1. Introduction

In [37]:
```python
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-encoding
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.optimizers import RMSprop, Adam, SGD
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

In [38]:
```python
# >>>>>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)
```

In [39]:
```python
# >>>>>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

X_train
```

Out[39]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel78
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4195	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4196	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4197	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4198	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4199	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	

4200 rows × 784 columns

2.2 Check for null and missing values

In [40]:
```
# 检查训练数据是否有空值
X_train.isnull().any().describe()
```

Out[40]:
```
count       784
unique        1
top       False
freq        784
dtype: object
```

In [41]:
```
# >>>>填写<<<< 检查训练数据是否有空值 >>>>填写<<<< ###
X_test.isnull().any().describe()
```

Out[41]:
```
count       784
unique        1
top       False
freq        784
dtype: object
```

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

In [42]:
```
# Normalize the data
X_train = X_train / 255.0
###### >>>填写<<< 标准化测试集合 #######
X_test = X_test / 255.0
X_train.shape
```

Out[42]:
```
(4200, 784)
```

2.3 Reshape

In [43]:
```
# >>>>填写<<<<< 利用 reshape 函数， 将X_train变换成 (height = 28px, width = 28px , canal = 0) ######
# CNN (batch=-1 取所有=4200, rows, cols, channels)
X_train = X_train.values.reshape(-1,28,28)  # 训练集合是4200个，28*28 通道数为1的输入）
                                            # （4200, 28, 28）
#对于RNN 输入为3D 张量，尺寸为 (batch_size, timesteps, input_dim)。
#X_train = X_train.values.reshape(-1,28,28,1)
X_test = X_test.values.reshape(-1,28,28)

X_train.shape
```

Out[43]:
```
(4200, 28, 28)
```

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

In [44]:
```
# 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
```

2.6 Split training and valdiation set

In [45]:
```
# Set the random seed
random_seed = 2
```

In [46]:
```
# 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.

In [47]:
```
# 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
```

4. RNN

Type *Markdown* and LaTeX: α^2

In [50]:
```
### RNN 答案 ########
batch_size = 100
num_classes = 10
epochs = 50
from keras.layers import SimpleRNN,LSTM
# 如果图片是28*32像素
# timsteps = 32 / 28 都可以
#(timesteps, input_dim)   28个时间节点的, 28个维vector
# keras.layers.LSTM(units, activation='tanh', recurrent_activation='hard_sigmoid', use_bias=True, kernel_initializer='glorot_uniform', rec
model = Sequential()
model.add(LSTM(128, input_shape=(28,28), return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(128))
model.add(Dropout(0.1))

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(10, activation='softmax'))
```

In [51]:
```
X_train.shape
```

Out[51]:
```
(3780, 28, 28)
```

In [52]:
```
### 运行model.summary () 回答下列问题 第二天课上一起讨论 ####
model.summary()
## LSTM中的参数 跟simpleRNN比 什么变化, 越多的参数会有什么结果? ###
#
```

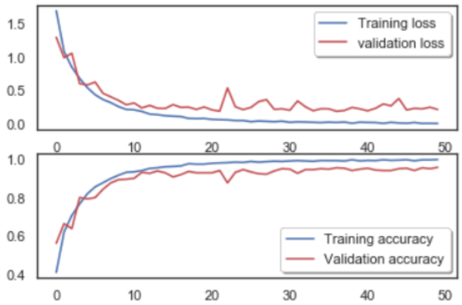
```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 28, 128)           80384
_________________________________________________________________
dropout_3 (Dropout)          (None, 28, 128)           0
_________________________________________________________________
lstm_2 (LSTM)                (None, 128)               131584
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 220,874
Trainable params: 220,874
Non-trainable params: 0
_________________________________________________________________
```

In [53]:
```
#优化器  尝试使用不同的优化器 至少以下三种 在DL 一个调节的点
## 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=0.001, rho=0.9, epsilon=0.0000001, 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
### 尝试用
###
### 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_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)


#### LSTM 尝试用 sparse_categorical_crossentropy 看看有什么变化结果 ##########
```

```
In [54]: history = model.fit(X_train, Y_train, batch_size=batch_size,
                             epochs = epochs, validation_data = (X_val, Y_val))
```

```
Train on 3780 samples, validate on 420 samples
Epoch 1/50
3780/3780 [==============================] - 4s 1ms/step - loss: 1.6891 - accuracy: 0.4108 - val_loss: 1.2944 - val_accuracy: 0.5619
Epoch 2/50
3780/3780 [==============================] - 3s 743us/step - loss: 1.0831 - accuracy: 0.6201 - val_loss: 0.9908 - val_accuracy: 0.6643
Epoch 3/50
3780/3780 [==============================] - 3s 765us/step - loss: 0.8485 - accuracy: 0.7071 - val_loss: 1.0563 - val_accuracy: 0.6381
Epoch 4/50
3780/3780 [==============================] - 3s 764us/step - loss: 0.6834 - accuracy: 0.7667 - val_loss: 0.6043 - val_accuracy: 0.8000
Epoch 5/50
3780/3780 [==============================] - 3s 761us/step - loss: 0.5438 - accuracy: 0.8198 - val_loss: 0.5852 - val_accuracy: 0.7929
Epoch 6/50
3780/3780 [==============================] - 3s 863us/step - loss: 0.4386 - accuracy: 0.8556 - val_loss: 0.6277 - val_accuracy: 0.8000
Epoch 7/50
3780/3780 [==============================] - 3s 860us/step - loss: 0.3684 - accuracy: 0.8770 - val_loss: 0.4610 - val_accuracy: 0.8429
Epoch 8/50
3780/3780 [==============================] - 3s 865us/step - loss: 0.3219 - accuracy: 0.8987 - val_loss: 0.4078 - val_accuracy: 0.8762
Epoch 9/50
3780/3780 [==============================] - 3s 823us/step - loss: 0.2623 - accuracy: 0.9161 - val_loss: 0.3544 - val_accuracy: 0.8929
Epoch 10/50
3780/3780 [==============================] - 3s 871us/step - loss: 0.2204 - accuracy: 0.9312 - val_loss: 0.2894 - val_accuracy: 0.8952
Epoch 11/50
3780/3780 [==============================] - 3s 845us/step - loss: 0.2143 - accuracy: 0.9336 - val_loss: 0.3163 - val_accuracy: 0.9000
Epoch 12/50
3780/3780 [==============================] - 3s 887us/step - loss: 0.1906 - accuracy: 0.9402 - val_loss: 0.2430 - val_accuracy: 0.9310
Epoch 13/50
3780/3780 [==============================] - 3s 839us/step - loss: 0.1515 - accuracy: 0.9513 - val_loss: 0.2799 - val_accuracy: 0.9262
Epoch 14/50
3780/3780 [==============================] - 3s 859us/step - loss: 0.1420 - accuracy: 0.9550 - val_loss: 0.2375 - val_accuracy: 0.9381
Epoch 15/50
3780/3780 [==============================] - 3s 881us/step - loss: 0.1268 - accuracy: 0.9601 - val_loss: 0.2348 - val_accuracy: 0.9286
Epoch 16/50
```

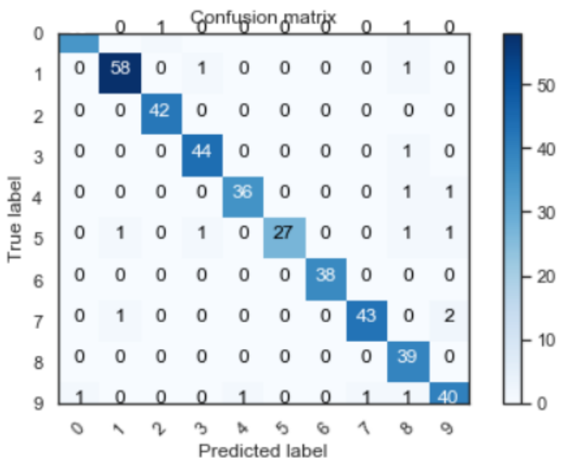
```
In [55]: # 生成学习曲线 和损失函数 随着epoch的变化曲线
         # 模型的学习效果怎么样? 能找到适合的epoch吗?
         # 简单的评价标准应该用什么?
         # 尝试改变模型参数 生成不同的学习曲线 比较
         # 提示 从epoch> 优化器> 损失函数> 学习率> dropout有无 依次调试

         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)
```



In [56]:

```
# 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')
```

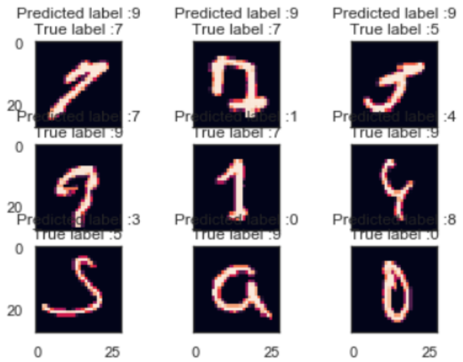
```
# Predict the values from the validation dataset
Y_pred = model.predict(X_val)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(Y_val,axis = 1)
# compute the confusion matrix
confusion_mtx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



In [57]:
```python
### 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 label
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)
```

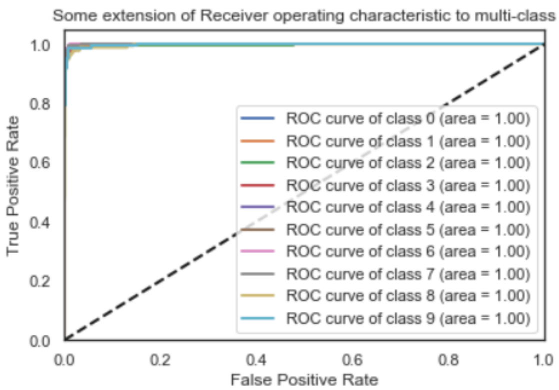

sorted_dela_errors = np.argsort(delta_pred_true_errors)


