Ansheng Li
CS6384

Project 2 Documentation

Libraries:

- Tensorflow: 2.8.0
- Numpy: 1.22.2
- Pandas: 1.4.2

The pre-trained model uses feature extraction and fine-tuning techniques. Feature extraction uses the layers of the previous network to extract meaningful features from the new samples. Fine-tuning allows us to freeze up to a certain top layer and jointly train both the newly-added classifier layers and the last layers of the base models. This allowed us to achieve a high accuracy rate after fitting the model.

```
train_dataset = tf.keras.utils.image_dataset_from_directory(images_path,
                                                            shuffle=True,
                                                            batch_size=BATCH_SIZE,
                                                            image_size=IMG_SIZE,
                                                            validation_split=0.2,
                                                            subset='training',
                                                            label_mode='categorical',
                                                            seed=69
                                                            )
```

```
validation_dataset = tf.keras.utils.image_dataset_from_directory(images_path,
                                                                 shuffle=True,
                                                                 batch_size=BATCH_SIZE,
                                                                 image_size=IMG_SIZE,
                                                                 validation_split=0.2,
                                                                 subset='validation',
                                                                 label_mode='categorical',
                                                                 seed=69)
```

Image size is the recommended default (224, 224) and batch size is the default of 32. I used categorical attributes to split my training dataset and validation dataset. The split is 80/20.

```
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.2),
])
```

Although the flower dataset is rather large, containing over 12,000 images, I used the recommended data augmentation to artificially introduce sample diversity by applying random flip and mild rotation.

```
preprocess_input = tf.keras.applications.mobilenet_v3.preprocess_input
```

```
IMG_SHAPE = (IMG_SIZE + (3,))
base_model = tf.keras.applications.MobileNetV3Large(input_shape=IMG_SHAPE,
                                                    include_top=False,
                                                    weights='imagenet')
```

I decided upon using MobileNetV3 Large as my backbone since it is the newest version of MobileNet. The upside of MobileNetV3 Large is that it is 3.2% more accurate on Imagenet classification and detection is 25% faster at roughly the same accuracy.

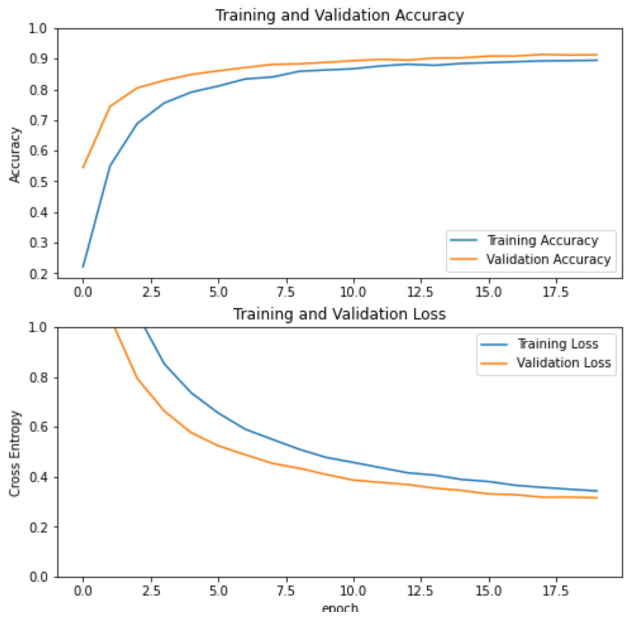
```
prediction_layer = tf.keras.layers.Dense(13, activation=tf.keras.activations.softmax)
prediction_batch = prediction_layer(feature_batch_average)
```

The activation function used is SoftMax since we have a multi-class classification problem. The outputs are mutually exclusive so SoftMax is the recommended activation function.

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.CategoricalCrossentropy(),
              metrics=['accuracy'])
```

I chose categorical cross entropy since we are doing a multi-class classification task, and the model must decide one of the 13 classification.

After fitting the model with epochs of 20, there seems to be no signs of overfitting as training accuracy loss decreased and validation loss decreased as seen by the model below. However, the accuracy seems to stall around the 15th iteration. A method that could have been implemented to determine overfitting is to have early callbacks, when validation rises which seems to occur towards the end of the 20 epochs. This would save time.



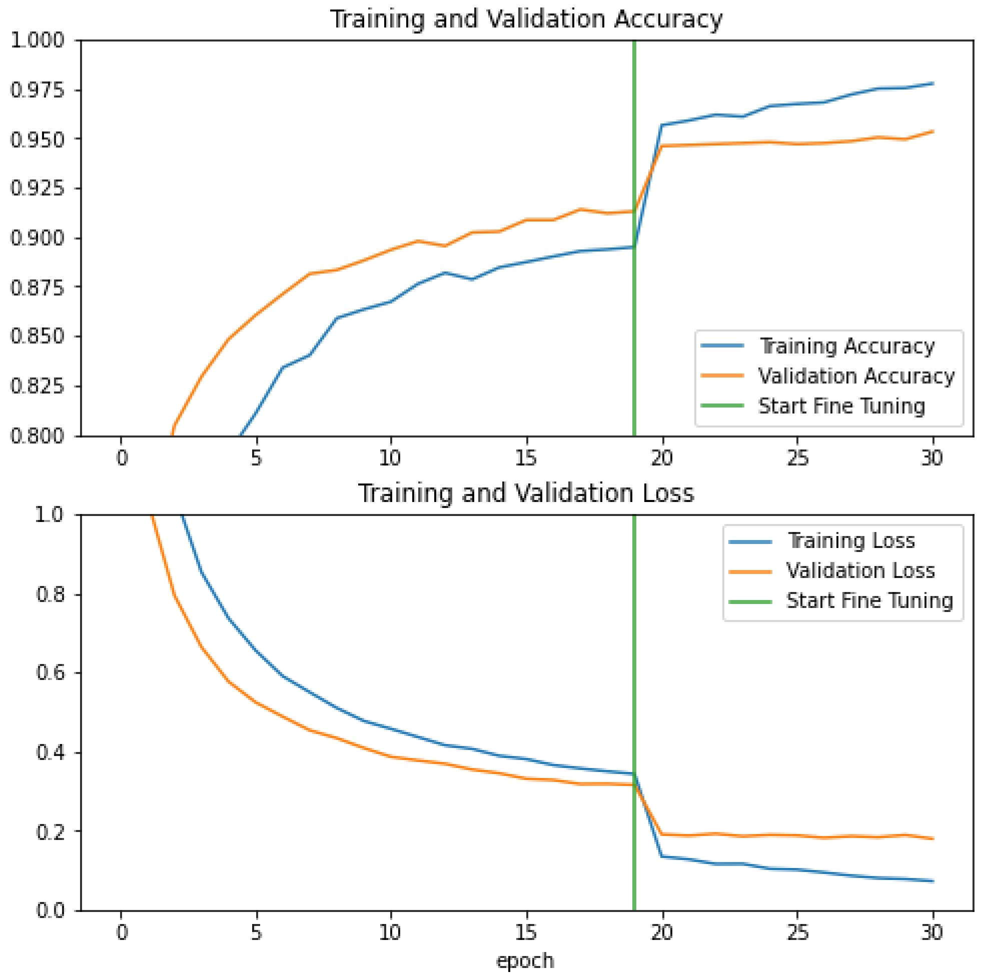
```
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 180

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable = False
```

```
Number of layers in the base model:  263
```

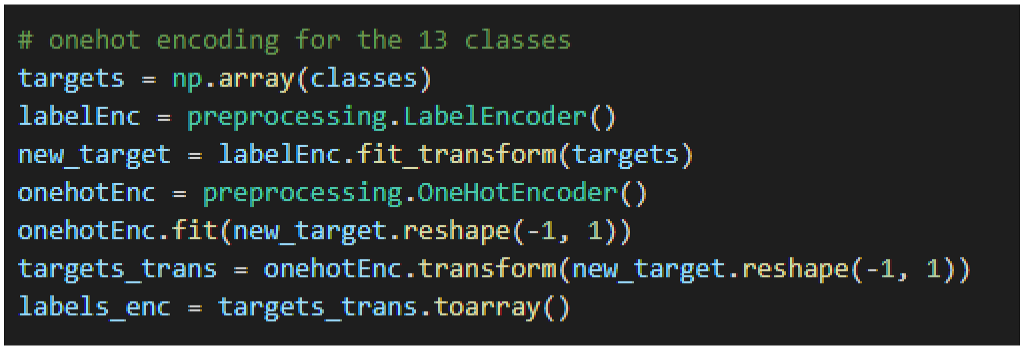
In feature extraction, we only trained a few layers on top of the MobileNetV3 Large base model. The weights of the network were not updated during training since we freeze the layers before fitting. To increase performance, we train the weights of the top layers alongside the training of the classier we added. This will force the weights to be tuned from generic to associated features with the given flowers dataset.



The figures above show the fine-tuning accuracy. There seems to be some overfitting as the validation accuracy loss seems to fluctuate while the train accuracy loss increases slightly. We can see the training accuracy reaching 97 while validation accuracy is at 95.

The first model I used was MobileNetV2 which produced a similar training accuracy of 95. The first model had a 75% accuracy against my custom test cases while MobileNetV3Large had 83%. The extra layers in MobileNetV3Large and bigger kernel size for the convolution helped with new test images.

For the proj2_test, the test labels are converted to the 13 classes one-hot encoding since we have a dense layer of 13. We can then map each label with the correct encoding and evaluate against my model. The preprocessing from the library sklern helps us convert the array of strings into integers. The onehotEnc creates the encoded object and fits the original feature. The transform function converts the original feature to the new feature. After the labels are encoded, we map the test labels in the csv file by assigning each test label to the right encoding that is stored in labels_enc. We can then use the images and the new test labels in my model.evaluate function.



```
# onehot encoding for the 13 classes
targets = np.array(classes)
labelEnc = preprocessing.LabelEncoder()
new_target = labelEnc.fit_transform(targets)
onehotEnc = preprocessing.OneHotEncoder()
onehotEnc.fit(new_target.reshape(-1, 1))
targets_trans = onehotEnc.transform(new_target.reshape(-1, 1))
labels_enc = targets_trans.toarray()
```