Kaggle competition: Digit Recognizer

Learn computer vision fundamentals with the famous MNIST data

Index

- Abstract
- Introduction
- Data Description
- Data Preparation
- Method – CNN
- Result
- Reference

Abstract

- This competition on Kaggle using MNIST dataset are for the beginners to better understand the basic of machine learning and application of neural networks by focusing on the classic topic "Handwritten Digit Recognition".
- MNIST ("Modified National Institute of Standards and Technology") is a database which has about 60,000 of digital handwriting image data, released in 1999 by Yann LeCun, Corinna Cortes and Christopher J.C. Burges.
- It was often used as a training data in classification models or other image processing algorithms.

Introduction

- In this competition, the goal is to correctly identify digits from a dataset of tens of thousands of handwritten images. We've curated a set of tutorial-style kernels which cover everything from regression to neural networks. We encourage you to experiment with different algorithms to learn first-hand what works well and how techniques compare.

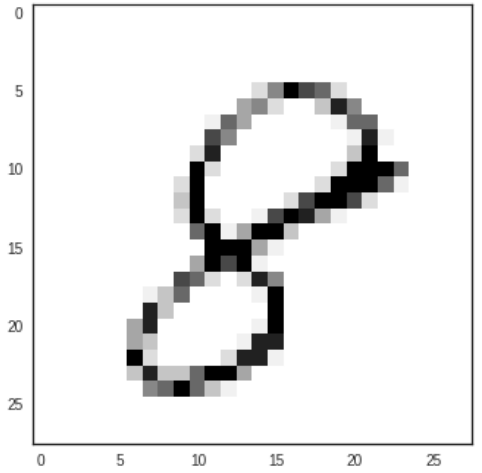
Source: https://www.kaggle.com/c/digit-recognizer/data

Data Description

- The data files contain gray-scale images of hand-drawn digits, from zero through nine.
  Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

- Visually, if we omit the "pixel" prefix, the pixels make up the image like this:

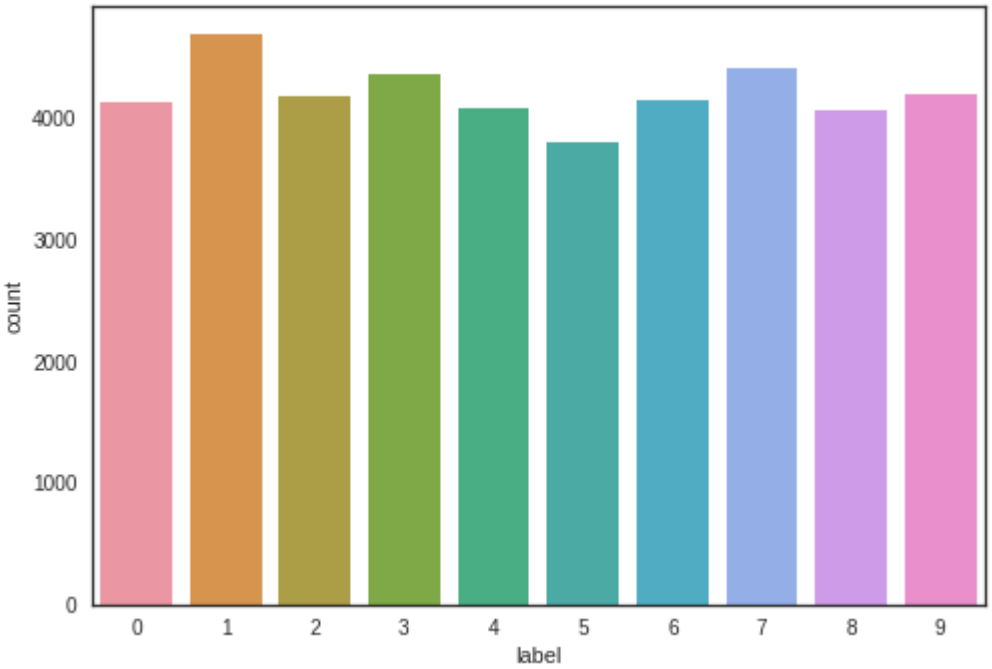
- 000 001 002 003 ... 026 027
- 028 029 030 031 ... 054 055
- 056 057 058 059 ... 082 083
- | | | | ... | |
- 728 729 730 731 ... 754 755
- 756 757 758 759 ... 782 783



Source: https://www.kaggle.com/c/digit-recognizer/data

Data Preparation

- Data distribution
  checking the counts of each digits.




```
1 4684
7 4401
3 4351
9 4188
2 4177
6 4137
0 4132
4 4072
8 4063
5 3795
Name: label, dtype: int64
```

Data Preparation

- Data cleaning

  Check for null and missing values

```
# Check the data
X_train.isnull().any().describe()
```

```
Count     784
Unique    1
Top       False
Freq      784
dtype:    object
```

```
test.isnull().any().describe()
```

```
Count     784
Unique    1
Top       False
Freq      784
dtype:    object
```

Data Preparation

- Normalization

  Normalize the dataset from [0..255] to [0..1]

```
# Normalize the data
X_train = X_train / 255.0
test = test / 255.0
```

Data Preparation

- Reshape

  reshaping all data to 28x28x1 3D matrices due to Keras requires an extra dimension in the end which correspond to channels.

```
# Reshape image in 3 dimensions (height = 28px, width = 28px , canal = 1)
X_train = X_train.values.reshape(-1,28,28,1)
test = test.values.reshape(-1,28,28,1)
```

Data Preparation

- One-hot encoding

  Labels in MNIST dataset are 0 to 9, need to encode these numbers in to one hot vectors.

```
# Encode labels to one hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0])
Y_train = to_categorical(Y_train, num_classes = 10)
```

Data Preparation

- Split training and valdiation set

  split the training set into two parts:

  validation set (10%)

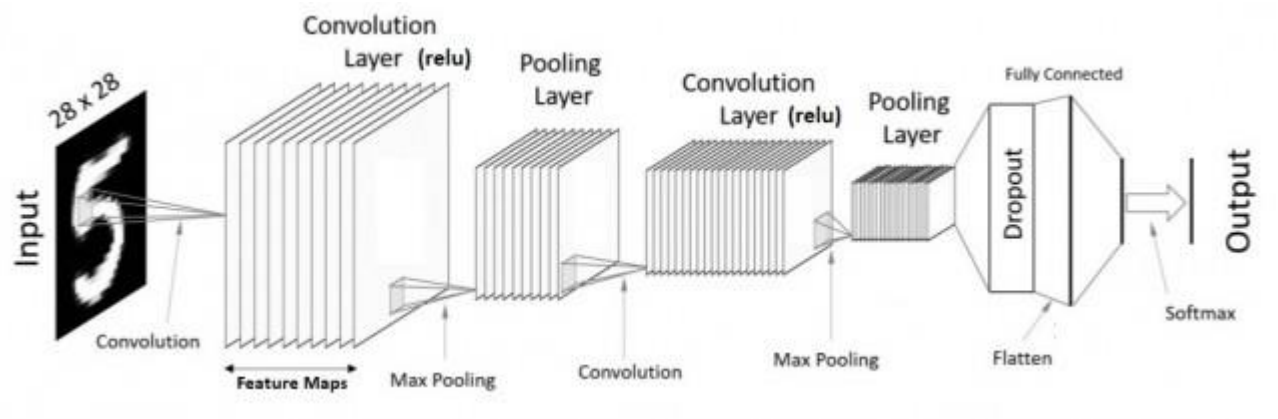
  training set (90%)

```
# Set the random seed
random_seed = 2

# Split the train and the validation set for the fitting
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size = 0.1, random_state=random_seed)
```

Method – CNN

- CNN is used for image classification, object detection
- Using the Keras Sequential API here so we can just add one layer at a time, starting from the input.



Source: https://www.kaggle.com/leo223/keras-model-99-485-leaderboard-cnn

Method – CNN

- Setting up an optimizer and annealer to minimise the loss.

```
# Define the optimizer
optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)

# Compile the model
model.compile(optimizer = optimizer , loss = "categorical_crossentropy",
metrics=["accuracy"])

# Set a learning rate annealer
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

Method – CNN

- Epoches and batch size

  Each epoch takes about 10 mins to run on a windows laptop with a batch size of 86

```
epochs = 30
batch_size = 86
```

Method – CNN

- Data augmentation
  - Without data augmentation, our accuracy is around 98.114%
  - With the data augmentation we reached the accuracy of 99.428%

Method – CNN

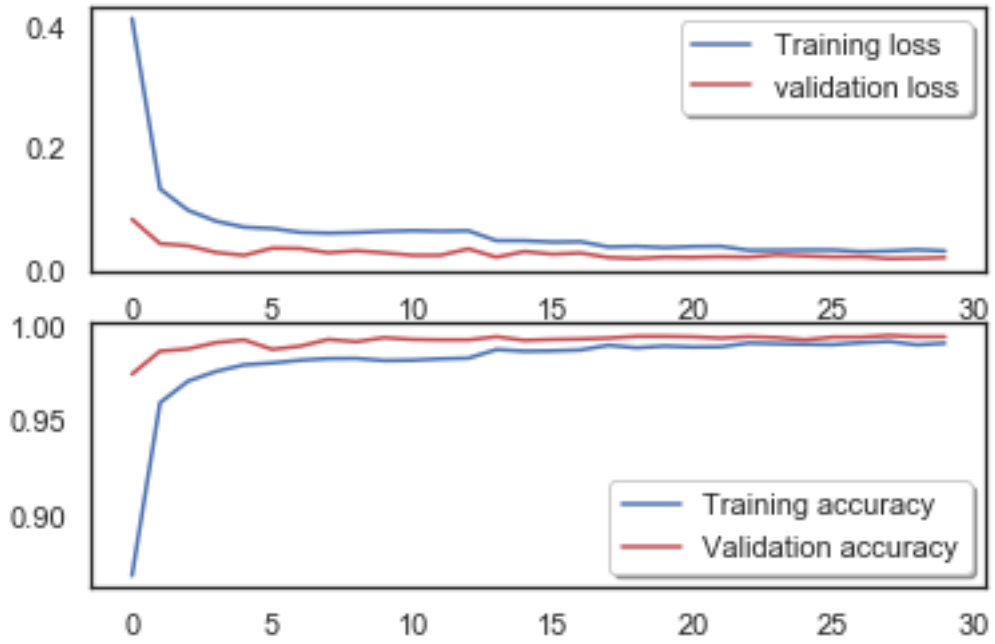
- Data augmentation

```
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images

datagen.fit(X_train)
```

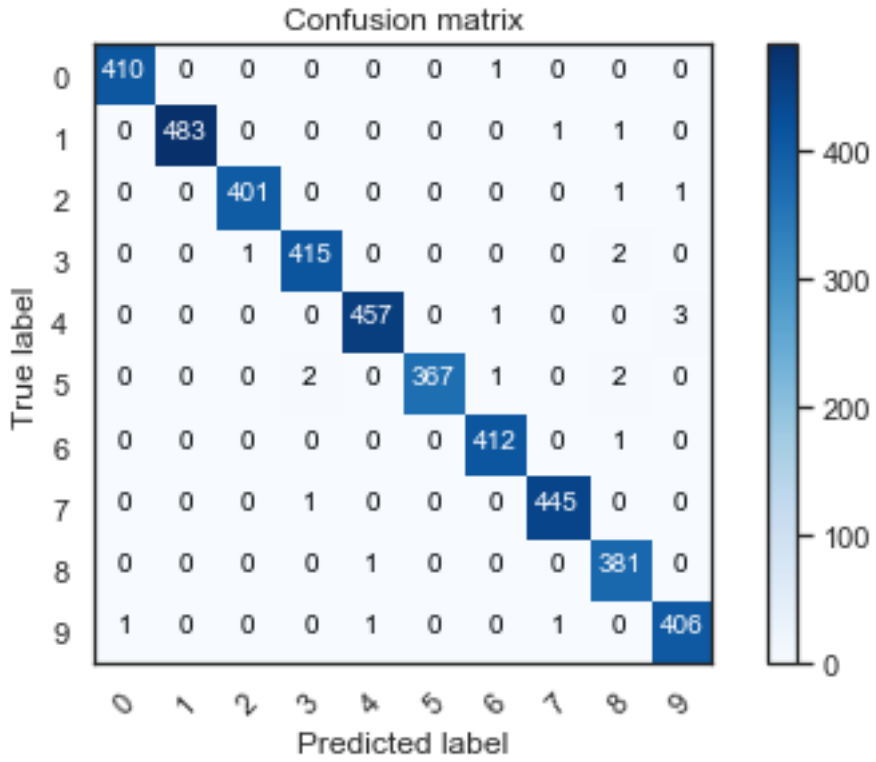
Result

- Train/Validation lost and accuracy



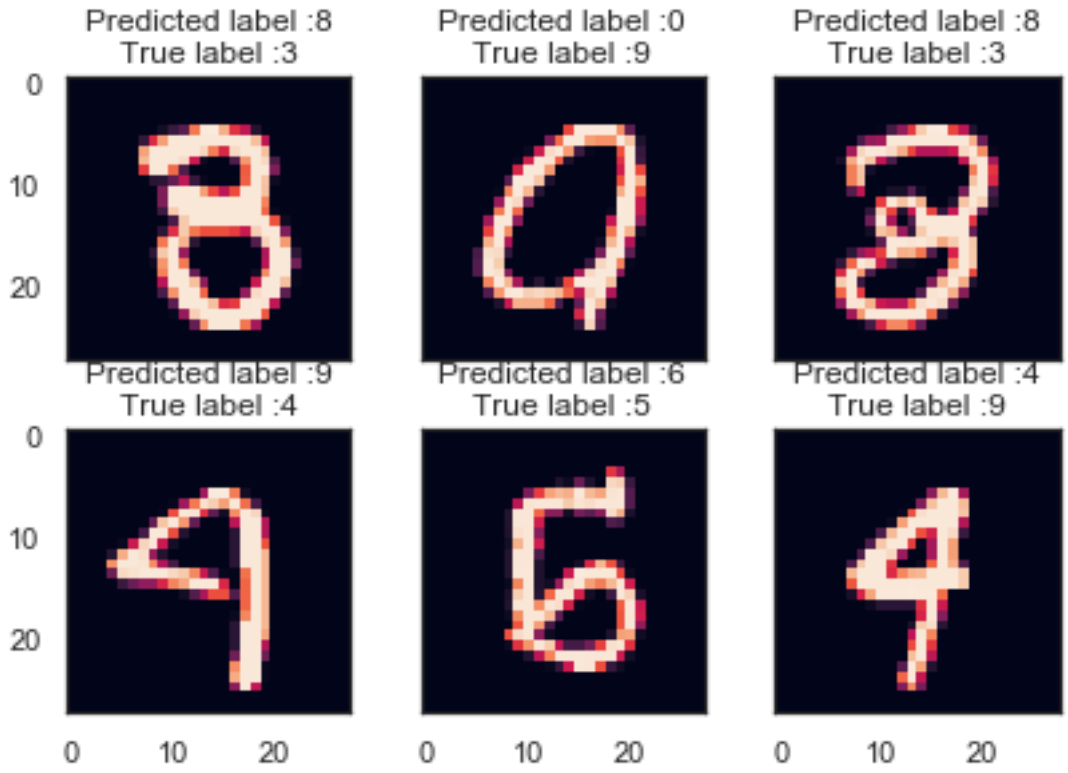
Result

- Confusion matrix



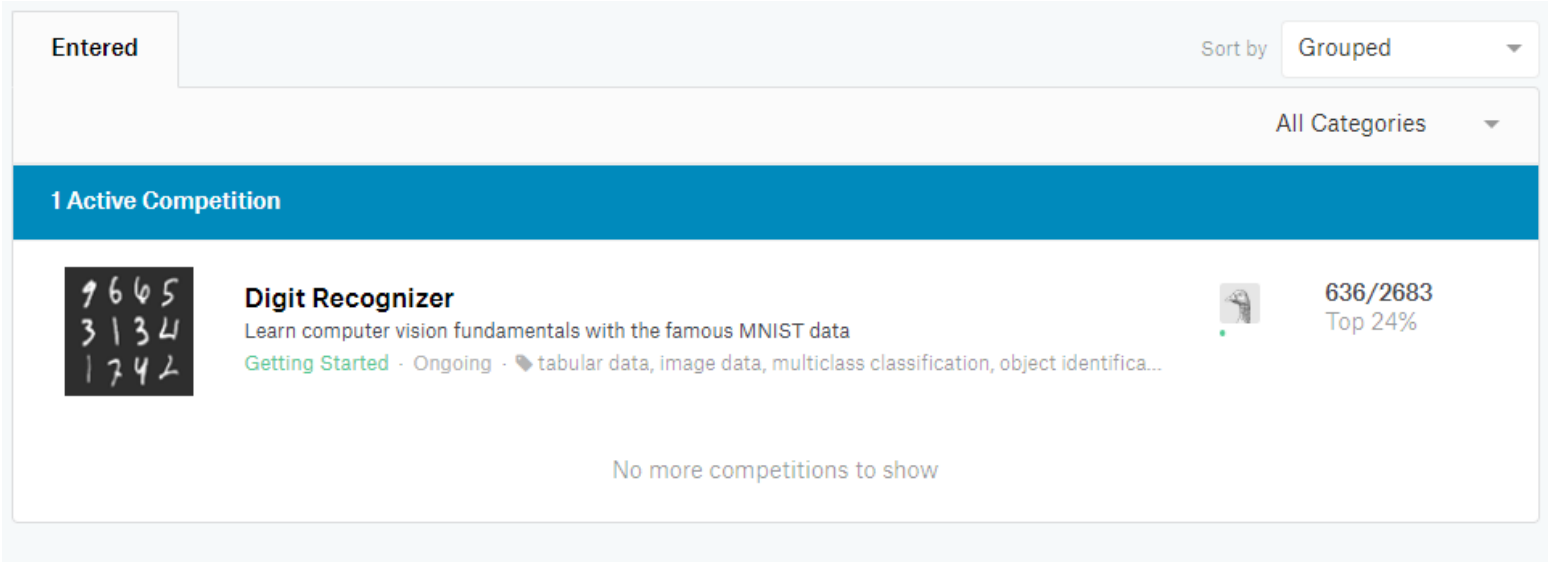
Result

- Top errors



Result

- Competition Ranking
  - 636/2683 Top 24%



Reference

- https://www.kaggle.com/c/digit-recognizer
- https://www.kaggle.com/yassineghouzam/introduction-to-cnn-keras-0-997-top-6
- https://www.kaggle.com/leo223/keras-model-99-485-leaderboard-cnn/
- https://en.wikipedia.org/wiki/MNIST_database
- http://yann.lecun.com/exdb/mnist/