Create CNN Model and Optimize it using Keras Tuner

```
!pip install keras-tuner
```

```
Collecting keras-tuner
  Downloading https://files.pythonhosted.org/packages/20/ec/1ef246787174b1e2bb591c95f29d
     |████████████████████████████████| 71kB 3.5MB/s
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages (from ke
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from ker
Requirement already satisfied: tabulate in /usr/local/lib/python3.7/dist-packages (from
Collecting terminaltables
  Downloading https://files.pythonhosted.org/packages/9b/c4/4a21174f32f8a7e1104798c445da
Collecting colorama
  Downloading https://files.pythonhosted.org/packages/44/98/5b86278fbbf250d239ae0ecb724f
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from kera
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from ker
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (f
Building wheels for collected packages: keras-tuner, terminaltables
  Building wheel for keras-tuner (setup.py) ... done
  Created wheel for keras-tuner: filename=keras_tuner-1.0.2-cp37-none-any.whl size=78938
  Stored in directory: /root/.cache/pip/wheels/bb/a1/8a/7c3de0efb3707a1701b36ebbfdbc4e67
  Building wheel for terminaltables (setup.py) ... done
  Created wheel for terminaltables: filename=terminaltables-3.1.0-cp37-none-any.whl size
  Stored in directory: /root/.cache/pip/wheels/30/6b/50/6c75775b681fb36cdfac7f19799888ef
Successfully built keras-tuner terminaltables
Installing collected packages: terminaltables, colorama, keras-tuner
Successfully installed colorama-0.4.4 keras-tuner-1.0.2 terminaltables-3.1.0
```

The Keras Tuner is a library that helps you pick the optimal set of hyperparameters for your TensorFlow program. The process of selecting the right set of hyperparameters for your machine learning (ML) application is called hyperparameter tuning or hypertuning.

Double-click (or enter) to edit

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
```

Tensorflow is a symbolic math library based on dataflow and differentiable programming which can be used for training and inference of deep neural networks.

NumPy is used for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

```
print(tf.__version__)
```

```
2.4.1
```

showing tensorflow version

```
fashion_mnist=keras.datasets.fashion_mnist
```

fashion_mnist is a datase which is in package dataset of keras library. It consists of training set of 60,000 examples and test set of 10,000 examples. Each example is a 28*28 grayscale image, associated wit a label from 10 classes.

```
(train_images,train_labels),(test_images,test_labels)=fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-
32768/29515 [================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-
26427392/26421880 [==============================] - 1s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-l
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-i
4423680/4422102 [==============================] - 0s 0us/step
```

downloading dataset

```
train_images=train_images/255.0
test_images=test_images/255.0
```

The data must be preprocessed before training the network. If you inspect the first image in the training set, you will see that the pixel values fall in the range of 0 to 255

Scale these values to a range of 0 to 1 before feeding them to the neural network model. To do so, divide the values by 255. It's important that the training set and the testing set be preprocessed in the same way

```
train_images[0].shape
```

```
(28, 28)
```

Images in the training set are represented as 28 x 28 pixels

```
train_images=train_images.reshape(len(train_images),28,28,1)
test_images=test_images.reshape(len(test_images),28,28,1)
```

The 28 x 28 pixels images are 1 dimensional images, so we reshape and point to the program about the image informtaion.

```
def build_model(hp):
  model = keras.Sequential([
    keras.layers.Conv2D(
        filters=hp.Int('conv_1_filter', min_value=32, max_value=128, step=16),
        kernel_size=hp.Choice('conv_1_kernel', values = [3,5]),
        activation='relu',
        input_shape=(28,28,1)
    ),
    keras.layers.Conv2D(
        filters=hp.Int('conv_2_filter', min_value=32, max_value=64, step=16),
        kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
        activation='relu'
    ),
    keras.layers.Flatten(),
    keras.layers.Dense(
        units=hp.Int('dense_1_units', min_value=32, max_value=128, step=16),
        activation='relu'
    ),
    keras.layers.Dense(10, activation='softmax')
  ])

  model.compile(optimizer=keras.optimizers.Adam(hp.Choice('learning_rate', values=[1e-2, 1e-3
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

  return model
```

We need to define a baseline convolutional neural network model for the problem.The model has two main aspects: the feature extraction front end comprised of convolutional and pooling layers, and the classifier backend that will make a prediction. For the convolutional front-end, we can start with a single convolutional layer with a small filter size and a modest number of filters followed by a max pooling layer. The filter maps can then be flattened to provide features to the classifier. Given

that the problem is a multi-class classification, we know that we will require an output layer with 10 nodes in order to predict the probability distribution of an image belonging to each of the 10 classes. This will also require the use of a softmax activation function. Between the feature extractor and the output layer, we can add a dense layer to interpret the features, in this case with 10 nodes. All layers will use the ReLU activation function, a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero

```
from kerastuner import RandomSearch
from kerastuner.engine.hyperparameters import HyperParameters
```

Importing random search algoritm that does not implement optimization of gradient.

Importing HyperParameters which are variables that are used to control the learning process of the model.

```
tuner_search=RandomSearch(build_model,
                          objective='val_accuracy',
                          max_trials=5,directory='output',project_name="Mnist Fashion")
```

RandomSearch function is used to optimize the hyperParameters for the CNN model.

val_accuracy is used as a metric to compare models.

max_trails attribute determines the number of hyperParameter combinations the tuner must test.

directory determines ehere the tuner should store the debugging data.

```
tuner_search.search(train_images,train_labels,epochs=3,validation_split=0.1)
```

```
Trial 5 Complete [00h 00m 45s]
val_accuracy: 0.878166675567627

Best val_accuracy So Far: 0.9026666879653931
Total elapsed time: 00h 03m 54s
INFO:tensorflow:Oracle triggered exit
```

We train the model to 3 epochs and get an accuracy of 90.91%. The validation accuracy is 86.26%.

At the moment the model has an accuracy of ~90% on the training set and ~86% on the validation set. This means that we can expect the model to perform with ~86% accuracy on new data

Double-click (or enter) to edit

```
model=tuner_search.get_best_models(num_models=1)[0]
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 24, 24, 112)       2912
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 20, 20, 48)        134448
_________________________________________________________________
flatten (Flatten)            (None, 19200)             0
_________________________________________________________________
dense (Dense)                (None, 32)                614432
_________________________________________________________________
dense_1 (Dense)              (None, 10)                330
=================================================================
Total params: 752,122
Trainable params: 752,122
Non-trainable params: 0
_________________________________________________________________
```

The number of parameters is 752122 because the total hidden unit is summed of input weights and one weight of connection with bias. This means that every hidden unit gives you ceratin parameters as shown above

For Convolution layers, the first layer shape is 24, 24, 112 and second layer is 20, 20, 48, which means the input size is shrinking. The number of output channels for each Conv2D layer is controlled by the first argument (e.g. 64). Typically, as the width and height shrink, you can afford (computationally) to add more output channels in each Conv2D layer.

For Flatten layer, the input is 20 * 20 * 48 so the shape is 19200

For 1st Dense layer, the parameter is 2478160 because the output of flatten layer us 19200 and when multiplied by 32 (input of dense layer)

For 2nd Dense layer, the parameter is 330 because 80 multiplied by 10 is 8320 and there are 10 nodes so adding 10 to 320 is 330

```
model.fit(train_images, train_labels, epochs=10, validation_split=0.1, initial_epoch=3)
```

```
Epoch 4/10
1688/1688 [==============================] - 15s 9ms/step - loss: 0.2034 - accuracy: 0.9
Epoch 5/10
1688/1688 [==============================] - 15s 9ms/step - loss: 0.1666 - accuracy: 0.9
Epoch 6/10
1688/1688 [==============================] - 15s 9ms/step - loss: 0.1365 - accuracy: 0.9
Epoch 7/10
1688/1688 [==============================] - 15s 9ms/step - loss: 0.1155 - accuracy: 0.9
Epoch 8/10
1688/1688 [==============================] - 15s 9ms/step - loss: 0.0988 - accuracy: 0.9
```

```
Epoch 9/10
1688/1688 [==============================] - 15s 9ms/step - loss: 0.0801 - accuracy: 0.9
Epoch 10/10
1688/1688 [==============================] - 15s 9ms/step - loss: 0.0685 - accuracy: 0.9
<tensorflow.python.keras.callbacks.History at 0x7fd2a018b450>
```

```
In this step we continue to train the model fr
from epoch 3
```

In this step we continue to train the model from where we left earlier, i.e. from epoch 3

+ Code + Text