Universidade Federal de Santa Catarina
Programa de Pós-Graduação em Ciência da Computação
Inteligência Artificial Conexionista

Augusto André Souza Berndt 202000735

Exercício 6 - Cat vs Non-Cat

Enunciado: Resolver o problema com um perceptron apenas (regressão logística), uma rede de camada rasa e uma rede convolucional. Um Relatório completo deve ser entregue, contendo informações de **Quantos e Quais experimentos foram feitos até chegar no resultado final**; Como foi o treinamento; Qual a taxa de acertos da rede; A matriz de confusão, etc…

Introdução: A cada página deste documento se realiza um experimento diferente. Cada experimento é numerado. Se realizou um total de 9 experimentos diferentes, tentando sempre reduzir o número de exemplares errados no conjunto de teste. O último experimento foi o que obteve o menor número de exemplares errados no teste, com um total de somente 3 exemplares errados.

1. Resultados Originais - Primeiramente armazenamos aqui as saídas originais como já disponibilizadas no código para futuras comparações. A Figura 1 demonstra que talvez a rede esteja sendo treinada com muitas épocas, uma vez que o loss da validação começa a se afastar do treino, próximo da época 125. Podemos ver também que a rede demonstrou uma ótima capacidade de generalização, confundindo apenas 9 exemplares.

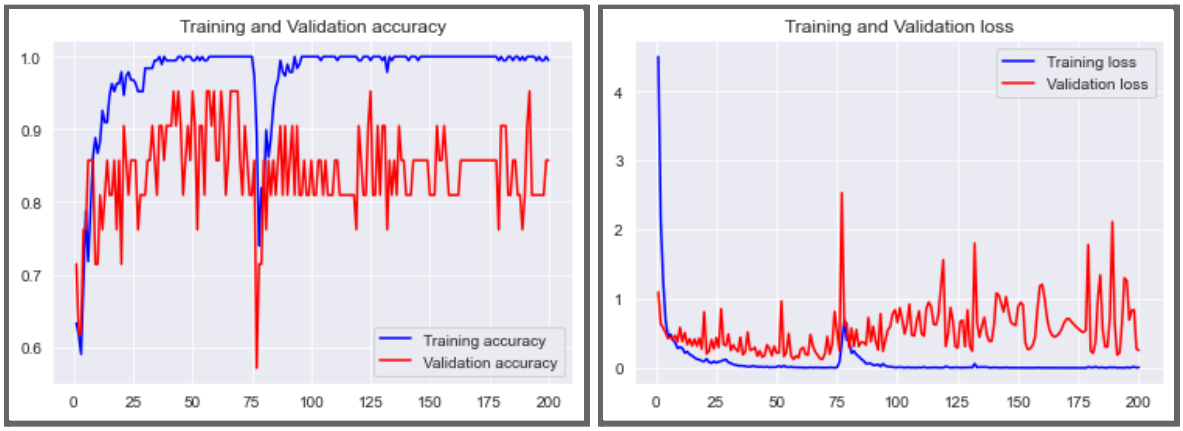


Figura 1 (treinamento com valores originais).

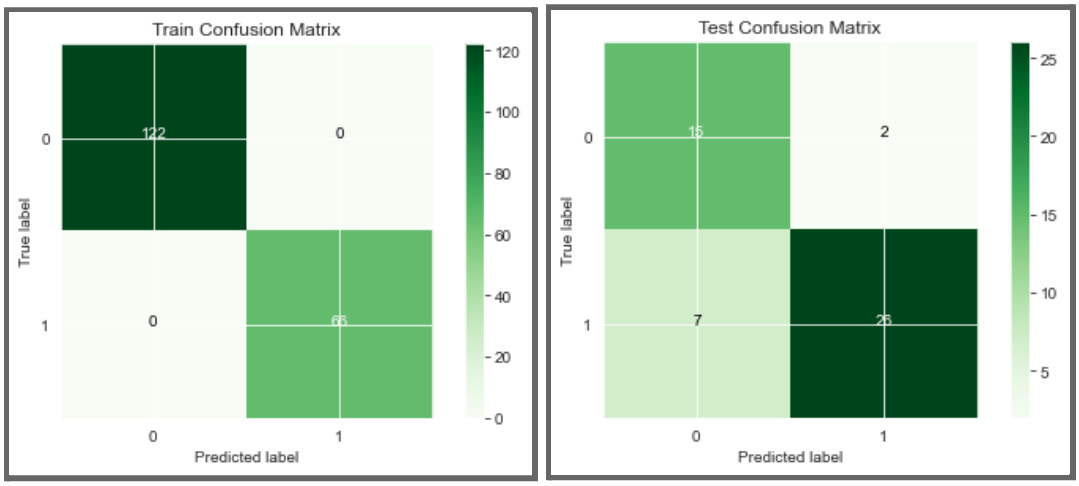


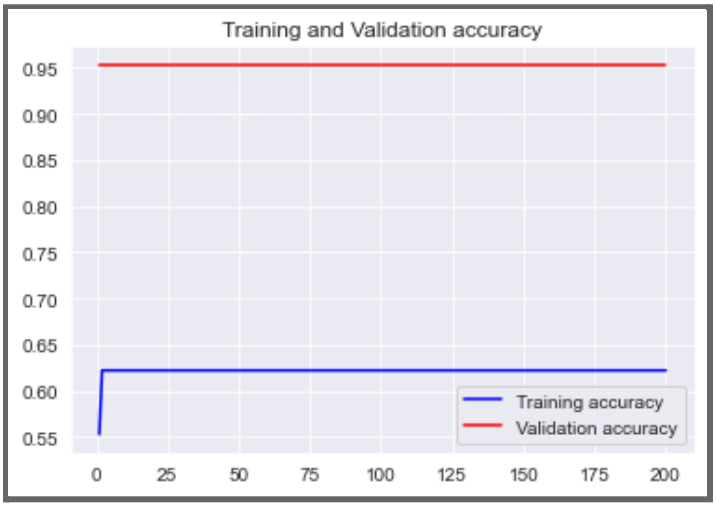
Figura 2 (matriz de confusão com valores originais).

