Deep Learning Fundus Image Analysis for Early Detection of Diabetic Retinopathy

Team ID:PNT2022TMID23190

Model Building

Now it's time to build our model. Let's use the pre-trained model which is Xception, one of the convolution neural net (CNN) architectures which is considered as a very good model for Image classification.Deep understanding on the Xception model – Link is referred to in the prior knowledge section. Kindly refer to it before starting the model-building part.

Pre-Trained CNN Model As A Feature Extractor

For one of the models, we will use it as a simple feature extractor by freezing all the five convolution blocks to make sure their weights don't get updated after each epoch as we train our own model.Here, we have considered images of dimension (229,229,3).Also, we have assigned include_top = False because we are using convolution layer for features extraction and wants to train fully connected layer for our images classification(since it is not the part of Imagenet dataset)Flatten layer flattens the input. Does not affect the batch size.

Pre-Training CNN Model As A Feature Extractor

```
In [13]: xception = Xception(input_shape = imageSize + [3],weights='imagenet',include_top = False)

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83683744/83683744 [==============================] - 0s 0us/step

In [14]: for layer in xception.layers:
           layer.trainable = False

In [15]: x = Flatten()(xception.output)
```

Adding Dense Layers

A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer.Let us create a model object named model with inputs as xception.input and output as dense 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, summary to get the full information about the model and its layers.

Adding Dense Layers

```
In [16]: prediction = Dense( 5,activation ='softmax')(x)
```

```
In [17]: model = Model(inputs=xception.input,outputs=prediction)
```

```
In [18]: model.summary()
Model : "model"
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 299, 299, 3  0           []
                                )]

 block1_conv1 (Conv2D)          (None, 149, 149, 32  864         ['input_1[0][0]']
                                )

 block1_conv1_bn (BatchNormaliz  (None, 149, 149, 32  128        ['block1_conv1[0][0]']
 ation)                         )

 block1_conv1_act (Activation)  (None, 149, 149, 32  0           ['block1_conv1_bn[0][0]']
                                )

 block1_conv2 (Conv2D)          (None, 147, 147, 64  18432       ['block1_conv1_act[0][0]']
                                )

 block1_conv2_bn (BatchNormaliz  (None, 147, 147, 64  256        ['block1_conv2[0][0]']
 ation)                         )

 block1_conv2_act (Activation)  (None, 147, 147, 64  0           ['block1_conv2_bn[0][0]']
                                )

                                 ,

 block13_pool (MaxPooling2D)    (None, 10, 10, 1024  0           ['block13_sepconv2_bn[0][0]']
                                )

 batch_normalization_3 (BatchNo  (None, 10, 10, 1024  4096       ['conv2d_3[0][0]']
 rmalization)                   )

 add_11 (Add)                   (None, 10, 10, 1024  0           ['block13_pool[0][0]',
                                )                                 'batch_normalization_3[0][0]']

 block14_sepconv1 (SeparableCon  (None, 10, 10, 1536  1582080    ['add_11[0][0]']
 v2D)                           )

 block14_sepconv1_bn (BatchNorm  (None, 10, 10, 1536  6144       ['block14_sepconv1[0][0]']
 alization)                     )

 block14_sepconv1_act (Activati  (None, 10, 10, 1536  0          ['block14_sepconv1_bn[0][0]']
 on)                            )

 block14_sepconv2 (SeparableCon  (None, 10, 10, 2048  3159552    ['block14_sepconv1_act[0][0]']
 v2D)                           )

 block14_sepconv2_bn (BatchNorm  (None, 10, 10, 2048  8192       ['block14_sepconv2[0][0]']
 alization)                     )

 block14_sepconv2_act (Activati  (None, 10, 10, 2048  0          ['block14_sepconv2_bn[0][0]']
 on)                            )

 flatten (Flatten)              (None, 204800)       0           ['block14_sepconv2_act[0][0]']

 dense (Dense)                  (None, 5)            1024005     ['flatten[0][0]']

==================================================================================================
Total params: 21,885,485
Trainable params: 1,024,005
Non-trainable params: 20,861,480
__________________________________________________________________________________________________
```

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 optimizerMetrics are used to evaluate the performance of your model. It is similar to the loss function,but not used in the training process

Configuring The Learning Process

In [19]:
```
model.compile(
    loss = 'categorical_crossentropy',
    optimizer = 'adam',
    metrics =['accuracy']
)
```

Train The Model

Now, let us train our model with our image dataset. The model is trained for 30 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 10 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 - a generator- an 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 argumentcan 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 pointsin your dataset divided by the validation batch size.

Training The Model

In [20]:
```
# fit the model

r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=50,
  steps_per_epoch=len (training_set)//32,
  validation_steps=len(test_set)//32
)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future ver
sion. Please use `Model.fit`, which supports generators.
```

```
Epoch 1/50
3/3 [==============================] - 57s 17s/step - loss: 12.9591 - accuracy: 0.3854
Epoch 2/50
3/3 [==============================] - 48s 14s/step - loss: 12.7431 - accuracy: 0.5208
Epoch 3/50
3/3 [==============================] - 50s 15s/step - loss: 10.1967 - accuracy: 0.4479
Epoch 4/50
3/3 [==============================] - 51s 16s/step - loss: 7.5058 - accuracy: 0.6042
Epoch 5/50
3/3 [==============================] - 50s 15s/step - loss: 6.0366 - accuracy: 0.6042
Epoch 6/50
3/3 [==============================] - 52s 15s/step - loss: 6.0740 - accuracy: 0.6042
Epoch 7/50
3/3 [==============================] - 50s 15s/step - loss: 2.8408 - accuracy: 0.6667
Epoch 8/50
3/3 [==============================] - 50s 15s/step - loss: 3.9515 - accuracy: 0.6458
Epoch 9/50
3/3 [==============================] - 51s 15s/step - loss: 2.7660 - accuracy: 0.7396
Epoch 10/50
3/3 [==============================] - 50s 15s/step - loss: 4.5604 - accuracy: 0.5729
Epoch 11/50
```

```
3/3 [==============================] - 49s 15s/step - loss: 3.4445 - accuracy: 0.6146
Epoch 33/50
3/3 [==============================] - 48s 14s/step - loss: 2.2584 - accuracy: 0.6875
Epoch 34/50
3/3 [==============================] - 49s 15s/step - loss: 3.1585 - accuracy: 0.7708
Epoch 35/50
3/3 [==============================] - 49s 15s/step - loss: 3.7990 - accuracy: 0.7083
Epoch 36/50
3/3 [==============================] - 49s 15s/step - loss: 3.2899 - accuracy: 0.6875
Epoch 37/50
3/3 [==============================] - 50s 15s/step - loss: 2.2749 - accuracy: 0.7708
Epoch 38/50
3/3 [==============================] - 49s 14s/step - loss: 2.6622 - accuracy: 0.7292
Epoch 39/50
3/3 [==============================] - 47s 14s/step - loss: 3.2610 - accuracy: 0.7083
Epoch 40/50
3/3 [==============================] - 56s 18s/step - loss: 2.9940 - accuracy: 0.7188
Epoch 41/50
3/3 [==============================] - 50s 15s/step - loss: 2.1867 - accuracy: 0.8229
Epoch 42/50
3/3 [==============================] - 48s 15s/step - loss: 4.3965 - accuracy: 0.6667
Epoch 43/50
3/3 [==============================] - 49s 15s/step - loss: 3.4446 - accuracy: 0.7083
Epoch 44/50
3/3 [==============================] - 49s 15s/step - loss: 4.5254 - accuracy: 0.6875
Epoch 45/50
3/3 [==============================] - 51s 15s/step - loss: 4.9313 - accuracy: 0.7188
Epoch 46/50
3/3 [==============================] - 50s 15s/step - loss: 3.7867 - accuracy: 0.6771
Epoch 47/50
3/3 [==============================] - 48s 14s/step - loss: 4.0749 - accuracy: 0.7188
Epoch 48/50
3/3 [==============================] - 50s 15s/step - loss: 3.8776 - accuracy: 0.7292
Epoch 49/50
3/3 [==============================] - 55s 18s/step - loss: 4.7070 - accuracy: 0.6146
Epoch 50/50
3/3 [==============================] - 49s 15s/step - loss: 2.7252 - accuracy: 0.7812
```

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.

Saving the Model

```
In [21]: model.save("Updated-xception-diabetic-retinopathy.h5")
```