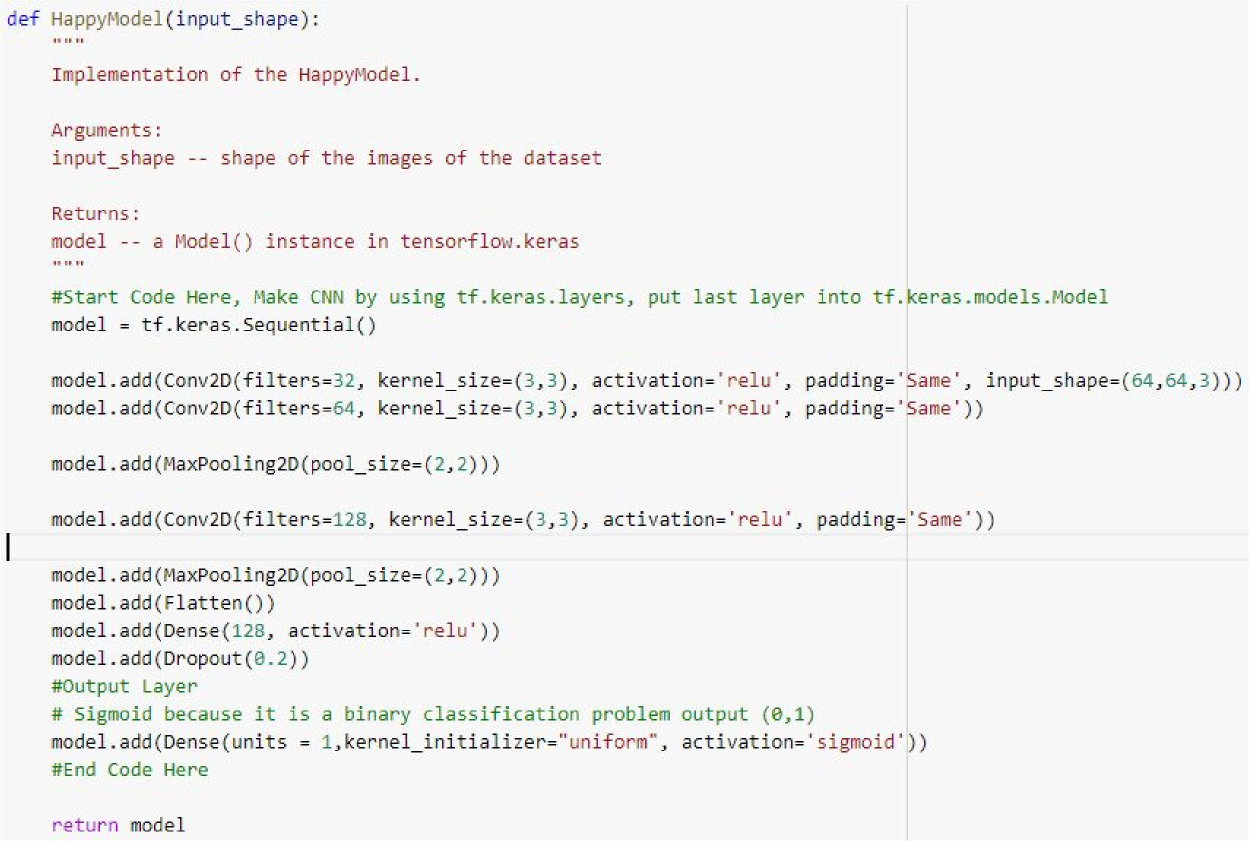
Deep Learning Assignment[3]

18.03.2020

Aya Ashraf Saber Mohamed [02]

CNN Model



```
def HappyModel(input_shape):
    """
    Implementation of the HappyModel.

    Arguments:
    input_shape -- shape of the images of the dataset

    Returns:
    model -- a Model() instance in tensorflow.keras
    """
    #Start Code Here, Make CNN by using tf.keras.layers, put last layer into tf.keras.models.Model
    model = tf.keras.Sequential()

    model.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu', padding='Same', input_shape=(64,64,3)))
    model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='Same'))

    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Conv2D(filters=128, kernel_size=(3,3), activation='relu', padding='Same'))

    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.2))
    #Output Layer
    # Sigmoid because it is a binary classification problem output (0,1)
    model.add(Dense(units = 1,kernel_initializer="uniform", activation='sigmoid'))
    #End Code Here

    return model
```

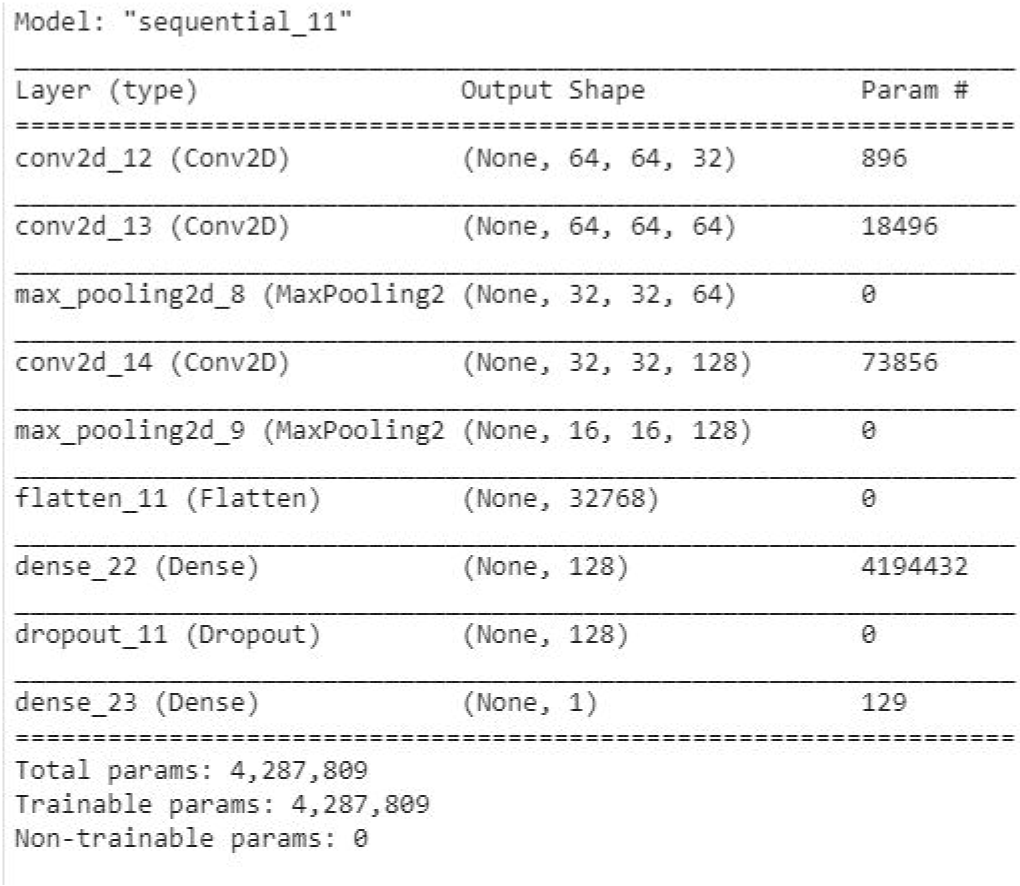
CNN Parameter Tuning:

1. Optimizer
2. Learning Rate
3. Loss Function

Evaluation:

- Testing Loss = 0.19356817821661632
- Testing Accuracy = 0.9266667

Number of Multiplication:



```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 64, 64, 32)        896
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 64, 64, 64)        18496
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 32, 32, 128)       73856
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
flatten_11 (Flatten)         (None, 32768)             0
_________________________________________________________________
dense_22 (Dense)             (None, 128)               4194432
_________________________________________________________________
dropout_11 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_23 (Dense)             (None, 1)                 129
=================================================================
Total params: 4,287,809
Trainable params: 4,287,809
Non-trainable params: 0
```

Based on this model

Input = (64 * 64 * 3)

Kernel1 = 3 * 3 * 32

Kernel2 = 3*3 * 64

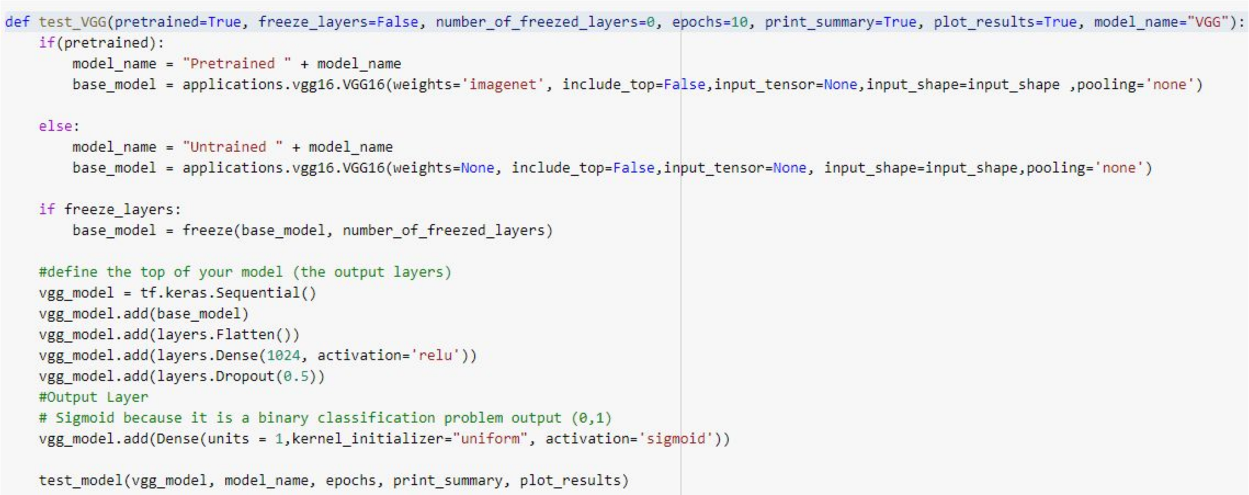
Kernel3 = 3*3 * 128

#nodeDense = 128

Dropout = 0.2

We multiply them all and get the answer.

VGG Models Architecture(VGG16)



```python
def test_VGG(pretrained=True, freeze_layers=False, number_of_freezed_layers=0, epochs=10, print_summary=True, plot_results=True, model_name="VGG"):
    if(pretrained):
        model_name = "Pretrained " + model_name
        base_model = applications.vgg16.VGG16(weights='imagenet', include_top=False,input_tensor=None,input_shape=input_shape ,pooling='none')

    else:
        model_name = "Untrained " + model_name
        base_model = applications.vgg16.VGG16(weights=None, include_top=False,input_tensor=None, input_shape=input_shape,pooling='none')

    if freeze_layers:
        base_model = freeze(base_model, number_of_freezed_layers)

    #define the top of your model (the output layers)
    vgg_model = tf.keras.Sequential()
    vgg_model.add(base_model)
    vgg_model.add(layers.Flatten())
    vgg_model.add(layers.Dense(1024, activation='relu'))
    vgg_model.add(layers.Dropout(0.5))
    #Output Layer
    # Sigmoid because it is a binary classification problem output (0,1)
    vgg_model.add(Dense(units = 1,kernel_initializer="uniform", activation='sigmoid'))

    test_model(vgg_model, model_name, epochs, print_summary, plot_results)
```

- Parameter Tuning # Freezing Layer
    - 0

      Testing Loss = 0.6906179396311442

      Testing Accuracy = 0.56
    - 5

      Testing Loss = 0.71372261206309

      Testing Accuracy = 0.44
    - 10

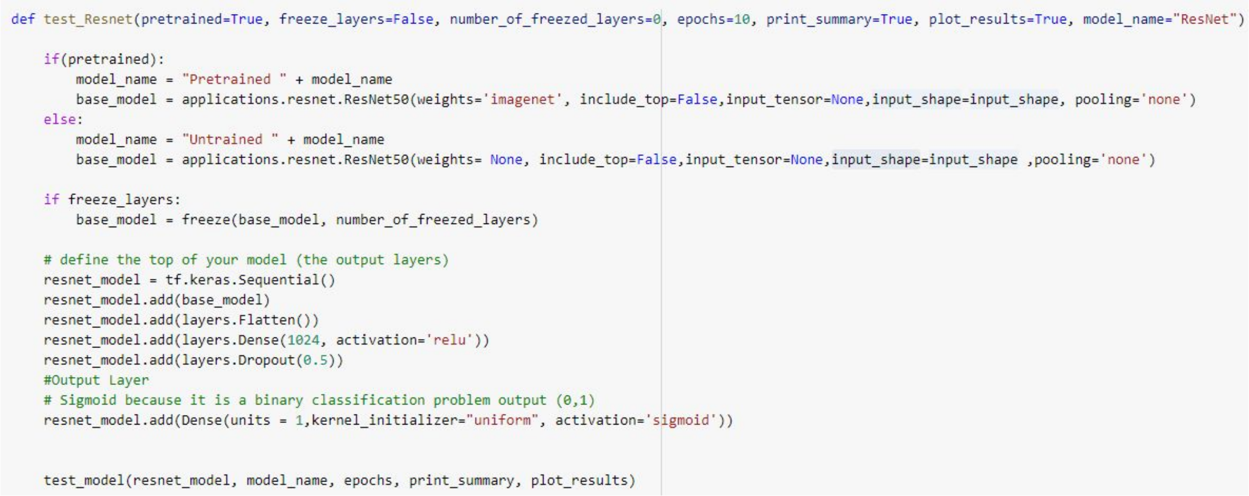
      Testing Loss = 0.6243742767969768

      Testing Accuracy = 0.64
    - 15

      Testing Loss = 0.6873150539398193

      Testing Accuracy = 0.56

Resenet model architecture



```
def test_Resnet(pretrained=True, freeze_layers=False, number_of_freezed_layers=0, epochs=10, print_summary=True, plot_results=True, model_name="ResNet")

    if(pretrained):
        model_name = "Pretrained " + model_name
        base_model = applications.resnet.ResNet50(weights='imagenet', include_top=False,input_tensor=None,input_shape=input_shape, pooling='none')
    else:
        model_name = "Untrained " + model_name
        base_model = applications.resnet.ResNet50(weights= None, include_top=False,input_tensor=None,input_shape=input_shape ,pooling='none')

    if freeze_layers:
        base_model = freeze(base_model, number_of_freezed_layers)

    # define the top of your model (the output layers)
    resnet_model = tf.keras.Sequential()
    resnet_model.add(base_model)
    resnet_model.add(layers.Flatten())
    resnet_model.add(layers.Dense(1024, activation='relu'))
    resnet_model.add(layers.Dropout(0.5))
    #Output Layer
    # Sigmoid because it is a binary classification problem output (0,1)
    resnet_model.add(Dense(units = 1,kernel_initializer="uniform", activation='sigmoid'))


    test_model(resnet_model, model_name, epochs, print_summary, plot_results)
```

- Parameter Tuning # Freezing Layer
    - 0

      Testing Loss = 13083.6949609375

      Testing Accuracy = 0.56

    - 10

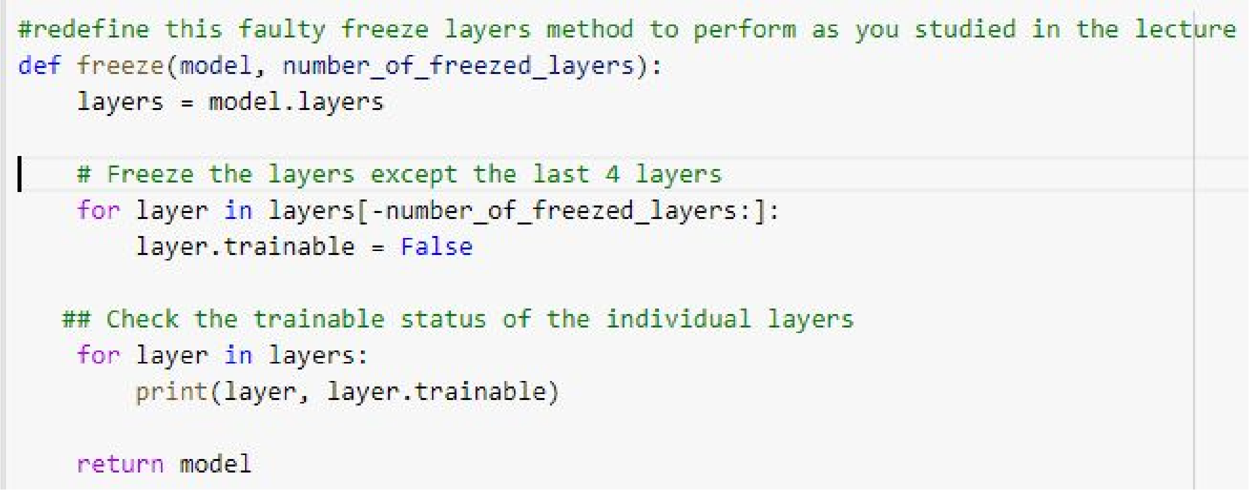
      Testing Loss = 0.6976297148068746

      Testing Accuracy = 0.44
    - 15

      Testing Loss = 1.5175211000442506

      Testing Accuracy = 0.56

Freeze Function



```
#redefine this faulty freeze layers method to perform as you studied in the lecture
def freeze(model, number_of_freezed_layers):
    layers = model.layers

    # Freeze the layers except the last 4 layers
    for layer in layers[-number_of_freezed_layers:]:
        layer.trainable = False

   ## Check the trainable status of the individual layers
    for layer in layers:
        print(layer, layer.trainable)

    return model
```

- Increasing the number of freezing layer helps to decrease the loss value
  This is because the size of the given dataset is small which leads to overfitting and increases the model accuracy.

Analysis

Model	Training Time	Testing Time
CNN	118.99391269683838	1.0268449783325195
VGG with pretrained	813.5019493103027	6.181899785995483
VGG without pretrained	635.5565857887268	6.20988130569458
ResNet with pretrained	674.2925441265106	2.0529401302337646
Resnet without pretrained	650.3117604255676	2.0192813873291016

Note:

The training time per epoch shown in the code running result.

1. Highest Accuracy:

CNN Model gives me 92% using (10) epoch.