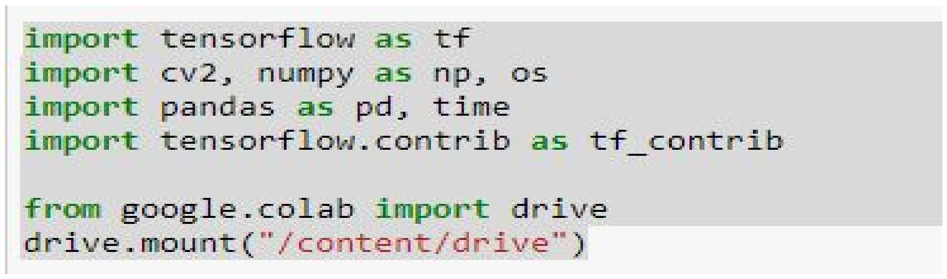
CV Assignment-03

R. Navanith,
S20160010071

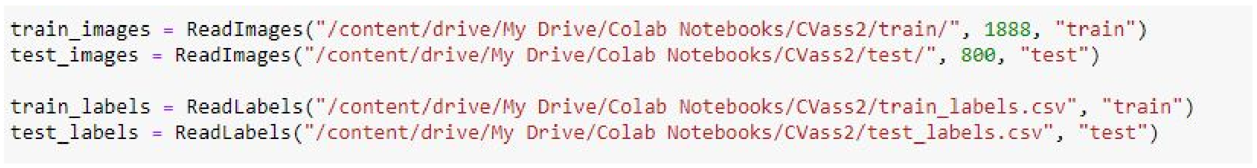
3.

(i) This peace of code is loading the required libraries and It connects to the tensorflow

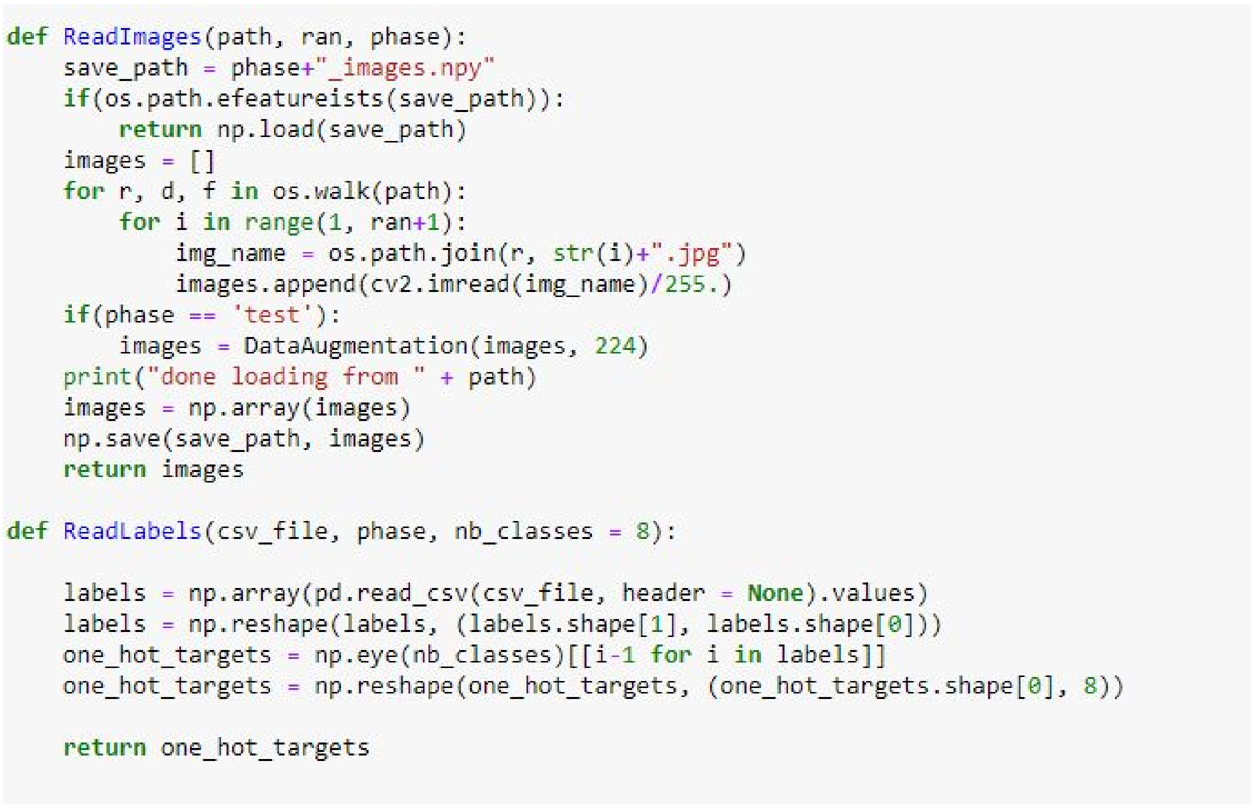


```
import tensorflow as tf
import cv2, numpy as np, os
import pandas as pd, time
import tensorflow.contrib as tf_contrib

from google.colab import drive
drive.mount("/content/drive")
```

