civica

Transforming the way you work

Image Classification using Machine Learning

Andrew Fletcher / Liam Fletcher

29th February 2020, SISTEM.

civica

Contents

civica

Introduction:

Background

In this workshop, we will step through the process of defining and training a Convolutional Neural Network to tackle a typical image classification problem.

This guide takes the format of a step-by-step tutorial that walks the reader through the machine learning process. At each point, Python snippets are provided alongside an explanation of their behaviour. In order to facilitate further learning, exercises are left for the reader to complete and experiment with.

As discussed in the accompanying presentation, this workshop aims to cover each section of the Machine Learning process, including:

- Acquisition and extraction of a dataset that has been prepared for this problem.
- Implementation of appropriate pre-processing logic to prepare the dataset for use.
- Configuration of the Convolutional Neural Network.
- Training and evaluation of the model.
- Use of the model to classify previously unseen samples.

The Problem:

This tutorial presents a typical image classification problem that can be tackled with a Convolutional Machine Learning model.

In this scenario, a dataset is provided containing 4,500 images of Cats, Dogs and Meerkats. The breakdown of the dataset is shown in the table below:

Training	
Cat	1,000
Dog	1,000
Meerkat	1,000
Test	
Cat	500
Dog	500
Meerkat	500

Pre-requisites and tools

In order to complete this tutorial, the following technologies and tools will be used:

Jupyter Notebook - An environment that allows individual blocks of Python code to be written and executed in a “note taking” style manner. Through Jupyter, code can be experimented with in sections without the need to run an entire script each time.

Keras - A Machine Learning framework that allows models to be defined quickly through a focus on ease of use and experimentation.

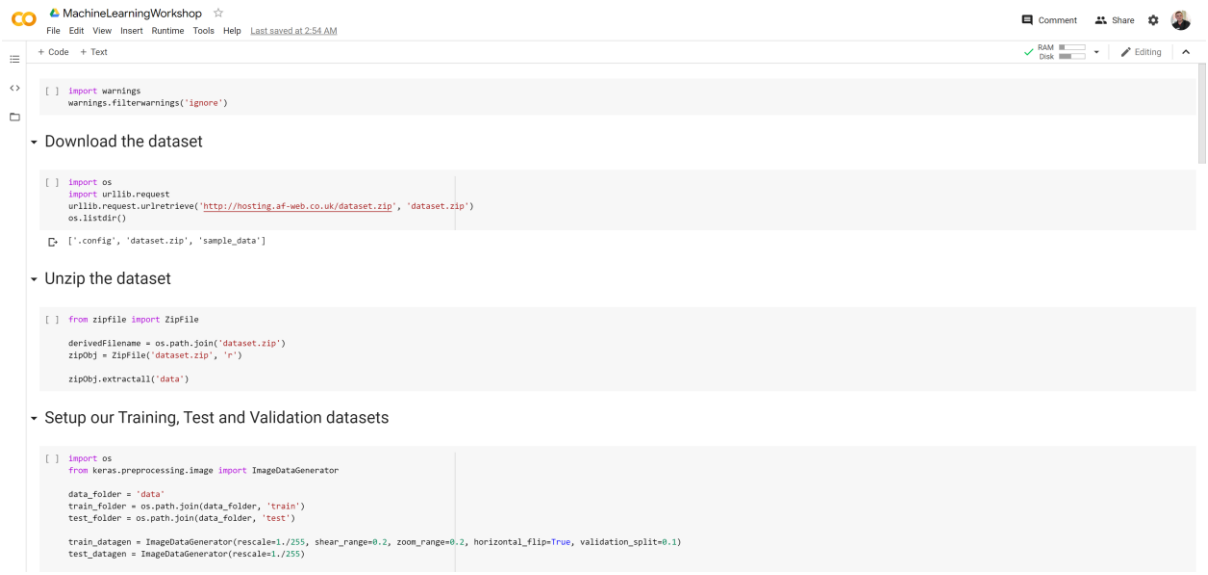
The dataset as described above

Outcomes

At the end of this workshop, our aim is that you will have an introductory understanding of the Machine Learning process alongside its application to an image classification problem.

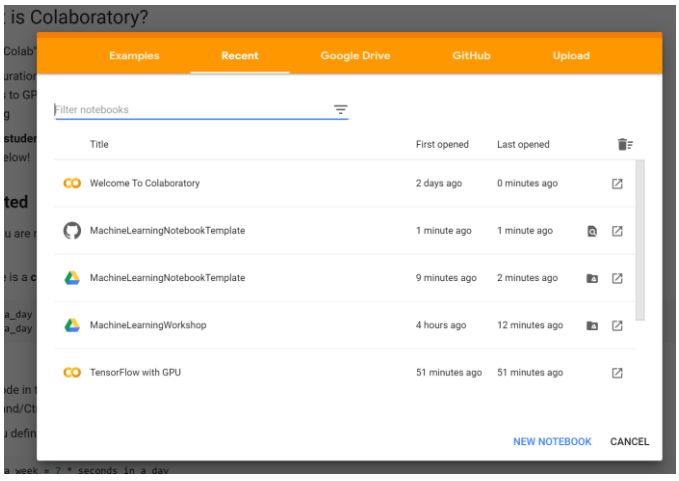
Getting setup on Google CoLab

When first building a Machine Learning Model, it is common to use a notebook environment such as Jupyter to complete your initial experiments. This environment provides the flexibility of a scripting environment with the ability to execute specific blocks as required. An example is shown below:



For this tutorial, we will be using Google CoLab. CoLab is a cloud-based, pre-configured instance of Jupyter provided by Google. CoLab can be setup as follows:

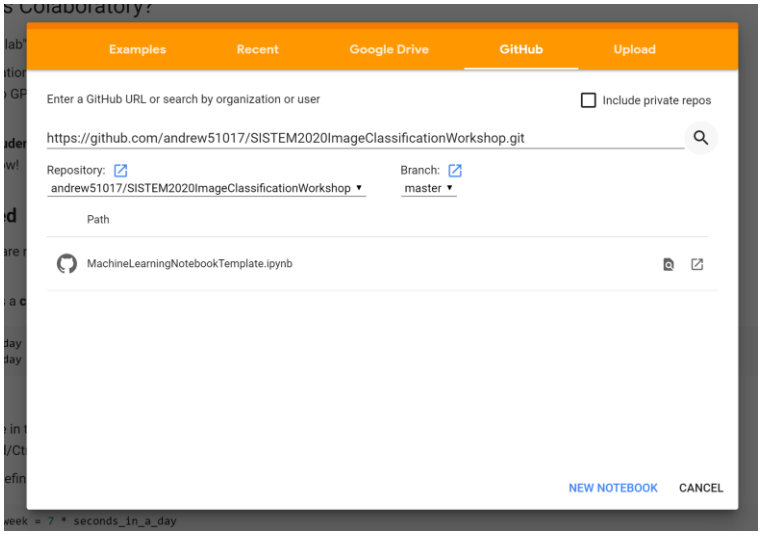
Navigate to "https://colab.research.google.com/" and login with your Google Account if prompted. The below screen will be presented



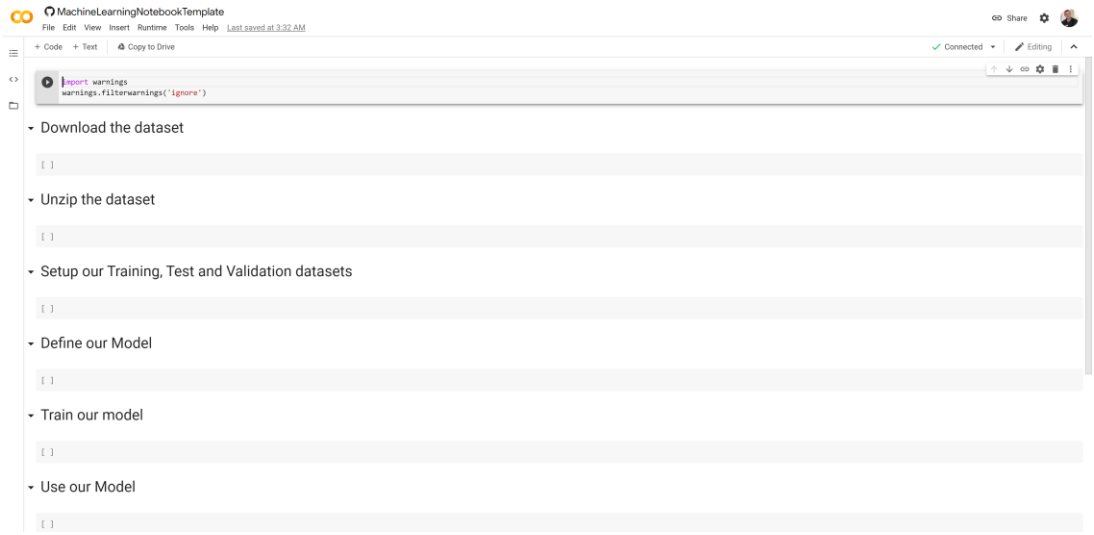
Click "GitHub" and enter the following URL to download the template notebook.

https://github.com/andrew51017/SISTEM2020ImageClassificationWorkshop.git

Click the Search icon and a single result will be shown as follows:



Click the result to create your new notebook in CoLab. The template notebook will be displayed.

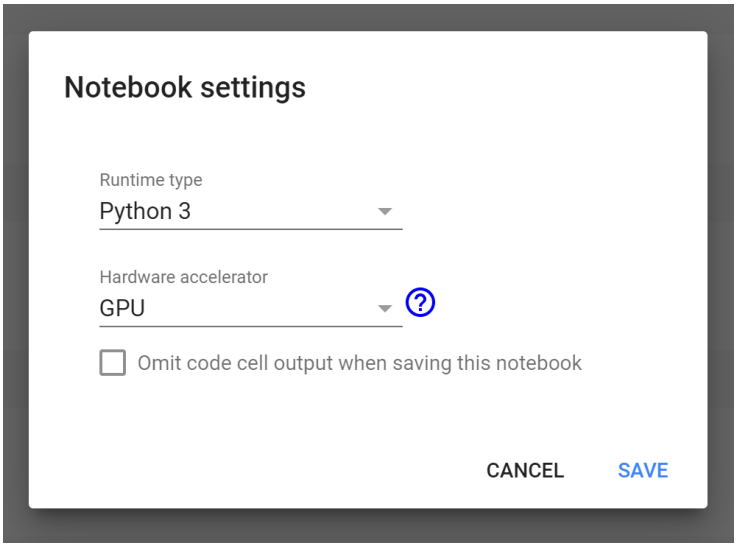


Enabling GPU Support on CoLab

Training a machine learning model can be particularly taxing! As such, it is very common to use a GPU whilst training. By using a GPU, complex models can be training significantly faster than if a CPU were used.

In order to reduce the time spent training our model, a GPU can be enabled in CoLab as follows:

From the “Edit” menu, click “Notebook Settings”
In the “Hardware Accelerator” field, select the GPU option.



Click Save to be returned to your notebook.

Download the dataset to our CoLab workspace

Before the dataset can be used for training, it must first be downloaded and extracted for use in our CoLab workspace. Follow the steps below to complete this process:

To download the dataset, enter the following logic below the “Download and Unzip the dataset” header:

```
import os
import urllib.request
urllib.request.urlretrieve('http://hosting.af-
web.co.uk/dataset.zip', 'dataset.zip')
os.listdir()
```

Click the “Play” button on the left hand size of the row to execute the block. The execution could take a few moments.

