ML/AI

CONVOLUTIONAL NEURAL NETWORK FOR CLASSIFYING YOGA POSES

I finished the Deep Learning Specialization on Coursera! Obviously, Andrew Ng has a track record of being a fantastic instructor, and it's a great introduction to the entire field and includes lots of fun projects. However, you realize by the end that you still have a ***lot*** to learn! So, hours later, I embarked on my first deep learning project - building a simple convolutional neural network (CNN) with Keras for classifying yoga poses.

I wanted to build everything from scratch. I did almost immediately find a knowledge gap. All of the courses that I've taken come with nice, ideal data sets and do a lot of the initial data cleaning work for you. We were always glossing over how to collect and structure data that you might want to use.

Now, I could have gone and just played with another nice dataset that already existed (Keras has a nice list here or you can look at datasets on Kaggle) to save time and energy. But I wanted to make sure I was using the proper data augmentation techniques and, as I added data and played around with optimization, that I understood every piece, which includes being very familiar with my dataset.

I decided to build a yoga pose classifier to be able to predict which of 10 yoga poses the image shows. I'm really bad at yoga, but it was fun and there are a lot of images with friendly licenses for me to use (Google Images and Flickr both let you filter by license).

Building a Convolutional Neural Network

Two resources were very helpful for getting me started on this project:

Building an Image Classifier using Deep Learning via becominghuman.ai

and

Image Classification using very little data from the Keras blog

However, neither were "plug and play", and were convolutional neural networks for binary classification. I recorded below some of the questions I asked myself while doing this quick project, and the edits I made to make it easy for me to test and build upon this foundation. Here's my complete code on Github.

DATASET

We're building a multi-class convolutional neural network, so let's collect the data and figure out how to feed it into the CNN.

Folder structure

I created a folder called /training_set/ and /test_set/ (Keras blog uses /data/ with a /validation/ and /train/ folder inside).

Within each folder, I have a folder for each class - so a folder for bridge, mountain, plank, tree, warrior1, and more - in both the training_set and test_set folder.

cnn_multiclass.py

cnn_multiclass_predict.py

/training_set/

  /bridge/

    TrainBrFile1.jpg

    TrainBrFile2.jpg

    ...

  /mountain/

    TrainMoFile1.jpg

    TrainMoFile2.jpg

    ...

  ...

/test_set/

  /bridge/

    TestBrFile1.jpg

    TestBrFile2.jpg

    ...

  /mountain/

    TestMoFile1.jpg

    TestMoFile2.jpg

    ...

  ...

What image types can I put in?

I spent some time converting everything to jpg because I couldn't find much in the Keras documentation on what types of images I could use, and every example online just uses .jpg. However, you don't need to do that! Within /keras/preprocessing/image.py, there's a white_list_formats = {'png', 'jpg', 'jpeg', 'bmp', 'ppm'}, so Keras accepts all these file types.

I had a mix of png, jpg, and jpeg, so I was able to leave it alone. However, if you have other types of images, this person changed white_list_formats to allow Keras to accept a tiff file.

How many images should you have in the training set vs test set?

Here, I defer to Andrew Ng's lecture video on "Size of the Dev and Test sets" in the Structuring Machine Learning Projects course on Coursera.

The "old way" of doing things was a split of 70% training, 30% test, or 60% training, 20% test, 20% dev. This was for smaller data sets. However, if you have a much larger dataset, a split that's more like 98%/1%/1% is reasonable.

Because I do have a smaller data sets (I collected images on my own, so I only had about 1000 total images for 10 classes), so I did do the old way, ~75%/25% split for training and testing.

DATA AUGMENTATION

While the goal of this project was to just get familiar building everything from scratch, I did want to use the augmentation techniques because the set really was small.

When should data augmentation happen?

The Keras blog and documentation showed precisely that you should just augment the training data (here, we split the data at the beginning, but didn't augment the test at all). Here's another explanation on StackExchange.

MODEL AND TRAINING

My biggest mistake here was not immediately realizing one of the documented examples forgot to divide by batch_size, and the steps_per_epoch and validation_steps were hardcoded for the number of images in the training and test sets, and I was running way too many steps for the data set I have. Oops! Took way longer and gave me some weird results.

For our last layer, since we have 10 classes, we use *units = 10*

classifier.add(Dense(units = 10, activation = 'softmax'))

To have multiclass softmax classification for the convolutional neural network, use loss = 'categorical_crossentropy' in classifier.compile(), and the class_mode in flow_from_directory() to have class_mode = 'categorical'.

SAVING THE MODEL

I want to run the predictor separately because I didn't want to retrain the model every single time I wanted to test it with a new image. So instead of adding the predictor to the end of the same file (cnn_multiclass.py), I split it out into a different file - cnn_multiclass_predict.py.

So when we are done training the model, the last line of the code is:

classifier.save('my_new_model.h5')

Remember to import!

from keras.models import load_model
import h5py

Nothing worse than training the CNN for a while and then the model fails to save because you forgot to import the package. I speak from experience.

PREDICTOR

Now that we have a model saved, we can just load it into our cnn_multiclass_predict.py file. To do so we need import the packages we need and use load_model to retrieve our saved model.

classifier = load_model('my_new_model.h5')

Then you can retrieve the image you want to test with, and run predict() on the classifier

result = classifier.predict(test_image)

This outputs a numpy array of predictions, like

[[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]]

This means that the convolutional neural network is predicting that this test_image is the 4th label (alphabetical). You can switch out the images and keep testing, without retraining the model.That's it! You can find the full code on Github and the data set here.

After training for 50 epochs, I got an accuracy of about 95% (of course, tiny training set, tiny test set!) The images were fairly clean and simple, and I spent time cleaning them by hand (for example, side shots of mountain pose were discarded, only kept pictures from the front), but it just shows that you can get a simple convolutional neural network working reasonably well very quickly.

Edit Dec 2019: I found this project from Stanford's Deep Learning on campus course that used this post and data and improved on it! Check out their improved results here: http://cs230.stanford.edu/projects_winter_2019/reports/15813480.pdf

Interested in taking this Specialization but not sure how much background you need? Check out my blog post on "How much math and programming do you need to take the Deep Learning Specialization on Coursera?"

LATEST POSTS

QUANTUM COMPUTING
QUANTUM TELEPORTATION TUTORIAL WITH OPENQASM

QUANTUM COMPUTING
QUANTUM SECURE CRYPTOCURRENCIES | QRL, MOCHIMO, IOTA, CARDANO

INFORMATION
about
contact for business inquiries

FOLLOW ME
youtube
instagram
twitter
tiktok