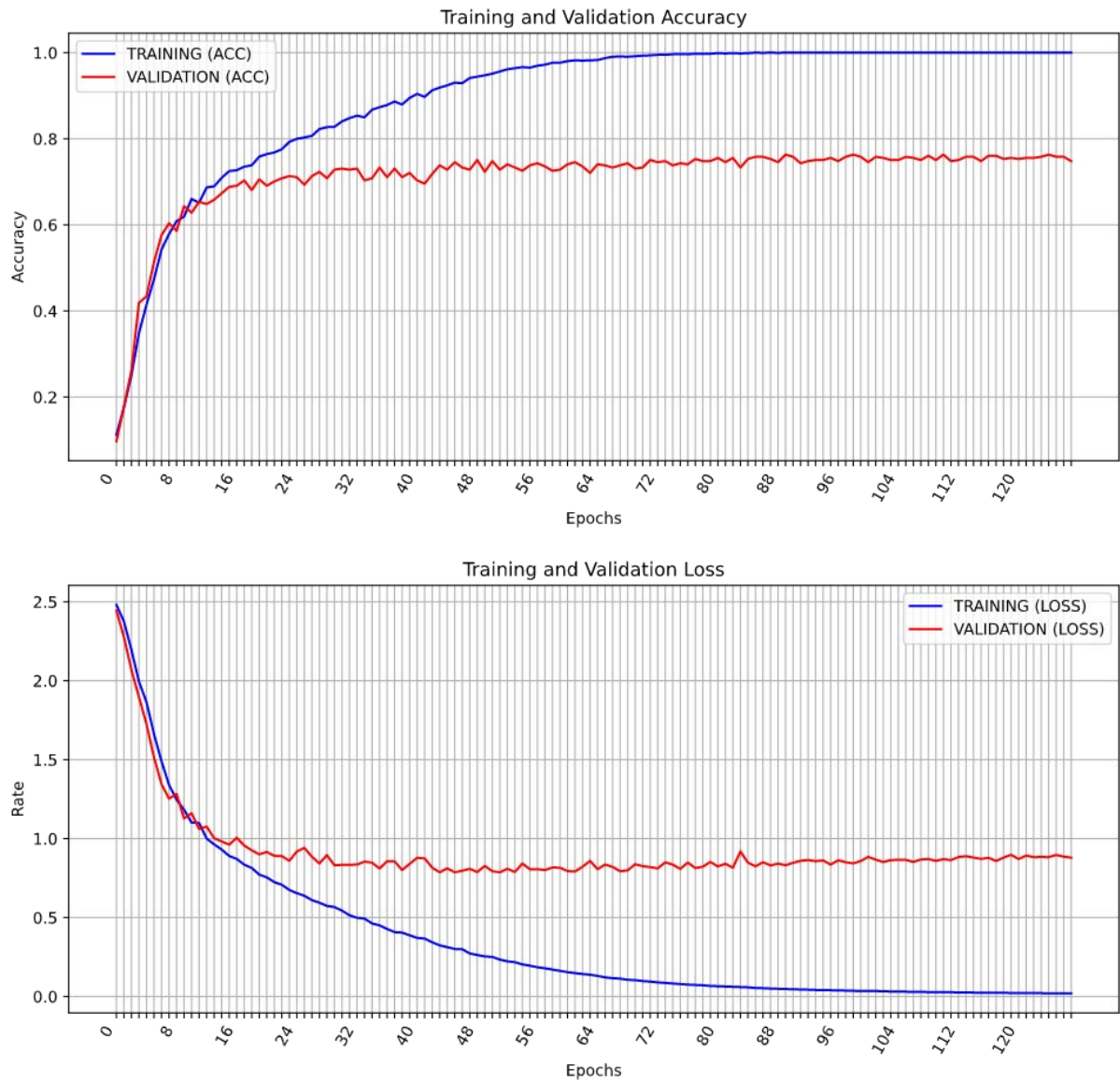
Nesse sentido, embora a situação SIT01 não tenha o melhor valor de acurácia, esse é o modelo em que o classificador consegue generalizar melhor a situação de validação do modelo. A Figura 7 apresenta os resultados obtidos da SIT01. Nota-se que os valores de erro da fase de treinamento e validação estão decaindo e convergindo quase que juntas. Essa é uma situação, aliado ao fato de que os valores de acurácia para os dois casos também estão convergindo para valores muito próximos, demonstra que o modelo consegue generalizar bem a base de validação e é um bom candidato para a classificação na prática.

