ch-en-u4aie20003-lab6

March 5, 2023

1 NAME- BACHU GANESH

2 ROLL NUMBER-CH.EN.U4AIE20003

Data pipeline

Baseline model: train a simple CNN from scratch Transfer learning: pretraiend ConvNet as a feature extractor Transfer learning: fine-tune a pretrained ConvNet Test accuracy & visualize predictions

[1]:
```
# Enable TensorFlow 2.0
#%tensorflow_version 2.x
```

```
Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
```

[2]:
```
!pip install tensorflow-gpu==2.11
# Import Tensorflow
import tensorflow as tf
tf.__version__
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting tensorflow-gpu==2.11
  Downloading
tensorflow_gpu-2.11.0-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(588.3 MB)
                           588.3/588.3

MB 2.9 MB/s eta 0:00:00
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.8/dist-packages (from tensorflow-gpu==2.11) (0.2.0)
Requirement already satisfied: protobuf<3.20,>=3.9.2 in
/usr/local/lib/python3.8/dist-packages (from tensorflow-gpu==2.11) (3.19.6)
Requirement already satisfied: flatbuffers>=2.0 in
/usr/local/lib/python3.8/dist-packages (from tensorflow-gpu==2.11) (23.1.21)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.8/dist-packages (from tensorflow-gpu==2.11) (15.0.6.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.8/dist-packages (from tensorflow-gpu==2.11) (0.31.0)
```

```
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.8/dist-
packages (from tensorflow-gpu==2.11) (3.1.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.8/dist-
packages (from tensorflow-gpu==2.11) (1.15.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.8/dist-packages (from tensorflow-gpu==2.11) (2.2.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.8/dist-
packages (from tensorflow-gpu==2.11) (57.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.8/dist-packages (from tensorflow-gpu==2.11) (1.6.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-
packages (from tensorflow-gpu==2.11) (23.0)
Requirement already satisfied: tensorflow-estimator<2.12,>=2.11.0 in
/usr/local/lib/python3.8/dist-packages (from tensorflow-gpu==2.11) (2.11.0)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.8/dist-packages (from tensorflow-gpu==2.11) (3.3.0)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.8/dist-
packages (from tensorflow-gpu==2.11) (1.22.4)
Requirement already satisfied: tensorboard<2.12,>=2.11 in
/usr/local/lib/python3.8/dist-packages (from tensorflow-gpu==2.11) (2.11.2)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.8/dist-packages (from tensorflow-gpu==2.11) (4.5.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.8/dist-packages (from tensorflow-gpu==2.11) (1.51.3)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in
/usr/local/lib/python3.8/dist-packages (from tensorflow-gpu==2.11) (0.4.0)
Requirement already satisfied: keras<2.12,>=2.11.0 in
/usr/local/lib/python3.8/dist-packages (from tensorflow-gpu==2.11) (2.11.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.8/dist-
packages (from tensorflow-gpu==2.11) (1.4.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.8/dist-
packages (from tensorflow-gpu==2.11) (1.15.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.8/dist-packages (from astunparse>=1.6.0->tensorflow-
gpu==2.11) (0.38.4)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in
/usr/local/lib/python3.8/dist-packages (from
tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (0.4.6)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.8/dist-packages (from
tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (2.16.1)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.8/dist-packages (from
tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (2.25.1)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in
/usr/local/lib/python3.8/dist-packages (from
tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (1.8.1)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.8/dist-
```

```
packages (from tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (2.2.3)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.8/dist-
packages (from tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (3.4.1)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in
/usr/local/lib/python3.8/dist-packages (from
tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (0.6.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.8/dist-packages (from google-
auth<3,>=1.6.3->tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (5.3.0)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.8/dist-packages (from google-
auth<3,>=1.6.3->tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (0.2.8)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.8/dist-
packages (from google-auth<3,>=1.6.3->tensorboard<2.12,>=2.11->tensorflow-
gpu==2.11) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.8/dist-packages (from google-auth-
oauthlib<0.5,>=0.4.1->tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (1.3.1)
Requirement already satisfied: importlib-metadata>=4.4 in
/usr/local/lib/python3.8/dist-packages (from
markdown>=2.6.8->tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (6.0.0)
Requirement already satisfied: chardet<5,>=3.0.2 in
/usr/local/lib/python3.8/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (4.0.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-
packages (from requests<3,>=2.21.0->tensorboard<2.12,>=2.11->tensorflow-
gpu==2.11) (2.10)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/usr/local/lib/python3.8/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (1.26.14)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.8/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (2022.12.7)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.8/dist-packages (from
werkzeug>=1.0.1->tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (2.1.2)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.8/dist-
packages (from importlib-
metadata>=4.4->markdown>=2.6.8->tensorboard<2.12,>=2.11->tensorflow-gpu==2.11)
(3.15.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in
/usr/local/lib/python3.8/dist-packages (from pyasn1-modules>=0.2.1->google-
auth<3,>=1.6.3->tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.8/dist-
packages (from requests-oauthlib>=0.7.0->google-auth-
oauthlib<0.5,>=0.4.1->tensorboard<2.12,>=2.11->tensorflow-gpu==2.11) (3.2.2)
Installing collected packages: tensorflow-gpu
Successfully installed tensorflow-gpu-2.11.0
```

[2]: '2.11.0'

[3]:
```
# Import TensorFlow datasets
import tensorflow_datasets as tfds
tfds.disable_progress_bar()

# Import Keras
from tensorflow import keras
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout,␣
↪MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam

# Import Numpy
import numpy as np
import matplotlib.pyplot as plt
```

3 Data pipeline

Load the tf_flowers dataset tf_flowers is one of the TensorFlow 2.0 datasets with 3670 samples.

[4]:
```
# Load train and validation datasets
(raw_train, raw_validation, raw_test), metadata = tfds.load(
    name='tf_flowers',
    split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'], # train,␣
↪validation, test split of 8:1:1
    with_info=True,
    as_supervised=True)
```

```
Downloading and preparing dataset Unknown size (download: Unknown size,
generated: Unknown size, total: Unknown size) to
/root/tensorflow_datasets/tf_flowers/3.0.1…
Dataset tf_flowers downloaded and prepared to
/root/tensorflow_datasets/tf_flowers/3.0.1. Subsequent calls will reuse this
data.
```

[5]:
```
print("Total number of samples:",  metadata.splits['train'].num_examples)
```

```
Total number of samples: 3670
```

[6]:
```
num_classes = metadata.features['label'].num_classes
num_train =  len(list(raw_train))
num_validation = len(list(raw_validation))
num_test = len(list(raw_test))

print("Number of classes:", num_classes)
```

```
print("Number of training samples:", num_train)
print("Number of validation samples:",  num_validation)
print("Number of test samples:", num_test)
```

```
Number of classes: 5
Number of training samples: 2936
Number of validation samples: 367
Number of test samples: 367
```

[7]:
```
# Inspect datasets before data preprocessing
print(raw_train)
print(raw_validation)
print(raw_test)
```

```
<PrefetchDataset element_spec=(TensorSpec(shape=(None, None, 3), dtype=tf.uint8,
name=None), TensorSpec(shape=(), dtype=tf.int64, name=None))>
<PrefetchDataset element_spec=(TensorSpec(shape=(None, None, 3), dtype=tf.uint8,
name=None), TensorSpec(shape=(), dtype=tf.int64, name=None))>
<PrefetchDataset element_spec=(TensorSpec(shape=(None, None, 3), dtype=tf.uint8,
name=None), TensorSpec(shape=(), dtype=tf.int64, name=None))>
```

