# Use-Case

Create a commitee of three neural networks using Fashion MNIST dataset. The commitee of deep-learning models can be formed by combining diverse deep learning models at the score level. The diverse deep learning models can be formed by varying their architectures. The committee of the deep learning models be formed by averaging the output probability values. Please evaluate each of the three deep-learning models individually and in a committee on Fashion MNIST dataset and report their individual accuracy rates along with the final accuracy of the committee.

# Fashion MNIST

Link: https://github.com/zalandoresearch/fashion-mnist

## Dataset Description:

The Fashion-MNIST images consisting of 70,000 28*28 grayscale images of fashion products from 10 categories from a dataset of Zalando article images, with 7,000 images per category. The training set consists of 60,000 images and the test set consists of 10,000 images. The set of images in the Fashion MNIST database was created in 2017 to pose a more challenging classification task than the simple MNIST digits data, which saw performance reaching upwards of 99.7%

**class_names**

Each training and test example is assigned to one of the following class_names:

| Label | Description |
|---|---|
| 0 | T-shirt/top |
| 1 | Trouser |
| 2 | Pullover |
| 3 | Dress |
| 4 | Coat |
| 5 | Sandal |

| Label | Description |
|---|---|
| 6 | Shirt |
| 7 | Sneaker |
| 8 | Bag |
| 9 | Ankle boot |

## Deep Learning Models

For the Image Classification prediction task, we will use three Convolutional Neural Network (CNN) with different architecture and hyper-parameters to tune them. Also, we will create a committee of three different CNN Deep Learning models by averaging the probability values from these models using the Tensorflow/Keras frameworks

## Deep Learning Framework

We will use TensorFlow/Keras backend for the classification of Fashion MNIST images using various layers like Sequential, Conv2D, MaxPooling2D, Dropout, Flatten, Dense, etc. and optimizer, loss functions, evaluation metric, etc.

## Load packages

In [1]:

```
# Warning Libraries
import warnings
warnings.filterwarnings("ignore")

# Data handle Libraries
import pandas as pd
import numpy as np

# Visualization Libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Configure visualisations
%matplotlib inline
sns.set(context="notebook", style='whitegrid', color_codes=True)

# Dataset Library
```

```
from tensorflow.keras.datasets import fashion_mnist

# Deep Learning Libraries
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization, Dropout
from keras.optimizers import Adam, SGD, Adagrad, Adadelta, RMSprop
from keras.callbacks import EarlyStopping, ReduceLROnPlateau, LearningRateScheduler
from keras.utils import to_categorical

# Evaluation Libraries
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

```
2022-11-24 20:17:30.541369: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized wi
th oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operation
s:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-11-24 20:17:31.256938: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libnvinfer.so.7'; dlerror: libnvinfer.so.7: cannot open shared object file: No such file or directory
2022-11-24 20:17:31.257020: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libnvinfer_plugin.so.7'; dlerror: libnvinfer_plugin.so.7: cannot open shared object file: No such file
or directory
2022-11-24 20:17:31.257027: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Cannot dlopen some
TensorRT libraries. If you would like to use Nvidia GPU with TensorRT, please make sure the missing libraries mentioned
above are installed properly.
```

In [2]:

```
# OS Library
import os
CPU_COUNT = os.cpu_count()
print('CPU_COUNT:', CPU_COUNT)
```

```
CPU_COUNT: 4
```

## Extract dataset

In [3]:

```
# Extract the Fashion MNIST training and testing dataset from Keras datasets
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

print('X_train shape:', X_train.shape)
print('y_train shape:', y_train.shape)
```

```
print('X_test shape:', X_test.shape)
print('y_test shape:', y_test.shape)
```

```
X_train shape: (60000, 28, 28)
y_train shape: (60000,)
X_test shape: (10000, 28, 28)
y_test shape: (10000,)
```

In [4]:
```
# X_train[:3]
```

In [5]:
```
# X_test[:3]
```

In [6]:
```
y_train[:3]
```

Out[6]:
```
array([9, 0, 0], dtype=uint8)
```

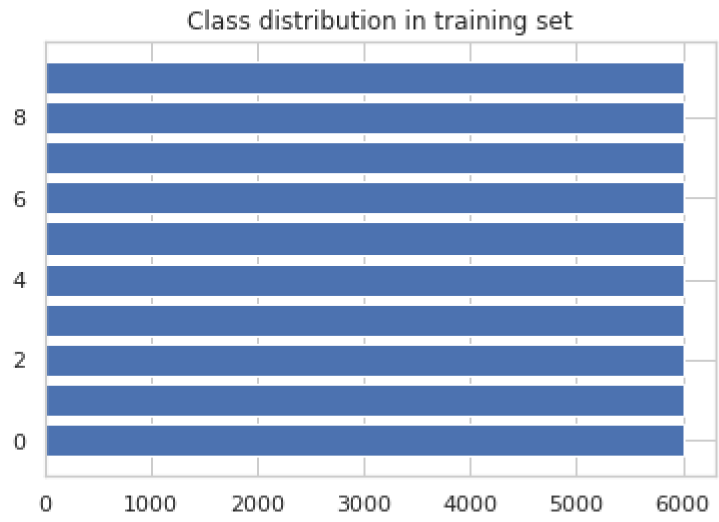
In [7]:
```
y_test[:3]
```

Out[7]:
```
array([9, 2, 1], dtype=uint8)
```

In [8]:
```
# Define the class_names of the dataset
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"
print('Labels in the dataset:', class_names)
```

```
Labels in the dataset: ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'An
kle boot']
```

In [9]:
```
# View unique class_names and distribution of class_names in the training dataset
train_classes, train_counts = np.unique(y_train, return_counts=True)

plt.title('Class distribution in training set')
plt.barh(train_classes, train_counts)
plt.show()
```




In [10]:
```
# View unique class_names and distribution of class_names in the testing dataset
test_classes, test_counts = np.unique(y_train, return_counts=True)

plt.title('Class distribution in testing set')
plt.barh(test_classes, test_counts)
plt.show()
```


## Visualize dataset

In [11]:

```
# Dataset visualization
W_grid = 15
L_grid = 5

fig, axes = plt.subplots(L_grid, W_grid, figsize=(15,5))
axes = axes.ravel()
n_train = len(X_train)

# Select a random number from 0 to n_train
for i in np.arange(0, W_grid*L_grid):
    index = np.random.randint(0, n_train)

    # read and display an image with the selected index
    axes[i].imshow(X_train[index,1:])
    label_index = int(y_train[index])
    axes[i].set_title(class_names[label_index], fontsize=10)
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.4)
```


## Pre-process dataset

### (a) Change Dimensions

In [12]:
```
X_train  = np.expand_dims(X_train, axis=-1)
X_test  = np.expand_dims(X_test, axis=-1)

print('X_train shape:', X_train.shape)
print('X_test shape:', X_test.shape)
```

```
X_train shape: (60000, 28, 28, 1)
X_test shape: (10000, 28, 28, 1)
```

### (b) Dataset Normalization

In [13]:
```
### Scale the dataset
X_train = X_train/255.0
X_test = X_test/255.0
```

In [14]:
```
# display(X_train[0])
```

In [15]:
```
# display(X_test[0])
```

### (c) One-hot Encoding of Labels

In [16]:
```
# Transform target variable into one-hot encoding
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

print('y_train shape:', y_train.shape)
print('y_test shape:', y_test.shape)
```

```
y_train shape: (60000, 10)
y_test shape: (10000, 10)
```

In [17]:
```
# View encoded training label dataset
y_train[0]
```

Out[17]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1.], dtype=float32)
```

In [18]:
```
# View encoded testing label dataset
y_test[0]
```

Out[18]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1.], dtype=float32)
```

---

## Convolutional Neural Network (CNN) Model Building

In [19]:
```
# Evaluation metrics
METRICS = [
    'accuracy',
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall')
]
```

```
2022-11-24 20:17:36.232723: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized wi
th oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operation
s:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-11-24 20:17:36.811344: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1613] Created device /job:localhost/repl
ica:0/task:0/device:GPU:0 with 14799 MB memory:  -> device: 0, name: Tesla T4, pci bus id: 0001:00:00.0, compute capabi
lity: 7.5
```

In [20]:
```
# EarlyStopping callback
early_stop = EarlyStopping(
    monitor = "val_loss",
    min_delta = 0,
    patience = 5,
    verbose = 10,
    mode = "auto",
    baseline = None,
    restore_best_weights = False
)
```

In [21]:
```
# LearningRateScheduler callback
def scheduler(epoch, lr):
    """
    This function keeps the initial learning rate for the first ten epochs, and
    decreases it exponentially after that.
    """
    if epoch < 10:
        return lr
    else:
        return lr * tf.math.exp(-0.1)
lr_scheduler = LearningRateScheduler(scheduler)
```

## CNN Model-1

In [22]:
```
# Model parameters for images and models
NUM_EPOCHS = 20
BATCH_SIZE = 32
VALIDATION_SPLIT_RATIO = 0.2
INPUT_SHAPE = (28, 28, 1)
```

In [23]:
```
# Build Model-1 architecture
model1 = Sequential()

# Convolutional Layer
model1.add(Conv2D(filters=32, kernel_size=3, strides=(2, 2), input_shape=INPUT_SHAPE, padding='valid', activation='relu
model1.add(MaxPooling2D(pool_size=(2, 2)))

# Flatten layer
model1.add(Flatten())
model1.add(Dense(256, activation='relu'))

# Output layer
model1.add(Dense(10, activation='softmax'))
```

### Model-1 Summary

In [24]:
```
# Model Summary
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 13, 13, 32)        320       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 6, 6, 32)         0         
 )                                                               
                                                                 
 flatten (Flatten)           (None, 1152)              0         
                                                                 
 dense (Dense)               (None, 256)               295168    
                                                                 
 dense_1 (Dense)             (None, 10)                2570      
                                                                 
=================================================================
Total params: 298,058
Trainable params: 298,058
Non-trainable params: 0
_________________________________________________________________
```

### Model-1 Compile

In [25]:

```
# Model Compile
model1.compile(
    loss = 'categorical_crossentropy',
    optimizer = 'adam',
    metrics = METRICS
)
```

### Model-1 Training

In [26]:

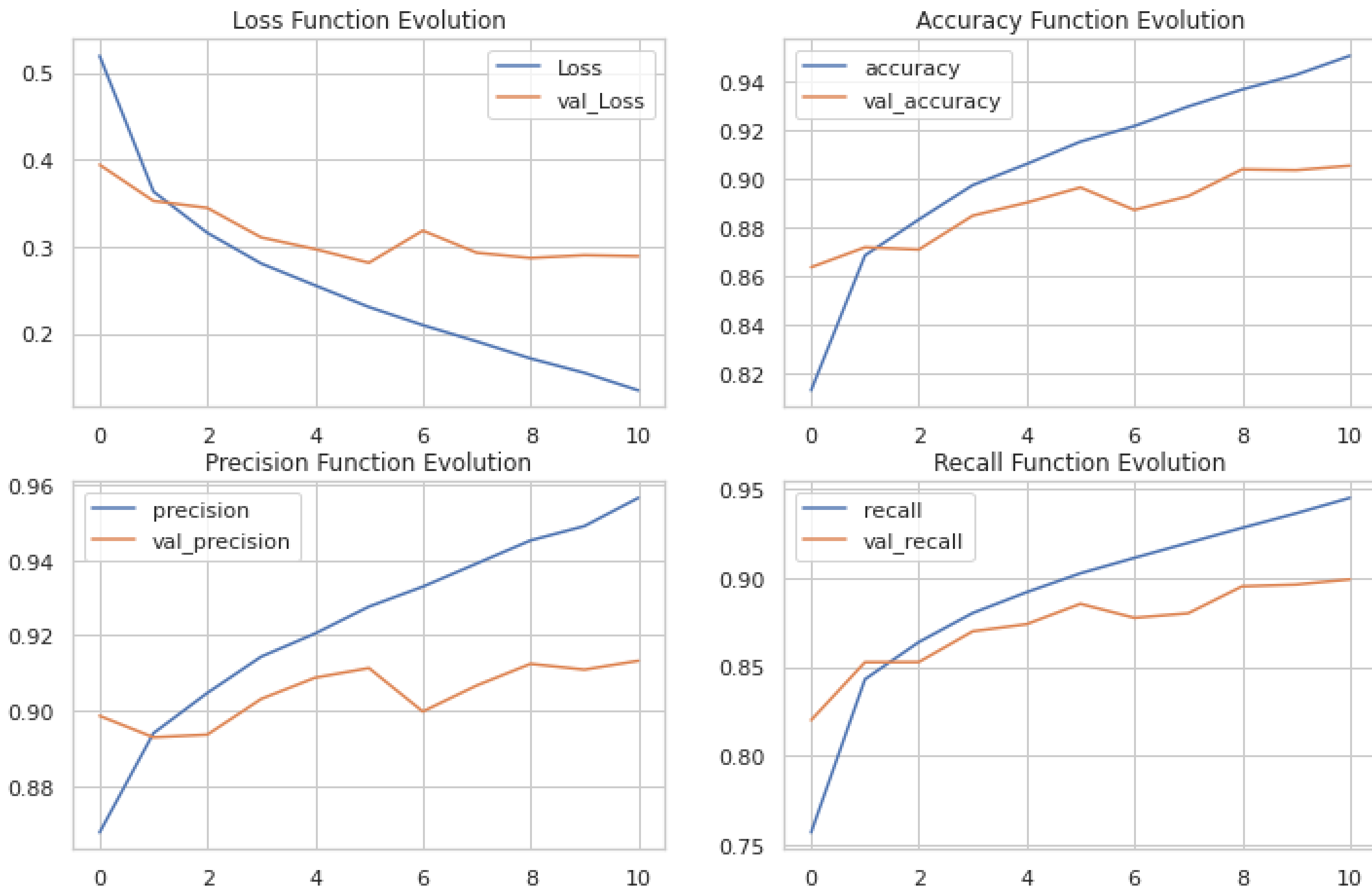
```
# Model Training
history1 = model1.fit(
    X_train,
    y_train,
    epochs = NUM_EPOCHS,
    batch_size = BATCH_SIZE,
    callbacks = [early_stop, lr_scheduler],
    validation_split = VALIDATION_SPLIT_RATIO,
    shuffle = True,
    workers = CPU_COUNT,
    use_multiprocessing = True
)
```

```
Epoch 1/20
2022-11-24 20:17:38.187505: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:428] Loaded cuDNN version 8201
2022-11-24 20:17:39.187708: I tensorflow/compiler/xla/service/service.cc:173] XLA service 0x7f015f6ec570 initialized fo
r platform CUDA (this does not guarantee that XLA will be used). Devices:
2022-11-24 20:17:39.187745: I tensorflow/compiler/xla/service/service.cc:181]   StreamExecutor device (0): Tesla T4, Co
mpute Capability 7.5
2022-11-24 20:17:39.191995: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:268] disabling MLIR crash rep
roducer, set env var `MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.
2022-11-24 20:17:39.310746: I tensorflow/compiler/jit/xla_compilation_cache.cc:477] Compiled cluster using XLA!  This l
ine is logged at most once for the lifetime of the process.
1500/1500 [==============================] - 7s 3ms/step - loss: 0.5193 - accuracy: 0.8130 - precision: 0.8676 - recal
l: 0.7565 - val_loss: 0.3940 - val_accuracy: 0.8636 - val_precision: 0.8986 - val_recall: 0.8198 - lr: 0.0010
Epoch 2/20
1500/1500 [==============================] - 4s 3ms/step - loss: 0.3632 - accuracy: 0.8684 - precision: 0.8940 - recal
l: 0.8429 - val_loss: 0.3524 - val_accuracy: 0.8717 - val_precision: 0.8929 - val_recall: 0.8525 - lr: 0.0010
Epoch 3/20
1500/1500 [==============================] - 4s 3ms/step - loss: 0.3157 - accuracy: 0.8832 - precision: 0.9047 - recal
l: 0.8639 - val_loss: 0.3444 - val_accuracy: 0.8708 - val_precision: 0.8936 - val_recall: 0.8526 - lr: 0.0010
Epoch 4/20
1500/1500 [==============================] - 4s 3ms/step - loss: 0.2804 - accuracy: 0.8973 - precision: 0.9143 - recal
l: 0.8802 - val_loss: 0.3105 - val_accuracy: 0.8848 - val_precision: 0.9031 - val_recall: 0.8699 - lr: 0.0010
Epoch 5/20
1500/1500 [==============================] - 4s 3ms/step - loss: 0.2553 - accuracy: 0.9059 - precision: 0.9205 - recal
l: 0.8919 - val_loss: 0.2970 - val_accuracy: 0.8901 - val_precision: 0.9088 - val_recall: 0.8739 - lr: 0.0010
Epoch 6/20
1500/1500 [==============================] - 4s 3ms/step - loss: 0.2304 - accuracy: 0.9151 - precision: 0.9277 - recal
l: 0.9026 - val_loss: 0.2813 - val_accuracy: 0.8963 - val_precision: 0.9113 - val_recall: 0.8854 - lr: 0.0010
Epoch 7/20
1500/1500 [==============================] - 4s 3ms/step - loss: 0.2096 - accuracy: 0.9215 - precision: 0.9329 - recal
l: 0.9113 - val_loss: 0.3182 - val_accuracy: 0.8871 - val_precision: 0.8998 - val_recall: 0.8775 - lr: 0.0010
Epoch 8/20
1500/1500 [==============================] - 4s 3ms/step - loss: 0.1909 - accuracy: 0.9295 - precision: 0.9391 - recal
l: 0.9198 - val_loss: 0.2928 - val_accuracy: 0.8928 - val_precision: 0.9067 - val_recall: 0.8801 - lr: 0.0010
Epoch 9/20
1500/1500 [==============================] - 4s 3ms/step - loss: 0.1712 - accuracy: 0.9365 - precision: 0.9452 - recal
l: 0.9282 - val_loss: 0.2868 - val_accuracy: 0.9038 - val_precision: 0.9124 - val_recall: 0.8953 - lr: 0.0010
Epoch 10/20
1500/1500 [==============================] - 4s 3ms/step - loss: 0.1547 - accuracy: 0.9425 - precision: 0.9490 - recal
l: 0.9364 - val_loss: 0.2900 - val_accuracy: 0.9034 - val_precision: 0.9109 - val_recall: 0.8963 - lr: 0.0010
Epoch 11/20
1500/1500 [==============================] - 4s 3ms/step - loss: 0.1345 - accuracy: 0.9504 - precision: 0.9565 - recal
l: 0.9451 - val_loss: 0.2888 - val_accuracy: 0.9053 - val_precision: 0.9132 - val_recall: 0.8992 - lr: 9.0484e-04
Epoch 11: early stopping
```

### Model-1 History Plot

In [27]:
```
# Model History Plot
plt.figure(figsize=(12, 16))

plt.subplot(4, 2, 1)
plt.plot(model1.history.history['loss'], label='Loss')
plt.plot(model1.history.history['val_loss'], label='val_Loss')
plt.title('Loss Function Evolution')
plt.legend()

plt.subplot(4, 2, 2)
plt.plot(model1.history.history['accuracy'], label='accuracy')
plt.plot(model1.history.history['val_accuracy'], label='val_accuracy')
plt.title('Accuracy Function Evolution')
plt.legend()

plt.subplot(4, 2, 3)
plt.plot(model1.history.history['precision'], label='precision')
plt.plot(model1.history.history['val_precision'], label='val_precision')
plt.title('Precision Function Evolution')
plt.legend()

plt.subplot(4, 2, 4)
plt.plot(model1.history.history['recall'], label='recall')
plt.plot(model1.history.history['val_recall'], label='val_recall')
plt.title('Recall Function Evolution')
plt.legend()
```

Out[27]:
```
<matplotlib.legend.Legend at 0x7f068447aac0>
```




### Model-1 Prediction

In [28]:
```python
# Model prediction
test_pred_prob1 = model1.predict(
    X_test,
    workers = CPU_COUNT,
    use_multiprocessing = True
)
print('Model-1 Probability Prediction:', test_pred_prob1[:5])
```

```
313/313 [==============================] - 0s 1ms/step
Model-1 Probability Prediction: [[9.0034838e-08 1.0565457e-08 8.0779659e-09 1.5250285e-08 1.5303561e-10
  1.4509588e-06 3.1406460e-09 2.5557805e-05 4.7956365e-07 9.9997234e-01]
 [1.0627570e-06 9.6917408e-15 9.9984300e-01 8.9641439e-08 5.2815452e-05
  3.4244711e-11 1.0296696e-04 7.1680312e-14 2.1741631e-10 1.8234346e-09]
 [1.1567830e-12 1.0000000e+00 3.8895636e-12 1.4267333e-10 1.7466944e-13
  4.9337548e-14 2.9360287e-13 1.6821461e-20 2.0849663e-15 9.6406292e-17]
 [9.6515270e-12 1.0000000e+00 8.8857758e-12 1.4159065e-09 2.1946537e-11
  4.7952483e-14 6.5169745e-12 1.7170084e-17 2.0769644e-15 7.0024392e-15]
 [5.0348222e-01 3.5983636e-08 1.9055469e-03 8.5907986e-06 3.2156964e-03
  5.3398995e-08 4.9138728e-01 1.0200069e-08 2.5463405e-09 5.3808697e-07]]
```

In [29]:

```python
# Label prediction
# test_pred_prob1 = [np.argmax(test_pred_prob1[i]) for i in range(len(test_pred_prob1))]
test_pred1 = np.argmax(test_pred_prob1, axis=1)
test_label1 = [class_names[i] for i in test_pred1]
print(test_label1[:5])
```

```
['Ankle boot', 'Pullover', 'Trouser', 'Trouser', 'T-shirt/top']
```

### Model-1 Evaluation

In [30]:

```python
# Model Evaluation
test_evaluation1 = model1.evaluate(
    X_test,
    y_test,
    workers = CPU_COUNT,
    use_multiprocessing=True
)
print('Test Loss:', round(test_evaluation1[0], 4))
print('Test Accuracy:', round(test_evaluation1[1], 4))
print('Test Precision:', round(test_evaluation1[2], 4))
print('Test Recall:', round(test_evaluation1[3], 4))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.3093 - accuracy: 0.9017 - precision: 0.9090 - recall:
0.8965
Test Loss: 0.3093
Test Accuracy: 0.9017
Test Precision: 0.909
Test Recall: 0.8965
```

---

## CNN Model-2

In [31]:
```
# Model parameters for images and models
NUM_EPOCHS = 20
BATCH_SIZE = 16
VALIDATION_SPLIT_RATIO = 0.2
INPUT_SHAPE = (28, 28, 1)
```

In [32]:
```
# Build Model-2 architecture
model2 = Sequential()

# Convolutional Layer
model2.add(Conv2D(filters=64, kernel_size=2, strides=(2, 2), input_shape=INPUT_SHAPE, padding='valid', activation='relu
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.2))

# Convolutional Layer
model2.add(Conv2D(filters=32, kernel_size=2, strides=(2, 2), input_shape=INPUT_SHAPE, padding='valid', activation='relu
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.2))

# Flatten layer
model2.add(Flatten())
model2.add(Dense(128, activation='relu'))
model2.add(Dropout(0.2))

# Output layer
model2.add(Dense(10, activation='softmax'))
```

### Model-2 Summary

In [33]:
```
# Model Summary
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_1 (Conv2D)           (None, 14, 14, 64)        320       
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 7, 7, 64)         0         
 2D)                                                             
                                                                 
 dropout (Dropout)           (None, 7, 7, 64)          0         
                                                                 
 conv2d_2 (Conv2D)           (None, 3, 3, 32)          8224      
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 1, 1, 32)         0         
 2D)                                                             
                                                                 
 dropout_1 (Dropout)         (None, 1, 1, 32)          0         
                                                                 
 flatten_1 (Flatten)         (None, 32)                0         
                                                                 
 dense_2 (Dense)             (None, 128)               4224      
                                                                 
 dropout_2 (Dropout)         (None, 128)               0         
                                                                 
 dense_3 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 14,058
Trainable params: 14,058
Non-trainable params: 0
_________________________________________________________________
```

### Model-2 Compile

In [53]:
```
# Model Compile
model2.compile(
    loss = 'categorical_crossentropy',
    optimizer = 'sgd',
    metrics = METRICS
)
```

### Model-2 Training

In [54]:
```
# Model Training
```

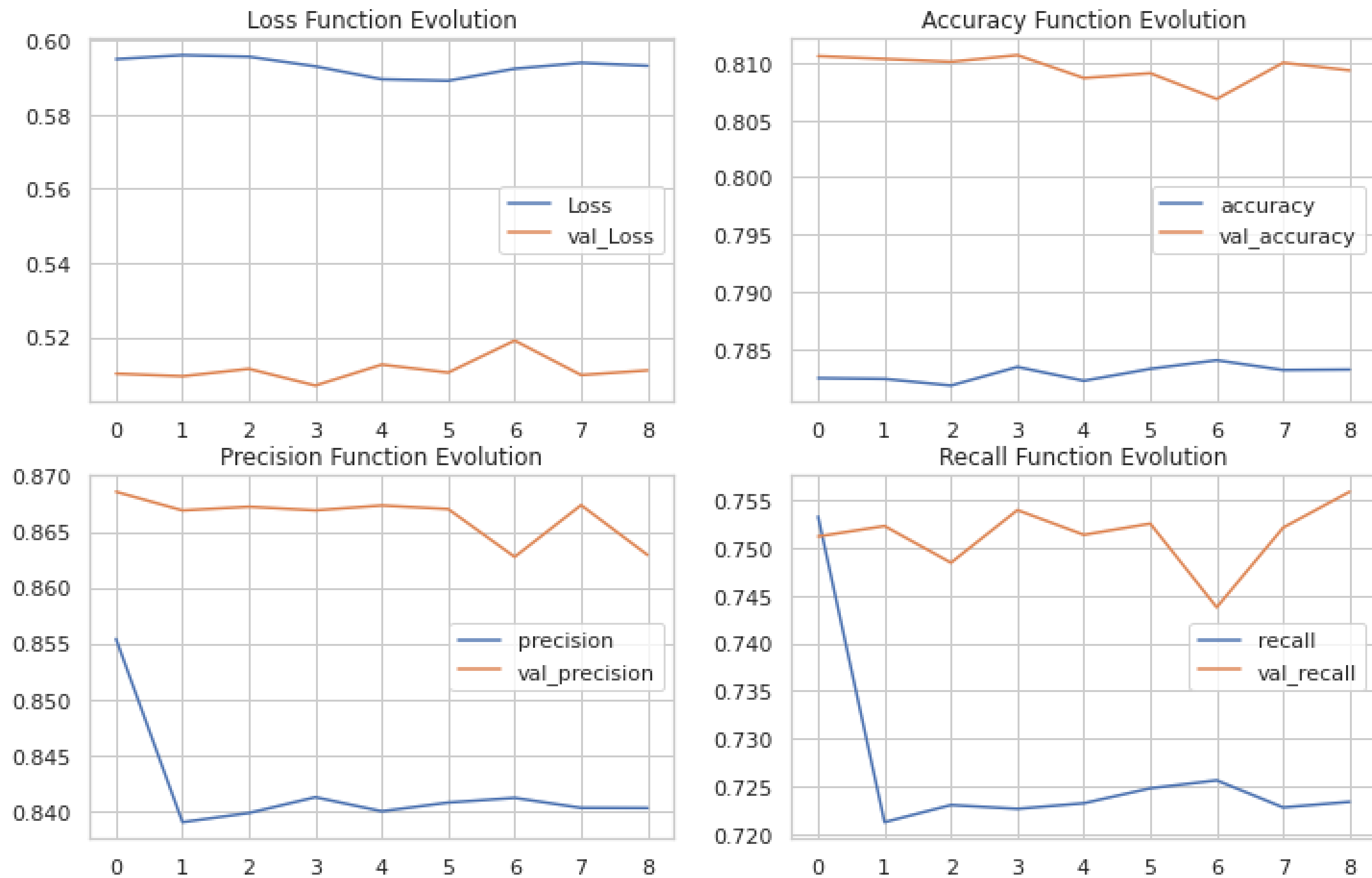
```python
history2 = model2.fit(
    X_train,
    y_train,
    epochs = NUM_EPOCHS,
    batch_size = BATCH_SIZE,
    callbacks = [early_stop, lr_scheduler],
    validation_split = VALIDATION_SPLIT_RATIO,
    shuffle = True,
    workers = CPU_COUNT,
    use_multiprocessing = True
)
```

```
Epoch 1/20
2022-11-24 20:27:28.831860: E tensorflow/core/grappler/optimizers/meta_optimizer.cc:954] layout failed: INVALID_ARGUMEN
T: Size of values 0 does not match size of permutation 4 @ fanin shape insequential_1/dropout/dropout/SelectV2-2-Transp
oseNHWCToNCHW-LayoutOptimizer
750/750 [==============================] - 4s 4ms/step - loss: 0.5950 - accuracy: 0.7824 - precision: 0.8554 - recall:
0.7532 - val_loss: 0.5101 - val_accuracy: 0.8106 - val_precision: 0.8686 - val_recall: 0.7512 - lr: 0.0100
Epoch 2/20
750/750 [==============================] - 3s 3ms/step - loss: 0.5961 - accuracy: 0.7824 - precision: 0.8390 - recall:
0.7213 - val_loss: 0.5094 - val_accuracy: 0.8103 - val_precision: 0.8669 - val_recall: 0.7523 - lr: 0.0100
Epoch 3/20
750/750 [==============================] - 3s 4ms/step - loss: 0.5957 - accuracy: 0.7818 - precision: 0.8398 - recall:
0.7231 - val_loss: 0.5114 - val_accuracy: 0.8101 - val_precision: 0.8672 - val_recall: 0.7484 - lr: 0.0100
Epoch 4/20
750/750 [==============================] - 3s 3ms/step - loss: 0.5931 - accuracy: 0.7834 - precision: 0.8412 - recall:
0.7227 - val_loss: 0.5069 - val_accuracy: 0.8107 - val_precision: 0.8669 - val_recall: 0.7539 - lr: 0.0100
Epoch 5/20
750/750 [==============================] - 3s 3ms/step - loss: 0.5896 - accuracy: 0.7822 - precision: 0.8400 - recall:
0.7233 - val_loss: 0.5125 - val_accuracy: 0.8087 - val_precision: 0.8673 - val_recall: 0.7513 - lr: 0.0100
Epoch 6/20
750/750 [==============================] - 3s 4ms/step - loss: 0.5892 - accuracy: 0.7832 - precision: 0.8407 - recall:
0.7248 - val_loss: 0.5104 - val_accuracy: 0.8091 - val_precision: 0.8670 - val_recall: 0.7525 - lr: 0.0100
Epoch 7/20
750/750 [==============================] - 3s 3ms/step - loss: 0.5924 - accuracy: 0.7840 - precision: 0.8411 - recall:
0.7257 - val_loss: 0.5190 - val_accuracy: 0.8068 - val_precision: 0.8627 - val_recall: 0.7437 - lr: 0.0100
Epoch 8/20
750/750 [==============================] - 3s 4ms/step - loss: 0.5940 - accuracy: 0.7831 - precision: 0.8403 - recall:
0.7228 - val_loss: 0.5097 - val_accuracy: 0.8100 - val_precision: 0.8674 - val_recall: 0.7521 - lr: 0.0100
Epoch 9/20
750/750 [==============================] - 3s 3ms/step - loss: 0.5932 - accuracy: 0.7832 - precision: 0.8402 - recall:
0.7234 - val_loss: 0.5110 - val_accuracy: 0.8093 - val_precision: 0.8629 - val_recall: 0.7558 - lr: 0.0100
Epoch 9: early stopping
```

### Model-2 History Plot

In [55]:
```
# Model History Plot
plt.figure(figsize=(12, 16))

plt.subplot(4, 2, 1)
plt.plot(model2.history.history['loss'], label='Loss')
plt.plot(model2.history.history['val_loss'], label='val_Loss')
plt.title('Loss Function Evolution')
plt.legend()

plt.subplot(4, 2, 2)
plt.plot(model2.history.history['accuracy'], label='accuracy')
plt.plot(model2.history.history['val_accuracy'], label='val_accuracy')
plt.title('Accuracy Function Evolution')
plt.legend()

plt.subplot(4, 2, 3)
plt.plot(model2.history.history['precision'], label='precision')
plt.plot(model2.history.history['val_precision'], label='val_precision')
plt.title('Precision Function Evolution')
plt.legend()

plt.subplot(4, 2, 4)
plt.plot(model2.history.history['recall'], label='recall')
plt.plot(model2.history.history['val_recall'], label='val_recall')
plt.title('Recall Function Evolution')
plt.legend()
```

Out[55]:
```
<matplotlib.legend.Legend at 0x7f06705d0e80>
```




### Model-2 Prediction

In [56]:
```python
# Model prediction
test_pred_prob2 = model2.predict(
    X_test,
    workers = CPU_COUNT,
    use_multiprocessing = True
)
print('Model-2 Probability Prediction:', test_pred_prob2[:5])
```

```
313/313 [==============================] - 0s 1ms/step
Model-2 Probability Prediction: [[6.5070208e-06 3.0062916e-05 5.1112002e-06 2.0587501e-04 2.2504428e-05
  2.9659614e-02 5.9319914e-06 1.3658513e-01 3.6367169e-04 8.3311558e-01]
 [1.7805321e-02 3.9246411e-04 8.7831992e-01 7.7229226e-03 6.7182789e-03
  1.9251005e-09 8.8741772e-02 5.6972053e-12 2.9885533e-04 4.8856168e-07]
 [3.7408822e-07 9.9999964e-01 2.2913133e-13 2.8871000e-08 1.8759234e-10
  2.9554226e-13 3.4907469e-08 1.2107884e-24 4.9305013e-09 8.3100923e-20]
 [1.2702860e-04 9.9965441e-01 7.1770977e-08 1.6830963e-04 3.8801336e-06
  4.1260716e-07 3.8388313e-05 2.0702431e-12 7.4594082e-06 3.3991851e-10]
 [1.6903520e-02 3.6757127e-03 3.0506834e-01 1.7012671e-02 3.5066074e-01
  1.7979768e-05 2.9593757e-01 5.0164740e-06 1.0411404e-02 3.0708627e-04]]
```

In [57]:

```python
# Label prediction
# test_pred_prob2 = [np.argmax(test_pred_prob2[i]) for i in range(len(test_pred_prob2))]
test_pred2 = np.argmax(test_pred_prob2, axis=1)
test_label2 = [class_names[i] for i in test_pred2]
print(test_label2[:5])
```

```
['Ankle boot', 'Pullover', 'Trouser', 'Trouser', 'Coat']
```

#### Model-2 Evaluation

In [58]:

```python
# Model Evaluation
test_evaluation2 = model2.evaluate(
    X_test,
    y_test,
    workers = CPU_COUNT,
    use_multiprocessing = True
)
print('Test Loss:', round(test_evaluation2[0], 4))
print('Test Accuracy:', round(test_evaluation2[1], 4))
print('Test Precision:', round(test_evaluation2[2], 4))
print('Test Recall:', round(test_evaluation2[3], 4))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.5277 - accuracy: 0.8048 - precision: 0.8609 - recall:
0.7516
Test Loss: 0.5277
Test Accuracy: 0.8048
Test Precision: 0.8609
Test Recall: 0.7516
```

---

### CNN Model-3

In [59]:
```
# Model parameters for images and models
NUM_EPOCHS = 20
BATCH_SIZE = 64
VALIDATION_SPLIT_RATIO = 0.2
INPUT_SHAPE = (28, 28, 1)
```

In [60]:
```
# Build Model-3 architecture
model3 = Sequential()

# Convolutional Layer
model3.add(Conv2D(filters=128, kernel_size=3, strides=(2, 2), input_shape=INPUT_SHAPE, padding='same', activation='relu
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(BatchNormalization())
model3.add(Dropout(0.3))

# Convolutional Layer
model3.add(Conv2D(filters=64, kernel_size=3, strides=(2, 2), input_shape=INPUT_SHAPE, padding='same', activation='relu'
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(BatchNormalization())
model3.add(Dropout(0.3))

# Convolutional Layer
model3.add(Conv2D(filters=32, kernel_size=3, strides=(2, 2), input_shape=INPUT_SHAPE, padding='same', activation='relu'
# model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(BatchNormalization())
model3.add(Dropout(0.3))

# Flatten layer
model3.add(Flatten())
model3.add(Dense(128, activation='relu'))
model3.add(Dropout(0.3))

# Output layer
model3.add(Dense(10, activation='softmax'))
```

### Model-3 Summary

In [61]:
```
# Model Summary
model3.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 14, 14, 128)       1280

 max_pooling2d_5 (MaxPooling  (None, 7, 7, 128)        0
 2D)

 batch_normalization_3 (Batc  (None, 7, 7, 128)        512
 hNormalization)

 dropout_7 (Dropout)         (None, 7, 7, 128)         0

 conv2d_7 (Conv2D)           (None, 4, 4, 64)          73792

 max_pooling2d_6 (MaxPooling  (None, 2, 2, 64)         0
 2D)

 batch_normalization_4 (Batc  (None, 2, 2, 64)         256
 hNormalization)

 dropout_8 (Dropout)         (None, 2, 2, 64)          0

 conv2d_8 (Conv2D)           (None, 1, 1, 32)          18464

 batch_normalization_5 (Batc  (None, 1, 1, 32)         128
 hNormalization)

 dropout_9 (Dropout)         (None, 1, 1, 32)          0

 flatten_3 (Flatten)         (None, 32)                0

 dense_6 (Dense)             (None, 128)               4224

 dropout_10 (Dropout)        (None, 128)               0

 dense_7 (Dense)             (None, 10)                1290

=================================================================
Total params: 99,946
Trainable params: 99,498
Non-trainable params: 448
_________________________________________________________________
```

### Model-3 Compile

In [71]:
```
# Model Compile
model3.compile(
    loss = 'categorical_crossentropy',
    optimizer = 'adagrad',
    metrics = METRICS
)
```

### Model-3 Training

In [72]:
```
# Model Training
history3 = model3.fit(
    X_train,
    y_train,
    epochs = NUM_EPOCHS,
    batch_size = BATCH_SIZE,
    callbacks = [early_stop, lr_scheduler],
    validation_split = VALIDATION_SPLIT_RATIO,
    shuffle = True,
    workers = CPU_COUNT,
    use_multiprocessing = True
)
```

```
Epoch 1/20
```

```
2022-11-24 20:40:44.927055: E tensorflow/core/grappler/optimizers/meta_optimizer.cc:954] layout failed: INVALID_ARGUMEN
T: Size of values 0 does not match size of permutation 4 @ fanin shape insequential_3/dropout_7/dropout/SelectV2-2-Tran
sposeNHWCToNCHW-LayoutOptimizer
```

```
750/750 [==============================] - 6s 5ms/step - loss: 1.3993 - accuracy: 0.5323 - precision: 0.8249 - recall:
0.1999 - val_loss: 0.8810 - val_accuracy: 0.7319 - val_precision: 0.9093 - val_recall: 0.4663 - lr: 0.0010
Epoch 2/20
750/750 [==============================] - 4s 5ms/step - loss: 1.0778 - accuracy: 0.6331 - precision: 0.8203 - recall:
0.3963 - val_loss: 0.7430 - val_accuracy: 0.7526 - val_precision: 0.8976 - val_recall: 0.5703 - lr: 0.0010
Epoch 3/20
750/750 [==============================] - 3s 5ms/step - loss: 0.9553 - accuracy: 0.6689 - precision: 0.8230 - recall:
0.4786 - val_loss: 0.6723 - val_accuracy: 0.7648 - val_precision: 0.8877 - val_recall: 0.6302 - lr: 0.0010
Epoch 4/20
750/750 [==============================] - 3s 5ms/step - loss: 0.8839 - accuracy: 0.6921 - precision: 0.8268 - recall:
0.5303 - val_loss: 0.6298 - val_accuracy: 0.7738 - val_precision: 0.8829 - val_recall: 0.6604 - lr: 0.0010
Epoch 5/20
750/750 [==============================] - 3s 5ms/step - loss: 0.8382 - accuracy: 0.7076 - precision: 0.8296 - recall:
0.5632 - val_loss: 0.5983 - val_accuracy: 0.7821 - val_precision: 0.8808 - val_recall: 0.6837 - lr: 0.0010
Epoch 6/20
750/750 [==============================] - 4s 5ms/step - loss: 0.7994 - accuracy: 0.7200 - precision: 0.8333 - recall:
0.5842 - val_loss: 0.5765 - val_accuracy: 0.7865 - val_precision: 0.8772 - val_recall: 0.6978 - lr: 0.0010
Epoch 7/20
750/750 [==============================] - 3s 5ms/step - loss: 0.7743 - accuracy: 0.7265 - precision: 0.8326 - recall:
0.6031 - val_loss: 0.5576 - val_accuracy: 0.7922 - val_precision: 0.8790 - val_recall: 0.7080 - lr: 0.0010
Epoch 8/20
750/750 [==============================] - 3s 5ms/step - loss: 0.7547 - accuracy: 0.7332 - precision: 0.8351 - recall:
0.6175 - val_loss: 0.5435 - val_accuracy: 0.7965 - val_precision: 0.8800 - val_recall: 0.7140 - lr: 0.0010
Epoch 9/20
750/750 [==============================] - 3s 5ms/step - loss: 0.7332 - accuracy: 0.7399 - precision: 0.8376 - recall:
0.6293 - val_loss: 0.5307 - val_accuracy: 0.8002 - val_precision: 0.8783 - val_recall: 0.7227 - lr: 0.0010
Epoch 10/20
750/750 [==============================] - 3s 5ms/step - loss: 0.7230 - accuracy: 0.7454 - precision: 0.8381 - recall:
0.6385 - val_loss: 0.5198 - val_accuracy: 0.8031 - val_precision: 0.8793 - val_recall: 0.7300 - lr: 0.0010
Epoch 11/20
750/750 [==============================] - 4s 5ms/step - loss: 0.7056 - accuracy: 0.7497 - precision: 0.8389 - recall:
0.6474 - val_loss: 0.5109 - val_accuracy: 0.8074 - val_precision: 0.8804 - val_recall: 0.7348 - lr: 9.0484e-04
Epoch 12/20
750/750 [==============================] - 3s 5ms/step - loss: 0.6950 - accuracy: 0.7549 - precision: 0.8410 - recall:
0.6552 - val_loss: 0.5030 - val_accuracy: 0.8112 - val_precision: 0.8810 - val_recall: 0.7384 - lr: 8.1873e-04
Epoch 13/20
750/750 [==============================] - 3s 5ms/step - loss: 0.6876 - accuracy: 0.7561 - precision: 0.8415 - recall:
0.6607 - val_loss: 0.4968 - val_accuracy: 0.8131 - val_precision: 0.8809 - val_recall: 0.7441 - lr: 7.4082e-04
Epoch 14/20
750/750 [==============================] - 3s 5ms/step - loss: 0.6750 - accuracy: 0.7603 - precision: 0.8445 - recall:
0.6641 - val_loss: 0.4914 - val_accuracy: 0.8163 - val_precision: 0.8814 - val_recall: 0.7471 - lr: 6.7032e-04
Epoch 15/20
750/750 [==============================] - 3s 5ms/step - loss: 0.6754 - accuracy: 0.7591 - precision: 0.8406 - recall:
0.6684 - val_loss: 0.4869 - val_accuracy: 0.8190 - val_precision: 0.8820 - val_recall: 0.7492 - lr: 6.0653e-04
Epoch 16/20
```

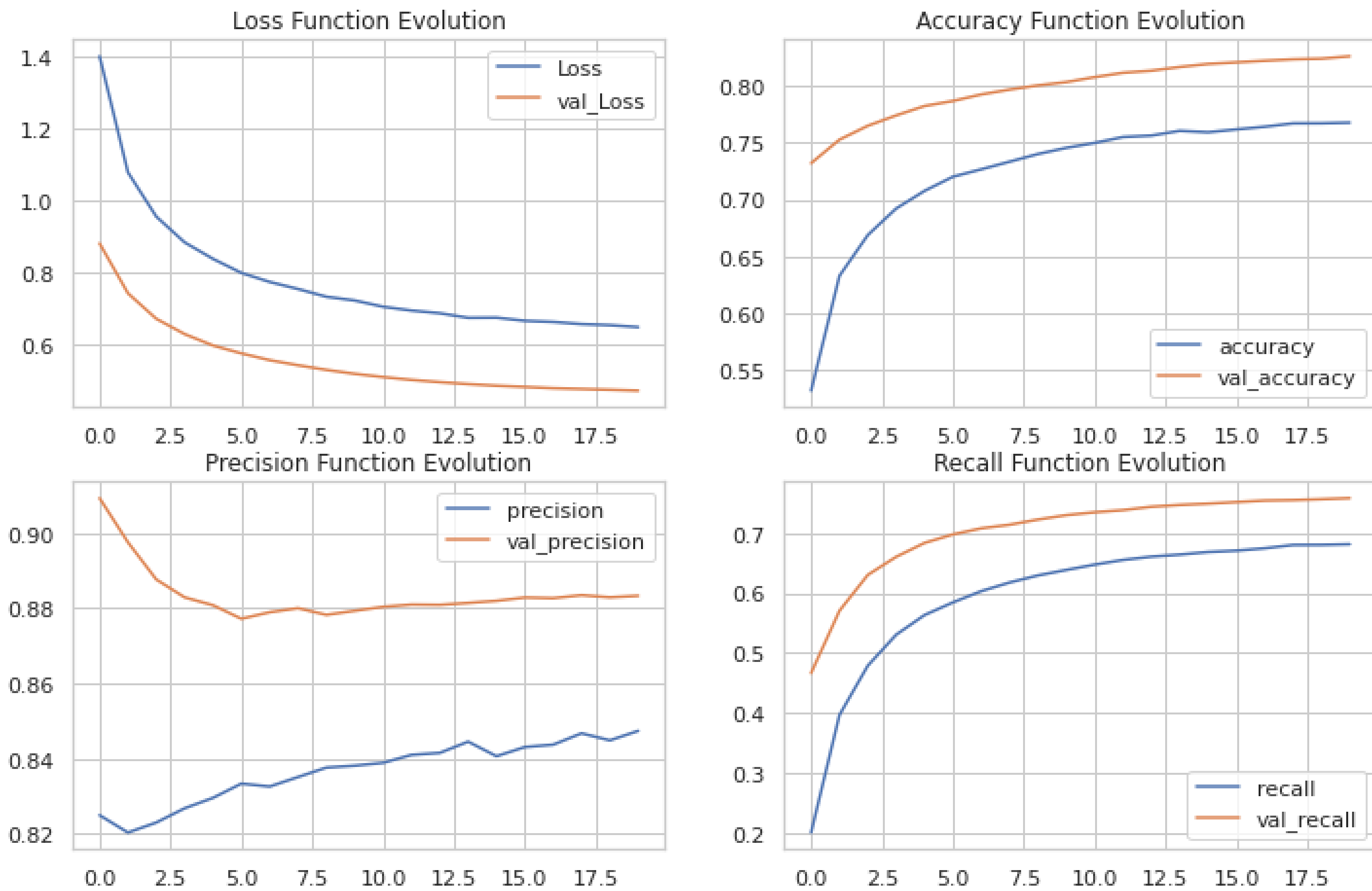
```
750/750 [==============================] - 4s 5ms/step - loss: 0.6666 - accuracy: 0.7616 - precision: 0.8431 - recall:
0.6707 - val_loss: 0.4835 - val_accuracy: 0.8204 - val_precision: 0.8829 - val_recall: 0.7518 - lr: 5.4881e-04
Epoch 17/20
750/750 [==============================] - 3s 5ms/step - loss: 0.6635 - accuracy: 0.7639 - precision: 0.8437 - recall:
0.6747 - val_loss: 0.4799 - val_accuracy: 0.8218 - val_precision: 0.8827 - val_recall: 0.7544 - lr: 4.9659e-04
Epoch 18/20
750/750 [==============================] - 3s 5ms/step - loss: 0.6575 - accuracy: 0.7669 - precision: 0.8467 - recall:
0.6802 - val_loss: 0.4778 - val_accuracy: 0.8231 - val_precision: 0.8835 - val_recall: 0.7552 - lr: 4.4933e-04
Epoch 19/20
750/750 [==============================] - 4s 5ms/step - loss: 0.6546 - accuracy: 0.7669 - precision: 0.8449 - recall:
0.6804 - val_loss: 0.4757 - val_accuracy: 0.8237 - val_precision: 0.8829 - val_recall: 0.7567 - lr: 4.0657e-04
Epoch 20/20
750/750 [==============================] - 3s 5ms/step - loss: 0.6492 - accuracy: 0.7675 - precision: 0.8474 - recall:
0.6815 - val_loss: 0.4732 - val_accuracy: 0.8257 - val_precision: 0.8833 - val_recall: 0.7584 - lr: 3.6788e-04
```

### Model-3 History Plot

In [73]:

```python
# Model History Plot
plt.figure(figsize=(12, 16))

plt.subplot(4, 2, 1)
plt.plot(model3.history.history['loss'], label='Loss')
plt.plot(model3.history.history['val_loss'], label='val_Loss')
plt.title('Loss Function Evolution')
plt.legend()

plt.subplot(4, 2, 2)
plt.plot(model3.history.history['accuracy'], label='accuracy')
plt.plot(model3.history.history['val_accuracy'], label='val_accuracy')
plt.title('Accuracy Function Evolution')
plt.legend()

plt.subplot(4, 2, 3)
plt.plot(model3.history.history['precision'], label='precision')
plt.plot(model3.history.history['val_precision'], label='val_precision')
plt.title('Precision Function Evolution')
plt.legend()

plt.subplot(4, 2, 4)
plt.plot(model3.history.history['recall'], label='recall')
plt.plot(model3.history.history['val_recall'], label='val_recall')
plt.title('Recall Function Evolution')
plt.legend()
```

Out[73]: <matplotlib.legend.Legend at 0x7f0618217d00>




### Model-3 Prediction

In [74]:
```
# Model prediction
test_pred_prob3 = model3.predict(
    X_test,
    workers = CPU_COUNT,
    use_multiprocessing = True
)
print('Model-3 Probability Prediction:', test_pred_prob3[:5])
```

```
313/313 [==============================] - 0s 1ms/step
Model-3 Probability Prediction: [[8.1419694e-04 6.7904638e-04 1.4504708e-03 1.0005566e-03 1.1016640e-03
  1.9495485e-02 1.1635824e-03 9.3938313e-02 5.6108371e-03 8.7474585e-01]
 [6.0489285e-03 7.0166524e-04 8.3101064e-01 2.7665107e-03 4.4970285e-02
  1.9089123e-03 1.0525276e-01 1.3450970e-03 3.4806242e-03 2.5145530e-03]
 [1.2786042e-05 9.9989045e-01 2.0027760e-06 6.7911627e-05 5.8929568e-06
  1.1419879e-06 4.8496890e-06 2.7467314e-07 9.2421260e-06 5.4080115e-06]
 [3.6944781e-05 9.9963582e-01 1.5424586e-05 1.5442182e-04 4.8551843e-05
  1.2977289e-05 1.7148559e-05 2.3860910e-06 3.3771939e-05 4.2567775e-05]
 [1.0240327e-01 1.4658290e-03 2.4612652e-01 1.2644647e-02 3.3891298e-02
  1.7499846e-03 5.9649611e-01 1.0056755e-03 1.4448620e-03 2.7717999e-03]]
```

In [75]:

```
# Label prediction
# test_pred_prob3 = [np.argmax(test_pred_prob3[i]) for i in range(len(test_pred_prob3))]
test_pred3 = np.argmax(test_pred_prob3, axis=1)
test_label3 = [class_names[i] for i in test_pred3]
print(test_label3[:5])
```

```
['Ankle boot', 'Pullover', 'Trouser', 'Trouser', 'Shirt']
```

### Model-3 Evaluation

In [76]:

```
# Model Evaluation
test_evaluation3 = model3.evaluate(
    X_test,
    y_test,
    workers = CPU_COUNT,
    use_multiprocessing = True
)
print('Test Loss:', round(test_evaluation3[0], 4))
print('Test Accuracy:', round(test_evaluation3[1], 4))
print('Test Precision:', round(test_evaluation3[2], 4))
print('Test Recall:', round(test_evaluation3[3], 4))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.4954 - accuracy: 0.8162 - precision: 0.8804 - recall:
0.7496
Test Loss: 0.4954
Test Accuracy: 0.8162
Test Precision: 0.8804
Test Recall: 0.7496
```

---

## CNN Combined Model

#### Combined Model Average Probability Prediction

In [77]:
```
test_pred_prob_ensemble = np.mean(np.array([test_pred_prob1, test_pred_prob2, test_pred_prob3]), axis=0)
test_pred_prob_ensemble[:5]
```

Out[77]:
```
array([[2.73598009e-04, 2.36373278e-04, 4.85196681e-04, 4.02148959e-04,
        3.74722877e-04, 1.63855162e-02, 3.89839173e-04, 7.68496692e-02,
        1.99166290e-03, 9.02611256e-01],
       [7.95177091e-03, 3.64709791e-04, 9.03057814e-01, 3.49650788e-03,
        1.72471274e-02, 6.36304787e-04, 6.46991655e-02, 4.48365667e-04,
        1.25982659e-03, 8.38347769e-04],
       [4.38671077e-06, 9.99963343e-01, 6.67593383e-07, 2.26468819e-05,
        1.96438145e-06, 3.80662755e-07, 1.62819890e-06, 9.15577161e-08,
        3.08235212e-06, 1.80267045e-06],
       [5.46577976e-05, 9.99763429e-01, 5.16545515e-06, 1.07577624e-04,
        1.74773322e-05, 4.46329841e-06, 1.85122935e-05, 7.95364315e-07,
        1.37437819e-05, 1.41893715e-05],
       [2.07596347e-01, 1.71385927e-03, 1.84366807e-01, 9.88863595e-03,
        1.29255906e-01, 5.89339237e-04, 4.61273670e-01, 3.36900732e-04,
        3.95208970e-03, 1.02647475e-03]], dtype=float32)
```

In [78]:
```
# Label prediction
# test_pred_prob3 = [np.argmax(test_pred_prob3[i]) for i in range(len(test_pred_prob3))]
test_pred_ensemble = np.argmax(test_pred_prob_ensemble, axis=1)
test_label_ensemble = [class_names[i] for i in test_pred_ensemble]
print(test_label_ensemble[:5])
```

```
['Ankle boot', 'Pullover', 'Trouser', 'Trouser', 'Shirt']
```

#### Combined Model Evaluation

In [79]:
```
test_loss_ensemble = np.mean([test_evaluation1[0], test_evaluation2[0], test_evaluation3[0]])
print('Loss:', round(test_loss_ensemble, 4))

test_accuracy_ensemble = np.mean([test_evaluation1[1], test_evaluation2[1], test_evaluation3[1]])
print('Accuracy:', round(test_accuracy_ensemble, 4))

test_precision_ensemble = np.mean([test_evaluation1[2], test_evaluation2[2], test_evaluation3[2]])
print('Precision:', round(test_precision_ensemble, 4))

test_recall_ensemble = np.mean([test_evaluation1[3], test_evaluation2[3], test_evaluation3[3]])
print('Recall:', round(test_recall_ensemble, 4))
```

```
Loss: 0.4441
Accuracy: 0.8409
Precision: 0.8835
Recall: 0.7992
```

In [ ]:

In [ ]:

# Analysis of the results

## Model-1

Model-1 consists of a Convolution layer with 32 output filters and kernel size of 3 specifying the same value for all spatial dimensions. Also, 'valid' padding (no padding) is used along with the 'Relu' activation at the Convolution layer. A stride of (2,2) is used to specify the strides of the convolution along the height and width. This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. It is followed by Max Pooling layer with a pool size of 2*2. The Convolution layers kernel is initialized by the default 'glorot_uniform' kernel initializer

The Convolution and Max Pooling layer is followed by the Flatten layer which converts the multi-dimensions into a single dimension vector which is followed by the Dense layer with 256 neuron units, and finally the output layer with 10 neuron units i.e. number of classes in the dataset. The Relu activation is applied in the Convolution layer and the Dense layer in the Flatten layer, and the 'Softmax' activation is applied in the Output layer to obtain the number of class probability distribution for the model to predict the output class of the images.

Model-1 is trained on the training dataset using the batch size of 32 and with the 'ADAM' optimizer. We have used the Early Stop callback to monitor the validation loss at the patient level of 5, and a Learning Rate Schdeuler callback to decrease the learning rate value exponentially after 10 epochs during the training of the model. The model is able to converge after the 11th epoch from total of 20 epochs and produced the below results:

- Loss: 0.3093
- Accuracy: 0.9017
- Precision: 0.909
- Recall: 0.8965

# Model-2

Model-2 consists of two Convolution layer with 64 and 32 output filters to each layer and kernel size of 2 specifying the same value for all spatial dimensions. Also, 'valid' padding (no padding) is used along with the 'Relu' activation at each Convolution layer. A stride of (2,2) is used to specify the strides of the convolution along the height and width. Both layers create a seperate convolution kernel that is convolved with the layer input to produce a tensor of outputs. It is followed by Max Pooling layer with a pool size of 2*2 and then followed by the Dropout layer with a decent drop rate of 0.2. Both Convolution layer kernels are initialized by the default 'glorot_uniform' kernel initializer.

The dual layer of Convolution, Max Pooling and Dropout layers are followed by the Flatten layer which converts the multi-dimensions into a single dimension vector which is followed by the Dense layer with 128 neuron units, and finally the output layer with 10 neuron units i.e. number of classes in the dataset. The 'Relu' activation is applied in both Convolution layers and the Dense layer in the Flatten layer, and the 'Softmax' activation is applied in the Output layer to obtain the number of class probability distribution for the model to predict the output class of the images.

Model-2 is trained on the training dataset using the batch size of 16 and with the 'SGD' optimizer. We have used the Early Stop callback to monitor the validation loss at the patient level of 5, and a Learning Rate Schdeuler callback to decrease the learning rate value exponentially after 10 epochs during the training of the model. The model is able to converge after the 9th epoch from total of 20 epochs and produced the below results:

- Loss: 0.5277
- Accuracy: 0.8048
- Precision: 0.8609
- Recall: 0.7516

# Model-3

Model-3 consists of three Convolution layer with 128, 64 and 32 output filters to each layer and kernel size of 3 specifying the same value for all spatial dimensions. Also, 'same' padding (no padding) is used along with the Relu activation at each Convolution layer. A stride of (2,2) is used to specify the strides of the convolution along the height and width. Both layers create a seperate convolution kernel that is convolved with the layer input to produce a tensor of outputs. It is followed by Max Pooling layer with a pool size of 2*2 and then followed by the Dropout layer with a decent drop rate of 0.3. Both Convolution layer kernels are initialized by the default 'glorot_uniform' kernel initializer.

Three layer of Convolution, Max Pooling and Dropout layers are followed by the Flatten layer which converts the multi-dimensions into a single dimension vector which is followed by the Dense layer with 128 neuron units, and finally the output layer with 10 neuron units i.e. number of classes in the dataset. The 'Relu' activation is applied in both Convolution layers and the Dense layer in the Flatten layer, and the 'Softmax' activation is applied in the Output layer to obtain the number of class probability distribution for the model to predict the output class of the images.

Model-3 is trained on the training dataset using the batch size of 64 and with the 'ADAGRAD' optimizer. We have used the Early Stop callback to monitor the validation loss at the patient level of 5, and a Learning Rate Schdeuler callback to decrease the learning rate value exponentially after 10 epochs during the training of the model. The model is able to converge after the 20th epoch and produced the below results:

- Loss: 0.4954
- Accuracy: 0.8162
- Precision: 0.8804
- Recall: 0.7496

### Combined Model

The combined model is created by averaging the predicted probability values from Model-1, Model-2 and Model-3 Deep Learning Image classification model. The combined model is evaulated on the similar evaluatio metrics based on the average outcomes from the 3 different models. It is an ensemble of the three models, so the results obtained would be equal weighted prediction from all models. As all the models are given equal weightage, so the results obtained from these models would have a direct impact on the results.

The combined model is able to provide the below results:

- Loss: 0.4441
- Accuracy: 0.8409
- Precision: 0.8835
- Recall: 0.7992

## Conclusion

After careful analysis of all model performances on the testing dataset, we found that **Model-1** is able to perform better than the

Model-2, Model-3 and the combined model with the highest accuracy score of **0.9017** and minimum loss of **0.3093**. Also, other evaluation metrics are also best for the Model-1 with Precision of **0.909** and Recall of **8965**.

The combined model is also able to perform well on the testing dataset and better than the Model-2 and Model-3, but there is a huge difference in the metrics from Model-1.

So, we should consider the Model-1 for the Fashion MNIST dataset images classification for the prediction purpose on the unseen dataset. Also, as the Fashion MNIST dataset is not a very complex dataset, the Model-1 is able to outperform other models with the simplest Neural Network architecture, which would result in faster inference on the unseen image dataset.

# END