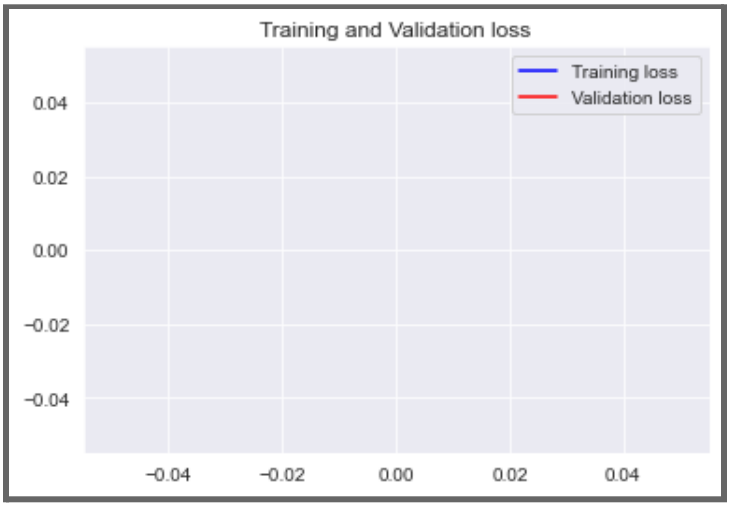
2. “Resolver o problema com um perceptron apenas (regressão logística)”: a regressão logística é feita com a função de ativação do tipo *sigmoid* em uma camada completamente conectada (*dense)* com apenas um neurônio, a camada de saída é mantida sem função de ativação para manter os formatos já esperados no resto do código. Não consegui deixar de usar a função flatten, sem ela os shapes sempre eram mantidos como (64,64,n) sendo n o número de neurônios da camada, quando o esperado seria como na Figura 4. O treinamento não se mostrou nada promissor, não tenho certeza se algo está errado, mas com um único perceptron o modelo não foi capaz de classificar as imagens. O formato printado pelo “model.summary” parece estar como esperando, uma única camada interna com apenas 1 neuronio e 12288 (=64*64*3) entradas e uma camada de saída com 2 neurônios sem ativação.

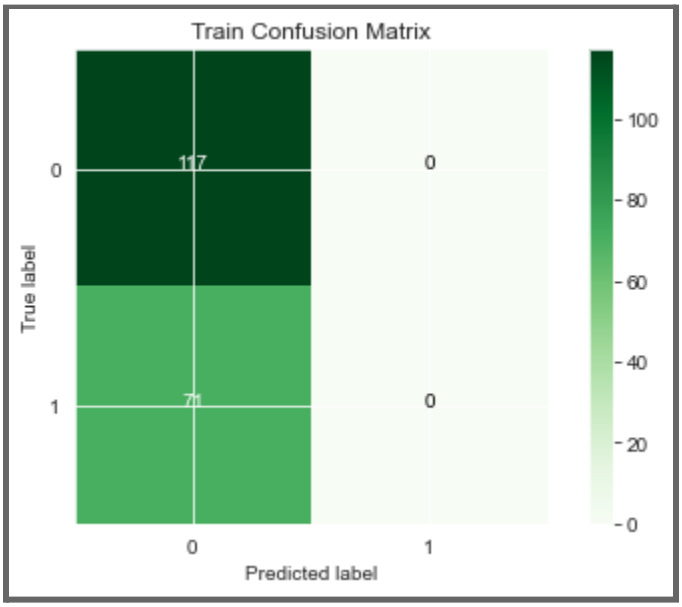
```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Flatten(),
        layers.Dense(1, activation="sigmoid"),
        layers.Dense(num_classes),
#         layers.Conv2D(32, kernel_size=(3, 3),
#         layers.MaxPooling2D(pool_size=(2, 2))
#         layers.Conv2D(64, kernel_size=(3, 3),
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 12288)             0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 12289
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 4
=================================================================
Total params: 12,293
Trainable params: 12,293
Non-trainable params: 0
```

Figura 3 (modelo com único perceptron).







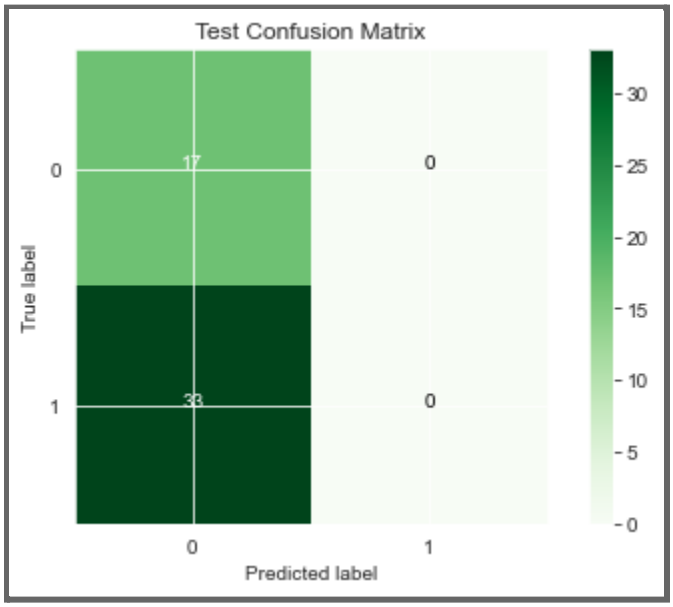


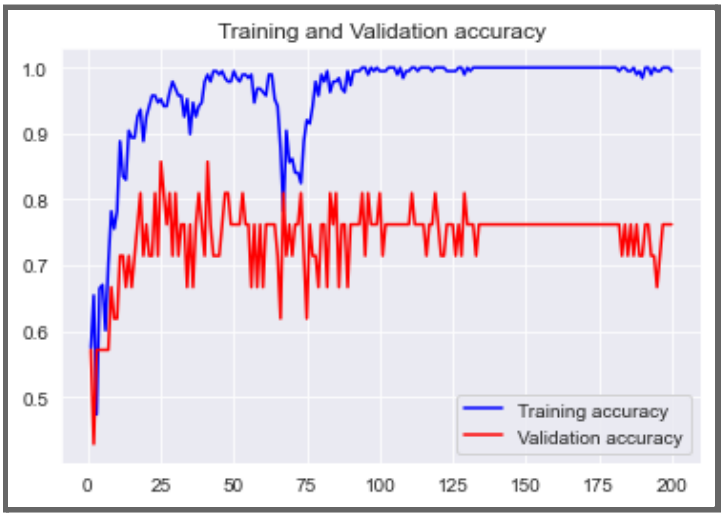
Figura 4 (treinamento de rede com 1 perceptron).

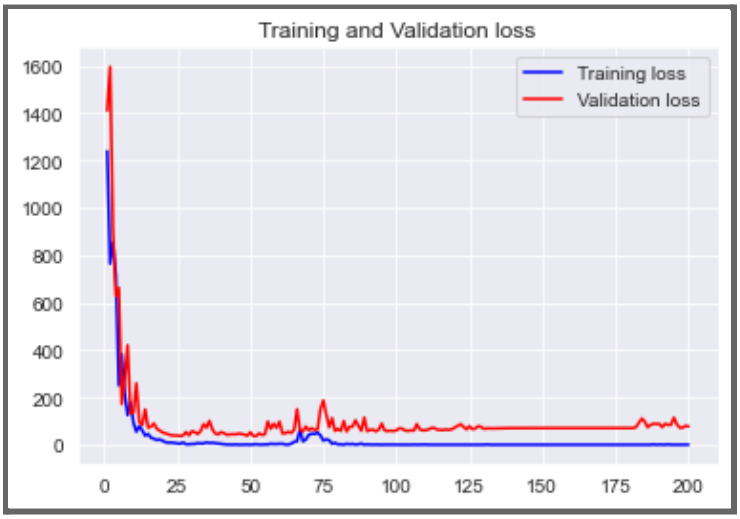
3. “Resolver o problema com uma rede de camada rasa”: Para isto utilizamos apenas uma camada interna com 50 neurônios e ativação ReLu, também se utiliza uma camada de saída com apenas dois neurônios e ativação *softmax*, como consta na Figura 5. Com 200 épocas de treinamento, o modelo já apresentou um treinamento bem bom, mas com um pequeno overfitting, uma vez que alguns indivíduos foram confundidos no teste e acertou todos no treinamento, como mostram as matrizes de confusão na Figura 6. A acurácia ao longo do treinamento demonstra o overfitting de maneira mais explícita.

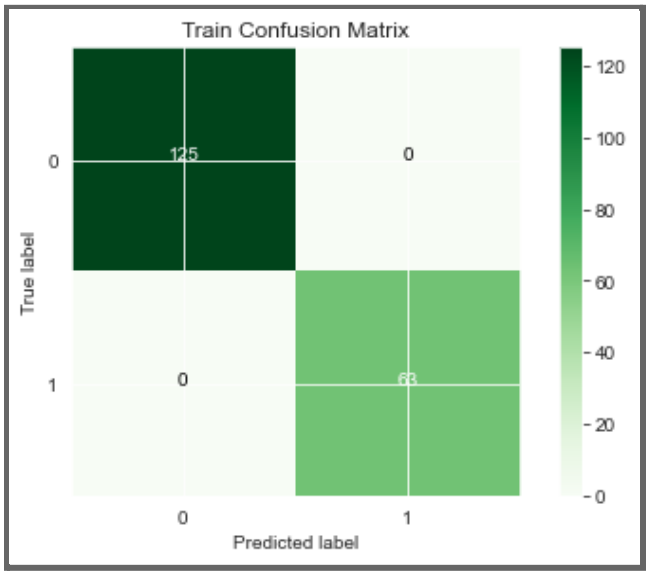
```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Dense(50, activation="relu"),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
#         layers.Conv2D(32, kernel_size=(3, 3), activatic
#         layers.MaxPooling2D(pool_size=(2, 2)),
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 64, 64, 50)        200
_________________________________________________________________
flatten_3 (Flatten)          (None, 204800)            0
_________________________________________________________________
dropout_2 (Dropout)          (None, 204800)            0
_________________________________________________________________
dense_7 (Dense)              (None, 2)                 409602
=================================================================
Total params: 409,802
Trainable params: 409,802
Non-trainable params: 0
```

Figura 5 (modelo com camada rasa).







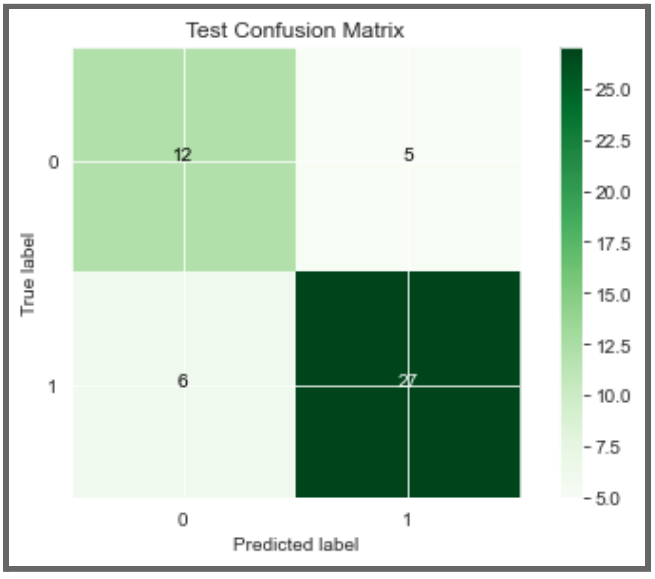


Figura 6 (treinamento com modelo de camada rasa).

4. “Resolver o problema com um perceptron apenas (regressão logística), uma rede de camada rasa e uma rede convolucional” (150 epocas): Para este desafio tentei ajustar a rede já disponibilizada pelo professor com diversas camadas convolucionais. Um treinamento somente fazendo menos épocas já demonstra resultados melhores que os originais. O modelo original erra 9 exemplares, já o modelo atual erra 6 exemplares. Esta melhora mostra que talvez seja possível encontrar uma configuração de treinamento que acerte todos, ou quase todos, exemplares de teste. A Figura 8 mostra uma boa evolução no treinamento, a função custo avança junto entre validação e treino, o que é um bom indicador. No entanto a acurácia demonstra um certo afastamento entre treino e validação, talvez seja interessante um treinamento com ainda menos épocas.

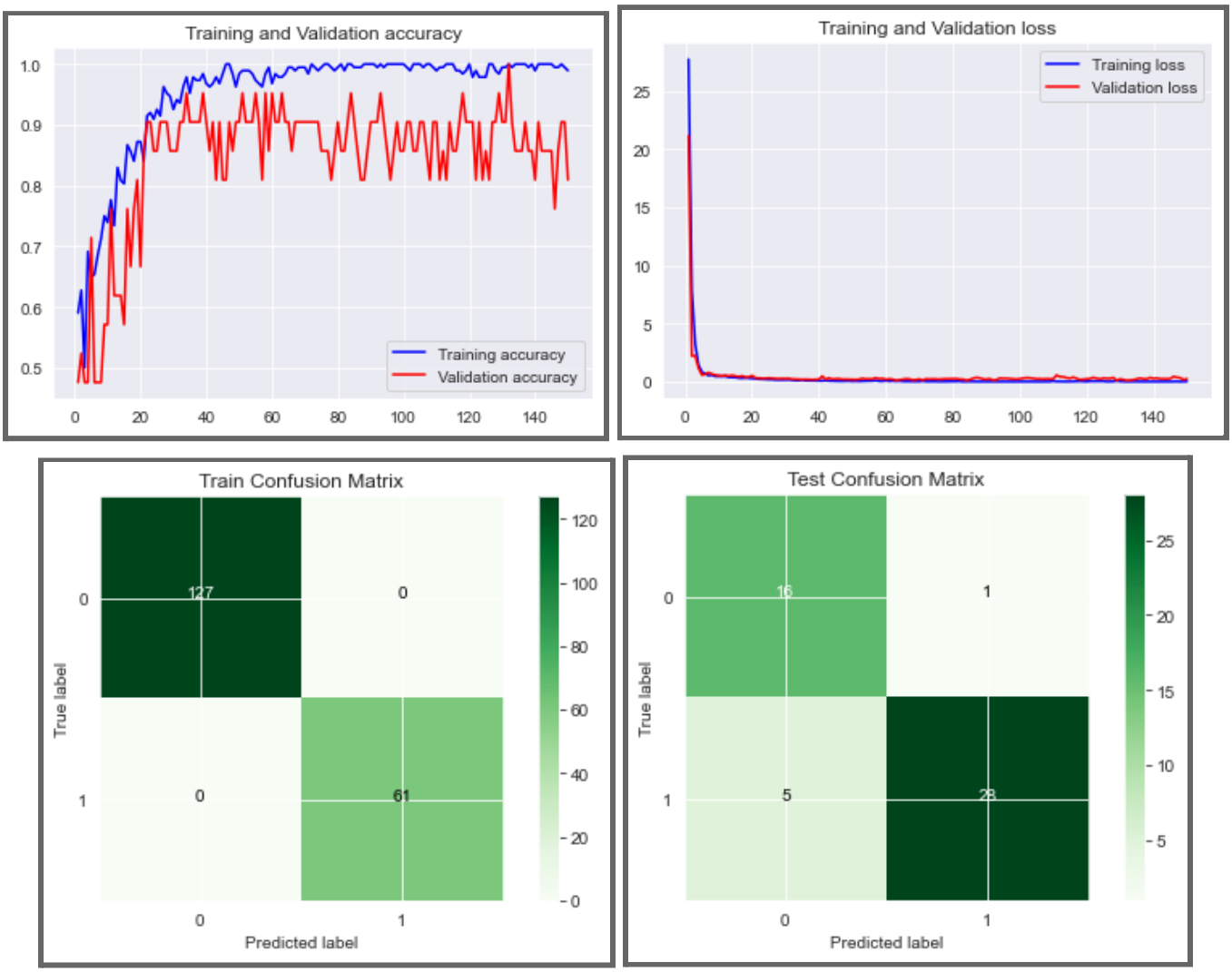


Figura 8 (treinamento com rede convolucional - **150 épocas).**

5. 110 épocas: A partir daqui irei tentar ajustar a rede já disponibilizada na intenção de melhorar os resultados de treinamento. O objetivo é reduzir o número de exemplares errados no conjunto de teste. Primeiro tentamos somente reduzir um pouco mais o número de épocas. A arquitetura se mantém a mesma, assim como no experimento anterior, só alterei o números de épocas do treinamento. Desta vez temos 10 exemplares errados no teste, ou seja, pior do que o original.

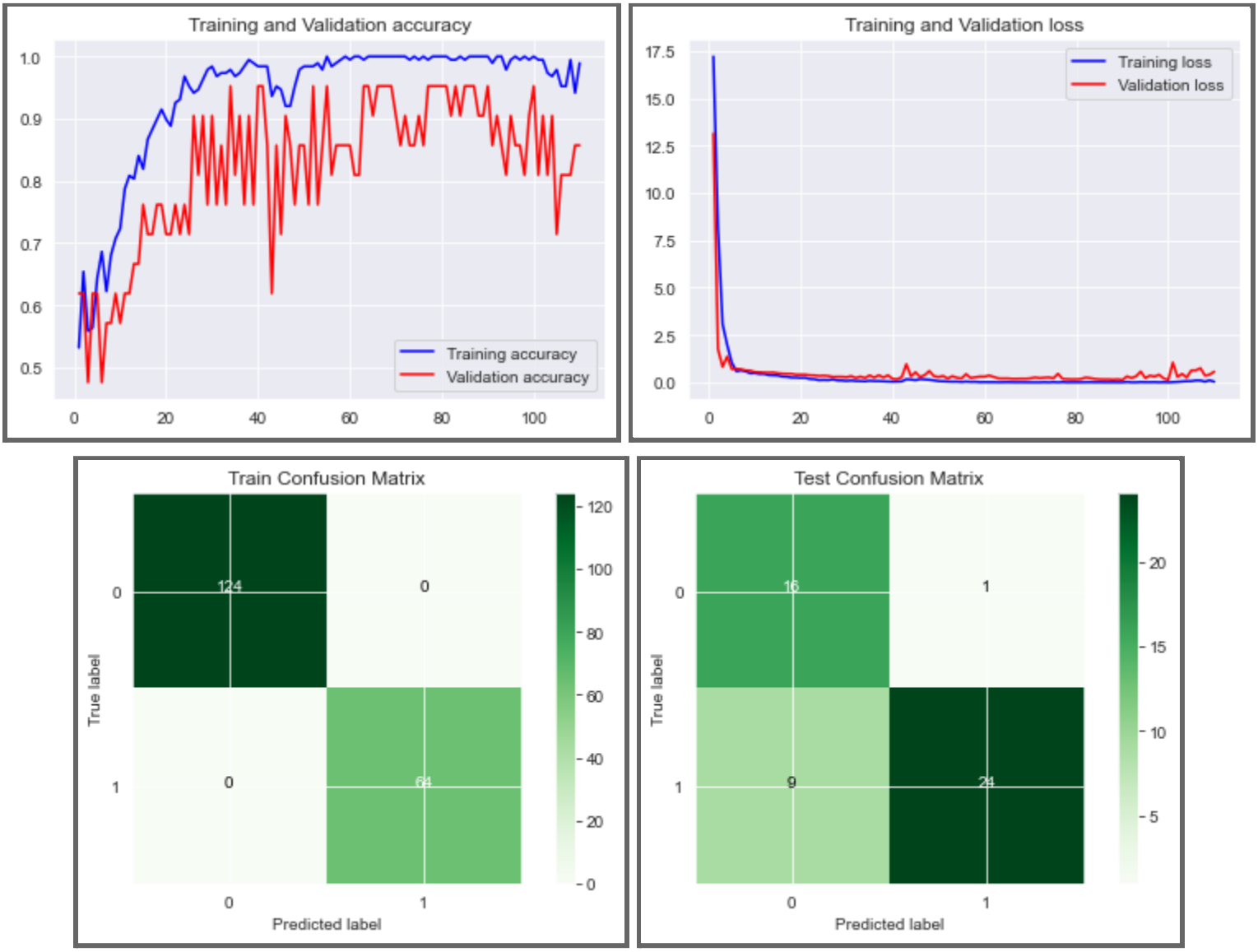


Figura 9 (treinamento com **110 épocas).**

6. 125 épocas: Desta vez temos 7 exemplares errados no conjunto de teste. Podemos ver que o número ótimo de épocas é próximo de 150. A partir de agora tentarei modificar a arquitetura da rede, mantendo 150 épocas, uma vez que até o momento foi o melhor experimento.

