SISTEMAS DE INFORMAÇÃO

EACH USP 2024

INTELIGÊNCIA ARTIFICIAL
CNN

MARIA EDUARDA GARCIA - 11796621
MIRELA MEI - 11208392

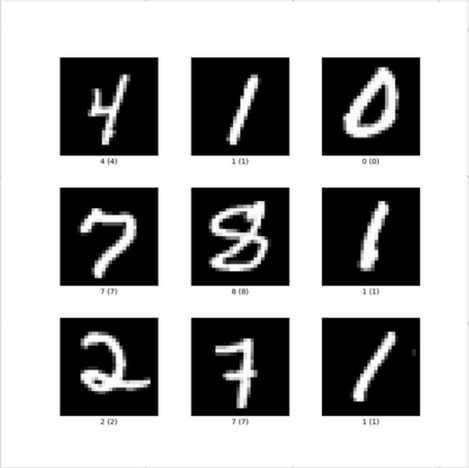
ESTRUTURA

- Python
- MNIST
- Bibliotecas
  - tensorflow
  - matplotlib
  - scikit-learn
  - numpy
  - seaborn

MODELAGEM DAS ENTRADAS

- conjunto de dados **MNIST**: imagens de dígitos manuscritos com tamanho 28x28 pixels
- cada imagem é **normalizada** para valores entre 0 e 1 dividindo por 255
- uma **dimensão extra** é adicionada para representar o canal (único canal, pois são imagens em escala de cinza)

```python
# data_preprocess.py
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()
train_images = train_images / 255.0
test_images = test_images / 255.0
train_images = train_images[..., tf.newaxis]
test_images = test_images[..., tf.newaxis]
```



CAMADAS DE APLICAÇÃO DE KERNEL E POOLING

- camadas **convolucionais** (Conv2D) aplicam filtros (**kernels**) para extrair características das imagens. usamos **filtros de tamanho 3x3**
- camadas de **pooling** (MaxPooling2D) reduzem a dimensionalidade das características extraídas, mantendo as mais importantes. usamos **pooling de tamanho 2x2**

```python
# cnn_model.py
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu')
])
```

MODELAGEM DAS CAMADAS DENSAS E CAMADA DE SAÍDA

- camadas **densas** (Dense) conectam todas as unidades da camada anterior com todas as unidades da próxima camada
- a camada **final** usa uma ativação **sigmoid para classificação binária** ou **softmax para classificação multiclasses**

```python
# cnn_model.py
model = models.Sequential([
    # Camadas convolucionais e de pooling...
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')  # Para classificação binária
    # layers.Dense(10, activation='softmax')  # Para classificação multiclasses
])
```

LÓGICA DO TREINAMENTO DA ESTRUTURA COMPLETA

- a função **train_and_evaluate** treina o modelo com os dados de treinamento e avalia o desempenho com os dados de teste
- utiliza o **otimizador Adam** e a função de perda adequada (**binary_crossentropy** ou **sparse_categorical_crossentropy**)

```python
# train_and_evaluate.py
def train_and_evaluate(model, train_images, train_labels, test_images, test_labels, epochs=5):
    history = model.fit(train_images, train_labels, epochs=epochs, validation_data=(test_images,
test_labels))
    test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
    print(f'Test accuracy: {test_acc}')
    predictions = model.predict(test_images)
    return history, test_loss, test_acc, predictions
```

CRITÉRIO DE PARADA DO TREINAMENTO

- o critério de parada é baseado no número de **épocas especificadas**. neste caso, o treinamento ocorre por um número fixo de épocas (**padrão: 5**)

```
# train_and_evaluate.py
history = model.fit(train_images, train_labels, epochs=epochs, validation_data=(test_images,
test_labels))
```

PROCEDIMENTOS DE CÁLCULO DE ERRO NA CAMADA DE SAÍDA

- a função de perda **binary_crossentropy** é usada para classificação **binária**. ela calcula a diferença entre os valores previstos e os valores reais
- Para classificação **multiclasses**, **sparse_categorical_crossentropy** é utilizada

```
# cnn_model.py
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

PROCEDIMENTO DE CÁLCULO DA RESPOSTA DA REDE EM TERMOS DE RECONHECIMENTO DE CARACTERE

- a previsão do modelo para uma imagem é realizada usando **model.predict**
- para a classificação **binária**, a saída é uma **probabilidade e é convertida para 0 ou 1** (usando um limiar de 0.5)

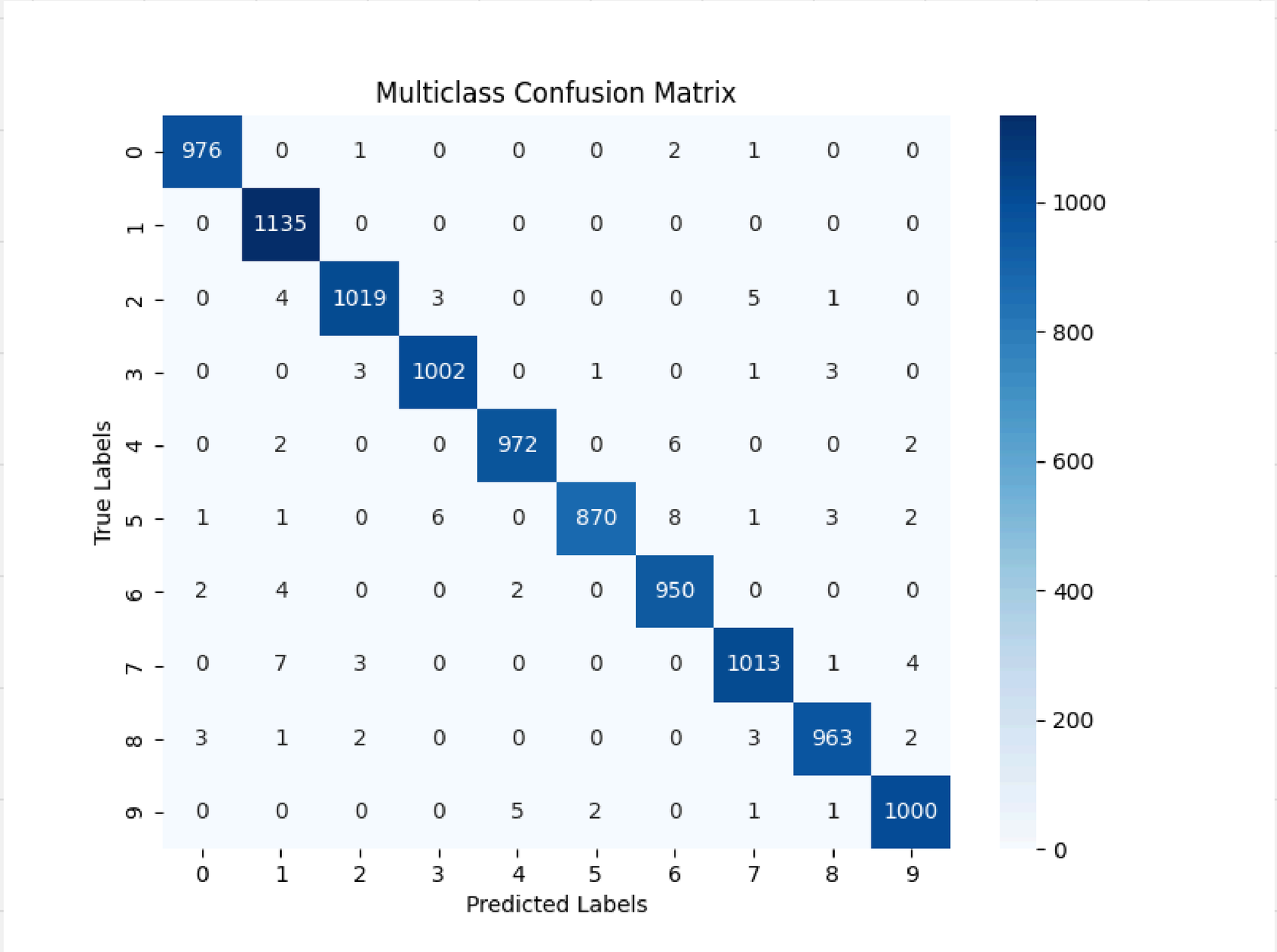
```
# main.py
binary_pred_labels = (binary_predictions > 0.5).astype(int).flatten()
```

PROCEDIMENTO DE CÁLCULO DA MATRIZ DE CONFUSÃO

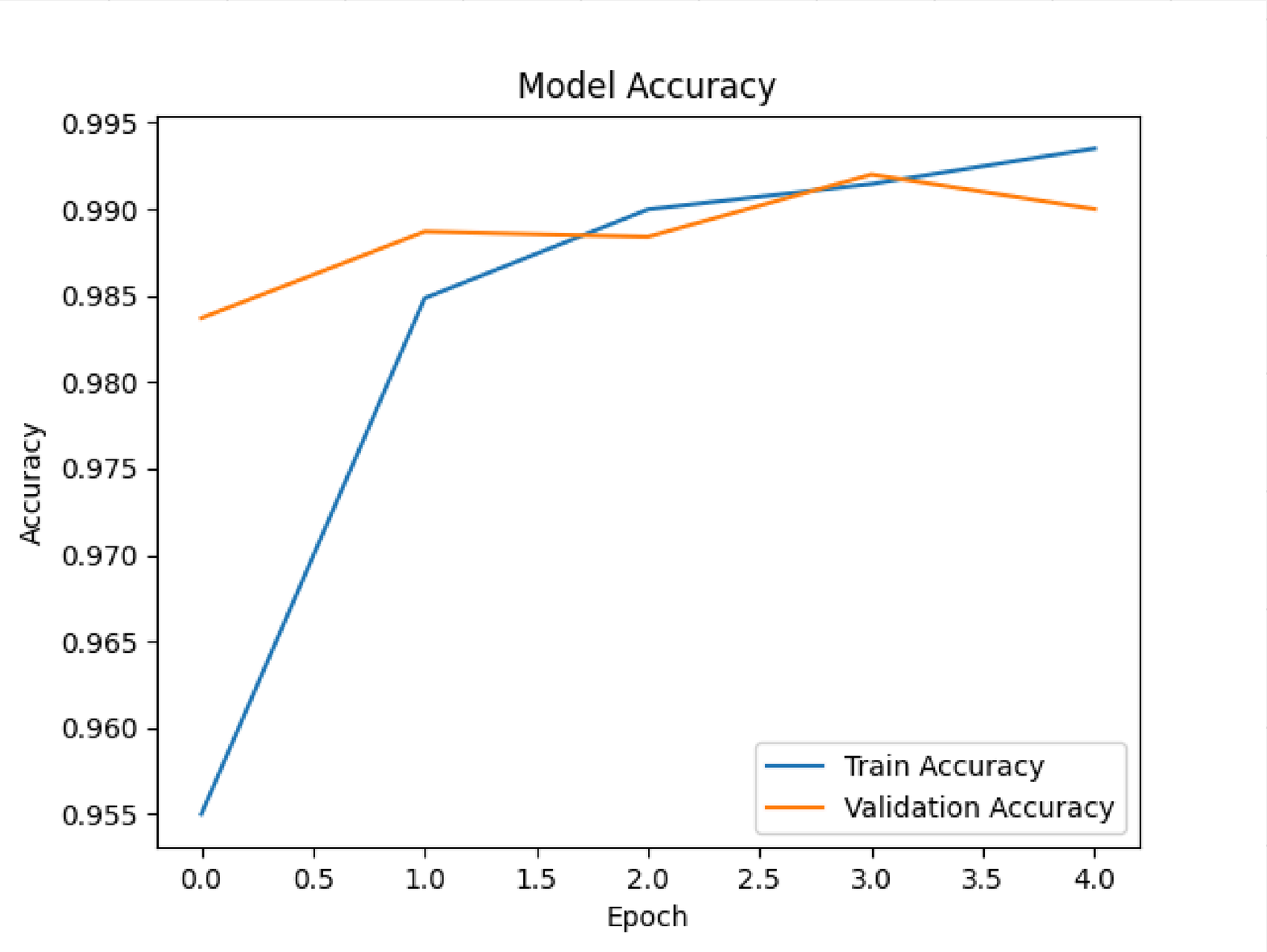
- a matriz de confusão é calculada usando **confusion_matrix** do sklearn.metrics

```
# main.py
binary_cm = confusion_matrix(binary_test_labels, binary_pred_labels)
plot_confusion_matrix(binary_cm, class_names=['Not ' + str(target_class), str(target_class)],
title=f'Binary Confusion Matrix (Not {target_class} vs {target_class})', filename=f'plot/
binary_confusion_matrix_{target_class}.png')
```

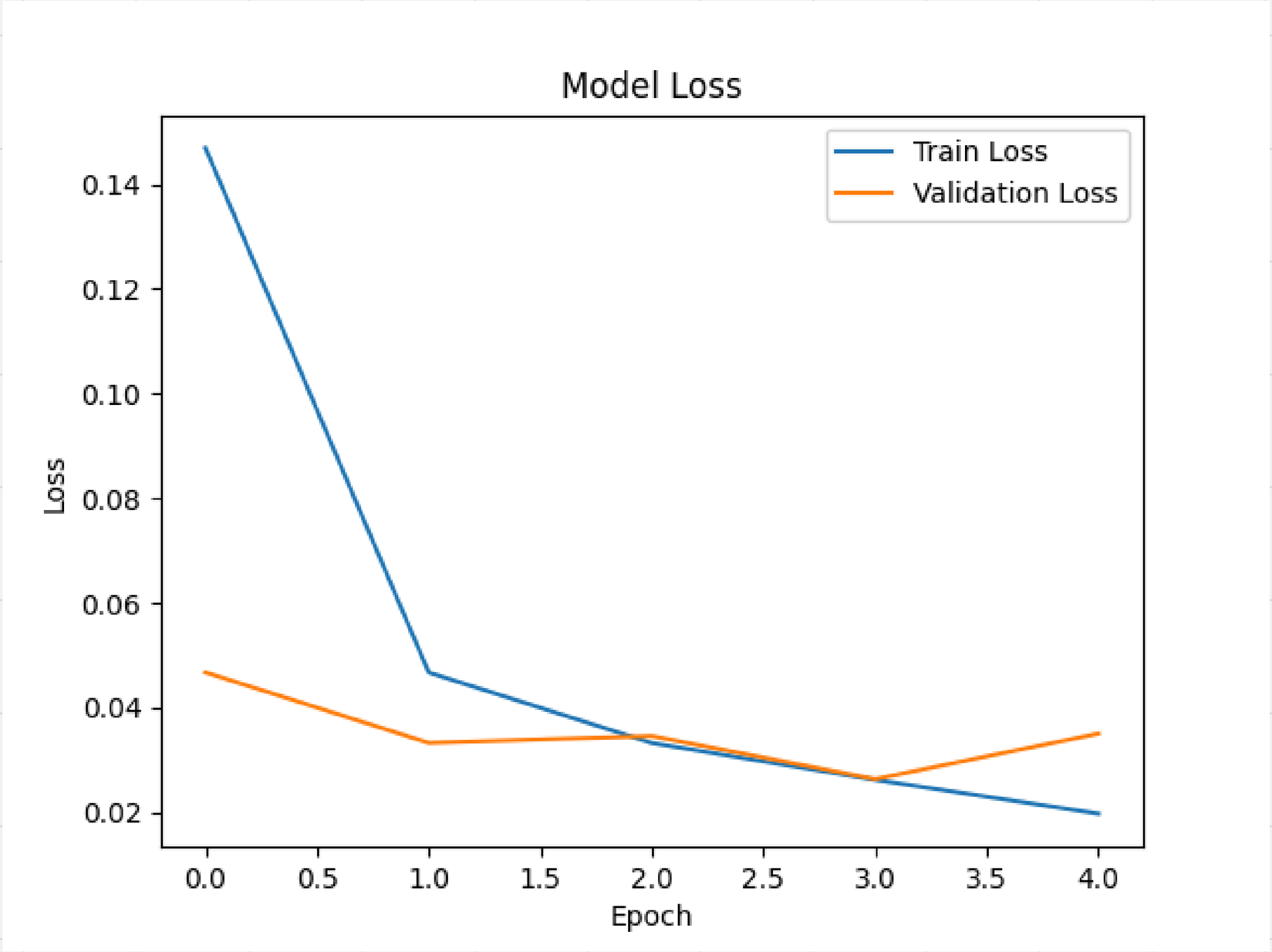
MATRIZ DE CONFUSÃO



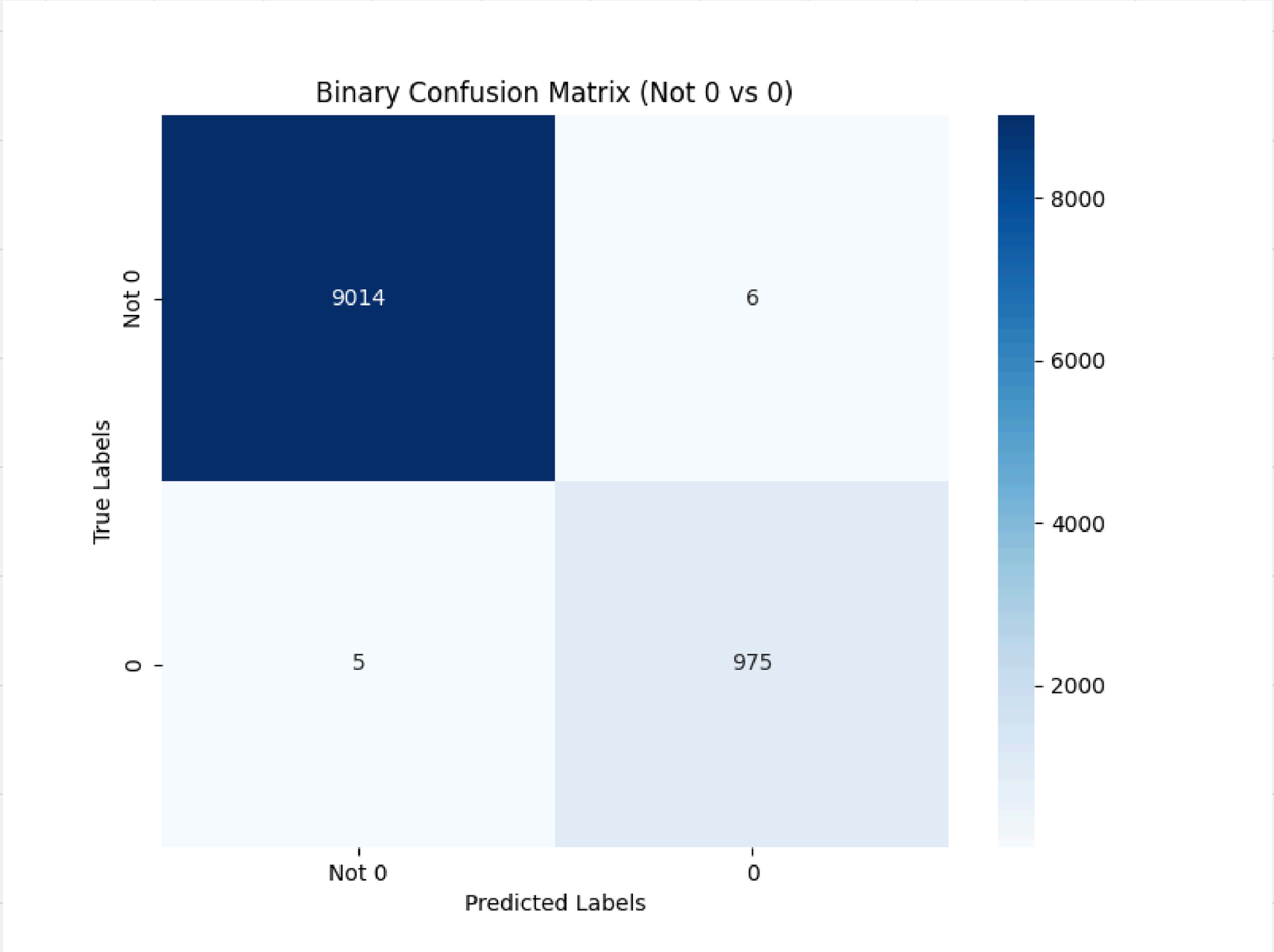
ACURÁCIA


Model Accuracy
Accuracy
Epoch
0.995
0.990
0.985
0.980
0.975
0.970
0.965
0.960
0.955
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Train Accuracy
Validation Accuracy

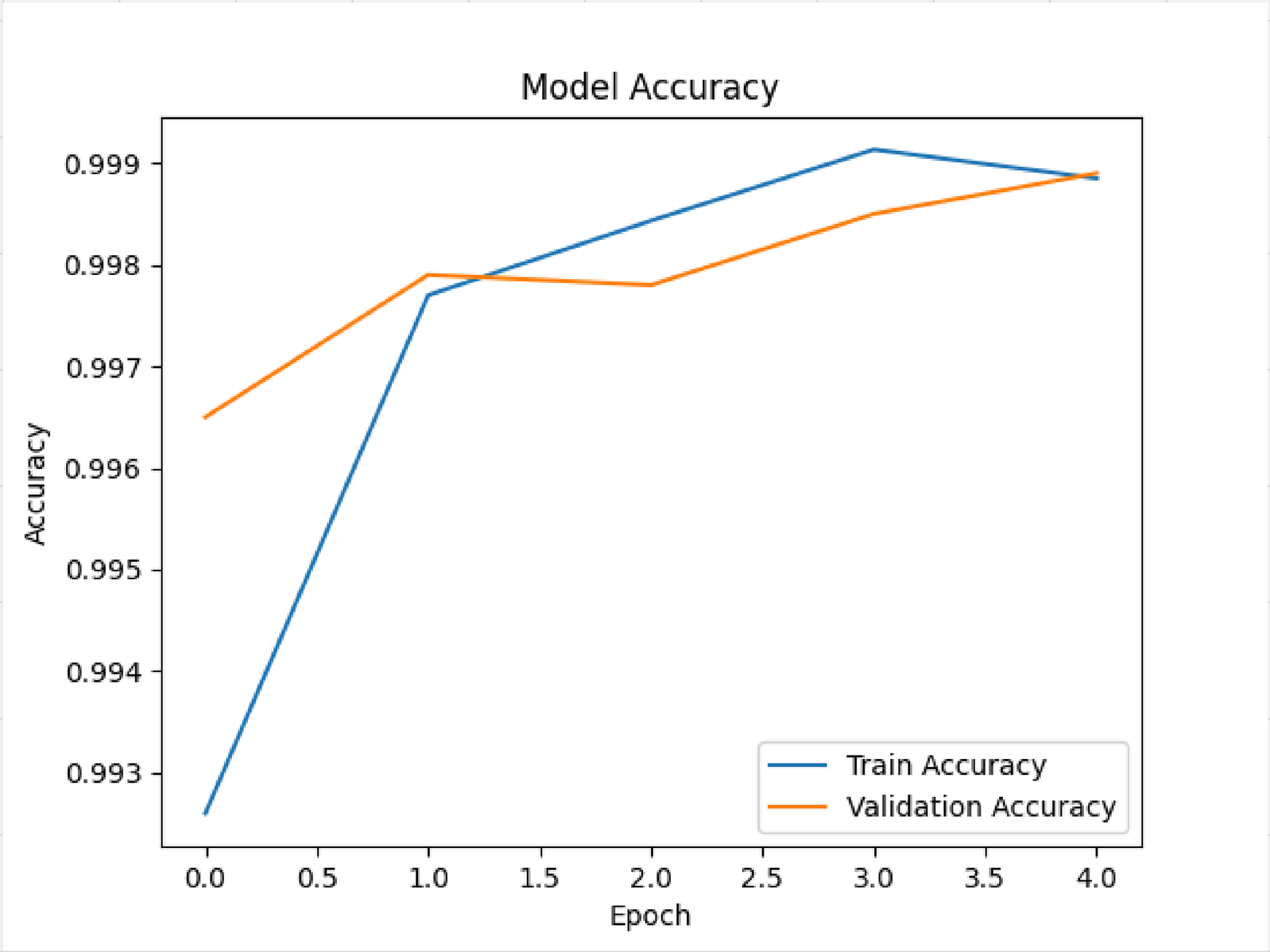
PERDA


Model Loss
Train Loss
Validation Loss
Loss
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epoch

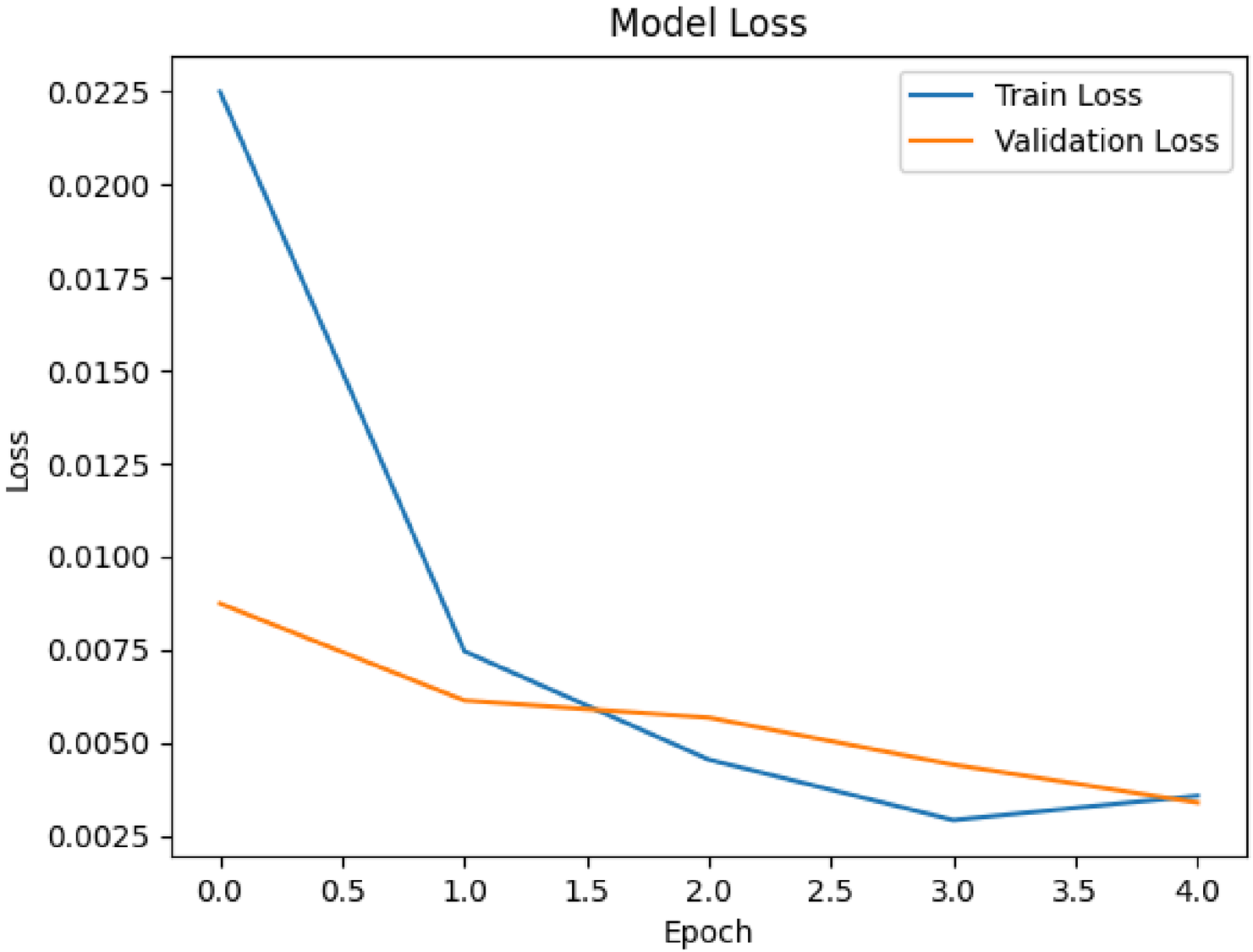
MATRIZ DE CONFUSÃO



ACURÁCIA


Model Accuracy
Accuracy
0.999
0.998
0.997
0.996
0.995
0.994
0.993
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epoch
Train Accuracy
Validation Accuracy

PERDA


Model Loss
Train Loss
Validation Loss
Loss
0.0225
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epoch

OBRIGADA