PIC16B HW4: Image Classification

PIC16B HW

AUTHOR
Zhe Shen

PUBLISHED
February 27, 2023

PIC16B HW4: Image Classification

In this blog, I will introduce seversl new skills and concepts related to image classification in **Tensorflow**. - Tensorflow is a **Datasets** which provide a convenient way to organize operations on our training, validation, and test data sets. - Data augmentation is a technology which help us to create our datasets' expanded versions, and allow models to learn patterns more robustly. - Transfer learning is used to applicating our pre-trained models for new tasks.

Before our bolg, I highly recommand enabling **GPU** runtime(under Runtime -> Change Runtime Type) when we training our model. It will lead to significant speed benefits.

Part 1: Overview

In this tutorial, we set the goal of image classification to distinguish between images of cats and dogs. [The samples for the dataset] are collected from the Tensorflow team.

Part 2: Load Packages and Obtain Data

Here are the libraries we need for this tutorial. The star of the show is Tensorflow, an open source library for machine learning for a variety of perceptual and language understanding tasks.

```
import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import utils, layers, models
```

Then, we can copy and paste the following codes. These codes will enable us to download and do some processing on the dataset： - Download and extract the data - Construct path and rescaled images' size as 160x160 - Divided the dataset as training set, validation set, and test set

```python
# location of data
_URL = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'

# download the data and extract it
path_to_zip = utils.get_file('cats_and_dogs.zip', origin=_URL, extract=True)

# construct paths
PATH = os.path.join(os.path.dirname(path_to_zip), 'cats_and_dogs_filtered')

train_dir = os.path.join(PATH, 'train')
validation_dir = os.path.join(PATH, 'validation')

# parameters for datasets
BATCH_SIZE = 32
IMG_SIZE = (160, 160)

# construct train and validation datasets
train_dataset = utils.image_dataset_from_directory(train_dir,
                                                   shuffle=True,
                                                   batch_size=BATCH_SIZE,
                                                   image_size=IMG_SIZE)

validation_dataset = utils.image_dataset_from_directory(validation_dir,
                                                        shuffle=True,
                                                        batch_size=BATCH_SIZE,
                                                        image_size=IMG_SIZE)

# construct the test dataset by taking every 5th observation out of the validation dataset
val_batches = tf.data.experimental.cardinality(validation_dataset)
test_dataset = validation_dataset.take(val_batches // 5)
validation_dataset = validation_dataset.skip(val_batches // 5)

# The following section is an optimized adjustment of the training speed

test_dataset = test_dataset.prefetch(buffer_size=tf.data.AUTOTUNE)
train_dataset = train_dataset.prefetch(buffer_size=tf.data.AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=tf.data.AUTOTUNE)
```

```
Downloading data from https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
68606236/68606236 [==============================] - 4s 0us/step
Found 2000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

Part 3: Checking the data sets

To be a good data scientist, one should never forget to examine the dataset.(**“Garbage in, garbage out.”**)

Since this dataset is composed of images, we need to do some visualization work. You can copy the following code to extract 8 random images. 4 of them are images of dogs and the remaining 4 are images of cats.

```python
# Get 3 images of cats and dogs respectively.
```

```
def random3images():
  # In train set, dogs marked label 1 and cats marked label 0
  for values, labels in train_dataset.take(1):
    # So, we splite dogs and cats to get images for each
    dogset = values[labels==1]
    catset = values[labels==0]

  # Extract one image in turn.

    plt.figure(figsize=(16,8)) # Set output image size

  # Dogs' images show first.
  for i in range(3):
    plt.subplot(2,3,i+1)
    plt.imshow(tf.cast(dogset[i],tf.uint8))
    plt.title("DOG "+str(i+1))
    plt.axis('off') # Hide axis for looking neat.

  # Secondly is cats' images
  for i in range(3):
    plt.subplot(2,3,i+4)
    plt.imshow(tf.cast(catset[i],tf.uint8))
    plt.title("CAT "+str(i+1))
    plt.axis('off') # Hide axis for looking neat.

  plt.show()
random3images()
```

Then let's simply count the size of the data volume for cats and dogs as follows.

