1. Introduction

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
%matplotlib inline


from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import itertools

from keras.utils.np_utils import to_categorical # convert to one-hot-encodin
g
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.optimizers import RMSprop
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.layers.normalization import BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
from sklearn.model_selection import train_test_split

sns.set(style='white', context='notebook', palette='deep')
```

2. Data preparation

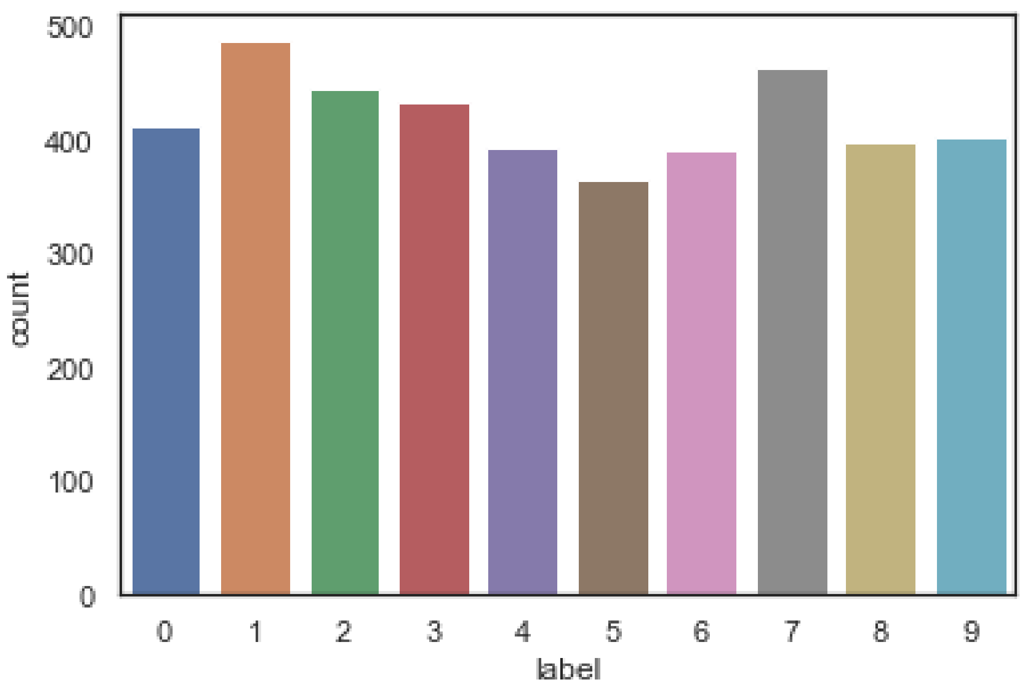
2.1 Load data

```
# >>>>>填写<<<< 利用pandas的load_csv函数，读取我们的train 和 test数
据集合 变量已经给出 >>>>>填写<<<< ######
train = pd.read_csv("subset_train.csv")
test = pd.read_csv("Small_test.csv")
#####train validation test(完全独立的，与训练过程无关的)
# >>>>>填写<<<< 利用pandas的header选择，将label列传递给Y_train
 >>>>>填写<<<<
```

```
Y_train = train["label"]
Y_test = test['label']
# 因为train.csv 中，第一列label 在上述代码已经传递给Y_label，这里对于x_train 我们不需要训练集的第一列 #####
X_train = train.drop(labels = ["label"],axis = 1)
X_test = test.drop(labels = ["label"],axis = 1)
# 释放内存


g = sns.countplot(Y_train)

Y_train.value_counts()
Y_train
Y_test.value_counts()
```

Out[142]:

```
3    227
7    220
2    220
1    218
6    214
9    209
4    209
5    201
0    199
8    183
Name: label, dtype: int64
```



We have similar counts for the 10 digits.

2.2 Check for null and missing values

```
# 检查训练数据是否有空值
X_train.isnull().any().describe()
```

Out[143]:

```
count      784
unique       1
top      False
freq       784
dtype: object
```

```
# >>>>填写<<<< 检查训练数据是否有空值 >>>>填写<<<< ###
X_test.isnull().any().describe()
```

Out[144]:

```
count      784
unique       1
top      False
freq       784
dtype: object
```

I check for corrupted images (missing values inside).

There is no missing values in the train and test dataset. So we can safely go ahead.

2.3 Normalization

We perform a grayscale normalization to reduce the effect of illumination's differences.

Moreover the CNN converg faster on [0..1] data than on [0..255]. 标准化，将灰度值 0-255 映射到 0 - 1 区间

