Deep Learning Concepts and Applications

GCE Raipur STTP, Mar 2021

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
```

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)
```

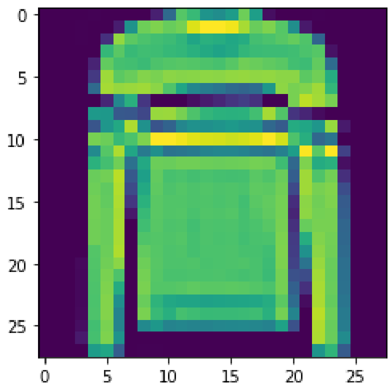
```
X_train shape: (60000, 28, 28) Y_train shape: (60000,)
```

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 0x7fca9fff09d0>
```



In [5]:
```
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 [7]:
```
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 [8]:
```
# 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 [10]:
```
# 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 [12]:
```
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()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 625)               490625
_________________________________________________________________
dense_1 (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))
```

```
Epoch 1/50
188/188 [==============================] - 4s 5ms/step - loss: 1.7770 - accuracy: 0.4724 - val_los
s: 1.0148 - val_accuracy: 0.6769
Epoch 2/50
188/188 [==============================] - 1s 3ms/step - loss: 0.9148 - accuracy: 0.7311 - val_los
s: 0.8062 - val_accuracy: 0.7266
Epoch 3/50
188/188 [==============================] - 1s 3ms/step - loss: 0.7509 - accuracy: 0.7500 - val_los
s: 0.7078 - val_accuracy: 0.7532
Epoch 4/50
188/188 [==============================] - 1s 3ms/step - loss: 0.6830 - accuracy: 0.7685 - val_los
s: 0.6616 - val_accuracy: 0.7684
Epoch 5/50
188/188 [==============================] - 1s 3ms/step - loss: 0.6475 - accuracy: 0.7761 - val_los
s: 0.6227 - val_accuracy: 0.7825
Epoch 6/50
188/188 [==============================] - 1s 3ms/step - loss: 0.6102 - accuracy: 0.7884 - val_los
s: 0.5982 - val_accuracy: 0.7918
Epoch 7/50
188/188 [==============================] - 1s 3ms/step - loss: 0.5824 - accuracy: 0.7992 - val_los
s: 0.5796 - val_accuracy: 0.7961
Epoch 8/50
188/188 [==============================] - 1s 3ms/step - loss: 0.5641 - accuracy: 0.8038 - val_los
s: 0.5686 - val_accuracy: 0.7996
Epoch 9/50
188/188 [==============================] - 1s 3ms/step - loss: 0.5505 - accuracy: 0.8066 - val_los
s: 0.5525 - val_accuracy: 0.8068
Epoch 10/50
188/188 [==============================] - 1s 3ms/step - loss: 0.5408 - accuracy: 0.8108 - val_los
s: 0.5405 - val_accuracy: 0.8100
Epoch 11/50
188/188 [==============================] - 1s 3ms/step - loss: 0.5346 - accuracy: 0.8132 - val_los
s: 0.5297 - val_accuracy: 0.8163
Epoch 12/50
188/188 [==============================] - 1s 3ms/step - loss: 0.5154 - accuracy: 0.8194 - val_los
s: 0.5224 - val_accuracy: 0.8173
Epoch 13/50
188/188 [==============================] - 1s 3ms/step - loss: 0.5058 - accuracy: 0.8243 - val_los
s: 0.5170 - val_accuracy: 0.8179
Epoch 14/50
188/188 [==============================] - 1s 3ms/step - loss: 0.5005 - accuracy: 0.8247 - val_los
s: 0.5146 - val_accuracy: 0.8200
Epoch 15/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4957 - accuracy: 0.8268 - val_los
s: 0.5078 - val_accuracy: 0.8206
Epoch 16/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4879 - accuracy: 0.8294 - val_los
s: 0.5037 - val_accuracy: 0.8229
Epoch 17/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4846 - accuracy: 0.8297 - val_los
s: 0.4950 - val_accuracy: 0.8263
Epoch 18/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4821 - accuracy: 0.8318 - val_los
s: 0.4946 - val_accuracy: 0.8267
Epoch 19/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4707 - accuracy: 0.8350 - val_los
s: 0.4883 - val_accuracy: 0.8283
Epoch 20/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4721 - accuracy: 0.8358 - val_los
s: 0.4827 - val_accuracy: 0.8303
Epoch 21/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4729 - accuracy: 0.8334 - val_los
s: 0.4830 - val_accuracy: 0.8326
Epoch 22/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4641 - accuracy: 0.8365 - val_los
s: 0.4863 - val_accuracy: 0.8245
Epoch 23/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4543 - accuracy: 0.8415 - val_los
s: 0.4708 - val_accuracy: 0.8357
Epoch 24/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4602 - accuracy: 0.8386 - val_los
s: 0.4679 - val_accuracy: 0.8332
Epoch 25/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4536 - accuracy: 0.8419 - val_los
s: 0.4643 - val_accuracy: 0.8363
Epoch 26/50
```

```
188/188 [==============================] - 1s 3ms/step - loss: 0.4548 - accuracy: 0.8441 - val_los
s: 0.4627 - val_accuracy: 0.8354
Epoch 27/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4477 - accuracy: 0.8437 - val_los
s: 0.4616 - val_accuracy: 0.8385
Epoch 28/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4454 - accuracy: 0.8452 - val_los
s: 0.4589 - val_accuracy: 0.8380
Epoch 29/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4439 - accuracy: 0.8457 - val_los
s: 0.4562 - val_accuracy: 0.8408
Epoch 30/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4400 - accuracy: 0.8466 - val_los
s: 0.4564 - val_accuracy: 0.8382
Epoch 31/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4403 - accuracy: 0.8454 - val_los
s: 0.4535 - val_accuracy: 0.8393
Epoch 32/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4452 - accuracy: 0.8438 - val_los
s: 0.4582 - val_accuracy: 0.8377
Epoch 33/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4392 - accuracy: 0.8452 - val_los
s: 0.4480 - val_accuracy: 0.8408
Epoch 34/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4363 - accuracy: 0.8473 - val_los
s: 0.4470 - val_accuracy: 0.8405
Epoch 35/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4346 - accuracy: 0.8470 - val_los
s: 0.4499 - val_accuracy: 0.8399
Epoch 36/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4363 - accuracy: 0.8475 - val_los
s: 0.4481 - val_accuracy: 0.8407
Epoch 37/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4340 - accuracy: 0.8484 - val_los
s: 0.4409 - val_accuracy: 0.8423
Epoch 38/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4286 - accuracy: 0.8500 - val_los
s: 0.4460 - val_accuracy: 0.8426
Epoch 39/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4254 - accuracy: 0.8520 - val_los
s: 0.4429 - val_accuracy: 0.8402
Epoch 40/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4266 - accuracy: 0.8522 - val_los
s: 0.4379 - val_accuracy: 0.8428
Epoch 41/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4274 - accuracy: 0.8501 - val_los
s: 0.4389 - val_accuracy: 0.8430
Epoch 42/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4251 - accuracy: 0.8519 - val_los
s: 0.4456 - val_accuracy: 0.8421
Epoch 43/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4228 - accuracy: 0.8536 - val_los
s: 0.4359 - val_accuracy: 0.8432
Epoch 44/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4143 - accuracy: 0.8556 - val_los
s: 0.4361 - val_accuracy: 0.8438
Epoch 45/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4229 - accuracy: 0.8510 - val_los
s: 0.4507 - val_accuracy: 0.8382
Epoch 46/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4124 - accuracy: 0.8553 - val_los
s: 0.4381 - val_accuracy: 0.8423
Epoch 47/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4095 - accuracy: 0.8575 - val_los
s: 0.4300 - val_accuracy: 0.8465
Epoch 48/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4125 - accuracy: 0.8558 - val_los
s: 0.4306 - val_accuracy: 0.8469
Epoch 49/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4107 - accuracy: 0.8582 - val_los
s: 0.4321 - val_accuracy: 0.8437
Epoch 50/50
188/188 [==============================] - 1s 3ms/step - loss: 0.4175 - accuracy: 0.8547 - val_los
s: 0.4346 - val_accuracy: 0.8418
```

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])
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.4583 - accuracy: 0.8335

MLP Test loss: 0.45833271741867065
MLP Test accuracy: 0.8335000276565552
```