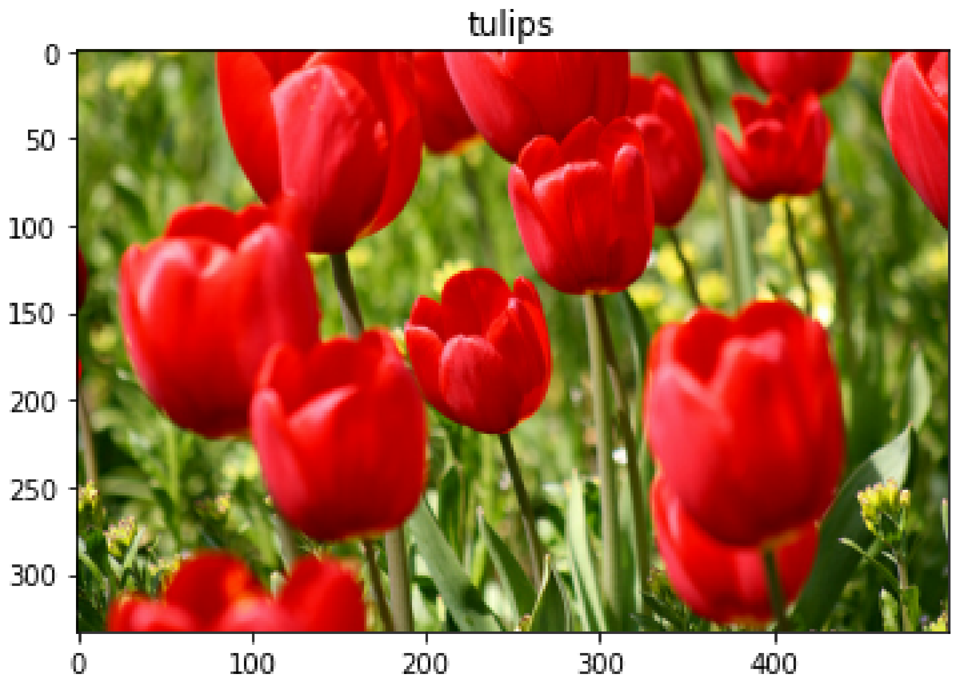
[8]:
```
# Get labels / class names
class_names = np.array(metadata.features['label'].names)
print(class_names)
```

```
['dandelion' 'daisy' 'tulips' 'sunflowers' 'roses']
```

4 Visualize the data

Let's take a look a few of the flower images

[9]:
```
label_names = metadata.features['label'].int2str

for image, label in raw_train.take(2):
  plt.figure()
  plt.imshow(image)
  plt.title(label_names(label))
```


tulips
0
50
100
150
200
250
300
0
100
200
300
400


sunflowers
0
25
50
75
100
125
150
175
200
0
50
100
150
200
250
300

5 Image preprocessing

Resize, normalize, augment, shuffle and batch the data.

Resize and normalize dataset

[10]:
```
IMG_SIZE = 224
IMG_SHAPE = (IMG_SIZE, IMG_SIZE, 3)

def format_example(image, label):
  image = tf.cast(image, tf.float32)
  image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
  image = image/255.0
  return image, label
```

[11]:
```
train = raw_train.map(format_example)
validation = raw_validation.map(format_example)
test = raw_test.map(format_example)
```

[12]:
```
def augment_data(image, label):
  image = tf.image.random_flip_left_right(image)
  image = tf.image.random_contrast(image, lower=0.0, upper=1.0)
  image = tf.stack(image, axis=0)
  image = tf.image.random_crop(image, size=[IMG_SIZE, IMG_SIZE, 3])
  return image, label
```

[13]:
```
train = train.map(augment_data)
```

Shuffle and batch dataset

[14]:
```
BATCH_SIZE = 32
SHUFFLE_BUFFER_SIZE = 1000

train_batches = train.shuffle(SHUFFLE_BUFFER_SIZE).batch(BATCH_SIZE).repeat()
validation_batches = validation.batch(BATCH_SIZE).repeat()
test_batches = test.batch(BATCH_SIZE)
```

[15]:
```
# Inspect datasets after data preprocessing
print(train_batches)
print(validation_batches)
print(test_batches)
```

```
<RepeatDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3),
dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int64,
name=None))>
<RepeatDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3),
dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int64,
name=None))>
<BatchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3),
```

dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int64, name=None))>

[16]:
```
# Inspect a batch of data
for image_batch, label_batch in train_batches.take(1):
    pass

image_batch.shape
```

[16]: TensorShape([32, 224, 224, 3])

6 Training

[17]:
```
# Set training parameters
NUM_EPOCHS = 10
steps_per_epoch = round(num_train)//BATCH_SIZE
validation_steps = round(num_validation)//BATCH_SIZE
```

[18]:
```
# Display training curves
def display_training_curves(history, title):
  acc = history.history['accuracy']
  val_acc = history.history['val_accuracy']

  loss = history.history['loss']
  val_loss = history.history['val_loss']

  epochs_range = range(NUM_EPOCHS)

  plt.plot(epochs_range, acc, label='Train accuracy')
  plt.plot(epochs_range, val_acc, label='Val accuracy')
  plt.title(title)
  plt.legend(loc='upper left')
  plt.figure()

  plt.show()
```

7 Baseline - train from scratch

Train a very simple CNN model and use the accuracy metrics as baseline to compare with transfer learning results.

Create model

[19]:
```
def build_model_from_scratch():

  model = Sequential([
```

```
    # Must define the input shape in the first layer of the neural network
    Conv2D(filters=32, kernel_size=3, padding='same', activation='relu',␣
↪input_shape=IMG_SHAPE),
    MaxPooling2D(pool_size=2),

    Conv2D(filters=64, kernel_size=3, padding='same', activation='relu'),
    MaxPooling2D(pool_size=2),

    Flatten(),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')
  ])

  return model
```

[20]:
```
simple_cnn_model = build_model_from_scratch()
```

[21]:
```
simple_cnn_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 32)      896

 max_pooling2d (MaxPooling2D  (None, 112, 112, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 112, 112, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 56, 56, 64)       0
 2D)

 flatten (Flatten)           (None, 200704)            0

 dense (Dense)               (None, 64)                12845120

 dense_1 (Dense)             (None, 5)                 325

=================================================================
Total params: 12,864,837
Trainable params: 12,864,837
Non-trainable params: 0
_________________________________________________________________
```

Compile and train the model

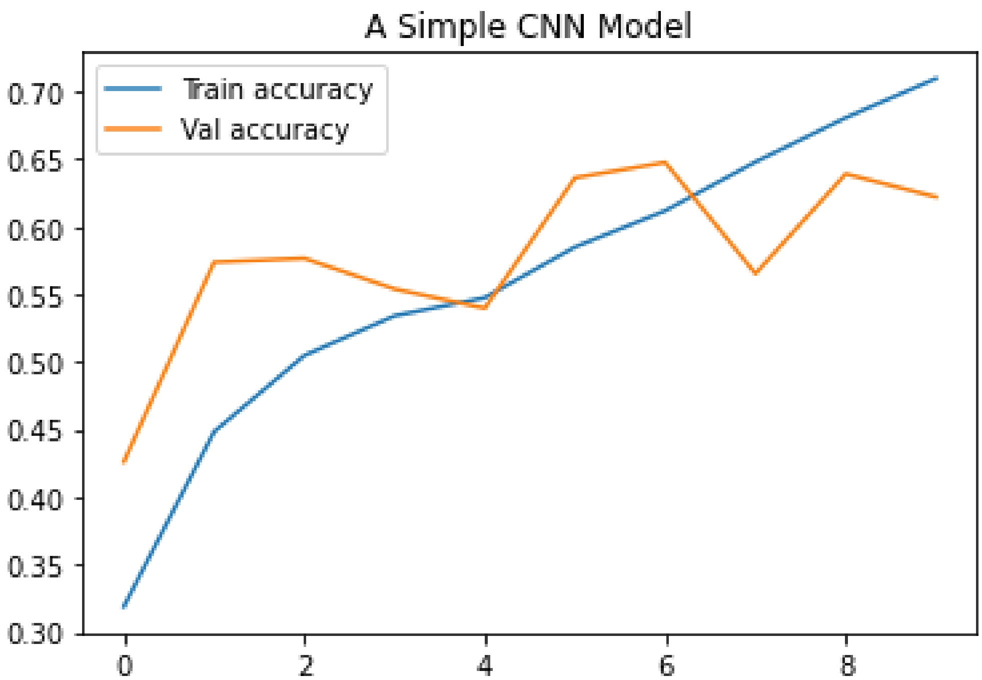
[22]:
```
def train_model(model):
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    history = model.fit(train_batches,
              epochs=NUM_EPOCHS,
              validation_data=validation_batches,
              steps_per_epoch=steps_per_epoch,
              validation_steps=validation_steps)

    return history
```

[23]:
```
%%time
history = train_model(simple_cnn_model)
```

```
Epoch 1/10
91/91 [==============================] - 16s 58ms/step - loss: 1.9384 -
accuracy: 0.3190 - val_loss: 1.3401 - val_accuracy: 0.4261
Epoch 2/10
91/91 [==============================] - 7s 70ms/step - loss: 1.2938 - accuracy:
0.4483 - val_loss: 1.1256 - val_accuracy: 0.5739
Epoch 3/10
91/91 [==============================] - 5s 56ms/step - loss: 1.2086 - accuracy:
0.5048 - val_loss: 1.0393 - val_accuracy: 0.5767
Epoch 4/10
91/91 [==============================] - 6s 65ms/step - loss: 1.1559 - accuracy:
0.5341 - val_loss: 1.1216 - val_accuracy: 0.5540
Epoch 5/10
91/91 [==============================] - 6s 66ms/step - loss: 1.1041 - accuracy:
0.5475 - val_loss: 1.2216 - val_accuracy: 0.5398
Epoch 6/10
91/91 [==============================] - 5s 53ms/step - loss: 1.0434 - accuracy:
0.5851 - val_loss: 1.0281 - val_accuracy: 0.6364
Epoch 7/10
91/91 [==============================] - 5s 58ms/step - loss: 0.9753 - accuracy:
0.6119 - val_loss: 0.9794 - val_accuracy: 0.6477
Epoch 8/10
91/91 [==============================] - 5s 59ms/step - loss: 0.9097 - accuracy:
0.6481 - val_loss: 1.3184 - val_accuracy: 0.5653
Epoch 9/10
91/91 [==============================] - 5s 52ms/step - loss: 0.8222 - accuracy:
0.6808 - val_loss: 1.1413 - val_accuracy: 0.6392
Epoch 10/10
91/91 [==============================] - 6s 64ms/step - loss: 0.7634 - accuracy:
0.7097 - val_loss: 1.2268 - val_accuracy: 0.6222
CPU times: user 1min 23s, sys: 5.52 s, total: 1min 28s
```

Wall time: 1min 5s

[24]:
```
# Display training curve
display_training_curves(history, "A Simple CNN Model")
```




```
<Figure size 432x288 with 0 Axes>
```

8 Transfer learning

Now let's see how transfer learning can help achieve better results.

Feature extractor Use MobileNetV2 as a feature extractor and add a classifier on top of it.

Create base model

[25]:
```
# Create base model from tf.keras pre-trained model MobileNetV2
base_model = MobileNetV2(input_shape=IMG_SHAPE,
                         weights="imagenet",
                         include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
ns/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h
5
9406464/9406464 [==============================] - 0s 0us/step
```

Freeze all layers of the base model

[26]:
```
base_model.trainable = False
```

Add a classifier head

Create a new model by adding a classifier on top of the base model.

[27]:
```
def build_mobilenetv2_model(base_model):
  model = Sequential([
      base_model,
      Conv2D(32, 3, activation='relu'),
      GlobalAveragePooling2D(),
      Dense(num_classes, activation='softmax')]
  )

  return model
```

[28]:
```
model = build_mobilenetv2_model(base_model)
```

Compile the model

[29]:
```
model.compile(optimizer= Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

[30]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 conv2d_2 (Conv2D)           (None, 5, 5, 32)          368672

 global_average_pooling2d (G  (None, 32)               0
 lobalAveragePooling2D)

 dense_2 (Dense)             (None, 5)                 165

=================================================================
Total params: 2,626,821
Trainable params: 368,837
Non-trainable params: 2,257,984
_________________________________________________________________
```

Train the model

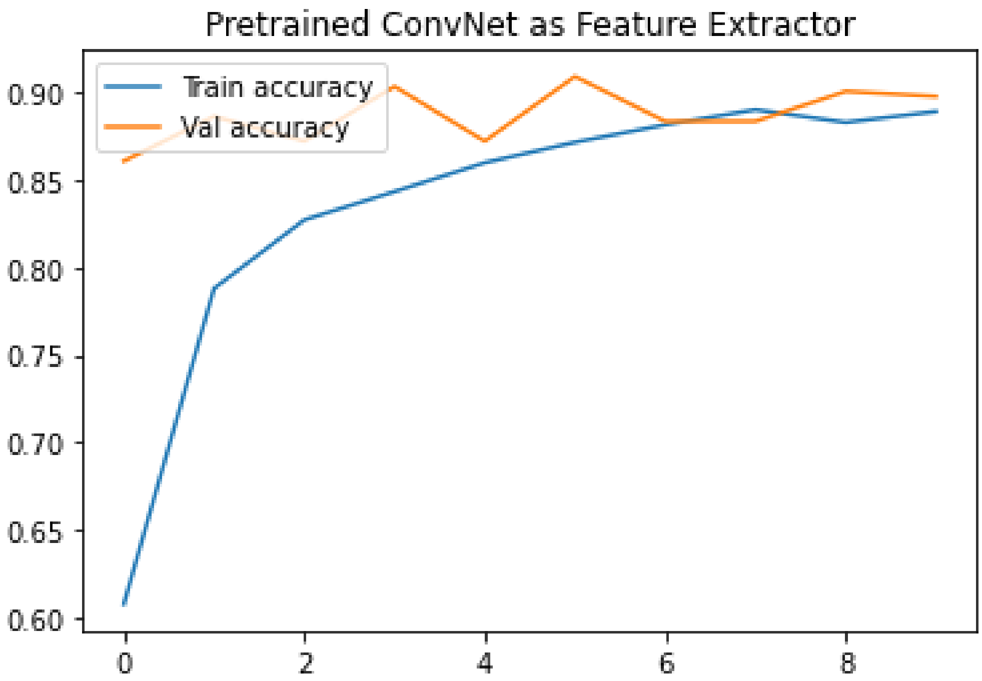
[31]:

```
%%time
history = model.fit(train_batches,
                    epochs=NUM_EPOCHS,
                    validation_data=validation_batches,
                    steps_per_epoch=steps_per_epoch,
                    validation_steps=validation_steps)
```

```
Epoch 1/10
91/91 [==============================] - 13s 86ms/step - loss: 0.9577 -
accuracy: 0.6075 - val_loss: 0.4419 - val_accuracy: 0.8608
Epoch 2/10
91/91 [==============================] - 5s 53ms/step - loss: 0.5474 - accuracy:
0.7879 - val_loss: 0.3968 - val_accuracy: 0.8864
Epoch 3/10
91/91 [==============================] - 6s 69ms/step - loss: 0.4731 - accuracy:
0.8271 - val_loss: 0.4475 - val_accuracy: 0.8722
Epoch 4/10
91/91 [==============================] - 6s 69ms/step - loss: 0.4196 - accuracy:
0.8433 - val_loss: 0.3957 - val_accuracy: 0.9034
Epoch 5/10
91/91 [==============================] - 6s 63ms/step - loss: 0.3663 - accuracy:
0.8598 - val_loss: 0.4629 - val_accuracy: 0.8722
Epoch 6/10
91/91 [==============================] - 5s 58ms/step - loss: 0.3311 - accuracy:
0.8716 - val_loss: 0.4141 - val_accuracy: 0.9091
Epoch 7/10
91/91 [==============================] - 5s 55ms/step - loss: 0.3174 - accuracy:
0.8815 - val_loss: 0.3956 - val_accuracy: 0.8835
Epoch 8/10
91/91 [==============================] - 5s 54ms/step - loss: 0.2944 - accuracy:
0.8902 - val_loss: 0.4223 - val_accuracy: 0.8835
Epoch 9/10
91/91 [==============================] - 6s 67ms/step - loss: 0.2925 - accuracy:
0.8829 - val_loss: 0.3939 - val_accuracy: 0.9006
Epoch 10/10
91/91 [==============================] - 5s 52ms/step - loss: 0.2910 - accuracy:
0.8891 - val_loss: 0.4393 - val_accuracy: 0.8977
CPU times: user 1min 24s, sys: 5.66 s, total: 1min 29s
Wall time: 1min 2s
```

[32]:

```
# Display training curve
display_training_curves(history, "Pretrained ConvNet as Feature Extractor")
```




```
<Figure size 432x288 with 0 Axes>
```

9 Fine tuning

Unfreeze top layers for fine tuning

[33]:
```
# Unfreeze all layers in base model
base_model.trainable = True
```

[34]:
```
# 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))
```

```
Number of layers in the base model:  154
```

[35]:
```
# Fine-tune from this layer onwards
fine_tune_at = 100

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

Compile the model

[36]:
```
model.compile(loss='sparse_categorical_crossentropy',
              optimizer = Adam(1e-5),
```

```
                metrics=['accuracy'])
```

[37]:
```
model.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 conv2d_2 (Conv2D)           (None, 5, 5, 32)          368672

 global_average_pooling2d (G  (None, 32)               0
 lobalAveragePooling2D)

 dense_2 (Dense)             (None, 5)                 165

=================================================================
Total params: 2,626,821
Trainable params: 2,230,277
Non-trainable params: 396,544
_________________________________________________________________
```

Continue to train the model

[38]:
```
INITIAL_EPOCHS = 10
FINE_TUNE_EPOCHS = 10
TOTAL_EPOCHS = INITIAL_EPOCHS + FINE_TUNE_EPOCHS #20
```

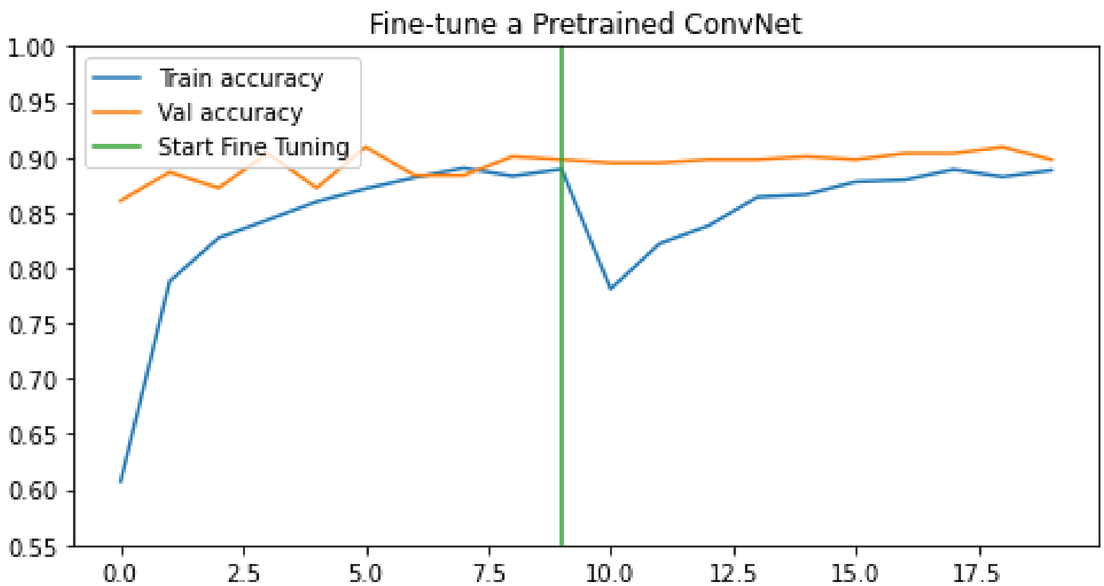
[39]:
```
%%time
history_fine = model.fit(train_batches,
                    epochs=TOTAL_EPOCHS,           #20
                    initial_epoch=INITIAL_EPOCHS, #10
                    validation_data=validation_batches,
                    steps_per_epoch=steps_per_epoch,
                    validation_steps=validation_steps)
```

```
Epoch 11/20
91/91 [==============================] - 24s 102ms/step - loss: 0.5985 -
accuracy: 0.7809 - val_loss: 0.4390 - val_accuracy: 0.8949
Epoch 12/20
91/91 [==============================] - 6s 67ms/step - loss: 0.5067 - accuracy:
0.8220 - val_loss: 0.4325 - val_accuracy: 0.8949
Epoch 13/20
91/91 [==============================] - 8s 85ms/step - loss: 0.4269 - accuracy:
0.8382 - val_loss: 0.4255 - val_accuracy: 0.8977
Epoch 14/20
```

```
91/91 [==============================] - 7s 77ms/step - loss: 0.3753 - accuracy:
0.8640 - val_loss: 0.4247 - val_accuracy: 0.8977
Epoch 15/20
91/91 [==============================] - 8s 83ms/step - loss: 0.3500 - accuracy:
0.8664 - val_loss: 0.4118 - val_accuracy: 0.9006
Epoch 16/20
91/91 [==============================] - 7s 72ms/step - loss: 0.3290 - accuracy:
0.8778 - val_loss: 0.4066 - val_accuracy: 0.8977
Epoch 17/20
91/91 [==============================] - 7s 79ms/step - loss: 0.3220 - accuracy:
0.8795 - val_loss: 0.3955 - val_accuracy: 0.9034
Epoch 18/20
91/91 [==============================] - 7s 74ms/step - loss: 0.3008 - accuracy:
0.8888 - val_loss: 0.3937 - val_accuracy: 0.9034
Epoch 19/20
91/91 [==============================] - 8s 84ms/step - loss: 0.3031 - accuracy:
0.8822 - val_loss: 0.3839 - val_accuracy: 0.9091
Epoch 20/20
91/91 [==============================] - 8s 85ms/step - loss: 0.2931 - accuracy:
0.8881 - val_loss: 0.4000 - val_accuracy: 0.8977
CPU times: user 1min 45s, sys: 4.89 s, total: 1min 50s
Wall time: 1min 27s
```

[40]:

```
# Display training curve

acc = history.history['accuracy'] + history_fine.history['accuracy']
val_acc = history.history['val_accuracy'] + history_fine.history['val_accuracy']

plt.figure(figsize=(8, 4))

plt.plot(acc, label='Train accuracy')
plt.plot(val_acc, label='Val accuracy')
plt.ylim([0.8, 1])
plt.plot([NUM_EPOCHS-1, NUM_EPOCHS-1], plt.ylim(ymin=0.55), label='Start Fine␣
↪Tuning')
plt.title("Fine-tune a Pretrained ConvNet")
plt.legend(loc='upper left')

plt.show()
```



10 Test accuracy

[41]:
```
# Evaluate the model on the test dataset
score = model.evaluate(test_batches, verbose=0)

# Print test accuracy
print('\n', 'Test accuracy:', score[1])
```

```
 Test accuracy: 0.9046321511268616
```

11 Visualize predictions

First we get images and labels from a test batch, and then use the retrained model to make predictions.

[42]:
```
image_batch, label_batch = next(iter(test_batches))

image_batch = image_batch.numpy()
label_batch = label_batch.numpy()

predicted_batch = model.predict(image_batch)
predicted_batch = tf.squeeze(predicted_batch).numpy()

predicted_class_ids = np.argmax(predicted_batch, axis=-1)
predicted_class_names = class_names[predicted_class_ids]
```

```
1/1 [==============================] - 1s 1s/step
```

Then we visualize some of the images and compare whether the predicted labels equal to the true labels.

[43]:
```
plt.figure(figsize=(8, 8))

# Display 16 test images with predictions
for i in range(16):
  plt.subplot(4, 4, i+1)
  # Display each image
  plt.imshow(image_batch[i])
  # Set title color: green if prediction correct and red if prediction incorrect
  title_color = "green" if predicted_class_ids[i] == label_batch[i] else "red"
  plt.title(predicted_class_names[i].title(), color=title_color)
  plt.axis('off')

_ = plt.suptitle("Model Predictions")
```


Model Predictions
Tulips
Dandelion
Dandelion
Tulips
Dandelion
Dandelion
Tulips
Daisy
Sunflowers
Dandelion
Dandelion
Dandelion
Sunflowers
Tulips
Dandelion
Sunflowers

```
[ ]:
```