```
# Normalize the data
X_train = X_train / 255.0
###### >>>填写<<< 标准化测试集合 #######
X_test = X_test / 255.0
```

2.3 Reshape

```
# >>>>填写<<<<< 利用 reshape 函数， 将X_train 变换成 (height = 28px, width = 28px , canal = 1)>>>>填写<<<<< ######
X_train = X_train.values.reshape(-1,28,28,1)
X_test = X_test.values.reshape(-1,28,28,1)
```

Train and test images (28px x 28px) has been stock into pandas.Dataframe as 1D vectors of 784 values. We reshape all data to 28x28x1 3D matrices.

Keras requires an extra dimension in the end which correspond to channels. MNIST images are gray scaled so it use only one channel. For RGB images, there is 3 channels, we would have reshaped 784px vectors to 28x28x3 3D matrices.

2.5 Label encoding

```
# 利用0 1编码 将0-9数字标签编码成10维向量 (ex : 9 -> [0,0,0,0,0,0,0,0,0,1])
##
Y_train = to_categorical(Y_train, num_classes = 10)
Y_test = to_categorical(Y_test, num_classes = 10)
## one-hot encoding
```

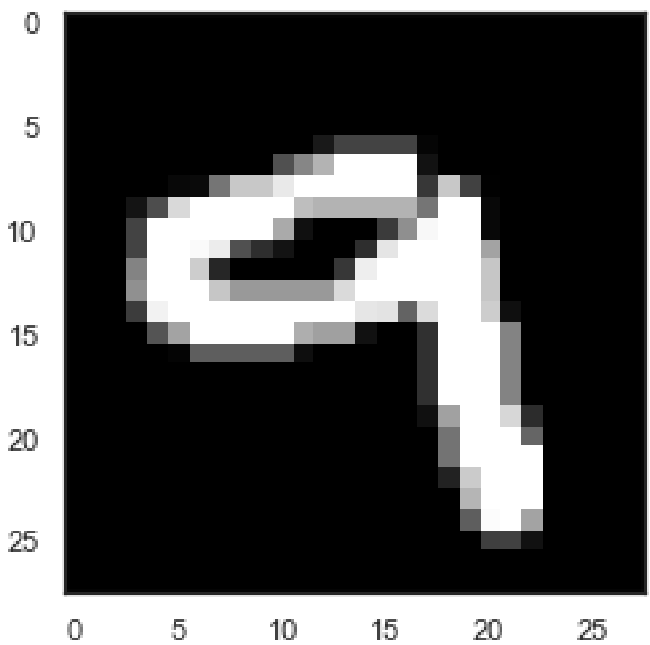
Labels are 10 digits numbers from 0 to 9. We need to encode these lables to one hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0]).

2.6 Split training and valdiation set

```
# Set the random seed
random_seed = 2
# 将训练集合按照9:1 分成训练集合 和验证集合 validation 10折交叉验证 10-fold validation  ####
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size = 0.1, random_state=random_seed)
```

We can get a better sense for one of these examples by visualising the image and looking at the label.

```
# Some examples #x-train 里面第一个sample 的 0:最大 0:最大 0 [:,:,0]
g = plt.imshow(X_train[0][:,:,0],cmap='gray') #plt 为什么把灰度可以生
```



3. CNN

3.1 Define the model

Type *Markdown* and LaTeX: $\alpha 2$α2

```
#####   填写 batch_size  epoch  请根据traindata  总量填写合适的值  ####
#####  我们的分配数量num_classes,提示  我们的任务是手写体0-9 的识别  ##
#####

batch_size = 40
num_classes = 10
epochs = 20

input_shape = (28,28,1)

#构建CNN  模型  这里我们利用Sequential  序列累加  ######
model = Sequential()
##  第一个  卷积层  32 个kernel  kernel 大小3*3  输出的激活函数relu  kernel
利用  He-正态分布  生成   ####
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',kernel_initializer='he_normal',input_shape=input_shape))

###   请自行构建第二个卷积层，此时kernel 的初始尝试用全零初始/全1 初始
/正态初始
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',kernel_initializer='he_normal'))
###  构建一个最大池化层
model.add(MaxPool2D((2, 2),strides=2))

model.add(Dropout(0.20))
###   在下述卷积层内  构建一个padding,  在之后构建一个kernel  size  =  2  *2
的池化层
model.add(Conv2D(64, (3, 3), activation='relu',padding='same',kernel_initializer='he_normal'))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Dropout(0.25))
model.add(Flatten())
###   构建一个全联接  其中包含128 个神经元  并使用relu 激活函数


model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
```

```
model.add(Dropout(0.25))

###  >>> 填写<<<<构建一个全联接，该全联接需要用特定的激活函数和适当的神经元个数 来实现我们的分类目标  提示：我们有多少个标签？什么激活适合最后的输出？
model.add(Dense(num_classes, activation='softmax'))
```

In [88]:

```
### 运行 model.summary（） 回答下列问题 第二天课上一起讨论 ####
### 能否画出这个模型的概括图?  >>>

### 这个模型有几个卷积层？ 3
### 这个模型最大的参数量是哪一层？full - connection
### 第一层卷积层为什么有 320 个实际变量需要调节      32 * 9 + 32 * 1  (W,bias)  y=wx+b


#最后一层 max——pooling 完 有 64 个 6*6 feature maps 64*6*6 = 2304

### para = 2304*128 + 128 #128 个 w,b

model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 24, 24, 32)        9248
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 12, 12, 32)        0
```

```
_________________________________________________________________
dropout_10 (Dropout)         (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 12, 12, 64)        18496
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
dropout_11 (Dropout)         (None, 6, 6, 64)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               295040
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512
_________________________________________________________________
dropout_12 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1290
=================================================================
Total params: 324,906
Trainable params: 324,650
Non-trainable params: 256
_________________________________________________________________
```

```
#优化器  尝试使用不同的优化器 至少以下三种
## 中文参考 https://keras.io/zh/optimizers/
##
## SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)
## RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)
## Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

optimizer = RMSprop(lr=0.01, rho=0.9, epsilon=1e-08, decay=0.0)

### 将模型compile 编译
### 调节loss 参数，即loss function
### mean_squared_error
### categorical_crossentropy/为什么不用binary_crossentropy
### mean_absolute_error
model.compile(optimizer = optimizer , loss = "categorical_crossentropy", metrics=["accuracy"])

### training 过程中的 自动调节函数
### Reduce LR On Plateau = 减少学习率，当某一个参数达到一个平台期 自动的 把上面优化器中的 lr 减小

learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
history = model.fit(X_train,Y_train, batch_size=batch_size,
                              epochs = epochs, validation_data = (X_val, Y_val),callbacks=[learning_rate_reduction])
```

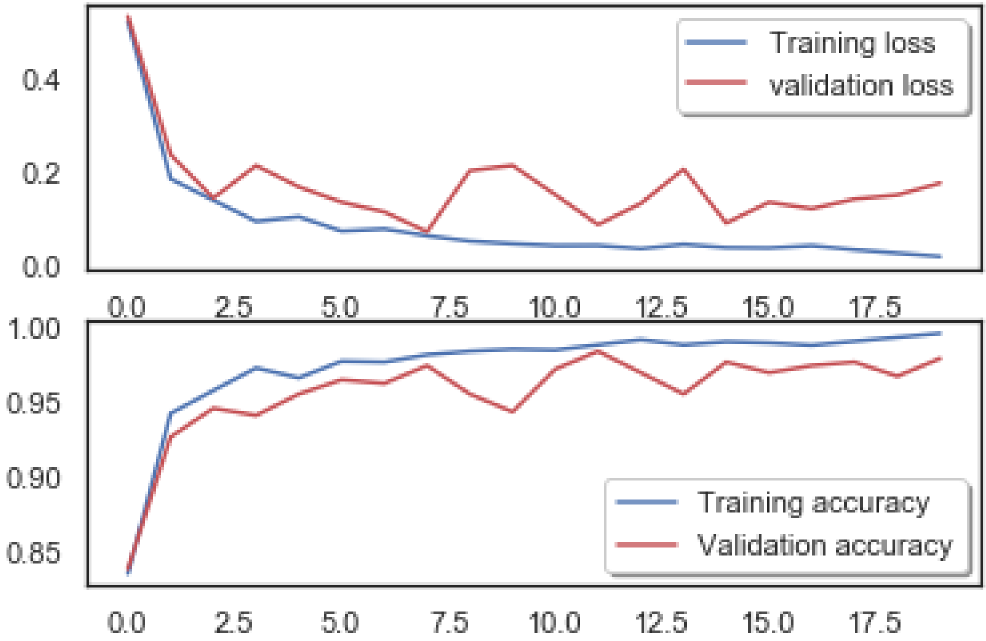
```
Train on 3780 samples, validate on 420 samples
Epoch 1/20
3780/3780 [==============================] - 5s 1ms/step - loss: 0.5209 - accuracy: 0.8349 - val_loss: 0.5312 - val_accuracy: 0.8381
Epoch 2/20
 120/3780 [..............................] - ETA: 4s - loss: 0.1108 - accuracy: 0.9667
//miniconda3/lib/python3.7/site-packages/keras/callbacks/callbacks.py:1042: RuntimeWarning: Reduce LR on plateau conditioned on metric `val_acc` which is not available. Available metrics are: val_loss,val_accuracy,loss,accuracy,lr
```

