Chapter 14 Programming Exercise: Transfer Learning

Exercise: Transfer Learning

In this exercise, you will use transfer learning to train a model to classify images belonging to a large number of classes. More in particular, you will use the `calltech101` dataset which is part of `tensorflow_datasets`. This dataset contains 101 classes of images as well as an additional `background clutter` classes. (The total number of labels is hence 102!) The dataset is described at https://www.tensorflow.org/datasets/catalog/caltech101.

The base model you are going to use is the `MobileNetV2` model. This link has more information.

Step 1: Load the dataset

Load the dataset using `tensorflow_datasets`. We will use the first 50% of the test set as our validation set, the last 50% of the test will be used as our test set. We will use the complete training set for training.

Verify that your datasets have the following sizes:

- Training set: 3060
- Validation set: 3042
- Test set: 3042

  Note: in this exercise we will **not** convert the `Datasets` to `numpy` arrays. We will use the `Datasets` directly.

Check that datatype of the images and the labels by completing the following code:

```python
for image, label in train_ds.take(1):
    # YOUR CODE HERE
```

Step 2: Some Data Exploration

Show the first nine image of the training set in a 3×3 grid.

- Print the label and the size of the image on top of each image.

You will notice that the images have different sizes. Resize the images to 224×224 using a keras preprocessing layer in conjunction with the `map` method of the `Dataset` class.

More in particular, complete the following code:

```python
def resize(image, height=224, width=224):
    # YOUR CODE HERE
```

```
train_ds = train_ds.map(lambda image, label : pass) # CHANGE pass TO THE CORRECT EXPRESSION
val_ds = # YOUR CODE HERE
test_ds = # YOUR CODE HERE
```

Look at the images again, and verify that they are all of the same size.

> Note: you may find that you need a small adaption to the previous code for showing images. The tensors are now of type `tf.float32`.

Step 3: Preprocess the Data

You should now change the `Dataset`s such that

- the images are preprocessed for the MobileNetV2 model. Use the preprocessing function `tf.keras.applications.mobilenet_v2.preprocess_input` for this.
- batch the images in batches of size 32
- shuffle the training set. There is no need to shuffle the validation and test set.
- Use the `prefetch` method on the training `Dataset` to prefetch the next batch while training.

Step 4: Create a Data Augmentation Layer

Since we don't have a whole lot of training data (only 3060 examples), it is probably useful to apply some data augmentation to the training images before feeding them to the model. This artificially increases the size of the training data somewhat.

Create a `Sequential` model that applies the following transformations to the images:

- each image is randomly flipped horizontally (or not)
- each image is randomly rotated by a random angle that is between -20 and $+20$ degrees. Pixels that are not part of the original image are filled with the value 0.
  - Note: you will need to determine the appropriate value for the `factor` parameter.
- a random zoom is applied to each image. Each image is randomly zoomed in or out by at most 15%. Pixels that are not part of the original image are filled with the value 0.

Make sure that the augmented images look reasonable by applying up to 9 augmentations on the first couple of images of the validation set. (Note that it is fine if you have to "manually" change the index of the image you want to look at.) Take into account that

- the `Dataset`s have been batched
- the pixel values have been scaled to the range $[-1, 1]$. Scale them back to the range $[0, 1]$ before showing them with `plt.imshow`.

Step 5: Load the Base Model

Load the `MobileNetV2` model.

- Do not include the top layer as we want to classify 102 classes instead of the 1000 classes for which the model was originally trained.
- Make sure to load the model with the `imagenet` weights.
- Tell the model that we will use the images of size 224×224 with 3 channels.

Using the functional API, create a new model that - Starts with the data augmentation layer you created earlier. - Passes the augmented images through the `MobileNetV2` base model. - Applies global average pooling to the results of the base model. - Adds a dropout layer with a dropout rate of 0.4. - Finishes with a fully connected layer. - Use the correct number of units. - Use the correct activation function.

Pay attention to the following:

- All the weights in the base model should be frozen.
- You should set `training=False` when calling the base model. This will prevent the batch normalization layers in the base model from updating their moving average statistics for the mean and standard deviation.
- Your model should have 2388646 parameters, of which 130662 are trainable.

Step 6: Train the Top Layer of the Model

Compile the model using - the `Adam` optimizer with a learning rate of 0.001. - the correct loss function - the accuracy as the metric to track.

Fit the model, for 100 epochs. To prevent the model from overfitting too much, use an `EarlyStopping` callback with a patience of 3. Track the validation accuracy. Only consider improvements of at least 0.1% to be worthwhile.

Step 7: Fine-tune the Model

Write code to find and print the different names of the layers in the base model. You should find that the base model consists of 16 "blocks". Assume that we want to fine-tune the last quarter of the blocks in the model, i.e. from block 13 onwards.

- Find the index of `block_13_expand` in the list of layer names.
- Set the `trainable` attribute of the `base_model` to `True`.
- Set the `trainable` attribute of all layers before `block_13_expand` to `False`.

Compile the model once more but this time using - the `Adam` optimizer with a learning rate of 0.00001.

You should find that your model now has 1812006 trainable parameters.

Train the model as before. (You may see only a very slight improvement in the validation accuracy.)

Step 8: Evaluate the Model

Evaluate the model on the test set. (Can you get an accuracy of more than 90%?.)

Step 9: Calculate Top-1, Top-3 and Top-5 Accuracy

Competitions like ImageNet are often scored using something like the top-5 (or top-3) accuracy.

Use the `tf.keras.metrics.SparseTopKCategoricalAccuracy` class to determine the top-1, top-3 and top-5 accuracy on the test set. (Results of 90.6%, 97.1% and 98.6% respectively are achievable.)

You will need to recompile the model (and track the appropriate metrics) prior to calling `evaluate` once more.