```
B211038- Srushti Gavale

Fashion MNIST
```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
```

In [2]:
```
# Load the data
train_df = pd.read_csv('fashion-mnist_train.csv')
test_df = pd.read_csv('fashion-mnist_test.csv')
```

In [3]:
```
train_df.head(20)
```

Out[3]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixe
0	2	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
1	9	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
2	6	0	0	0	0	0	0	0	5	0	...	0.0	0.0	0.0	30.0	43.0	0.0	
3	0	0	0	0	1	2	0	0	0	0	...	3.0	0.0	0.0	0.0	0.0	1.0	
4	3	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
5	4	0	0	0	5	4	5	5	3	5	...	7.0	8.0	7.0	4.0	3.0	7.0	
6	4	0	0	0	0	0	0	0	0	0	...	14.0	0.0	0.0	0.0	0.0	0.0	
7	5	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
8	4	0	0	0	0	0	0	3	2	0	...	1.0	0.0	0.0	0.0	0.0	0.0	
9	8	0	0	0	0	0	0	0	0	0	...	203.0	214.0	166.0	0.0	0.0	0.0	
10	0	0	0	0	0	1	0	0	0	0	...	164.0	177.0	163.0	0.0	0.0	1.0	
11	8	0	0	0	0	0	0	0	0	0	...	9.0	10.0	9.0	9.0	8.0	1.0	
12	9	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
13	0	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
14	2	0	0	0	0	1	1	0	0	0	...	0.0	0.0	118.0	190.0	162.0	82.0	
15	2	0	0	0	0	0	0	0	0	16	...	0.0	0.0	1.0	1.0	1.0	1.0	
16	9	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
17	3	0	0	0	0	0	0	0	0	0	...	101.0	20.0	0.0	0.0	1.0	0.0	
18	3	0	0	0	0	0	0	0	0	0	...	0.0	11.0	15.0	0.0	0.0	0.0	
19	3	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	

20 rows × 785 columns

In [4]:

```
train_df.tail(20)
```

Out[4]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	p
2345	1	0	0	0	0	0	0	0	160	172	...	175.0	198.0	148.0	0.0	0.0	0.0	
2346	2	0	0	0	0	0	1	0	0	0	...	0.0	1.0	0.0	35.0	90.0	35.0	
2347	5	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
2348	5	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
2349	8	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
2350	1	0	0	0	0	0	4	0	19	172	...	145.0	135.0	134.0	141.0	11.0	0.0	
2351	9	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
2352	6	0	0	0	0	0	0	2	0	0	...	0.0	1.0	0.0	20.0	0.0	0.0	
2353	5	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
2354	1	0	0	0	0	0	0	0	0	0	...	135.0	53.0	0.0	0.0	0.0	0.0	
2355	9	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
2356	3	0	0	0	0	0	0	0	4	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
2357	3	0	0	0	0	0	0	0	0	0	...	125.0	1.0	0.0	0.0	0.0	0.0	
2358	2	0	0	0	0	1	0	1	0	1	...	91.0	37.0	0.0	69.0	157.0	55.0	
2359	8	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
2360	7	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
2361	7	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
2362	7	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	
2363	0	0	0	0	0	1	0	0	39	44	...	99.0	107.0	94.0	85.0	86.0	18.0	
2364	3	0	0	0	0	0	1	0	0	0	...	NaN	NaN	NaN	NaN	NaN	NaN	

20 rows × 785 columns

In [5]:

```
train_df.label.unique
```

Out[5]:

```
<bound method Series.unique of 0       2
1       9
2       6
3       0
4       3
       ..
2360    7
2361    7
2362    7
2363    0
2364    3
Name: label, Length: 2365, dtype: int64>
```

In [6]:

```
train_df.shape
```

Out[6]:

```
(2365, 785)
```

In [7]:

```
test_df.shape
```

Out[7]:

```
(2357, 785)
```

In [8]:

```
# Prepare the data
X_train = train_df.iloc[:, 1:].values.astype('float32') / 255.0
y_train = train_df.iloc[:, 0].values.astype('int32')
X_test = test_df.iloc[:, 1:].values.astype('float32') / 255.0
y_test = test_df.iloc[:, 0].values.astype('int32')
```

In [9]:

```
X_train = X_train.reshape((-1, 28, 28, 1))
X_test = X_test.reshape((-1, 28, 28, 1))
```

In [10]:

```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

In [11]:
```
X_train
```

Out[11]:
```
array([[[[0.        ],
         [0.        ],
         [0.        ],
         ...,
         [0.        ],
         [0.        ],
         [0.        ]],

        [[0.        ],
         [0.        ],
         [0.        ],
         ...,
         [0.        ],
         [0.        ],
         [0.        ]],

        [[0.        ],
         [0.        ],
         [0.        ],
```

In [12]:
```
# Define the model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', padding='same', input_shape=(28,28,1)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation='relu', padding='same'),
    MaxPooling2D((2,2)),
    Conv2D(128, (3,3), activation='relu', padding='same'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])
```

In [13]:
```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [14]:
```
# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=128, validation_split=0.2)
```

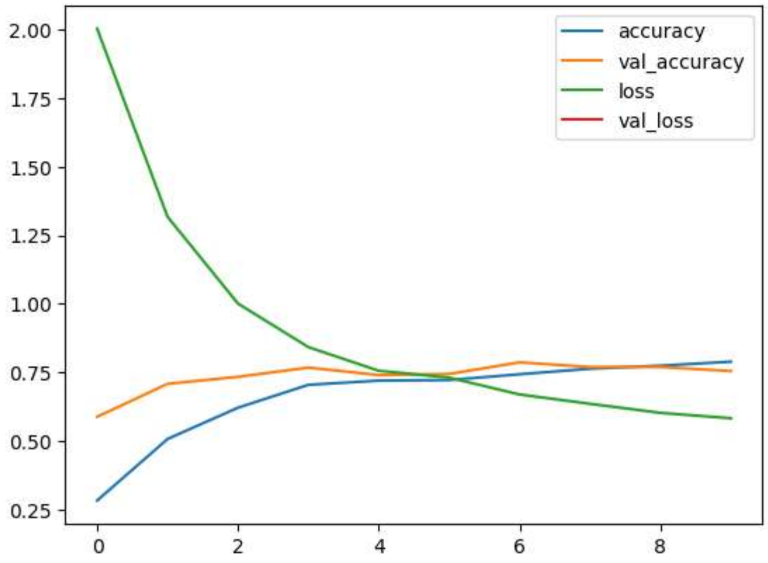
```
Epoch 1/10
15/15 [==============================] - 6s 283ms/step - loss: 2.0041 - accuracy: 0.2822 - val_loss: nan - val_acc
uracy: 0.5877
Epoch 2/10
15/15 [==============================] - 6s 405ms/step - loss: 1.3178 - accuracy: 0.5069 - val_loss: nan - val_acc
uracy: 0.7082
Epoch 3/10
15/15 [==============================] - 7s 467ms/step - loss: 1.0010 - accuracy: 0.6210 - val_loss: nan - val_acc
uracy: 0.7336
Epoch 4/10
15/15 [==============================] - 3s 218ms/step - loss: 0.8421 - accuracy: 0.7045 - val_loss: nan - val_acc
uracy: 0.7674
Epoch 5/10
15/15 [==============================] - 3s 218ms/step - loss: 0.7558 - accuracy: 0.7199 - val_loss: nan - val_acc
uracy: 0.7400
Epoch 6/10
15/15 [==============================] - 5s 329ms/step - loss: 0.7307 - accuracy: 0.7220 - val_loss: nan - val_acc
uracy: 0.7442
Epoch 7/10
15/15 [==============================] - 3s 227ms/step - loss: 0.6692 - accuracy: 0.7431 - val_loss: nan - val_acc
uracy: 0.7865
Epoch 8/10
15/15 [==============================] - 3s 224ms/step - loss: 0.6353 - accuracy: 0.7632 - val_loss: nan - val_acc
uracy: 0.7696
Epoch 9/10
15/15 [==============================] - 4s 243ms/step - loss: 0.6022 - accuracy: 0.7743 - val_loss: nan - val_acc
uracy: 0.7696
Epoch 10/10
15/15 [==============================] - 4s 287ms/step - loss: 0.5825 - accuracy: 0.7891 - val_loss: nan - val_acc
uracy: 0.7548
```

In [15]:
```
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
```

```
74/74 [==============================] - 1s 17ms/step - loss: nan - accuracy: 0.8044
Test accuracy: 0.8044123649597168
```

In [16]:

```
# Plot the accuracy and loss for training and validation data
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.legend()
plt.show()
```



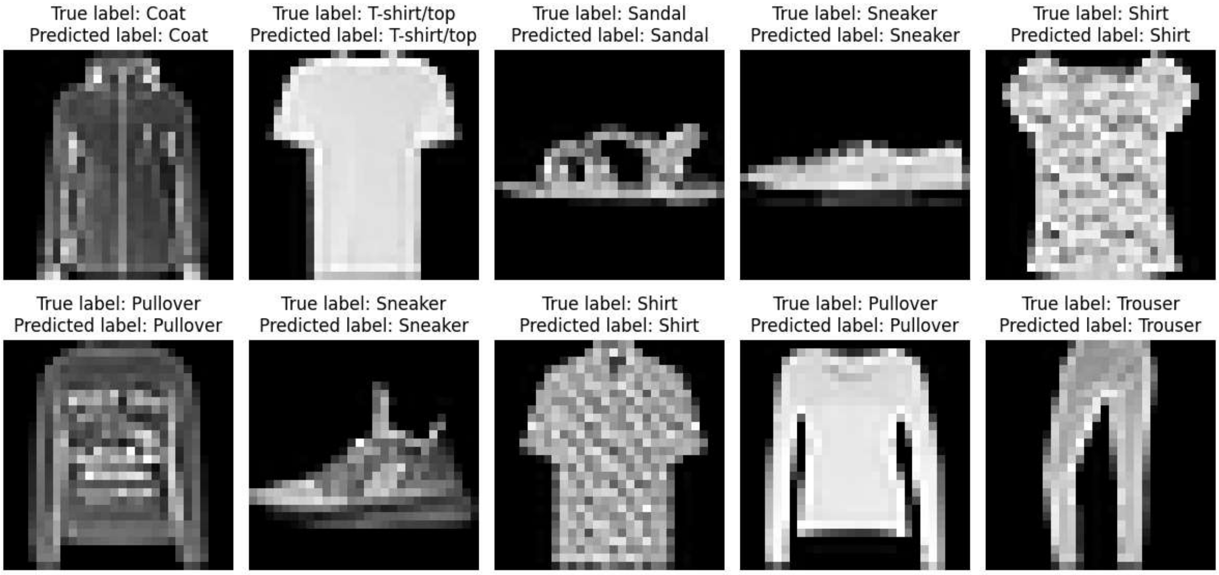
In [19]:

```
model.save('fashion_mnist_cnn.h5')
```

In [21]:

```
from keras.models import load_model
# Load the saved model
model = load_model('fashion_mnist_cnn.h5')

# Load the test dataset
test_data = pd.read_csv('fashion-mnist_test.csv')

# Extract the image data and labels
test_images = np.array(test_data.iloc[:, 1:])
test_labels = np.array(test_data.iloc[:, 0])

# Define the labels dictionary
labels = {
    0: 'T-shirt/top',
    1: 'Trouser',
    2: 'Pullover',
    3: 'Dress',
    4: 'Coat',
    5: 'Sandal',
    6: 'Shirt',
    7: 'Sneaker',
    8: 'Bag',
    9: 'Ankle boot'
}

# Choose 10 random images from the test set
indices = np.random.choice(test_images.shape[0], size=10, replace=False)
images = test_images[indices]
true_labels = test_labels[indices]

# Reshape the images to a 4D array
images = images.reshape(-1, 28, 28, 1)

# Make predictions on the images
predictions = model.predict(images)

# Plot the images with their true labels and predicted labels
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(12, 6))
axes = axes.flatten()
for i, ax in enumerate(axes):
    # Plot the image
    ax.imshow(images[i].reshape(28, 28), cmap='gray')
    ax.set_title('True label: {}\nPredicted label: {}'.format(labels[true_labels[i]], labels[np.argmax(predictions[
    ax.axis('off')
plt.tight_layout()
plt.show()
```

```
1/1 [==============================] - 0s 266ms/step
```



In [ ]: