



File Edit View Insert Runtime Tools Help Cannot save changes

RAM Disk

Editing

+ Code + Text Copy to Drive



## Basic transfer learning with cats and dogs data

## Import tensorflow

```
[1] try:
    # %tensorflow_version only exists in Colab.
    %tensorflow_version 2.x
except Exception:
    pass
```

## Import modules and download the cats and dogs dataset.

```
[2] import urllib.request
import os
import zipfile
import random
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers
from tensorflow.keras import Model
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.optimizers import RMSprop
from shutil import copyfile

data_url = "https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-4869-8368-6DEBA77B919F/kagglecatsanddogs_3367a.zip"
data_file_name = "catsdogs.zip"
download_dir = '/tmp/'
urllib.request.urlretrieve(data_url, data_file_name)
zip_ref = zipfile.ZipFile(data_file_name, 'r')
zip_ref.extractall(download_dir)
zip_ref.close()
```

Check that the dataset has the expected number of examples.

```
[3] print("Number of cat images:", len(os.listdir('/tmp/PetImages/Cat/')))
print("Number of dog images:", len(os.listdir('/tmp/PetImages/Dog/')))

# Expected Output:
# Number of cat images: 12501
# Number of dog images: 12501

Number of cat images: 12501
Number of dog images: 12501
```

Create some folders that will store the training and test data.

- There will be a training folder and a testing folder.
- Each of these will have a subfolder for cats and another subfolder for dogs.

```
[4] try:
    os.mkdir('/tmp/cats-v-dogs')
    os.mkdir('/tmp/cats-v-dogs/training')
    os.mkdir('/tmp/cats-v-dogs/testing')
    os.mkdir('/tmp/cats-v-dogs/training/cats')
    os.mkdir('/tmp/cats-v-dogs/training/dogs')
    os.mkdir('/tmp/cats-v-dogs/testing/cats')
    os.mkdir('/tmp/cats-v-dogs/testing/dogs')
except OSError:
    pass
```

## Split data into training and test sets

- The following code put first checks if an image file is empty (zero length)
- Of the files that are not empty, it puts 90% of the data into the training set, and 10% into the test set.

```
[5] import random
from shutil import copyfile
def split_data(SOURCE, TRAINING, TESTING, SPLIT_SIZE):
    files = []
    for filename in os.listdir(SOURCE):
        file = SOURCE + filename
        if os.path.getsize(file) > 0:
            files.append(filename)
        else:
            print(filename + " is zero length, so ignoring.")

    training_length = int(len(files) * SPLIT_SIZE)
    testing_length = int(len(files) - training_length)
    shuffled_set = random.sample(files, len(files))
    training_set = shuffled_set[0:training_length]
    testing_set = shuffled_set[training_length:]

    for filename in training_set:
        this_file = SOURCE + filename
        destination = TRAINING + filename
        copyfile(this_file, destination)

    for filename in testing_set:
```

```

this_file = SOURCE + filename
destination = TESTING + filename
copyfile(this_file, destination)

CAT_SOURCE_DIR = "/tmp/PetImages/Cat/"
TRAINING_CATS_DIR = "/tmp/cats-v-dogs/training/cats/"
TESTING_CATS_DIR = "/tmp/cats-v-dogs/testing/cats/"
DOG_SOURCE_DIR = "/tmp/PetImages/Dog/"
TRAINING_DOGS_DIR = "/tmp/cats-v-dogs/training/dogs/"
TESTING_DOGS_DIR = "/tmp/cats-v-dogs/testing/dogs/"

split_size = .9
split_data(CAT_SOURCE_DIR, TRAINING_CATS_DIR, TESTING_CATS_DIR, split_size)
split_data(DOG_SOURCE_DIR, TRAINING_DOGS_DIR, TESTING_DOGS_DIR, split_size)

# Expected output
# 666.jpg is zero length, so ignoring
# 11702.jpg is zero length, so ignoring

666.jpg is zero length, so ignoring.
11702.jpg is zero length, so ignoring.

```

Check that the training and test sets are the expected lengths.

```

[6]
os
    print("Number of training cat images", len(os.listdir('/tmp/cats-v-dogs/training/cats/')))
    print("Number of training dog images", len(os.listdir('/tmp/cats-v-dogs/training/dogs/')))
    print("Number of testing cat images", len(os.listdir('/tmp/cats-v-dogs/testing/cats/')))
    print("Number of testing dog images", len(os.listdir('/tmp/cats-v-dogs/testing/dogs/')))

# expected output
# Number of training cat images 11250
# Number of training dog images 11250
# Number of testing cat images 1250
# Number of testing dog images 1250

Number of training cat images 11250
Number of training dog images 11250
Number of testing cat images 1250
Number of testing dog images 1250

```

#### ▼ Data augmentation (try adjusting the parameters)!

Here, you'll use the `ImageDataGenerator` to perform data augmentation.

- Things like rotating and flipping the existing images allows you to generate training data that is more varied, and can help the model generalize better during training.
- You can also use the data generator to apply data augmentation to the validation set.

You can use the default parameter values for a first pass through this lab.

- Later, try to experiment with the parameters of `ImageDataGenerator` to improve the model's performance.
- Try to drive reach 99.9% validation accuracy or better.

```

[7]
os
    TRAINING_DIR = "/tmp/cats-v-dogs/training/"
    # Experiment with your own parameters to reach 99.9% validation accuracy or better
    train_datagen = ImageDataGenerator(rescale=1./255,
                                        rotation_range=40,
                                        width_shift_range=0.2,
                                        height_shift_range=0.2,
                                        shear_range=0.2,
                                        zoom_range=0.2,
                                        horizontal_flip=True,
                                        fill_mode='nearest')
    train_generator = train_datagen.flow_from_directory(TRAINING_DIR,
                                                        batch_size=100,
                                                        class_mode='binary',
                                                        target_size=(150, 150))

    VALIDATION_DIR = "/tmp/cats-v-dogs/testing/"

    validation_datagen = ImageDataGenerator(rescale=1./255)
    validation_generator = validation_datagen.flow_from_directory(VALIDATION_DIR,
                                                                batch_size=100,
                                                                class_mode='binary',
                                                                target_size=(150, 150))

```

```

Found 22499 images belonging to 2 classes.
Found 2499 images belonging to 2 classes.

```

#### ▼ Get and prepare the model

You'll be using the InceptionV3 model.

- Since you're making use of transfer learning, you'll load the pre-trained weights of the model.
- You'll also freeze the existing layers so that they aren't trained on your downstream task with the cats and dogs data.
- You'll also get a reference to the last layer, 'mixed7' because you'll add some layers after this last layer.

```

[8]
11s
weights_url = "https://storage.googleapis.com/mledu-datasets/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5"
weights_file = "inception_v3.h5"
urllib.request.urlretrieve(weights_url, weights_file)

# Instantiate the model
pre_trained_model = InceptionV3(input_shape=(150, 150, 3),
                                  include_top=False,
                                  weights=None)

# load pre-trained weights
pre_trained_model.load_weights(weights_file)

```

```

# freeze the layers
for layer in pre_trained_model.layers:
    layer.trainable = False

# pre_trained_model.summary()

last_layer = pre_trained_model.get_layer('mixed7')
print('last layer output shape: ', last_layer.output_shape)
last_output = last_layer.output

```

last layer output shape: (None, 7, 7, 768)

## ▼ Add layers

Add some layers that you will train on the cats and dogs data.

- **Flatten**: This will take the output of the `last_layer` and flatten it to a vector.
- **Dense**: You'll add a dense layer with a relu activation.
- **Dense**: After that, add a dense layer with a sigmoid activation. The sigmoid will scale the output to range from 0 to 1, and allow you to interpret the output as a prediction between two categories (cats or dogs).

Then create the model object.

```

[9] # Flatten the output layer to 1 dimension
x = layers.Flatten()(last_output)
# Add a fully connected layer with 1,024 hidden units and ReLU activation
x = layers.Dense(1024, activation='relu')(x)
# Add a final sigmoid layer for classification
x = layers.Dense(1, activation='sigmoid')(x)

model = Model(pre_trained_model.input, x)

```

## ▼ Train the model

Compile the model, and then train it on the test data using `model.fit`

- Feel free to adjust the number of epochs. This project was originally designed with 20 epochs.
- For the sake of time, you can use fewer epochs (2) to see how the code runs.
- You can ignore the warnings about some of the images having corrupt EXIF data. Those will be skipped.

```

[10]
# compile the model
model.compile(optimizer=RMSprop(lr=0.0001),
              loss='binary_crossentropy',
              metrics=['acc'])

# train the model (adjust the number of epochs from 1 to improve performance)
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=2,
    verbose=1)

/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/optimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
"The `lr` argument is deprecated, use `learning_rate` instead.")
Epoch 1/2
19/225 [=====] - ETA: 2:27 - loss: 0.6273 - acc: 0.7995/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.
" Skipping tag %s" % (size, len(data), tag)
/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data. Expecting to read 5 bytes but only got 0. Skipping tag 271
" Skipping tag %s" % (size, len(data), tag)
/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data. Expecting to read 8 bytes but only got 0. Skipping tag 272
" Skipping tag %s" % (size, len(data), tag)
/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data. Expecting to read 8 bytes but only got 0. Skipping tag 282
" Skipping tag %s" % (size, len(data), tag)
/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data. Expecting to read 8 bytes but only got 0. Skipping tag 283
" Skipping tag %s" % (size, len(data), tag)
/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data. Expecting to read 20 bytes but only got 0. Skipping tag 306
" Skipping tag %s" % (size, len(data), tag)
/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data. Expecting to read 48 bytes but only got 0. Skipping tag 532
" Skipping tag %s" % (size, len(data), tag)
/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:788: UserWarning: Corrupt EXIF data. Expecting to read 2 bytes but only got 0.
warnings.warn(str(msg))
225/225 [=====] - 203s 758ms/step - loss: 0.2325 - acc: 0.9053 - val_loss: 0.0769 - val_acc: 0.9736
Epoch 2/2
225/225 [=====] - 168s 744ms/step - loss: 0.1487 - acc: 0.9372 - val_loss: 0.0741 - val_acc: 0.9696

```

## ▼ Visualize the training and validation accuracy

You can see how the training and validation accuracy change with each epoch on an x-y plot.

```

[11] %matplotlib inline

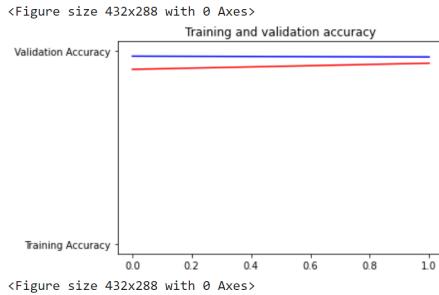
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

#-----
# Retrieve a list of list results on training and test data
# sets for each training epoch
#-----
acc=history.history['acc']
val_acc=history.history['val_acc']
loss=history.history['loss']
val_loss=history.history['val_loss']

epochs=range(len(acc)) # Get number of epochs
#-----
# Plot training and validation accuracy per epoch
#-----

```

```
plt.plot(epochs, acc, 'r', "Training Accuracy")
plt.plot(epochs, val_acc, 'b', "Validation Accuracy")
plt.title('Training and validation accuracy')
plt.figure()
```



#### ▼ Predict on a test image

You can upload any image and have the model predict whether it's a dog or a cat.

- Find an image of a dog or cat
- Run the following code cell. It will ask you to upload an image.
- The model will print "is a dog" or "is a cat" depending on the model's prediction.

```
2h 2h
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

    # predicting images
    path = '/content/' + fn
    img = image.load_img(path, target_size=(150, 150))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)

    image_tensor = np.vstack([x])
    classes = model.predict(image_tensor)
    print(classes)
    print(classes[0])
    if classes[0]>0.5:
        print(fn + " is a dog")
    else:
        print(fn + " is a cat")
```

Choose Files 3408.jpg

- 3408.jpg(image/jpeg) - 103363 bytes, last modified: 8/5/2021 - 100% done

Saving 3408.jpg to 3408.jpg

[[1.]]  
[1.]  
3408.jpg is a dog