```
  (self.monitor, ','.join(list(logs.keys()))), RuntimeWarn
ing
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.1837 - accuracy: 0.9418 - val_loss: 0.2356 - val_a
ccuracy: 0.9262
Epoch 3/20
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.1379 - accuracy: 0.9571 - val_loss: 0.1425 - val_a
ccuracy: 0.9452
Epoch 4/20
3780/3780 [==============================] - 4s 1ms/step -
 loss: 0.0927 - accuracy: 0.9722 - val_loss: 0.2121 - val_a
ccuracy: 0.9405
Epoch 5/20
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.1025 - accuracy: 0.9656 - val_loss: 0.1666 - val_a
ccuracy: 0.9548
Epoch 6/20
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.0722 - accuracy: 0.9767 - val_loss: 0.1338 - val_a
ccuracy: 0.9643
Epoch 7/20
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.0767 - accuracy: 0.9762 - val_loss: 0.1127 - val_a
ccuracy: 0.9619
Epoch 8/20
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.0621 - accuracy: 0.9810 - val_loss: 0.0707 - val_a
ccuracy: 0.9738
Epoch 9/20
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.0504 - accuracy: 0.9833 - val_loss: 0.2012 - val_a
ccuracy: 0.9548
Epoch 10/20
3780/3780 [==============================] - 6s 1ms/step -
 loss: 0.0450 - accuracy: 0.9847 - val_loss: 0.2125 - val_a
ccuracy: 0.9429
Epoch 11/20
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.0409 - accuracy: 0.9841 - val_loss: 0.1498 - val_a
ccuracy: 0.9714
Epoch 12/20
```

```
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.0413 - accuracy: 0.9876 - val_loss: 0.0858 - val_a
ccuracy: 0.9833
Epoch 13/20
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.0343 - accuracy: 0.9910 - val_loss: 0.1311 - val_a
ccuracy: 0.9690
Epoch 14/20
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.0436 - accuracy: 0.9876 - val_loss: 0.2045 - val_a
ccuracy: 0.9548
Epoch 15/20
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.0366 - accuracy: 0.9897 - val_loss: 0.0892 - val_a
ccuracy: 0.9762
Epoch 16/20
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.0359 - accuracy: 0.9889 - val_loss: 0.1337 - val_a
ccuracy: 0.9690
Epoch 17/20
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.0409 - accuracy: 0.9873 - val_loss: 0.1197 - val_a
ccuracy: 0.9738
Epoch 18/20
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.0321 - accuracy: 0.9899 - val_loss: 0.1404 - val_a
ccuracy: 0.9762
Epoch 19/20
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.0247 - accuracy: 0.9926 - val_loss: 0.1492 - val_a
ccuracy: 0.9667
Epoch 20/20
3780/3780 [==============================] - 5s 1ms/step -
 loss: 0.0175 - accuracy: 0.9952 - val_loss: 0.1745 - val_a
ccuracy: 0.9786
```

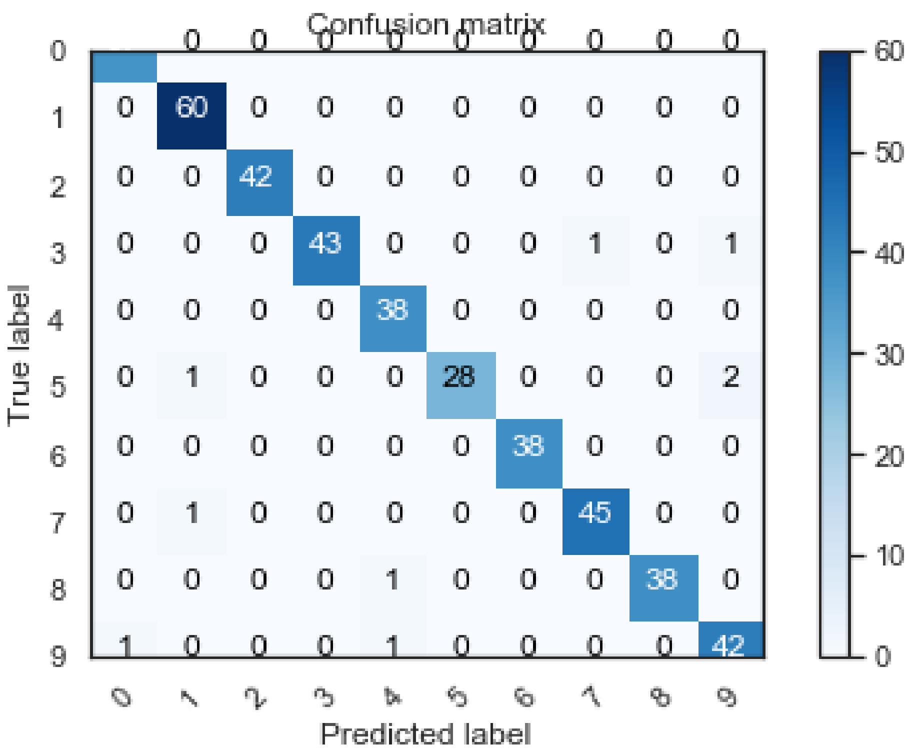
```python
# 生成学习曲线 和损失函数 随着 epoch 的变化曲线
# 模型的学习效果怎么样？ 能找到适合的 epoch 吗？
# 简单的评价标准应该用什么？
# 尝试改变模型参数 生成不同的学习曲线 比较
# 提示 从 epoch>优化器>损失函数>学习率>dropout 有无 依次调试
fig, ax = plt.subplots(2,1)

ax[0].plot(history.history['loss'], color='b', label="Training loss")
```

```
ax[0].plot(history.history['val_loss'], color='r', label="validation loss",axes =ax
[0])
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['accuracy'], color='b', label="Training accuracy")
ax[1].plot(history.history['val_accuracy'], color='r',label="Validation accuracy")
legend = ax[1].legend(loc='best', shadow=True)
```



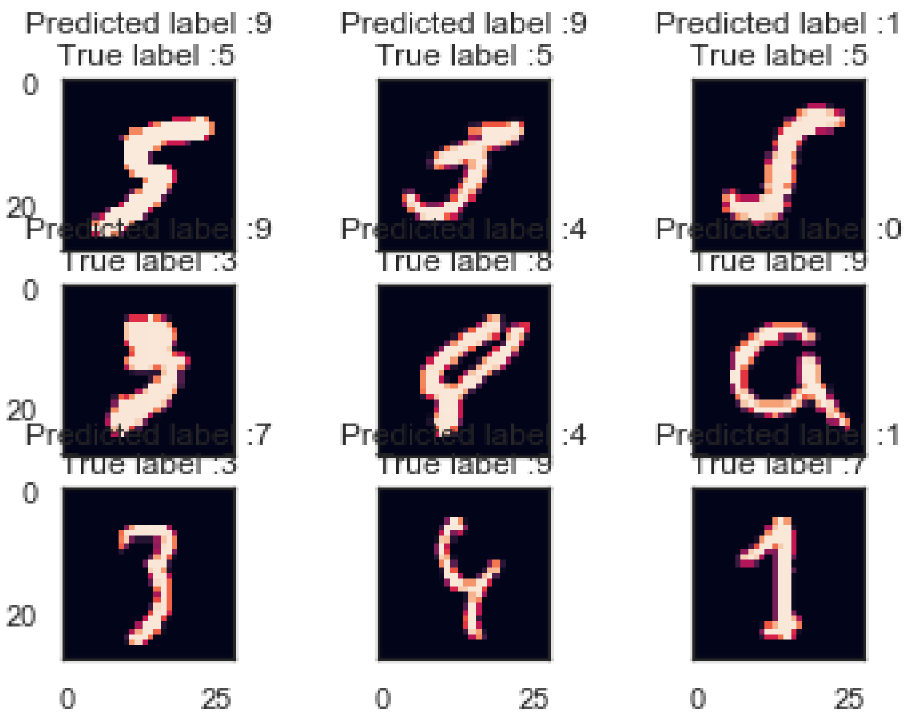

```
# 生成 10 标签混淆矩阵

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
```

```
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Predict the values from the validation dataset
Y_pred = model.predict(X_val)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(Y_val,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(10))
```




```
### 打印出认错的数字

errors = (Y_pred_classes - Y_true != 0)

Y_pred_classes_errors = Y_pred_classes[errors]
```

```
Y_pred_errors = Y_pred[errors]
Y_true_errors = Y_true[errors]
X_val_errors = X_val[errors]

def display_errors(errors_index,img_errors,pred_errors, obs_errors):
    """ This function shows 6 images with their predicted and real labels
"""
    n = 0
    nrows = 3
    ncols = 3
    fig, ax = plt.subplots(nrows,ncols,sharex=True,sharey=True)
    for row in range(nrows):
        for col in range(ncols):
            error = errors_index[n]
            ax[row,col].imshow((img_errors[error]).reshape((28,28)))
            ax[row,col].set_title("Predicted label :{}\nTrue label :{}".format
(pred_errors[error],obs_errors[error]))
            n += 1

# Probabilities of the wrong predicted numbers
Y_pred_errors_prob = np.max(Y_pred_errors,axis = 1)

# Predicted probabilities of the true values in the error set
true_prob_errors = np.diagonal(np.take(Y_pred_errors, Y_true_errors, axis=1))

# Difference between the probability of the predicted label and the true lab
el
delta_pred_true_errors = Y_pred_errors_prob - true_prob_errors

# Sorted list of the delta prob errors
sorted_dela_errors = np.argsort(delta_pred_true_errors)

# Top 9 errors
most_important_errors = sorted_dela_errors[-9:]

# Show the top 9 errors
display_errors(most_important_errors, X_val_errors, Y_pred_classes_errors, Y_t
rue_errors)
```




```
#optional 画出 roc
from sklearn.metrics import roc_curve, auc
fpr = dict()
tpr = dict()
roc_auc = dict()
y_score = model.predict(X_test)
for i in range(num_classes):
    fpr[i], tpr[i], _ = roc_curve(Y_test[:,i], y_score[:,i]) #
   # AUC Area Under the Curve
    roc_auc[i] = auc(fpr[i], tpr[i])
#y_pred_keras = model.predict(X_test).ravel()
##fpr_keras, tpr_keras, thresholds_keras = roc_curve(Y_test, y_pred_keras)
#y_pred_keras
```

Out[152]:

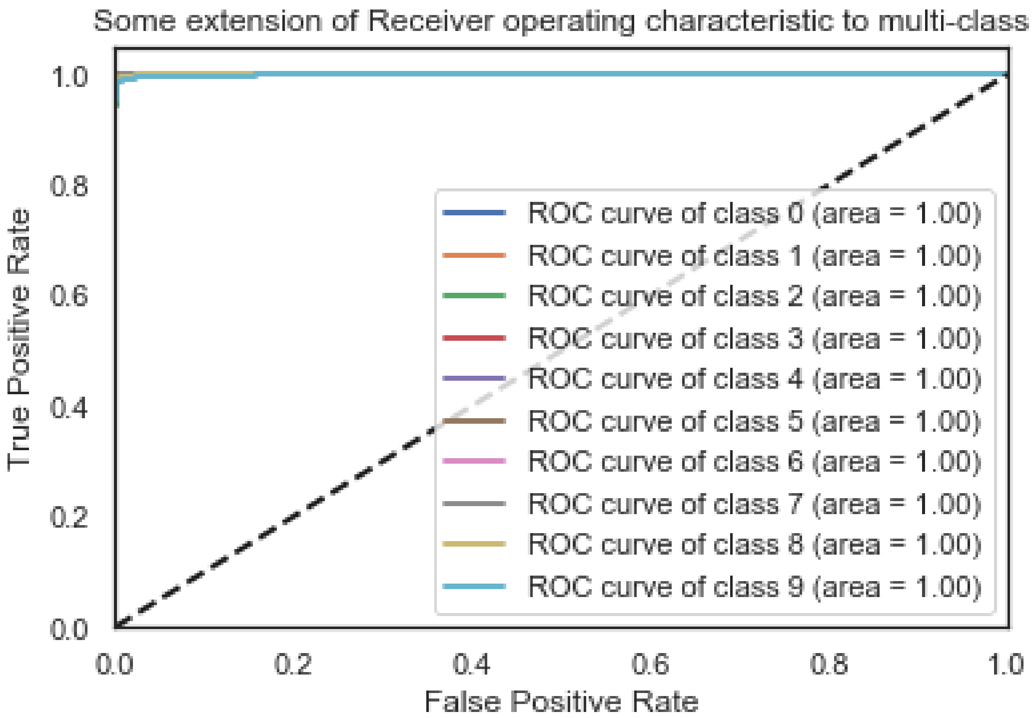
```
{0: array([0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 5.2
6038927e-04,
        5.26038927e-04, 1.05207785e-03, 1.05207785e-03, 8.4
7974750e-01,
        8.49026828e-01, 1.00000000e+00]),
 1: array([0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.0
0000000e+00,
```

```
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.0
0000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.0
0000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.0
0000000e+00,
       0.00000000e+00, 0.00000000e+00, 5.31349628e-04, 5.3
1349628e-04,
       5.84484591e-03, 5.84484591e-03, 6.90754516e-03, 6.9
0754516e-03,
       1.85972370e-02, 1.85972370e-02, 1.00000000e+00]),
 2: array([0.        , 0.        , 0.        , 0.        , 0.
     ,
       0.        , 0.        , 0.        , 0.        , 0.
 ,
       0.        , 0.        , 0.00106383, 0.00106383, 0.0015
9574,
       0.00159574, 0.00212766, 0.00212766, 0.00265957, 0.0
0265957,
       0.00319149, 0.00319149, 0.0037234 , 0.0037234 , 0.00
425532,
       0.00425532, 0.00531915, 0.00531915, 0.80212766, 0.8
0319149,
       1.        ]),
 3: array([0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.0
0000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.0
0000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.0
0000000e+00,
       0.00000000e+00, 0.00000000e+00, 5.33902830e-04, 5.3
3902830e-04,
       1.06780566e-03, 1.06780566e-03, 2.13561132e-03, 2.1
3561132e-03,
       4.27122264e-03, 4.27122264e-03, 6.24666311e-01, 6.2
5734116e-01,
       1.00000000e+00]),
 4: array([0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 5.2
8820730e-04,
```