This logic will import the Python libraries that are required to download a file and will then download the dataset to your workspace.

The following output should be displayed. The presence of “dataset.zip” indicates a successful download.

```
['.config', 'dataset.zip', 'sample_data']
```

Next, we need to unzip the dataset in order to use it. To do this, first add a new code block by clicking the “+ Code” button in the top left.

Populate the new code block as follows:

```
from zipfile import ZipFile

derivedFilename = os.path.join('dataset.zip')
zipObj = ZipFile('dataset.zip', 'r')

zipObj.extractall('data')
```

After clicking the “Play” button on this row, the contents of the ZIP file will be extracted to your workspace.

To confirm the contents of your workspace, click the “Folder” icon on the left hand side. Note that the folder structure is as per the below:

Note that the format above is used whilst loading the dataset to derive the classes we are targeting. An extra animal could be added to our scenario by adding extra subfolders to the test and train datasets.

Setup our Training, Test and Validation datasets

Machine learning is a form of AI where the algorithm gets better as it sees more data over time. As such, the data being used is critical to the success of the AI model.

When applying machine learning to a problem, ideally, our data set should be:

- Sufficiently large, with equal representation of each class.
- Free from bias that may adversely impact the output of the model.

In order to build a reliable model, it is important that we maintain a split between the data used to train the model and the data used to evaluate the performance of the model. As such, we split our dataset into a number of sub-sets as follows:

- Training data: Data that is used to train the model.
- Validation data: A subset of the training data that is used for testing in each cycle
- Test data: Data that we hold in reserve to evaluate the performance of the model.

Through the use of a test dataset, unique variations of the model can be evaluated and their performance against previously unseen samples compared.

Our dataset can be split as follows

In the code block below “Setup our Training, Test and Validation datasets”, import the necessary Keras objects and specify the paths to our training and test datasets.

```
import os
from keras.preprocessing.image import ImageDataGenerator

data_folder = 'data'
train_folder = os.path.join(data_folder, 'train')
test_folder = os.path.join(data_folder, 'test')
```

Create an “ImageDataGenerator” object for the training and test datasets.

Whilst our training dataset contains 3,000 images, it is beneficial to augment this even further. An ImageDataGenerator is augment the dataset with zoomed, flipped and cropped versions of the images provided.

Set up the Image Data Generators as follows.

```
train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2, z
oom_range=0.2, horizontal_flip=True, validation_split=0.1)
test_datagen = ImageDataGenerator(rescale=1./255)
```

Try experimenting with the rescale, shear range, zoom range and flip parameters to investigate how these impact the training time and accuracy metrics.

Use our data generators to setup the various datasets we will use.

```
img_size = 150
batch_size = 32
```

To ensure consistency, it is important that our images are all the same size. In this scenario, we have specified a size of 150x150 pixels.

```
train_generator = train_datagen.flow_from_directory(train_folder, tar
get_size=(img_size, img_size), batch_size=batch_size, class_mode='cat
egorical', subset='training')
validation_generator = train_datagen.flow_from_directory(train_folder
, target_size=(img_size, img_size), batch_size=batch_size, class_mode
='categorical', subset='validation')
test_generator = test_datagen.flow_from_directory(test_folder, target
_size=(img_size, img_size), batch_size=batch_size, class_mode='catego
rical')

label_map = (train_generator.class_indices)
```

```
print(label_map)
```

Following this, the 3 classes that are picked up from the dataset subfolders will be output as follows:

```
Found 2700 images belonging to 3 classes.
Found 300 images belonging to 3 classes.
Found 1500 images belonging to 3 classes.
{'cat': 0, 'dog': 1, 'meerkat': 2}
```

If an extra class were added by adding an extra folder to each dataset, the Image Data Generator will pick it up and display it here.

Define our model

At the core of a machine learning solution is the model. A model is a “function” that is designed to “fit” our training data in that it should give the expected output class for a given input sample in our training set.

Whilst many models are available, we will specifically be looking at Neural Networks. Neural Networks work similarly to our own brains and use a number of layers to capture the patterns in our training data.

For this problem, we will be applying a Convolutional Neural Network (CNN). A CNN works by learning to identify a number of features that differentiate the image classes. A standard Neural Network can then use these features to provide an output.

Within a Neural Network layer, each model has a number of “hyper parameters”. These parameters control how the layer behaves.

When devising a model, our ability to control the structure of the model and the hyper parameters can have a significant impact on our models performance.

Within our CNN, we will be using the following types of layers:

Layer	Description	Parameters
Convolutional Layer	A layer that is designed to learn the features in our data.	Filter count – the number of features to be learned by the model. Filter size – the size of the filter.
Max Pooling Layer	A layer that is designed to sample the data and provide a more compact representation. This allows the model to focus on the information that matters. This layer takes the max activation in an area of a given size.	Pool size – the size of area to consider when outputting a value.

Dense / Fully Connected layer	A standard Neural Network layer that can learn patterns. These layers are used to determine a class from a set of features.	Neuron count – the number of neurons in the layer.

In order to train our model, a loss function must be specified. A loss function provides a single value to express how the model is performing. Using the result of the loss function, the parameters of the model can be adjusted during training to correct for the error, therefore improving the performance of the model.

Our initial model can be defined as follows in our notebook:

```
import keras
from keras.models import Sequential, model_from_json
from keras.layers import Dense, Conv2D, MaxPooling2D, BatchNormalizat
ion, Dropout, Flatten, Activation
from keras.optimizers import RMSprop
from keras.callbacks import Callback

model = Sequential()
# A Convolutional Layer with 64 3x3 filters.
model.add(Conv2D(64, (3, 3), input_shape=(150, 150, 3)))
model.add(Activation('relu'))
# A 2x2 max pooling layer.
model.add(MaxPooling2D(pool_size=(2, 2)))

# A Convolutional Layer with 32 3x3 filters.
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
# A 2x2 max pooling layer.
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
# A fully connected layer with 254 neurons.
model.add(Dense(254))
model.add(Activation("relu"))
# A fully connected layer with 64 neurons.
model.add(Dense(64))
model.add(Activation("relu"))
# A fully connected layer with 3 neurons. Each output here correspond
s to a class. Through the use of the Softmax function, the 3 values w
ill each represent a probability and will sum up to 1.
model.add(Dense(3))
model.add(Activation('softmax'))
```

```
model.summary()
```

Upon running this block, the structure of the model will be output.

```
model.compile(loss='categorical_crossentropy', optimizer='rmsprop', m
etrics=['accuracy'])
```

Now that we are happy, we compile our model. We use a loss function that is suitable for multi-class problems such as this. We use the rmsprop optimiser as it is suited for larger models such as this.

Training the model

Using our training data generator, we can train the model to solve our image classification problem.

When training the model, we specify a number of “epochs”. An epoch is a single training cycle where the full training dataset is ran through the model.

Over successive epochs, the model should identify the patterns in our dataset. An increased number of epochs will require further training time.

With a limited dataset, there is a risk that a higher number of epochs will result in the “overfitting” the dataset. This is a scenario whereby the model learns what the exact data it has been trained with rather than learning the patterns of interest.

Our model can be trained as follows:

```
n_epochs = 10

history = model.fit_generator(train_generator, steps_per_epoch = trai
n_generator.samples // batch_size, validation_data = validation_gener
ator,  validation_steps = validation_generator.samples // batch_size,
 epochs = n_epochs)

score = model.evaluate_generator(test_generator, steps= test_generato
r.samples // batch_size)

print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

The model will first be trained on the training dataset. Upon completion, it will be evaluated using the previously unseen test dataset and an overall accuracy will be displayed. Accuracy is one of many metrics that can be used to evaluate a machine learning model.

Experiment with increasing the epoch count and with adjusting the model parameters discussed in the last section. See how these impact the training time and the performance of the model.

civica

Using our model

Now that our model has been trained, we can use it to classify any image.

To classify an image, it must first be processed in the same way as the images used in the training and testing datasets. In our case, this means that the image must be resized to 150x150.

Our model can be consumed as per the below:

Import the necessary libraries and fetch our image.

```
from keras.preprocessing.image import ImageDataGenerator
from IPython.display import Image
import cv2
import numpy as np
from matplotlib import pyplot as plt

resp = urllib.request.urlopen('https://upload.wikimedia.org/wikipedia
/commons/thumb/3/3a/Cat03.jpg/1200px-Cat03.jpg')
```

Parse our image and display it

```
image = np.asarray(bytearray(resp.read()), dtype="uint8")
image = cv2.imdecode(image, cv2.IMREAD_COLOR)

rgbImage = image[:,:,::-1]
plt.imshow(rgbImage)
```

Resize our image and load it into a Data Generator, just like we did with our training and testing datasets.

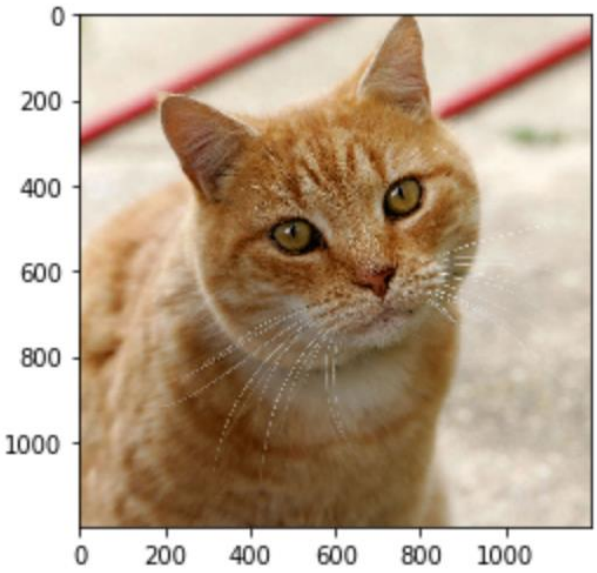
```
r_img = cv2.resize(image, dsize=(img_size, img_size), interpolation=c
v2.INTER_CUBIC)

data = np.array(r_img, dtype=np.float32)

r_data = np.reshape(data, (1,150,150,3))

test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow(r_data, batch_size=1)
```

Classify our sample and show the results

```
res = model.predict_generator(test_generator, steps=1)
```

```
print(label_map)
print(res)
```

```
{'cat': 0, 'dog': 1, 'meerkat': 2}
[[0.6074592  0.25372368 0.13881709]]
```



In this case, we see our model has correctly classified our image as a cat.

Try the model with images of your choosing by switching the URL specified earlier in this section

civica

Resources

We hope you have found this tutorial useful. If you want to explore this area further, here are some links:

Creating your first image recognition classifier	A similar tutorial covering Image classification.	https://medium.com/nybles/create-your-first-image-recognition-classifier-using-cnn-keras-and-tensorflow-backend-6eaab98d14dd
Keras Tutorial: The Ultimate Beginner's Guide to Deep Learning in Python	A tutorial that covers building a Convolutional Neural Network against the famous MNIST dataset.	https://elitedatascience.com/keras-tutorial-deep-learning-in-python
Towards Data Science	A Medium site providing a number of useful examples and tutorials on Machine Learning.	https://towardsdatascience.com/
Google Machine Learning Crash Course	Google's fast-paced, practical introduction to machine learning.	https://developers.google.com/machine-learning/crash-course
The Keras Documentation Examples	The examples provided by the Keras project.	https://keras.io/