Hands-on Workshop: Deep Learning for Computer Vision

March 2019

Instructor: Santosh Chapaneri

- The Fashion MNIST database has a database of fashion accessories.
- Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. Han Xiao, Kashif Rasul, Roland Vollgraf. arXiv:1708.07747, 2017.
- The training set has 60,000 samples. The test set has 10,000 samples.
- The fashion accessories are size-normalized and centered in a fixed-size image.
- We will train Multi-layer Perceptron, Deep Multi-layer Perceptron and CNN classifier using Keras for Fashion MNIST dataset.

In [1]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
import keras
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
```

```
Using TensorFlow backend.
```

In [2]:
```
# Load the fashion-mnist pre-shuffled train data and test data
(X_train, Y_train), (X_test, Y_test) = tf.keras.datasets.fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx
1-ubyte.gz
32768/29515 [=================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx
3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1
-ubyte.gz
8192/5148 [================================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3
-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
```

In [3]:
```
# Print training set shape - note there are 60,000 training data of image size of 28x28, 60,000 tr
ain labels)
print("X_train shape:", X_train.shape, "Y_train shape:", Y_train.shape)
```

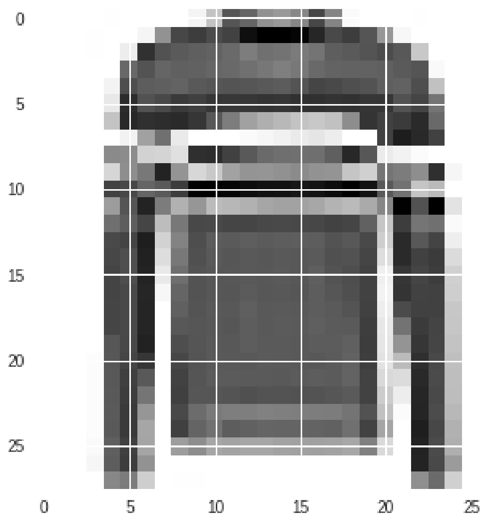
```
X_train shape: (60000, 28, 28) Y_train shape: (60000,)
```

Visualize the data

In [4]:
```
# Define the labels
fashion_mnist_labels = ["T-shirt/top",  # index 0
                        "Trouser",      # index 1
                        "Pullover",     # index 2
                        "Dress",        # index 3
                        "Coat",         # index 4
                        "Sandal",       # index 5
                        "Shirt",        # index 6
                        "Sneaker",      # index 7
                        "Bag",          # index 8
                        "Ankle boot"]   # index 9

# Image index, you can pick any number between 0 and 59,999
img_index = 5

# y_train contains the lables, ranging from 0 to 9
label_index = Y_train[img_index]

# Print the label, for example 2 Pullover
print ("Y = " + str(label_index) + " " +(fashion_mnist_labels[label_index]))

# # Show one of the images from the training dataset
plt.imshow(X_train[img_index])
```

```
Y = 2 Pullover
```

Out[4]: <matplotlib.image.AxesImage at 0x7f0d6e51c748>



In [0]:
```
img_rows, img_cols = 28, 28

# MLP
X_train_mlp = X_train.reshape(X_train.shape[0],img_rows*img_cols)
Y_train_mlp = Y_train
X_test_mlp = X_test.reshape(X_test.shape[0],img_rows*img_cols)
Y_test_mlp = Y_test

# CNN
X_train_cnn = X_train.reshape(X_train.shape[0],img_rows,img_cols,1)
Y_train_cnn = Y_train
X_test_cnn = X_test.reshape(X_test.shape[0],img_rows,img_cols,1)
Y_test_cnn = Y_test
```

In [6]:
```
print(X_train_mlp.shape)
print(X_train_cnn.shape)
```

```
(60000, 784)
(60000, 28, 28, 1)
```

In [0]:
```
X_train_mlp = X_train_mlp.astype('float32')
X_test_mlp = X_test_mlp.astype('float32')
X_train_mlp /= 255
X_test_mlp /= 255

X_train_cnn = X_train_cnn.astype('float32')
X_test_cnn = X_test_cnn.astype('float32')
X_train_cnn /= 255
X_test_cnn /= 255
```

In [0]:
```
# Convert class vectors to binary class matrices
num_classes = 10

Y_train_mlp = keras.utils.to_categorical(Y_train_mlp, num_classes)
Y_test_mlp = keras.utils.to_categorical(Y_test_mlp, num_classes)

Y_train_cnn = keras.utils.to_categorical(Y_train_cnn, num_classes)
Y_test_cnn = keras.utils.to_categorical(Y_test_cnn, num_classes)
```

In [9]:
```
Y_train_cnn[:5,:]
```

Out[9]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

In [0]:
```
# Split data to optimize classifier during training
X_train_mlp, X_val_mlp, Y_train_mlp, Y_val_mlp = train_test_split(X_train_mlp,
                                                                  Y_train_mlp,
                                                                  test_size=0.2)

X_train_cnn, X_val_cnn, Y_train_cnn, Y_val_cnn = train_test_split(X_train_cnn,
                                                                  Y_train_cnn,
                                                                  test_size=0.2)
```

In [11]:
```
print(X_train_mlp.shape)
print(X_val_mlp.shape)

print(X_train_cnn.shape)
print(X_val_cnn.shape)
```

```
(48000, 784)
(12000, 784)
(48000, 28, 28, 1)
(12000, 28, 28, 1)
```

Multi Layer Perceptron

In [0]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
from keras.datasets import mnist
from keras.utils import np_utils
```

In [13]:
```
# Multilayer Perceptron model

batch_size = 256
num_epochs = 50

model = Sequential()

model.add(Dense(input_dim=784, activation='sigmoid',
                units=625, kernel_initializer='normal'))

model.add(Dense(input_dim=625, activation='softmax',
                units=10, kernel_initializer='normal'))

model.compile(optimizer=SGD(lr=0.05),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/op_def_
library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be rem
oved in a future version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 625)               490625
_________________________________________________________________
dense_2 (Dense)              (None, 10)                6260
=================================================================
Total params: 496,885
Trainable params: 496,885
Non-trainable params: 0
_________________________________________________________________
```

In [14]:
```
history = model.fit(X_train_mlp, Y_train_mlp,
          batch_size = batch_size,
          epochs = num_epochs,
          verbose = 1,
          validation_data = (X_val_mlp, Y_val_mlp))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_ops.py:3
066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
Train on 48000 samples, validate on 12000 samples
Epoch 1/50
48000/48000 [==============================] - 4s 75us/step - loss: 1.3990 - acc: 0.6144 - val_los
s: 0.9937 - val_acc: 0.7046
Epoch 2/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.8636 - acc: 0.7324 - val_los
s: 0.7843 - val_acc: 0.7523
Epoch 3/50
48000/48000 [==============================] - 1s 20us/step - loss: 0.7346 - acc: 0.7546 - val_los
s: 0.7099 - val_acc: 0.7500
Epoch 4/50
48000/48000 [==============================] - 1s 20us/step - loss: 0.6726 - acc: 0.7682 - val_los
s: 0.6556 - val_acc: 0.7751
Epoch 5/50
48000/48000 [==============================] - 1s 22us/step - loss: 0.6325 - acc: 0.7810 - val_los
s: 0.6275 - val_acc: 0.7865
Epoch 6/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.6035 - acc: 0.7891 - val_los
s: 0.6007 - val_acc: 0.7917
Epoch 7/50
48000/48000 [==============================] - 1s 20us/step - loss: 0.5812 - acc: 0.7975 - val_los
s: 0.5873 - val_acc: 0.7985
Epoch 8/50
48000/48000 [==============================] - 1s 20us/step - loss: 0.5636 - acc: 0.8042 - val_los
s: 0.5678 - val_acc: 0.8022
Epoch 9/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.5478 - acc: 0.8093 - val_los
s: 0.5555 - val_acc: 0.8078
Epoch 10/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.5351 - acc: 0.8136 - val_los
s: 0.5463 - val_acc: 0.8105
Epoch 11/50
48000/48000 [==============================] - 1s 20us/step - loss: 0.5242 - acc: 0.8172 - val_los
s: 0.5323 - val_acc: 0.8153
Epoch 12/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.5157 - acc: 0.8200 - val_los
s: 0.5211 - val_acc: 0.8212
Epoch 13/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.5077 - acc: 0.8228 - val_los
s: 0.5156 - val_acc: 0.8193
Epoch 14/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.5002 - acc: 0.8249 - val_los
s: 0.5198 - val_acc: 0.8162
Epoch 15/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4940 - acc: 0.8273 - val_los
s: 0.5021 - val_acc: 0.8285
Epoch 16/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4880 - acc: 0.8286 - val_los
s: 0.5011 - val_acc: 0.8268
Epoch 17/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4828 - acc: 0.8315 - val_los
s: 0.5003 - val_acc: 0.8242
Epoch 18/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4779 - acc: 0.8328 - val_los
s: 0.4903 - val_acc: 0.8293
Epoch 19/50
48000/48000 [==============================] - 1s 20us/step - loss: 0.4738 - acc: 0.8343 - val_los
s: 0.4849 - val_acc: 0.8333
Epoch 20/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4691 - acc: 0.8364 - val_los
s: 0.4831 - val_acc: 0.8330
Epoch 21/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4655 - acc: 0.8373 - val_los
s: 0.4781 - val_acc: 0.8349
Epoch 22/50
48000/48000 [==============================] - 1s 20us/step - loss: 0.4625 - acc: 0.8374 - val_los
s: 0.4790 - val_acc: 0.8344
Epoch 23/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4598 - acc: 0.8400 - val_los
s: 0.4719 - val_acc: 0.8370
Epoch 24/50
```

```
48000/48000 [==============================] - 1s 21us/step - loss: 0.4563 - acc: 0.8400 - val_los
s: 0.4691 - val_acc: 0.8380
Epoch 25/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4527 - acc: 0.8421 - val_los
s: 0.4739 - val_acc: 0.8372
Epoch 26/50
48000/48000 [==============================] - 1s 22us/step - loss: 0.4503 - acc: 0.8420 - val_los
s: 0.4665 - val_acc: 0.8373
Epoch 27/50
48000/48000 [==============================] - 1s 23us/step - loss: 0.4474 - acc: 0.8436 - val_los
s: 0.4620 - val_acc: 0.8401
Epoch 28/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4455 - acc: 0.8439 - val_los
s: 0.4609 - val_acc: 0.8403
Epoch 29/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4426 - acc: 0.8461 - val_los
s: 0.4661 - val_acc: 0.8357
Epoch 30/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4408 - acc: 0.8459 - val_los
s: 0.4592 - val_acc: 0.8416
Epoch 31/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4391 - acc: 0.8474 - val_los
s: 0.4551 - val_acc: 0.8418
Epoch 32/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4370 - acc: 0.8478 - val_los
s: 0.4527 - val_acc: 0.8430
Epoch 33/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4345 - acc: 0.8482 - val_los
s: 0.4537 - val_acc: 0.8420
Epoch 34/50
48000/48000 [==============================] - 1s 20us/step - loss: 0.4323 - acc: 0.8491 - val_los
s: 0.4547 - val_acc: 0.8409
Epoch 35/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4304 - acc: 0.8500 - val_los
s: 0.4514 - val_acc: 0.8432
Epoch 36/50
48000/48000 [==============================] - 1s 20us/step - loss: 0.4287 - acc: 0.8494 - val_los
s: 0.4450 - val_acc: 0.8464
Epoch 37/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4274 - acc: 0.8499 - val_los
s: 0.4478 - val_acc: 0.8425
Epoch 38/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4259 - acc: 0.8510 - val_los
s: 0.4436 - val_acc: 0.8456
Epoch 39/50
48000/48000 [==============================] - 1s 22us/step - loss: 0.4242 - acc: 0.8515 - val_los
s: 0.4447 - val_acc: 0.8445
Epoch 40/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4224 - acc: 0.8523 - val_los
s: 0.4516 - val_acc: 0.8419
Epoch 41/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4212 - acc: 0.8522 - val_los
s: 0.4557 - val_acc: 0.8372
Epoch 42/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4197 - acc: 0.8530 - val_los
s: 0.4459 - val_acc: 0.8455
Epoch 43/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4185 - acc: 0.8541 - val_los
s: 0.4445 - val_acc: 0.8437
Epoch 44/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4170 - acc: 0.8539 - val_los
s: 0.4406 - val_acc: 0.8453
Epoch 45/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4158 - acc: 0.8548 - val_los
s: 0.4342 - val_acc: 0.8482
Epoch 46/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4146 - acc: 0.8538 - val_los
s: 0.4365 - val_acc: 0.8484
Epoch 47/50
48000/48000 [==============================] - 1s 20us/step - loss: 0.4129 - acc: 0.8551 - val_los
s: 0.4342 - val_acc: 0.8472
Epoch 48/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4112 - acc: 0.8565 - val_los
s: 0.4324 - val_acc: 0.8472
Epoch 49/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4110 - acc: 0.8548 - val_los
```

```
s: 0.4358 - val_acc: 0.8477
Epoch 50/50
48000/48000 [==============================] - 1s 21us/step - loss: 0.4101 - acc: 0.8566 - val_los
s: 0.4328 - val_acc: 0.8489
```

