MODEL BUILDING

Date	13 NOVEMBER 2022
Team ID	PNT2022TMID48373
Project Name	AI-Powered Nutrition Analyzer for fitness enthusiasts

Steps to Build a Deep Learning Model

1. Defining model architecture

This is a very crucial step in our deep learning model building process. We have to define how our model will look and that requires

- **Importing the libraries**

IMPORTING NECESSARY LIBRARIES

```
[3]  import numpy as np#used for numerical analysis
     import tensorflow #open source used for both ML and DL for computation
     from tensorflow.keras.models import Sequential #it is a plain stack of layers
     from tensorflow.keras import layers #A layer consists of a tensor-in tensor-out computation function
     #Dense layer is the regular deeply connected neural network layer
     from tensorflow.keras.layers import Dense,Flatten
     #Faltten-used fot flattening the input or change the dimension
     from tensorflow.keras.layers import Conv2D,MaxPooling2D,Dropout #Convolutional layer
     #MaxPooling2D-for downsampling the image
     from keras.preprocessing.image import ImageDataGenerator
```

- **Adding CNN (Convolution Neural Network) Layers**

Keras has 2 ways to define a neural network:

- Sequential
- Function API

The Sequential class is used to define a linear initialization of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add() method

- **Adding Dense layers**

We will be adding three layers for CNN

- Convolution layer
- Pooling layer
- Flattening layer

- **Initializing the model**

```
# Initializing the CNN
classifier = Sequential()
# First convolution layer and pooling
classifier.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
# Second convolution layer and pooling
classifier.add(Conv2D(32, (3, 3), activation='relu'))
# input_shape is going to be the pooled feature maps from the previous convolution layer
classifier.add(MaxPooling2D(pool_size=(2, 2)))
# Flattening the layers
classifier.add(Flatten())
# Adding a fully connected layer
```

classifier.add(Dense(units=128, activation='relu'))
classifier.add(Dense(units=5, activation='softmax')) # softmax for more than 2

```
[13] # Initializing the CNN
     classifier = Sequential()

     # First convolution layer and pooling
     classifier.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), activation='relu'))
     classifier.add(MaxPooling2D(pool_size=(2, 2)))

     # Second convolution layer and pooling
     classifier.add(Conv2D(32, (3, 3), activation='relu'))

     # input_shape is going to be the pooled feature maps from the previous convolution layer
     classifier.add(MaxPooling2D(pool_size=(2, 2)))

     # Flattening the layers
     classifier.add(Flatten())

     # Adding a fully connected layer
     classifier.add(Dense(units=128, activation='relu'))
     classifier.add(Dense(units=5, activation='softmax')) # softmax for more than 2
```

classifier.summary()#summary of our model

```
[14] Model: "sequential_2"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #
     =================================================================
      conv2d_2 (Conv2D)           (None, 62, 62, 32)        896

      max_pooling2d_2 (MaxPooling  (None, 31, 31, 32)       0
      2D)

      conv2d_3 (Conv2D)           (None, 29, 29, 32)        9248

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

      flatten (Flatten)           (None, 6272)              0

      dense (Dense)               (None, 128)               802944

      dense_1 (Dense)             (None, 5)                 645

     =================================================================
     Total params: 813,733
     Trainable params: 813,733
     Non-trainable params: 0
     _________________________________________________________________
```

2. Configure the learning process

With both the training data defined and model defined, it's time configure the learning process. This is accomplished with a call to the compile() method of the Sequential model class. Compilation requires 3 arguments: an optimizer, a loss function, and a list of metrics.
In our example, set up as a multi-class classification problem, we will use the Adam optimizer, the categorical cross entropy loss function, and include solely the accuracy metric.

```python
# Compiling the CNN
# categorical_crossentropy for more than 2
classifier.compile(optimizer='adam', loss='sparse_categorical_crossentr
opy', metrics=['accuracy'])
```

