In [16]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Input,Convolution2D,MaxPooling2D,Flatten,Dense,Dropout
from keras.utils import np_utils
import tensorflow
```

Data Preparation

In [22]:

```
x = pd.read_csv("fashion-mnist_train.csv")
X_ = np.array(x)
X = X_[:,1:]
X = X/255.0
Y = X_[:,0]
print(X.shape,Y.shape)
```

```
(60000, 784) (60000,)
```

In [23]:

```
np.unique(Y,return_counts=True) # Balanced split
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=int64),
 array([6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000],
       dtype=int64))
```

In [24]:

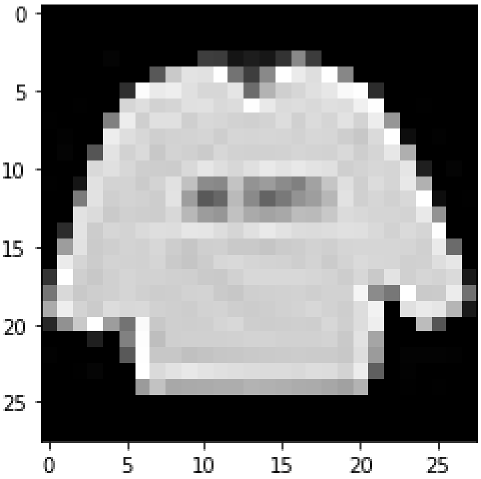
```
X_Train = X.reshape((-1,28,28,1))  # Gray Scale Image
Y_Train = np_utils.to_categorical(Y)

print(X_Train.shape, Y_Train.shape)
```

```
(60000, 28, 28, 1) (60000, 10)
```

In [25]:

```
for i in range(10):
    plt.imshow(X_Train[i].reshape(28,28),cmap="gray")
    plt.show()
```



CNN Model

In [26]:

```
model = Sequential()
model.add(Convolution2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
model.add(Convolution2D(64,(3,3),activation='relu'))
model.add(Dropout(0.25))
model.add(MaxPooling2D(2,2))
model.add(Convolution2D(32,(5,5),activation='relu'))
model.add(Convolution2D(8,(5,5),activation='relu'))
model.add(Flatten())
model.add(Dense(10,activation='softmax'))
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_6 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 24, 24, 64)        18496     
_________________________________________________________________
dropout_2 (Dropout)          (None, 24, 24, 64)        0         
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 64)        0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 8, 8, 32)          51232     
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 4, 4, 8)           6408      
_________________________________________________________________
flatten_2 (Flatten)          (None, 128)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290      
=================================================================
Total params: 77,746
Trainable params: 77,746
Non-trainable params: 0
_________________________________________________________________
```

In [27]:

```
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

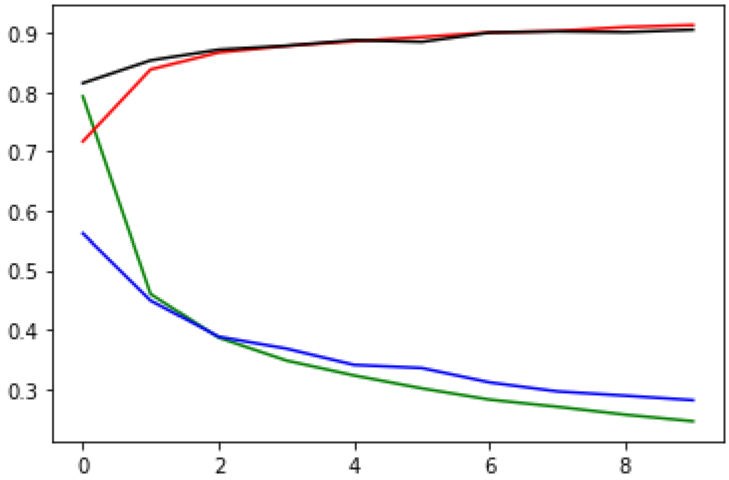
In [28]:

```
hist = model.fit(X_Train,Y_Train,epochs=10,shuffle=True,batch_size=256,validation_spli
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 [==============================] - 195s 4ms/step - loss: 0.7925 - accuracy: 0.7167 - val_loss: 0.5
Epoch 2/10
48000/48000 [==============================] - 188s 4ms/step - loss: 0.4602 - accuracy: 0.8379 - val_loss: 0.4
Epoch 3/10
48000/48000 [==============================] - 186s 4ms/step - loss: 0.3866 - accuracy: 0.8666 - val_loss: 0.3
Epoch 4/10
48000/48000 [==============================] - 187s 4ms/step - loss: 0.3480 - accuracy: 0.8776 - val_loss: 0.3
Epoch 5/10
48000/48000 [==============================] - 185s 4ms/step - loss: 0.3229 - accuracy: 0.8855 - val_loss: 0.3
Epoch 6/10
48000/48000 [==============================] - 183s 4ms/step - loss: 0.3011 - accuracy: 0.8927 - val_loss: 0.3
Epoch 7/10
48000/48000 [==============================] - 187s 4ms/step - loss: 0.2822 - accuracy: 0.9002 - val_loss: 0.3
Epoch 8/10
48000/48000 [==============================] - 185s 4ms/step - loss: 0.2701 - accuracy: 0.9029 - val_loss: 0.2
Epoch 9/10
48000/48000 [==============================] - 182s 4ms/step - loss: 0.2568 - accuracy: 0.9094 - val_loss: 0.2
Epoch 10/10
48000/48000 [==============================] - 178s 4ms/step - loss: 0.2457 - accuracy: 0.9129 - val_loss: 0.2
```