In [15]:

```
score = model.evaluate(X_test_mlp, Y_test_mlp, verbose = 1)
print()
print('MLP Test loss:', score[0])
print('MLP Test accuracy:', score[1])
```

```
10000/10000 [==============================] - 0s 50us/step

MLP Test loss: 0.4536403777599335
MLP Test accuracy: 0.8374
```

Deep Multi Layer Perceptron

In [0]:

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import RMSprop
from keras.datasets import mnist
from keras.utils import np_utils
```

In [17]:
```
# Deep Multilayer Perceptron model
model_deepmlp = Sequential()

model_deepmlp.add(Dense(input_dim=784, units=625, kernel_initializer='normal'))
model_deepmlp.add(Activation('relu'))
model_deepmlp.add(Dropout(0.2))

model_deepmlp.add(Dense(input_dim=625, units=625, kernel_initializer='normal'))
model_deepmlp.add(Activation('relu'))
model_deepmlp.add(Dropout(0.2))

model_deepmlp.add(Dense(input_dim=625, units=625, kernel_initializer='normal'))
model_deepmlp.add(Activation('relu'))
model_deepmlp.add(Dropout(0.2))

model_deepmlp.add(Dense(input_dim=625, units=10, kernel_initializer='normal'))
model_deepmlp.add(Activation('softmax'))

model_deepmlp.compile(optimizer=RMSprop(lr=0.001, rho=0.9),
              loss='categorical_crossentropy', metrics=['accuracy'])

model_deepmlp.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.p
y:3445: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will
be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_3 (Dense)              (None, 625)               490625    
_________________________________________________________________
activation_1 (Activation)    (None, 625)               0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 625)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 625)               391250    
_________________________________________________________________
activation_2 (Activation)    (None, 625)               0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 625)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 625)               391250    
_________________________________________________________________
activation_3 (Activation)    (None, 625)               0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 625)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 10)                6260      
_________________________________________________________________
activation_4 (Activation)    (None, 10)                0         
=================================================================
Total params: 1,279,385
Trainable params: 1,279,385
Non-trainable params: 0
_________________________________________________________________
```

In [18]:
```
history_deepmlp = model_deepmlp.fit(X_train_mlp, Y_train_mlp,
                                    batch_size = batch_size,
                                    epochs = num_epochs,
                                    verbose = 1,
                                    validation_data = (X_val_mlp, Y_val_mlp))
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/50
48000/48000 [==============================] - 2s 48us/step - loss: 0.6924 - acc: 0.7516 - val_los
s: 0.4532 - val_acc: 0.8327
Epoch 2/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.4491 - acc: 0.8334 - val_los
s: 0.4684 - val_acc: 0.8244
Epoch 3/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.4029 - acc: 0.8515 - val_los
s: 0.3793 - val_acc: 0.8577
Epoch 4/50
48000/48000 [==============================] - 2s 42us/step - loss: 0.3701 - acc: 0.8630 - val_los
s: 0.3968 - val_acc: 0.8519
Epoch 5/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.3472 - acc: 0.8709 - val_los
s: 0.3730 - val_acc: 0.8653
Epoch 6/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.3308 - acc: 0.8777 - val_los
s: 0.4574 - val_acc: 0.8343
Epoch 7/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.3244 - acc: 0.8816 - val_los
s: 0.3770 - val_acc: 0.8657
Epoch 8/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.3103 - acc: 0.8848 - val_los
s: 0.3821 - val_acc: 0.8638
Epoch 9/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.3030 - acc: 0.8879 - val_los
s: 0.3970 - val_acc: 0.8518
Epoch 10/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2967 - acc: 0.8897 - val_los
s: 0.3914 - val_acc: 0.8653
Epoch 11/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2907 - acc: 0.8944 - val_los
s: 0.4039 - val_acc: 0.8596
Epoch 12/50
48000/48000 [==============================] - 2s 41us/step - loss: 0.2867 - acc: 0.8943 - val_los
s: 0.3828 - val_acc: 0.8724
Epoch 13/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2835 - acc: 0.8964 - val_los
s: 0.3726 - val_acc: 0.8772
Epoch 14/50
48000/48000 [==============================] - 2s 41us/step - loss: 0.2770 - acc: 0.8993 - val_los
s: 0.3600 - val_acc: 0.8855
Epoch 15/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2713 - acc: 0.9010 - val_los
s: 0.3859 - val_acc: 0.8720
Epoch 16/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2693 - acc: 0.9032 - val_los
s: 0.4364 - val_acc: 0.8602
Epoch 17/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2723 - acc: 0.9011 - val_los
s: 0.3938 - val_acc: 0.8866
Epoch 18/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2666 - acc: 0.9038 - val_los
s: 0.3551 - val_acc: 0.8853
Epoch 19/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2612 - acc: 0.9052 - val_los
s: 0.3677 - val_acc: 0.8811
Epoch 20/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2630 - acc: 0.9059 - val_los
s: 0.3541 - val_acc: 0.8827
Epoch 21/50
48000/48000 [==============================] - 2s 41us/step - loss: 0.2591 - acc: 0.9059 - val_los
s: 0.3798 - val_acc: 0.8892
Epoch 22/50
48000/48000 [==============================] - 2s 41us/step - loss: 0.2542 - acc: 0.9084 - val_los
s: 0.3815 - val_acc: 0.8923
Epoch 23/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2539 - acc: 0.9084 - val_los
s: 0.4108 - val_acc: 0.8827
Epoch 24/50
48000/48000 [==============================] - 2s 41us/step - loss: 0.2562 - acc: 0.9083 - val_los
s: 0.4041 - val_acc: 0.8888
Epoch 25/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2510 - acc: 0.9101 - val_los
s: 0.3880 - val_acc: 0.8908
```

```
Epoch 26/50
48000/48000 [==============================] - 2s 43us/step - loss: 0.2469 - acc: 0.9120 - val_los
s: 0.4053 - val_acc: 0.8815
Epoch 27/50
48000/48000 [==============================] - 2s 42us/step - loss: 0.2484 - acc: 0.9106 - val_los
s: 0.3897 - val_acc: 0.8869
Epoch 28/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2410 - acc: 0.9148 - val_los
s: 0.4094 - val_acc: 0.8841
Epoch 29/50
48000/48000 [==============================] - 2s 39us/step - loss: 0.2445 - acc: 0.9141 - val_los
s: 0.3949 - val_acc: 0.8767
Epoch 30/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2443 - acc: 0.9137 - val_los
s: 0.4137 - val_acc: 0.8869
Epoch 31/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2397 - acc: 0.9164 - val_los
s: 0.3946 - val_acc: 0.8863
Epoch 32/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2423 - acc: 0.9174 - val_los
s: 0.3852 - val_acc: 0.8914
Epoch 33/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2380 - acc: 0.9163 - val_los
s: 0.3963 - val_acc: 0.8973
Epoch 34/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2317 - acc: 0.9176 - val_los
s: 0.3979 - val_acc: 0.8960
Epoch 35/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2418 - acc: 0.9154 - val_los
s: 0.4151 - val_acc: 0.8902
Epoch 36/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2335 - acc: 0.9181 - val_los
s: 0.3894 - val_acc: 0.8967
Epoch 37/50
48000/48000 [==============================] - 2s 39us/step - loss: 0.2310 - acc: 0.9186 - val_los
s: 0.4532 - val_acc: 0.8892
Epoch 38/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2291 - acc: 0.9191 - val_los
s: 0.4253 - val_acc: 0.8852
Epoch 39/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2278 - acc: 0.9200 - val_los
s: 0.4500 - val_acc: 0.8894
Epoch 40/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2270 - acc: 0.9213 - val_los
s: 0.4191 - val_acc: 0.8964
Epoch 41/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2371 - acc: 0.9199 - val_los
s: 0.5268 - val_acc: 0.8788
Epoch 42/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2301 - acc: 0.9207 - val_los
s: 0.4266 - val_acc: 0.8864
Epoch 43/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2269 - acc: 0.9228 - val_los
s: 0.3987 - val_acc: 0.8958
Epoch 44/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2246 - acc: 0.9228 - val_los
s: 0.3923 - val_acc: 0.8974
Epoch 45/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2221 - acc: 0.9238 - val_los
s: 0.4584 - val_acc: 0.8782
Epoch 46/50
48000/48000 [==============================] - 2s 39us/step - loss: 0.2225 - acc: 0.9248 - val_los
s: 0.4604 - val_acc: 0.8788
Epoch 47/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2256 - acc: 0.9239 - val_los
s: 0.4541 - val_acc: 0.8908
Epoch 48/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2204 - acc: 0.9253 - val_los
s: 0.4905 - val_acc: 0.8909
Epoch 49/50
48000/48000 [==============================] - 2s 40us/step - loss: 0.2180 - acc: 0.9265 - val_los
s: 0.4492 - val_acc: 0.8821
Epoch 50/50
48000/48000 [==============================] - 2s 39us/step - loss: 0.2190 - acc: 0.9262 - val_los
s: 0.4613 - val_acc: 0.8936
```

In [19]:
```
score = model_deepmlp.evaluate(X_test_mlp, Y_test_mlp, verbose = 1)
print()
print('Deep MLP Test loss:', score[0])
print('Deep MLP Test accuracy:', score[1])
```

```
10000/10000 [==============================] - 1s 60us/step

Deep MLP Test loss: 0.5017395622313022
Deep MLP Test accuracy: 0.8868
```

Convolutional Neural Networks

In [0]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
```

In [21]:
```
input_shape = (img_rows, img_cols, 1)

model_cnn = Sequential()
model_cnn.add(Conv2D(32, (3, 3), activation='relu',
                     kernel_initializer='normal', input_shape=input_shape))
model_cnn.add(MaxPooling2D((2, 2)))
model_cnn.add(Dropout(0.25))

model_cnn.add(Conv2D(64, (3, 3), activation='relu'))
model_cnn.add(MaxPooling2D((2, 2)))
model_cnn.add(Dropout(0.25))

model_cnn.add(Conv2D(128, (3, 3), activation='relu'))
model_cnn.add(Dropout(0.4))

model_cnn.add(Flatten())

model_cnn.add(Dense(128, activation='relu'))
model_cnn.add(Dropout(0.3))

model_cnn.add(Dense(num_classes, activation='softmax'))

model_cnn.compile(loss=keras.losses.categorical_crossentropy,
                  optimizer=keras.optimizers.Adam(),
                  metrics=['accuracy'])

model_cnn.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 13, 13, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0         
_________________________________________________________________
dropout_5 (Dropout)          (None, 5, 5, 64)          0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 128)         73856     
_________________________________________________________________
dropout_6 (Dropout)          (None, 3, 3, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0         
_________________________________________________________________
dense_7 (Dense)              (None, 128)               147584    
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1290      
=================================================================
Total params: 241,546
Trainable params: 241,546
Non-trainable params: 0
_________________________________________________________________
```

In [22]:
```
history_cnn = model_cnn.fit(X_train_cnn, Y_train_cnn,
                            batch_size = batch_size,
                            epochs = num_epochs,
                            verbose = 1,
                            validation_data = (X_val_cnn, Y_val_cnn))
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/50
48000/48000 [==============================] - 7s 139us/step - loss: 0.8736 - acc: 0.6707 - val_lo
ss: 0.5498 - val_acc: 0.7913
Epoch 2/50
48000/48000 [==============================] - 4s 80us/step - loss: 0.5542 - acc: 0.7915 - val_los
s: 0.4488 - val_acc: 0.8332
Epoch 3/50
48000/48000 [==============================] - 4s 80us/step - loss: 0.4777 - acc: 0.8215 - val_los
s: 0.3928 - val_acc: 0.8549
Epoch 4/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.4367 - acc: 0.8394 - val_los
s: 0.3685 - val_acc: 0.8675
Epoch 5/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.4037 - acc: 0.8513 - val_los
s: 0.3436 - val_acc: 0.8734
Epoch 6/50
48000/48000 [==============================] - 4s 77us/step - loss: 0.3791 - acc: 0.8603 - val_los
s: 0.3188 - val_acc: 0.8816
Epoch 7/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.3639 - acc: 0.8670 - val_los
s: 0.3067 - val_acc: 0.8857
Epoch 8/50
48000/48000 [==============================] - 4s 77us/step - loss: 0.3471 - acc: 0.8725 - val_los
s: 0.3077 - val_acc: 0.8867
Epoch 9/50
48000/48000 [==============================] - 4s 79us/step - loss: 0.3370 - acc: 0.8747 - val_los
s: 0.2952 - val_acc: 0.8902
Epoch 10/50
48000/48000 [==============================] - 4s 79us/step - loss: 0.3251 - acc: 0.8800 - val_los
s: 0.2897 - val_acc: 0.8923
Epoch 11/50
48000/48000 [==============================] - 4s 77us/step - loss: 0.3142 - acc: 0.8839 - val_los
s: 0.2761 - val_acc: 0.8978
Epoch 12/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.3043 - acc: 0.8877 - val_los
s: 0.2727 - val_acc: 0.8976
Epoch 13/50
48000/48000 [==============================] - 4s 79us/step - loss: 0.2984 - acc: 0.8913 - val_los
s: 0.2662 - val_acc: 0.9023
Epoch 14/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2914 - acc: 0.8924 - val_los
s: 0.2656 - val_acc: 0.9022
Epoch 15/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2838 - acc: 0.8968 - val_los
s: 0.2677 - val_acc: 0.9008
Epoch 16/50
48000/48000 [==============================] - 4s 77us/step - loss: 0.2797 - acc: 0.8970 - val_los
s: 0.2557 - val_acc: 0.9060
Epoch 17/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2743 - acc: 0.8977 - val_los
s: 0.2546 - val_acc: 0.9070
Epoch 18/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2691 - acc: 0.9016 - val_los
s: 0.2662 - val_acc: 0.8978
Epoch 19/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2643 - acc: 0.9011 - val_los
s: 0.2558 - val_acc: 0.9074
Epoch 20/50
48000/48000 [==============================] - 4s 79us/step - loss: 0.2604 - acc: 0.9032 - val_los
s: 0.2401 - val_acc: 0.9116
Epoch 21/50
48000/48000 [==============================] - 4s 77us/step - loss: 0.2544 - acc: 0.9052 - val_los
s: 0.2380 - val_acc: 0.9135
Epoch 22/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2525 - acc: 0.9066 - val_los
s: 0.2366 - val_acc: 0.9117
Epoch 23/50
48000/48000 [==============================] - 4s 77us/step - loss: 0.2485 - acc: 0.9082 - val_los
s: 0.2394 - val_acc: 0.9123
Epoch 24/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2490 - acc: 0.9072 - val_los
s: 0.2378 - val_acc: 0.9122
Epoch 25/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2424 - acc: 0.9086 - val_los
s: 0.2274 - val_acc: 0.9153
```

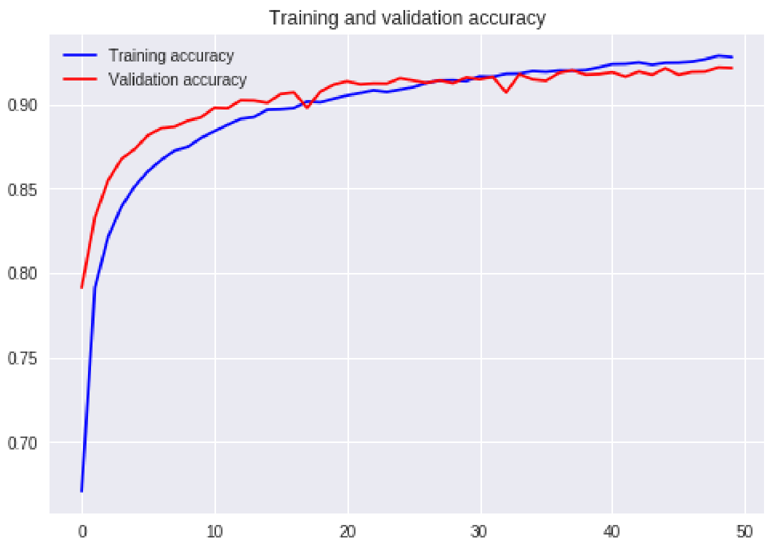
```
Epoch 26/50
48000/48000 [==============================] - 4s 77us/step - loss: 0.2389 - acc: 0.9101 - val_los
s: 0.2333 - val_acc: 0.9140
Epoch 27/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2362 - acc: 0.9126 - val_los
s: 0.2377 - val_acc: 0.9126
Epoch 28/50
48000/48000 [==============================] - 4s 76us/step - loss: 0.2311 - acc: 0.9140 - val_los
s: 0.2265 - val_acc: 0.9141
Epoch 29/50
48000/48000 [==============================] - 4s 79us/step - loss: 0.2328 - acc: 0.9142 - val_los
s: 0.2307 - val_acc: 0.9124
Epoch 30/50
48000/48000 [==============================] - 4s 79us/step - loss: 0.2296 - acc: 0.9136 - val_los
s: 0.2292 - val_acc: 0.9158
Epoch 31/50
48000/48000 [==============================] - 4s 77us/step - loss: 0.2250 - acc: 0.9165 - val_los
s: 0.2303 - val_acc: 0.9147
Epoch 32/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2227 - acc: 0.9162 - val_los
s: 0.2233 - val_acc: 0.9162
Epoch 33/50
48000/48000 [==============================] - 4s 77us/step - loss: 0.2216 - acc: 0.9180 - val_los
s: 0.2486 - val_acc: 0.9069
Epoch 34/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2186 - acc: 0.9181 - val_los
s: 0.2213 - val_acc: 0.9177
Epoch 35/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2151 - acc: 0.9196 - val_los
s: 0.2278 - val_acc: 0.9147
Epoch 36/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2176 - acc: 0.9191 - val_los
s: 0.2331 - val_acc: 0.9138
Epoch 37/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2130 - acc: 0.9199 - val_los
s: 0.2203 - val_acc: 0.9186
Epoch 38/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2130 - acc: 0.9199 - val_los
s: 0.2208 - val_acc: 0.9201
Epoch 39/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2071 - acc: 0.9202 - val_los
s: 0.2239 - val_acc: 0.9174
Epoch 40/50
48000/48000 [==============================] - 4s 79us/step - loss: 0.2055 - acc: 0.9218 - val_los
s: 0.2197 - val_acc: 0.9178
Epoch 41/50
48000/48000 [==============================] - 4s 79us/step - loss: 0.2050 - acc: 0.9237 - val_los
s: 0.2245 - val_acc: 0.9189
Epoch 42/50
48000/48000 [==============================] - 4s 80us/step - loss: 0.2049 - acc: 0.9238 - val_los
s: 0.2279 - val_acc: 0.9162
Epoch 43/50
48000/48000 [==============================] - 4s 77us/step - loss: 0.2005 - acc: 0.9247 - val_los
s: 0.2197 - val_acc: 0.9193
Epoch 44/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.2016 - acc: 0.9233 - val_los
s: 0.2251 - val_acc: 0.9173
Epoch 45/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.1986 - acc: 0.9245 - val_los
s: 0.2185 - val_acc: 0.9213
Epoch 46/50
48000/48000 [==============================] - 4s 77us/step - loss: 0.1965 - acc: 0.9246 - val_los
s: 0.2208 - val_acc: 0.9173
Epoch 47/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.1988 - acc: 0.9251 - val_los
s: 0.2224 - val_acc: 0.9191
Epoch 48/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.1965 - acc: 0.9265 - val_los
s: 0.2183 - val_acc: 0.9193
Epoch 49/50
48000/48000 [==============================] - 4s 78us/step - loss: 0.1931 - acc: 0.9287 - val_los
s: 0.2150 - val_acc: 0.9217
Epoch 50/50
48000/48000 [==============================] - 4s 79us/step - loss: 0.1913 - acc: 0.9279 - val_los
s: 0.2177 - val_acc: 0.9214
```

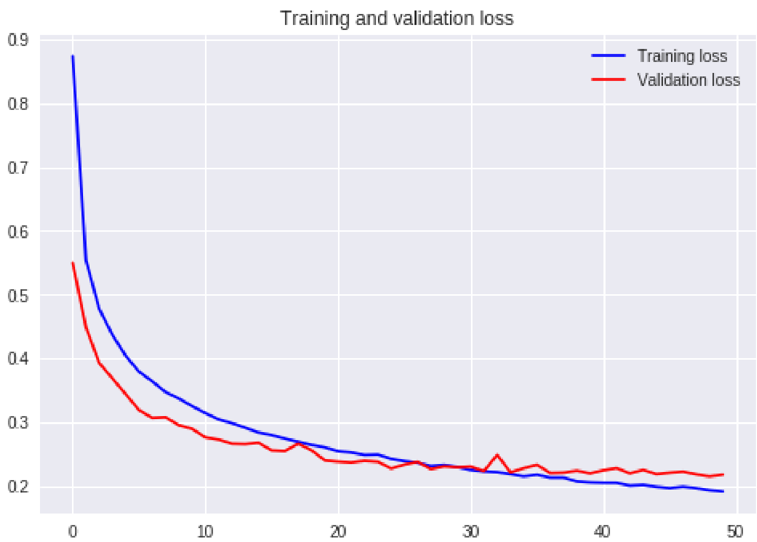
In [23]:
```
score = model_cnn.evaluate(X_test_cnn, Y_test_cnn, verbose = 1)
print()
print('CNN Test loss:', score[0])
print('CNN Test accuracy:', score[1])
```

```
10000/10000 [==============================] - 1s 82us/step

CNN Test loss: 0.22900302994847296
CNN Test accuracy: 0.9187
```

Results

In [24]:
```
accuracy = history_cnn.history['acc']
val_accuracy = history_cnn.history['val_acc']

loss = history_cnn.history['loss']
val_loss = history_cnn.history['val_loss']

epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'b', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





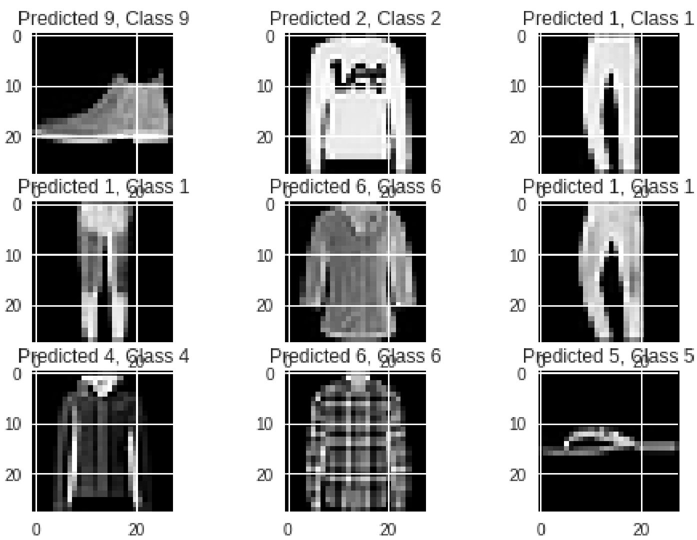
Classification Report

In [25]:
```
# Get the predictions for the test data
predicted_classes = model_cnn.predict_classes(X_test_cnn)

# Get the indices to be plotted
Y_true_cnn = Y_test
correct = np.nonzero(predicted_classes == Y_true_cnn)
incorrect = np.nonzero(predicted_classes != Y_true_cnn)

from sklearn.metrics import classification_report

target_names = ["Class {}".format(i) for i in range(num_classes)]

print(classification_report(Y_true_cnn, predicted_classes, target_names = target_names))
```

```
              precision    recall  f1-score   support

     Class 0       0.85      0.89      0.87      1000
     Class 1       0.99      0.98      0.99      1000
     Class 2       0.85      0.91      0.88      1000
     Class 3       0.91      0.93      0.92      1000
     Class 4       0.89      0.88      0.88      1000
     Class 5       0.99      0.98      0.98      1000
     Class 6       0.80      0.70      0.75      1000
     Class 7       0.94      0.99      0.96      1000
     Class 8       0.97      0.98      0.98      1000
     Class 9       0.99      0.95      0.97      1000

   micro avg       0.92      0.92      0.92     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.92      0.92      0.92     10000
```

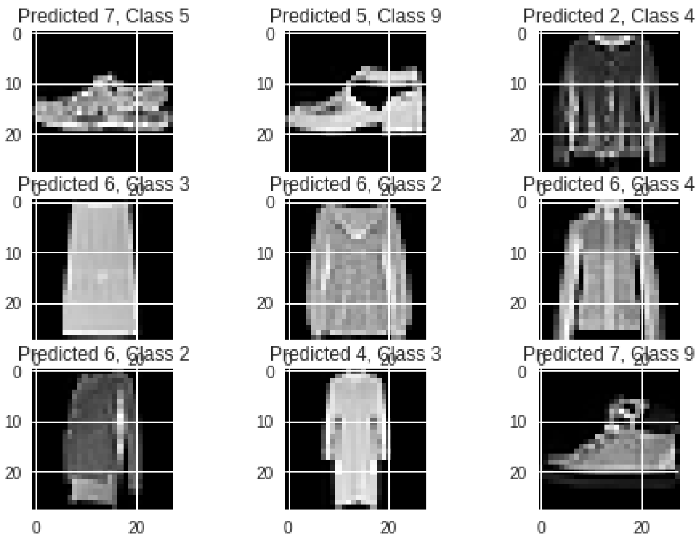
Subset of correctly predicted classes:

In [26]:
```
i=1
for j in range(9):
    plt.subplot(3,3,i)
    plt.imshow(X_test_cnn[correct[0][j]].reshape(28,28), cmap='gray', interpolation='none')
    plt.title("Predicted {}, Class {}".format(predicted_classes[correct[0][j]], Y_true_cnn[correct
[0][j]]))
    i+=1
```



Subset of incorrectly predicted classes:

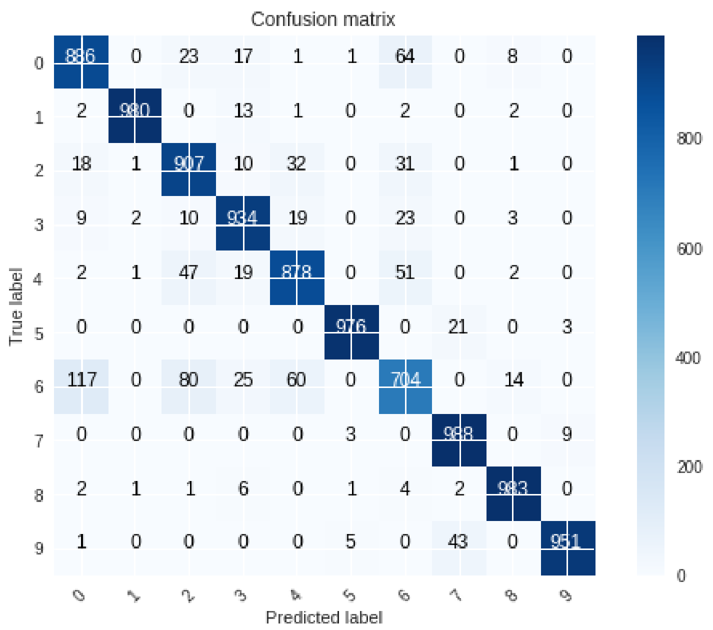
In [27]:
```
i=1
for j in range(9):
    plt.subplot(3,3,i)
    plt.imshow(X_test_cnn[incorrect[0][j]].reshape(28,28), cmap='gray', interpolation='none')
    plt.title("Predicted {}, Class {}".format(predicted_classes[incorrect[0][j]], Y_true_cnn[incor
rect[0][j]]))
    i+=1
```



It looks like diversity of the similar patterns present on multiple classes effect the performance of the classifier although CNN is a robust architechture. A jacket, a shirt, and a long-sleeve blouse has similar patterns: long sleeves (or not!), buttons (or not!), and so on.

In [0]:
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
import itertools

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')
```

In [29]:
```
# Predict the values from the validation dataset
Y_pred = model_cnn.predict(X_test_cnn)

# 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_test_cnn,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))
```



Transfer Learning

Based on our past experience, we humans can learn a new skill easily. We are more efficient in learning, particularly if the task in hand similar to what we have done have done in similar, for example learning a new programming language for a computer professional or driving a new type of vehicle for a seasoned driver is relatively easy based on our past experience.

Transfer learning is an area in machine learning that aims to utilize the knowledge gained while solving one problem to solve a different but related problem.

We will train a CNN model of two level layers i.e., a **feature layer** and a **classification layer** on the first 5 digits (0 to 4) of MNIST dataset

Apply **Transfer Learning** to **freeze features layer** and **fine-tune dense layers** for the classification of digits 5 to 9.

In [30]:
```
(X_train, Y_train), (X_test, Y_test) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

In [31]:
```
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

In [0]:
```
# Create two datasets: one with digits below 5 and one with 5 and above

X_train_lt5 = X_train[Y_train < 5]
Y_train_lt5 = Y_train[Y_train < 5]
X_test_lt5 = X_test[Y_test < 5]
Y_test_lt5 = Y_test[Y_test < 5]

X_train_gte5 = X_train[Y_train >= 5]
Y_train_gte5 = Y_train[Y_train >= 5] - 5  # make classes start at 0
X_test_gte5 = X_test[Y_test >= 5]
Y_test_gte5 = Y_test[Y_test >= 5] - 5
```

In [0]:
```
# Initialize variables
batch_size = 128
num_classes = 5 # for training stage of transfer learning
epochs = 5

# number of convolutional filters to use
num_filters = 32

# size of pooling area for max pooling
pool_size = 2

# convolution kernel size
kernel_size = 3
```

Train model for digits 0 to 4

In [0]:
```
# Function for training model for transfer learning
def train_model(model, train, test, num_classes):
    # reshape
    X_train = train[0].reshape((train[0].shape[0],) + input_shape)
    X_test = test[0].reshape((test[0].shape[0],) + input_shape)

    # convert data type
    X_train = X_train.astype('float32')
    X_test = X_test.astype('float32')

    # normalize pixel values
    X_train /= 255
    X_test /= 255

    # convert class vectors to binary class matrices
    Y_train = np_utils.to_categorical(train[1], num_classes)
    Y_test = np_utils.to_categorical(test[1], num_classes)

    # compile
    model.compile(loss='categorical_crossentropy',
                  optimizer='adadelta',
                  metrics=['accuracy'])

    # fitting of model
    model.fit(X_train, Y_train,
              batch_size=batch_size, epochs=epochs,
              verbose=1,
              validation_data=(X_test, Y_test))

    # evaluation
    score = model.evaluate(X_test, Y_test, verbose=1)

    # report scores
    print()
    print('Test score:', score[0])
    print('Test accuracy:', score[1])
    print()
```

In [35]:
```
from keras.layers import Convolution2D, MaxPooling2D
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten

# Define two groups of layers: feature (convolutions) and classification (dense)
feature_layers = [
    Convolution2D(num_filters, (kernel_size, kernel_size),
                  input_shape = input_shape,
                  padding = 'valid'),
    Activation('relu'),
    Convolution2D(num_filters, (kernel_size, kernel_size)),
    Activation('relu'),
    MaxPooling2D((pool_size, pool_size)),
    Dropout(0.25),
    Flatten(),
]

classification_layers = [
    Dense(128),
    Activation('relu'),
    Dropout(0.5),
    Dense(num_classes),
    Activation('softmax')
]

# Create complete model
model = Sequential(feature_layers + classification_layers)
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_4 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
activation_5 (Activation)    (None, 26, 26, 32)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 24, 24, 32)        9248      
_________________________________________________________________
activation_6 (Activation)    (None, 24, 24, 32)        0         
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 32)        0         
_________________________________________________________________
dropout_8 (Dropout)          (None, 12, 12, 32)        0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 4608)              0         
_________________________________________________________________
dense_9 (Dense)              (None, 128)               589952    
_________________________________________________________________
activation_7 (Activation)    (None, 128)               0         
_________________________________________________________________
dropout_9 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_10 (Dense)             (None, 5)                 645       
_________________________________________________________________
activation_8 (Activation)    (None, 5)                 0         
=================================================================
Total params: 600,165
Trainable params: 600,165
Non-trainable params: 0
_________________________________________________________________
```

In [36]:
```python
# Train model for 5-digit classification [0 to 4]
train_model(model, (X_train_lt5, Y_train_lt5), (X_test_lt5, Y_test_lt5), num_classes)
```

```
Train on 30596 samples, validate on 5139 samples
Epoch 1/5
30596/30596 [==============================] - 4s 133us/step - loss: 0.1532 - acc: 0.9510 - val_lo
ss: 0.0252 - val_acc: 0.9914
Epoch 2/5
30596/30596 [==============================] - 3s 113us/step - loss: 0.0433 - acc: 0.9869 - val_lo
ss: 0.0136 - val_acc: 0.9955
Epoch 3/5
30596/30596 [==============================] - 3s 110us/step - loss: 0.0296 - acc: 0.9910 - val_lo
ss: 0.0119 - val_acc: 0.9957
Epoch 4/5
30596/30596 [==============================] - 3s 111us/step - loss: 0.0261 - acc: 0.9923 - val_lo
ss: 0.0084 - val_acc: 0.9973
Epoch 5/5
30596/30596 [==============================] - 3s 110us/step - loss: 0.0204 - acc: 0.9940 - val_lo
ss: 0.0082 - val_acc: 0.9967
5139/5139 [==============================] - 0s 81us/step

Test score: 0.008182254664061858
Test accuracy: 0.9966919634170072
```

Transfer existing trained model on 0 to 4 to build model for digits 5 to 9

In [37]:
```python
# Freeze feature layers and rebuild model
for layer in feature_layers:
    layer.trainable = False

# Transfer: train dense layers for new classification task [5 to 9]
train_model(model, (X_train_gte5, Y_train_gte5), (X_test_gte5, Y_test_gte5), num_classes)
```

```
Train on 29404 samples, validate on 4861 samples
Epoch 1/5
29404/29404 [==============================] - 3s 90us/step - loss: 0.2423 - acc: 0.9300 - val_los
s: 0.0507 - val_acc: 0.9844
Epoch 2/5
29404/29404 [==============================] - 2s 68us/step - loss: 0.0721 - acc: 0.9778 - val_los
s: 0.0350 - val_acc: 0.9889
Epoch 3/5
29404/29404 [==============================] - 2s 68us/step - loss: 0.0561 - acc: 0.9828 - val_los
s: 0.0295 - val_acc: 0.9897
Epoch 4/5
29404/29404 [==============================] - 2s 67us/step - loss: 0.0465 - acc: 0.9855 - val_los
s: 0.0252 - val_acc: 0.9907
Epoch 5/5
29404/29404 [==============================] - 2s 69us/step - loss: 0.0408 - acc: 0.9877 - val_los
s: 0.0252 - val_acc: 0.9916
4861/4861 [==============================] - 0s 88us/step

Test score: 0.025249082497051406
Test accuracy: 0.9915655214976342
```