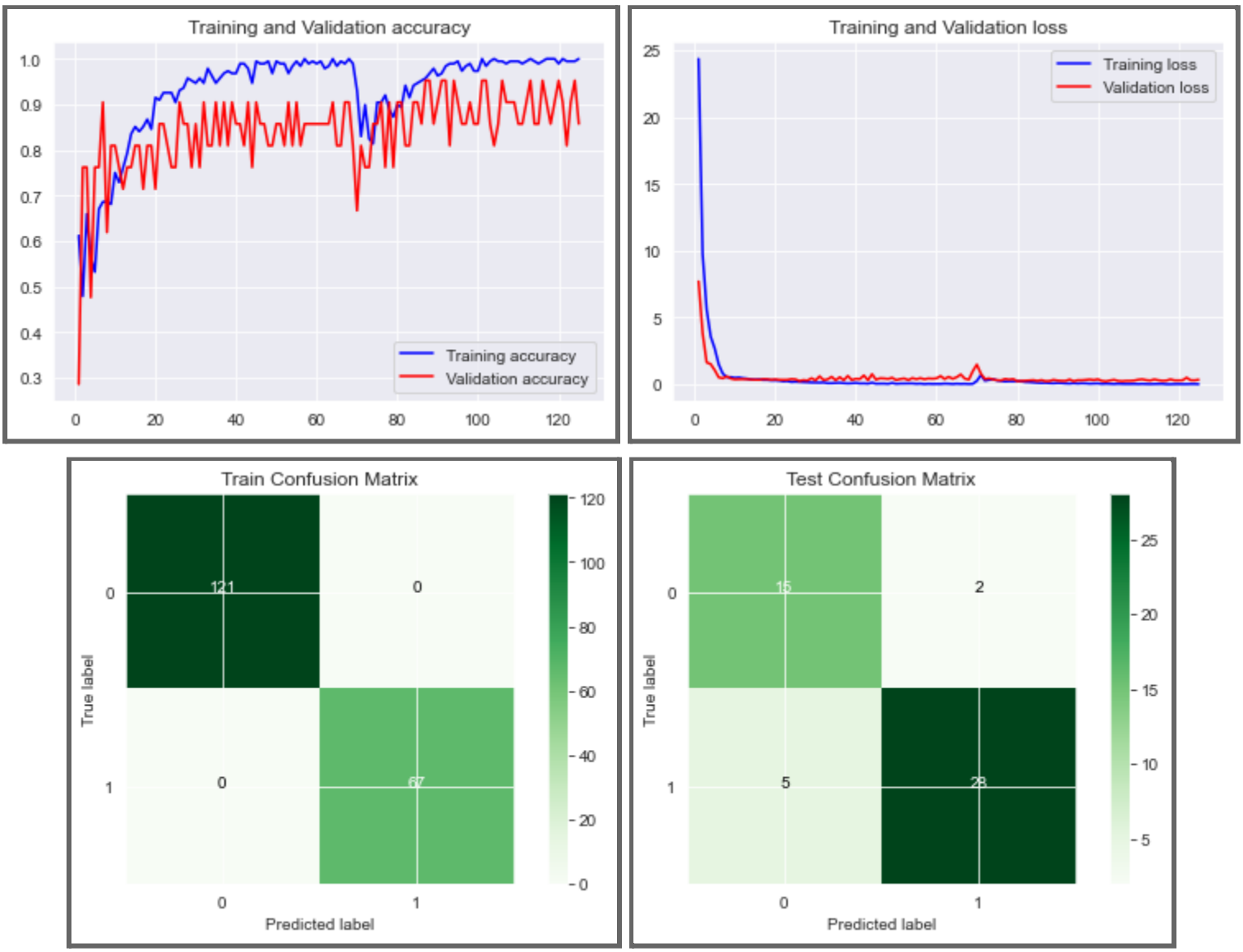


Figura 10 (treinamento com **125 épocas).**

7. Substituindo alguns ReLu's por SeLu (exercício anterior [5] foram melhores do que ReLu) com 150 épocas: Obtivemos 8 exemplares errados intercalando entre ReLu e SeLu. Ainda não foi melhor do que o experimento 4 que obteve somente 6 exemplares errados no teste.

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="selu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(128, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="selu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
```

Figura 11 (modelo ReLu e SeLu).

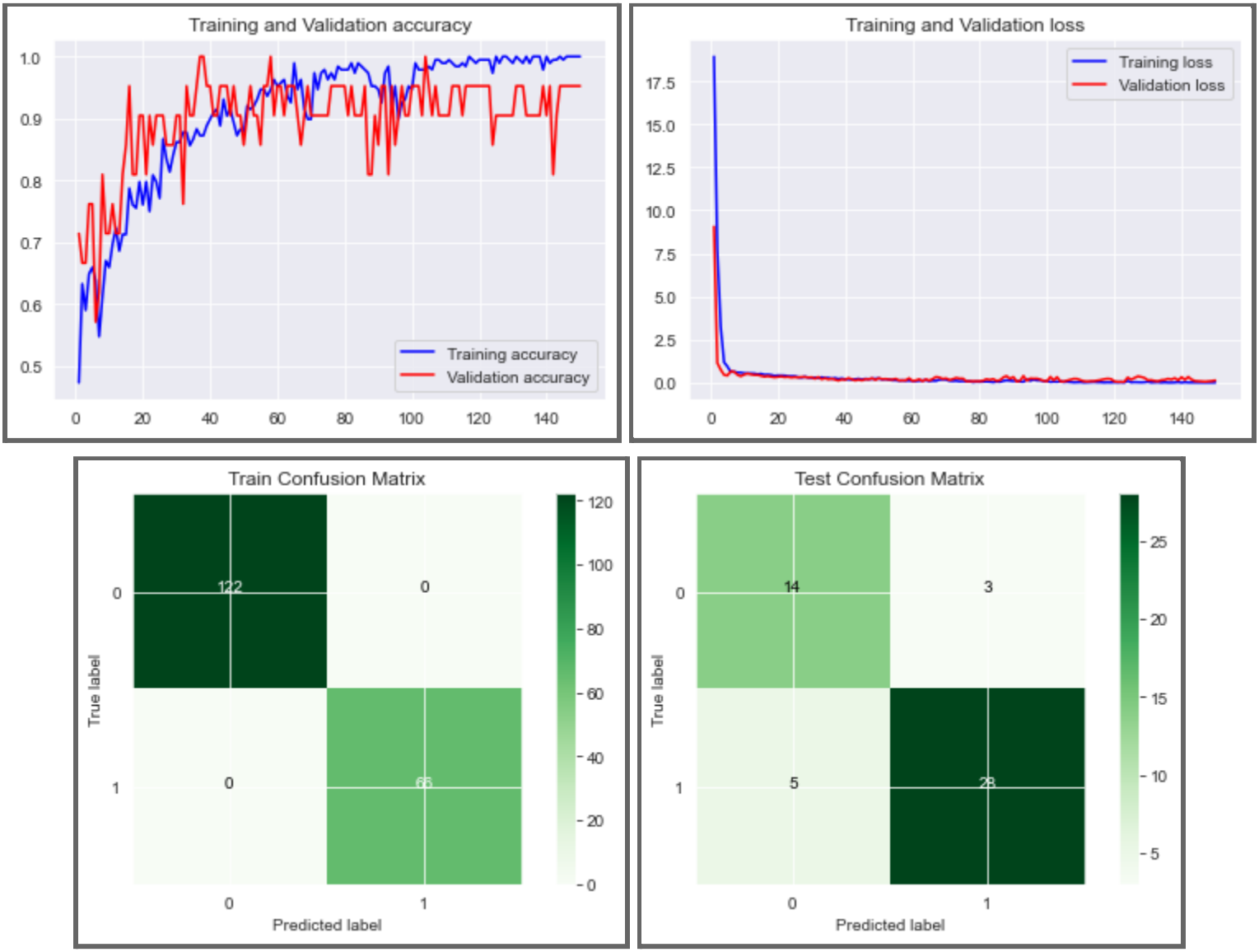


Figura 12 (treinamento com modelo ReLu e SeLu).

8. Todas camadas internas como SeLu e 150 épocas: Esta configuração também errou somente 6 exemplares, como o experimento 4, no entano, o treinamento se mostrou um pouco melhor, uma vez que a acurácia avança mais próxima entre validação e treino. No experimento 4 a acurácia de treinamento converge mais do que a de validação, como mostra a Figura 8, a linha vermelha de validação fica mais abaixo do que a linha vermelha do presente experimento.

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="selu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="selu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(128, kernel_size=(3, 3), activation="selu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="selu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
```

Figura 13 (modelo com todas camadas SeLu).

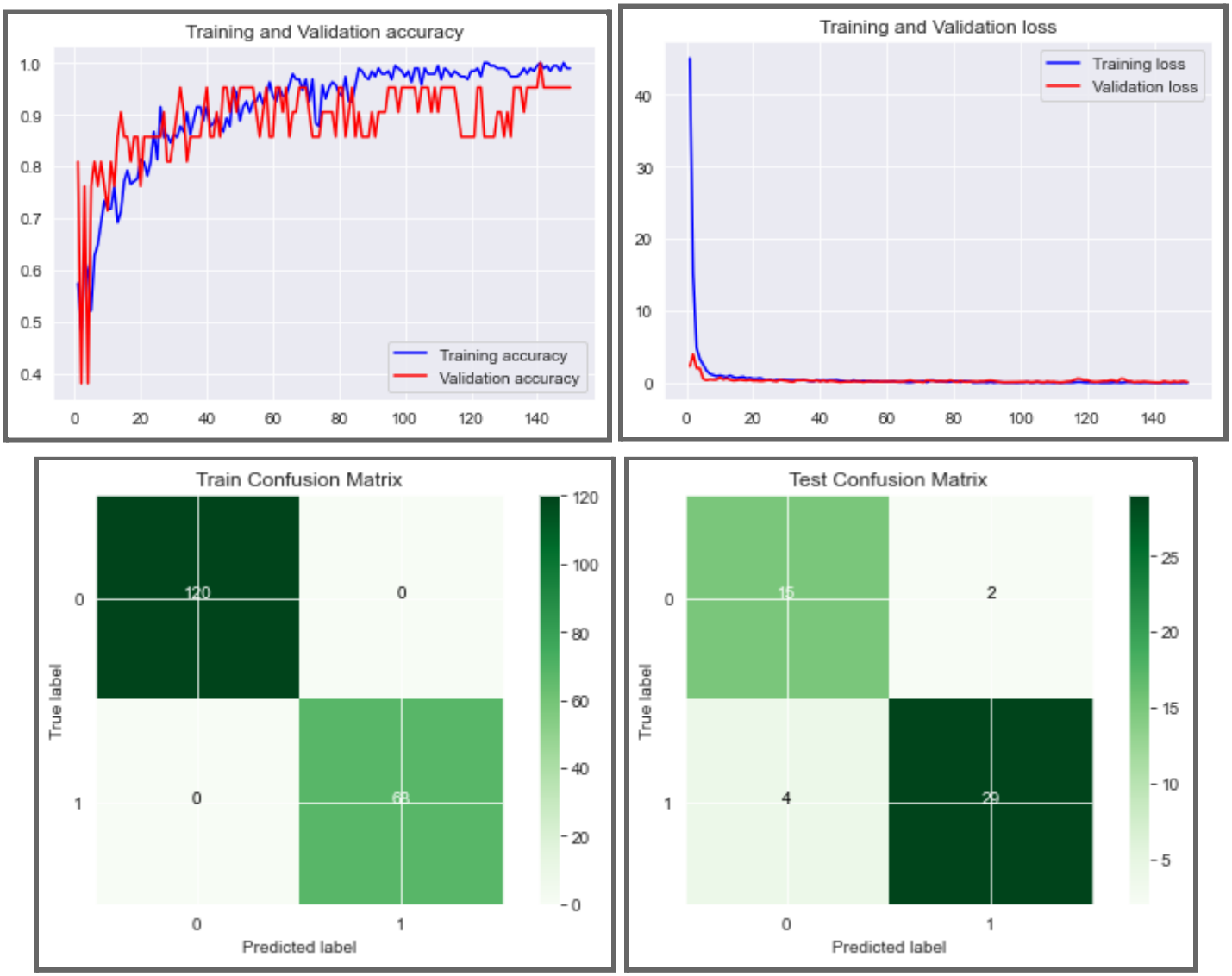


Figura 14 (treinamento com todas camadas SeLu).

9. Camada de saída sigmoid: Na mosca! Utilizando todas camadas escondidas como SeLu e a camada de saída como sigmoid em vez de softmax se obteve um modelo que errou somente 3 exemplares no teste! Acredito que dificilmente teremos um resultado melhor do que esse!

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="selu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="selu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(128, kernel_size=(3, 3), activation="selu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="selu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="sigmoid"),
    ]
)
```

Figura 15 (modelo SeLu e saída Sigmoid).

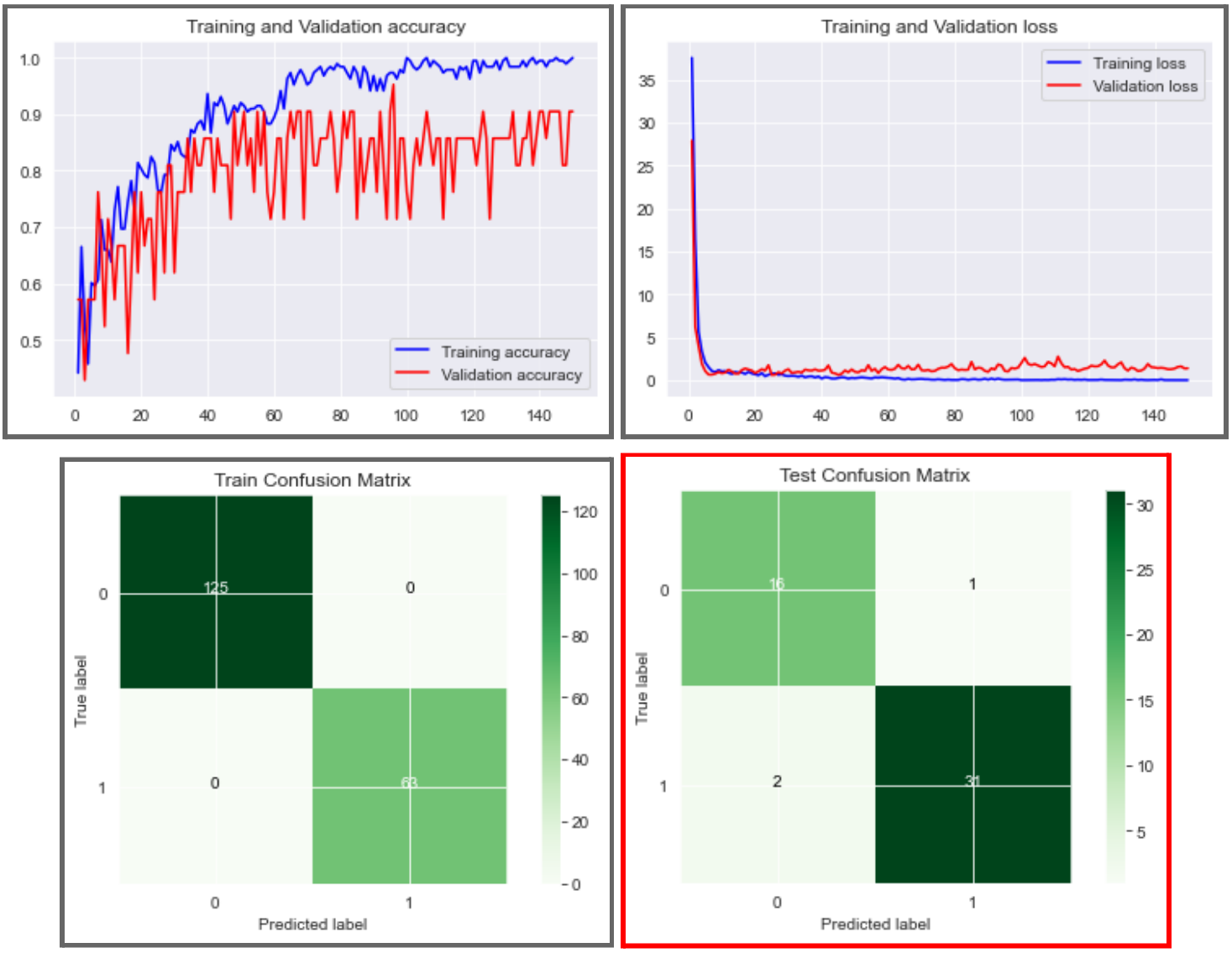


Figura 16 (treinamento vencedor com modelo Selu e saída Sigmoid).