Neural Networks Project

Team ID:
Team names: Mohamed Galal Mohamed, Sameh Maged Abdelrahman, Nadine Walid Adly, Lojain Wail Mohamed, Youssef Amr Abdelmoneim

- 1st submission: 10% accuracy – the create test function was wrong as The images' names **weren't ordered** as in create test function it classified images while they were random but when we output the test we match the output with an **ordered test list** so the classification of testing images were wrong.

```
img_names=[]
def create_test_data():
  test = []

  for ImgName in sorted(os.listdir("/content/gdrive/MyDrive/[NN'22] Project Dataset/Test")):
    ImgPath = os.path.join("/content/gdrive/MyDrive/[NN'22] Project Dataset/Test", ImgName)
    img_names.append(ImgName)
    imgdata = cv2.imread(ImgPath, 1)

    if(ImgPath=="/content/gdrive/MyDrive/[NN'22] Project Dataset/Test/test_1297.jpg"):
      imgdata = cv2.imread("/content/gdrive/MyDrive/[NN'22] Project Dataset/Test/test_1.jpg", 1)

      imgdata = cv2.resize(np.array(imgdata), (IMG_SIZE, IMG_SIZE))
      test.append(np.array(imgdata))
      #test.append([np.array(imgdata), create_label(ImgName)])
      continue

    else:

      imgdata = cv2.resize(np.array(imgdata), (IMG_SIZE, IMG_SIZE))
      test.append(np.array(imgdata))

  return test
```

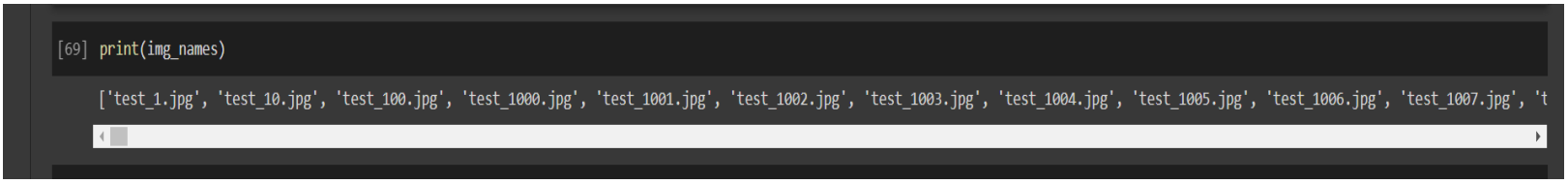
- 2nd and 3rd submissions: 9%, 8% accuracy – same issue in testing we couldn't detect it at first but we tried to change the model to **LENET** and then another manually created neural network architecture with convolutions, max pooling and dense layers so as to increase the accuracy but we failed because of the major testing issue.

```
from keras import models, layers
import keras
model = keras.models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',input_shape=(244,244,3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(11, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy, optimizer=tf.keras.optimizers.Adam()
                    ,metrics=['accuracy'] )

model.summary()
xt,xte,yt,yte=train_test_split( x_train, y_train,test_size=0.1,random_state=42)

history= model.fit(
     xt,
     yt,
     epochs=10,batch_size=32,validation_data=(xte,yte))
```

- 4th submission: 7% accuracy - we kept trying to increase the terrible accuracy that we had thinking the problem is in the model we used as our testing code was correct except for the fact that we matched the outputs with the test images wrongly, so we tried different models such as **ALEXNET**.

```
#Model 1 - ALEXNET
input_conv = input_data(shape=[None, 227, 227, 3], name='input')

conv_1 = conv_2d(input_conv,96,11,strides=4,activation='relu')
pool_1 = max_pool_2d(conv_1,3,strides=2)

conv_2 = conv_2d(pool_1,256,5,activation='relu')
pool_2 = max_pool_2d(conv_2,3,strides=2)

conv_3 = conv_2d(pool_2,384,3,activation='relu')
conv_4 = conv_2d(conv_3,384,3,activation='relu')
conv_5 = conv_2d(conv_4,256,3,activation='relu')

pool_3 = max_pool_2d(conv_5,3,strides=2)
fully_layer1 = fully_connected(pool_3, 4096, activation='relu')
fully_layer2 = fully_connected(fully_layer1, 4096, activation='relu')

cnn_layers = fully_connected(fully_layer2, 11, activation='softmax')
cnn_layers = regression(cnn_layers, optimizer='SGD', learning_rate=0.1, loss='categorical_crossentropy', name='targets')

model = tflearn.DNN(cnn_layers, tensorboard_dir='log', tensorboard_verbose=3)

model.fit({'input': x_train}, {'targets': y_train}, batch_size=64 ,n_epoch=10,
          validation_set=({'input': x_test}, {'targets': y_test}),
          run_id='Weather-classification')
```

- 5th submission: 64% accuracy - we fixed the testing issue so instead of testing on ordered images that misclassified, we saved the order of the images of the test file beforehand in a list named **"img_names"** so as to classify/ match the output (prediction) of the testing from the predict function at the end using this list, so the classes were more accurately classified giving us a way higher accuracy.




```
df_test = pd.DataFrame ({ "image_name" :   img_names  })

prediction = model.predict(test)

df_test["label"] = np.argmax(prediction,axis=1)
```

- 6th submission: 66% accuracy – we used a regular CNN architecture with **convolutions, pooling, dense layers** and dropouts specifically the one we took during the CNN lab. That was after we tried reimplementing LENET and ALEXNET after fixing the testing issue, but their accuracies weren't more than 64% but the following CNN network increased our accuracy by about 2%.

```
conv_input = input_data(shape=[None, IMG_SIZE, IMG_SIZE, 1], name='input')
conv1 = conv_2d(conv_input, 32, 5, activation='relu')
pool1 = max_pool_2d(conv1, 5)

conv2 = conv_2d(pool1, 64, 5, activation='relu')
pool2 = max_pool_2d(conv2, 5)

conv3 = conv_2d(pool2, 128, 5, activation='relu')
pool3 = max_pool_2d(conv3, 5)

conv4 = conv_2d(pool3, 64, 5, activation='relu')
pool4 = max_pool_2d(conv4, 5)

conv5 = conv_2d(pool4, 32, 5, activation='relu')
pool5 = max_pool_2d(conv5, 5)

fully_layer = fully_connected(pool5, 1024, activation='relu')
fully_layer = dropout(fully_layer, 0.5)

cnn_layers = fully_connected(fully_layer, 2, activation='softmax')

cnn_layers = regression(cnn_layers, optimizer='adam', learning_rate=LR, loss='categorical_crossentropy', name='targets')
model = tflearn.DNN(cnn_layers, tensorboard_dir='log', tensorboard_verbose=3)
```

- 7th, 8th and 9th submissions: 68% & 69% accuracies - we sticked to the previous CNN architecture and we started tuning the model parameters so we added:

1. **Batch Normalization layers:** as it is a process to make neural networks faster and more stable through adding extra layers in a deep neural network. The new layer performs the standardizing and normalizing operations on the input of a layer coming from a previous layer. So it 'resets' the distribution of the output of the previous layer to be more efficiently processed by the subsequent layer and dramatically reducing the number of training epochs required to train deep networks.

2. Increased the **IMG_SIZE** to train better and so as not to cause vanishing gradient.

3. Added a **dropout layer** to avoid overfitting by turning off useless neurons to enhance the training performance and accuracy.

```
import keras

IMG_SIZE = 150
model=keras.models.Sequential([

 keras.layers.Conv2D( 32, 3, activation='relu',input_shape=(IMG_SIZE,IMG_SIZE,3)),
keras.layers.BatchNormalization(),

keras.layers.MaxPool2D( 2),

keras.layers.Conv2D( 64, 2, activation='relu'),
keras.layers.BatchNormalization(),

keras.layers.MaxPool2D( 2),
keras.layers.Dropout(0.3),

keras.layers.Conv2D( 128, 3, activation='relu'),
keras.layers.BatchNormalization(),

keras.layers.MaxPool2D( 2),

keras.layers.Conv2D( 64, 2, activation='relu'),
keras.layers.BatchNormalization(),

keras.layers.MaxPool2D( 2),
keras.layers.Dropout(0.2),

keras.layers.Conv2D( 32, 3, activation='relu'),
keras.layers.BatchNormalization(),

keras.layers.MaxPool2D( 2),
#keras.layers.Conv2D( 32, 2, activation='relu'),

#keras.layers.MaxPool2D( 2),
#keras.layers.Conv
keras.layers.Flatten(),
keras.layers.Dense(500,activation='relu'),
keras.layers.Dropout(0.2),
keras.layers.Dense(11,activation='softmax')
])
```

- 10th and 11th submissions: 73% and 74% accuracies. We applied more tuning options for our CNN architecture such as:

    1. increasing the **number of dense layers** in our network hence, increasing the number of neurons and weights so the network becomes deeper and our model could train better to predict the classes but this increase shouldn't be huge as it might cause overfitting and in turn reduce accuracy on the test data. We noticed the optimal dense layers were 700.

2. We adjusted the optimizers to minimize the error/loss function so we tried: adagrad, adamax and many others and the best optimizer was **ADAM** optimizer as it doesn't roll so fast so as not to jump over the global minimum loss, it decreases the velocity a little bit for a careful search. In addition to storing an exponentially decaying average of past gradients.

3. We added an **Early Stopping** and adjusted its value with trial and error. The early stopping allows us to specify an arbitrarily large number of training epochs and stop training once the model performance stops improving on the validation dataset. So, it avoids overfitting by monitoring the validation loss.

```
keras.layers.Flatten(),
keras.layers.Dense(700,activation='relu'),
keras.layers.Dropout(0.2),
keras.layers.Dense(11,activation='softmax')
])
xt,xte,yt,yte=train_test_split(x_train, y_train,test_size=0.2,random_state=42)

model.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.Adam(0.0005),
    metrics=['accuracy']
)

callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',patience=7)
history= model.fit(
     xt, yt,
     epochs=30,batch_size=32,validation_data=(xte,yte))
```

- 12th, 13th and 14th submissions: 64% - 66% - 70% accuracies – we noticed that our model overfits on the test data so, we tried to adjust the learning rate parameter and search for more ways to reduce overfitting but the accuracy decreased as the learning rate we used was very small.

- 15th,16th and 17th submissions: 75% - 76% accuracies – more towards solving overfitting we applied some methods that increased our accuracy by around 10% which are:
    1. We applied regularization using 'ridge regularization' (L2) that tunes the weights according to the loss to avoid overfitting.
    2. We used **"kernel_regularizer"** and **"bias_regularizer"** for our dense layers, the first is for the filters so it adjusts the weights and the second regularizes the bias.
    3. We also added **"reduced learning rate on plateau"** as it will allow the optimizer to more efficiently find the minimum in the loss surface, allows large weight changes in the beginning of the learning process and small changes or fine-tuning towards the end of the learning process, so decreasing the learning rate over time helped in reducing overfitting and stabilizing the validation loss.

```
keras.layers.Flatten(),
keras.layers.Dense(700,activation='relu',kernel_regularizer='l2'),
keras.layers.Dropout(0.4),
keras.layers.Dense(11,activation='softmax',kernel_regularizer='l2')
])
xt,xte,yt,yte=train_test_split(x_train, y_train,test_size=0.1,random_state=42)

model.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.Adam(0.0005),
    metrics=['accuracy']
)

callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',patience=12)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', min_lr=0.0000001,patience=5)
history= model.fit(
     xt, yt,
     epochs=50,batch_size=32,validation_data=(xte,yte),callbacks=[reduce_lr] )
```

- 18^{th},19^{th} and 20^{th} submissions: 72% & 75% accuracies - we tried applying several **data preprocessing techniques** such as centering, scaling and normalization as it is better to normalize your data for training a Neural Network to obtain a mean close to 0. Normalizing the data generally speeds up learning and leads to faster convergence. But unfortunately, the accuracy wasn't bad but it wasn't the best that we reached.

```
[ ]  x_train = []
     y_train = []
     for idx in range (len(train)):
       pixels = train[idx][0]
       pixels = pixels.astype('float32')
       means = pixels.mean(axis=(0,1), dtype='float64')   #Centring
       pixels -= means

       ###################################
       pixels = pixels.astype('float32')
       mean, std = pixels.mean(), pixels.std()
       print('Mean: %.3f, Standard Deviation: %.3f' % (mean, std))   #Postive Global Stand
       pixels = (pixels - mean) / std                                #like zscore
       pixels = np.clip(pixels, -1.0, 1.0)
       pixels = (pixels + 1.0) / 2.0


       x_train.append(pixels)
     for idx in range (len(train)):
       y_train.append(train[idx][1])
     x_train = np.array(x_train)
     y_train = np.array(y_train)
```

- <u>The final submissions 21^{st} , 22^{nd} and 23^{rd}:</u> 82% & 83% accuracies - We used a different model which is **"VGG19"** so, we applied the data on all the model layers which consists of convolutions and max pooling with padding = "same" and activation = "relu".

Moreover, we applied **transfer learning** in which: we uploaded some pretrained weights on the model VGG19 to be used at first so as to start with a more accurate initialization and then we used these weights on the model we created to train it and test on our data. We used SGD optimizer as it is the best optimizer for VGG19 model by trial and error. In a comparison between SGD and ADAM, SGD is more locally unstable and is more likely to converge to the minima which often has better generalization performance.

```
WEIGHTS_PATH_NO_TOP = ('https://github.com/fchollet/deep-learning-models/'
                       'releases/download/v0.1/'
                       'vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5')

weights_path = tf.keras.utils.get_file('vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5',WEIGHTS_PATH_NO_TOP)
model.load_weights(weights_path)

model.add(Flatten())

model.add(Dense(units=11, activation="softmax",kernel_regularizer='l2',bias_regularizer='l2'))

model.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.SGD(0.001),
    metrics=['accuracy']
)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', min_lr=0.00001)

history= model.fit(
     xt, yt,
     epochs=30,validation_data=(xte,yte),callbacks=[reduce_lr] )


model.summary()
```

```
#VGG19
from keras.models import Sequential
from keras.layers.core import Flatten, Dense, Dropout
from keras.layers.convolutional import Convolution2D, MaxPooling2D, ZeroPadding2D
from keras.layers import BatchNormalization
model = Sequential()
model.add(Convolution2D(64, 3, 3, activation='relu',input_shape=(IMG_SIZE,IMG_SIZE,3)))

model.add(Convolution2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))

model.add(Convolution2D(filters=128,kernel_size=(3,3),padding="same", activation="relu"))
model.add(Convolution2D(filters=128,kernel_size=(3,3),padding="same", activation="relu"))

model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Convolution2D(filters=256,kernel_size=(3,3),padding="same", activation="relu"))

model.add(Convolution2D(filters=256,kernel_size=(3,3),padding="same", activation="relu"))
model.add(Convolution2D(filters=256,kernel_size=(3,3),padding="same", activation="relu"))

model.add(Convolution2D(filters=256,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))

model.add(Convolution2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))

model.add(Convolution2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(Convolution2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))

model.add(Convolution2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))

model.add(Convolution2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))

model.add(Convolution2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(Convolution2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))

model.add(Convolution2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
```