```python
# 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_true_errors)
```



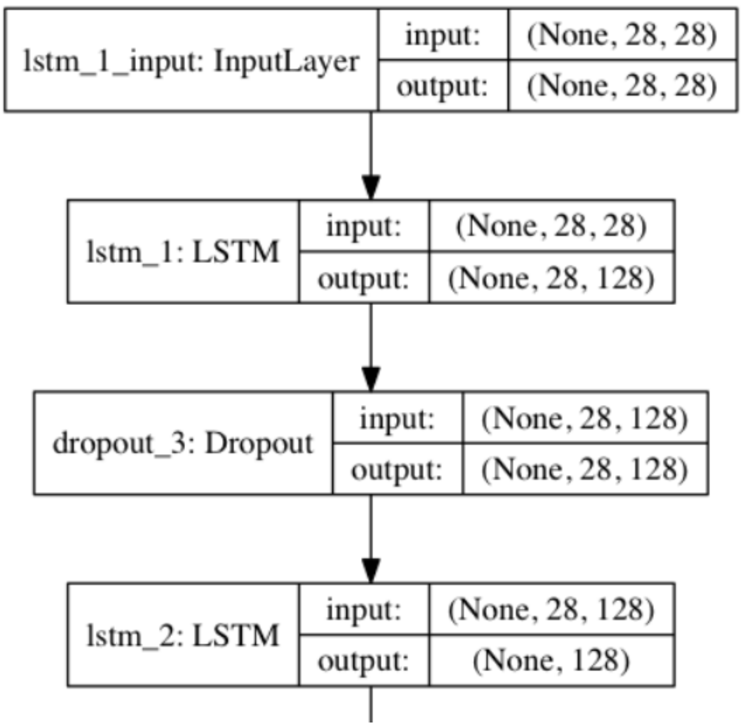
In [58]:
```python
#optional 画出roc
from sklearn.metrics import roc_curve, auc
fpr = dict()
tpr = dict()
roc_auc = dict()
y_score = model.predict(X_test)
# 在前天的作业中 y_test Pandas下的DataFrame类型: y_test
# 让数据为 Pandas DataFrame类型的话 调用/使用他 第i行第j列的数据:
#  y_test.iloc[i,j]

# 在今天的作业中, y_test是 numpy的 numarry数据类型
# 让数据为numarray 类型的话 调用/使用他 第i行第j列的数据:
# y_test[i,j]
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
```

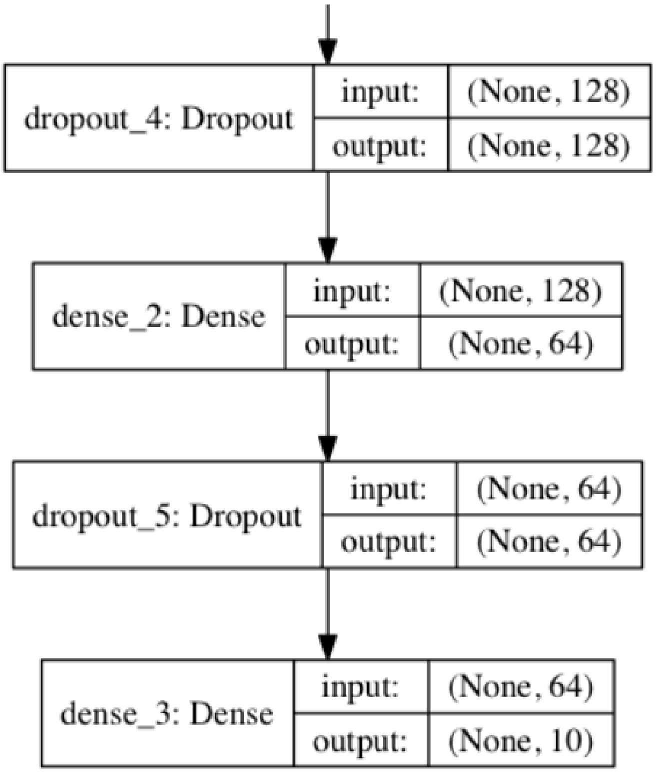
In [59]:
```
for i in range(num_classes):
    plt.plot(fpr[i], tpr[i], lw=2, label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

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()
```



In [60]:
```
from keras.utils import plot_model
plot_model(model, to_file='model.png', show_shapes=True)
```

Out[60]:





dropout_4: Dropout
input:
(None, 128)
output:
(None, 128)
dense_2: Dense
input:
(None, 128)
output:
(None, 64)
dropout_5: Dropout
input:
(None, 64)
output:
(None, 64)
dense_3: Dense
input:
(None, 64)
output:
(None, 10)