TRANSFER LEARNING

In this exercise, you will use the Tensorflow Dataset's Splits API and its concepts which you looked at in the week 2 lecture videos. Also

Also, you will look at some additional ways of loading things using Tensorflow Hub using the cats_vs_dogs v4 dataset.

Finally, you will use transfer learning using a pretrained feature vector from mobilenet to define a new classification model in the end.

Upon completion of the exercise, you will have

- Loaded a learnt feature set from mobilenet model
- Split the cats_vs_dogs dataset in custom train,validation and test sets.
- Shuffled and batched the custom sets.
- Defined the model which is ready for tranfer learning using the mobilenet feature vector.

Step 0 - Import libraries and set up the splits

Note : The assignment uses TF version `2.1.0` and TFDS version `3.2.1` so if you run this notebook on TF 1.x or some other version of TFDS, some things might not work.

```
import tensorflow as tf
import tensorflow_datasets as tfds
from os import getcwd

print(tf.__version__)
print(tfds.__version__)
```

```
2.9.1
4.7.0
```

Step 1 - Load the Mobilenet model and its features

The next code block will download the `mobilenet model` from TensorFlow Hub, and use its learned features, extracted as feature_extractor and set to be fine tuned by making them trainable.

It is already downloaded for you but feel free to use the commented part to download the latest version from the tfhub.dev website when experimenting on your local machine.

```
import tensorflow_hub as hub

model_selection = ("mobilenet_v2", 224, 1280)
handle_base, pixels, FV_SIZE = model_selection
IMAGE_SIZE = (pixels, pixels)
```

```
filePath = f"{getcwd()}/data"

# You can also use directly to download from the source.

# MODULE_HANDLE
="https://tfhub.dev/google/tf2-preview/{}/feature_vector/4".format(han
dle_base)
# feature_extractor = hub.KerasLayer(MODULE_HANDLE,
#                                    input_shape=IMAGE_SIZE + (3,))

# The data is already downloaded for you at the filepath

feature_extractor =
hub.KerasLayer(filePath+'/mobilenet_v2_feature_vector',input_shape=IMA
GE_SIZE + (3,))
feature_extractor.trainable = True
```

Step 2 - Split the dataset

You need to use subsets of the original cats_vs_dogs data, which is entirely in the 'train' split. I.E. 'train' contains `25000` records with `1738` corrupted images to in total you have `23,262` images.

You will split it up to get

- The first 10% is as the 'new' training set
- The last 10% is as the new validation and test sets, split down the middle
  - i.e. the first half of the last 10% is validation (first 5%)
  - the second half is test (last 5%)

These 3 recordsets should be called `train_examples`, `validation_examples` and `test_examples` respectively.

Note: Remember to use `cats_vs_dogs:4.*.*` as dataset because 4.0 support the new Splits API.

```
# EXERCISE: Split the dataset

splits = ['train[:10%]', 'train[90%:95%]', 'train[95%:]']

# Remember to use `cats_vs_dogs:4.*.*`
# https://www.tensorflow.org/datasets/catalog/cats_vs_dogs

# It has been downloaded for you so use the data_dir parameter
# else it will try to download the dataset and give you an error here

splits, info = tfds.load('cats_vs_dogs', data_dir = filePath, split =
splits, with_info = True)

(train_examples, validation_examples, test_examples) = splits
```

```
# Testing lengths of the data if they are loaded correctly. Do not
edit the code below

train_len = len(list(train_examples))
validation_len = len(list(validation_examples))
test_len = len(list(test_examples))
print(train_len)
print(validation_len)
print(test_len)
```

```
2326
1163
1163
```

Expected Output

```
2326
1163
1163
```

Step 3 - Shuffle and map the new batches

Now, you will take few of the examples from train set and shuffle them initially.

Then, you will map a custom function `format_image` formats the image by resizing them first to `(224, 224)` as that is the input image size for mobilenet and post resizing, it normalizes the image by dividing each pixel by 255.

