

PROJECT DEVELOPMENT PHASE

SPRINT-2

Topic: AI-powered Nutrition Analyzer for Fitness Enthusiasts

Team-ID: PNT2022TMID15800

GitHub ID: IBM-Project-24498-1659943754

Mentor Name: Mrs S. Selvi

Team Members: P Jai Siva Ranjani (Team Leader) – 111719104118

Ishwarya S – 111719104062

Lavanya S – 111719104084

Manasa G C – 111719104090

MODEL BUILDING

Here we are going to build our Convolutional Neural Networking which contains an input layer along with the convolution, max-pooling, and finally an output layer.

This is an important step because the model is the main thing needed for prediction.

- **Importing The Model Building Libraries**

We have imported all the necessary libraries needed for model building.

Importing The Model Building Libraries

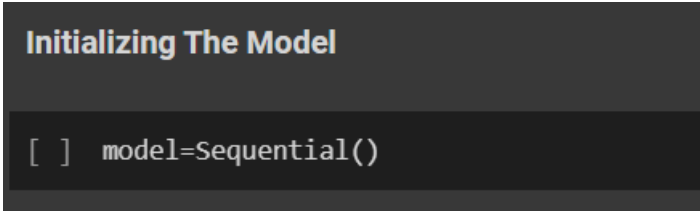
```
[ ] import numpy as np
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout
from keras.preprocessing.image import ImageDataGenerator
```

- **Initializing The Model**

Keras has 2 ways to define a neural network:

- Sequential
- Function API

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



Initializing The Model

```
[ ] model=Sequential()
```

- **Adding CNN Layers**

- As the input image contains three channels, we are specifying the input shape as (64,64,3).
- We are adding a two-convolution layer with an activation function as “relu” and with a small filter size (3,3) and the number of filters (32) followed by a max-pooling layer.
- Max pool layer is used to downsample the input (Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter)
- Flatten layer flattens the input. Does not affect the batch size.

Adding CNN Layers

First Convolution Layer and pooling

```
[ ] model.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), activation='relu'))  
    model.add(MaxPooling2D(pool_size=(2, 2)))
```

Second Convolution Layer and pooling

```
▶ model.add(Conv2D(32, (3, 3), activation='relu'))  
    model.add(MaxPooling2D(pool_size=(2, 2)))
```

Flatten layer

```
[ ] model.add(Flatten())
```

- **Adding Dense Layers**

- A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer.
- The number of neurons in the Dense layer is the same as the number of classes in the training set. The neurons in the last Dense layer, use softmax activation to convert their outputs into respective probabilities.
- Understanding the model is a very important phase to properly use it for training and prediction purposes. Keras provides a simple method, and a summary to get the full information about the model and its layers.

Adding Dense Layers

```
[ ] model.add(Dense(units=128, activation='relu'))  
    model.add(Dense(units=5, activation='softmax'))
```

```
▶ model.summary()
```

```
↳ Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 62, 62, 32)	896
max_pooling2d (MaxPooling2D)	(None, 31, 31, 32)	0
conv2d_1 (Conv2D)	(None, 29, 29, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 32)	0
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		

- **Configure The Learning Process**

- The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires a loss function during the model compilation process.
- Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer.
- Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process.

Configure The Learning Process

```
[ ] model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

• Train The Model

Now, let us train our model with our image dataset. The model is trained for 15 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 15 epochs and probably there is further scope to improve the model.

`fit_generator` functions used to train a deep-learning neural network

Arguments:

- `steps_per_epoch`: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of `steps_per_epoch` as the total number of samples in your dataset divided by the batch size.
- `Epochs`: an integer and number of epochs we want to train our model for.
- `validation_data` can be either:
 - an inputs and targets list
 - a generator
 - inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.
- `validation_steps`: only if the `validation_data` is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

Train The Model

```
model.fit_generator(
    generator=x_train, steps_per_epoch = len(x_train),
    epochs=15, validation_data=x_test, validation_steps = len(x_test))
```

`/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarning: 'Model.fit_generator' is deprecated and will be removed in a future version. Please use 'Model.fit', which supports generators. This is separate from the ipykernel package so we can avoid doing imports until`

Epoch	Time	Loss	Accuracy	Val Loss	Val Accuracy
1/15	1032s	0.6563	0.7358	0.5077	0.8019
2/15	40s	0.4958	0.8105	0.4646	0.8138
3/15	37s	0.4596	0.8205	0.5185	0.7761
4/15	37s	0.4293	0.8297	0.5085	0.8084
5/15	39s	0.4104	0.8338	0.4118	0.8375
6/15	40s	0.3796	0.8563	0.4090	0.8418
7/15	38s	0.3710	0.8568	0.4093	0.8342
8/15	38s	0.3537	0.8660	0.4260	0.8482
9/15	38s	0.3258	0.8737	0.4051	0.8558
10/15	37s	0.3148	0.8772	0.3917	0.8590
11/15	35s	0.2993	0.8825	0.3850	0.8751
12/15	37s	0.2875	0.8837	0.4000	0.8698
13/15	39s	0.2688	0.8999	0.4117	0.8536
14/15	37s	0.2494	0.9073	0.5154	0.8041
15/15	36s	0.2249	0.9109	0.4428	0.8547

`<keras.callbacks.History at 0x7f0648c9fa50>`

- **Save The Model**

The model is saved with .h5 extension as follows

An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

Save The Model

```
[21] model.save('nutrition.h5')
```

- **Test The Model**

- Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data.
- Load the saved model using load_model
- Taking an image as input and checking the results
- By using the model we are predicting the output for the given input image
- The predicted class index name will be printed here.

Test The Model

```
[22] from tensorflow.keras.models import load_model
      from keras.preprocessing import image
      final_model = load_model("nutrition.h5")

[23] from tensorflow.keras.utils import img_to_array

img = tensorflow.keras.utils.load_img("/content/drive/MyDrive/Nutrition Image Analysis using CNN and Rapid API/Nutrition Analysis Using Image Classification/Flask/Sample_Images/Test_Image4.jpg", grayscale=False)
x = img_to_array(img)
x = np.expand_dims(x, axis = 0)
pred = np.argmax(final_model.predict(x), axis=-1)
pred

1/1 [=====] - 0s 103ms/step
array([2])

[25] index=['APPLES', 'BANANA', 'ORANGE']
      result=index[pred[0]]
      result

'ORANGE'
```

