Deep Learning with OpenCV

# Deep Learning with OpenCV

In the first part of this post, we’ll discuss the OpenCV 3.3 release and the overhauled dnn module.

We’ll then write a Python script that will use OpenCV and GoogleLeNet (pre-trained on ImageNet) to classify images.

Finally, we’ll explore the results of our classifications.

# Deep Learning inside OpenCV 4

The [dnn module](https://github.com/opencv/opencv/tree/master/modules/dnn) of OpenCV has been part of the ***opencv\_contrib*** repository since version v3.1. Now in OpenCV 3.3 and onward it is included in the main repository.

Why should you care?

Deep Learning is a fast growing domain of Machine Learning and if you’re working in the field of computer vision/image processing already (or getting up to speed), it’s a crucial area to explore.

With OpenCV 4, we can utilize pre-trained networks with popular deep learning frameworks. The fact that they are pre-trained implies that we don’t need to spend many hours training the network — rather we can complete a forward pass and utilize the output to make a decision within our application.

OpenCV does not (and does not intend to be) to be a tool for training networks — there are already great frameworks available for that purpose. Since a network (such as a CNN) can be used as a classifier, it makes logical sense that OpenCV has a Deep Learning module that we can leverage easily within the OpenCV ecosystem.

Popular network architectures compatible with OpenCV 3.3 include:

* GoogleLeNet (used in this blog post)
* AlexNet
* SqueezeNet
* VGGNet (and associated flavors)
* ResNet

The release notes for this module are available on the OpenCV repository [page](https://github.com/opencv/opencv/wiki/Deep-Learning-in-OpenCV).

Aleksandr Rybnikov, the main contributor for this module, has ambitious plans for this module so be sure to stay on the lookout and read his [release notes](https://habrahabr.ru/company/intel/blog/333612/) (in Russian, so make sure you have Google Translation enabled in your browser if Russian is not your native language).

It’s my opinion that the ***dnn*** module will have a big impact on the OpenCV community, so let’s get the word out.

# Configure your machine with OpenCV 4

Simply follow these instructions for MacOS or Ubuntu while making sure to use the opencv and opencv\_contrib releases for OpenCV 4. If you opt for the MacOS + homebrew install instructions, be sure to use the --HEAD switch (among the others mentioned) to get the bleeding edge version of OpenCV.

If you’re using virtual environments (highly recommended), you can easily install OpenCV 4 alongside a previous version. Just create a brand new virtual environment (and name it appropriately) as you follow the tutorial corresponding to your system.

**ADD Comments for my IDE**

# OpenCV deep learning functions and frameworks

OpenCV 4 supports the Caffe, TensorFlow, and Torch/PyTorch frameworks.

Keras is currently not supported (since Keras is actually a wrapper around backends such as TensorFlow and Theano), although I imagine it’s only a matter of time until Keras is directly supported given the popularity of the deep learning library.

Using OpenCV 4 we can load images from disk using the following functions inside dnn :

* cv2.dnn.blobFromImage
* cv2.dnn.blobFromImages

We can directly import models from various frameworks via the “create” methods:

* cv2.dnn.createCaffeImporter
* cv2.dnn.createTensorFlowImporter
* cv2.dnn.createTorchImporter

Although I think it’s easier to simply use the “read” methods and load a serialized model from disk directly:

* cv2.dnn.readNetFromCaffe
* cv2.dnn.readNetFromTensorFlow
* cv2.dnn.readNetFromTorch
* cv2.dnn.readhTorchBlob

Once we have loaded a model from disk, the .forward method is used to forward-propagate our image and obtain the actual classification.

To learn how all these OpenCV deep learning pieces fit together, let’s move on to the next section.

# Classifying images using deep learning and OpenCV

In this section, we’ll be creating a Python script that can be used to classify input images using OpenCV and GoogLeNet (pre-trained on ImageNet) using the Caffe framework.

The GoogLeNet architecture (now known as “Inception” after the novel micro-architecture) was introduced by Szegedy et al. in their 2014 paper, [Going deeper with convolutions](https://arxiv.org/abs/1409.4842).

Other architectures are also supported with OpenCV 3.3 including AlexNet, ResNet, and SqueezeNet — we’ll be examining these architectures for deep learning with OpenCV in a future blog post.

In the meantime, let’s learn how we can load a pre-trained Caffe model and use it to classify an image using OpenCV.

To begin, open up a new file, name it deep\_learning\_with\_opencv.py , and insert the following code:

|  |  |
| --- | --- |
| Deep Learning with OpenCV | |
| 1 | *# import the necessary packages* |
| 2 | import numpy as np |
| 3 | import argparse |
| 4 | import time |
| 5 | import cv2 |

On **Lines 2-5** we import our necessary packages.

Then we parse command line arguments:

|  |  |
| --- | --- |
| Deep Learning with OpenCV | |
| 6 | *# construct the argument parse and parse the arguments* |
| 7 | ap = argparse.ArgumentParser() |
| 8 | ap.add\_argument("-i", "--image", required=True, help="path to input image") |
| 9 | ap.add\_argument("-p", "--prototxt", required=True, help="path to Caffe 'deploy' prototxt file") |
| 10 | ap.add\_argument("-m", "--model", required=True, help="path to Caffe pre-trained model") |
| 11 | ap.add\_argument("-l", "--labels", required=True, help="path to ImageNet labels (i.e., syn-sets)") |
| 12 | args = vars(ap.parse\_args()) |

On **Line 8** we create an argument parser followed by establishing four required command line arguments (**Lines 9-12**):

* --image : The path to the input image.
* --prototxt : The path to the Caffe “deploy” prototxt file.
* --model : The pre-trained Caffe model (i.e,. the network weights themselves).
* --labels : The path to ImageNet labels (i.e., “syn-sets”).

Now that we’ve established our arguments, we parse them and store them in a variable, ***args*** , for easy access later.

Let’s load the input image and class labels:

|  |  |
| --- | --- |
| Deep Learning with OpenCV | |
| 13 | *# load the input image from disk* |
| 14 | image = cv2.imread(args["image"]) |
| 15 |  |
| 15 | *# load the class labels from disk* |
| 17 | rows = open(args["labels"]).read().strip().split("\n") |
| 18 | classes = [r[r.find(" ") + 1:].split(",")[0] for r in rows] |

On Line 14, we load the image from disk via cv2.imread .

Let’s take a closer look at the class label data which we load on Lines 17 and 18:

|  |  |
| --- | --- |
| Deep Learning with OpenCV | |
| 1 | n01440764 tench, Tinca tinca |
| 2 | n01443537 goldfish, Carassius auratus |
| 3 | n01484850 great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias |
| 4 | n01491361 tiger shark, Galeocerdo cuvieri |
| 5 | n01494475 hammerhead, hammerhead shark |
| 6 | n01496331 electric ray, crampfish, numbfish, torpedo |
| 7 | n01498041 stingray |
| 8 | ... |

As you can see, we have a unique identifier followed by a space, some class labels, and a new-line. Parsing this file line-by-line is straightforward and efficient using Python.

First, we load the class label rows from disk into a list. To do this we strip whitespace from the beginning and end of each line while using the new-line (‘ \n ‘) as the row delimiter (Line 17). The result is a list of IDs and labels:

|  |  |
| --- | --- |
| Deep Learning with OpenCV | |
| 1 | ['n01440764 tench, Tinca tinca', 'n01443537 goldfish, Carassius auratus', |
| 2 | 'n01484850 great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias', |
| 3 | 'n01491361 tiger shark, Galeocerdo cuvieri', |
| 4 | 'n01494475 hammerhead, hammerhead shark', |
| 5 | 'n01496331 electric ray, crampfish, numbfish, torpedo', |
| 6 | 'n01498041 stingray', ...] |

Second, we use list comprehension to extract the relevant class labels from rows by looking for the space (‘ ‘) after the ID, followed by delimiting class labels with a comma (‘ , ‘). The result is simply a list of class labels:

|  |  |
| --- | --- |
| Deep Learning with OpenCV | |
| 1 | ['tench', 'goldfish', 'great white shark', 'tiger shark', |
| 2 | 'hammerhead', 'electric ray', 'stingray', ...] |

Now that we’ve taken care of the labels, let’s dig into the ***dnn*** module of OpenCV 4:

|  |  |
| --- | --- |
| Deep Learning with OpenCV | |
| 19 | *# our CNN requires fixed spatial dimensions for our input image(s)* |
| 20 | *# so we need to ensure it is resized to 224x224 pixels while* |
| 21 | *# performing mean subtraction (104, 117, 123) to normalize the input;* |
| 22 | *# after executing this command our "blob" now has the shape:* |
| 23 | *# (1, 3, 224, 224)* |
| 24 | blob = cv2.dnn.blobFromImage(image, 1, (224, 224), (104, 117, 123)) |

Taking note of the comment in the block above, we use ***cv2.dnn.blobFromImage*** to perform mean subtraction to normalize the input image which results in a known blob shape (Line 24).

We then load our model from disk:

|  |  |
| --- | --- |
| Deep Learning with OpenCV | |
| 25 | *# load our serialized model from disk* |
| 26 | print("[INFO] loading model...") |
| 27 | net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"]) |

Since we’ve opted to use Caffe, we utilize cv2.dnn.readNetFromCaffe to load our Caffe model definition prototxt and pre-trained model from disk (Line 27).

If you are familiar with Caffe, you’ll recognize the prototxt file as a plain text configuration which follows a JSON-like structure — I recommend that you open bvlc\_googlenet.prototxt from the “Downloads” section in a text editor to inspect it.

Now let’s complete a forward pass through the network with ***blob*** as the input:

|  |  |
| --- | --- |
| Deep Learning with OpenCV | |
| 28 | *# set the blob as input to the network and perform a forward-pass to* |
| 29 | # obtain our output classification |
| 30 | net.setInput(blob) |
| 31 | start = time.time() |
| 32 | preds = net.forward() |
| 33 | end = time.time() |
| 34 | print("[INFO] classification took {:.5} seconds".format(end - start)) |

It is important to note at this step that we aren’t training a CNN — rather, we are making use of a pre-trained network. Therefore we are just passing the blob through the network (i.e., forward propagation) to obtain the result (no back-propagation).

First, we specify ***blob*** as our input (**Line 30**). Second, we make a ***start*** timestamp (**Line 31**), followed by passing our input image through the network and storing the predictions. Finally, we set an ***end*** timestamp (**Line 33**) so we can calculate the difference and print the elapsed time (**Line 34**).

Let’s finish up by determining the top five predictions for our input image:

|  |  |
| --- | --- |
| Deep Learning with OpenCV | |
| 35 | *# sort the indexes of the probabilities in descending order (higher* |
| 36 | # probability first) and grab the top-5 predictions |
| 37 | idxs = np.argsort(preds[0])[::-1][:5] |

Using NumPy, we can easily sort and extract the top five predictions on Line 37.

Next, we will display the top five class predictions:

|  |  |
| --- | --- |
| Deep Learning with OpenCV | |
| 38 | *# loop over the top-5 predictions and display them* |
| 39 | for (i, idx) in enumerate(idxs): |
| 40 | # draw the top prediction on the input image |
| 41 | if i == 0: |
| 42 | text = "Label: {}, {:.2f}%".format(classes[idx], preds[0][idx] \* 100) |
| 43 | cv2.putText(image, text, (5, 25), cv2.FONT\_HERSHEY\_SIMPLEX, |
| 44 | 0.7, (0, 0, 255), 2) |
| 45 |  |
| 46 | # display the predicted label + associated probability to the |
| 47 | # console |
| 48 | print("[INFO] {}. label: {}, probability: {:.5}".format(i + 1, |
| 49 | classes[idx], preds[0][idx])) |
| 50 |  |
| 51 | # display the output image |
| 52 | cv2.imshow("Image", image) |
| 53 | cv2.waitKey(0) |

The idea for this loop is to (1) draw the top prediction label on the image itself and (2) print the associated class label probabilities to the terminal.

Lastly, we display the image to the screen (**Line 52**) and wait for the user to press a key before exiting (**Line 53**).

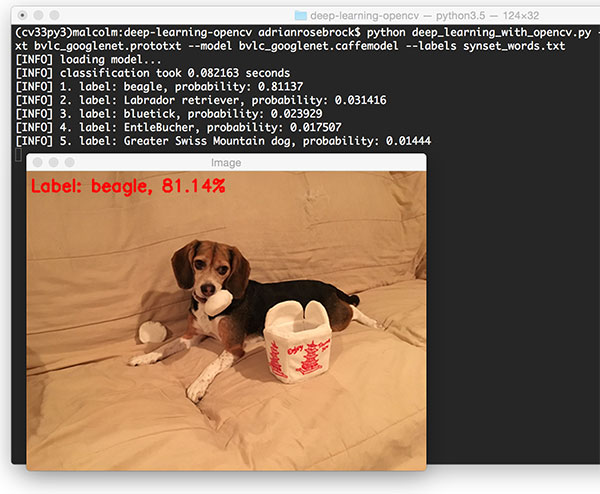
# Deep learning and OpenCV classification results

Now that we have implemented our Python script to utilize deep learning with OpenCV, let’s go ahead and apply it to a few example images.

Make sure you use the “Downloads” section of this blog post to download the source code + pre-trained GoogLeNet architecture + example images.

From there, open up a terminal and execute the following command:

|  |  |
| --- | --- |
| Deep Learning with OpenCV | |
| 1 | **$ python deep\_learning\_with\_opencv.py --image images/jemma.png --prototxt bvlc\_googlenet.prototxt --model bvlc\_googlenet.caffemodel --labels synset\_words.txt** |
| 2 | *[INFO] loading model...* |
| 3 | *[INFO] classification took 0.075035 seconds* |
| 4 | *[INFO] 1. label: beagle, probability: 0.81137* |
| 5 | *[INFO] 2. label: Labrador retriever, probability: 0.031416* |
| 6 | *[INFO] 3. label: bluetick, probability: 0.023929* |
| 7 | *[INFO] 4. label: EntleBucher, probability: 0.017507* |
| 8 | *[INFO] 5. label: Greater Swiss Mountain dog, probability: 0.01444* |



In the above example, we have Jemma, the family beagle. Using OpenCV and GoogLeNet we have correctly classified this image as “beagle”.

Furthermore, inspecting the top-5 results we can see that the other top predictions are also relevant, all of them of which are dogs that have similar physical appearances as beagles.

Taking a look at the timing we also see that the forward pass took < 1 second, even though we are using our CPU.

Keep in mind that the forward pass is substantially faster than the backward pass as we do not need to compute the gradient and backpropagate through the network.

Let’s classify another image using OpenCV and deep learning:

|  |  |
| --- | --- |
| Deep Learning with OpenCV | |
| 1 | **$ python deep\_learning\_with\_opencv.py --image images/traffic\_light.png--prototxt bvlc\_googlenet.prototxt --model bvlc\_googlenet.caffemodel --labels synset\_words.txt** |
| 2 | *[INFO] loading model...* |
| 3 | *[INFO] classification took 0.080521 seconds* |
| 4 | *[INFO] 1. label: traffic light, probability: 1.0* |
| 5 | *[INFO] 2. label: pole, probability: 4.9961e-07* |
| 6 | *[INFO] 3. label: spotlight, probability: 3.4974e-08* |
| 7 | *[INFO] 4. label: street sign, probability: 3.3623e-08* |
| 8 | *[INFO] 5. label: loudspeaker, probability: 2.0235e-08* |



OpenCV and GoogLeNet correctly label this image as “**traffic light**” with 100% certainty.

# Summary

In this exercise we learned how to use OpenCV for deep learning.

With the release of OpenCV 4 the deep neural network ( dnn ) library has been substantially overhauled, allowing us to load pre-trained networks via the Caffe, TensorFlow, and Torch/PyTorch frameworks and then use them to classify input images.

I imagine Keras support will also be coming soon, given how popular the framework is. This will likely take be a non-trivial implementation as Keras itself can support multiple numeric computation backends.