TABELA 1 – RESULTADOS OBTIDOS PARA DIFERENTES SITUAÇÕES

SITUAÇÕES	BATCH SIZE	N EPOCHS	LEARNING RATE	TEST LOSS	TEST ACCURACY
SIT01	64	64	0,01	0,968	0,693
SIT02	128	64	0,01	1,275	0,613
SIT03	128	128	0,01	1,011	0,673
SIT04	128	128	0,001	2,326	0,251
SIT05	64	64	0,001	2,403	0,211
SIT06	64	64	0,1	0,951	0,768
SIT07	128	64	0,1	0,818	0,741
SIT08	64	128	0,1	0,974	0,775
SIT09	128	128	0,1	0,935	0,758
SIT010	64	256	0,1	1,081	0,758

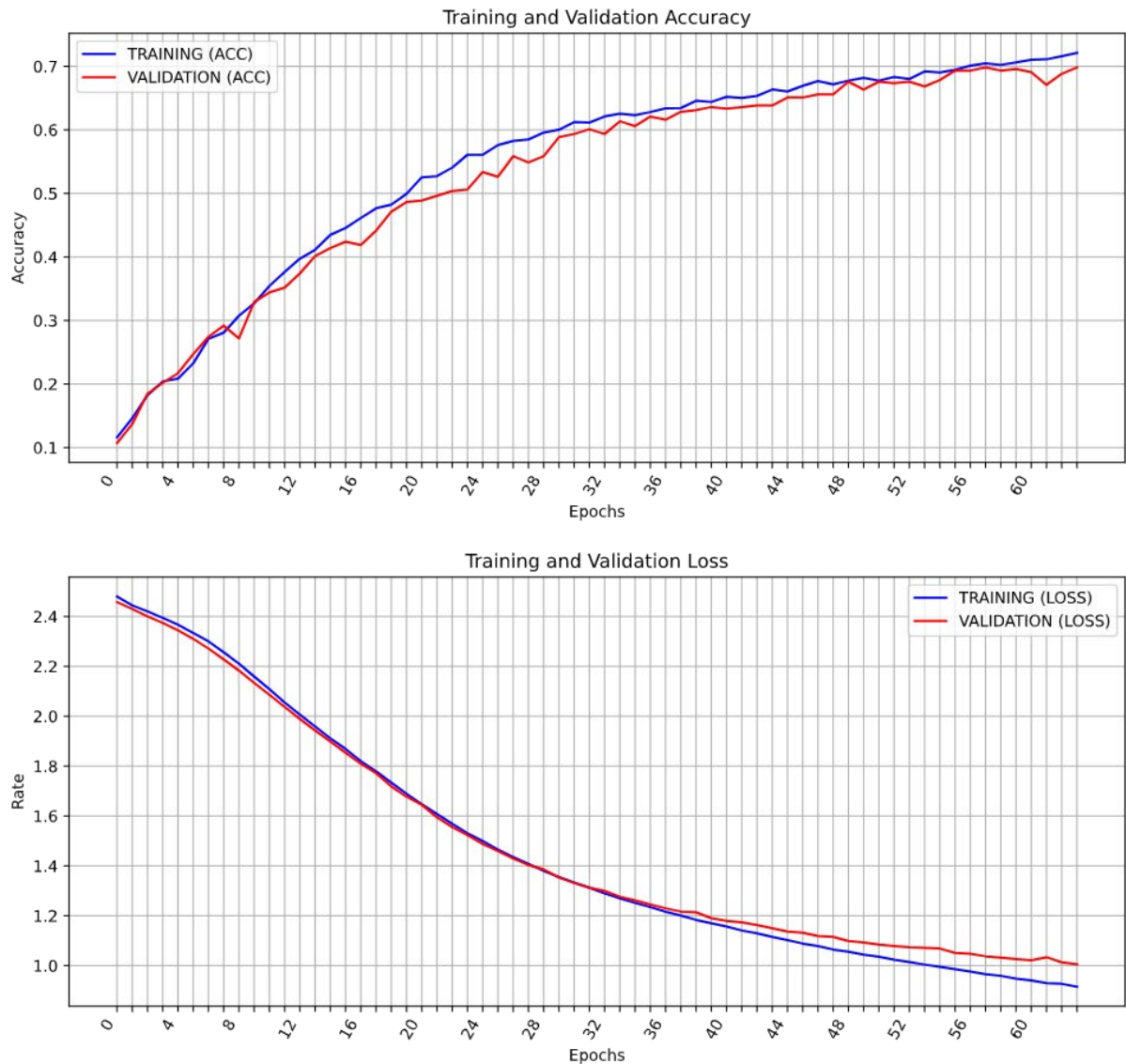
FONTE: O autor (2021).