```
       5.28820730e-04, 1.05764146e-03, 1.05764146e-03, 2.6
4410365e-03,
       2.64410365e-03, 1.00000000e+00]),
 5: array([0.        , 0.        , 0.        , 0.        , 0.
    ,
       0.        , 0.        , 0.        , 0.        , 0.
 ,
       0.00263296, 0.00263296, 1.        ]),
 6: array([0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.0
0000000e+00,
       0.00000000e+00, 5.30222694e-04, 5.30222694e-04, 2.1
2089077e-03,
       2.12089077e-03, 1.00000000e+00]),
 7: array([0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00,
       0.00000000e+00, 5.31914894e-04, 5.31914894e-04, 1.0
6382979e-03,
       1.06382979e-03, 1.59574468e-03, 1.59574468e-03, 1.0
0000000e+00]),
 8: array([0.        , 0.        , 0.        , 0.        , 0.
    ,
       0.        , 0.        , 0.        , 0.        , 0.
 ,
       0.00156495, 0.00156495, 0.00260824, 0.00260824, 0.0
0521648,
       0.00521648, 0.02399583, 0.02399583, 1.        ]),
 9: array([0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.0
0000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.0
0000000e+00,
       0.00000000e+00, 5.28820730e-04, 5.28820730e-04, 1.0
5764146e-03,
       1.05764146e-03, 2.11528292e-03, 2.11528292e-03, 7.4
0349022e-03,
       7.40349022e-03, 2.37969328e-02, 2.37969328e-02, 1.5
7588577e-01,
       1.57588577e-01, 1.00000000e+00])}
```

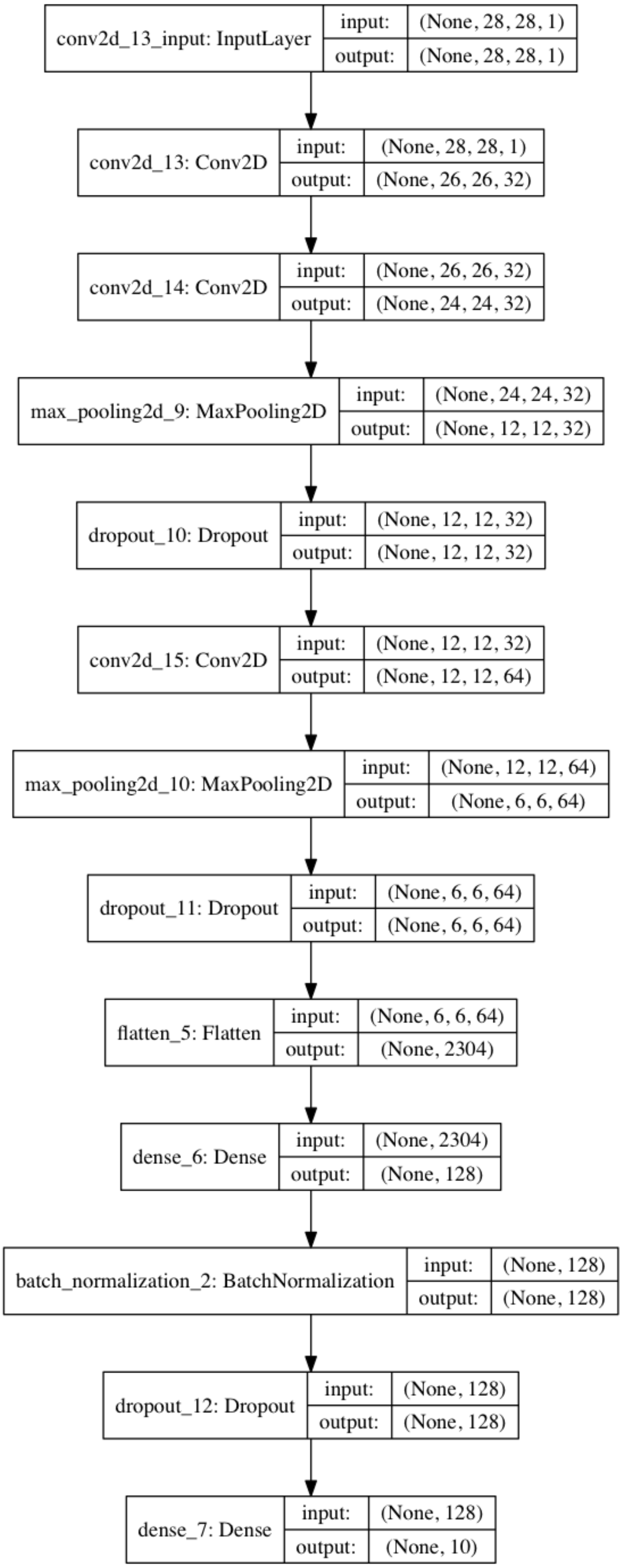
```
for i in range(num_classes):
    plt.plot(fpr[i], tpr[i], lw=2, label='ROC curve of class {0} (area = {1:0.2
f})'
```

```
                    ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Some extension of Receiver operating characteristic to multi-class')
plt.legend(loc="lower right")
plt.show()
```




```
from keras.utils import plot_model
plot_model(model, to_file='model.png', show_shapes=True)
```

Out[94]:


conv2d_13_input: InputLayer
input: (None, 28, 28, 1)
output: (None, 28, 28, 1)
conv2d_13: Conv2D
input: (None, 28, 28, 1)
output: (None, 26, 26, 32)
conv2d_14: Conv2D
input: (None, 26, 26, 32)
output: (None, 24, 24, 32)
max_pooling2d_9: MaxPooling2D
input: (None, 24, 24, 32)
output: (None, 12, 12, 32)
dropout_10: Dropout
input: (None, 12, 12, 32)
output: (None, 12, 12, 32)
conv2d_15: Conv2D
input: (None, 12, 12, 32)
output: (None, 12, 12, 64)
max_pooling2d_10: MaxPooling2D
input: (None, 12, 12, 64)
output: (None, 6, 6, 64)
dropout_11: Dropout
input: (None, 6, 6, 64)
output: (None, 6, 6, 64)
flatten_5: Flatten
input: (None, 6, 6, 64)
output: (None, 2304)
dense_6: Dense
input: (None, 2304)
output: (None, 128)
batch_normalization_2: BatchNormalization
input: (None, 128)
output: (None, 128)
dropout_12: Dropout
input: (None, 128)
output: (None, 128)
dense_7: Dense
input: (None, 128)
output: (None, 10)