Lista Prática: Análise Compartiva de Modelos com MNIST

MLP e Redes Convolucionais

Introdução à Aprendizagem Profunda

Camila Barbosa Vieira (cbv2)

Imports & Functions

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.applications import VGG16, VGG19, ResNet50, DenseNet121
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from tabulate import tabulate
import random
import time

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


#Reproducibility and Model Comparison
seed = 42
np.random.seed(seed)
tf.random.set_seed(seed)

#Result Data
models_list = []
accuracies_list = []
execution_times_list = []
most_difficult_digits_list = []
TPR_min = []


def train_base_model(model):
  start_time_dt = time.time()
  model.fit(train_x, train_y)
  end_time_dt = time.time()
  execution_time_dt = end_time_dt - start_time_dt
  return execution_time_dt
```

```
def train_nn_model(model, batch_size=128, epochs=10):
  start_time = time.time()
  model.fit(train_x, train_y, batch_size=batch_size, epochs=epochs, validation_split=0.2)
  end_time = time.time()
  execution_time = end_time - start_time
  return execution_time

def evaluate_base_model(name, model, execution_time):
  y_pred = model.predict(test_x)
  accuracy = accuracy_score(test_y, y_pred)
  print(name, f"accuracy: {accuracy:.4f}\n")

  confusionMatrix = confusion_matrix(test_y, y_pred)
  print("Classification Report")
  print(classification_report(test_y, y_pred))

  print("Confusion Matrix:")
  print(confusionMatrix)
  print("\nExecution Time:", execution_time, "seconds\n")

  TPR = confusionMatrix.diagonal() / confusionMatrix.sum(axis=1)
  print("True Positive Rate for each class:")
  print(TPR)

  models_list.append(name)
  accuracies_list.append(accuracy)
  execution_times_list.append(execution_time)
  most_difficult_digits_list.append(TPR.argmin())
  TPR_min.append(TPR.min())

def evaluate_nn_model(name, model, execution_time):
  y_pred = model.predict(test_x)
  y_pred = np.argmax(y_pred, axis=1)
  accuracy = accuracy_score(np.argmax(test_y, axis=1), y_pred)
  print(name, f"accuracy: {accuracy:.4f}\n")

  confusionMatrix = confusion_matrix(np.argmax(test_y, axis=1), y_pred)
  print("Classification Report")
  print(classification_report(np.argmax(test_y, axis=1), y_pred))

  print("Confusion Matrix:")
  print(confusionMatrix)
  print("\nExecution Time:", execution_time, "seconds\n")

  TPR = confusionMatrix.diagonal() / confusionMatrix.sum(axis=1)
  print("True Positive Rate for each class:")
  print(TPR)

  models_list.append(name)
  accuracies_list.append(accuracy)
  execution_times_list.append(execution_time)
  most_difficult_digits_list.append(TPR.argmin())
  TPR_min.append(TPR.min())
```

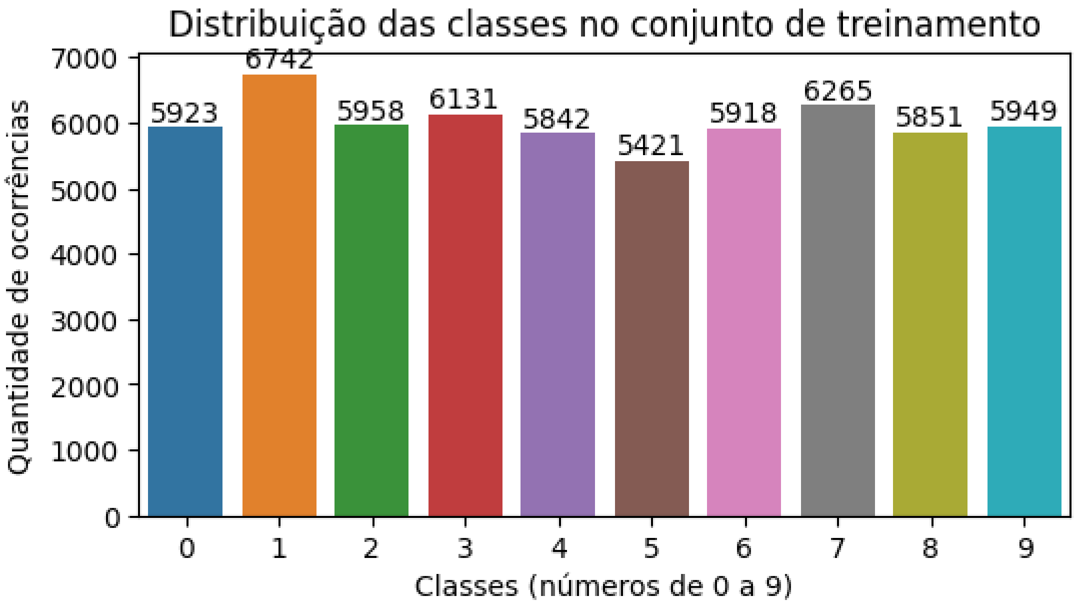
Base Model

Data Preparation & Analysis

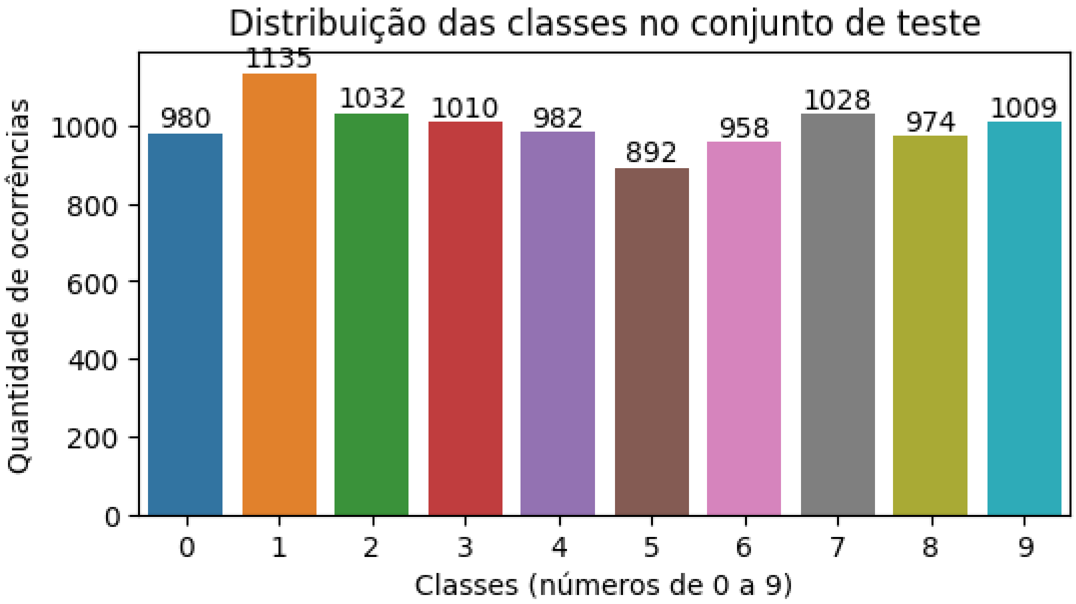
```
#Loading Dataset MNIST
(train_x, train_y), (test_x, test_y) = tf.keras.datasets.mnist.load_data()

