CNN - Redes Neurais Convolucionais para classificação de manuscritos

Autor: André Heidemann Iarozinski

1. Criação das bases e Data Augmentation

Neste experimento, foi utilizada uma base de dados com imagens dos meses do ano, totalizando 12 classes. Esta base contém no total **1979** amostras, sendo que **1578** foram separadas para treinamento e **401** separadas para validação. Todas as imagens são binárias e estão em formato *.jpeg*. A primeira etapa do experimento foi realizada no ambiente *Jupyter Notebook*, pois a leitura e carregamento de arquivos é realizada de forma mais rápida em relação ao *Google Colab*. Nesta etapa foram criadas 6 bases distintas com as seguintes características:

Nome	Resolução dos amostras	Amostras de treinamento	Amostras de teste
base_32	32x32px	1578	401
base_32_a	32x32px	4734	401
base_64	64x64px	1578	401
base_64_a	64x64px	4734	401
base_128	128x128px	1578	401
base_128_a	128x128px	4734	401

Tabela 1. Bases de dados criadas para o experimento

Para a criação das bases com **4734** amostras de treinamento, foi utilizada a técnica de *Data Augmentation* e para cada uma das amostras originais, foram criadas duas amostras extras, totalizando uma base de treinamento com o triplo do tamanho original. Cada amostra foi rotacionada aleatoriamente entre -25 e + 25 graus, seguido pela adição de um ruído aleatório do tipo gaussiano. O exemplo abaixo ilustra o processo de *Data Augmentation* e as funções utilizadas:

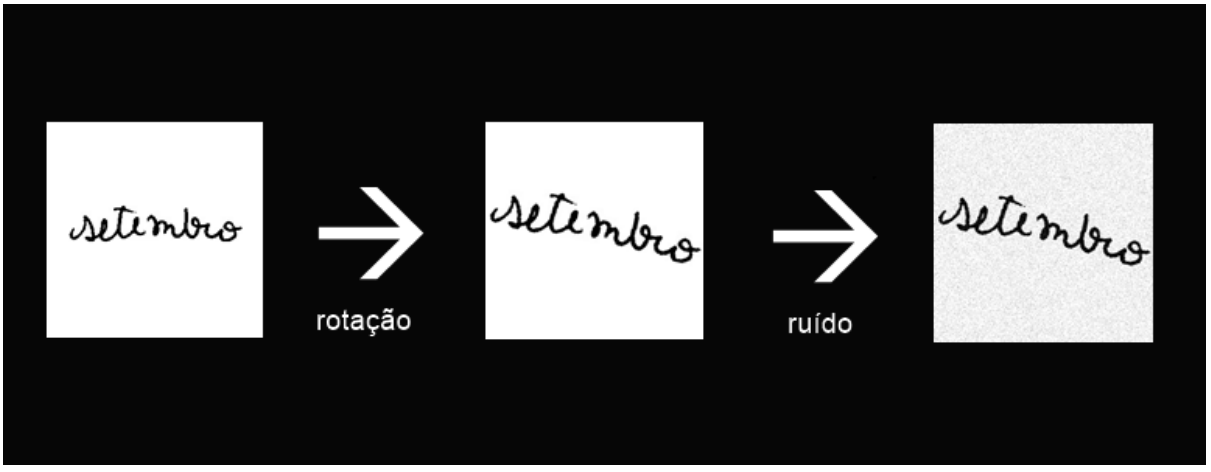


Figura 1. Exemplo do processo de *Data Augmentation*

```
def random_rotation(image_array: ndarray):
    random_degree = random.uniform(-25, 25)
    return sk.transform.rotate(image_array, random_degree,mode='edge')

def random_noise(image_array: ndarray):
    return sk.util.random_noise(image_array, mode='gaussian')
```

Figura 2. Funções implementadas para realização do *Data Augmentation*

Foram utilizados os frameworks **Keras** e **Sklearn** para realizar todo processo desta primeira etapa, incluindo a normalização das imagens, geração dos labels para a matriz de confusão e a conversão dos vetores das classes para matrizes binárias. Como resultado final desta primeira etapa, cada uma das 6 bases foi salva em uma pasta separada no *Google Drive* com os seguintes arquivos no formato .npy:

- x_train
- x_test
- y_train
- y_test
- labels

A partir deste momento, todo o restante do experimento foi realizado no *Google Colab*. Esta migração de plataforma foi escolhida logo no início dos trabalhos, pois foi verificada uma diferença significativa (10 vezes menor) nos tempos de execução e treinamento das CNN´s utilizando o *Google Colab* com processamento via GPU.

2. Definindo os modelos das CNN's

Nesta etapa, foram implementadas duas redes neurais convolucionais. A primeira já conhecida **LeNet 5** de Yann LeCun, originalmente desenvolvida para classificar dígitos manuscritos e possui uma arquitetura de 7 camadas mostrada a seguir:

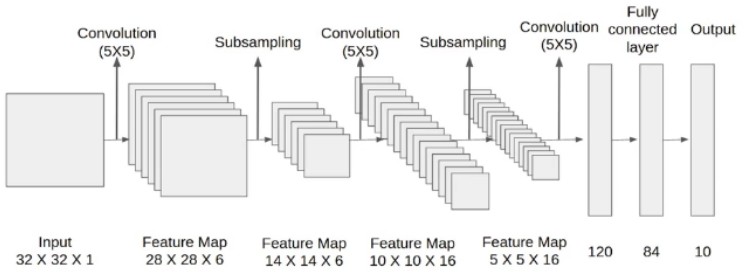


Figura 3. Estrutura da rede LeNet-5

A rede possui cinco camadas com parâmetros "aprendíveis" e, portanto, denominada **LeNet 5**. Possui três conjuntos de camadas de convolução com uma combinação de pooling. Após as camadas de convolução e pooling, temos duas camadas totalmente conectadas. Por fim, um classificador Softmax que classifica as imagens nas respectivas classes. A entrada para este modelo é uma imagem em tons de cinza de **32 x 32**, portanto, o número de canais é **1**.

Em seguida, temos uma operação de convolução com o tamanho do filtro **5 x 5** e temos **6** desses filtros. Como resultado, obtemos um *Feature Map* de tamanho **28 x 28 x 6**. Após outras camadas de convolução e agrupamento, temos uma camada totalmente conectada com **84** neurônios. Por fim, temos uma camada de saída com **10** neurônios, já que os dados do problema da época possuíam dez classes.

Para este experimento, foi alterado o tamanho da entrada, para a rede se ajustar ao tamanho das amostras das diferentes bases e foi estabelecido na saída **12** classes ao invés de **10**. Estas alterações estão destacadas em verde na imagem abaixo:

```
model = Sequential()
model.add(layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu',
input_shape=input_shape))
model.add(layers.AveragePooling2D())
model.add(layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'))
model.add(layers.AveragePooling2D())
model.add(layers.Flatten())
model.add(layers.Dense(units=120, activation='relu'))
model.add(layers.Dense(units=84, activation='relu'))
model.add(layers.Dense(num_classes, activation = 'softmax'))
```

Figura 4. Implementação da LeNet-5 no *Google Colab*

A segunda rede foi baseada na rede **LeNet5**, porém com várias alterações. Realizei alguns testes e baseado nos resultados fui realizando algumas modificações no número de camadas de convolução e nas camadas totalmente conectadas. Para realizar as comparações de desempenho, esta rede foi definida com o nome "**CNNL3**". Os detalhes da arquitetura da rede são mostrados na figura a seguir:

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))##########camada adicional
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))##########camada adicional
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))########vetor de características
model.add(Dense(72, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

Figura 5. Implementação da rede customizada CNNL3

3. Comparação do desempenho

Para todas as execuções a seguir, as 2 redes neurais (CNNL3 e LeNet5) utilizaram todas bases da tabela 1:

- base_32
- base_32_a
- base_64
- base_64_a
- base_128
- base_128_a

E foram utilizados os seguintes parâmetros e configurações para o treinamento:

- Shuffle = True
- Learning Rate = 0.007
- Epochs = 150
- Batch Size = 64

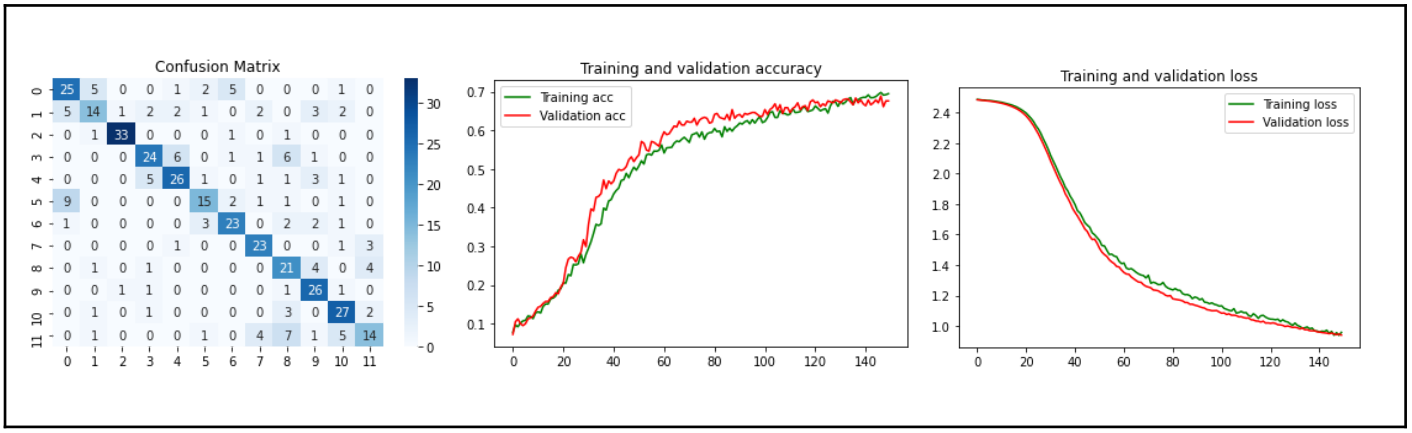
```
model.compile(metrics=['accuracy'],
loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adadelta(learning_rate=0.007))

history = model.fit(x=x_train, y=y_train, batch_size=64, shuffle=True,
epochs=150, verbose=1, validation_data=(x_test, y_test))
```

Figura 6. Configurando parâmetros para o treinamento das redes

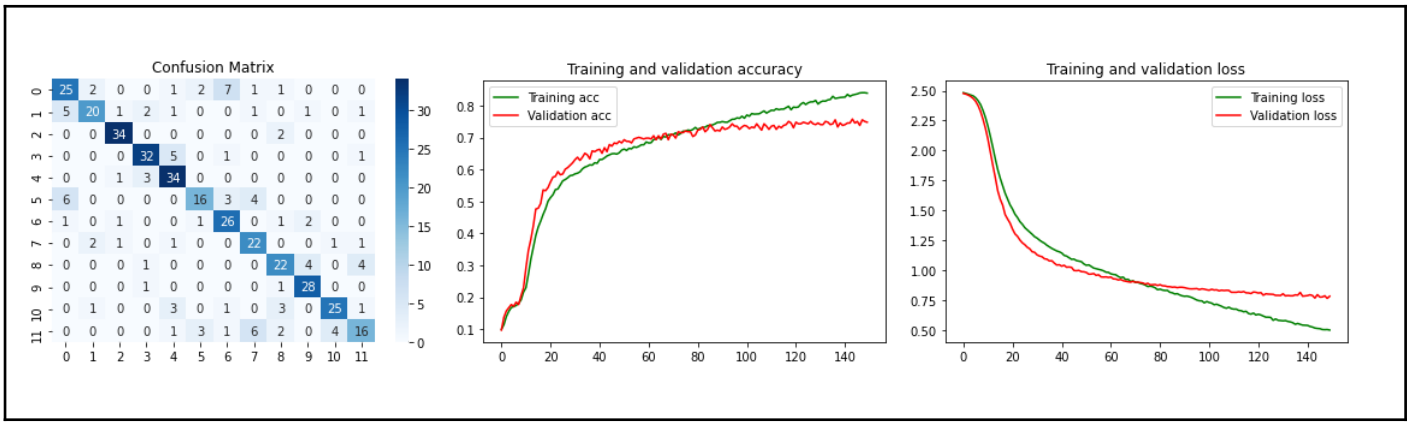
Os parâmetros para o treinamento foram definidos após a execução de vários testes prévios que não foram incluídos, com o objetivo de tornar este relatório mais breve e conciso. Nas próximas páginas serão mostradas para cada rede, o resultado do treinamento com cada uma das 6 bases e em seguida os resultados das curvas de Loss, Accuracy e a matriz de confusão.

3.1 CNNL3

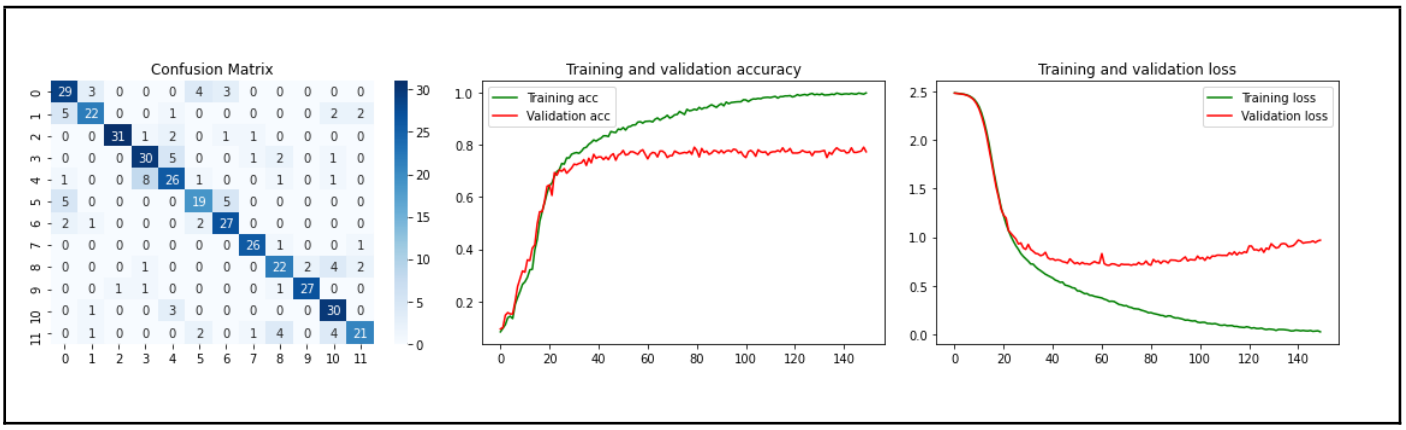
- **base_32:** **Test loss: 0.94120** **Test accuracy: 0.67581**



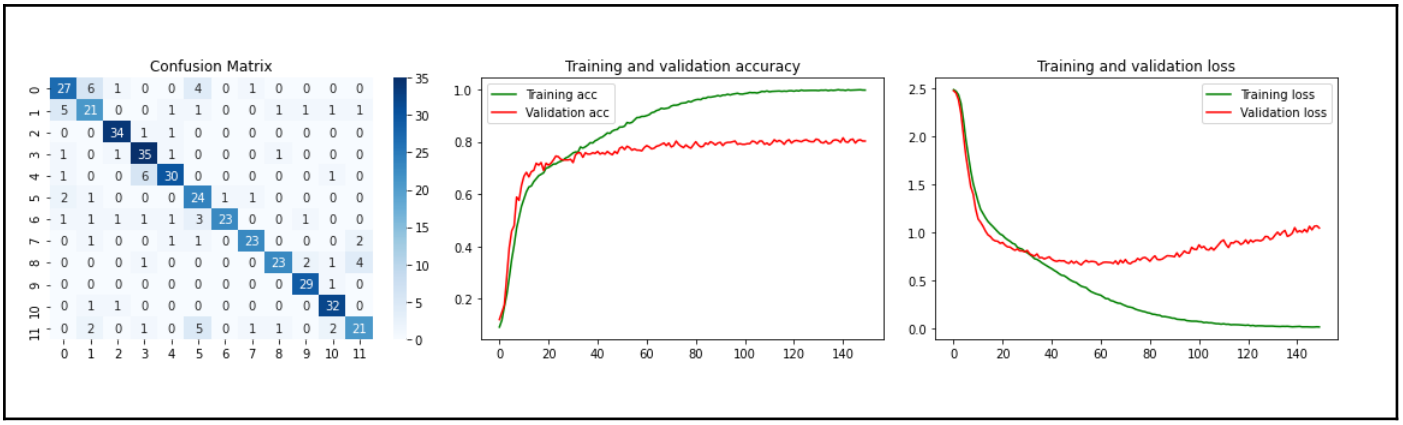
- **base_32_a:** **Test loss: 0.78049** **Test accuracy: 0.74812**



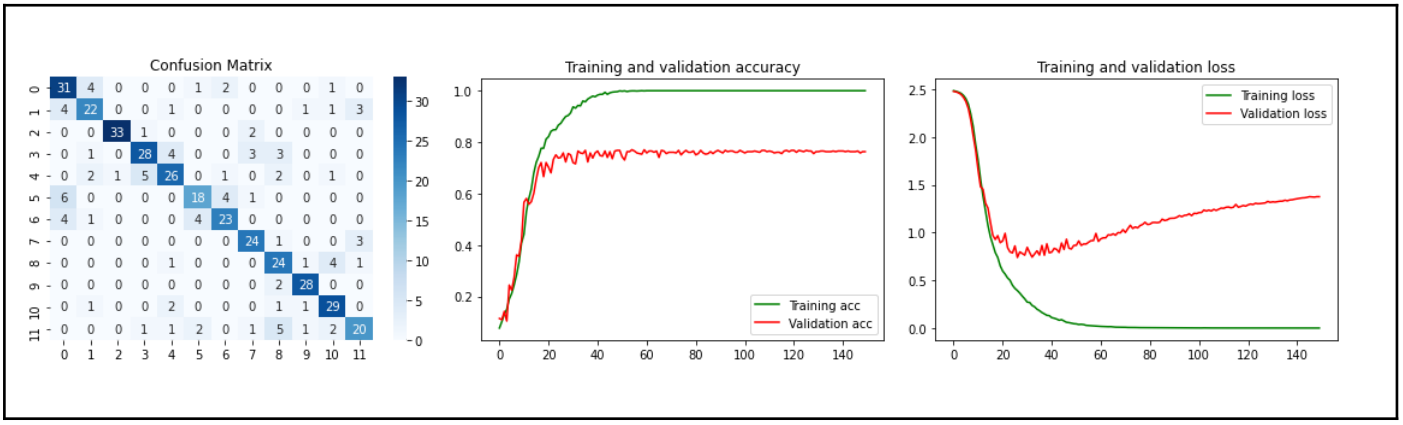
- **base_64:** **Test loss: 0.96855** **Test accuracy: 0.77306**



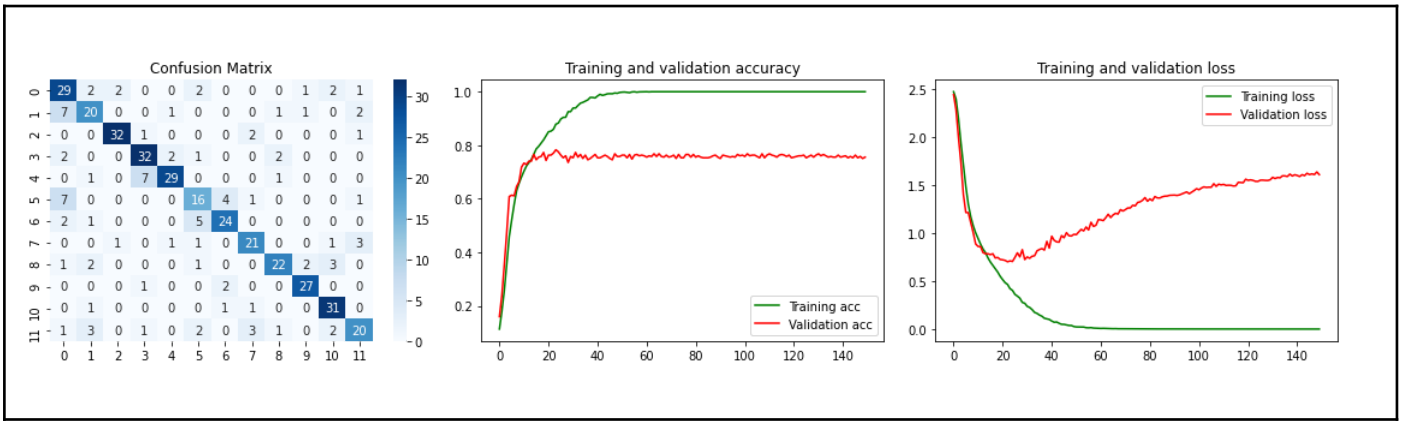
- **base_64_a:** **Test loss: 1.041446** **Test accuracy: 0.802992**



- **base_128:** **Test loss: 1.376332** **Test accuracy: 0.763092**

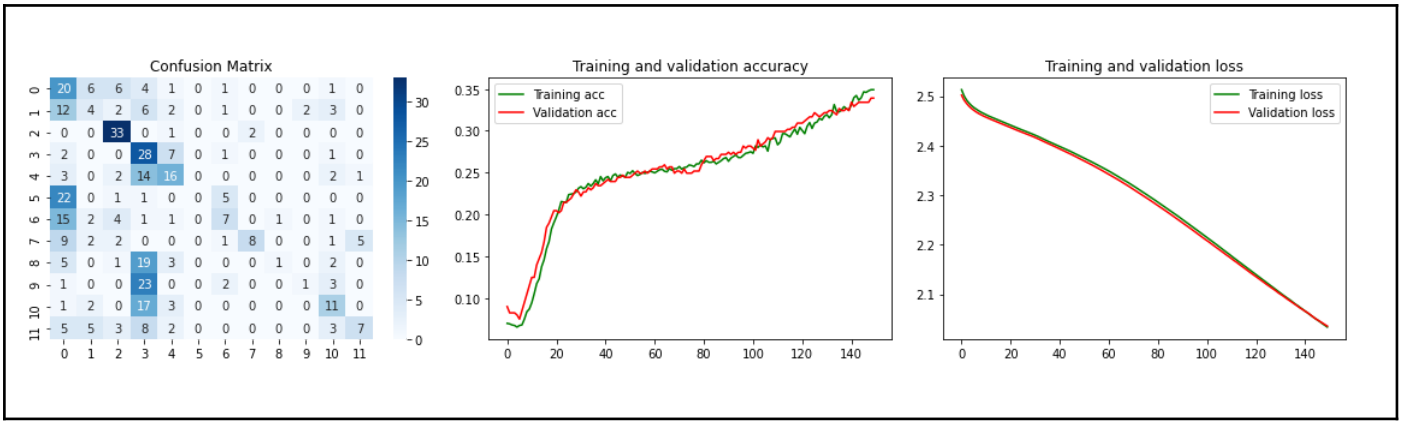


- **base_128_a:** **Test loss: 1.607017** **Test accuracy: 0.755610**

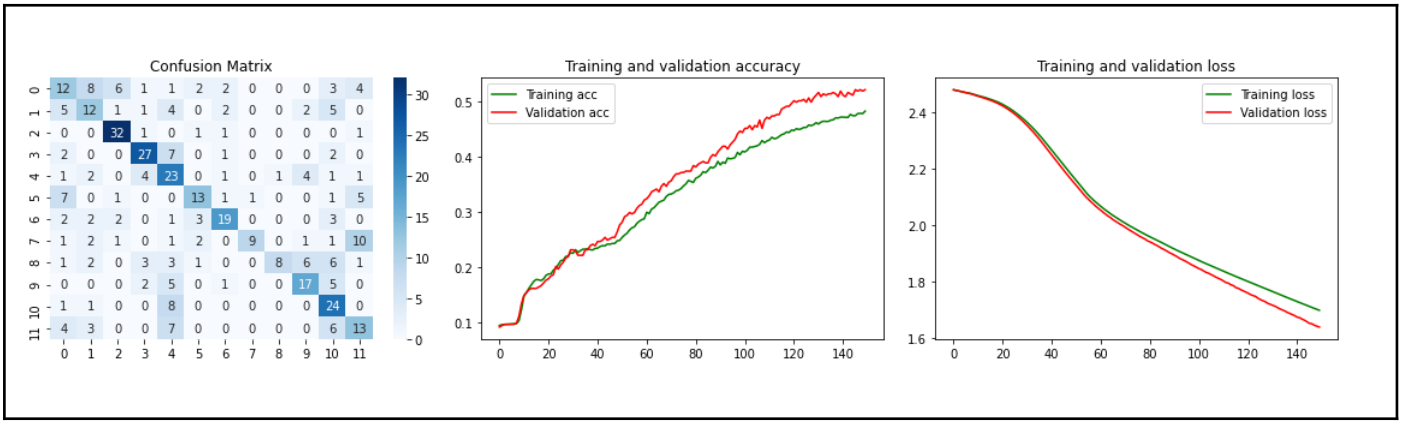


3.2 LeNet 5

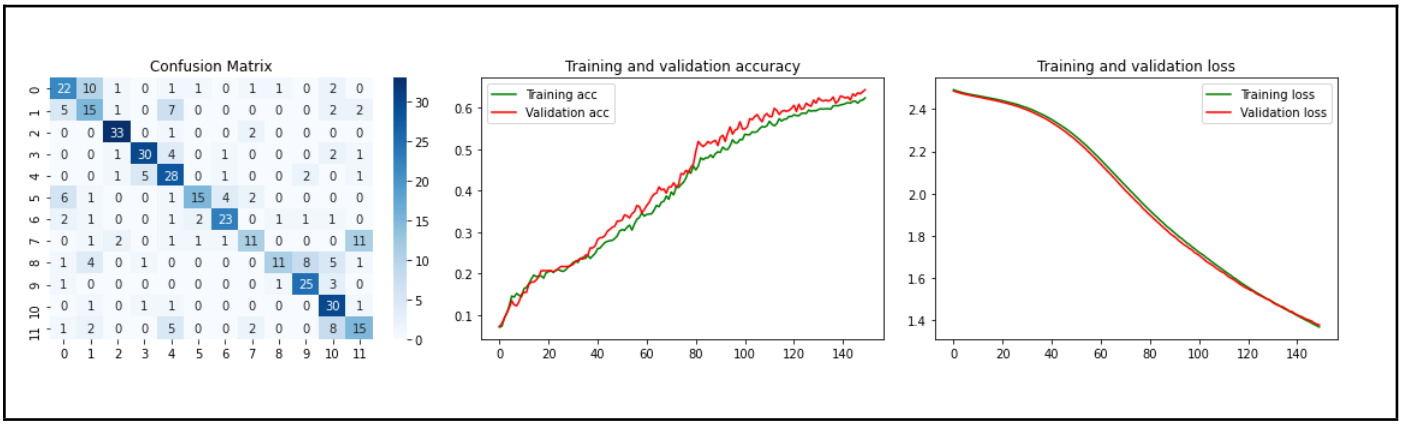
- **base_32:** **Test loss: 2.035769** **Test accuracy: 0.339152**



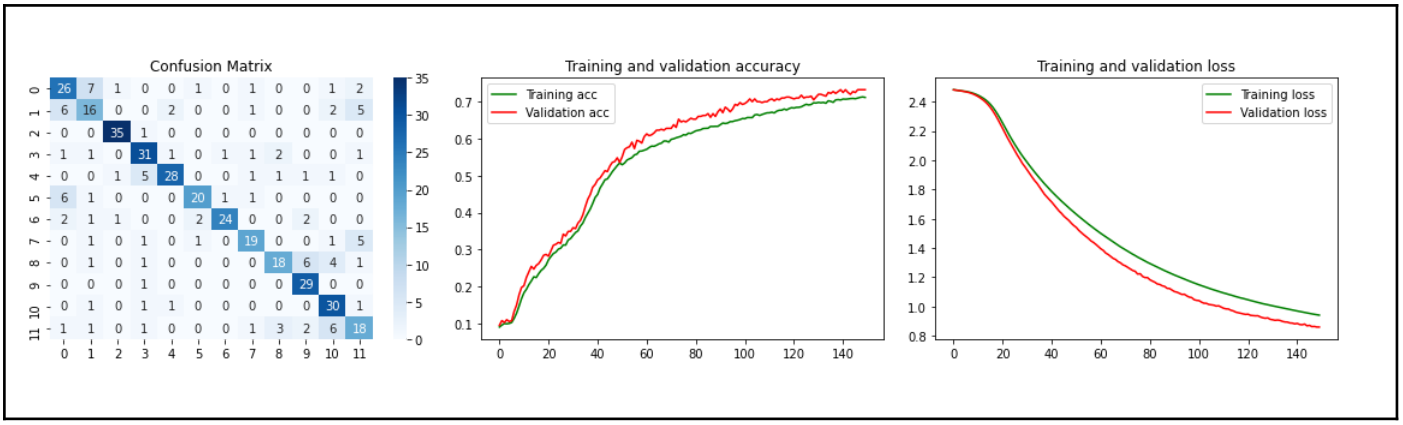
- **base_32_a:** **Test loss: 1.638130** **Test accuracy: 0.521197**



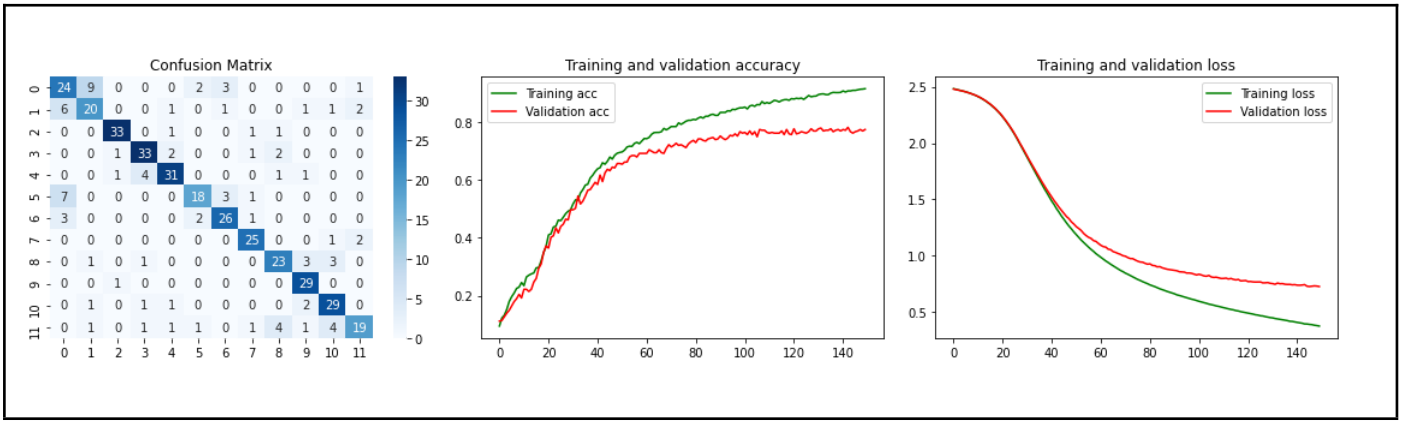
- **base_64:** **Test loss: 1.378709** **Test accuracy: 0.643391**