3. Train The Model

At this point we have training data and a fully configured neural network to train with said data. All that is left is to pass the data to the model for the training process to commence, a process which is completed by iterating on the training data. Training begins by calling the fit() method.

```
>>>
classifier.fit_generator(
 generator=x_train,steps_per_epoch = len(x_train),
epochs=30,validation_data=x_test,validation_steps=len(x_test))
```

```
    epochs=30, validation_data=x_test,validation_steps = len(x_test))# No of images in test set

Epoch 1/30
  1/678 [..............................] - ETA: 47s - loss: 0.6063 - accuracy: 0.8000/usr/local/lib/python3.7/dist-packa
  This is separate from the ipykernel package so we can avoid doing imports until
678/678 [==============================] - 36s 52ms/step - loss: 0.5161 - accuracy: 0.8120 - val_loss: 0.4547 - val_accu
Epoch 2/30
678/678 [==============================] - 33s 49ms/step - loss: 0.4484 - accuracy: 0.8279 - val_loss: 0.4405 - val_accu
Epoch 3/30
678/678 [==============================] - 34s 50ms/step - loss: 0.4261 - accuracy: 0.8347 - val_loss: 0.3995 - val_accu
Epoch 4/30
678/678 [==============================] - 32s 47ms/step - loss: 0.4153 - accuracy: 0.8436 - val_loss: 0.4683 - val_accu
Epoch 5/30
678/678 [==============================] - 36s 54ms/step - loss: 0.3928 - accuracy: 0.8539 - val_loss: 0.3956 - val_accu
Epoch 6/30
678/678 [==============================] - 37s 55ms/step - loss: 0.3664 - accuracy: 0.8622 - val_loss: 0.3828 - val_accu
Epoch 7/30
678/678 [==============================] - 44s 65ms/step - loss: 0.3563 - accuracy: 0.8645 - val_loss: 0.4204 - val_accu
Epoch 8/30
678/678 [==============================] - 46s 67ms/step - loss: 0.3568 - accuracy: 0.8568 - val_loss: 0.3587 - val_accu
Epoch 9/30
678/678 [==============================] - 34s 51ms/step - loss: 0.3356 - accuracy: 0.8722 - val_loss: 0.5090 - val_accu
Epoch 10/30
678/678 [==============================] - 36s 53ms/step - loss: 0.3240 - accuracy: 0.8787 - val_loss: 0.3447 - val_accu
```

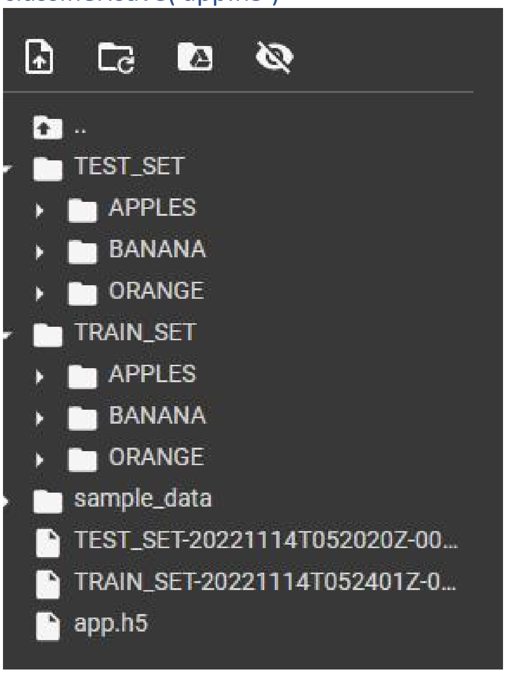
```
Epoch 11/30
678/678 [==============================] - 33s 49ms/step - loss: 0.2949 - accuracy: 0.8867 - val_loss: 0.6004 - val_acc
Epoch 12/30
678/678 [==============================] - 33s 48ms/step - loss: 0.2796 - accuracy: 0.8899 - val_loss: 0.3544 - val_acc
Epoch 13/30
678/678 [==============================] - 35s 52ms/step - loss: 0.2675 - accuracy: 0.8940 - val_loss: 0.3448 - val_acc
Epoch 14/30
678/678 [==============================] - 32s 47ms/step - loss: 0.2561 - accuracy: 0.9044 - val_loss: 0.3706 - val_acc
Epoch 15/30
678/678 [==============================] - 33s 48ms/step - loss: 0.2335 - accuracy: 0.9138 - val_loss: 0.3851 - val_acc
Epoch 16/30
678/678 [==============================] - 31s 46ms/step - loss: 0.2233 - accuracy: 0.9123 - val_loss: 0.3951 - val_acc
Epoch 17/30
678/678 [==============================] - 33s 49ms/step - loss: 0.1957 - accuracy: 0.9271 - val_loss: 0.3559 - val_acc
Epoch 18/30
678/678 [==============================] - 32s 48ms/step - loss: 0.1936 - accuracy: 0.9247 - val_loss: 0.3762 - val_acc
Epoch 19/30
678/678 [==============================] - 34s 50ms/step - loss: 0.1726 - accuracy: 0.9298 - val_loss: 0.5053 - val_acc
Epoch 20/30
678/678 [==============================] - 33s 48ms/step - loss: 0.1775 - accuracy: 0.9348 - val_loss: 0.3801 - val_acc
```

```
Epoch 21/30
678/678 [==============================] - 33s 48ms/step - loss: 0.1677 - accuracy: 0.9398 - val_loss: 0.3862 - val_acc
Epoch 22/30
678/678 [==============================] - 32s 46ms/step - loss: 0.1497 - accuracy: 0.9448 - val_loss: 0.4547 - val_acc
Epoch 23/30
678/678 [==============================] - 33s 49ms/step - loss: 0.1434 - accuracy: 0.9489 - val_loss: 0.4884 - val_acc
Epoch 24/30
678/678 [==============================] - 33s 48ms/step - loss: 0.1327 - accuracy: 0.9483 - val_loss: 0.3746 - val_acc
Epoch 25/30
678/678 [==============================] - 39s 58ms/step - loss: 0.1233 - accuracy: 0.9525 - val_loss: 0.5094 - val_acc
Epoch 26/30
678/678 [==============================] - 32s 47ms/step - loss: 0.1125 - accuracy: 0.9604 - val_loss: 0.4257 - val_acc
Epoch 27/30
678/678 [==============================] - 32s 47ms/step - loss: 0.1465 - accuracy: 0.9486 - val_loss: 0.5247 - val_acc
Epoch 28/30
678/678 [==============================] - 33s 48ms/step - loss: 0.1174 - accuracy: 0.9575 - val_loss: 0.4778 - val_acc
Epoch 29/30
678/678 [==============================] - 31s 46ms/step - loss: 0.0719 - accuracy: 0.9728 - val_loss: 0.5071 - val_acc
Epoch 30/30
678/678 [==============================] - 33s 49ms/step - loss: 0.1144 - accuracy: 0.9578 - val_loss: 0.4887 - val_acc
<keras.callbacks.History at 0x7f6e6b4fba50>
```

4. <u>Save the Model</u>

Your model is to be saved for the future purpose. This saved model ac also be integrated with an android application or web application in order to predict something

```
>># Save the model
classifier.save('app.h5')
```



5. <u>Predictions</u>

The last and final step is to make use of the Saved model to make predictions. We use load model class to load the model. We use imread() class from opencv library to read an image and give it to the model to predict the result. Before giving the original image to predict the class, we have to pre-process that image and apply predictions to get accurate result.

```
>>>
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
model = load_model("app.h5") #loading the model for testing
img=tensorflow.keras.utils.load_img("/content/TEST_SET/ORANGE/n07749192_1081.jpg",grayscale=False,target_size= (64,64))#loading of the image
x = image.img_to_array(img)#image to array
```

X

```
x
array([[[255., 255., 255.],
        [255., 255., 255.],
        [255., 255., 255.],
        ...,
        [255., 255., 255.],
        [255., 255., 255.],
        [255., 255., 255.]],

       [[255., 255., 255.],
        [255., 255., 255.],
        [255., 255., 255.],
        ...,
        [255., 255., 255.],
        [255., 255., 255.],
        [255., 255., 255.]],

       [[255., 255., 255.],
        [255., 255., 255.],
        [255., 255., 255.],
        ...,
```

```
[77] x = np.expand_dims(x,axis = 0)#changing the shape
     pred = model.predict(x)#predicting the classes
     classes=np.argmax(pred,axis=1)
     classes

     1/1 [==============================] - 0s 29ms/step
     array([2])
```

```
[78] index=['APPLES', 'BANANA', 'ORANGE']
     result=str(index[classes[0]])
     result

     'ORANGE'
```