#Input Shape (reshape and normalization)
train_x = train_x.reshape(train_x.shape[0], -1).astype('float32') / 255.0
test_x = test_x.reshape(test_x.shape[0], -1).astype('float32') / 255.0

inp_shape = train_x.shape
inp_shape
```

```
(60000, 784)
```

```
#Distribuição das classes no conjunto de treinamento
classes, counts = np.unique(train_y, return_counts=True)

plt.figure(figsize = (6, 3))
plt.title("Distribuição das classes no conjunto de treinamento")
plt.xlabel('Classes (números de 0 a 9)')
plt.ylabel('Quantidade de ocorrências')
sns.barplot(x=classes, y=counts)
for index, value in enumerate(counts):
    plt.text(index, value, str(value), ha='center', va='bottom')

plt.show()
```



```
#Distribuição das classes no conjunto de teste
classes, counts = np.unique(test_y, return_counts=True)

plt.figure(figsize = (6, 3))
plt.title("Distribuição das classes no conjunto de teste")
plt.xlabel('Classes (números de 0 a 9)')
plt.ylabel('Quantidade de ocorrências')
sns.barplot(x=classes, y=counts)
for index, value in enumerate(counts):
    plt.text(index, value, str(value), ha='center', va='bottom')

plt.show()
```



Decision Tree

```
dt_classifier = DecisionTreeClassifier(random_state=42)
execution_time = train_base_model(dt_classifier)
evaluate_base_model("Decision Tree", dt_classifier, execution_time)
```

```
Decision Tree accuracy: 0.8754

Classification Report
              precision    recall  f1-score   support

           0       0.91      0.93      0.92       980
           1       0.95      0.96      0.95      1135
           2       0.86      0.86      0.86      1032
           3       0.83      0.85      0.84      1010
           4       0.86      0.87      0.87       982
           5       0.85      0.83      0.84       892
           6       0.90      0.88      0.89       958
           7       0.91      0.90      0.91      1028
           8       0.82      0.81      0.81       974
           9       0.85      0.85      0.85      1009

    accuracy                           0.88     10000
```

```
   macro avg       0.87      0.87      0.87     10000
weighted avg       0.88      0.88      0.88     10000

Confusion Matrix:
[[ 914    1    7    4    6    9   16    5    8   10]
 [   0 1084    9    8    2    9    5    3   14    1]
 [  13   11  887   29   15    6    9   24   30    8]
 [   7    8   34  861    8   40    3    6   17   26]
 [   8    4   11    6  858    5   18   10   20   42]
 [  15    8    5   39    6  740   25    6   32   16]
 [  21    5   11    9   23   15  846    3   20    5]
 [   2    7   21   24   12    5    3  925    9   20]
 [   8    9   33   34   21   32   14   12  785   26]
 [  14    5   10   22   45   10    6   20   23  854]]

Execution Time: 26.345789432525635 seconds

True Positive Rate for each class:
[0.93265306 0.95506608 0.85949612 0.85247525 0.87372709 0.82959641
 0.88308977 0.89980545 0.80595483 0.84638256]
```

Random Forest

```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
execution_time = train_base_model(rf_classifier)
evaluate_base_model("Random Forest", rf_classifier, execution_time)
```

```
Random Forest accuracy: 0.9704

Classification Report
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.99      0.99      0.99      1135
           2       0.96      0.97      0.97      1032
           3       0.96      0.96      0.96      1010
           4       0.97      0.97      0.97       982
           5       0.98      0.96      0.97       892
           6       0.98      0.98      0.98       958
           7       0.97      0.96      0.97      1028
           8       0.96      0.95      0.96       974
           9       0.96      0.95      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

Confusion Matrix:
[[ 971    0    0    0    0    2    3    1    3    0]
 [   0 1127    2    2    0    1    2    0    1    0]
 [   6    0 1002    5    3    0    3    8    5    0]
 [   1    0    9  972    0    9    0    9    8    2]
 [   1    0    0    0  955    0    5    1    4   16]
 [   5    1    1    9    2  860    5    2    5    2]
 [   7    3    0    0    3    3  937    0    5    0]
 [   1    4   20    2    0    0    0  989    2   10]
 [   4    0    6    7    5    5    5    4  930    8]
 [   7    6    2   12   12    1    0    4    4  961]]

Execution Time: 38.150798082351685 seconds

True Positive Rate for each class:
[0.99081633 0.99295154 0.97093023 0.96237624 0.97250509 0.96412556
 0.97807933 0.96206226 0.95482546 0.95242815]
```

Support Vector Machine (SVM)

```
svm_classifier = SVC(kernel='linear')
execution_time = train_base_model(svm_classifier)
evaluate_base_model("SVM", svm_classifier, execution_time)
```

```
SVM accuracy: 0.9404

Classification Report
              precision    recall  f1-score   support

           0       0.95      0.98      0.96       980
           1       0.97      0.99      0.98      1135
           2       0.93      0.94      0.93      1032
           3       0.91      0.94      0.92      1010
           4       0.94      0.96      0.95       982
           5       0.91      0.90      0.91       892
           6       0.96      0.95      0.95       958
           7       0.95      0.93      0.94      1028
           8       0.94      0.90      0.92       974
           9       0.95      0.91      0.93      1009

    accuracy                           0.94     10000
   macro avg       0.94      0.94      0.94     10000
weighted avg       0.94      0.94      0.94     10000

Confusion Matrix:
[[ 957    0    4    1    1    6    9    1    0    1]
 [   0 1122    3    2    0    1    2    1    4    0]
 [   8    6  967   11    3    3    7    8   17    2]
 [   4    3   16  947    1   16    0    9   12    2]
 [   1    1   10    1  942    2    4    2    3   16]
 [  10    4    3   36    6  803   13    1   14    2]
 [   9    2   13    1    5   16  910    1    1    0]
 [   1    8   21   10    8    1    0  957    3   19]
 [   8    4    6   25    7   26    6    7  877    8]
 [   7    7    2   11   33    4    0   18    5  922]]

Execution Time: 246.75818538665771 seconds

True Positive Rate for each class:
[0.97653061 0.98854626 0.9370155  0.93762376 0.9592668  0.90022422
 0.94989562 0.93093385 0.90041068 0.91377602]
```

K-Nearest Neighbors (KNN)

```
knn_classifier = KNeighborsClassifier(n_neighbors=5)
execution_time = train_base_model(knn_classifier)
evaluate_base_model("KNN", knn_classifier, execution_time)
```

```
KNN accuracy: 0.9688

Classification Report
              precision    recall  f1-score   support

           0       0.96      0.99      0.98       980
           1       0.95      1.00      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.96      0.97      0.97      1010
           4       0.98      0.96      0.97       982
           5       0.97      0.97      0.97       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.99      0.94      0.96       974
           9       0.96      0.95      0.95      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

Confusion Matrix:
[[ 974    1    1    0    0    1    2    1    0    0]
 [   0 1133    2    0    0    0    0    0    0    0]
 [  11    8  991    2    1    0    1   15    3    0]
 [   0    3    3  976    1   13    1    6    3    4]
 [   3    7    0    0  944    0    4    2    1   21]
 [   5    0    0   12    2  862    4    1    2    4]
 [   5    3    0    0    3    2  945    0    0    0]
 [   0   22    4    0    3    0    0  988    0   11]
 [   8    3    5   13    6   12    5    5  913    4]
 [   5    7    3    9    7    3    1   10    2  962]]

Execution Time: 0.02273726463317871 seconds

True Positive Rate for each class:
[0.99387755 0.99823789 0.96027132 0.96633663 0.96130346 0.96636771
 0.98643006 0.96108949 0.93737166 0.95341923]
```

Multilayer Perceptron (MLP)

```
#Pre-Processing Dataset
(train_x, train_y), (test_x, test_y) = tf.keras.datasets.mnist.load_data()

train_x = tf.reshape(train_x, (train_x.shape[0], 28, 28, 1))
test_x = tf.reshape(test_x, (test_x.shape[0], 28, 28, 1))
```

```
train_y = tf.one_hot(train_y, 10)
test_y = tf.one_hot(test_y, 10)


model_mlp = models.Sequential([
    layers.Flatten(input_shape=(28, 28, 1)),
    layers.Dense(512, activation='relu'),
    layers.Dense(256, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])
model_mlp.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

execution_time = train_nn_model(model_mlp)
evaluate_nn_model("MLP", model_mlp, execution_time)
```

```
375/375 [==============================] - 5s 6ms/step - loss: 1.8498 - accuracy:
Epoch 2/10
375/375 [==============================] - 1s 4ms/step - loss: 0.2224 - accuracy:
Epoch 3/10
375/375 [==============================] - 1s 4ms/step - loss: 0.1432 - accuracy:
Epoch 4/10
375/375 [==============================] - 2s 4ms/step - loss: 0.0985 - accuracy:
Epoch 5/10
375/375 [==============================] - 1s 4ms/step - loss: 0.0803 - accuracy:
Epoch 6/10
375/375 [==============================] - 1s 4ms/step - loss: 0.0800 - accuracy:
Epoch 7/10
375/375 [==============================] - 1s 4ms/step - loss: 0.0903 - accuracy:
Epoch 8/10
375/375 [==============================] - 2s 4ms/step - loss: 0.0758 - accuracy:
Epoch 9/10
375/375 [==============================] - 2s 5ms/step - loss: 0.0812 - accuracy:
Epoch 10/10
375/375 [==============================] - 1s 4ms/step - loss: 0.0699 - accuracy:
313/313 [==============================] - 1s 1ms/step
MLP accuracy: 0.9615