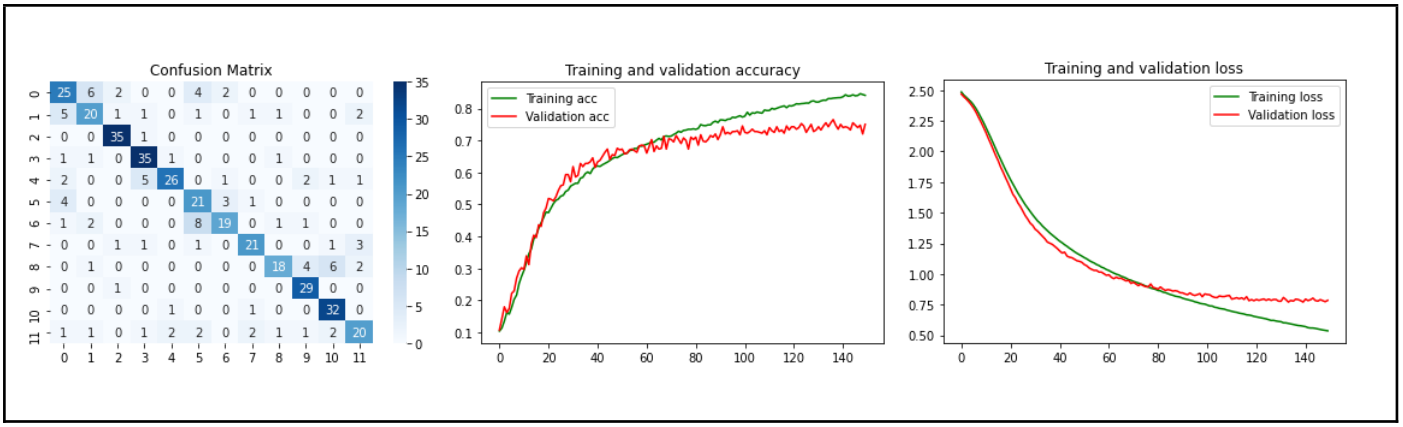
- **base_64_a:** **Test loss: 0.858596** **Test accuracy: 0.733167**



- **base_128:** **Test loss: 0.727214** **Test accuracy: 0.773067**



- **base_128_a:** **Test loss: 0.784319** **Test accuracy: 0.750623**



3.3 Conclusões dos experimentos

A expectativa inicial, era de se obter uma acurácia de pelo menos 85% que não foi alcançada. O experimento que chegou mais próximo desse valor foi o que utilizou a **CNNL3** com a **base_64_a** chegando a uma acurácia de **0.8029** porém, o *Loss* da validação ficou relativamente alto (1.04). Analisando o gráfico deste resultado nota-se claramente que após a época 50, o modelo já começou a entrar em overfitting, a acurácia de treinamento chegou próximo a 1 e a curva de Loss da validação começou a subir:

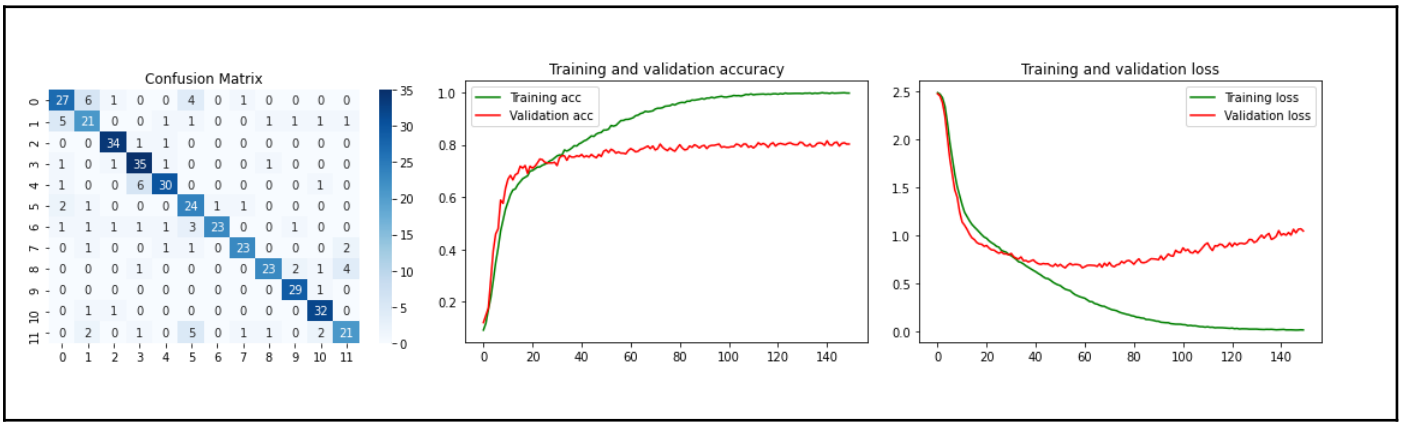


Figura 7. Resultado da rede CNNL3 com a base_64_a

Outro experimento que mostrou um resultado razoável foi a **LeNet5** utilizando a **base_128** com uma acurácia um pouco menor (**0.7730**) mas um Loss muito menor (**0.7272**). A **LeNet5,** ao utilizar bases com entradas menores, não conseguiu convergir com o número de épocas definido e as curvas não mostram quase nenhuma saturação após 150 épocas. Esta saturação só foi observada após a utilização das bases com entradas de 128 pixels:

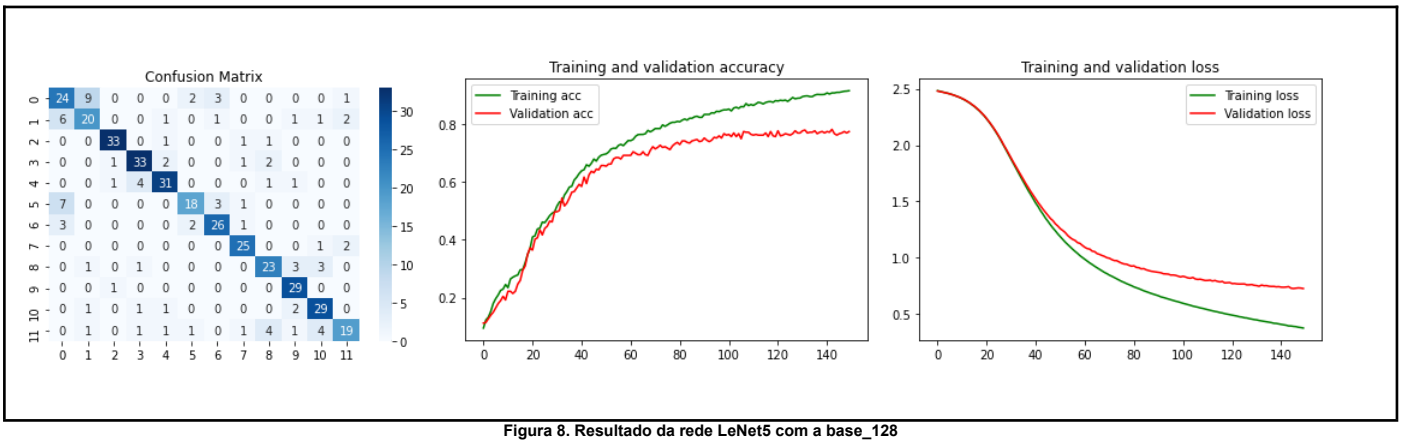


Figura 8. Resultado da rede LeNet5 com a base_128

Foi observado que em quase todos os experimentos, ao comparar a execução com e sem *Data Augmentation*, a acurácia sempre foi maior nos casos em que a base de treinamento possuía mais amostras. Isso só não foi observado no último experimento com a **LeNet5**. Analisando agora as matrizes de confusão, foi possível visualizar que existem erros comuns entre determinadas classes. A palavra “janeiro” e "fevereiro", por exemplo, ou “dezembro” e “novembro” possuem as mesmas terminações. Isso se refletiu nas matrizes de confusão e quase todos os experimentos cometeram erros entre estas classes:

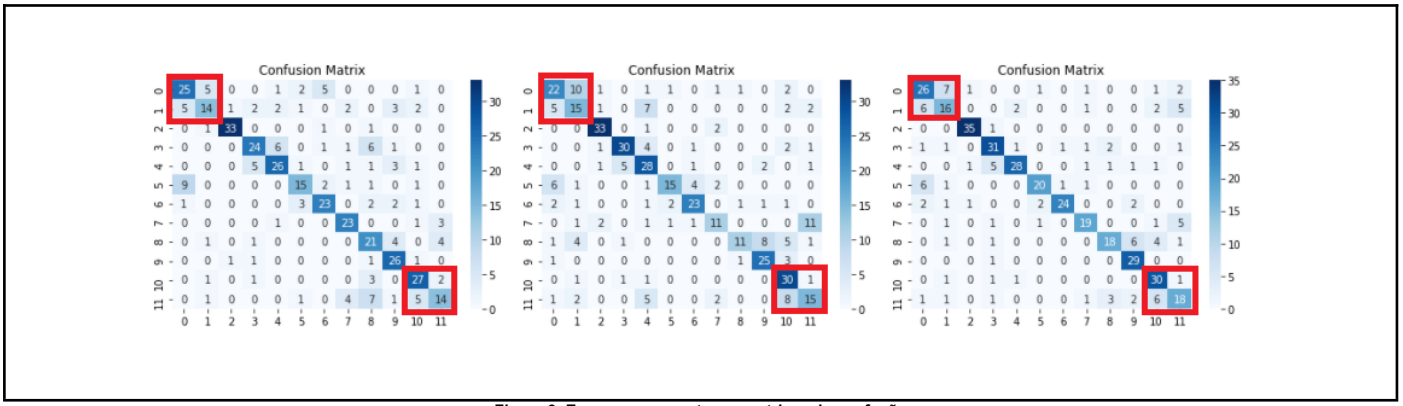
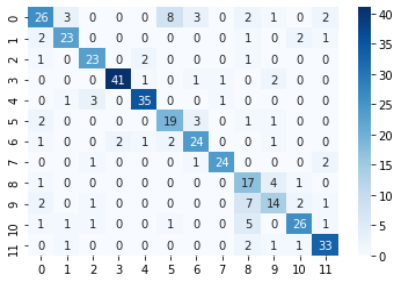


Figura 9. Erros comuns entre as matrizes de confusão

4. Utilização de redes pré-treinadas

Foram utilizadas 2 redes pré-treinadas da ImageNet para se obter um vetor de características para ser utilizado em outro classificador. As redes escolhidas foram a **InceptionV3** e a **DenseNet201**. As bases utilizadas para gerar os vetores foram a **'base_64'** e a **'base_64_a'**. Como a rede **Inceptionv3** só aceita entradas de no mínimo 75x75, foi feito o *resize* das imagens antes do processo. Já a **DenseNet201** aceita entradas com no mínimo 32x32, então foi utilizado o tamanho original das imagens das bases.

Como resultado foram obtidos 4 arquivos no formato **.svm** e em seguida foi utilizado um classificador de **regressão logística** para realizar as previsões. O classificador foi usado no modo *default* e o único parâmetro configurado foi o número máximo de iterações(max_iter=2000). Segue abaixo os resultados para cada rede escolhida:



-Rede utilizada = **DenseNet201**

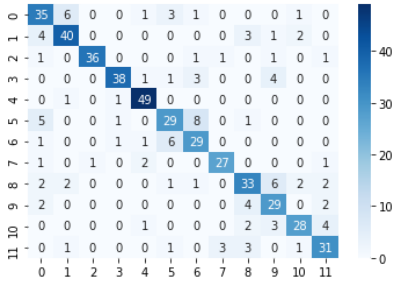
-Base utilizada = **base_64**

-Classificador **= LogisticRegression**

-N.º de parâmetros do vetor de características **= 7680**

-N.º de amostras utilizadas **= 1979 (test_size=0.2)**

-F1-Score **=** Acurácia = **0.770202**



-Rede utilizada = **DenseNet201**

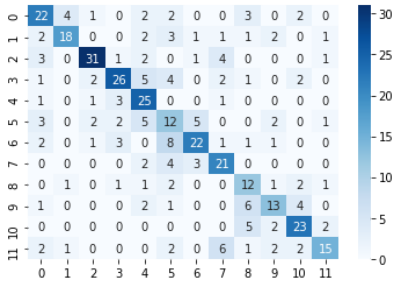
-Base utilizada = **base_64_a**

-Classificador **= LogisticRegression**

-N.º de parâmetros do vetor de características **= 7680**

-N.º de amostras utilizadas **= 5135 (test_size=0.1)**

-F1-Score **=** Acurácia = **0.785992**



-Rede utilizada = **InceptionV3**

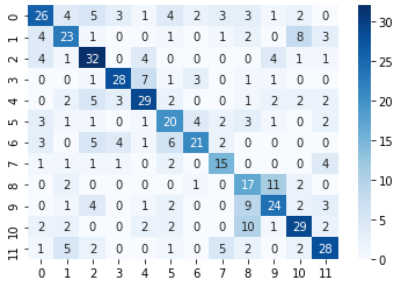
-Base utilizada = **base_64**

-Classificador **= LogisticRegression**

-N.º de parâmetros do vetor de características **= 2048**

-N.º de amostras utilizadas **= 1979 (test_size=0.2)**

-F1-Score **=** Acurácia = **0.60601**



-Rede utilizada = **InceptionV3**

-Base utilizada = **base_64_a**

-Classificador **= LogisticRegression**

-N.º de parâmetros do vetor de características **= 2048**

-N.º de amostras utilizadas **= 5135 (test_size=0.1)**

-F1-Score **=** Acurácia = **0.5680933**

5. Conclusões finais

A tabela a seguir mostra o resultado de todos os experimentos mencionados neste relatório ordenados em relação a acurácia:

Experimento	Dataset	Acurácia	Loss
CNNL3	**64_a**	**0.8029**	**1.0414**
DenseNet201	**64_a**	**0.7859**	**x**
LeNet5	**128**	**0.7730**	**0.7272**
CNNL3	**64**	**0.7730**	**0.9685**
DenseNet201	**64**	**0.7702**	**x**
CNNL3	**128**	**0.7630**	**1.3763**
CNNL3	**128_a**	**0.7556**	**1.6070**
LeNet5	**128_a**	**0.7506**	**0.7843**
CNNL3	**32_a**	**0.7481**	**0.7804**
LeNet5	**64_a**	**0.7331**	**0.8585**
CNNL3	**32**	**0.6758**	**0.9412**
LeNet5	**64**	**0.6433**	**1.3787**
InceptionV3	**64**	**0.6060**	**x**
InceptionV3	**64_a**	**0.5680**	**x**
LeNet5	**32_a**	**0.5211**	**1.6381**
LeNet5	**32**	**0.3391**	**2.0357**

Tabela 2. Resumo dos resultados obtidos

- A **DenseNet201** conseguiu um ótimo resultado (0.7859) e mostrou que a técnica de *Transfer Learning* cumpre bem seu papel extraindo de forma robusta as características dos dados de treinamento.

- Os resultados poderiam ser melhores caso fossem geradas imagens com mais variações no processo de *Data Augmentation* como translação e zoom in/out. Principalmente para as classes com maiores erros nas matrizes de confusão, como por exemplo: janeiro/fevereiro, junho/julho e dezembro/setembro.

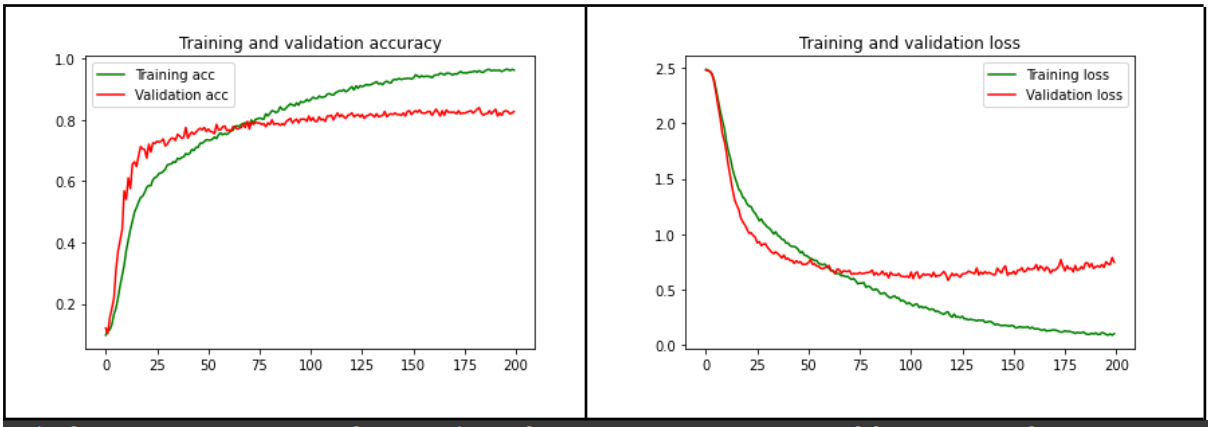
- Apesar de parecer um pouco contra-intuitivo, entradas com maior resolução, não necessariamente tiveram os melhores resultados.

- Com exceção dos experimentos com a **InceptionV3** e a **LeNet5** com a base 128x128px, todos os demais tiveram melhores resultados com o *Data Augmentation*.

A rede neural customizada **CNNL3**, apesar do modelo ter entrado em *overfitting*, conseguiu obter a maior acurácia (**0.8029**). A rede realmente superou as expectativas e por este motivo, decidi fazer uma última tentativa utilizando todos os conhecimentos adquiridos com os testes realizados.

Foi implementada então, uma segunda versão da rede **CNNL3** com as seguintes modificações:

- Foram introduzidas 2 camadas de *dropout*, uma entre a camada do vetor de características e a camada das classes e outra após a segunda camada de convolução, com o objetivo de ajudar a prevenir o *overfitting*.
- O *learning rate* foi reduzido para 0.003
- O parâmetro *optimizer* foi alterado de 'Adadelta' para 'SGD'
- O *dataset* utilizado foi o "64_a"
- O número de épocas foi alterado de 150 para 200




```
74/74 [==============================] - 3s 36ms/step - loss: 0.1069 - accuracy: 0.9584 - val_loss: 0.7264 - val_accuracy: 0.8204
Epoch 199/200
74/74 [==============================] - 3s 37ms/step - loss: 0.0961 - accuracy: 0.9621 - val_loss: 0.7897 - val_accuracy: 0.8229
Epoch 200/200
74/74 [==============================] - 3s 36ms/step - loss: 0.1127 - accuracy: 0.9576 - val_loss: 0.7483 - val_accuracy: 0.8279

-----------------------------------------------------

Test loss: 0.7483105063438416
Test accuracy: 0.8279301524162292
```

Figura 10. Resultados da segunda versão da rede CNNL3