Deep Multi Layer Perceptron

In [16]:
```
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()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 625)               490625
_________________________________________________________________
activation (Activation)      (None, 625)               0
_________________________________________________________________
dropout (Dropout)            (None, 625)               0
_________________________________________________________________
dense_3 (Dense)              (None, 625)               391250
_________________________________________________________________
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, 10)                6260
_________________________________________________________________
activation_3 (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))
```

```
Epoch 1/50
188/188 [==============================] - 1s 5ms/step - loss: 0.9659 - accuracy: 0.6651 - val_los
s: 0.4769 - val_accuracy: 0.8237
Epoch 2/50
188/188 [==============================] - 1s 4ms/step - loss: 0.4639 - accuracy: 0.8281 - val_los
s: 0.4019 - val_accuracy: 0.8542
Epoch 3/50
188/188 [==============================] - 1s 4ms/step - loss: 0.4035 - accuracy: 0.8525 - val_los
s: 0.3939 - val_accuracy: 0.8557
Epoch 4/50
188/188 [==============================] - 1s 4ms/step - loss: 0.3795 - accuracy: 0.8616 - val_los
s: 0.3630 - val_accuracy: 0.8687
Epoch 5/50
188/188 [==============================] - 1s 4ms/step - loss: 0.3413 - accuracy: 0.8751 - val_los
s: 0.5285 - val_accuracy: 0.8321
Epoch 6/50
188/188 [==============================] - 1s 4ms/step - loss: 0.3404 - accuracy: 0.8732 - val_los
s: 0.3483 - val_accuracy: 0.8760
Epoch 7/50
188/188 [==============================] - 1s 4ms/step - loss: 0.3211 - accuracy: 0.8822 - val_los
s: 0.3584 - val_accuracy: 0.8686
Epoch 8/50
188/188 [==============================] - 1s 4ms/step - loss: 0.3067 - accuracy: 0.8834 - val_los
s: 0.3548 - val_accuracy: 0.8777
Epoch 9/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2954 - accuracy: 0.8900 - val_los
s: 0.3719 - val_accuracy: 0.8791
Epoch 10/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2906 - accuracy: 0.8930 - val_los
s: 0.3684 - val_accuracy: 0.8731
Epoch 11/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2861 - accuracy: 0.8948 - val_los
s: 0.3857 - val_accuracy: 0.8662
Epoch 12/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2850 - accuracy: 0.8925 - val_los
s: 0.4125 - val_accuracy: 0.8640
Epoch 13/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2818 - accuracy: 0.8961 - val_los
s: 0.3664 - val_accuracy: 0.8752
Epoch 14/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2789 - accuracy: 0.8974 - val_los
s: 0.3801 - val_accuracy: 0.8762
Epoch 15/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2724 - accuracy: 0.8990 - val_los
s: 0.4090 - val_accuracy: 0.8723
Epoch 16/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2703 - accuracy: 0.9016 - val_los
s: 0.4151 - val_accuracy: 0.8819
Epoch 17/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2693 - accuracy: 0.8990 - val_los
s: 0.3886 - val_accuracy: 0.8802
Epoch 18/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2671 - accuracy: 0.9014 - val_los
s: 0.4346 - val_accuracy: 0.8840
Epoch 19/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2633 - accuracy: 0.9052 - val_los
s: 0.3891 - val_accuracy: 0.8930
Epoch 20/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2513 - accuracy: 0.9092 - val_los
s: 0.3949 - val_accuracy: 0.8841
Epoch 21/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2508 - accuracy: 0.9087 - val_los
s: 0.4253 - val_accuracy: 0.8814
Epoch 22/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2501 - accuracy: 0.9097 - val_los
s: 0.4527 - val_accuracy: 0.8860
Epoch 23/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2533 - accuracy: 0.9098 - val_los
s: 0.4035 - val_accuracy: 0.8892
Epoch 24/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2465 - accuracy: 0.9114 - val_los
s: 0.4303 - val_accuracy: 0.8917
Epoch 25/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2497 - accuracy: 0.9117 - val_los
s: 0.4996 - val_accuracy: 0.8882
Epoch 26/50
```

```
188/188 [==============================] - 1s 4ms/step - loss: 0.2487 - accuracy: 0.9100 - val_los
s: 0.4108 - val_accuracy: 0.8908
Epoch 27/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2404 - accuracy: 0.9138 - val_los
s: 0.4911 - val_accuracy: 0.8730
Epoch 28/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2481 - accuracy: 0.9107 - val_los
s: 0.4596 - val_accuracy: 0.8907
Epoch 29/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2409 - accuracy: 0.9150 - val_los
s: 0.4313 - val_accuracy: 0.8723
Epoch 30/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2540 - accuracy: 0.9143 - val_los
s: 0.4751 - val_accuracy: 0.8806
Epoch 31/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2347 - accuracy: 0.9155 - val_los
s: 0.4183 - val_accuracy: 0.8878
Epoch 32/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2353 - accuracy: 0.9155 - val_los
s: 0.4518 - val_accuracy: 0.8863
Epoch 33/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2343 - accuracy: 0.9171 - val_los
s: 0.5363 - val_accuracy: 0.8877
Epoch 34/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2385 - accuracy: 0.9177 - val_los
s: 0.4669 - val_accuracy: 0.8863
Epoch 35/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2291 - accuracy: 0.9170 - val_los
s: 0.4700 - val_accuracy: 0.8908
Epoch 36/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2239 - accuracy: 0.9199 - val_los
s: 0.5012 - val_accuracy: 0.8845
Epoch 37/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2186 - accuracy: 0.9215 - val_los
s: 0.5193 - val_accuracy: 0.8904
Epoch 38/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2318 - accuracy: 0.9172 - val_los
s: 0.4736 - val_accuracy: 0.8902
Epoch 39/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2278 - accuracy: 0.9232 - val_los
s: 0.5255 - val_accuracy: 0.8907
Epoch 40/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2167 - accuracy: 0.9214 - val_los
s: 0.5565 - val_accuracy: 0.8767
Epoch 41/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2217 - accuracy: 0.9220 - val_los
s: 0.6791 - val_accuracy: 0.8903
Epoch 42/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2267 - accuracy: 0.9233 - val_los
s: 0.4967 - val_accuracy: 0.8916
Epoch 43/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2093 - accuracy: 0.9236 - val_los
s: 0.5142 - val_accuracy: 0.8907
Epoch 44/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2150 - accuracy: 0.9254 - val_los
s: 0.4983 - val_accuracy: 0.8845
Epoch 45/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2090 - accuracy: 0.9259 - val_los
s: 0.5386 - val_accuracy: 0.8932
Epoch 46/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2207 - accuracy: 0.9245 - val_los
s: 0.5312 - val_accuracy: 0.8902
Epoch 47/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2126 - accuracy: 0.9260 - val_los
s: 0.5300 - val_accuracy: 0.8955
Epoch 48/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2157 - accuracy: 0.9268 - val_los
s: 0.4838 - val_accuracy: 0.8953
Epoch 49/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2088 - accuracy: 0.9254 - val_los
s: 0.6550 - val_accuracy: 0.8887
Epoch 50/50
188/188 [==============================] - 1s 4ms/step - loss: 0.2136 - accuracy: 0.9262 - val_los
s: 0.5736 - val_accuracy: 0.8894
```

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])
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.6697 - accuracy: 0.8838

Deep MLP Test loss: 0.6696637272834778
Deep MLP Test accuracy: 0.8838000297546387
```

Convolutional Neural Networks

In [20]:
```
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()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 13, 13, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 5, 5, 64)          0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 128)         73856     
_________________________________________________________________
dropout_5 (Dropout)          (None, 3, 3, 128)         0         
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0         
_________________________________________________________________
dense_6 (Dense)              (None, 128)               147584    
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_7 (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))
```

```
Epoch 1/50
188/188 [==============================] - 32s 10ms/step - loss: 1.2897 - accuracy: 0.5236 - val_l
oss: 0.5645 - val_accuracy: 0.7825
Epoch 2/50
188/188 [==============================] - 2s 8ms/step - loss: 0.5822 - accuracy: 0.7792 - val_los
s: 0.4508 - val_accuracy: 0.8296
Epoch 3/50
188/188 [==============================] - 2s 8ms/step - loss: 0.4945 - accuracy: 0.8129 - val_los
s: 0.4000 - val_accuracy: 0.8555
Epoch 4/50
188/188 [==============================] - 2s 8ms/step - loss: 0.4396 - accuracy: 0.8388 - val_los
s: 0.3625 - val_accuracy: 0.8682
Epoch 5/50
188/188 [==============================] - 2s 8ms/step - loss: 0.4044 - accuracy: 0.8528 - val_los
s: 0.3414 - val_accuracy: 0.8758
Epoch 6/50
188/188 [==============================] - 2s 8ms/step - loss: 0.3787 - accuracy: 0.8590 - val_los
s: 0.3198 - val_accuracy: 0.8836
Epoch 7/50
188/188 [==============================] - 2s 9ms/step - loss: 0.3575 - accuracy: 0.8699 - val_los
s: 0.3064 - val_accuracy: 0.8881
Epoch 8/50
188/188 [==============================] - 2s 8ms/step - loss: 0.3485 - accuracy: 0.8713 - val_los
s: 0.3043 - val_accuracy: 0.8874
Epoch 9/50
188/188 [==============================] - 2s 9ms/step - loss: 0.3326 - accuracy: 0.8782 - val_los
s: 0.2895 - val_accuracy: 0.8940
Epoch 10/50
188/188 [==============================] - 2s 9ms/step - loss: 0.3174 - accuracy: 0.8826 - val_los
s: 0.2898 - val_accuracy: 0.8946
Epoch 11/50
188/188 [==============================] - 2s 8ms/step - loss: 0.3089 - accuracy: 0.8867 - val_los
s: 0.2747 - val_accuracy: 0.8983
Epoch 12/50
188/188 [==============================] - 2s 8ms/step - loss: 0.3040 - accuracy: 0.8889 - val_los
s: 0.2677 - val_accuracy: 0.9025
Epoch 13/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2946 - accuracy: 0.8921 - val_los
s: 0.2596 - val_accuracy: 0.9061
Epoch 14/50
188/188 [==============================] - 2s 8ms/step - loss: 0.2792 - accuracy: 0.8972 - val_los
s: 0.2595 - val_accuracy: 0.9052
Epoch 15/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2756 - accuracy: 0.8988 - val_los
s: 0.2566 - val_accuracy: 0.9051
Epoch 16/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2746 - accuracy: 0.8990 - val_los
s: 0.2542 - val_accuracy: 0.9073
Epoch 17/50
188/188 [==============================] - 2s 8ms/step - loss: 0.2643 - accuracy: 0.9028 - val_los
s: 0.2628 - val_accuracy: 0.9013
Epoch 18/50
188/188 [==============================] - 2s 8ms/step - loss: 0.2714 - accuracy: 0.8995 - val_los
s: 0.2510 - val_accuracy: 0.9082
Epoch 19/50
188/188 [==============================] - 2s 8ms/step - loss: 0.2618 - accuracy: 0.9035 - val_los
s: 0.2445 - val_accuracy: 0.9109
Epoch 20/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2568 - accuracy: 0.9045 - val_los
s: 0.2414 - val_accuracy: 0.9120
Epoch 21/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2471 - accuracy: 0.9081 - val_los
s: 0.2365 - val_accuracy: 0.9139
Epoch 22/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2473 - accuracy: 0.9076 - val_los
s: 0.2354 - val_accuracy: 0.9135
Epoch 23/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2462 - accuracy: 0.9078 - val_los
s: 0.2451 - val_accuracy: 0.9104
Epoch 24/50
188/188 [==============================] - 2s 8ms/step - loss: 0.2402 - accuracy: 0.9111 - val_los
s: 0.2340 - val_accuracy: 0.9133
Epoch 25/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2406 - accuracy: 0.9106 - val_los
s: 0.2386 - val_accuracy: 0.9113
Epoch 26/50
```

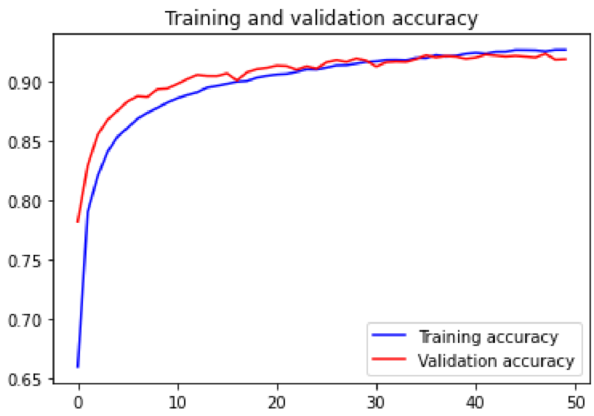
```
188/188 [==============================] - 2s 9ms/step - loss: 0.2284 - accuracy: 0.9131 - val_los
s: 0.2310 - val_accuracy: 0.9168
Epoch 27/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2244 - accuracy: 0.9175 - val_los
s: 0.2287 - val_accuracy: 0.9186
Epoch 28/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2227 - accuracy: 0.9163 - val_los
s: 0.2305 - val_accuracy: 0.9171
Epoch 29/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2192 - accuracy: 0.9166 - val_los
s: 0.2236 - val_accuracy: 0.9200
Epoch 30/50
188/188 [==============================] - 2s 8ms/step - loss: 0.2264 - accuracy: 0.9162 - val_los
s: 0.2250 - val_accuracy: 0.9182
Epoch 31/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2151 - accuracy: 0.9203 - val_los
s: 0.2378 - val_accuracy: 0.9131
Epoch 32/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2202 - accuracy: 0.9171 - val_los
s: 0.2255 - val_accuracy: 0.9168
Epoch 33/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2046 - accuracy: 0.9229 - val_los
s: 0.2265 - val_accuracy: 0.9172
Epoch 34/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2099 - accuracy: 0.9201 - val_los
s: 0.2252 - val_accuracy: 0.9170
Epoch 35/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2103 - accuracy: 0.9217 - val_los
s: 0.2225 - val_accuracy: 0.9193
Epoch 36/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2120 - accuracy: 0.9222 - val_los
s: 0.2220 - val_accuracy: 0.9227
Epoch 37/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2140 - accuracy: 0.9200 - val_los
s: 0.2230 - val_accuracy: 0.9208
Epoch 38/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2059 - accuracy: 0.9226 - val_los
s: 0.2182 - val_accuracy: 0.9220
Epoch 39/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2041 - accuracy: 0.9220 - val_los
s: 0.2210 - val_accuracy: 0.9215
Epoch 40/50
188/188 [==============================] - 2s 9ms/step - loss: 0.2019 - accuracy: 0.9232 - val_los
s: 0.2198 - val_accuracy: 0.9197
Epoch 41/50
188/188 [==============================] - 2s 9ms/step - loss: 0.1998 - accuracy: 0.9238 - val_los
s: 0.2187 - val_accuracy: 0.9207
Epoch 42/50
188/188 [==============================] - 2s 9ms/step - loss: 0.1969 - accuracy: 0.9249 - val_los
s: 0.2174 - val_accuracy: 0.9233
Epoch 43/50
188/188 [==============================] - 2s 9ms/step - loss: 0.1949 - accuracy: 0.9265 - val_los
s: 0.2180 - val_accuracy: 0.9224
Epoch 44/50
188/188 [==============================] - 2s 9ms/step - loss: 0.1979 - accuracy: 0.9250 - val_los
s: 0.2195 - val_accuracy: 0.9216
Epoch 45/50
188/188 [==============================] - 2s 9ms/step - loss: 0.1899 - accuracy: 0.9287 - val_los
s: 0.2187 - val_accuracy: 0.9222
Epoch 46/50
188/188 [==============================] - 2s 9ms/step - loss: 0.1930 - accuracy: 0.9266 - val_los
s: 0.2167 - val_accuracy: 0.9215
Epoch 47/50
188/188 [==============================] - 2s 9ms/step - loss: 0.1868 - accuracy: 0.9275 - val_los
s: 0.2224 - val_accuracy: 0.9208
Epoch 48/50
188/188 [==============================] - 2s 9ms/step - loss: 0.1914 - accuracy: 0.9261 - val_los
s: 0.2125 - val_accuracy: 0.9240
Epoch 49/50
188/188 [==============================] - 2s 9ms/step - loss: 0.1881 - accuracy: 0.9296 - val_los
s: 0.2230 - val_accuracy: 0.9189
Epoch 50/50
188/188 [==============================] - 2s 9ms/step - loss: 0.1915 - accuracy: 0.9266 - val_los
s: 0.2259 - val_accuracy: 0.9193
```

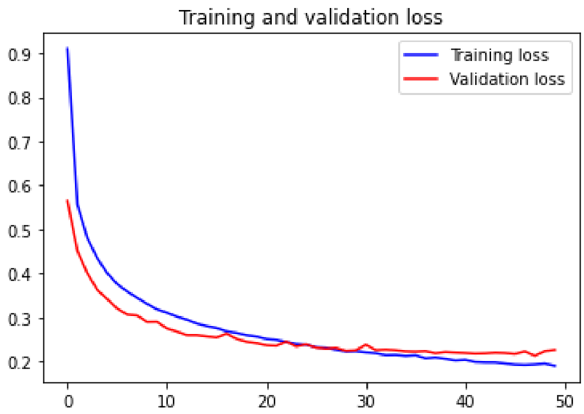
In [23]:
```
score = model_cnn.evaluate(X_test_cnn, Y_test_cnn, verbose = 1)
print()
print('CNN Test loss:', score[0])
print('CNN Test accuracy:', score[1])
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.2348 - accuracy: 0.9145

CNN Test loss: 0.23478259146213531
CNN Test accuracy: 0.9144999980926514
```

Results

In [25]:
```
accuracy = history_cnn.history['accuracy']
val_accuracy = history_cnn.history['val_accuracy']

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()
```





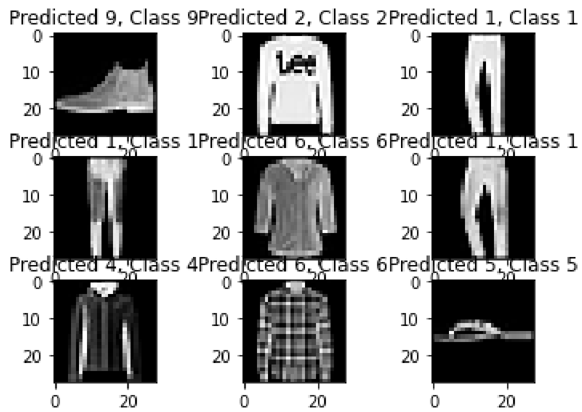
Classification Report

In [26]:
```python
# 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))
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserWarni
ng: `model.predict_classes()` is deprecated and will be removed after 2021-01-01. Please use inste
ad:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.
g. if it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   i
f your model does binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '

              precision    recall  f1-score   support

     Class 0       0.84      0.88      0.86      1000
     Class 1       1.00      0.98      0.99      1000
     Class 2       0.83      0.92      0.87      1000
     Class 3       0.91      0.94      0.92      1000
     Class 4       0.88      0.85      0.87      1000
     Class 5       0.99      0.97      0.98      1000
     Class 6       0.79      0.69      0.74      1000
     Class 7       0.95      0.98      0.96      1000
     Class 8       0.98      0.98      0.98      1000
     Class 9       0.97      0.96      0.97      1000

    accuracy                           0.91     10000
   macro avg       0.91      0.91      0.91     10000
weighted avg       0.91      0.91      0.91     10000
```

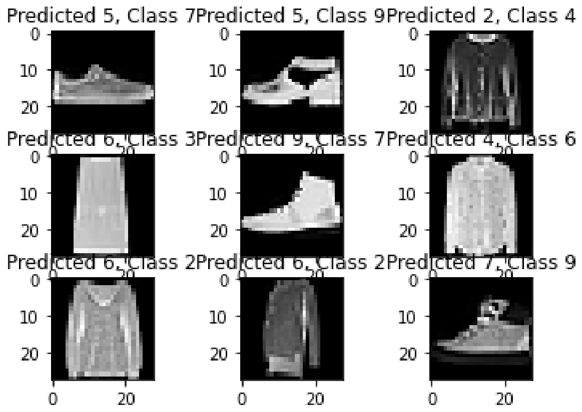
Subset of correctly predicted classes:

In [27]:
```python
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 [28]:
```
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 [29]:
```
# 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 [30]:
```
# 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))
```