```
# Create an iterator called labels.
labels_iterator= train_dataset.unbatch().map(lambda image, label: label).as_numpy_iterator()
```

```
cats = 0
dogs = 0
for label in labels_iterator:
    if label == 1:
        dogs = dogs+ 1
    else:
        cats = cats+ 1
print("Number of dogs: "+str(dogs))
print("Number of cats: "+str(cats))
```

```
Number of dogs: 1000
Number of cats: 1000
```

From the above code we know that the number of cats and dogs is 1000 each. This means that if we use the dumbest way to guess all the data as cats (or dogs), we can get at least 50% accuracy. So in the next machine learning process, we should guarantee at least 50% accuracy.

Part 4: First Model and its Evaluation

We will first construct a **CNN** model. The structure of the model is: - 3 Convolution and Maxpooling layers - 1 Flatten layer - 2 Fully connected layers.

Since we need to put the 3 layers into **tensorflow.models.Sequential**, we set the same input size as we did at the part 2 **(160, 160, 3)**.

The final output Dsense layer contains 2 neurons, one for the cat and one for the dog.

```
model1 = models.Sequential([

    # 3 Convolution and Maxpooling layers
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(160, 160, 3)),layers.MaxPooling2
    layers.Conv2D(32, (3, 3), activation='relu'),layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),layers.MaxPooling2D((2, 2)),

    # One Flatten layer
    layers.Flatten(),

    # 2 Fully connected layers(Dense and Dropout)
    layers.Dense(1024, activation='relu'),layers.Dropout(0.2),
    layers.Dense(64, activation='relu'),layers.Dropout(0.2),layers.Dense(2)
    ])
```

Now, let’s evaluate how **model1** performs.

To get more reliable results, we set **epochs=20** here. This means that the model will perform 20 loops on the training set. **validation_dataset** will verify and evaluate the accuracy.

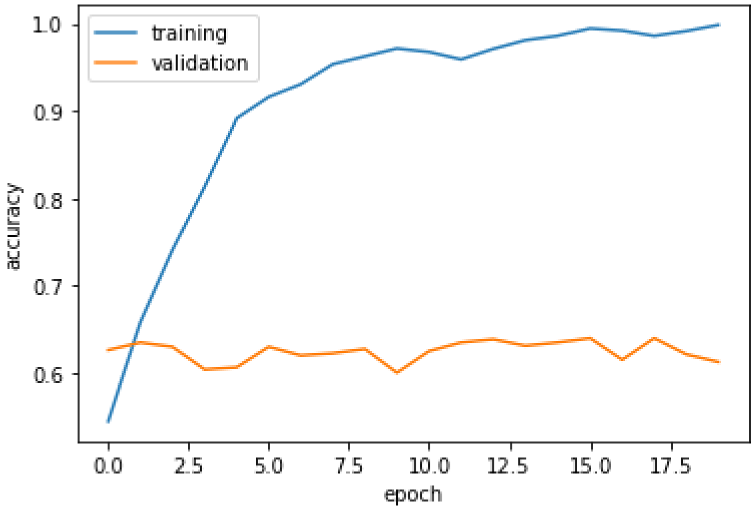
In addition, we will visualize the results to show accuracy.

```
# Compile model1
model1.compile(optimizer='adam',
              loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
```

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

# Train the model
history1 = model1.fit(train_dataset,
                      epochs = 20,
                      validation_data=validation_dataset)

# Plot the training history accuracy
plt.gca().set(xlabel = "epoch", ylabel = "accuracy")

# Statistical accuracy for the training and validation sets
plt.plot(history1.history["accuracy"], label = "training")
plt.plot(history1.history["val_accuracy"], label = "validation")
plt.legend()
plt.show()
```

```
Epoch 1/20
63/63 [==============================] - 26s 402ms/step - loss: 14.7487 - accuracy: 0.5450 -
val_loss: 0.6775 - val_accuracy: 0.6262
Epoch 2/20
63/63 [==============================] - 25s 395ms/step - loss: 0.6220 - accuracy: 0.6580 -
val_loss: 0.6747 - val_accuracy: 0.6349
Epoch 3/20
63/63 [==============================] - 25s 393ms/step - loss: 0.5063 - accuracy: 0.7410 -
val_loss: 0.7266 - val_accuracy: 0.6300
Epoch 4/20
63/63 [==============================] - 25s 394ms/step - loss: 0.3883 - accuracy: 0.8135 -
val_loss: 1.0656 - val_accuracy: 0.6040
Epoch 5/20
63/63 [==============================] - 25s 395ms/step - loss: 0.2592 - accuracy: 0.8915 -
val_loss: 1.4055 - val_accuracy: 0.6064
Epoch 6/20
63/63 [==============================] - 25s 395ms/step - loss: 0.2044 - accuracy: 0.9160 -
val_loss: 1.6607 - val_accuracy: 0.6300
Epoch 7/20
63/63 [==============================] - 25s 397ms/step - loss: 0.1910 - accuracy: 0.9305 -
val_loss: 1.4431 - val_accuracy: 0.6200
Epoch 8/20
63/63 [==============================] - 25s 395ms/step - loss: 0.1443 - accuracy: 0.9535 -
val_loss: 1.2551 - val_accuracy: 0.6225
Epoch 9/20
63/63 [==============================] - 25s 396ms/step - loss: 0.1321 - accuracy: 0.9625 -
val_loss: 1.6952 - val_accuracy: 0.6275
Epoch 10/20
63/63 [==============================] - 25s 398ms/step - loss: 0.0791 - accuracy: 0.9715 -
val_loss: 2.2905 - val_accuracy: 0.6002
Epoch 11/20
63/63 [==============================] - 26s 408ms/step - loss: 0.0906 - accuracy: 0.9675 -
val_loss: 1.5666 - val_accuracy: 0.6250
Epoch 12/20
63/63 [==============================] - 25s 401ms/step - loss: 0.1083 - accuracy: 0.9590 -
val_loss: 1.9527 - val_accuracy: 0.6349
Epoch 13/20
63/63 [==============================] - 25s 396ms/step - loss: 0.0823 - accuracy: 0.9710 -
```

```
val_loss: 1.7760 - val_accuracy: 0.6386
Epoch 14/20
63/63 [==============================] - 25s 396ms/step - loss: 0.0639 - accuracy: 0.9810 -
val_loss: 1.9893 - val_accuracy: 0.6312
Epoch 15/20
63/63 [==============================] - 25s 397ms/step - loss: 0.0415 - accuracy: 0.9860 -
val_loss: 2.0026 - val_accuracy: 0.6349
Epoch 16/20
63/63 [==============================] - 25s 397ms/step - loss: 0.0201 - accuracy: 0.9945 -
val_loss: 2.1845 - val_accuracy: 0.6399
Epoch 17/20
63/63 [==============================] - 25s 395ms/step - loss: 0.0270 - accuracy: 0.9920 -
val_loss: 2.3798 - val_accuracy: 0.6151
Epoch 18/20
63/63 [==============================] - 25s 396ms/step - loss: 0.0491 - accuracy: 0.9860 -
val_loss: 1.9512 - val_accuracy: 0.6399
Epoch 19/20
63/63 [==============================] - 25s 396ms/step - loss: 0.0215 - accuracy: 0.9915 -
val_loss: 2.6699 - val_accuracy: 0.6213
Epoch 20/20
63/63 [==============================] - 26s 408ms/step - loss: 0.0068 - accuracy: 0.9985 -
val_loss: 3.6838 - val_accuracy: 0.6126
```



We can observe that the accuracy of **the validation sets around 62%**. It is better compared to the baseline model. But the accuracy of the validation and training sets are extremely different. There must be an overfitting problem in this.

Part 5: Model with Data Augmentation

Based on the overfitting problem we have identified. We will try to use data augmentation to help our model learn so-called "invariant" features of our input images.

In short, we will flip a portion of the images in the dataset in the next step, allowing the machine learning model to focus more on the relatively "invariant" features in the images during training. In addition, these are two example of flipping and rotating under below:

```
# Choose one cat image and flip it
image = catset[1]
flip = tf.keras.Sequential([layers.RandomFlip()])
```

```python
flip_image = flip(image)

plt.figure(figsize=(10,16))

# Plot original image
plt.subplot(1,2,1)
plt.imshow(tf.cast(image, tf.uint8))
plt.title("Original")
plt.axis('off')

# Plot flipped image
plt.subplot(1,2,2)
plt.imshow(tf.cast(flip_image, tf.uint8))
plt.title("Flipped")
plt.axis('off')
plt.show()
```


```python
# Choose one dog image and rotate it
image = dogset[0]
rotate = tf.keras.Sequential([layers.RandomRotation(factor=0.25)])
rotated_image = rotate(image)

# Plot original image
plt.figure(figsize=(8,16))
plt.subplot(1,2,1)
plt.imshow(tf.cast(image, tf.uint8))
plt.title("Original")
plt.axis('off')

# Plot rotated image
plt.subplot(1,2,2)
plt.imshow(tf.cast(rotated_image, tf.uint8))
plt.title("Rotated")
plt.axis('off')

plt.show()
```

Now, let’s start building the second model. This will be highly based on our first model, but we’ll add some codes for flipping the image in it: - layers.RandomFlip() - layers.RandomRotation(factor=0.25)

The first line code will flip the image and the second line code set the flip angle is 90 degrees.

```
model2 = models.Sequential([
    # different part with model1
    layers.RandomFlip(),
    layers.RandomRotation(factor=0.25),

    # same with model1
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(160, 160, 3)),layers.MaxPooling2
    layers.Conv2D(32, (3, 3), activation='relu'),layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),layers.MaxPooling2D((2, 2)),

    layers.Flatten(),

    layers.Dense(1024, activation='relu'),layers.Dropout(0.2),
    layers.Dense(64, activation='relu'),layers.Dropout(0.2),
    layers.Dense(2)
    ])
tf.get_logger().setLevel('ERROR')
```

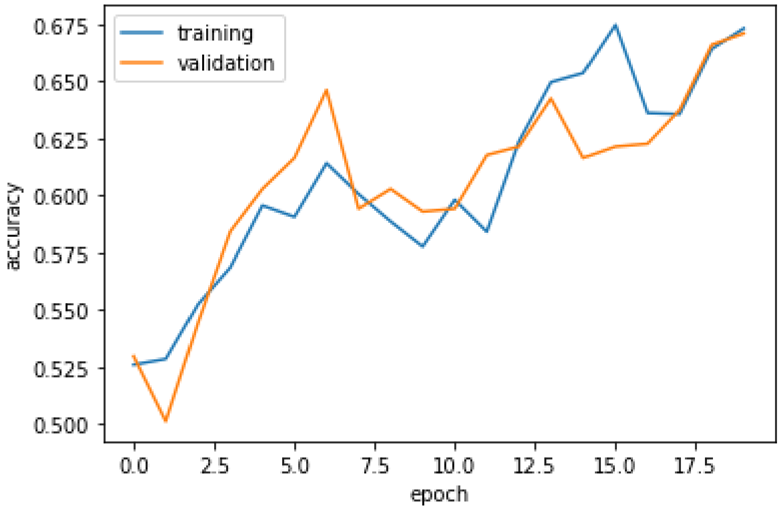
Let’s see how the model 2 performs next.

```
model2.compile(optimizer='adam',
            loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
            metrics = ['accuracy'])
history2 = model2.fit(train_dataset,
                    epochs=20,
                    validation_data=validation_dataset)
```

```
Epoch 1/20
63/63 [==============================] - 28s 420ms/step - loss: 9.2991 - accuracy: 0.5260 -
val_loss: 0.6895 - val_accuracy: 0.5297
Epoch 2/20
63/63 [==============================] - 26s 418ms/step - loss: 0.6921 - accuracy: 0.5285 -
val_loss: 0.6869 - val_accuracy: 0.5012
Epoch 3/20
```

```
63/63 [==============================] - 26s 419ms/step - loss: 0.6809 - accuracy: 0.5525 -
val_loss: 0.6863 - val_accuracy: 0.5446
Epoch 4/20
63/63 [==============================] - 27s 422ms/step - loss: 0.6834 - accuracy: 0.5685 -
val_loss: 0.6861 - val_accuracy: 0.5842
Epoch 5/20
63/63 [==============================] - 26s 418ms/step - loss: 0.6772 - accuracy: 0.5955 -
val_loss: 0.6576 - val_accuracy: 0.6027
Epoch 6/20
63/63 [==============================] - 26s 420ms/step - loss: 0.6714 - accuracy: 0.5905 -
val_loss: 0.6484 - val_accuracy: 0.6163
Epoch 7/20
63/63 [==============================] - 27s 420ms/step - loss: 0.6620 - accuracy: 0.6140 -
val_loss: 0.6628 - val_accuracy: 0.6460
Epoch 8/20
63/63 [==============================] - 26s 420ms/step - loss: 0.6831 - accuracy: 0.6005 -
val_loss: 0.6691 - val_accuracy: 0.5941
Epoch 9/20
63/63 [==============================] - 26s 419ms/step - loss: 0.6832 - accuracy: 0.5885 -
val_loss: 0.6630 - val_accuracy: 0.6027
Epoch 10/20
63/63 [==============================] - 27s 420ms/step - loss: 0.6733 - accuracy: 0.5775 -
val_loss: 0.6816 - val_accuracy: 0.5928
Epoch 11/20
63/63 [==============================] - 27s 421ms/step - loss: 0.6724 - accuracy: 0.5980 -
val_loss: 0.6626 - val_accuracy: 0.5941
Epoch 12/20
63/63 [==============================] - 27s 420ms/step - loss: 0.6750 - accuracy: 0.5840 -
val_loss: 0.6664 - val_accuracy: 0.6176
Epoch 13/20
63/63 [==============================] - 27s 421ms/step - loss: 0.6579 - accuracy: 0.6235 -
val_loss: 0.6614 - val_accuracy: 0.6213
Epoch 14/20
63/63 [==============================] - 27s 422ms/step - loss: 0.6354 - accuracy: 0.6495 -
val_loss: 0.6237 - val_accuracy: 0.6423
Epoch 15/20
63/63 [==============================] - 26s 419ms/step - loss: 0.6367 - accuracy: 0.6535 -
val_loss: 0.6434 - val_accuracy: 0.6163
Epoch 16/20
63/63 [==============================] - 27s 421ms/step - loss: 0.6229 - accuracy: 0.6745 -
val_loss: 0.6556 - val_accuracy: 0.6213
Epoch 17/20
63/63 [==============================] - 27s 422ms/step - loss: 0.6647 - accuracy: 0.6360 -
val_loss: 0.6432 - val_accuracy: 0.6225
Epoch 18/20
63/63 [==============================] - 26s 419ms/step - loss: 0.6441 - accuracy: 0.6355 -
val_loss: 0.6345 - val_accuracy: 0.6374
Epoch 19/20
63/63 [==============================] - 26s 419ms/step - loss: 0.6187 - accuracy: 0.6640 -
val_loss: 0.6144 - val_accuracy: 0.6658
Epoch 20/20
63/63 [==============================] - 26s 420ms/step - loss: 0.5975 - accuracy: 0.6730 -
val_loss: 0.6098 - val_accuracy: 0.6708
```

```
# Plot training history of model2
plt.gca().set(xlabel = "epoch", ylabel = "accuracy")
plt.plot(history2.history["accuracy"], label = "training")
plt.plot(history2.history["val_accuracy"], label = "validation")
plt.legend()
plt.show()
```



Wow! **model2** seems to have exceeded expectations by a very large margin! **The verification accuracy is between 60% and 67%**. This is an improvement of about 5% compared to **model1**. And it is worth mentioning that the accuracy of the training set is very close to the validation accuracy. This means that we have solved the overfitting problem very well.

Part 6: Data Preprocessing

Now let's consider another way to improve accuracy. We find the data has pixels with RGB values between 0 and 255, but many models train faster when the RGB values are normalized between -1 and 1. So we can make it smaller to help the model train and descent faster. In model3, we will use **keras.application.mobilnet_v2.preprocess_input()** to rescale the values to [-1, 1] range.

```
i = tf.keras.Input(shape=(160, 160, 3))
x = tf.keras.applications.mobilenet_v2.preprocess_input(i)
preprocessor = tf.keras.Model(inputs = [i], outputs = [x])
```

Let **preprocessor** be integrated in **model2** and we rename it as **model3**.

```
model3 = models.Sequential([
    # different part with model2
    preprocessor,

    # same with model2
    layers.RandomFlip(),
    layers.RandomRotation(factor=0.25),

    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(160, 160, 3)),layers.MaxPooling2
    layers.Conv2D(32, (3, 3), activation='relu'),layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),layers.MaxPooling2D((2, 2)),
```

```python
    layers.Flatten(),

    layers.Dense(1024, activation='relu'),layers.Dropout(0.2),
    layers.Dense(64, activation='relu'),layers.Dropout(0.2),
    layers.Dense(2)
    ])
```

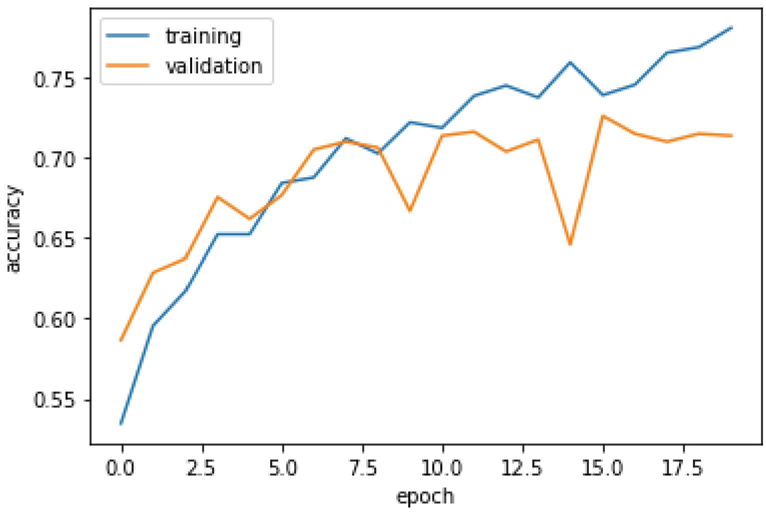
Now let's show how model3 performs with added **preprocessor**.

```python
model3.compile(optimizer='adam',
              loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics = ['accuracy'])
history3 = model3.fit(train_dataset,
                     epochs=20,
                     validation_data=validation_dataset)
```

```
Epoch 1/20
63/63 [==============================] - 28s 425ms/step - loss: 0.7171 - accuracy: 0.5345 -
val_loss: 0.6638 - val_accuracy: 0.5866
Epoch 2/20
63/63 [==============================] - 27s 422ms/step - loss: 0.6580 - accuracy: 0.5955 -
val_loss: 0.6454 - val_accuracy: 0.6287
Epoch 3/20
63/63 [==============================] - 26s 419ms/step - loss: 0.6438 - accuracy: 0.6170 -
val_loss: 0.6281 - val_accuracy: 0.6374
Epoch 4/20
63/63 [==============================] - 26s 417ms/step - loss: 0.6267 - accuracy: 0.6525 -
val_loss: 0.6130 - val_accuracy: 0.6757
Epoch 5/20
63/63 [==============================] - 26s 417ms/step - loss: 0.6311 - accuracy: 0.6525 -
val_loss: 0.6036 - val_accuracy: 0.6621
Epoch 6/20
63/63 [==============================] - 26s 415ms/step - loss: 0.5924 - accuracy: 0.6845 -
val_loss: 0.5906 - val_accuracy: 0.6770
Epoch 7/20
63/63 [==============================] - 26s 417ms/step - loss: 0.5932 - accuracy: 0.6880 -
val_loss: 0.5673 - val_accuracy: 0.7054
Epoch 8/20
63/63 [==============================] - 26s 418ms/step - loss: 0.5674 - accuracy: 0.7125 -
val_loss: 0.5920 - val_accuracy: 0.7104
Epoch 9/20
63/63 [==============================] - 26s 418ms/step - loss: 0.5780 - accuracy: 0.7030 -
val_loss: 0.5688 - val_accuracy: 0.7067
Epoch 10/20
63/63 [==============================] - 26s 417ms/step - loss: 0.5567 - accuracy: 0.7225 -
val_loss: 0.6187 - val_accuracy: 0.6671
Epoch 11/20
63/63 [==============================] - 27s 421ms/step - loss: 0.5441 - accuracy: 0.7190 -
val_loss: 0.5632 - val_accuracy: 0.7141
Epoch 12/20
63/63 [==============================] - 26s 418ms/step - loss: 0.5356 - accuracy: 0.7390 -
val_loss: 0.5682 - val_accuracy: 0.7166
```

```
Epoch 13/20
63/63 [==============================] - 26s 418ms/step - loss: 0.5289 - accuracy: 0.7455 -
val_loss: 0.5820 - val_accuracy: 0.7042
Epoch 14/20
63/63 [==============================] - 27s 420ms/step - loss: 0.5146 - accuracy: 0.7380 -
val_loss: 0.5895 - val_accuracy: 0.7116
Epoch 15/20
63/63 [==============================] - 26s 420ms/step - loss: 0.5099 - accuracy: 0.7600 -
val_loss: 0.6207 - val_accuracy: 0.6460
Epoch 16/20
63/63 [==============================] - 26s 418ms/step - loss: 0.5206 - accuracy: 0.7395 -
val_loss: 0.5552 - val_accuracy: 0.7265
Epoch 17/20
63/63 [==============================] - 26s 418ms/step - loss: 0.5087 - accuracy: 0.7460 -
val_loss: 0.5731 - val_accuracy: 0.7153
Epoch 18/20
63/63 [==============================] - 26s 420ms/step - loss: 0.4967 - accuracy: 0.7660 -
val_loss: 0.5817 - val_accuracy: 0.7104
Epoch 19/20
63/63 [==============================] - 26s 419ms/step - loss: 0.4680 - accuracy: 0.7695 -
val_loss: 0.7054 - val_accuracy: 0.7153
Epoch 20/20
63/63 [==============================] - 26s 417ms/step - loss: 0.4606 - accuracy: 0.7815 -
val_loss: 0.6037 - val_accuracy: 0.7141
```

```
# Plot training history of model3
plt.gca().set(xlabel = "epoch", ylabel = "accuracy")
plt.plot(history3.history["accuracy"], label = "training")
plt.plot(history3.history["val_accuracy"], label = "validation")
plt.legend()
plt.show()
```



As shown in the plot, the verification accuracy of model3 is about **71%** after 20 epochs. This result is more accurate for the previous models and does not show overfitting problems. However, the accuracy is still insufficient.

Part 7: Transfer Learning

If we want to use someone else's pre-trained model, we will need to use MobileNetV2. To do this, we need to first access a pre-existing "base model", merge it into the full model for our current task, and then train that model.

```
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
base_model.trainable = False

i = tf.keras.Input(shape=IMG_SHAPE)
x = base_model(i, training = False)
base_model_layer = tf.keras.Model(inputs = [i], outputs = [x])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_160_no_top.h5
9406464/9406464 [==============================] - 0s 0us/step
```

Here, we add **keras.applications.MobileNetV2** to the model and name the new model **model4**. **model4** body is based on **model3**.

```
model4 = models.Sequential([

    preprocessor,
    layers.RandomFlip(),
    layers.RandomRotation(factor=0.25),

    base_model_layer,
    layers.GlobalMaxPooling2D(),

    layers.Flatten(),

    layers.Dense(1024),layers.Dropout(0.2),
    layers.Dense(64),layers.Dropout(0.2),
    layers.Dense(2)
    ])
```

Now, let's show how **model4** perfoms.

```
model4.compile(optimizer='adam',
              loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics = ['accuracy'])
history4 = model4.fit(train_dataset,
                     epochs=20,
                     validation_data=validation_dataset)
```

```
Epoch 1/20
63/63 [==============================] - 21s 296ms/step - loss: 4.4619 - accuracy: 0.8570 -
val_loss: 0.4986 - val_accuracy: 0.9666
Epoch 2/20
63/63 [==============================] - 18s 288ms/step - loss: 1.6888 - accuracy: 0.9070 -
val_loss: 0.3190 - val_accuracy: 0.9765
```

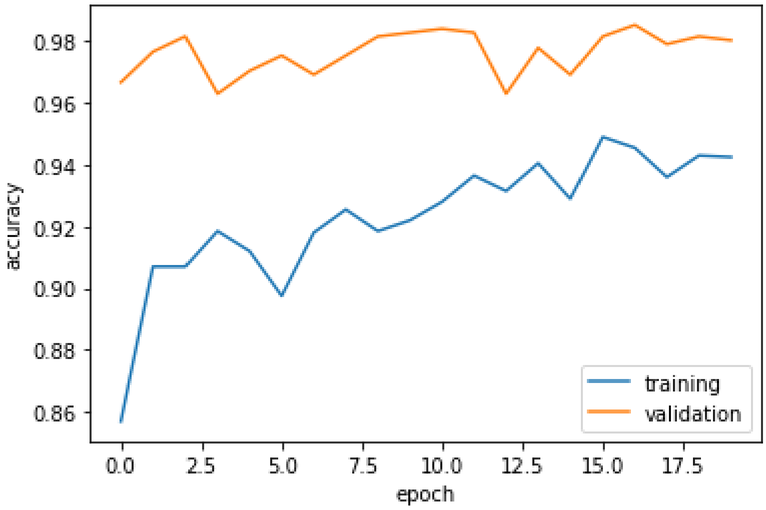
```
Epoch 3/20
63/63 [==============================] - 18s 289ms/step - loss: 2.0459 - accuracy: 0.9070 -
val_loss: 0.3164 - val_accuracy: 0.9814
Epoch 4/20
63/63 [==============================] - 18s 288ms/step - loss: 1.8678 - accuracy: 0.9185 -
val_loss: 0.7034 - val_accuracy: 0.9629
Epoch 5/20
63/63 [==============================] - 18s 291ms/step - loss: 1.7940 - accuracy: 0.9120 -
val_loss: 0.4713 - val_accuracy: 0.9703
Epoch 6/20
63/63 [==============================] - 18s 289ms/step - loss: 2.4879 - accuracy: 0.8975 -
val_loss: 0.2816 - val_accuracy: 0.9752
Epoch 7/20
63/63 [==============================] - 18s 288ms/step - loss: 1.8170 - accuracy: 0.9180 -
val_loss: 0.5227 - val_accuracy: 0.9691
Epoch 8/20
63/63 [==============================] - 18s 289ms/step - loss: 1.2003 - accuracy: 0.9255 -
val_loss: 0.3207 - val_accuracy: 0.9752
Epoch 9/20
63/63 [==============================] - 18s 290ms/step - loss: 1.1533 - accuracy: 0.9185 -
val_loss: 0.1759 - val_accuracy: 0.9814
Epoch 10/20
63/63 [==============================] - 18s 288ms/step - loss: 1.1663 - accuracy: 0.9220 -
val_loss: 0.2121 - val_accuracy: 0.9827
Epoch 11/20
63/63 [==============================] - 18s 288ms/step - loss: 1.0459 - accuracy: 0.9280 -
val_loss: 0.2251 - val_accuracy: 0.9839
Epoch 12/20
63/63 [==============================] - 18s 289ms/step - loss: 0.9446 - accuracy: 0.9365 -
val_loss: 0.2204 - val_accuracy: 0.9827
Epoch 13/20
63/63 [==============================] - 18s 289ms/step - loss: 0.7152 - accuracy: 0.9315 -
val_loss: 0.2980 - val_accuracy: 0.9629
Epoch 14/20
63/63 [==============================] - 18s 288ms/step - loss: 0.6093 - accuracy: 0.9405 -
val_loss: 0.1655 - val_accuracy: 0.9777
Epoch 15/20
63/63 [==============================] - 18s 289ms/step - loss: 0.5765 - accuracy: 0.9290 -
val_loss: 0.2894 - val_accuracy: 0.9691
Epoch 16/20
63/63 [==============================] - 18s 289ms/step - loss: 0.5553 - accuracy: 0.9490 -
val_loss: 0.1682 - val_accuracy: 0.9814
Epoch 17/20
63/63 [==============================] - 18s 288ms/step - loss: 0.5034 - accuracy: 0.9455 -
val_loss: 0.1221 - val_accuracy: 0.9851
Epoch 18/20
63/63 [==============================] - 18s 289ms/step - loss: 0.4973 - accuracy: 0.9360 -
val_loss: 0.1336 - val_accuracy: 0.9790
Epoch 19/20
63/63 [==============================] - 18s 287ms/step - loss: 0.3915 - accuracy: 0.9430 -
val_loss: 0.0998 - val_accuracy: 0.9814
Epoch 20/20
63/63 [==============================] - 18s 288ms/step - loss: 0.3586 - accuracy: 0.9425 -
val_loss: 0.1353 - val_accuracy: 0.9802
```

```python
model4.summary()
```

```
Model: "sequential_13"

_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 model (Functional)          (None, 160, 160, 3)       0         
                                                                 
 random_flip_8 (RandomFlip)  (None, 160, 160, 3)       0         
                                                                 
 random_rotation_8 (RandomRo  (None, 160, 160, 3)      0         
 tation)                                                         
                                                                 
 model_1 (Functional)        (None, 5, 5, 1280)        2257984   
                                                                 
 global_max_pooling2d (Globa  (None, 1280)             0         
 lMaxPooling2D)                                                  
                                                                 
 flatten_7 (Flatten)         (None, 1280)              0         
                                                                 
 dense_21 (Dense)            (None, 1024)              1311744   
                                                                 
 dropout_14 (Dropout)        (None, 1024)              0         
                                                                 
 dense_22 (Dense)            (None, 64)                65600     
                                                                 
 dropout_15 (Dropout)        (None, 64)                0         
                                                                 
 dense_23 (Dense)            (None, 2)                 130       
                                                                 
=================================================================
Total params: 3,635,458
Trainable params: 1,377,474
Non-trainable params: 2,257,984
_________________________________________________________________
```

The above table demonstrates the **MobileNetV2** model in action. It makes the input (160, 160, 3) into a (5, 5, 1280) output. The parameters are 2257984. From the data we can find that the 1377474 parameters still need to be trained in model4.

```python
# Plot training history of model4
plt.gca().set(xlabel = "epoch", ylabel = "accuracy")
plt.plot(history4.history["accuracy"], label = "training")
plt.plot(history4.history["val_accuracy"], label = "validation")
plt.legend()
plt.show()
```



The results are impressive! Our verification accuracy is between **96% and 98%**! This is the best performing models we have and there were no overfitting issues.

Part 8: Final Test

Eventually, we will use the test set for the final evaluation of model4. Let's see how the model4 performs.

```
labels = np.array([])
predicted_labels = np.array([])
for test_values, test_labels in test_dataset:
    y_axis = model4.predict(test_values)
    max_predicted_labels = y_axis.argmax(axis=1)
    labels = np.concatenate((labels, test_labels))
    predicted_labels = np.concatenate((predicted_labels, max_predicted_labels))

print(f"Accuracy of test set: {np.average(labels == predicted_labels)}")
```

```
1/1 [==============================] - 0s 191ms/step
1/1 [==============================] - 0s 195ms/step
1/1 [==============================] - 0s 189ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 196ms/step
1/1 [==============================] - 0s 195ms/step
Accuracy of test set: 0.9895833333333334
```

The final result is about **99% accurate**. I believe this model has been able to fully recognize images of dogs and cats outside of our dataset.