In [1]:

```
import tensorflow as tf
import keras
import matplotlib.pyplot as plt
import numpy as np
import pydot
import graphviz
```

In [2]:

```
# Loading the Fashion MNIST dataset and looking at the shape of the training and test
fashion_mnist = tf.keras.datasets.fashion_mnist
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
print('Training Set: {} and Training Targets: {}'.format(X_train.shape, y_train.shape)
print('Test Set: {} and Test Targets: {}'.format(X_test.shape, y_test.shape))
```

```
Training Set: (60000, 28, 28) and Training Targets: (60000,)
Test Set: (10000, 28, 28) and Test Targets: (10000,)
```

Convolutional Neural Network (CNN)

In [3]:

```
# Reshaping the dataset to feed to the CNN
X_train_1 = np.reshape(X_train, [X_train.shape[0], X_train.shape[1], X_train.shape[2],
X_test_1 = np.reshape(X_test, [X_test.shape[0], X_test.shape[1], X_test.shape[2], 1])
print('training data:', X_train_1.shape, 'test data:', X_test_1.shape)

# Converting pixel values to the range 0 to 1
X_train_1 = X_train_1.astype('float32')
X_test_1 = X_test_1.astype('float32')
X_train_1 = X_train_1 / 255
X_test_1 = X_test_1 / 255

# Converting categorical class labels to one-hot encoding vectors
y_train_1 = keras.utils.to_categorical(y_train)
y_test_1 = keras.utils.to_categorical(y_test)
```

```
training data: (60000, 28, 28, 1) test data: (10000, 28, 28, 1)
```

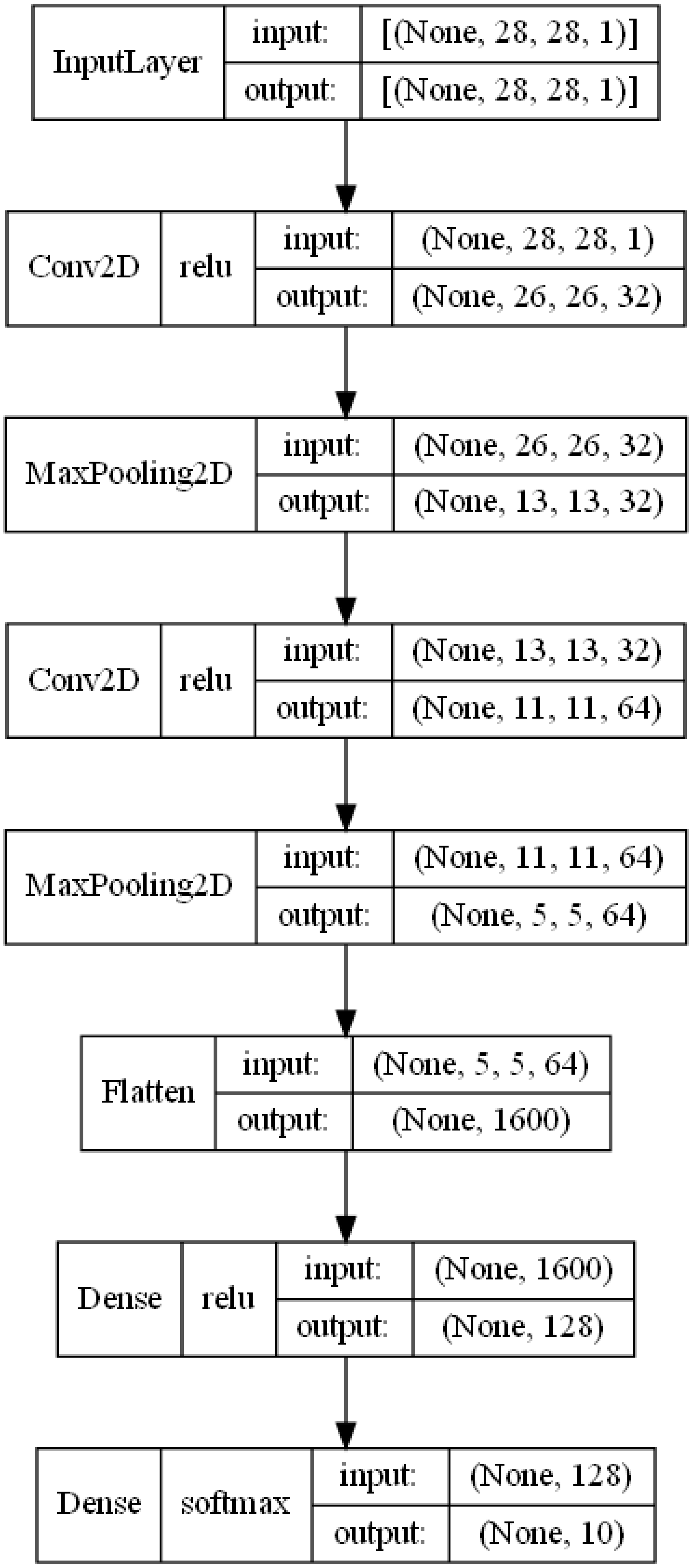
In [4]:

```
# CNN Model
model1 = keras.models.Sequential(
    [
        keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (28, 28, 1)
        keras.layers.MaxPooling2D((2,2)),
        keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
        keras.layers.MaxPooling2D((2,2)),
        keras.layers.Flatten(),
        keras.layers.Dense(128, activation = 'relu'),
        keras.layers.Dense(10, activation = 'softmax')
    ]
)

# Compilation of CNN model
model1.compile(optimizer = keras.optimizers.Adam(),
              loss = keras.losses.categorical_crossentropy,
              metrics = ['accuracy'])

model1.summary()
```

```
# Plotting the CNN Model Architecture - Included in Final Report
tf.keras.utils.plot_model(model1, to_file='model1.png', show_shapes = True, show_layer
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 1600)              0         
                                                                 
 dense (Dense)               (None, 128)               204928    
                                                                 
 dense_1 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 225,034
Trainable params: 225,034
Non-trainable params: 0
_________________________________________________________________
```

