DTSA 5511 - Introduction to Deep Learning

University of Colorado, Boulder

Master of Science in Data Science

Digit Recognizer |Kaggle |Final Project

1 Introduction

Description

MNIST (”Modified National Institute of Standards and Technology”) is the de facto “hello world” dataset of computer vision. Since its release in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike.
In this competition, our goal is to correctly identify digits from a dataset of tens of thousands of handwritten images.

Evaluation

- The goal in this competition is to take an image of a handwritten single digit, and determine what that digit is. For every in the test set, you should predict the correct label.
- This competition is evaluated on the categorization accuracy of your predictions (the percentage of images you get correct).
- Submission File Format The file should contain a header and have the following format:

  ImageId,Label
  1,0
  2,0
  3,0
  etc.

2 Exploratory Data Analysis(EDA)

2.1 Data Loading and Libraries Import

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

train=pd.read_csv("/kaggle/input/digit-recognizer/train.csv")
train.head()
```

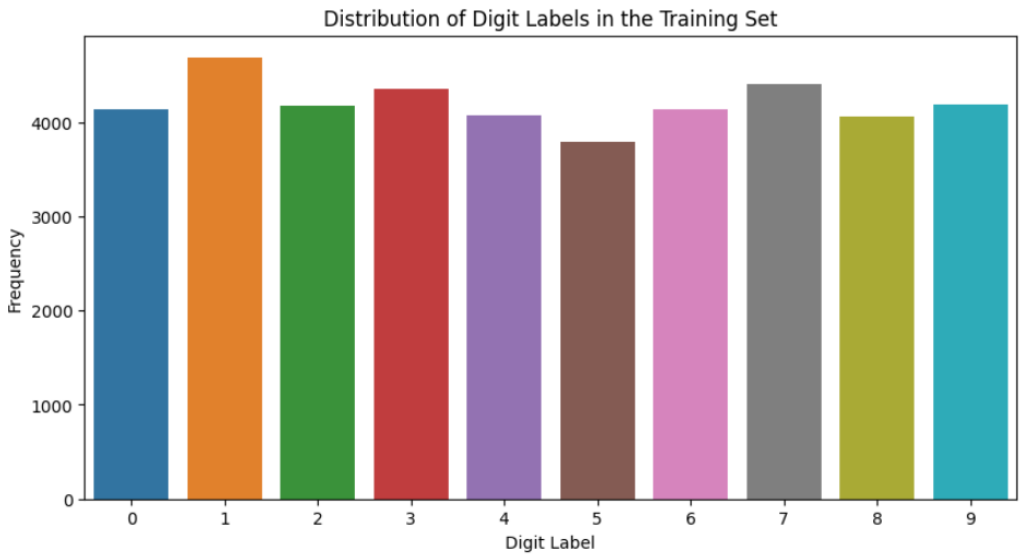
```
test=pd.read_csv("/kaggle/input/digit-recognizer/test.csv")
test.head()
```

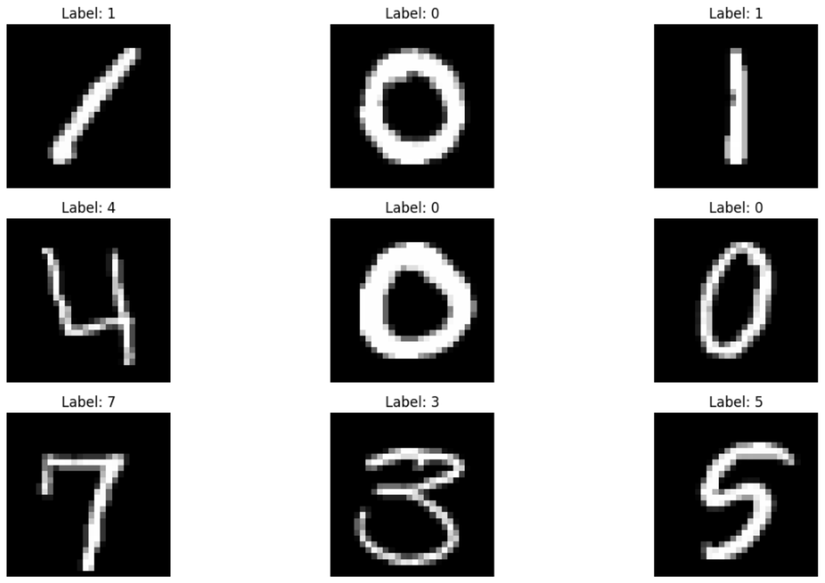
2.2 EDA and Preprocessing

We can see that there are not any missing values in the training and test set. We also try to see the distribution of the labels in the training set which seems pretty balanced across the categories. At the end, we show a bunch of training images just for eyeballing if we detect anything wrong :

```
# Check unique values in the label column
sorted_unique_labels = sorted(train['label'].unique())
print("Sorted unique labels:", sorted_unique_labels)


# Visualization of the Distribution of Digit Labels
plt.figure(figsize=(10, 5))
sns.countplot(x='label', data=train)
plt.title('Distribution of Digit Labels in the Training Set')
plt.xlabel('Digit Label')
plt.ylabel('Frequency')
plt.show()


# Check for Missing Values
if train.isnull().any().any():
    print("There are missing values in the training set.")
else:
    print("No missing values in the training set.")

if test.isnull().any().any():
    print("There are missing values in the test set.")
else:
    print("No missing values in the test set.")

# Visualization of a Subset of Digit Images
num_images = 15
plt.figure(figsize=(12, 12))

for i in range(num_images):
    plt.subplot(5, 3, i + 1)
    img = train.iloc[i, 1:].values.reshape(28, 28)
    plt.imshow(img, cmap='grey')
```

```
        plt.title("Label: {}".format(train.iloc[i, 0]))
        plt.axis('off')

    plt.tight_layout()
    plt.show()
```





3 Model Selection and Training

There are some preprocessing steps that I included in this chapter as they make more sense next to our model selection. We need to normalize our input pixels(by dividing them to 255) and also one hot encode the labels using to_categorical from TensorFlow:

```
import tensorflow as tf
import tensorflow.keras as keras
from tensorflow.keras.utils import to_categorical

labels = train.label

train = train.drop(['label'],axis = 1)

train_norm =train.values.reshape(train.shape[0],28,28,1)/255.0
test_norm = test.values.reshape(test.shape[0],28,28,1)/255.0

labels_encoded = to_categorical(labels,num_classes=10)
```

In this project,we will use Keras sequential models. We start off by a simple model with one convolution layer, one flattener layer and one dense layers for predicting our 10 labels. We'll add some extra Maxpooling,convolution and dense layers for our more complex models. We'll see their performance in the next section. For completeness, below is the most complex model we used:

```
model = keras.models.Sequential([

  keras.layers.Conv2D(64, (3,3), activation='relu', input_shape
     =(28, 28, 1)),
  keras.layers.MaxPooling2D(2, 2),
  keras.layers.Conv2D(128, (3,3), activation='relu'),
  keras.layers.MaxPooling2D(2,2),

  keras.layers.Flatten(),
  keras.layers.Dense(256, activation='relu'),
  keras.layers.Dense(10, activation='softmax')
])

model.summary()

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

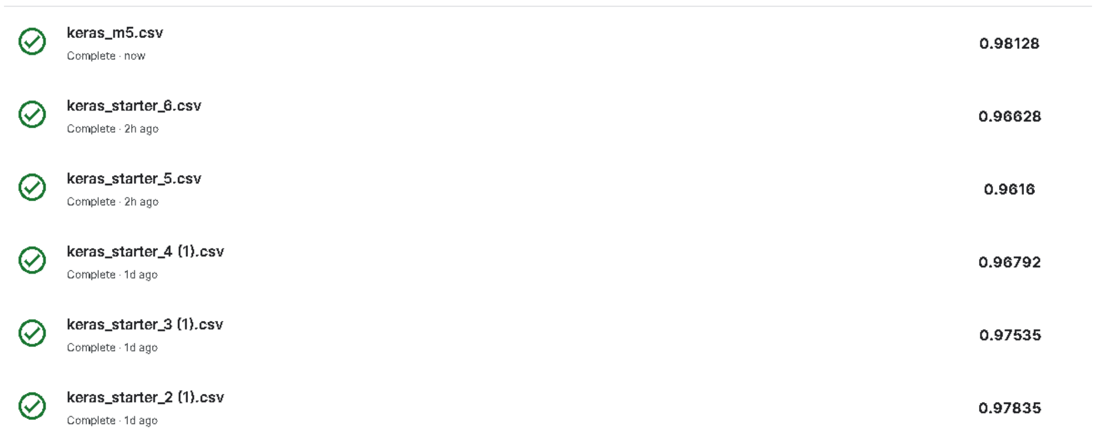
```
model.fit(train_norm,labels_encoded,epochs=5,validation_split
   =0.1)
```

4 Predictions & Submission

After finishing up with the training of our model, we can make predictions on the normalized test set and we create the submission file with asked format.

```
predictions=model.predict(test_norm)
predicted_labels=np.argmax(predictions,axis=1)
submission=pd.DataFrame({"ImageId":range(1,len(predicted_labels
   )+1),"Label":predicted_labels})
submission.to_csv("keras_starter.csv",index=False)
```

5 Conclusion and Future Work



keras_m5.csv
Complete · now
0.98128

keras_starter_6.csv
Complete · 2h ago
0.96628

keras_starter_5.csv
Complete · 2h ago
0.9616

keras_starter_4 (1).csv
Complete · 1d ago
0.96792

keras_starter_3 (1).csv
Complete · 1d ago
0.97535

keras_starter_2 (1).csv
Complete · 1d ago
0.97835

It seems all our models are performing well even though the simpler ones(Just one Conv2D, one Flatten and one Dense layer of softmax) performed a bit worse than the complex ones(by 2%). The reason may be due to the balanced nature of our training set and also the usefulness of CNNS for basic image recognition tasks such as digit recognizers. It can be shown that there are improvements that can make the score almost perfect(score of 1). One may wish to explore by experimenting with different structures and also different validation methods(such as k_fold) to further improve the performance of the model.

6 References

[1] https://www.kaggle.com/competitions/digit-recognizer
[2] https://www.kaggle.com/code/samarjeet09/digit-recognizer-cnn-basic
[3] https://www.kaggle.com/code/poonaml/deep-neural-network-keras-way