mnist

April 18, 2020

1 Importing Libraries

[1]:

```python
import numpy as np
import keras
from keras.utils import np_utils
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
import tensorflow.keras.layers as Layers
import tensorflow.keras.models as Models
import sklearn.utils as shuffle
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from keras.callbacks import LearningRateScheduler
```

```
Using TensorFlow backend.
```

Loading data

[2]:

```python
train = pd.read_csv("../input/digit-recognizer/train.csv")
test = pd.read_csv("../input/digit-recognizer/test.csv")
```

[3]:

```python
train.head()
```

[3]:

```
   label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0      1       0       0       0       0       0       0       0       0
1      0       0       0       0       0       0       0       0       0
2      1       0       0       0       0       0       0       0       0
3      4       0       0       0       0       0       0       0       0
4      0       0       0       0       0       0       0       0       0

   pixel8  …  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       0  …         0         0         0         0         0         0
1       0  …         0         0         0         0         0         0
2       0  …         0         0         0         0         0         0
3       0  …         0         0         0         0         0         0
4       0  …         0         0         0         0         0         0
```

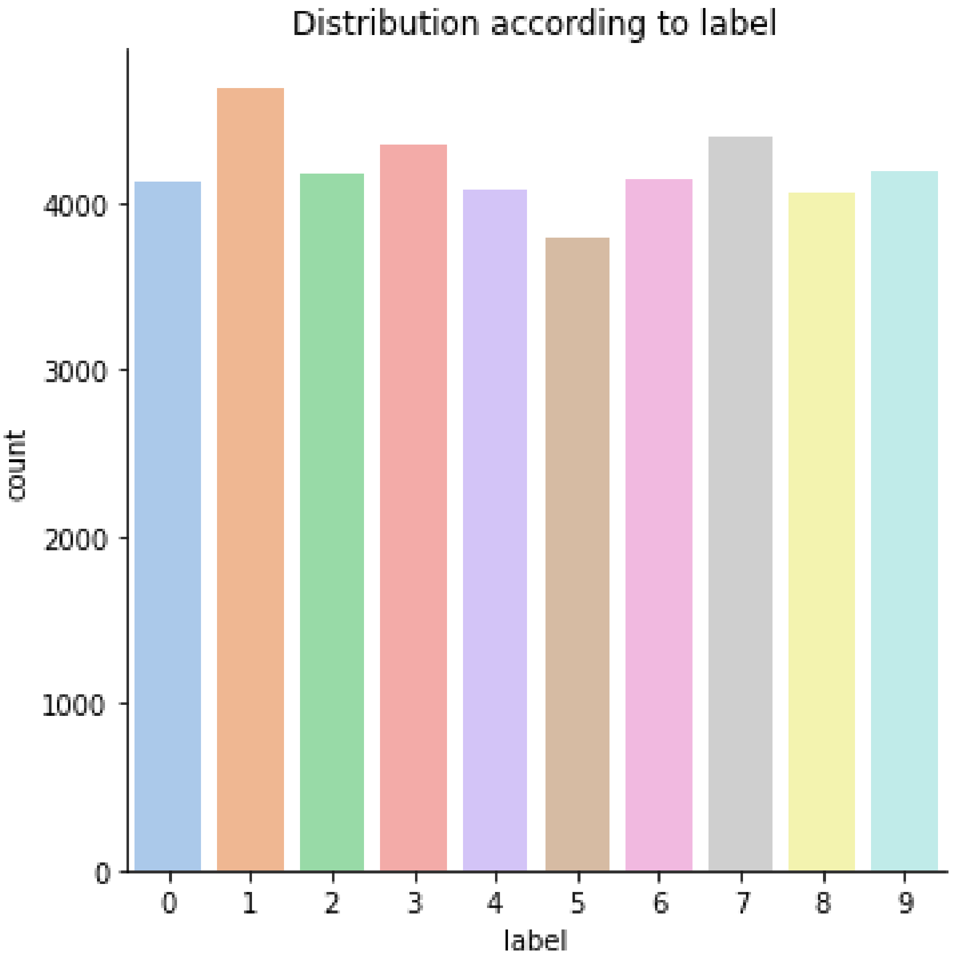
```
   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 785 columns]
```

Visualizing predicting label

```
[4]: plt.figure(figsize = (15,15))
     sns.catplot(x = 'label', kind = 'count' ,data = train, palette = "pastel")
     plt.title("Distribution according to label")
     plt.show()
```

```
<Figure size 1080x1080 with 0 Axes>
```



[5]:
```
y = train["label"]
train.drop(["label"], axis = 1, inplace = True)
```

[6]:
```
train.head()
```

[6]:
```
   pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
0       0       0       0       0       0       0       0       0       0
1       0       0       0       0       0       0       0       0       0
2       0       0       0       0       0       0       0       0       0
3       0       0       0       0       0       0       0       0       0
4       0       0       0       0       0       0       0       0       0

   pixel9  …  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       0  …         0         0         0         0         0         0
1       0  …         0         0         0         0         0         0
2       0  …         0         0         0         0         0         0
3       0  …         0         0         0         0         0         0
4       0  …         0         0         0         0         0         0

   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 784 columns]
```

[7]:
```
y.head()
```

[7]:
```
0    1
1    0
2    1
3    4
4    0
Name: label, dtype: int64
```

[8]:
```
y.unique()
```

[8]:
```
array([1, 0, 4, 7, 3, 5, 8, 9, 2, 6])
```

[9]:
```
y = np_utils.to_categorical(y, 10)
```

[10]:
```
y.shape
```

[10]:
```
(42000, 10)
```

[11]:
```
train.shape
```

[11]:
```
(42000, 784)
```

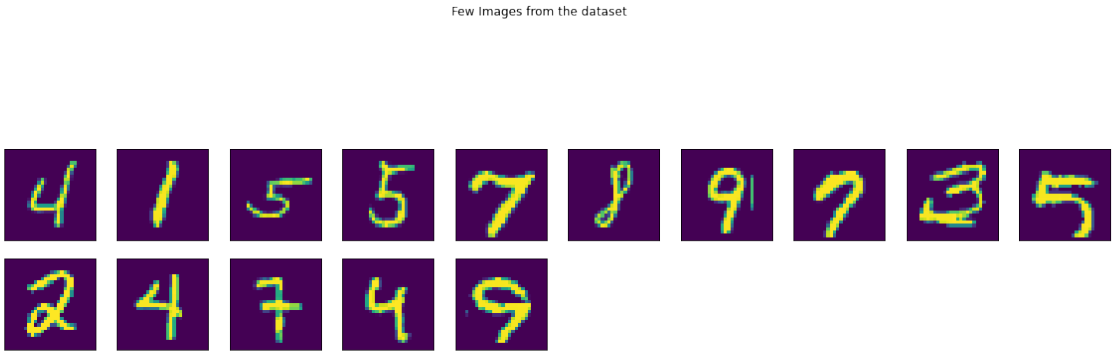
2 Visualizing Images

[12]:
```
def image_show(train):
    fig = plt.figure(figsize = (20,20))
    fig.suptitle("Few Images from the dataset")
    for i in range(15):
        index = np.random.randint(train.shape[0])
        plt.subplot(10,10,i+1)
        plt.imshow(train[index][:,:, 0])
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
    plt.show()
```

[13]:
```
train = train.values.reshape(-1,28,28,1)
```

[14]:
```
test = test.values.reshape(-1,28,28,1)
```

[15]:
```
image_show(train)
```



[16]:
```
train = train / 255
```

[17]:
```
test = test / 255
```

Image Augmentation

[18]:
```
image_generator= ImageDataGenerator(rotation_range = 10,zoom_range = 0.10,width_shift_range=0.1,height_shift_range=0.1)
```

3 CNN Network

[19]:

```
model = [0] * 10
for i in range(10):
    model[i] = Models.Sequential()
    model[i].add(Layers.Conv2D(64, kernel_size = 3, activation='relu',␣
 ↪input_shape = (28, 28, 1)))
    model[i].add(Layers.BatchNormalization())
    model[i].add(Layers.Conv2D(64, kernel_size = 3, activation='relu'))
    model[i].add(Layers.BatchNormalization())
    model[i].add(Layers.Conv2D(64, kernel_size = 5, strides=2, padding='same',␣
 ↪activation='relu'))
    model[i].add(Layers.BatchNormalization())
    model[i].add(Layers.Dropout(0.4))
    model[i].add(Layers.Conv2D(128, kernel_size = 3, activation='relu'))
    model[i].add(Layers.BatchNormalization())
    model[i].add(Layers.Conv2D(128, kernel_size = 3, activation='relu'))
    model[i].add(Layers.BatchNormalization())
    model[i].add(Layers.Conv2D(128, kernel_size = 5, strides=2, padding='same',␣
 ↪activation='relu'))
    model[i].add(Layers.BatchNormalization())
    model[i].add(Layers.Dropout(0.4))
    model[i].add(Layers.Conv2D(256, kernel_size = 4, activation='relu'))
    model[i].add(Layers.BatchNormalization())
    model[i].add(Layers.Flatten())
    model[i].add(Layers.Dense(512, activation = 'relu'))
    model[i].add(Layers.Dropout(0.4))
    model[i].add(Layers.Dense(10, activation='softmax'))
    model[i].compile(optimizer="adam", loss="categorical_crossentropy",␣
 ↪metrics=["accuracy"])
```

[20]:

```
call_back =  LearningRateScheduler(lambda x: 1e-3 * 0.95 ** x)
history = [0] * 10
epochs = 30
for i in range(10):
    train_x, val_x, train_y, val_y = train_test_split(train, y, test_size = 0.1)
    history[i] = model[i].fit_generator(image_generator.flow(train_x,train_y,␣
 ↪batch_size= 64),
        epochs = 30, steps_per_epoch = (train_x.shape[0]// 64) ,
        validation_data = (val_x,val_y), callbacks=[call_back], verbose= 0)
    print("CNN {0:d}: Epochs={1:d}, Train accuracy={2:.5f}, Validation␣
 ↪accuracy={3:.5f}".format(
        i+1,epochs,max(history[i].history['accuracy']),max(history[i].
 ↪history['val_accuracy']) ))
```

```
CNN 1: Epochs=30, Train accuracy=0.99740, Validation accuracy=0.99571
CNN 2: Epochs=30, Train accuracy=0.99663, Validation accuracy=0.99571
CNN 3: Epochs=30, Train accuracy=0.99653, Validation accuracy=0.99548
CNN 4: Epochs=30, Train accuracy=0.99656, Validation accuracy=0.99524
CNN 5: Epochs=30, Train accuracy=0.99640, Validation accuracy=0.99738
CNN 6: Epochs=30, Train accuracy=0.99671, Validation accuracy=0.99429
CNN 7: Epochs=30, Train accuracy=0.99658, Validation accuracy=0.99500
CNN 8: Epochs=30, Train accuracy=0.99703, Validation accuracy=0.99548
CNN 9: Epochs=30, Train accuracy=0.99637, Validation accuracy=0.99643
CNN 10: Epochs=30, Train accuracy=0.99650, Validation accuracy=0.99667
```

4 Prediction

[21]:

```
results = np.zeros((test.shape[0],10))
for i in range(10):
    results = results + model[i].predict(test)
results = np.argmax(results,axis = 1)
results = pd.Series(results,name="Label")
submission = pd.concat([pd.Series(range(1,28001),name =␣
 ↪"ImageId"),results],axis = 1)
submission.to_csv("MNIST.csv",index=False)
```

If you like please upvote