(ii) This code is used for reading the images and the labels from the data

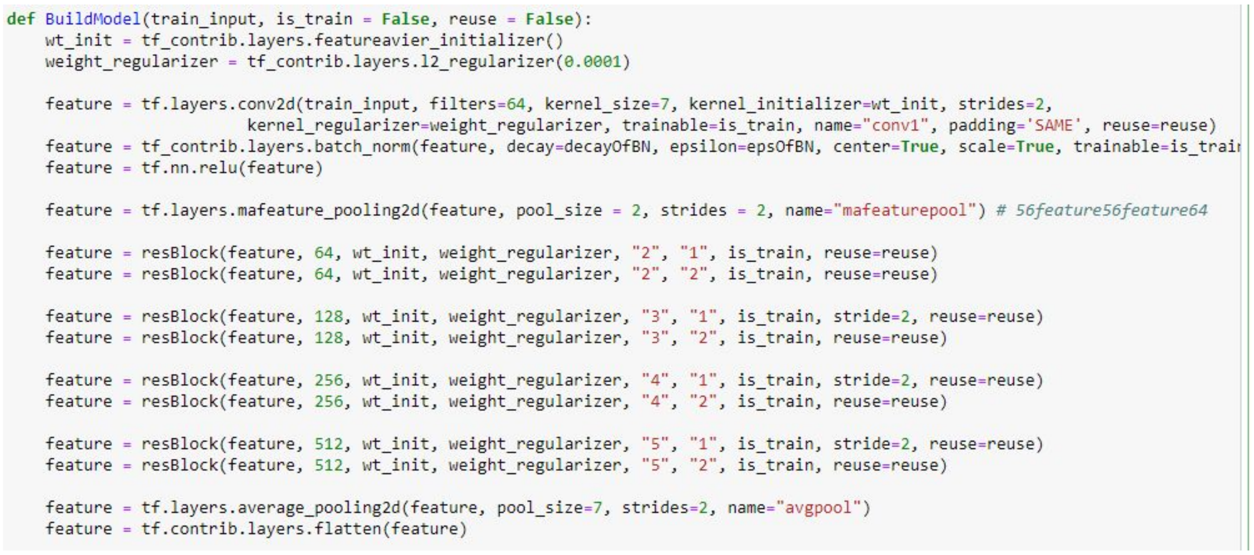


```
train_images = ReadImages("/content/drive/My Drive/Colab Notebooks/CVass2/train/", 1888, "train")
test_images = ReadImages("/content/drive/My Drive/Colab Notebooks/CVass2/test/", 800, "test")

train_labels = ReadLabels("/content/drive/My Drive/Colab Notebooks/CVass2/train_labels.csv", "train")
test_labels = ReadLabels("/content/drive/My Drive/Colab Notebooks/CVass2/test_labels.csv", "test")
```

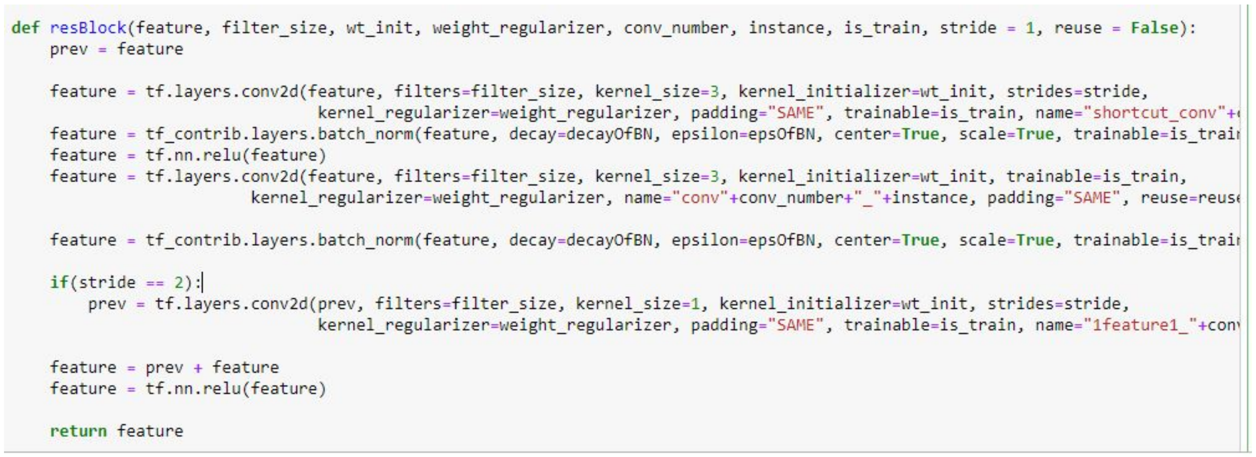


```python
def ReadImages(path, ran, phase):
    save_path = phase+"_images.npy"
    if(os.path.efeatureists(save_path)):
        return np.load(save_path)
    images = []
    for r, d, f in os.walk(path):
        for i in range(1, ran+1):
            img_name = os.path.join(r, str(i)+".jpg")
            images.append(cv2.imread(img_name)/255.)
    if(phase == 'test'):
        images = DataAugmentation(images, 224)
    print("done loading from " + path)
    images = np.array(images)
    np.save(save_path, images)
    return images

def ReadLabels(csv_file, phase, nb_classes = 8):

    labels = np.array(pd.read_csv(csv_file, header = None).values)
    labels = np.reshape(labels, (labels.shape[1], labels.shape[0]))
    one_hot_targets = np.eye(nb_classes)[[i-1 for i in labels]]
    one_hot_targets = np.reshape(one_hot_targets, (one_hot_targets.shape[0], 8))

    return one_hot_targets
```

(iii) Code to build the the model using tensorflow

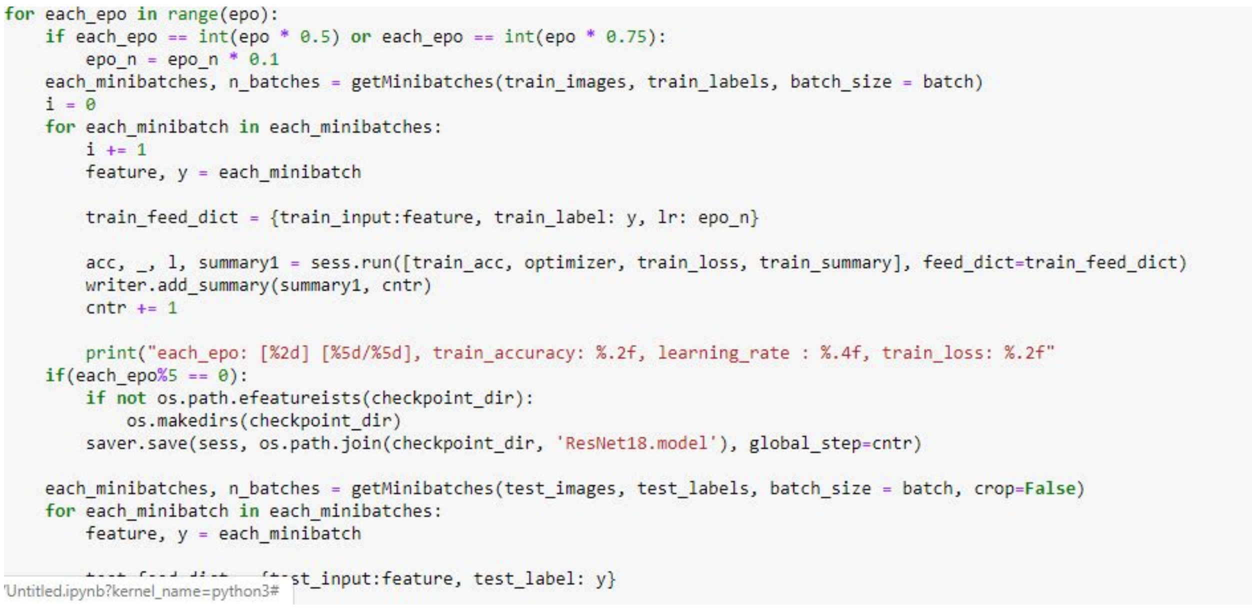


```python
def BuildModel(train_input, is_train = False, reuse = False):
    wt_init = tf_contrib.layers.featureavier_initializer()
    weight_regularizer = tf_contrib.layers.l2_regularizer(0.0001)

    feature = tf.layers.conv2d(train_input, filters=64, kernel_size=7, kernel_initializer=wt_init, strides=2,
                     kernel_regularizer=weight_regularizer, trainable=is_train, name="conv1", padding='SAME', reuse=reuse)
    feature = tf_contrib.layers.batch_norm(feature, decay=decayOfBN, epsilon=epsOfBN, center=True, scale=True, trainable=is_trai
    feature = tf.nn.relu(feature)

    feature = tf.layers.mafeature_pooling2d(feature, pool_size = 2, strides = 2, name="mafeaturepool") # 56feature56feature64

    feature = resBlock(feature, 64, wt_init, weight_regularizer, "2", "1", is_train, reuse=reuse)
    feature = resBlock(feature, 64, wt_init, weight_regularizer, "2", "2", is_train, reuse=reuse)

    feature = resBlock(feature, 128, wt_init, weight_regularizer, "3", "1", is_train, stride=2, reuse=reuse)
    feature = resBlock(feature, 128, wt_init, weight_regularizer, "3", "2", is_train, reuse=reuse)

    feature = resBlock(feature, 256, wt_init, weight_regularizer, "4", "1", is_train, stride=2, reuse=reuse)
    feature = resBlock(feature, 256, wt_init, weight_regularizer, "4", "2", is_train, reuse=reuse)

    feature = resBlock(feature, 512, wt_init, weight_regularizer, "5", "1", is_train, stride=2, reuse=reuse)
    feature = resBlock(feature, 512, wt_init, weight_regularizer, "5", "2", is_train, reuse=reuse)

    feature = tf.layers.average_pooling2d(feature, pool_size=7, strides=2, name="avgpool")
    feature = tf.contrib.layers.flatten(feature)
```



```python
def resBlock(feature, filter_size, wt_init, weight_regularizer, conv_number, instance, is_train, stride = 1, reuse = False):
    prev = feature

    feature = tf.layers.conv2d(feature, filters=filter_size, kernel_size=3, kernel_initializer=wt_init, strides=stride,
                               kernel_regularizer=weight_regularizer, padding="SAME", trainable=is_train, name="shortcut_conv"+
    feature = tf_contrib.layers.batch_norm(feature, decay=decayOfBN, epsilon=epsOfBN, center=True, scale=True, trainable=is_trai
    feature = tf.nn.relu(feature)
    feature = tf.layers.conv2d(feature, filters=filter_size, kernel_size=3, kernel_initializer=wt_init, trainable=is_train,
                          kernel_regularizer=weight_regularizer, name="conv"+conv_number+"_"+instance, padding="SAME", reuse=reus

    feature = tf_contrib.layers.batch_norm(feature, decay=decayOfBN, epsilon=epsOfBN, center=True, scale=True, trainable=is_trai

    if(stride == 2):
        prev = tf.layers.conv2d(prev, filters=filter_size, kernel_size=1, kernel_initializer=wt_init, strides=stride,
                               kernel_regularizer=weight_regularizer, padding="SAME", trainable=is_train, name="1feature1_"+con

    feature = prev + feature
    feature = tf.nn.relu(feature)

    return feature
```

(iv) Code to train the model and print the accuracy



```python
for each_epo in range(epo):
    if each_epo == int(epo * 0.5) or each_epo == int(epo * 0.75):
        epo_n = epo_n * 0.1
    each_minibatches, n_batches = getMinibatches(train_images, train_labels, batch_size = batch)
    i = 0
    for each_minibatch in each_minibatches:
        i += 1
        feature, y = each_minibatch

        train_feed_dict = {train_input:feature, train_label: y, lr: epo_n}

        acc, _, l, summary1 = sess.run([train_acc, optimizer, train_loss, train_summary], feed_dict=train_feed_dict)
        writer.add_summary(summary1, cntr)
        cntr += 1

        print("each_epo: [%2d] [%5d/%5d], train_accuracy: %.2f, learning_rate : %.4f, train_loss: %.2f"
    if(each_epo%5 == 0):
        if not os.path.efeatureists(checkpoint_dir):
            os.makedirs(checkpoint_dir)
        saver.save(sess, os.path.join(checkpoint_dir, 'ResNet18.model'), global_step=cntr)

    each_minibatches, n_batches = getMinibatches(test_images, test_labels, batch_size = batch, crop=False)
    for each_minibatch in each_minibatches:
        feature, y = each_minibatch

        test_feed_dict = {test_input:feature, test_label: y}
```

'Untitled.ipynb?kernel_name=python3#

Train accuracy of the results are : 93.2
Test accuracy is 89.32

Thank you