Classification Report
              precision    recall  f1-score   support

           0       0.99      0.97      0.98       980
           1       0.99      0.97      0.98      1135
           2       0.97      0.95      0.96      1032
           3       0.95      0.95      0.95      1010
           4       0.97      0.97      0.97       982
           5       0.94      0.98      0.96       892
           6       0.97      0.98      0.97       958
           7       0.98      0.91      0.94      1028
           8       0.91      0.97      0.94       974
           9       0.93      0.97      0.95      1009

    accuracy                           0.96     10000
   macro avg       0.96      0.96      0.96     10000
weighted avg       0.96      0.96      0.96     10000

Confusion Matrix:
[[ 955    0    6    0    1    0    7    1    6    4]
```

```
 [   1    1  976   19    2    0    3    7   22    1]
 [   0    0    6  963    0   25    0    3   10    3]
 [   2    1    2    0  955    0    4    2    2   14]
 [   3    0    0    5    1  871    3    0    7    2]
 [   3    2    3    0    4    4  937    0    5    0]
 [   0    4    8   19    6    1    1  932   17   40]
 [   0    0    2    4    5   10    2    1  946    4]
 [   3    2    0    2   10    9    2    2    5  974]]

Execution Time: 18.79372215270996 seconds

True Positive Rate for each class:
[0.9744898  0.97444934 0.94573643 0.95346535 0.97250509 0.9764574
 0.97807933 0.90661479 0.97125257 0.96531219]
```

Convolutional Neural Network (CNN)

```
#Pre-Processing Dataset
(train_x, train_y), (test_x, test_y) = tf.keras.datasets.mnist.load_data()

train_x = tf.reshape(train_x, (train_x.shape[0], 28, 28, 1))
test_x = tf.reshape(test_x, (test_x.shape[0], 28, 28, 1))
train_y = tf.one_hot(train_y, 10)
test_y = tf.one_hot(test_y, 10)
```

```
model_cnn = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])
model_cnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

execution_time = train_nn_model(model_cnn)
evaluate_nn_model("CNN", model_cnn, execution_time)
```

```
375/375 [==============================] - 5s 6ms/step - loss: 0.9109 - accuracy:
Epoch 2/10
375/375 [==============================] - 2s 5ms/step - loss: 0.0580 - accuracy:
Epoch 3/10
375/375 [==============================] - 3s 7ms/step - loss: 0.0381 - accuracy:
Epoch 4/10
375/375 [==============================] - 2s 5ms/step - loss: 0.0259 - accuracy:
Epoch 5/10
375/375 [==============================] - 2s 5ms/step - loss: 0.0213 - accuracy:
Epoch 6/10
375/375 [==============================] - 2s 5ms/step - loss: 0.0176 - accuracy:
Epoch 7/10
375/375 [==============================] - 2s 5ms/step - loss: 0.0165 - accuracy:
```

```
375/375 [==============================] - 2s 5ms/step - loss: 0.0125 - accuracy: (
Epoch 9/10
375/375 [==============================] - 2s 6ms/step - loss: 0.0156 - accuracy: (
Epoch 10/10
375/375 [==============================] - 2s 5ms/step - loss: 0.0137 - accuracy: (
313/313 [==============================] - 1s 2ms/step
CNN accuracy: 0.9861

Classification Report
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.98      1.00      0.99      1135
           2       0.98      0.99      0.98      1032
           3       0.98      0.99      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.99      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       0.98      0.98      0.98      1028
           8       0.99      0.98      0.99       974
           9       0.99      0.97      0.98      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000

Confusion Matrix:
[[ 970    1    1    0    2    0    3    2    1    0]
 [   0 1130    2    1    0    1    0    1    0    0]
 [   1    7 1021    1    0    0    0    2    0    0]
 [   0    0    3 1002    0    3    0    2    0    0]
 [   0    2    0    0  968    0    2    1    2    7]
 [   1    0    0    8    0  881    1    1    0    0]
 [   2    2    0    0    2    2  946    0    4    0]
 [   0    8   12    1    1    0    0 1006    0    0]
 [   3    0    1    3    2    0    1    3  957    4]
 [   1    1    3    3    6    5    0    7    3  980]]

Execution Time: 22.54736614227295 seconds

True Positive Rate for each class:
[0.98979592 0.99559471 0.98934109 0.99207921 0.98574338 0.98766816
 0.9874739  0.97859922 0.9825462  0.97125867]
```

Transfer Learning

Pre-Processing Dataset

```
(train_x, train_y), (test_x, test_y) = tf.keras.datasets.mnist.load_data()

#Limit the number of training and testing examples to not consume all RAM
max_examples = 1000
train_indices = random.sample(range(len(train_x)), max_examples)
```

```
test_indices = random.sample(range(len(test_x)), max_examples)

train_x = train_x[train_indices]
train_y = train_y[train_indices]
test_x = test_x[test_indices]
test_y = test_y[test_indices]

#Expand grayscale images to have three channels (RGB models)
train_x = tf.expand_dims(train_x, axis=-1)
train_x = tf.concat([train_x, train_x, train_x], axis=-1)
test_x = tf.expand_dims(test_x, axis=-1)
test_x = tf.concat([test_x, test_x, test_x], axis=-1)

#One-Hot Encoder
train_y = tf.one_hot(train_y, 10)
test_y = tf.one_hot(test_y, 10)

#Resize data (models input shape like (224x224x3))
train_x = tf.image.resize(train_x, (224, 224))
test_x = tf.image.resize(test_x, (224, 224))
```

VGG16

```
vgg16_base = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
for layer in vgg16_base.layers:
    layer.trainable = False

modelVGG16 = models.Sequential()
modelVGG16.add(vgg16_base)
modelVGG16.add(layers.GlobalAveragePooling2D())
modelVGG16.add(layers.Dense(512, activation='relu'))
modelVGG16.add(layers.Dropout(0.5))
modelVGG16.add(layers.Dense(10, activation='softmax'))

modelVGG16.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']

execution_time = train_nn_model(modelVGG16)
evaluate_nn_model("VGG16", modelVGG16, execution_time)
```

```
7/7 [==============================] - 33s 3s/step - loss: 4.7327 - accuracy: 0.18
Epoch 2/10
7/7 [==============================] - 4s 602ms/step - loss: 1.8168 - accuracy: 0.
Epoch 3/10
7/7 [==============================] - 4s 604ms/step - loss: 0.9717 - accuracy: 0.
Epoch 4/10
7/7 [==============================] - 4s 527ms/step - loss: 0.6204 - accuracy: 0.
Epoch 5/10
7/7 [==============================] - 4s 519ms/step - loss: 0.3971 - accuracy: 0.
Epoch 6/10
7/7 [==============================] - 4s 606ms/step - loss: 0.3242 - accuracy: 0.
Epoch 7/10
7/7 [==============================] - 4s 612ms/step - loss: 0.2878 - accuracy: 0.
Epoch 8/10
Epoch 9/10
7/7 [==============================] - 4s 608ms/step - loss: 0.2227 - accuracy: 0.9
Epoch 10/10
7/7 [==============================] - 4s 614ms/step - loss: 0.1858 - accuracy: 0.9
32/32 [==============================] - 5s 157ms/step
VGG16 accuracy: 0.9360

Classification Report
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       104
           1       0.98      0.99      0.99       107
           2       0.90      0.92      0.91       106
           3       0.92      0.90      0.91        99
           4       0.94      0.92      0.93       101
           5       0.90      0.89      0.89        80
           6       0.94      0.96      0.95       106
           7       0.93      0.87      0.90       101
           8       0.99      0.94      0.96        98
           9       0.91      0.97      0.94        98

    accuracy                           0.94      1000
   macro avg       0.94      0.93      0.93      1000
weighted avg       0.94      0.94      0.94      1000

Confusion Matrix:
[[103   0   0   0   0   0   1   0   0   0]
 [  0 106   0   0   1   0   0   0   0   0]
 [  1   0  97   1   0   2   2   2   0   1]
 [  0   0   5  89   0   3   1   0   1   0]
 [  0   0   1   0  93   0   1   2   0   4]
 [  0   0   1   4   0  71   1   3   0   0]
 [  3   0   0   0   0   1 102   0   0   0]
 [  0   2   2   1   4   1   0  88   0   3]
 [  0   0   2   2   0   0   1   0  92   1]
 [  1   0   0   0   1   1   0   0   0  95]]

Execution Time: 69.48288297653198 seconds

True Positive Rate for each class:
[0.99038462 0.99065421 0.91509434 0.8989899  0.92079208 0.8875
 0.96226415 0.87128713 0.93877551 0.96938776]
```

VGG19

```
vgg19_base = VGG19(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
for layer in vgg19_base.layers:
    layer.trainable = False

modelVGG19 = models.Sequential()
modelVGG19.add(vgg19_base)
modelVGG19.add(layers.GlobalAveragePooling2D())
modelVGG19.add(layers.Dense(512, activation='relu'))
modelVGG19.add(layers.Dropout(0.5))
modelVGG19.add(layers.Dense(10, activation='softmax'))
```

```
modelVGG19.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']

execution_time = train_nn_model(modelVGG19)
evaluate_nn_model("VGG19", modelVGG19, execution_time)
```

```
7/7 [==============================] - 6s 710ms/step - loss: 4.9009 - accuracy: 0.
Epoch 2/10
7/7 [==============================] - 5s 701ms/step - loss: 1.7190 - accuracy: 0.
Epoch 3/10
7/7 [==============================] - 4s 645ms/step - loss: 0.8661 - accuracy: 0.7
Epoch 4/10
7/7 [==============================] - 4s 646ms/step - loss: 0.5618 - accuracy: 0.8
Epoch 5/10
7/7 [==============================] - 4s 642ms/step - loss: 0.3957 - accuracy: 0.8
Epoch 6/10
7/7 [==============================] - 5s 705ms/step - loss: 0.3528 - accuracy: 0.8
Epoch 7/10
7/7 [==============================] - 4s 643ms/step - loss: 0.2258 - accuracy: 0.9
Epoch 8/10
7/7 [==============================] - 4s 644ms/step - loss: 0.1874 - accuracy: 0.9
Epoch 9/10
7/7 [==============================] - 5s 714ms/step - loss: 0.1512 - accuracy: 0.9
Epoch 10/10
7/7 [==============================] - 5s 706ms/step - loss: 0.1668 - accuracy: 0.9
32/32 [==============================] - 4s 138ms/step
VGG19 accuracy: 0.9340

Classification Report
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       104
           1       0.98      0.97      0.98       107
           2       0.87      0.91      0.89       106
           3       0.92      0.90      0.91        99
           4       0.93      0.94      0.94       101
           5       0.95      0.86      0.90        80
           6       0.93      0.96      0.94       106
           7       0.89      0.92      0.90       101
           8       0.96      0.93      0.94        98
           9       0.97      0.95      0.96        98

    accuracy                           0.93      1000
   macro avg       0.93      0.93      0.93      1000
weighted avg       0.93      0.93      0.93      1000

Confusion Matrix:
[[102   0   0   0   0   0   1   0   1   0]
 [  0 104   0   0   1   0   0   2   0   0]
 [  1   0  96   2   1   2   2   2   0   0]
 [  0   0   4  89   0   2   2   0   2   0]
 [  0   1   0   0  95   0   1   3   0   1]
 [  1   0   1   4   0  69   1   3   1   0]
 [  1   1   0   0   2   0 102   0   0   0]
 [  0   0   3   1   2   0   0  93   0   2]
 [  1   0   4   0   0   0   1   1  91   0]
 [  0   0   2   1   1   0   0   1   0  93]]

Execution Time: 47.81148028373718 seconds
```

```
True Positive Rate for each class:
[0.98076923 0.97196262 0.90566038 0.8989899  0.94059406 0.8625
 0.96226415 0.92079208 0.92857143 0.94897959]
```

ResNet50

```
resnet50_base = ResNet50(include_top=False, weights='imagenet', input_shape=(224, 224, 3))
for layer in resnet50_base.layers:
    layer.trainable = False

modelResNet50 = models.Sequential()
modelResNet50.add(resnet50_base)
modelResNet50.add(layers.GlobalAveragePooling2D())
modelResNet50.add(layers.Dense(512, activation='relu'))
modelResNet50.add(layers.Dropout(0.5))
modelResNet50.add(layers.Dense(10, activation='softmax'))

modelResNet50.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac

execution_time = train_nn_model(modelResNet50)
evaluate_nn_model("ResNet50", modelResNet50, execution_time)
```

```
7/7 [==============================] - 17s 2s/step - loss: 2.1973 - accuracy: 0.31
Epoch 2/10
7/7 [==============================] - 3s 415ms/step - loss: 0.9769 - accuracy: 0.
Epoch 3/10
7/7 [==============================] - 3s 419ms/step - loss: 0.7166 - accuracy: 0.7
Epoch 4/10
7/7 [==============================] - 3s 421ms/step - loss: 0.5181 - accuracy: 0.8
Epoch 5/10
7/7 [==============================] - 3s 417ms/step - loss: 0.4301 - accuracy: 0.8
Epoch 6/10
7/7 [==============================] - 3s 413ms/step - loss: 0.3541 - accuracy: 0.8
Epoch 7/10
7/7 [==============================] - 3s 419ms/step - loss: 0.3169 - accuracy: 0.9
Epoch 8/10
7/7 [==============================] - 3s 420ms/step - loss: 0.2548 - accuracy: 0.9
Epoch 9/10
7/7 [==============================] - 3s 414ms/step - loss: 0.2389 - accuracy: 0.9
Epoch 10/10
7/7 [==============================] - 3s 416ms/step - loss: 0.2322 - accuracy: 0.9
32/32 [==============================] - 5s 113ms/step
ResNet50 accuracy: 0.9030

Classification Report
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       104
           1       0.99      0.98      0.99       107
           2       0.86      0.90      0.88       106
           3       0.65      0.94      0.77        99
           4       0.91      0.93      0.92       101
           5       1.00      0.39      0.56        80
           6       0.94      0.97      0.96       106
           7       0.92      0.90      0.91       101
           9       0.98      0.95      0.96        98

    accuracy                           0.90      1000
   macro avg       0.92      0.89      0.89      1000
weighted avg       0.92      0.90      0.90      1000

Confusion Matrix:
[[104   0   0   0   0   0   0   0   0   0]
 [  0 105   0   0   1   0   1   0   0   0]
 [  1   0  95   8   0   0   0   2   0   0]
 [  0   0   2  93   0   0   0   0   3   1]
 [  0   0   2   0  94   0   0   4   0   1]
 [  0   0   3  40   0  31   4   2   0   0]
 [  0   0   2   0   0   0 103   0   1   0]
 [  0   1   4   1   4   0   0  91   0   0]
 [  0   0   0   0   3   0   1   0  94   0]
 [  0   0   2   2   1   0   0   0   0  93]]

Execution Time: 44.414286375045776 seconds

True Positive Rate for each class:
[1.         0.98130841 0.89622642 0.93939394 0.93069307 0.3875
 0.97169811 0.9009901  0.95918367 0.94897959]
```

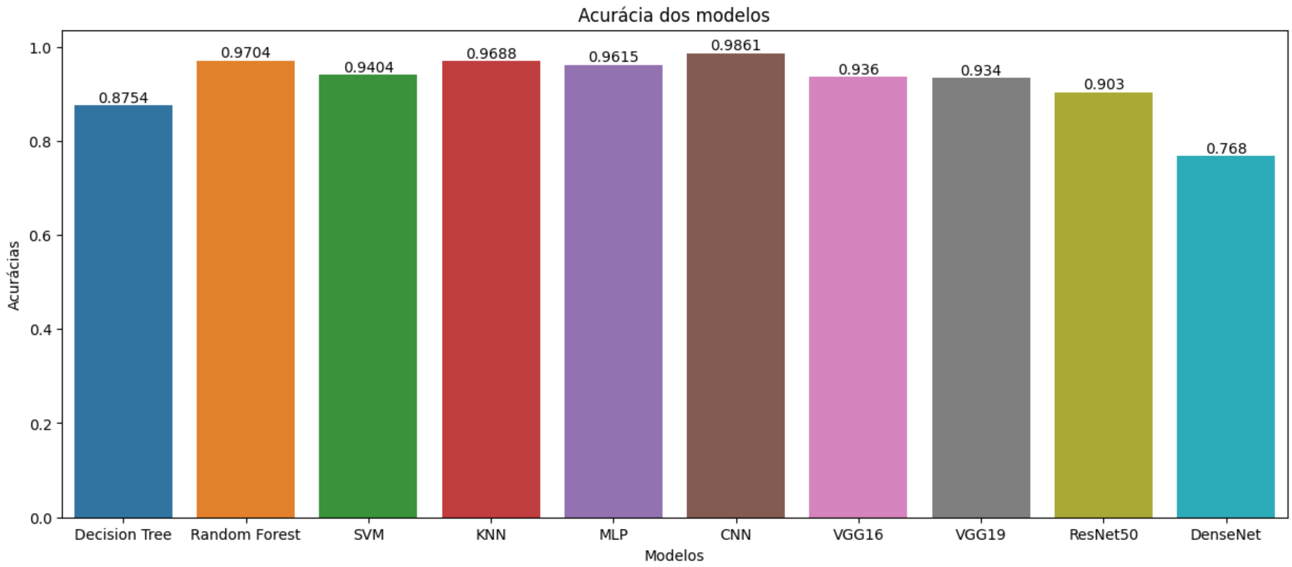
DenseNet121

```
denseNet121_base = DenseNet121(weights='imagenet', include_top=False, input_shape=(224, 22
for layer in denseNet121_base.layers:
    layer.trainable = False

modelDenseNet = models.Sequential()
modelDenseNet.add(denseNet121_base)
modelDenseNet.add(layers.GlobalAveragePooling2D())
modelDenseNet.add(layers.Dense(512, activation='relu'))
modelDenseNet.add(layers.Dropout(0.5))
modelDenseNet.add(layers.Dense(10, activation='softmax'))

modelDenseNet.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac

execution_time = train_nn_model(modelDenseNet)
evaluate_nn_model("DenseNet", modelDenseNet, execution_time)
```

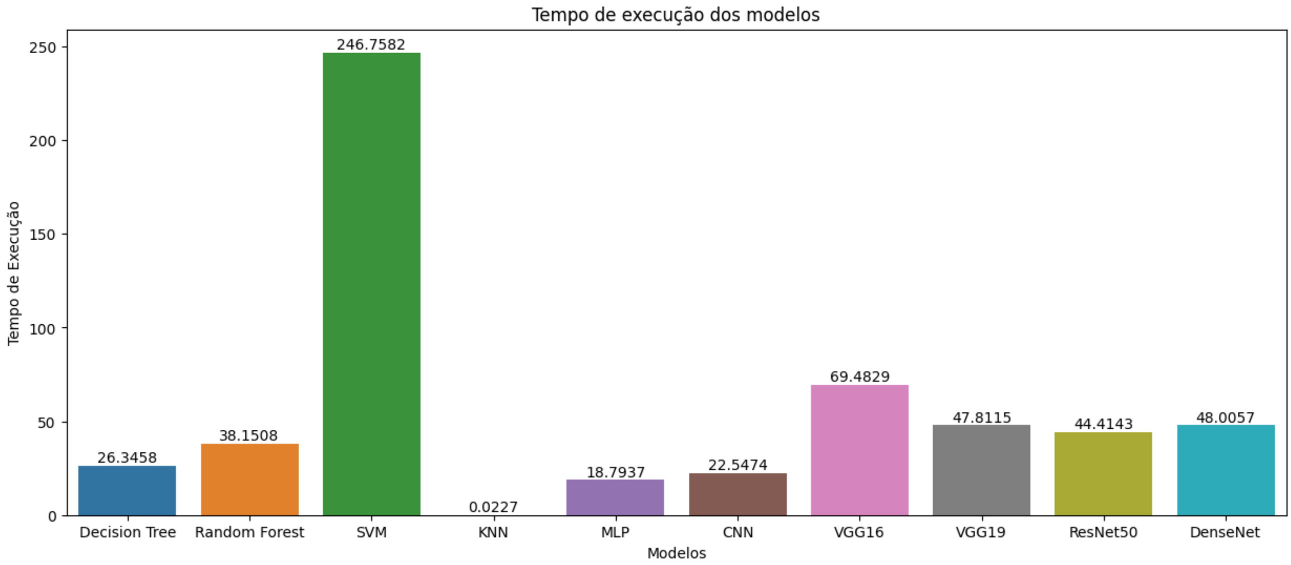
```
7/7 [==============================] - 23s 2s/step - loss: 9.0623 - accuracy: 0.16
Epoch 2/10
7/7 [==============================] - 3s 389ms/step - loss: 3.9799 - accuracy: 0.
Epoch 3/10
7/7 [==============================] - 3s 398ms/step - loss: 2.2184 - accuracy: 0.
Epoch 4/10
7/7 [==============================] - 3s 425ms/step - loss: 1.5045 - accuracy: 0.
Epoch 5/10
7/7 [==============================] - 3s 393ms/step - loss: 1.2106 - accuracy: 0.
Epoch 6/10
7/7 [==============================] - 3s 401ms/step - loss: 1.1906 - accuracy: 0.
Epoch 7/10
Epoch 8/10
7/7 [==============================] - 3s 409ms/step - loss: 0.9648 - accuracy: 0.
Epoch 9/10
7/7 [==============================] - 3s 396ms/step - loss: 0.8946 - accuracy: 0.
Epoch 10/10
7/7 [==============================] - 3s 388ms/step - loss: 0.8479 - accuracy: 0.7
32/32 [==============================] - 6s 134ms/step
DenseNet accuracy: 0.7680

Classification Report
              precision    recall  f1-score   support

           0       0.86      0.91      0.89       104
           1       0.92      0.93      0.93       107
           2       0.75      0.71      0.73       106
           3       0.54      0.80      0.65        99
           4       0.85      0.74      0.79       101
           5       0.78      0.36      0.50        80
           6       0.86      0.89      0.87       106
           7       0.76      0.85      0.80       101
           8       0.63      0.74      0.69        98
           9       0.84      0.63      0.72        98

    accuracy                           0.77      1000
   macro avg       0.78      0.76      0.76      1000
weighted avg       0.78      0.77      0.76      1000

Confusion Matrix:
[[ 95   0   1   0   0   0   3   4   1   0]
 [  1 100   0   0   4   0   0   1   1   0]
 [  1   0  75  11   1   0   2   3  11   2]
 [  0   0   7  79   0   7   1   0   5   0]
 [  0   0   1   1  75   0   6   9   0   9]
 [  2   0   6  34   0  29   1   5   3   0]
 [  5   3   1   0   0   1  94   1   1   0]
 [  2   3   1   5   3   0   0  86   1   0]
 [  2   0   5  14   2   0   1   0  73   1]
 [  2   3   3   1   3   0   1   4  19  62]]

Execution Time: 48.00566053390503 seconds

True Positive Rate for each class:
[0.91346154 0.93457944 0.70754717 0.7979798  0.74257426 0.3625
 0.88679245 0.85148515 0.74489796 0.63265306]
```

Comparative Analysis

```
plt.figure(figsize = (15, 6))
plt.title("Acurácia dos modelos")
plt.xlabel('Modelos')
plt.ylabel('Acurácias')
sns.barplot(x=models_list, y=accuracies_list)
for index, value in enumerate(accuracies_list):
    plt.text(index, value, str(value), ha='center', va='bottom')

plt.show()
```



```
plt.figure(figsize = (15, 6))
plt.title("Tempo de execução dos modelos")
plt.xlabel('Modelos')
plt.ylabel('Tempo de Execução')
sns.barplot(x=models_list, y=execution_times_list)
for index, value in enumerate(execution_times_list):
    plt.text(index, value, "{:.4f}".format(value), ha='center', va='bottom')

plt.show()
```



```
TPR_min_rounded = [round(val, 4) for val in TPR_min]
data = {
    'Model': models_list,
    'Worst Digit': most_difficult_digits_list,
    'True Positive Rate': TPR_min_rounded
}
df = pd.DataFrame(data)
print(tabulate(df, headers='keys', tablefmt='pretty', showindex=False))
print(tabulate(df, headers='keys', tablefmt='pretty', showindex=False))
```

```
+---------------+-------------+--------------------+
|     Model     | Worst Digit | True Positive Rate |
+---------------+-------------+--------------------+
| Decision Tree |      8      |       0.806        |
| Random Forest |      9      |       0.9524       |
|      SVM      |      5      |       0.9002       |
|      KNN      |      8      |       0.9374       |
|      MLP      |      7      |       0.9066       |
|      CNN      |      9      |       0.9713       |
|     VGG16     |      7      |       0.8713       |
|     VGG19     |      5      |       0.8625       |
|   ResNet50    |      5      |       0.3875       |
|   DenseNet    |      5      |       0.3625       |
+---------------+-------------+--------------------+
```

Ao observar a acurácia (proporção de exemplos corretamente classificados em relação ao total de exemplos no conjunto de dados de teste), distoa o desempenho usando Decision Tree e Transfer Learning com DenseNet e ResNet50, sendo os únicos abaixo de 92% de acerto.

Já ao observar o tempo de execução, destaca-se os modelos SVM, VGG16, DenseNet novamente, VGG19 e ResNet50 (os dois últimos sendo técnicas de Transfer Learning), ultrapassando os 42 segundos. O SVM teve o tempo muito maior devido a sua complexidade, o processo de otimização é computacionalmente intensivo, especialmente com um grande conjunto de dados. Além de precisar de ter treinamento iterativo, ou seja, precisa de várias iterações para encontrar o hiperplano de melhor separação, o que demanda mais tempo que algoritmos mais simples.

Positivamente, temos quatro modelos com acurácia maior que 95% (CNN, Random Forest, KNN, MLP). Em relação ao tempo de execução, três modelos ficaram abaixo de 30 segundos (KNN, CNN, Decision Tree). O KNN tem um tempo de execução muito menor devido a sua simplicidade, o algoritmo não envolve cálculos complexos ou treinamento de parâmetros. A classificação é feita utilizando os exemplos mais próximos, sem etapas iterativas para ajuste de parâmetros, não exigindo várias etapas para convergir como as redes neurais.

Em conclusão, ao considerar acurácia e tempo de execução, o melhor modelo que temos é o KNN, um algoritmo simples que chegou a 96.88% de acurácia em menos de um segundo. Outra excelente opção seria CNN, com 98.61% de acurácia em menos de 30 segundos. Analisando o pior dígito, através da taxa de verdadeiro positivo, seria o 8 (93.74%) e o 9 (97.13%) respectivamente. Aparentemente, o 5 é mais difícil de identificar, principalmente ao usar a estratégia de Transfer Learning.

✓ 0s completed at 2:40 AM ● ×