In [30]:

```
plt.plot(hist.history['loss'],'g')
plt.plot(hist.history['val_loss'],'b')
plt.plot(hist.history['accuracy'],'r')
plt.plot(hist.history['val_accuracy'],'black')
plt.show()
```



In [31]:

```
xt = pd.read_csv("fashion-mnist_test.csv")
Xt_ = np.array(xt)
Xt = Xt_[:,1:]
Xt = Xt/255.0
Yt = Xt_[:,0]
print(Xt.shape,Yt.shape)
```

```
(10000, 784) (10000,)
```

In [32]:

```
X_Test = Xt.reshape((-1,28,28,1))  # Gray Scale Image
Y_Test = np_utils.to_categorical(Yt)

print(X_Test.shape, Y_Test.shape)
```

```
(10000, 28, 28, 1) (10000, 10)
```

In [33]:

```
pred = model.predict(X_Test)
```

In [34]:

```
print(pred.shape)
```

```
(10000, 10)
```

In [35]:

```
ans = []
for i in pred:
    ans.append(np.argmax(i))
```

In [38]:

```
pred_op = np.array(ans)
print(pred_op.shape)
```

```
(10000,)
```

In [40]:

```
acc = np.sum(pred_op==Yt)/Yt.shape[0]
print(acc)
```

```
0.9084
```

In [41]:

```
from sklearn.metrics import confusion_matrix
from visualizes import plot_confusion_matrix
```

In [42]:

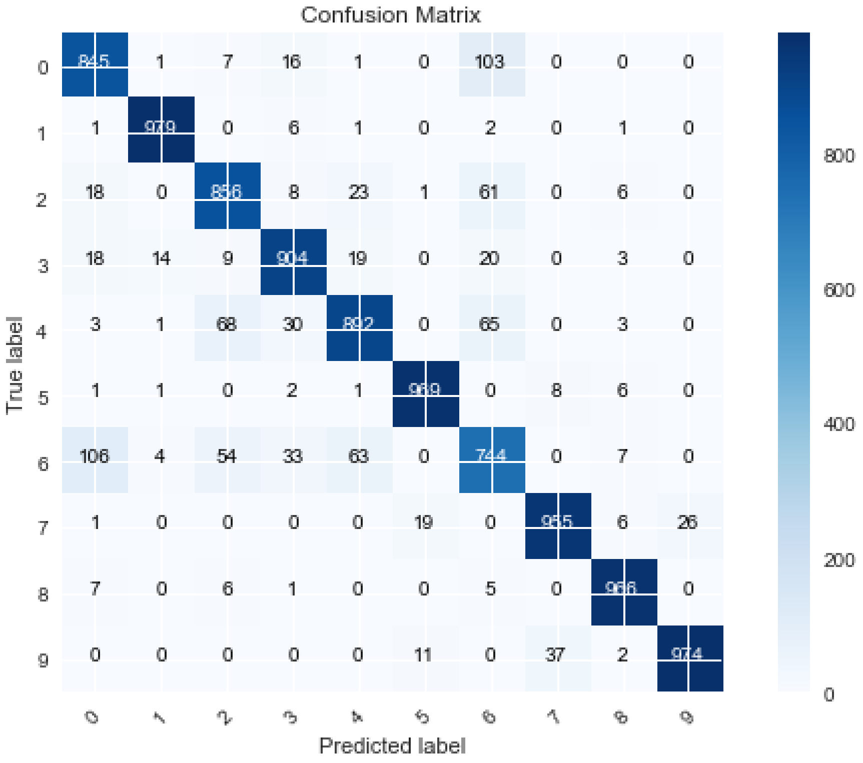
```
cnf_matrix = confusion_matrix(pred_op,Yt)
print(cnf_matrix)
```

```
[[845   1   7  16   1   0 103   0   0   0]
 [  1 979   0   6   1   0   2   0   1   0]
 [ 18   0 856   8  23   1  61   0   6   0]
 [ 18  14   9 904  19   0  20   0   3   0]
 [  3   1  68  30 892   0  65   0   3   0]
 [  1   1   0   2   1 969   0   8   6   0]
 [106   4  54  33  63   0 744   0   7   0]
 [  1   0   0   0   0  19   0 955   6  26]
 [  7   0   6   1   0   0   5   0 966   0]
 [  0   0   0   0   0  11   0  37   2 974]]
```

In [44]:

```
plt.style.use('seaborn')
plot_confusion_matrix(cnf_matrix,classes=[0,1,2,3,4,5,6,7,8,9],title="Confusion Matrix'
```

```
Confusion matrix, without normalization
[[845   1   7  16   1   0 103   0   0   0]
 [  1 979   0   6   1   0   2   0   1   0]
 [ 18   0 856   8  23   1  61   0   6   0]
 [ 18  14   9 904  19   0  20   0   3   0]
 [  3   1  68  30 892   0  65   0   3   0]
 [  1   1   0   2   1 969   0   8   6   0]
 [106   4  54  33  63   0 744   0   7   0]
 [  1   0   0   0   0  19   0 955   6  26]
 [  7   0   6   1   0   0   5   0 966   0]
 [  0   0   0   0   0  11   0  37   2 974]]
```



In [ ]: