TEAM ID:

Model Building

1.Pre-trained CNN Model as feature extractor :

CNNs are known for providing good performance and high generalizability for classification tasks. The concept of generalization is directly linked to the performance achieved in data not seen by the network. Thus, we say that the network has generalization if it performs well with the known data (used during training) and with the unknown data (used for testing). However, for a CNN to be able to learn concepts intrinsic to the task, a lot of training data is necessary and, rarely, we will have this amount sufficient. So, an alternative presents itself as a possible solution: **using a CNN that has previously been trained as a feature extractor**.

```
[ ]
xception = Xception ( input_shape = img_size + [ 3 ] , weights = 'imagenet' , include_top = False )
# don't train existing weights
for layer in xception.layers :
    layer.trainable = False

x = Flatten ( ) ( xception.output )
```

Adding dense layers and configure the learning process:

In any neural network, a dense layer is a layer that is deeply connected with its preceding layer which means the neurons of the layer are connected to every neuron of its preceding layer. This layer is the most commonly used layer in artificial neural network networks.

It is important to find a good value for the learning rate for your model on your training dataset.

The learning rate may, in fact, be the most important hyper parameter to configure for your model.

Adding Dense Layers

```
[ ]
prediction = Dense ( 5 , activation = 'softmax' ) ( x )
model = Model ( inputs = xception.input , outputs = prediction )
```

```
model.summary()
```

```
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)                         )
```

Configure the Learning Process

```
# tell the model what cost and optimization method to use
model.compile (
 loss = 'categorical_crossentropy' ,
 optimizer = 'adam',
 metrics = [ 'accuracy' ])
```

Train the model:

- Load the dataset.
- Build the model (mention how many hidden layers we want along with their activation function)
- Define the loss function.
- Obtain training data and use an optimizer in your model.

```python
# fit the model
r = model.fit(
 training_set ,
 validation_data = test_set ,
 epochs = 30 ,
 steps_per_epoch = len ( training_set ) // 32 ,
 validation_steps = len ( test_set ) // 32)
```

```
Epoch 1/30
3/3 [==============================] - 38s 11s/step - loss: 11.2760 - accuracy: 0.4062
Epoch 2/30
3/3 [==============================] - 30s 9s/step - loss: 14.2826 - accuracy: 0.6354
Epoch 3/30
3/3 [==============================] - 30s 9s/step - loss: 16.8780 - accuracy: 0.5208
Epoch 4/30
3/3 [==============================] - 30s 9s/step - loss: 9.6413 - accuracy: 0.5104
```

Save the Model:

- Step 1 – Import the library. f
- Step 2 – Setting up the Data.
- Step 3 – Training and Saving the model.
- Step 4 – Loading the saved model.

```python
model.save ( "Updated-xception-diabetic-retinopathy.h5")
```