Out[4]:



In [5]:

```
# Monitoring the 'val_accuracy' and stopping the training early if 'val_accuracy' is r
early_stopping = keras.callbacks.EarlyStopping(
    monitor = 'val_accuracy', patience = 5
)

# Training the Model
```

```
h2 = model1.fit(X_train_1, y_train_1,
               batch_size = 128,
               epochs = 100,
               validation_split = 0.25,
               callbacks = [early_stopping],
               verbose = 1)
```

```
Epoch 1/100
352/352 [==============================] - 13s 37ms/step - loss: 0.5897 - accuracy:
0.7881 - val_loss: 0.4274 - val_accuracy: 0.8449
Epoch 2/100
352/352 [==============================] - 15s 42ms/step - loss: 0.3697 - accuracy:
0.8669 - val_loss: 0.3451 - val_accuracy: 0.8755
Epoch 3/100
352/352 [==============================] - 15s 43ms/step - loss: 0.3197 - accuracy:
0.8839 - val_loss: 0.3061 - val_accuracy: 0.8886
Epoch 4/100
352/352 [==============================] - 15s 42ms/step - loss: 0.2867 - accuracy:
0.8961 - val_loss: 0.2957 - val_accuracy: 0.8921
Epoch 5/100
352/352 [==============================] - 15s 43ms/step - loss: 0.2640 - accuracy:
0.9038 - val_loss: 0.2785 - val_accuracy: 0.8991
Epoch 6/100
352/352 [==============================] - 15s 43ms/step - loss: 0.2458 - accuracy:
0.9104 - val_loss: 0.2689 - val_accuracy: 0.9021
Epoch 7/100
352/352 [==============================] - 15s 44ms/step - loss: 0.2279 - accuracy:
0.9176 - val_loss: 0.2792 - val_accuracy: 0.8983
Epoch 8/100
352/352 [==============================] - 15s 43ms/step - loss: 0.2127 - accuracy:
0.9229 - val_loss: 0.2650 - val_accuracy: 0.9014
Epoch 9/100
352/352 [==============================] - 16s 45ms/step - loss: 0.1980 - accuracy:
0.9273 - val_loss: 0.2692 - val_accuracy: 0.9033
Epoch 10/100
352/352 [==============================] - 16s 44ms/step - loss: 0.1842 - accuracy:
0.9320 - val_loss: 0.2678 - val_accuracy: 0.9042
Epoch 11/100
352/352 [==============================] - 16s 46ms/step - loss: 0.1693 - accuracy:
0.9376 - val_loss: 0.2785 - val_accuracy: 0.9020
Epoch 12/100
352/352 [==============================] - 16s 46ms/step - loss: 0.1595 - accuracy:
0.9419 - val_loss: 0.2546 - val_accuracy: 0.9116
Epoch 13/100
352/352 [==============================] - 16s 46ms/step - loss: 0.1483 - accuracy:
0.9446 - val_loss: 0.2732 - val_accuracy: 0.9055
Epoch 14/100
352/352 [==============================] - 17s 47ms/step - loss: 0.1388 - accuracy:
0.9490 - val_loss: 0.2570 - val_accuracy: 0.9121
Epoch 15/100
352/352 [==============================] - 17s 48ms/step - loss: 0.1222 - accuracy:
0.9556 - val_loss: 0.2789 - val_accuracy: 0.9101
Epoch 16/100
352/352 [==============================] - 16s 45ms/step - loss: 0.1170 - accuracy:
0.9572 - val_loss: 0.2708 - val_accuracy: 0.9153
Epoch 17/100
352/352 [==============================] - 16s 47ms/step - loss: 0.1088 - accuracy:
0.9596 - val_loss: 0.2983 - val_accuracy: 0.9042
Epoch 18/100
352/352 [==============================] - 16s 46ms/step - loss: 0.0970 - accuracy:
0.9643 - val_loss: 0.2903 - val_accuracy: 0.9086
Epoch 19/100
352/352 [==============================] - 16s 46ms/step - loss: 0.0887 - accuracy:
0.9670 - val_loss: 0.2867 - val_accuracy: 0.9126
Epoch 20/100
352/352 [==============================] - 18s 51ms/step - loss: 0.0802 - accuracy:
0.9714 - val_loss: 0.3277 - val_accuracy: 0.9070
```

```
Epoch 21/100
352/352 [==============================] - 17s 49ms/step - loss: 0.0713 - accuracy:
0.9739 - val_loss: 0.3411 - val_accuracy: 0.9087
```

In [6]:

```
# Evaluating Model on Training Set
y_train_pred = np.argmax(model1.predict(X_train_1), axis=-1)
train_acc = sum(y_train == y_train_pred)/y_train.shape[0]

# Evaluating Model on Test Set
y_test_pred = np.argmax(model1.predict(X_test_1), axis=-1)
test_acc = sum(y_test == y_test_pred)/y_test.shape[0]

print('training acc is', train_acc)
print('test acc is', test_acc)
```

```
1875/1875 [==============================] - 7s 4ms/step
313/313 [==============================] - 1s 4ms/step
training acc is 0.9566333333333333
test acc is 0.9033
```

In [ ]: