a3-dl

April 23, 2024

1 ASSIGNMENT - 3

1.0.1 Problem Statement

Convolutional neural network (CNN) (Any One from the following) - Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

[ ]:
```
from keras.datasets import fashion_mnist
(train_X,train_Y), (test_X,test_Y) = fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 2s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 1s 0us/step
```

[ ]:
```
import numpy as np
from keras.utils import to_categorical
import matplotlib.pyplot as plt
%matplotlib inline

print('Training data shape : ', train_X.shape, train_Y.shape)

print('Testing data shape : ', test_X.shape, test_Y.shape)
```

```
Training data shape :  (60000, 28, 28) (60000,)
Testing data shape :  (10000, 28, 28) (10000,)
```

[ ]:
```
# Find the unique numbers from the train labels
classes = np.unique(train_Y)
nClasses = len(classes)
print('Total number of outputs : ', nClasses)
```

```python
print('Output classes : ', classes)
```

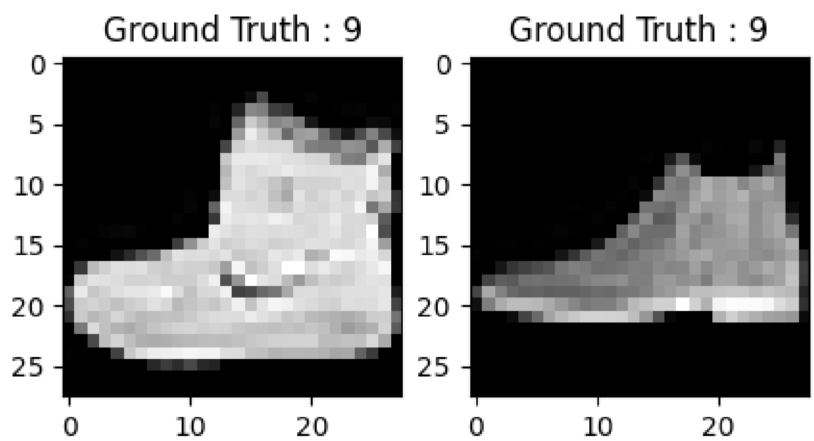
```
Total number of outputs :  10
Output classes :  [0 1 2 3 4 5 6 7 8 9]
```

[ ]:
```python
plt.figure(figsize=[5,5])

# Display the first image in training data
plt.subplot(121)
plt.imshow(train_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(train_Y[0]))

# Display the first image in testing data
plt.subplot(122)
plt.imshow(test_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(test_Y[0]))
```

[ ]:
```
Text(0.5, 1.0, 'Ground Truth : 9')
```



[ ]:
```python
train_X = train_X.reshape(-1, 28,28, 1)
test_X = test_X.reshape(-1, 28,28, 1)
train_X.shape, test_X.shape
```

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

[ ]:
```python
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255.
test_X = test_X / 255.
```

```
[ ]: # Change the labels from categorical to one-hot encoding
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

# Display the change for category label using one-hot encoding
print('Original label:', train_Y[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.]
```

```
[ ]: from sklearn.model_selection import train_test_split
train_X,valid_X,train_label,valid_label = train_test_split(train_X,␣
 ↪train_Y_one_hot, test_size=0.2, random_state=13)
```

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

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

```
[ ]: import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization
#from keras.layers.normalization import BatchNormalization
#from keras.layers.advanced_activations import LeakyReLU
from keras.layers import LeakyReLU
```

```
[ ]: #from keras.models import Input
from keras.models import Model
```

```
[ ]: batch_size = 64
epochs = 20
num_classes = 10
```

```
[ ]: fashion_model = Sequential()
fashion_model.add(Conv2D(32, kernel_size=(3,␣
 ↪3),activation='linear',input_shape=(28,28,1),padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D((2, 2),padding='same'))
fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Flatten())
```

```
fashion_model.add(Dense(128, activation='linear'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(Dense(num_classes, activation='softmax'))
```

[ ]:
```
fashion_model.compile(loss=keras.losses.categorical_crossentropy,␣
 ↪optimizer=keras.optimizers.Adam(),metrics=['accuracy'])
```

[ ]:
```
fashion_model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 leaky_re_lu (LeakyReLU)     (None, 28, 28, 32)        0

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

 conv2d_1 (Conv2D)           (None, 14, 14, 64)        18496

 leaky_re_lu_1 (LeakyReLU)   (None, 14, 14, 64)        0

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

 conv2d_2 (Conv2D)           (None, 7, 7, 128)         73856

 leaky_re_lu_2 (LeakyReLU)   (None, 7, 7, 128)         0

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 128)               262272

 leaky_re_lu_3 (LeakyReLU)   (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 356,234
Trainable params: 356,234
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
fashion_train = fashion_model.fit(train_X, train_label,␣
↪batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(valid_X,␣
↪valid_label))
```

```
Epoch 1/20
750/750 [==============================] - 14s 5ms/step - loss: 0.4698 -
accuracy: 0.8284 - val_loss: 0.3380 - val_accuracy: 0.8756
Epoch 2/20
750/750 [==============================] - 4s 5ms/step - loss: 0.2900 -
accuracy: 0.8951 - val_loss: 0.2792 - val_accuracy: 0.8992
Epoch 3/20
750/750 [==============================] - 4s 6ms/step - loss: 0.2416 -
accuracy: 0.9116 - val_loss: 0.2549 - val_accuracy: 0.9075
Epoch 4/20
750/750 [==============================] - 3s 4ms/step - loss: 0.2133 -
accuracy: 0.9220 - val_loss: 0.2620 - val_accuracy: 0.9057
Epoch 5/20
750/750 [==============================] - 4s 5ms/step - loss: 0.1842 -
accuracy: 0.9310 - val_loss: 0.2373 - val_accuracy: 0.9191
Epoch 6/20
750/750 [==============================] - 3s 4ms/step - loss: 0.1634 -
accuracy: 0.9386 - val_loss: 0.2434 - val_accuracy: 0.9130
Epoch 7/20
750/750 [==============================] - 3s 5ms/step - loss: 0.1433 -
accuracy: 0.9471 - val_loss: 0.2262 - val_accuracy: 0.9189
Epoch 8/20
750/750 [==============================] - 3s 4ms/step - loss: 0.1219 -
accuracy: 0.9542 - val_loss: 0.2442 - val_accuracy: 0.9202
Epoch 9/20
750/750 [==============================] - 4s 6ms/step - loss: 0.1057 -
accuracy: 0.9606 - val_loss: 0.2665 - val_accuracy: 0.9188
Epoch 10/20
750/750 [==============================] - 3s 4ms/step - loss: 0.0903 -
accuracy: 0.9660 - val_loss: 0.3007 - val_accuracy: 0.9141
Epoch 11/20
750/750 [==============================] - 3s 4ms/step - loss: 0.0741 -
accuracy: 0.9721 - val_loss: 0.2660 - val_accuracy: 0.9251
Epoch 12/20
750/750 [==============================] - 4s 5ms/step - loss: 0.0648 -
accuracy: 0.9754 - val_loss: 0.2994 - val_accuracy: 0.9163
Epoch 13/20
750/750 [==============================] - 3s 5ms/step - loss: 0.0567 -
accuracy: 0.9791 - val_loss: 0.3027 - val_accuracy: 0.9193
Epoch 14/20
750/750 [==============================] - 3s 5ms/step - loss: 0.0501 -
accuracy: 0.9814 - val_loss: 0.3468 - val_accuracy: 0.9216
Epoch 15/20
```

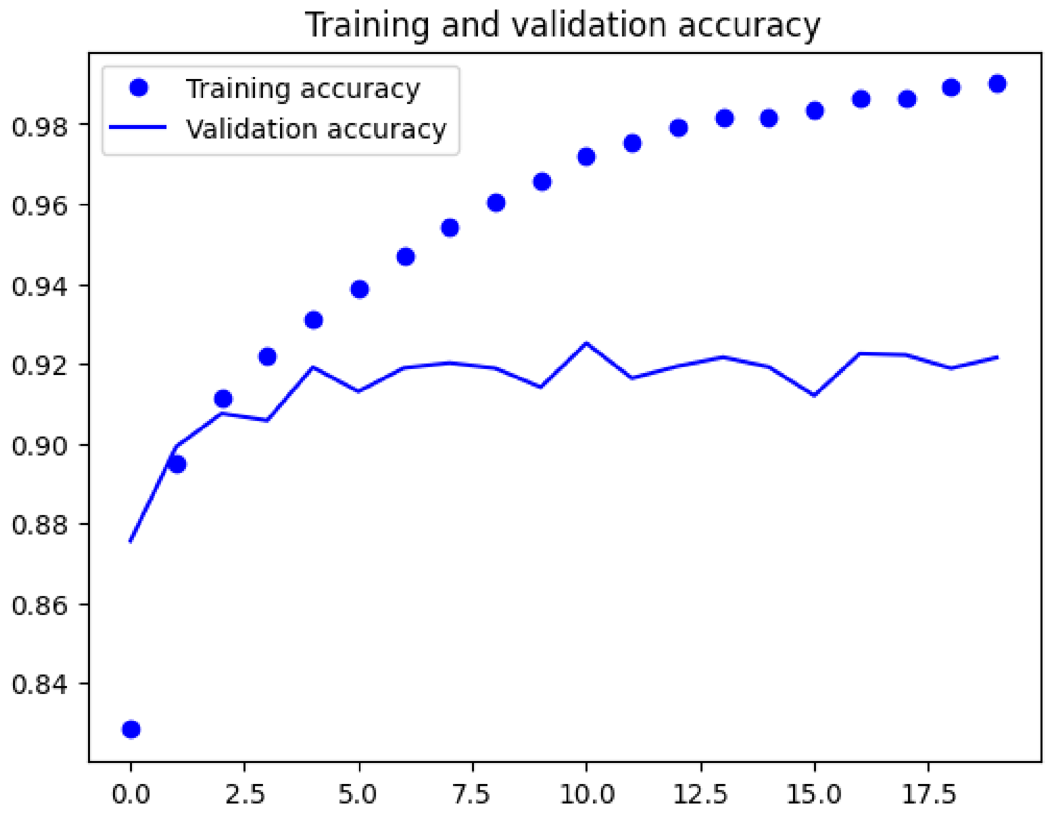
```
750/750 [==============================] - 3s 4ms/step - loss: 0.0480 -
accuracy: 0.9818 - val_loss: 0.3380 - val_accuracy: 0.9192
Epoch 16/20
750/750 [==============================] - 4s 5ms/step - loss: 0.0416 -
accuracy: 0.9838 - val_loss: 0.3953 - val_accuracy: 0.9120
Epoch 17/20
750/750 [==============================] - 3s 4ms/step - loss: 0.0365 -
accuracy: 0.9864 - val_loss: 0.3937 - val_accuracy: 0.9225
Epoch 18/20
750/750 [==============================] - 3s 4ms/step - loss: 0.0354 -
accuracy: 0.9867 - val_loss: 0.4184 - val_accuracy: 0.9222
Epoch 19/20
750/750 [==============================] - 4s 6ms/step - loss: 0.0283 -
accuracy: 0.9895 - val_loss: 0.4743 - val_accuracy: 0.9188
Epoch 20/20
750/750 [==============================] - 3s 5ms/step - loss: 0.0268 -
accuracy: 0.9901 - val_loss: 0.4631 - val_accuracy: 0.9215
```

[ ]:
```
test_eval = fashion_model.evaluate(test_X, test_Y_one_hot, verbose=0)
```

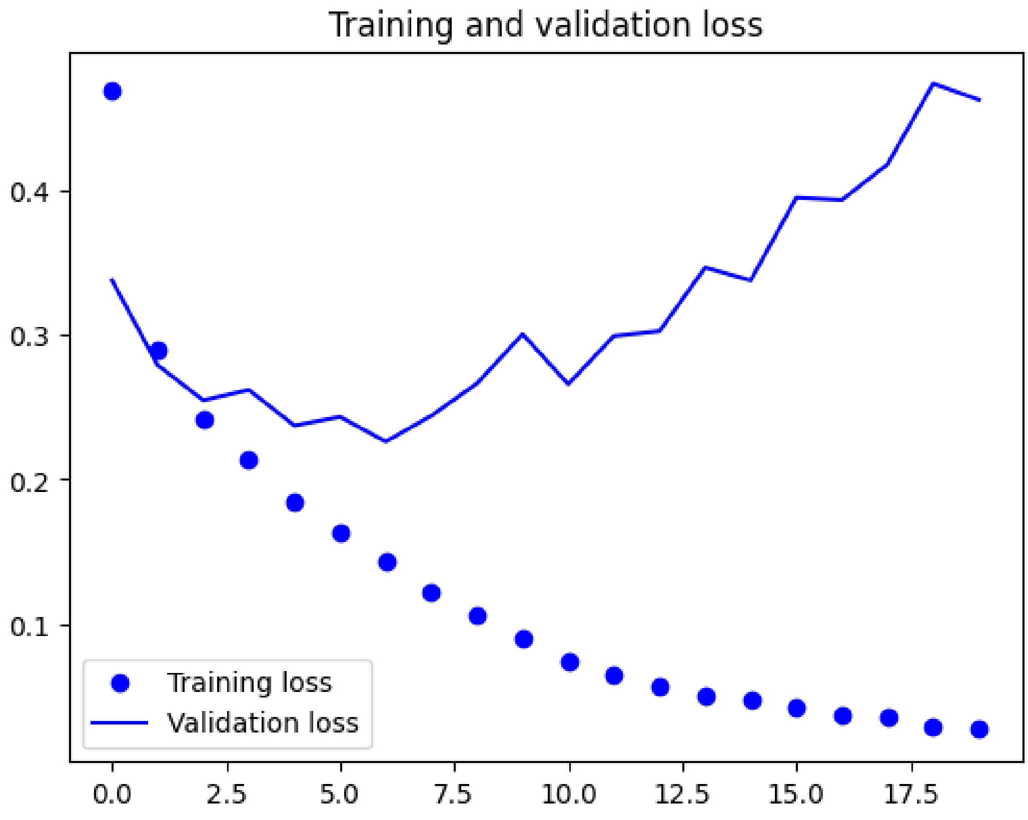
[ ]:
```
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

```
Test loss: 0.4764760136604309
Test accuracy: 0.9204000234603882
```

[ ]:
```
accuracy = fashion_train.history['accuracy']
val_accuracy = fashion_train.history['val_accuracy']
loss = fashion_train.history['loss']
val_loss = fashion_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()
```



Training and validation accuracy
Training accuracy
Validation accuracy
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5



Adding Dropout into the Network

[ ]:
```
batch_size = 64
epochs = 20
num_classes = 10
```

[ ]:
```
fashion_model = Sequential()
fashion_model.add(Conv2D(32, kernel_size=(3,␣
↪3),activation='linear',padding='same',input_shape=(28,28,1)))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D((2, 2),padding='same'))
fashion_model.add(Dropout(0.25))
fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Dropout(0.25))
fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Dropout(0.4))
fashion_model.add(Flatten())
```

```python
fashion_model.add(Dense(128, activation='linear'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(Dropout(0.3))
fashion_model.add(Dense(num_classes, activation='softmax'))
```

[ ]:
```python
fashion_model.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 28, 28, 32)        320

 leaky_re_lu_4 (LeakyReLU)   (None, 28, 28, 32)        0

 max_pooling2d_3 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 dropout (Dropout)           (None, 14, 14, 32)        0

 conv2d_4 (Conv2D)           (None, 14, 14, 64)        18496

 leaky_re_lu_5 (LeakyReLU)   (None, 14, 14, 64)        0

 max_pooling2d_4 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 7, 7, 64)          0

 conv2d_5 (Conv2D)           (None, 7, 7, 128)         73856

 leaky_re_lu_6 (LeakyReLU)   (None, 7, 7, 128)         0

 max_pooling2d_5 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 dropout_2 (Dropout)         (None, 4, 4, 128)         0

 flatten_1 (Flatten)         (None, 2048)              0

 dense_2 (Dense)             (None, 128)               262272

 leaky_re_lu_7 (LeakyReLU)   (None, 128)               0

 dropout_3 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 10)                1290
```

```
=================================================================
Total params: 356,234
Trainable params: 356,234
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
fashion_model.compile(loss=keras.losses.categorical_crossentropy,␣
↪optimizer=keras.optimizers.Adam(),metrics=['accuracy'])
```

[ ]:
```
fashion_train_dropout = fashion_model.fit(train_X, train_label,␣
↪batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(valid_X,␣
↪valid_label))
```

```
Epoch 1/20
750/750 [==============================] - 4s 5ms/step - loss: 0.5853 -
accuracy: 0.7844 - val_loss: 0.3605 - val_accuracy: 0.8705
Epoch 2/20
750/750 [==============================] - 4s 6ms/step - loss: 0.3728 -
accuracy: 0.8625 - val_loss: 0.3110 - val_accuracy: 0.8841
Epoch 3/20
750/750 [==============================] - 4s 5ms/step - loss: 0.3200 -
accuracy: 0.8832 - val_loss: 0.2835 - val_accuracy: 0.8924
Epoch 4/20
750/750 [==============================] - 4s 5ms/step - loss: 0.2945 -
accuracy: 0.8922 - val_loss: 0.2725 - val_accuracy: 0.8990
Epoch 5/20
750/750 [==============================] - 5s 7ms/step - loss: 0.2788 -
accuracy: 0.8954 - val_loss: 0.2391 - val_accuracy: 0.9108
Epoch 6/20
750/750 [==============================] - 4s 5ms/step - loss: 0.2672 -
accuracy: 0.8998 - val_loss: 0.2474 - val_accuracy: 0.9088
Epoch 7/20
750/750 [==============================] - 4s 5ms/step - loss: 0.2517 -
accuracy: 0.9060 - val_loss: 0.2288 - val_accuracy: 0.9168
Epoch 8/20
750/750 [==============================] - 4s 6ms/step - loss: 0.2459 -
accuracy: 0.9082 - val_loss: 0.2270 - val_accuracy: 0.9163
Epoch 9/20
750/750 [==============================] - 4s 5ms/step - loss: 0.2385 -
accuracy: 0.9114 - val_loss: 0.2262 - val_accuracy: 0.9160
Epoch 10/20
750/750 [==============================] - 4s 5ms/step - loss: 0.2298 -
accuracy: 0.9149 - val_loss: 0.2178 - val_accuracy: 0.9199
Epoch 11/20
750/750 [==============================] - 4s 6ms/step - loss: 0.2282 -
accuracy: 0.9148 - val_loss: 0.2398 - val_accuracy: 0.9114
Epoch 12/20
```

```
750/750 [==============================] - 4s 5ms/step - loss: 0.2243 -
accuracy: 0.9166 - val_loss: 0.2133 - val_accuracy: 0.9234
Epoch 13/20
750/750 [==============================] - 4s 5ms/step - loss: 0.2149 -
accuracy: 0.9192 - val_loss: 0.2216 - val_accuracy: 0.9178
Epoch 14/20
750/750 [==============================] - 5s 6ms/step - loss: 0.2166 -
accuracy: 0.9181 - val_loss: 0.2109 - val_accuracy: 0.9247
Epoch 15/20
750/750 [==============================] - 4s 5ms/step - loss: 0.2104 -
accuracy: 0.9207 - val_loss: 0.2123 - val_accuracy: 0.9251
Epoch 16/20
750/750 [==============================] - 4s 5ms/step - loss: 0.2060 -
accuracy: 0.9228 - val_loss: 0.2050 - val_accuracy: 0.9274
Epoch 17/20
750/750 [==============================] - 4s 6ms/step - loss: 0.2066 -
accuracy: 0.9213 - val_loss: 0.2162 - val_accuracy: 0.9237
Epoch 18/20
750/750 [==============================] - 4s 5ms/step - loss: 0.2017 -
accuracy: 0.9248 - val_loss: 0.2121 - val_accuracy: 0.9252
Epoch 19/20
750/750 [==============================] - 4s 5ms/step - loss: 0.2006 -
accuracy: 0.9252 - val_loss: 0.2090 - val_accuracy: 0.9243
Epoch 20/20
750/750 [==============================] - 4s 5ms/step - loss: 0.1968 -
accuracy: 0.9255 - val_loss: 0.2092 - val_accuracy: 0.9255
```

[ ]:
```
fashion_model.save("fashion_model_dropout.h5py")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
3 of 3). These functions will not be directly callable after loading.
```

[ ]:
```
test_eval = fashion_model.evaluate(test_X, test_Y_one_hot, verbose=1)
```

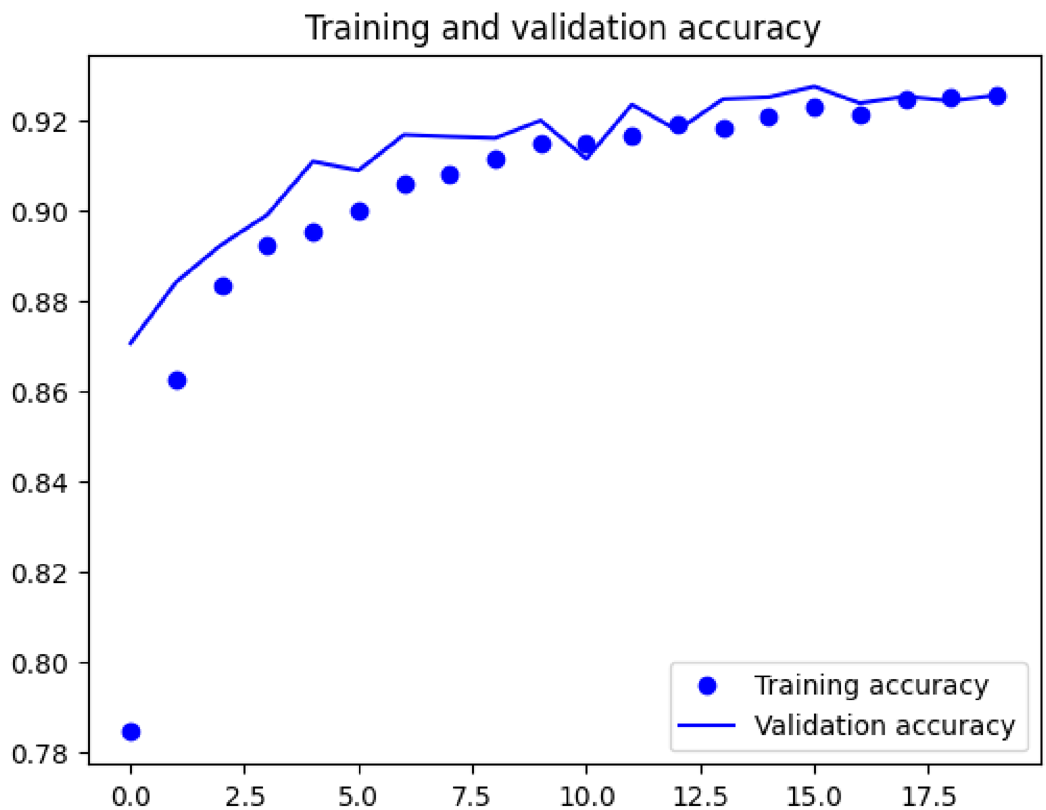
```
313/313 [==============================] - 1s 3ms/step - loss: 0.2202 -
accuracy: 0.9213
```

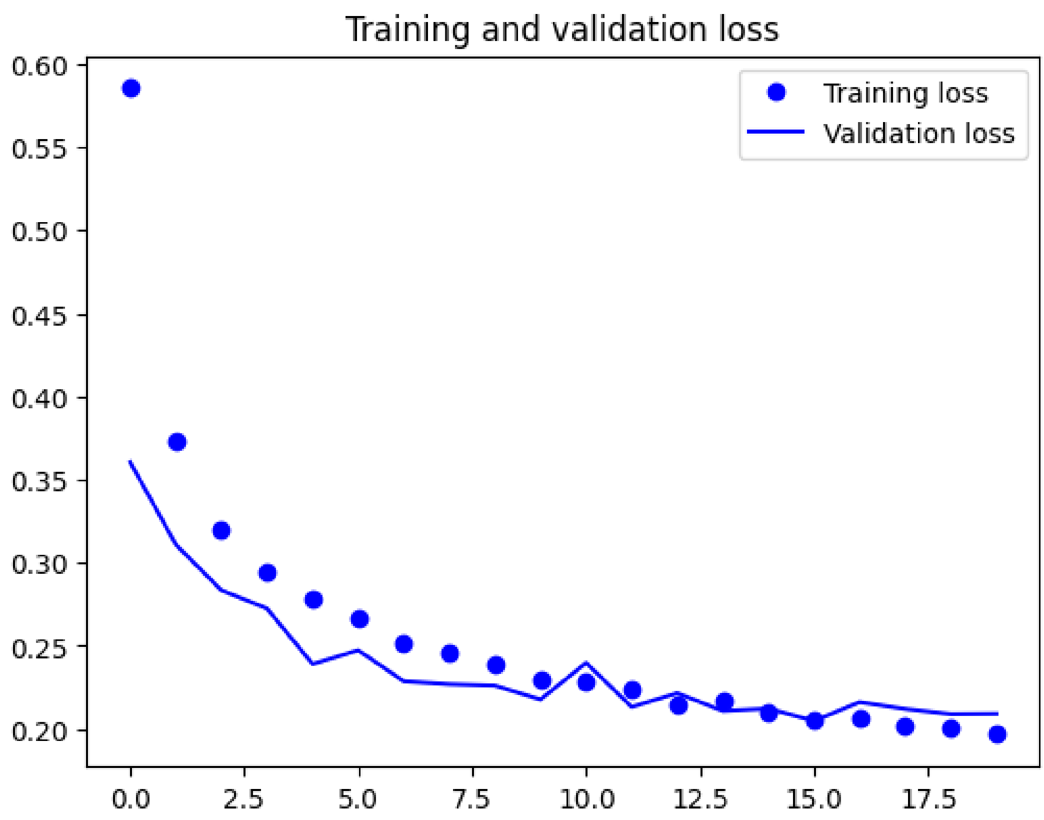
[ ]:
```
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

```
Test loss: 0.22022153437137604
Test accuracy: 0.9212999939918518
```

[ ]:
```
accuracy = fashion_train_dropout.history['accuracy']
val_accuracy = fashion_train_dropout.history['val_accuracy']
loss = fashion_train_dropout.history['loss']
val_loss = fashion_train_dropout.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()
```





[ ]:
```
predicted_classes = fashion_model.predict(test_X)
```

```
313/313 [==============================] - 1s 2ms/step
```

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

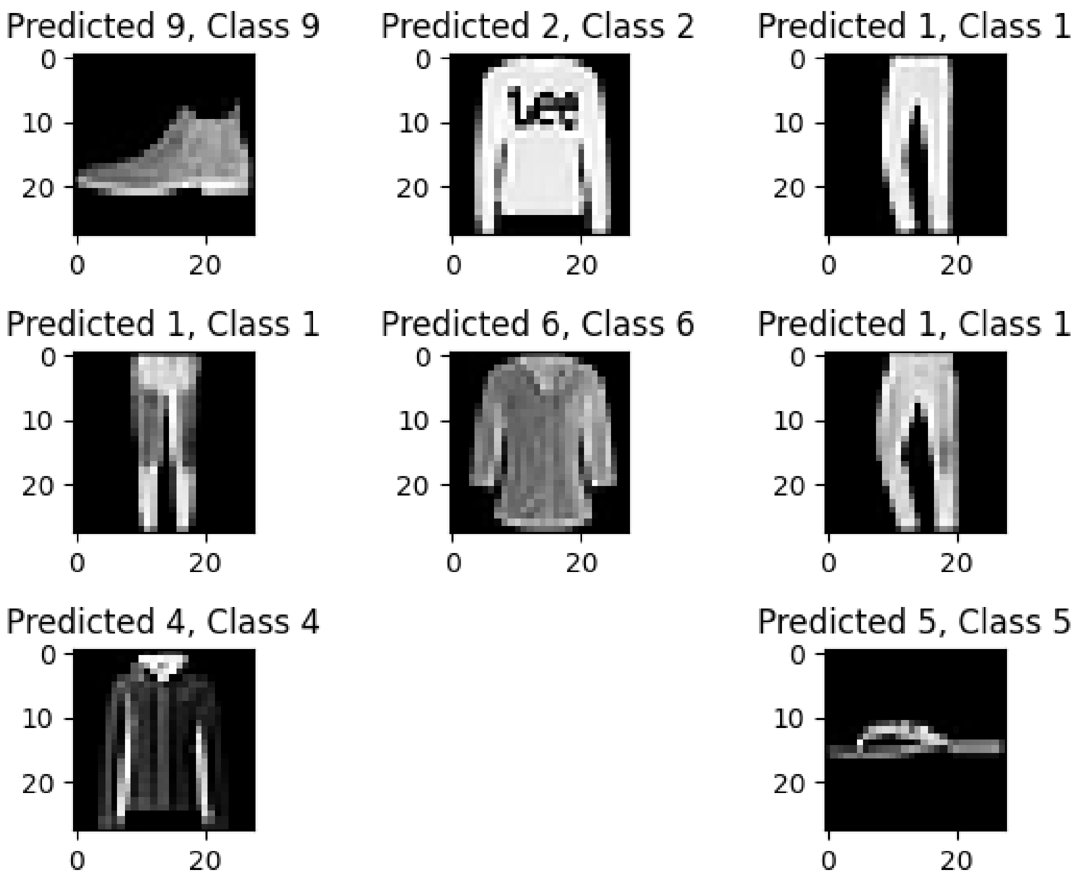
[ ]:
```
predicted_classes.shape, test_Y.shape
```

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

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

```
Found 9176 correct labels
```

```
<ipython-input-37-0178221d62f4>:4: MatplotlibDeprecationWarning: Auto-removal of
overlapping axes is deprecated since 3.6 and will be removed two minor releases
later; explicitly call ax.remove() as needed.
  plt.subplot(3,3,i+1)
```

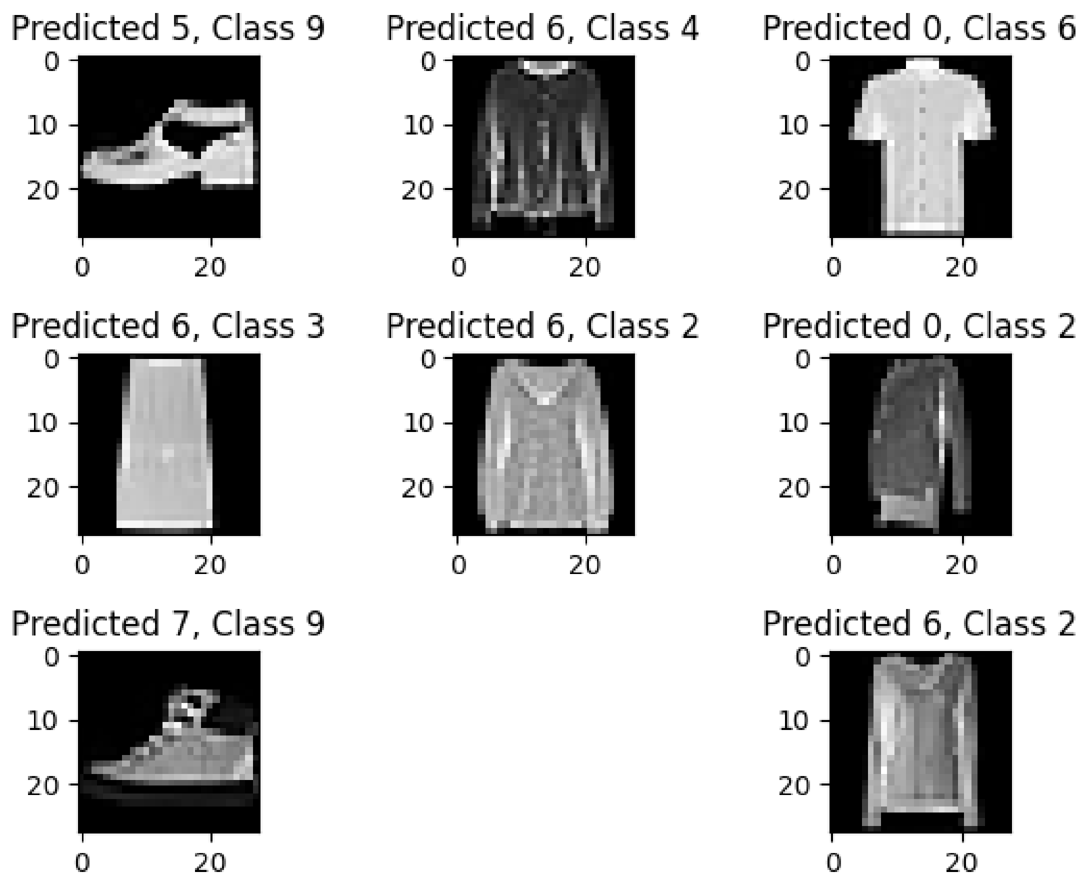


[ ]:
```python
incorrect = np.where(predicted_classes!=test_Y)[0]
print("Found %d incorrect labels" % len(incorrect))
for i, incorrect in enumerate(incorrect[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(test_X[incorrect].reshape(28,28), cmap='gray', interpolation='none')
    plt.title("Predicted {},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

```
Found 824 incorrect labels
```

```
<ipython-input-38-0bf9e7d6e015>:4: MatplotlibDeprecationWarning: Auto-removal of
overlapping axes is deprecated since 3.6 and will be removed two minor releases
later; explicitly call ax.remove() as needed.
```

```
plt.subplot(3,3,i+1)
```



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

```
              precision    recall  f1-score   support

     Class 0       0.79      0.89      0.84      1000
     Class 1       0.99      0.98      0.99      1000
     Class 2       0.93      0.81      0.87      1000
     Class 3       0.93      0.91      0.92      1000
     Class 4       0.86      0.89      0.87      1000
     Class 5       0.99      0.99      0.99      1000
     Class 6       0.77      0.77      0.77      1000
     Class 7       0.96      0.98      0.97      1000
     Class 8       0.99      0.99      0.99      1000
     Class 9       0.98      0.96      0.97      1000
```

```
    accuracy                           0.92     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.92      0.92      0.92     10000
```