In [1]:

```
from scipy import io as sio
import tensorflow as tf
import numpy as np
import keras
from keras.models import Sequential,Input,Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU

import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split


#step 1 loading and preprocessing EMNIST dataset

mat = sio.loadmat('emnist-letters.mat')
data = mat['dataset']



# this type of sample did not work as the other way of splitting so I p
ass for now
x_train, x_test, y_train, y_test = train_test_split(data['train'][0,0][
'images'][0,0],
                                                    data['train'][0,0][
'images'][0,0],
                                                    test_size=0.2)


#                                                               random_s
tate=42

#define what the training and testing data are

x_train = data['train'][0,0]['images'][0,0]
```

```
y_train = data['train'][0,0]['labels'][0,0]
x_test = data['test'][0,0]['images'][0,0]
y_test = data['test'][0,0]['labels'][0,0]

#I did not use cross-validation by now since it was not that inportant
 in this case
#_train = data['test'][0,0]['labels'][0,0]
#using cros validation
#al_start = x_train.shape[0] - x_test.shape[0]
#_val =x_train[val_start:x_train.shape[0]]
#_val = y_train[val_start:y_train.shape[0]]
#_train = x_train[0:val_start]
#_train = y_train[0:val_start]


#reshape the arrays into image

x_train = x_train.reshape( (x_train.shape[0], 28, 28), order='F')









#_train.shape, y_train.shape

#_test.shape, y_test.shape


#_train = x_train[..., tf.newaxis]
#_test = x_test[..., tf.newaxis]

#_train.shape

#_test.shape

#p.min(x_train), np.max(x_train)
```

```
#x_train = x_train / 255.
#x_test=x_test/255

#p.min(x_train), np.max(x_train)
#x_train = x_train.reshape( (x_train.shape[0], 28, 28), order='F')

#y_train = y_train.reshape( (y_train.shape[0], 28, 28), order='F')'''
x_test = x_test.reshape( (x_test.shape[0], 28, 28), order='F')


#number of unique classes
classes = np.unique(y_train)
nClasses = len(classes)
print('Total number of outputs : ', nClasses)
print('Output classes : ', classes)




print('train shape', x_train.shape)
print('test shape', x_test.shape)
```

```
Using TensorFlow backend.
```

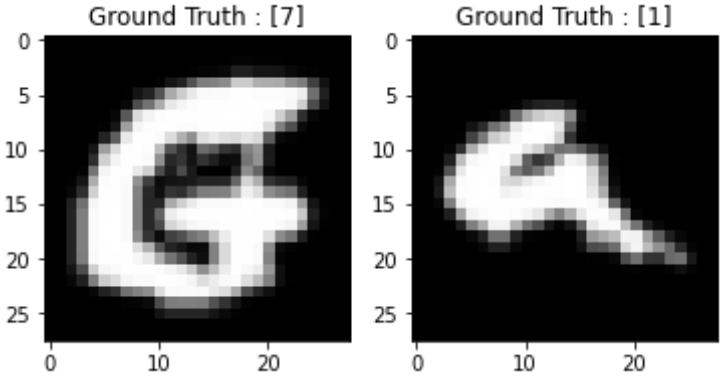
```
Total number of outputs :  26
Output classes :  [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 1
8 19 20 21 22 23 24
 25 26]
train shape (124800, 28, 28)
test shape (20800, 28, 28)
```

In [2]:

```
# Display the image in training data
plt.subplot(121)
plt.imshow(x_train[1,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(y_train[1]))

# Display the first image in testing data
plt.subplot(122)
```

```
plt.imshow(x_test[0,:], cmap='gray')
plt.title("Ground Truth : {}".format(y_test[0]))
```

Out[2]: Text(0.5, 1.0, 'Ground Truth : [1]')



In [3]:
```
#reshaping the training and testing data
x_train = x_train.reshape(-1, 28,28, 1)
x_test = x_test.reshape(-1, 28,28, 1)
x_train.shape, x_test.shape
```

Out[3]: ((124800, 28, 28, 1), (20800, 28, 28, 1))

In [4]:
```
#converting the data from int8 to float32

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train = x_train / 255.
x_test = x_test / 255.
```

In [5]:
```
#need to convert the class label data in one encoding vector.
#So let's convert the training and testing labels into one-hot encoding
 vectors

from keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)
```

```
# Display the change for category label using one-hot encoding
print('Original label:', y_train[3])
print('After conversion to one-hot:', y_train_one_hot[0])

#from sklearn.model_selection import train_test_split
#x_train,valid_X,train_label,valid_label=train_test_split(x_train, y_tr
ain_one_hot, test_size=0.2, random_state=13)

#x_train.shape,valid_X.shape,train_label.shape,valid_label.shape
```

```
Original label: [15]
After conversion to one-hot: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
 0. 0. 0.]
```

In [6]:
```
batch_size = 64
epochs = 13
num_classes = 26
```

In [7]:
```
#built the model

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',input_shape
=(28,28,1),padding='same'))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D((2, 2),padding='same'))
model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
model.add(Flatten())
model.add(Dense(128, activation='linear'))
model.add(LeakyReLU(alpha=0.1))
model.add(Dense(num_classes+1, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy, optimizer=ker
as.optimizers.Adam(),metrics=['accuracy'])
```

```
model.summary()

'''Let's visualize the layers that you created in the above step by usi
ng the summary function.
This will show some parameters (weights and biases) in each layer and a
lso the total parameters in your model'''

#train the model/fiting

model_train = model.fit(x_train, y_train_one_hot, batch_size=batch_size
,epochs=epochs, verbose =1)
#no validation set needed for now validation_data=(valid_X, valid_labe
l))
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 28, 28, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 14, 14, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 128)         73856
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 7, 7, 128)         0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
```

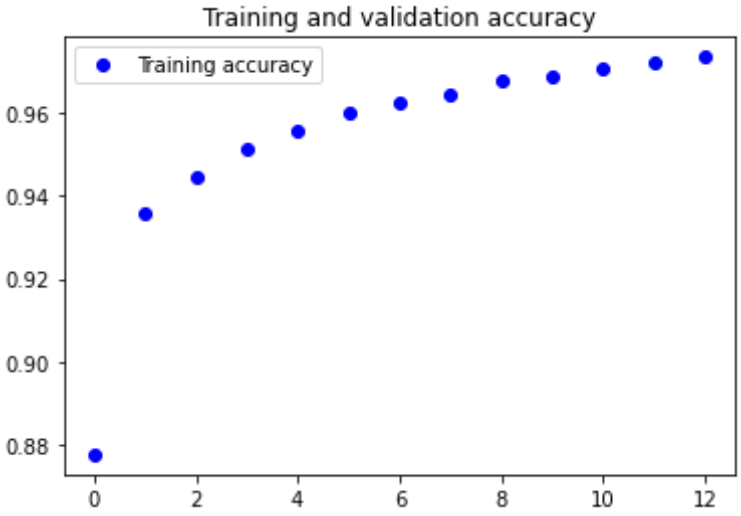
```
dense_1 (Dense)              (None, 128)               262272
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 27)                3483
=================================================================
Total params: 358,427
Trainable params: 358,427
Non-trainable params: 0
_________________________________________________________________
Epoch 1/13
124800/124800 [==============================] - 208s 2ms/step - loss:
0.3779 - accuracy: 0.8778
Epoch 2/13
124800/124800 [==============================] - 208s 2ms/step - loss:
0.1880 - accuracy: 0.9355
Epoch 3/13
124800/124800 [==============================] - 195s 2ms/step - loss:
0.1565 - accuracy: 0.9446
Epoch 4/13
124800/124800 [==============================] - 182s 1ms/step - loss:
0.1350 - accuracy: 0.95101s -
Epoch 5/13
124800/124800 [==============================] - 178s 1ms/step - loss:
0.1191 - accuracy: 0.9555
Epoch 6/13
124800/124800 [==============================] - 168s 1ms/step - loss:
0.1062 - accuracy: 0.9597
Epoch 7/13
124800/124800 [==============================] - 166s 1ms/step - loss:
0.0959 - accuracy: 0.9625
Epoch 8/13
124800/124800 [==============================] - 165s 1ms/step - loss:
0.0876 - accuracy: 0.9644
Epoch 9/13
124800/124800 [==============================] - 185s 1ms/step - loss:
0.0789 - accuracy: 0.9675
Epoch 10/13
124800/124800 [==============================] - 183s 1ms/step - loss:
0.0752 - accuracy: 0.9688
```

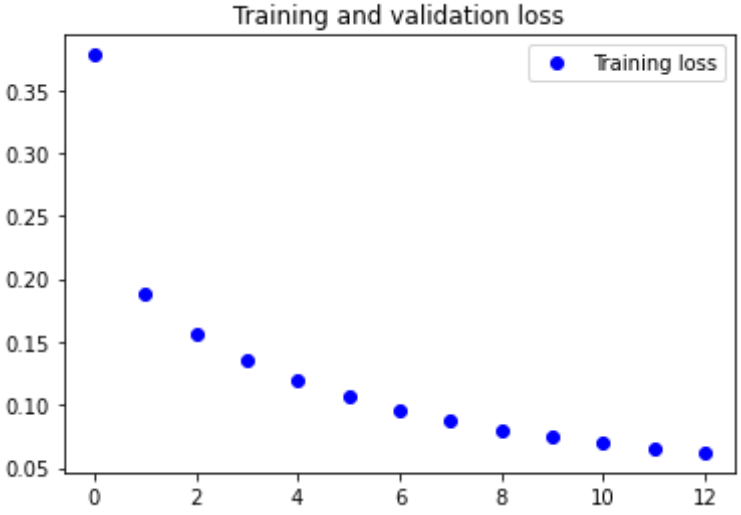
```
Epoch 11/13
124800/124800 [==============================] - 182s 1ms/step - loss:
0.0693 - accuracy: 0.9706
Epoch 12/13
124800/124800 [==============================] - 185s 1ms/step - loss:
0.0657 - accuracy: 0.9718
Epoch 13/13
124800/124800 [==============================] - 189s 2ms/step - loss:
0.0622 - accuracy: 0.9733
```

In [8]:
```
#test evaluation
test_eval = model.evaluate(x_test, y_test_one_hot, verbose=0)
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

```
Test loss: 0.24865888032202707
Test accuracy: 0.938605785369873
```

In [9]:
```
#now we analyse our model and see what can be done to make it better

accuracy = model_train.history['accuracy']
#val_acc = model_train.history['val_accuracy']
loss = model_train.history['loss']
#val_loss = model_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 [16]:
```
#dropout
batch_size = 64
epochs = 2
num_classes = 26
```

In [17]:
```
#CNN with the drop out layer
```

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',padding='sa
me',input_shape=(28,28,1)))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D((2, 2),padding='same'))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(128, activation='linear'))
model.add(LeakyReLU(alpha=0.1))
model.add(Dropout(0.3))
model.add(Dense(num_classes+1, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy, optimizer=ker
as.optimizers.Adam(),metrics=['accuracy'])
model_train_dropout = model.fit(x_train, y_train_one_hot, batch_size=ba
tch_size,epochs=epochs,verbose=1)


'''We add a dropout layer to overcome the problem of overfitting to som
e extent. Dropout randomly turns off a fraction of neurons during the t
raining process, reducing the dependency on the training set by some am
ount.
How many fractions of neurons you want to turn off is decided by a hype
rparameter, which can be tuned accordingly.
This way, turning off some neurons will not allow the network to memori
ze the training data since not all the neurons will be active at the sa
me time and the inactive neurons will not be able to learn anything.'''
```

```
Epoch 1/2
124800/124800 [==============================] - 208s 2ms/step - loss:
0.5960 - accuracy: 0.8099
```

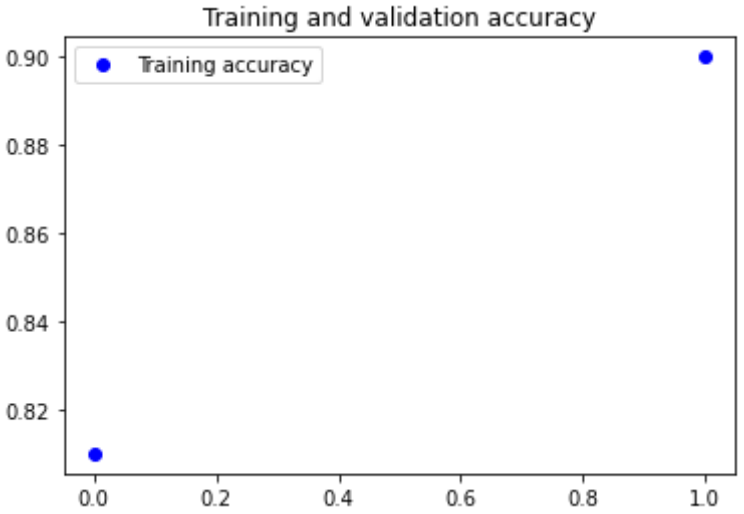
```
Epoch 2/2
124800/124800 [==============================] - 225s 2ms/step - loss:
0.3037 - accuracy: 0.9000
```

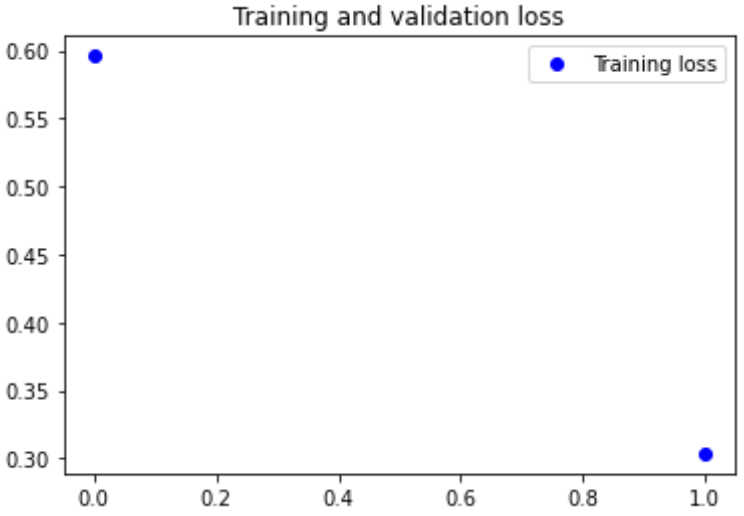
Out[17]:
```
'We add a dropout layer to overcome the problem of overfitting to some
extent. Dropout randomly turns off a fraction of neurons during the tra
ining process, reducing the dependency on the training set by some amou
nt.\nHow many fractions of neurons you want to turn off is decided by a
hyperparameter, which can be tuned accordingly.\nThis way, turning off
some neurons will not allow the network to memorize the training data s
ince not all the neurons will be active at the same time and the inacti
ve neurons will not be able to learn anything.'
```

In [29]:
```
#model summary
model.summary()


#now we analyse our model and see what can be done to make it better

accuracy = model_train_dropout.history['accuracy']
#val_acc = model_train.history['val_accuracy']
loss = model_train_dropout.history['loss']
#val_loss = model_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()



#save the model to use for testing
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
leaky_re_lu_17 (LeakyReLU)   (None, 28, 28, 32)        0
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
dropout_13 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 14, 14, 64)        18496
_________________________________________________________________
leaky_re_lu_18 (LeakyReLU)   (None, 14, 14, 64)        0
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_14 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 7, 7, 128)         73856
_________________________________________________________________
leaky_re_lu_19 (LeakyReLU)   (None, 7, 7, 128)         0
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
dropout_15 (Dropout)         (None, 4, 4, 128)         0
_________________________________________________________________
flatten_5 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_9 (Dense)              (None, 128)               262272
_________________________________________________________________
leaky_re_lu_20 (LeakyReLU)   (None, 128)               0
_________________________________________________________________
dropout_16 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_10 (Dense)             (None, 27)                3483
=================================================================
Total params: 358,427
```

```
Trainable params: 358,427
Non-trainable params: 0
_________________________________________________________________
```





In [30]:
```
model.save("epic_let_read.model")
```

In [31]:
```
new_model = tf.keras.models.load_model("epic_let_read.model")
```

In [32]:
```
prediktions = new_model.predict(x_test)
print(prediktions)
```

```
[[1.2393689e-06 9.2087340e-01 8.9413326e-05 ... 3.9757852e-04
  3.4936613e-05 6.1814341e-04]
 [1.2356492e-09 9.9693751e-01 3.0252834e-06 ... 8.4265207e-07
  2.5167697e-06 3.3734898e-06]
 [6.6081758e-11 9.9269682e-01 3.5546526e-07 ... 1.1386664e-08
  2.4139798e-08 3.6830606e-06]
 ...
 [7.4849111e-12 1.1405632e-05 3.9312499e-07 ... 7.0474493e-06
  1.6392974e-08 9.9944478e-01]
 [3.8649004e-09 1.2394096e-04 7.3254792e-05 ... 7.2232610e-06
  3.1839065e-06 9.9828905e-01]
 [5.6445500e-14 2.4358385e-06 2.4650814e-07 ... 5.2607788e-06
  9.4131170e-10 9.9993896e-01]]
```

In [33]:
```
print(np.argmax(prediktions[6452]))
```

```
12
```

In [ ]:
```

```

In [34]:
```
test_eval = model.evaluate(x_test, y_test_one_hot, verbose=1)
```

```
20800/20800 [==============================] - 8s 398us/step
```

In [35]:
```
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

```
Test loss: 0.21073072996581546
Test accuracy: 0.927307665348053
```

In [36]:
```
predicted_classes = model.predict(x_test)
```

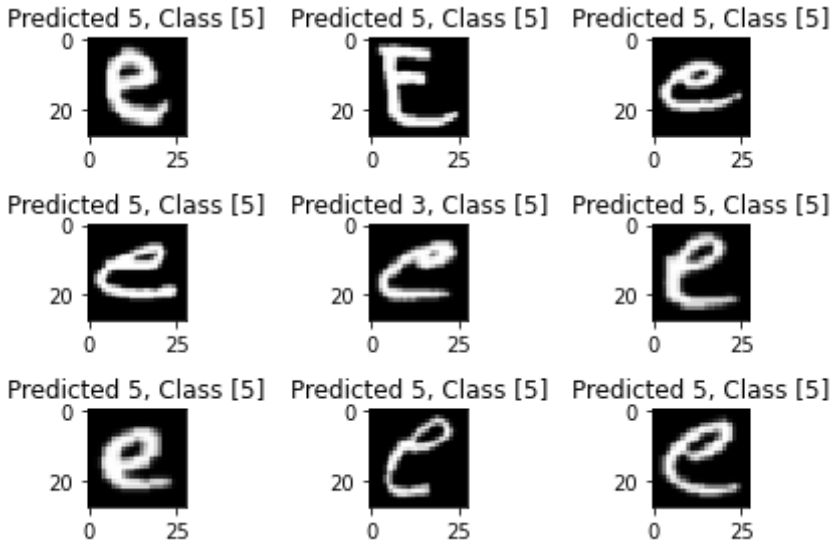
In [37]:
```
#need to use argmax function to find the one prediction with the highes
t probability
predicted_classes = np.argmax(np.round(predicted_classes),axis=1)
```

In [38]:
```
predicted_classes.shape, y_test.shape
```

Out[38]:
```
((20800,), (20800, 1))
```

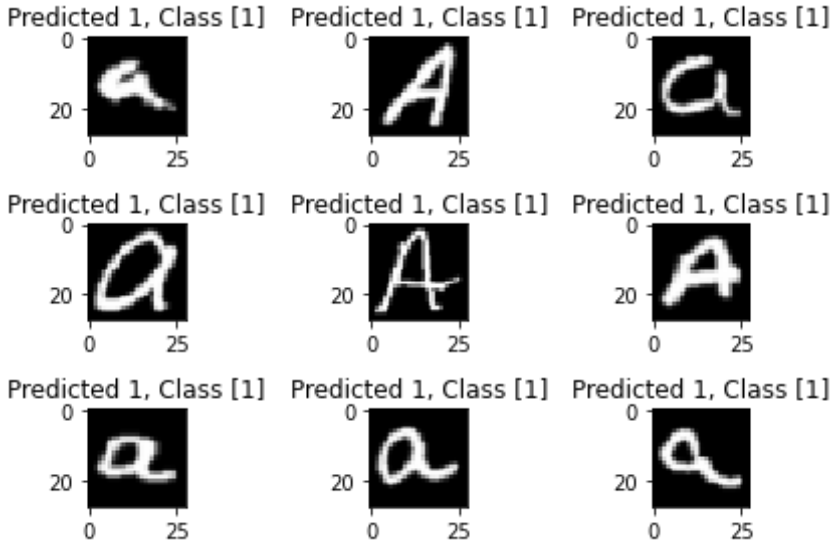
In [39]:
```
correct = np.where(predicted_classes[3422]==y_test)[0]
print ("Found %d correct labels" % len(correct))
for i, correct in enumerate(correct[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(x_test[correct].reshape(28,28), cmap='gray', interpolati
on='none')
    plt.title("Predicted {}, Class {}".format(predicted_classes[correct
], y_test[correct]))
    plt.tight_layout()
```

```
Found 800 correct labels
```



In [40]:
```
#now for the incorrect ones
incorrect = np.where(predicted_classes[3422]!=y_test)[0]
print ("Found %d incorrect labels" % len(incorrect))
for i, incorrect in enumerate(incorrect[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(x_test[incorrect].reshape(28,28), cmap='gray', interpola
tion='none')
    plt.title("Predicted {}, Class {}".format(predicted_classes[incorre
ct], y_test[incorrect]))
    plt.tight_layout()
```

```
Found 20000 incorrect labels
```



In [41]:
```
'''Classification report will help us in identifying the misclassified
 classes in more detail.
You will be able to observe for which class the model performed bad out
 of the given 26 classes'''


from sklearn.metrics import classification_report
target_names = ["Class {}".format(i) for i in range(num_classes+1)]
```

```
print(classification_report(y_test, predicted_classes, target_names=tar
get_names))
```

```
              precision    recall  f1-score   support

     Class 0       0.00      0.00      0.00         0
     Class 1       0.93      0.92      0.93       800
     Class 2       0.99      0.95      0.97       800
     Class 3       0.96      0.94      0.95       800
     Class 4       0.97      0.92      0.94       800
     Class 5       0.95      0.95      0.95       800
     Class 6       0.99      0.94      0.96       800
     Class 7       0.90      0.75      0.82       800
     Class 8       0.94      0.93      0.94       800
     Class 9       0.67      0.85      0.75       800
    Class 10       0.97      0.92      0.95       800
    Class 11       0.99      0.92      0.96       800
    Class 12       0.79      0.61      0.69       800
    Class 13       0.98      0.97      0.98       800
    Class 14       0.94      0.96      0.95       800
    Class 15       0.91      0.99      0.95       800
    Class 16       0.98      0.98      0.98       800
    Class 17       0.83      0.84      0.84       800
    Class 18       0.98      0.92      0.95       800
    Class 19       0.97      0.98      0.98       800
    Class 20       0.94      0.98      0.96       800
    Class 21       0.95      0.93      0.94       800
    Class 22       0.93      0.93      0.93       800
    Class 23       0.99      0.97      0.98       800
    Class 24       0.97      0.95      0.96       800
    Class 25       0.95      0.97      0.96       800
    Class 26       0.98      0.98      0.98       800

    accuracy                           0.92     20800
   macro avg       0.90      0.89      0.89     20800
weighted avg       0.94      0.92      0.93     20800
```

```
c:\users\serem1\appdata\local\programs\python\python38\lib\site-packa
ges\sklearn\metrics\_classification.py:1221: UndefinedMetricWarning:
```

```
Recall and F-score are ill-defined and being set to 0.0 in labels wit
h no true samples. Use `zero_division` parameter to control this beha
vior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [ ]:

In [ ]: