In [1]:
```
#Setting up GPU for training
import os
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"]="0" #model will be trained on GPU 0 NVIDIA RTX 3060
```

In [2]:
```
import keras
from matplotlib import pyplot as plt
import numpy as np
import gzip
%matplotlib inline
from keras.models import Model
from keras.optimizers import RMSprop
from keras.layers import Input,Dense,Flatten,Dropout,Reshape,Conv2D,MaxPooling2D,UpSam
from keras.layers import add, concatenate, merging
from keras.models import Model,Sequential
from keras.callbacks import ModelCheckpoint
from keras.optimizers import Adadelta, RMSprop,SGD,Adam
from keras import regularizers
from keras import backend as K
from keras.utils import to_categorical
```

Extracting the MNIST Fashion dataset

1. Extractin the dataset.
2. reading into buffer
3. convertinf to tuple of (no_of_imgs, widht,height, channel) (60k,28,28)

In [3]:
```
def extract_data(filename, num_images):
    with gzip.open(filename) as bytestream:
        bytestream.read(16)
        buf = bytestream.read(28 * 28 * num_images)
        data = np.frombuffer(buf, dtype=np.uint8).astype(np.float32)
        data = data.reshape(num_images, 28,28)
        return data
```

In [4]:
```
train_data = extract_data('train-images-idx3-ubyte.gz', 60000)
test_data = extract_data('t10k-images-idx3-ubyte.gz', 10000)
```

In [5]:
```
def extract_labels(filename, num_images):
    with gzip.open(filename) as bytestream:
        bytestream.read(8)
        buf = bytestream.read(1 * num_images)
        labels = np.frombuffer(buf, dtype=np.uint8).astype(np.int64)
        return labels
```

In [6]:
```
train_labels = extract_labels('train-labels-idx1-ubyte.gz',60000)
test_labels = extract_labels('t10k-labels-idx1-ubyte.gz',10000)
```

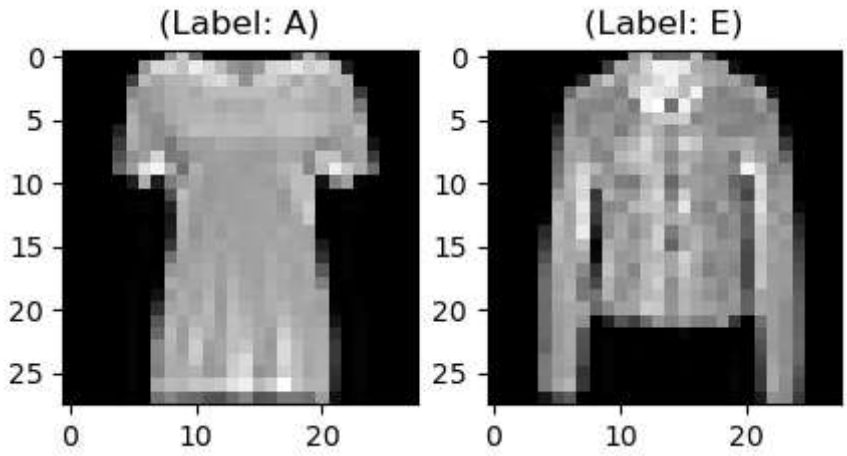
In [7]:
```
print("Training set (images) shape: {shape}".format(shape=train_data.shape))
print("Test set (images) shape: {shape}".format(shape=test_data.shape))
```

```
Training set (images) shape: (60000, 28, 28)
Test set (images) shape: (10000, 28, 28)
```

In [9]:
```
# Create dictionary of target classes
label_dict = {
 0: 'A',
 1: 'B',
 2: 'C',
 3: 'D',
 4: 'E',
 5: 'F',
 6: 'G',
 7: 'H',
 8: 'I',
 9: 'J',
}
```

In [10]:
```
plt.figure(figsize=[5,5])
plt.subplot(121)
curr_img = np.reshape(train_data[10], (28,28))
curr_lbl = train_labels[10]
plt.imshow(curr_img, cmap='gray')
plt.title("(Label: " + str(label_dict[curr_lbl]) + ")")


plt.subplot(122)
curr_img = np.reshape(test_data[10], (28,28))
curr_lbl = test_labels[10]
plt.imshow(curr_img, cmap='gray')
plt.title("(Label: " + str(label_dict[curr_lbl]) + ")")
```

Out[10]:
```
Text(0.5, 1.0, '(Label: E)')
```



In [11]:
```
# reshaping the dataset with no_of_imgs, width, height, channel
train_data = train_data.reshape(-1, 28,28, 1)
test_data = test_data.reshape(-1, 28,28, 1)
train_data.shape, test_data.shape
```

Out[11]:
```
((60000, 28, 28, 1), (10000, 28, 28, 1))
```

In [12]:
```
# normalizing the dataset
train_data.dtype, test_data.dtype
```

Out[12]:
```
(dtype('float32'), dtype('float32'))
```

In [13]:
```
np.max(train_data), np.max(test_data)
```

Out[13]:
```
(255.0, 255.0)
```

In [14]:
```
train_data = train_data / np.max(train_data)
test_data = test_data / np.max(test_data)
```