Finally, you will create train, test and validation batches with size `16` here because of memory constraints. Do not edit the `BATCH_SIZE` in the code cell and while submitting the assignment.

Feel free to play around the `BATCH_SIZE` and other parameters if you are running this locally just for experimenting.

```
num_examples = 500 # DO NOT CHANGE THIS FOR THE GRADER, UPDATE AND USE
IT WHEN PLAYING AROUND AND TRAINING IT LOCALLY.
num_classes = 2

# EXERCISE: shuffle and map the batches

# This will turn the 3 sets into batches
# so you can train and load batches


def format_image(features):
    image = features['image']
    image = tf.image.resize(image, IMAGE_SIZE) / 255.0
    return  image, features['label']
```

```
BATCH_SIZE =  16 # DO NOT EDIT

# For training batches, shuffle the examples by num_examples, map
using the function defined above
# and batching using the batch_size.

# For validation and test batches, just avoid shuffling and follow the
rest as training batch example

train_batches =
train_examples.shuffle(num_examples).map(format_image).batch(BATCH_SIZ
E)
validation_batches =
validation_examples.map(format_image).batch(BATCH_SIZE)
test_batches = test_examples.map(format_image).batch(BATCH_SIZE)
```

Step 4 - Define your transfer learning model

Here, you will use the mobilenet feature vector which you loaded before from Tensorflow Hub to create a new model for training.

This is a simple model where you are just using the feature vectors and adding the final dense layer to get the cat/dog classification.

Note: Always be careful of the input and output dimensions for a model loaded for transfer learning and use summary to check the dimensions.

```
# EXERCISE: Define the model

# The new model will take the features from the mobilenet via transfer
learning
# And add a new dense layer at the bottom, with 2 classes -- for cats
and dogs

model = tf.keras.Sequential([
        feature_extractor,
        tf.keras.layers.Dense(2, activation='softmax')
])

# Model summary to test your model layers, output shape and number of
parameters.
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 1280)              2257984

 dense (Dense)               (None, 2)                 2562
```

```
=================================================================
Total params: 2,260,546
Trainable params: 2,226,434
Non-trainable params: 34,112
_________________________________________________________________
```

Expected output

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 1280)              2257984
_________________________________________________________________
dense (Dense)                (None, 2)                 2562
=================================================================
Total params: 2,260,546
Trainable params: 2,226,434
Non-trainable params: 34,112
```

Submission Instructions

```
# Now click the 'Submit Assignment' button above.
```

[Optional] Step 5 - Training your model

Training is not in the scope of this assignment but you can go ahead and train the network to achieve decent accuracy of 90% and above by training for epochs less than 5.

Note:This would take quite a lot of time to train on CPU (30-40 minutes per epoch) while here it can take 2-3 minutes per epoch.

Remember to submit your assignment before you uncomment and run the next cells.

```
# # Compile the model with adam optimizer and sparse categorical
crossentropy,
# # and track the accuracy metric

# model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['acc'])

# # Train it for a number of epochs. You should not need many (max 5)
# # Train on the train_batches, and validation on the
validation_batches

# EPOCHS = # YOUR CODE HERE
# history = model.fit(train_batches, epochs=EPOCHS,
validation_data=validation_batches,verbose=1)

# model.summary()

# # Evaluate the model on the test batches
# eval_results = model.evaluate(test_batches)

# # And print the results.
# for metric, value in zip(model.metrics_names, eval_results):
#     print(metric + ': {:.4}'.format(value))
```

When you're done or would like to take a break, please run the two cells below to save your work and close the Notebook. This frees up resources for your fellow learners.

```
%%javascript
<!-- Save the notebook -->
IPython.notebook.save_checkpoint();
```

```
%%javascript
<!-- Shutdown and close the notebook -->
window.onbeforeunload = null
window.close();
IPython.notebook.session.delete();
```