FIGURA 6 – RESULTADOS OBTIDOS PARA A SIT09



FONTE: O autor (2021).

FIGURA 7 – RESULTADOS OBTIDOS PARA A SIT01



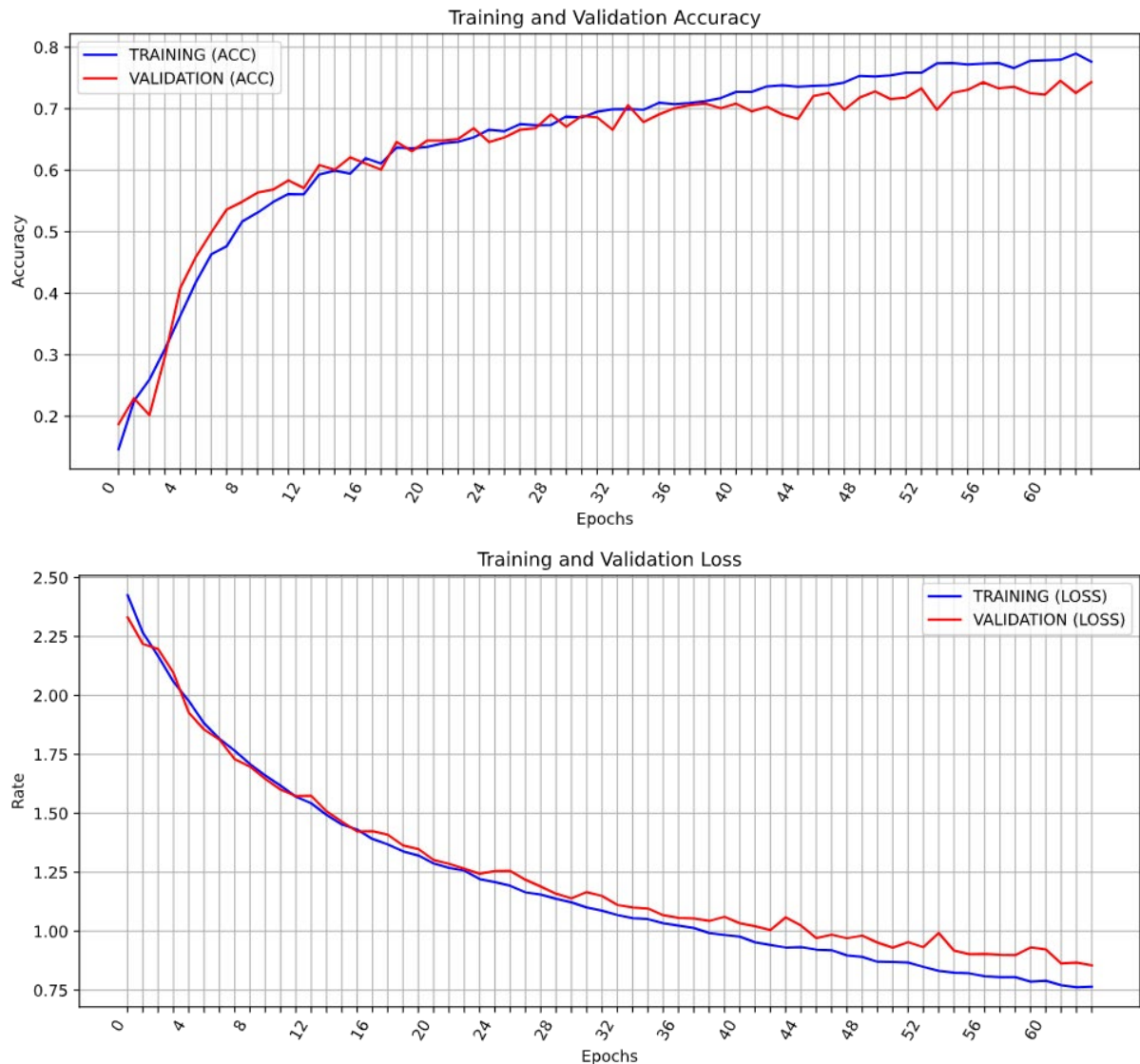
FONTE: O autor (2021).

3.2 MATT CNN

Os resultados para a MATT CNN estão apresentados na Figura 8. O valor máximo de acurácia atingido por esse modelo foi de 0,743, com um valor de “*test loss*” igual a 0,855. Esse valor foi obtido mudando os parâmetros de função de ativação, adicionando e removendo determinadas camadas, mudando o tamanho do kernel e para diferentes valores para o parâmetro *optimizer* na compilação da CNN. Apesar de ser um valor pouco superior ao LeNet 5, tentei fazer com que esse modelo apresentasse uma convergência semelhante ao modelo base. Outras medidas de acurácia acima de 0,8 foram obtidas, mas ao verificar o gráfico para

comparação entre as acurácias de treinamento e validação, bem como os erros nessas duas fases, o modelo entrou em *overfitting*.

FIGURA 8 – RESULTADOS OBTIDOS PARA A MATT CNN



FONTE: O autor (2021).

A Figura 9 apresenta a comparação entre as duas matrizes de confusão para a LeNet5 e a MATT CNN.

FIGURA 9 – RESULTADOS OBTIDOS

<pre>[[19 8 1 0 0 3 5 1 0 0 2 0] [3 16 1 1 2 2 2 2 2 0 1 0] [0 0 33 0 0 0 0 1 1 0 0 1] [1 0 1 23 10 0 0 0 2 0 1 1] [0 0 0 4 28 0 0 1 2 3 0 0] [4 0 0 0 0 20 3 0 0 0 1 1] [1 2 0 0 0 4 24 0 0 0 1 0] [0 0 1 0 2 0 0 17 0 0 1 7] [0 0 0 1 0 0 0 0 21 5 3 1] [0 0 0 1 0 0 1 0 0 26 2 0] [0 1 0 0 2 0 0 0 4 0 26 1] [0 2 0 0 0 2 0 4 2 1 7 15]]</pre>	<pre>[[24 10 0 0 0 3 0 0 0 0 0 2] [4 20 0 1 1 0 1 2 0 2 1 0] [0 0 33 0 2 0 0 1 0 0 0 0] [0 0 0 31 3 0 1 2 0 1 1 0] [0 1 1 5 28 0 0 1 0 2 0 0] [5 0 0 0 0 18 2 1 0 0 1 2] [1 0 0 0 1 3 26 0 1 0 0 0] [0 0 1 0 1 1 0 22 0 0 1 2] [0 3 1 1 0 0 0 0 21 2 1 2] [0 0 0 1 0 0 0 0 0 29 0 0] [0 1 0 0 0 0 0 0 2 0 28 3] [1 3 0 0 0 1 0 2 3 0 5 18]]</pre>
a) LeNet5	b) LeNet5

FONTE: O autor (2021).

3.3 EXTRAÇÃO DE CARACTERÍSTICAS

Uma das redes pré-treinadas escolhidas para reaplicar os testes foi a rede InceptionV3. Essa rede foi utilizada para extrair as características dos arquivos de treino e teste para utilizarmos o arquivo de saída, no formato LIBSVM, em um outro classificador. Nesse caso, o classificador escolhido foi um *Support Vector Machine*. A Figura 10 apresenta o resultado obtido de acurácia e a matriz de confusão para esse classificador.

FIGURA 10 – RESULTADOS OBTIDOS COM A INCEPTION V3 PARA EXTRAÇÃO DE CARACTERÍSTICAS E CLASSIFICAÇÃO EM UM SVM

```
[60] ▶ MI
train_archive = './libsvm/train.svm'
test_archive = './libsvm/test.svm'

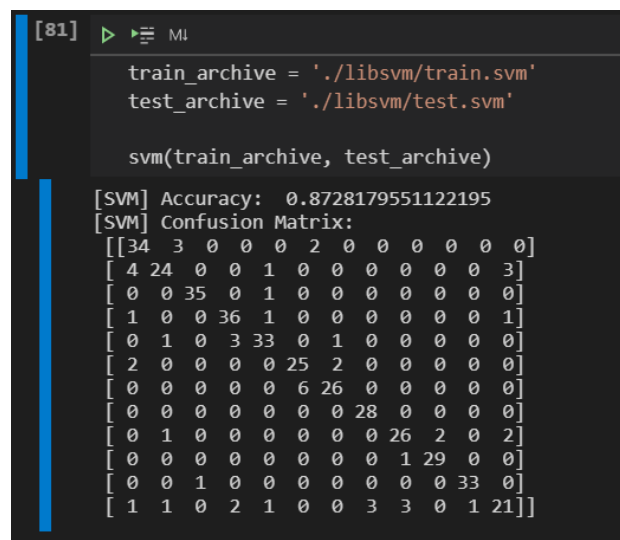
svm(train_archive, test_archive)

[SVM] Accuracy: 0.5985037406483791
[SVM] Confusion Matrix:
[[15 12 3 0 0 2 1 1 2 0 1 2]
 [ 6 21 0 1 1 0 0 0 0 0 1 2]
 [ 2 2 26 0 2 1 0 1 0 0 1 1]
 [ 2 0 0 29 3 0 2 2 0 1 0 0]
 [ 2 0 2 8 23 1 2 0 0 0 0 0]
 [ 9 0 3 0 1 10 4 2 0 0 0 0]
 [ 3 3 1 0 0 4 17 3 0 1 0 0]
 [ 2 0 2 1 1 0 1 20 0 1 0 0]
 [ 0 5 0 2 0 0 1 0 18 2 0 3]
 [ 0 1 0 1 1 1 0 0 5 20 0 1]
 [ 0 0 0 0 1 0 0 1 0 1 30 1]
 [ 1 4 0 2 2 3 1 2 3 0 4 11]]
```

FONTE: O autor (2021).

Por outro lado, foi utilizado uma segunda rede pré-treinada para reaplicar os testes, chamada de VGG16. Essa rede também foi utilizada para extrair as características dos arquivos de treino e teste para utilizarmos o arquivo de saída, no formato LIBSVM, no classificador SVM. A Figura 11 apresenta o resultado obtido de acurácia e a matriz de confusão para esse classificador. Nota-se que a medida de acurácia para esse classificador, utilizando a rede pré-treinada VGG16 foi muito superior quando comparado o desempenho do mesmo classificador utilizando a rede InceptionV3.

FIGURA 11 – RESULTADOS OBTIDOS COM A VGG16 PARA EXTRAÇÃO DE CARACTERÍSTICAS E CLASSIFICAÇÃO EM UM SVM



```
[81] ▶ MI

train_archive = './libsvm/train.svm'
test_archive = './libsvm/test.svm'

svm(train_archive, test_archive)

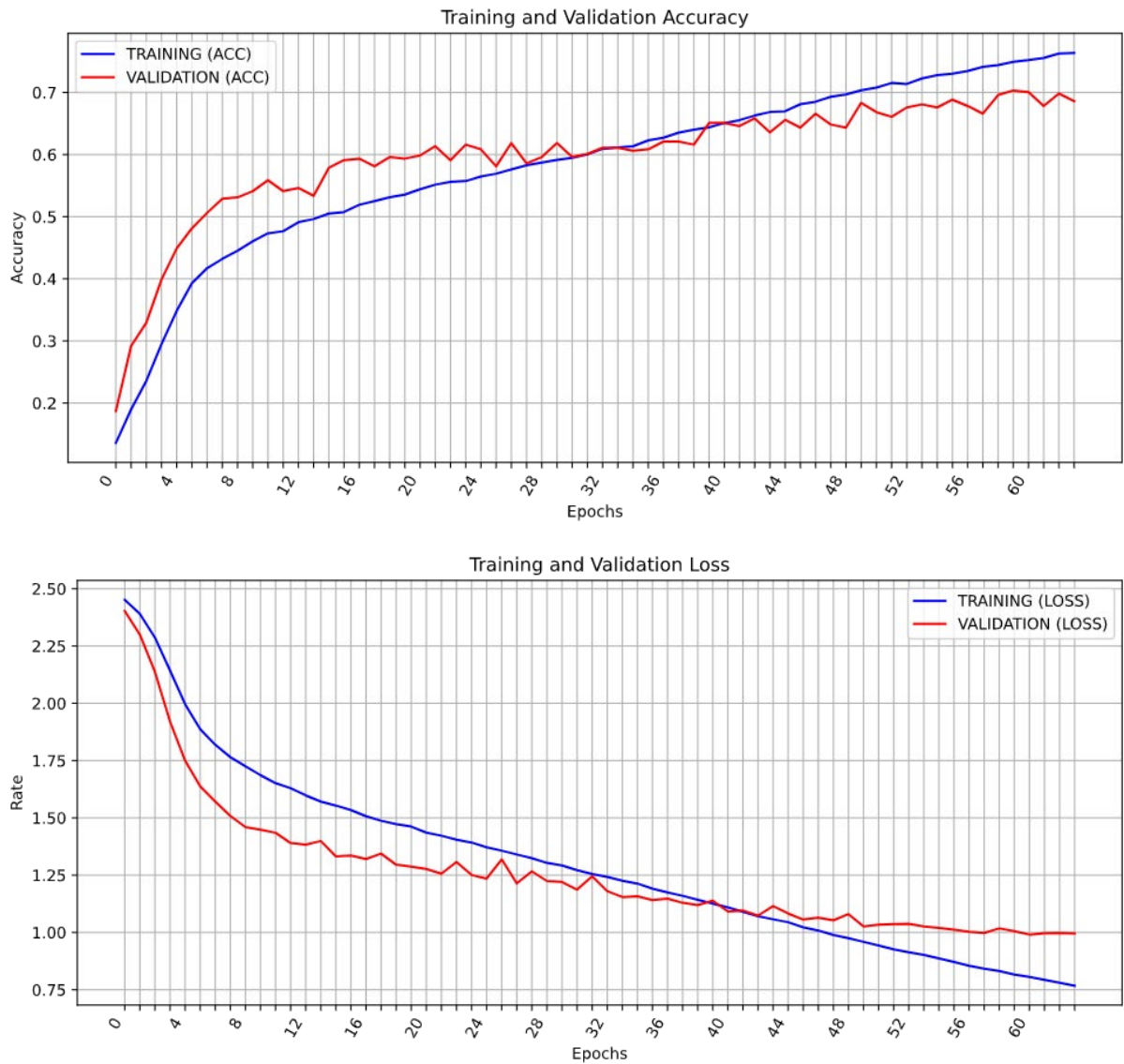
[SVM] Accuracy: 0.8728179551122195
[SVM] Confusion Matrix:
[[34  3  0  0  0  2  0  0  0  0  0  0]
 [ 4 24  0  0  1  0  0  0  0  0  0  3]
 [ 0  0 35  0  1  0  0  0  0  0  0  0]
 [ 1  0  0 36  1  0  0  0  0  0  0  1]
 [ 0  1  0  3 33  0  1  0  0  0  0  0]
 [ 2  0  0  0  0 25  2  0  0  0  0  0]
 [ 0  0  0  0  0  6 26  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 28  0  0  0  0]
 [ 0  1  0  0  0  0  0  0 26  2  0  2]
 [ 0  0  0  0  0  0  0  0  1 29  0  0]
 [ 0  0  1  0  0  0  0  0  0  0 33  0]
 [ 1  1  0  2  1  0  0  3  3  0  1 21]]
```

FONTE: O autor (2021).

3.4 DATA AUGMENTATION

A Figura 12 apresenta os resultados obtidos utilizando a função de Data Augmentation para o LeNet 5.

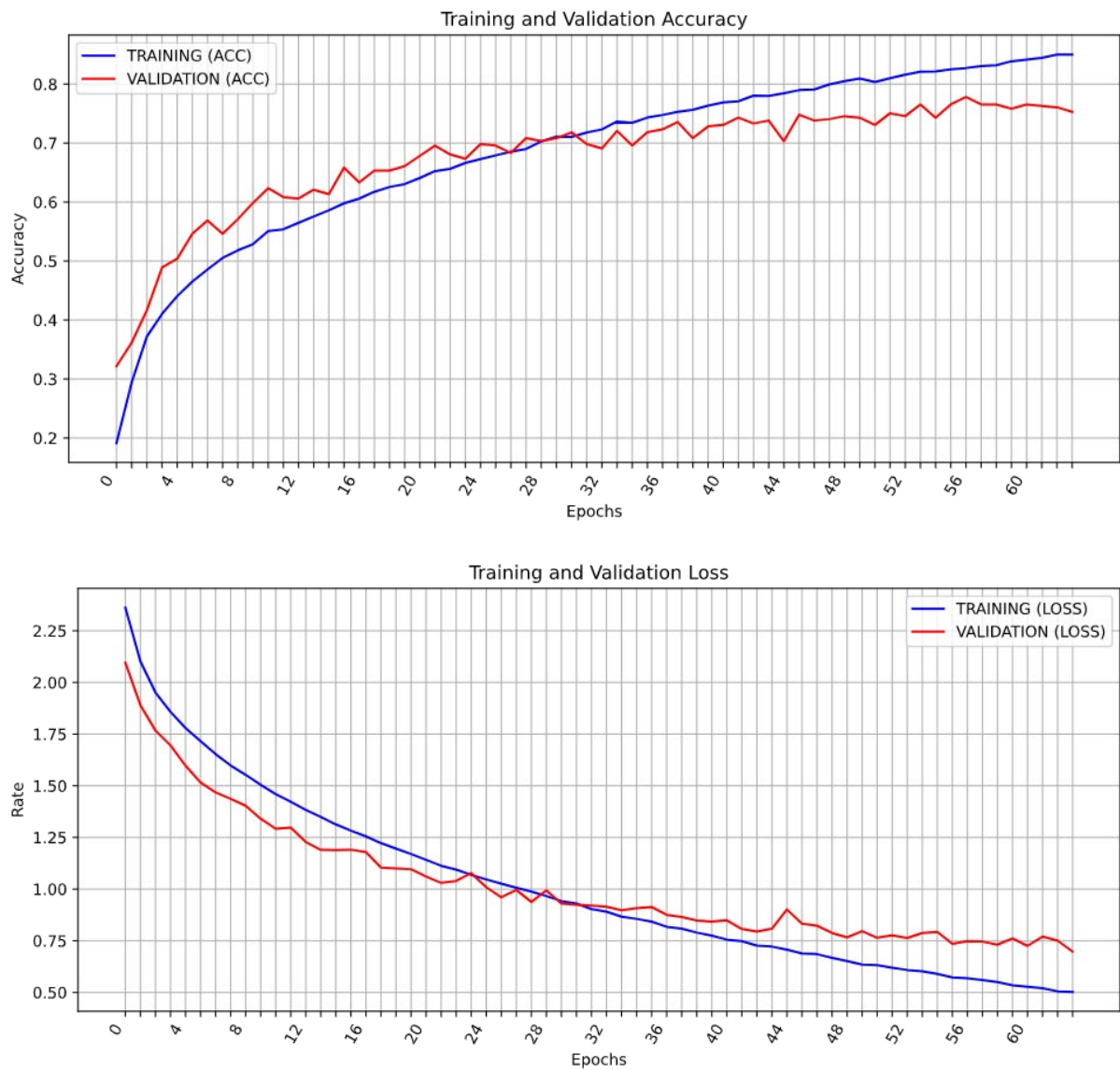
FIGURA 12 – RESULTADOS OBTIDOS COM DATA AUGEMENTATION PARA A LENET5



FONTE: O autor (2021).

A Figura 13 apresenta os resultados obtidos utilizando a função de Data Augmentation para o MATT CNN.

FIGURA 13 – RESULTADOS OBTIDOS COM DATA AUGEMENTATION PARA A LENET



FONTE: O autor (2021).

4 CONCLUSÃO

A partir dos resultados do experimento nota-se que existe um significativo impacto na escolha dos parâmetros e configurações utilizadas (camadas) em uma CNN. De maneira geral, as CNN apresentaram bons resultados quando comparado com alguns outros classificadores. Foi possível notar também que o uso de uma CNN para extração de vetores de características pode ser uma solução muito benéfica para alguns problemas, visto que nesse laboratório foi possível obter uma acurácia acima de 0,8 para o problema utilizando o classificador SVM a partir da extração de um vetor de características pelo VGG61.

De maneira geral, a CNN obtém ótimos resultados e que se tornam ainda mais positivos ao utilizarmos as funções de Data Augmentation para a resolução do problema.