In [15]:
```
np.max(train_data), np.max(test_data)
```

Out[15]:
```
(1.0, 1.0)
```

In [16]:
```
#splitting the dataset for train validation
from sklearn.model_selection import train_test_split
train_X,valid_X,train_ground,valid_ground = train_test_split(train_data,
                                                             train_data,
                                                             test_size=0.2,
                                                             random_state=13)
```

In [21]:
```
batch_size = 64
epochs = 5
inChannel = 1
x, y = 28, 28
input_img = Input(shape = (x, y, inChannel))
num_classes = 10
```

In [18]:
```
def encoder(input_img):
    #input = 28 x 28 x 1 (wide and thin)
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img) #28 x 28
    conv1 = BatchNormalization()(conv1)
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv1)
    conv1 = BatchNormalization()(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1) #14 x 14 x 32
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(pool1) #14 x 14 x 64
    conv2 = BatchNormalization()(conv2)
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv2)
    conv2 = BatchNormalization()(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2) #7 x 7 x 64
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool2) #7 x 7 x 128
    conv3 = BatchNormalization()(conv3)
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv3)
    conv3 = BatchNormalization()(conv3)
    conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv3) #7 x 7 x 256
    conv4 = BatchNormalization()(conv4)
    conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv4)
    conv4 = BatchNormalization()(conv4)
    return conv4

def decoder(conv4):
    #decoder
    conv5 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv4) #7 x 7 x 128
    conv5 = BatchNormalization()(conv5)
    conv5 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv5)
    conv5 = BatchNormalization()(conv5)
    conv6 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv5) #7 x 7 x 64
    conv6 = BatchNormalization()(conv6)
    conv6 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv6)
    conv6 = BatchNormalization()(conv6)
    up1 = UpSampling2D((2,2))(conv6) #14 x 14 x 64
    conv7 = Conv2D(32, (3, 3), activation='relu', padding='same')(up1) # 14 x 14 x 32
```

```
    conv7 = BatchNormalization()(conv7)
    conv7 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv7)
    conv7 = BatchNormalization()(conv7)
    up2 = UpSampling2D((2,2))(conv7) # 28 x 28 x 32
    decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(up2) # 28 x 28 x
    return decoded
```

In [19]:
```
autoencoder = Model(input_img, decoder(encoder(input_img)))
autoencoder.compile(loss='mean_squared_error', optimizer = RMSprop())
```

In [20]:
```
autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 28, 28, 32)        320

 batch_normalization (BatchN  (None, 28, 28, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 28, 28, 32)        9248

 batch_normalization_1 (Batc  (None, 28, 28, 32)       128
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 14, 14, 64)        18496

 batch_normalization_2 (Batc  (None, 14, 14, 64)       256
 hNormalization)

 conv2d_3 (Conv2D)           (None, 14, 14, 64)        36928

 batch_normalization_3 (Batc  (None, 14, 14, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 conv2d_4 (Conv2D)           (None, 7, 7, 128)         73856

 batch_normalization_4 (Batc  (None, 7, 7, 128)        512
 hNormalization)

 conv2d_5 (Conv2D)           (None, 7, 7, 128)         147584

 batch_normalization_5 (Batc  (None, 7, 7, 128)        512
 hNormalization)

 conv2d_6 (Conv2D)           (None, 7, 7, 256)         295168

 batch_normalization_6 (Batc  (None, 7, 7, 256)        1024
 hNormalization)

 conv2d_7 (Conv2D)           (None, 7, 7, 256)         590080

 batch_normalization_7 (Batc  (None, 7, 7, 256)        1024
 hNormalization)

 conv2d_8 (Conv2D)           (None, 7, 7, 128)         295040

 batch_normalization_8 (Batc  (None, 7, 7, 128)        512
 hNormalization)

 conv2d_9 (Conv2D)           (None, 7, 7, 128)         147584

 batch_normalization_9 (Batc  (None, 7, 7, 128)        512
```

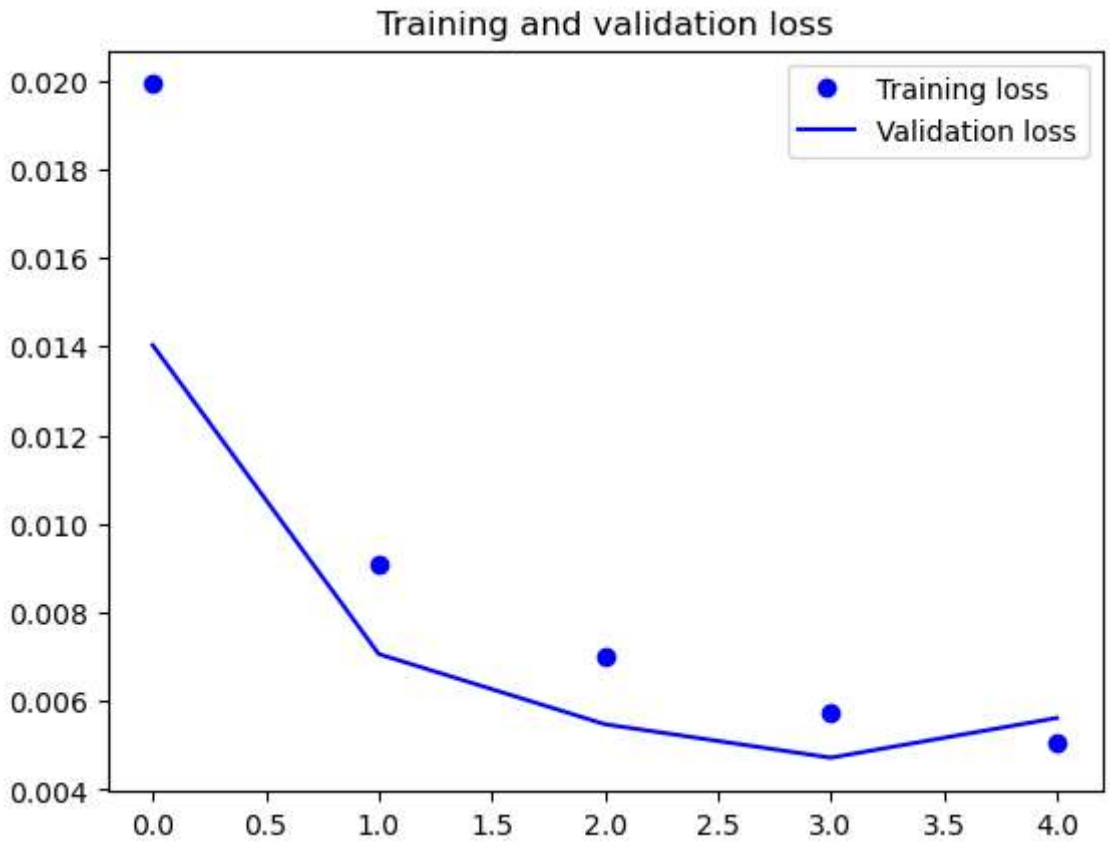
```
hNormalization)

conv2d_10 (Conv2D)          (None, 7, 7, 64)          73792

batch_normalization_10 (Bat  (None, 7, 7, 64)         256
chNormalization)

conv2d_11 (Conv2D)          (None, 7, 7, 64)          36928

batch_normalization_11 (Bat  (None, 7, 7, 64)         256
chNormalization)

up_sampling2d (UpSampling2D  (None, 14, 14, 64)       0
)

conv2d_12 (Conv2D)          (None, 14, 14, 32)        18464

batch_normalization_12 (Bat  (None, 14, 14, 32)       128
chNormalization)

conv2d_13 (Conv2D)          (None, 14, 14, 32)        9248

batch_normalization_13 (Bat  (None, 14, 14, 32)       128
chNormalization)

up_sampling2d_1 (UpSampling  (None, 28, 28, 32)       0
2D)

conv2d_14 (Conv2D)          (None, 28, 28, 1)         289

=================================================================
Total params: 1,758,657
Trainable params: 1,755,841
Non-trainable params: 2,816
_________________________________________________________________
```

In [22]:
```
autoencoder_train = autoencoder.fit(train_X, train_ground, batch_size=batch_size,epoch
```

```
Epoch 1/5
750/750 [==============================] - 414s 546ms/step - loss: 0.0199 - val_loss:
0.0140
Epoch 2/5
750/750 [==============================] - 236s 315ms/step - loss: 0.0091 - val_loss:
0.0071
Epoch 3/5
750/750 [==============================] - 234s 313ms/step - loss: 0.0070 - val_loss:
0.0055
Epoch 4/5
750/750 [==============================] - 236s 314ms/step - loss: 0.0057 - val_loss:
0.0047
Epoch 5/5
750/750 [==============================] - 234s 313ms/step - loss: 0.0050 - val_loss:
0.0056
```

In [24]:
```
loss = autoencoder_train.history['loss']
val_loss = autoencoder_train.history['val_loss']
epochs = range(5)
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
```

```
plt.title('Training and validation loss')
plt.legend()
plt.show()
```



In [25]:
```
autoencoder.save_weights('autoencoder.h5')
```

In [26]:
```
# Change the labels from categorical to one-hot encoding
train_Y_one_hot = to_categorical(train_labels)
test_Y_one_hot = to_categorical(test_labels)

print('Original label:', train_labels[0])
print('After conversion to one-hot:', train_Y_one_hot[0])
```

```
Original label: 9
After conversion to one-hot: [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
```

In [27]:
```
train_X,valid_X,train_label,valid_label = train_test_split(train_data,train_Y_one_hot,
```

In [28]:
```
train_X.shape,valid_X.shape,train_label.shape,valid_label.shape
```

Out[28]:
```
((48000, 28, 28, 1), (12000, 28, 28, 1), (48000, 10), (12000, 10))
```

In [29]:
```
# creating the classification model with same as encoder network and assigning the sam
def encoder(input_img):
    #encoder
    #input = 28 x 28 x 1 (wide and thin)
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img) #28 x 28
    conv1 = BatchNormalization()(conv1)
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv1)
    conv1 = BatchNormalization()(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1) #14 x 14 x 32
```

```
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(pool1) #14 x 14 x 64
    conv2 = BatchNormalization()(conv2)
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv2)
    conv2 = BatchNormalization()(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2) #7 x 7 x 64
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool2) #7 x 7 x 128
    conv3 = BatchNormalization()(conv3)
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv3)
    conv3 = BatchNormalization()(conv3)
    conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv3) #7 x 7 x 256
    conv4 = BatchNormalization()(conv4)
    conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv4)
    conv4 = BatchNormalization()(conv4)
    return conv4
```

In [30]:
```
# adding the FCNN as an head to encoder and decoder network
# using functional API for layer sequencing
def fc(enco):
    flat = Flatten()(enco)
    den = Dense(128, activation='relu')(flat)
    out = Dense(num_classes, activation='softmax')(den)
    return out
```

In [31]:
```
encode = encoder(input_img)
full_model = Model(input_img,fc(encode))
```

In [32]:
```
for l1,l2 in zip(full_model.layers[:19],autoencoder.layers[0:19]):
    l1.set_weights(l2.get_weights())
```

In [33]:
```
autoencoder.get_weights()[0][1]
```

Out[33]:
```
array([[[-1.54724228e-04,  8.53857622e-02,  1.40017807e-01,
          3.70581564e-03, -1.22385100e-01, -9.03333724e-02,
         -1.42092910e-02,  9.53786969e-02,  5.75400293e-02,
          9.30630192e-02,  8.25757384e-02, -8.72964738e-05,
         -3.02403383e-02, -6.17197976e-02,  1.08013727e-01,
         -6.76150322e-02,  3.28186192e-02,  4.71103564e-02,
         -9.78118256e-02,  1.47139803e-01,  1.22941025e-01,
          1.51436487e-02, -9.05194283e-02,  3.97238359e-02,
         -1.15449227e-01,  6.40309453e-02,  8.10088292e-02,
         -3.40074711e-02, -1.34132281e-01, -1.19687431e-01,
         -8.79123956e-02,  1.08853234e-02]],

       [[ 1.91540793e-01,  1.69836655e-01,  5.17360261e-03,
          9.11975279e-02, -7.66993240e-02, -9.51103047e-02,
         -4.71934676e-02,  2.36582085e-01, -3.66107076e-02,
          1.46808177e-01,  1.19001545e-01,  1.35637134e-01,
          1.28245085e-01, -3.67860310e-02, -3.72047164e-02,
          6.57114238e-02, -2.11761147e-02, -1.54801562e-01,
         -8.48870128e-02, -5.92605136e-02,  3.50146443e-02,
          4.84344959e-02, -2.60472689e-02,  1.74833670e-01,
         -4.50035511e-03,  1.51497841e-01,  5.65965995e-02,
         -1.27874196e-01,  1.68987781e-01,  9.66395438e-02,
          1.07529864e-01,  2.21004054e-01]],

       [[ 3.11180577e-02,  1.43947929e-01,  1.48395285e-01,
         -5.66556975e-02,  1.27784222e-01, -1.94676250e-01,
         -1.12775555e-02,  1.49947613e-01,  7.85046890e-02,
          4.91623543e-02, -1.16249183e-02,  1.75571308e-01,
          1.36704937e-01,  1.91662177e-01, -6.67674765e-02,
          1.46571219e-01, -1.78442031e-01, -5.34193926e-02,
          5.86946197e-02, -1.33571669e-01,  3.41310240e-02,
         -1.57855704e-01,  3.69145907e-02,  7.27996230e-02,
         -5.03489859e-02,  2.08528087e-01, -2.12112620e-01,
          1.44237895e-02, -1.03738010e-01,  4.29570861e-02,
         -9.62668881e-02, -6.54554814e-02]]], dtype=float32)
```

In [34]:
```
full_model.get_weights()[0][1]
```

Out[34]:
```
array([[[-1.54724228e-04,  8.53857622e-02,  1.40017807e-01,
          3.70581564e-03, -1.22385100e-01, -9.03333724e-02,
         -1.42092910e-02,  9.53786969e-02,  5.75400293e-02,
          9.30630192e-02,  8.25757384e-02, -8.72964738e-05,
         -3.02403383e-02, -6.17197976e-02,  1.08013727e-01,
         -6.76150322e-02,  3.28186192e-02,  4.71103564e-02,
         -9.78118256e-02,  1.47139803e-01,  1.22941025e-01,
          1.51436487e-02, -9.05194283e-02,  3.97238359e-02,
         -1.15449227e-01,  6.40309453e-02,  8.10088292e-02,
         -3.40074711e-02, -1.34132281e-01, -1.19687431e-01,
         -8.79123956e-02,  1.08853234e-02]],

       [[ 1.91540793e-01,  1.69836655e-01,  5.17360261e-03,
          9.11975279e-02, -7.66993240e-02, -9.51103047e-02,
         -4.71934676e-02,  2.36582085e-01, -3.66107076e-02,
          1.46808177e-01,  1.19001545e-01,  1.35637134e-01,
          1.28245085e-01, -3.67860310e-02, -3.72047164e-02,
          6.57114238e-02, -2.11761147e-02, -1.54801562e-01,
         -8.48870128e-02, -5.92605136e-02,  3.50146443e-02,
          4.84344959e-02, -2.60472689e-02,  1.74833670e-01,
         -4.50035511e-03,  1.51497841e-01,  5.65965995e-02,
         -1.27874196e-01,  1.68987781e-01,  9.66395438e-02,
          1.07529864e-01,  2.21004054e-01]],

       [[ 3.11180577e-02,  1.43947929e-01,  1.48395285e-01,
         -5.66556975e-02,  1.27784222e-01, -1.94676250e-01,
         -1.12775555e-02,  1.49947613e-01,  7.85046890e-02,
          4.91623543e-02, -1.16249183e-02,  1.75571308e-01,
          1.36704937e-01,  1.91662177e-01, -6.67674765e-02,
          1.46571219e-01, -1.78442031e-01, -5.34193926e-02,
          5.86946197e-02, -1.33571669e-01,  3.41310240e-02,
         -1.57855704e-01,  3.69145907e-02,  7.27996230e-02,
         -5.03489859e-02,  2.08528087e-01, -2.12112620e-01,
          1.44237895e-02, -1.03738010e-01,  4.29570861e-02,
         -9.62668881e-02, -6.54554814e-02]]], dtype=float32)
```

In [35]:
```
for layer in full_model.layers[0:19]:
    layer.trainable = False
```

In [36]:
```
full_model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimiz
```

In [37]:
```
full_model.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d_15 (Conv2D)          (None, 28, 28, 32)        320

 batch_normalization_14 (Bat  (None, 28, 28, 32)       128
 chNormalization)

 conv2d_16 (Conv2D)          (None, 28, 28, 32)        9248

 batch_normalization_15 (Bat  (None, 28, 28, 32)       128
 chNormalization)

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 conv2d_17 (Conv2D)          (None, 14, 14, 64)        18496

 batch_normalization_16 (Bat  (None, 14, 14, 64)       256
 chNormalization)

 conv2d_18 (Conv2D)          (None, 14, 14, 64)        36928

 batch_normalization_17 (Bat  (None, 14, 14, 64)       256
 chNormalization)

 max_pooling2d_3 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 conv2d_19 (Conv2D)          (None, 7, 7, 128)         73856

 batch_normalization_18 (Bat  (None, 7, 7, 128)        512
 chNormalization)

 conv2d_20 (Conv2D)          (None, 7, 7, 128)         147584

 batch_normalization_19 (Bat  (None, 7, 7, 128)        512
 chNormalization)

 conv2d_21 (Conv2D)          (None, 7, 7, 256)         295168

 batch_normalization_20 (Bat  (None, 7, 7, 256)        1024
 chNormalization)

 conv2d_22 (Conv2D)          (None, 7, 7, 256)         590080

 batch_normalization_21 (Bat  (None, 7, 7, 256)        1024
 chNormalization)

 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 128)               1605760

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 2,782,570
```

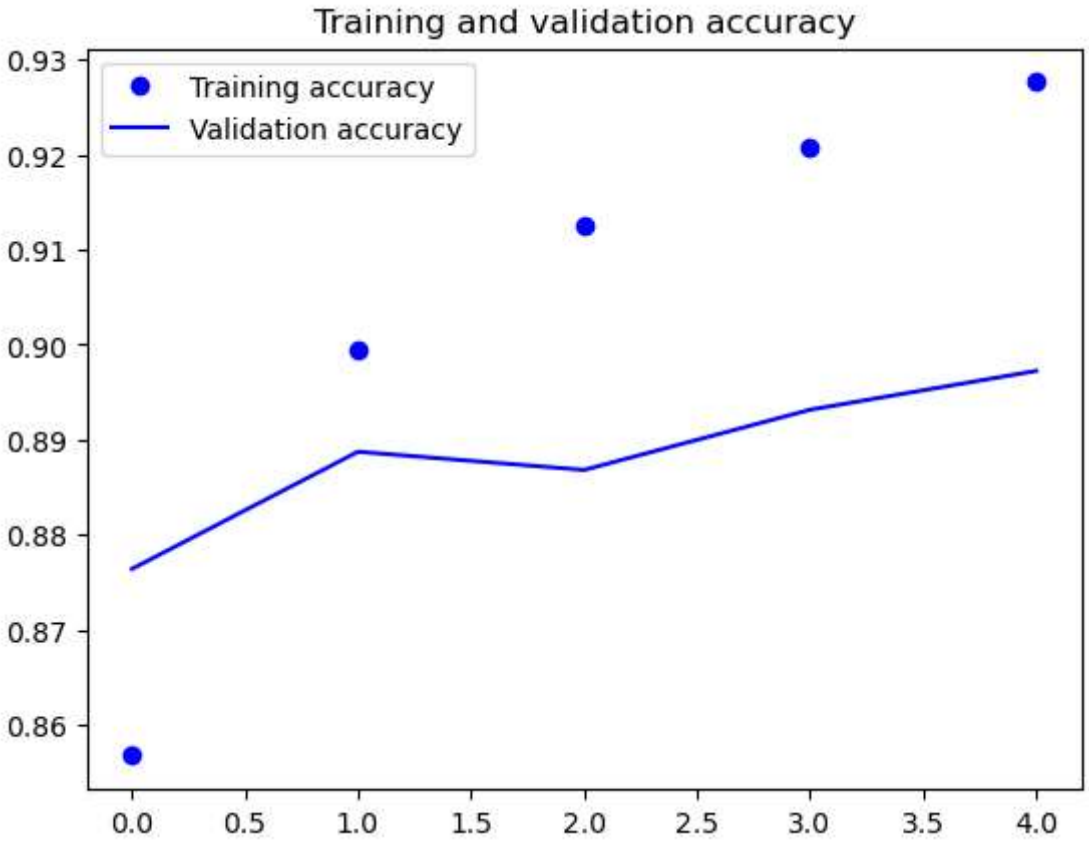
```
Trainable params: 1,607,050
Non-trainable params: 1,175,520
```

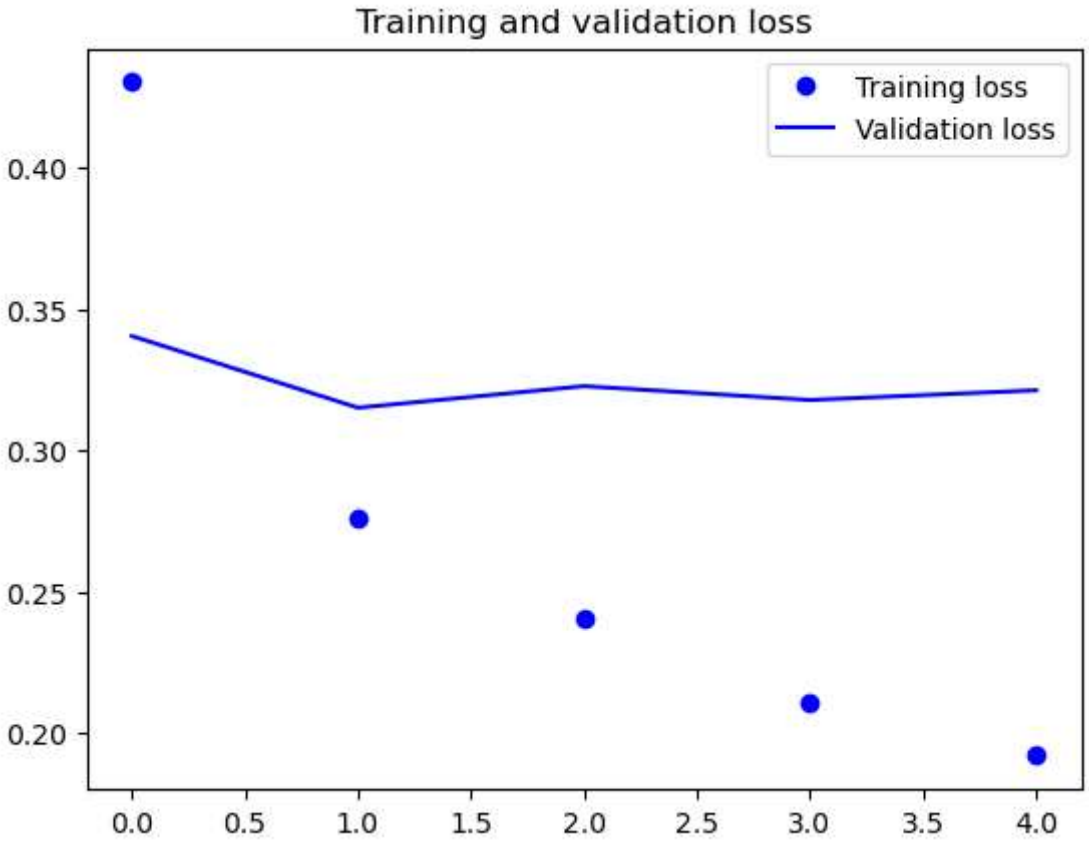
In [38]:
```
classify_train = full_model.fit(train_X, train_label, batch_size=64,epochs=5,verbose=1
```

```
Epoch 1/5
750/750 [==============================] - 68s 90ms/step - loss: 0.4303 - accuracy:
0.8568 - val_loss: 0.3404 - val_accuracy: 0.8764
Epoch 2/5
750/750 [==============================] - 65s 87ms/step - loss: 0.2759 - accuracy:
0.8995 - val_loss: 0.3151 - val_accuracy: 0.8888
Epoch 3/5
750/750 [==============================] - 66s 88ms/step - loss: 0.2401 - accuracy:
0.9126 - val_loss: 0.3227 - val_accuracy: 0.8868
Epoch 4/5
750/750 [==============================] - 66s 88ms/step - loss: 0.2105 - accuracy:
0.9207 - val_loss: 0.3178 - val_accuracy: 0.8932
Epoch 5/5
750/750 [==============================] - 69s 91ms/step - loss: 0.1919 - accuracy:
0.9276 - val_loss: 0.3212 - val_accuracy: 0.8972
```

In [39]:
```
full_model.save_weights('autoencoder_classification.h5')
```

In [42]:
```
accuracy = classify_train.history['accuracy']
val_accuracy = classify_train.history['val_accuracy']
loss = classify_train.history['loss']
val_loss = classify_train.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





In [43]:
```
test_eval = full_model.evaluate(test_data, test_Y_one_hot, verbose=0)
```

In [44]:
```
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

```
Test loss: 0.34199246764183044
Test accuracy: 0.8942999839782715
```

In [45]:
```
predicted_classes = full_model.predict(test_data)
```

```
313/313 [==============================] - 15s 48ms/step
```

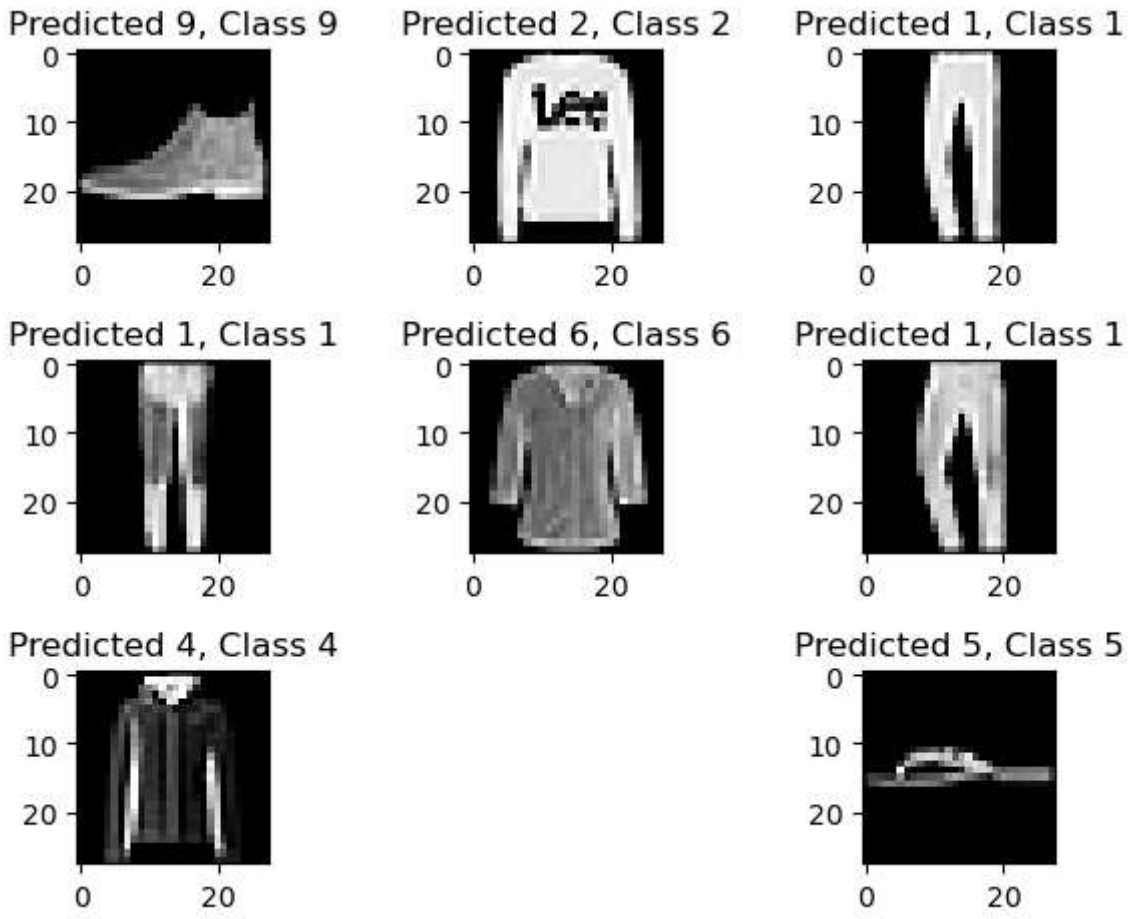
In [53]:
```
predicted_classes = np.argmax(np.round(predicted_classes),axis=1)
```

In [54]:
```
predicted_classes.shape, test_labels.shape
```

Out[54]:
```
((10000,), (10000,))
```

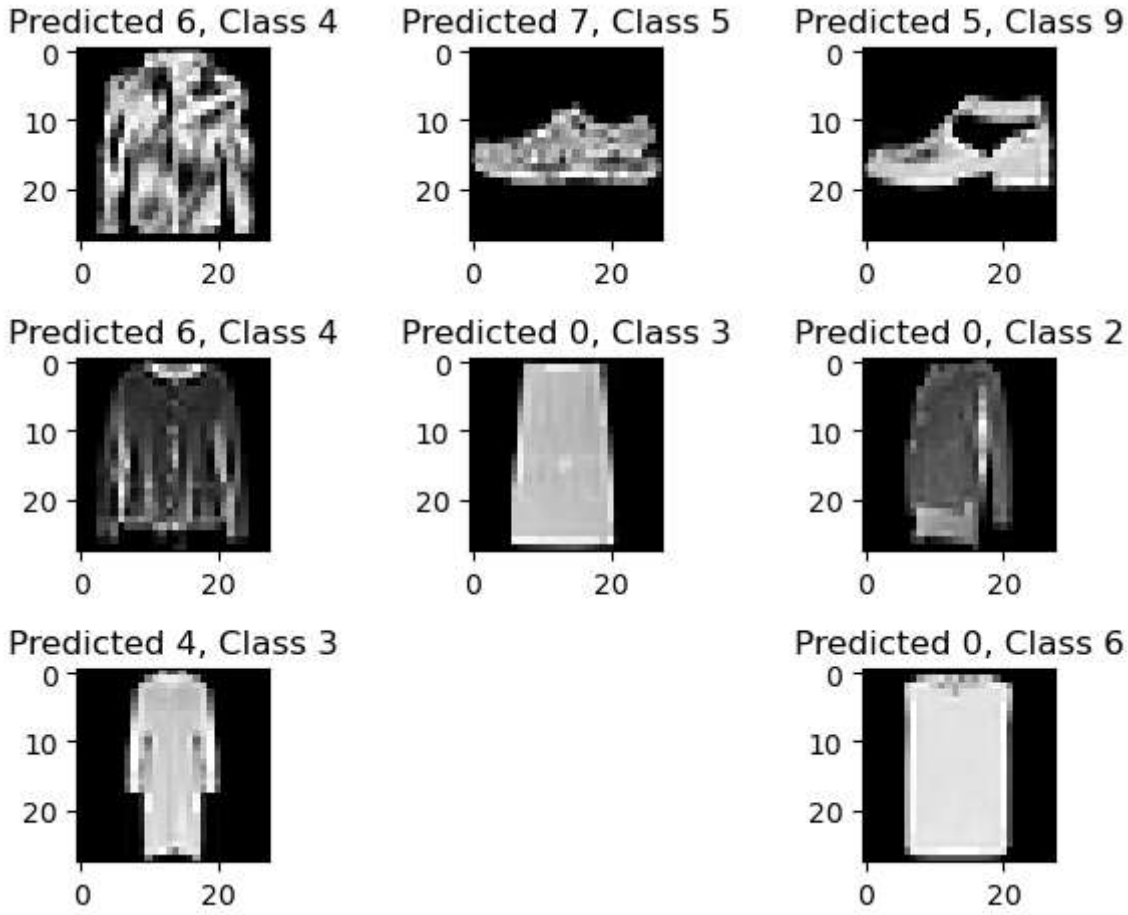
In [55]:
```
correct = np.where(predicted_classes==test_labels)[0]
print("Found %d correct labels" % len(correct))
for i, correct in enumerate(correct[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(test_data[correct].reshape(28,28), cmap='gray', interpolation='none')
    plt.title("Predicted {}, Class {}".format(predicted_classes[correct], test_labels[
    plt.tight_layout()
```

```
Found 8921 correct labels
```



In [56]:
```
incorrect = np.where(predicted_classes!=test_labels)[0]
print("Found %d incorrect labels" % len(incorrect))
for i, incorrect in enumerate(incorrect[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(test_data[incorrect].reshape(28,28), cmap='gray', interpolation='none')
    plt.title("Predicted {}, Class {}".format(predicted_classes[incorrect], test_label
    plt.tight_layout()
```

```
Found 1079 incorrect labels
```



In [58]:
```
from sklearn.metrics import classification_report
target_names = ["Class {}".format(i) for i in range(num_classes)]
print(classification_report(test_labels, predicted_classes, target_names=target_names))
```

```
              precision    recall  f1-score   support

     Class 0       0.74      0.89      0.81      1000
     Class 1       0.99      0.97      0.98      1000
     Class 2       0.81      0.85      0.83      1000
     Class 3       0.90      0.90      0.90      1000
     Class 4       0.83      0.82      0.82      1000
     Class 5       0.99      0.96      0.97      1000
     Class 6       0.82      0.61      0.70      1000
     Class 7       0.94      0.97      0.95      1000
     Class 8       0.97      0.98      0.97      1000
     Class 9       0.97      0.96      0.97      1000

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

In [ ]: