Douglas Richardson

Simple Pose Recognition

With Small Datasets

Introduction:

For this project I decided to tackle the problem of pose recognition using machine learning on a deliberately small scale. While pose recognition is a problem that is well explored, most of the time if machine learning is involved it uses massive databases of information. Hundreds of thousands of images classified and gathered in a way that makes the overall program efficient for a general case. The issue with these methods is that they require extremely high amounts of data to create. My goal with the project was to create a program that could recognize a single pose in an image with a database that was so small I could create it myself in only a few hours. The idea behind this is that if the program is intended to only work for a small number of cases, it will only need to be given a small dataset. Knowing this, because the number of images in the dataset are so small, in order to create a program that will be properly able to recognize images, I will need to modify the images in a particular way to maximize the amount of helpful information in each image to identify the pose.

The method I will be using involves first finding the parameters to maximize the accuracy of the program on the images as they are, then gradually add filters onto the images to minimize the amount of irrelevant data. Every step of the way I will keep track of the accuracy of the program, and use the same original images for testing and validation.

Background:

Pose recognition is not something new. There are many libraries, programs, apps and devices that use pose recognition. Famously the microsoft kinect was used as a gaming application of object and pose recognition. Almost all sources that use machine learning for the purposes of image and pose recognition do so on astoundingly large databases. Hundreds of thousands of images large. For this project, we were going small. To create a program that was only expected to predict a small range of images, however at the same time, only had a small set of images in it's database. The idea being, by forcing the constraint of a small database, it forces the application of smart use of computer vision technology, such as filters and other forms of image manipulation, in order to create a neural network that is able to accomplish what it has been set out to do.

Approach:

My approach is simple. Start with a basic dataset and send it through a neural network. This would give me a network that is able to identify one pose over another. At that point, the network is unlikely to be much better than guessing. Without doing much feature extraction or modifying the dataset in a way that emphasizes the pose it is trying to estimate, it is unlikely the network will be very good at identifying the proper pose. From there I will tinker with the layout of the network to try and get it as good as possible, Altering the number of nodes and perhaps adding or removing layers. After I have found what appears to be the most optimal setup for the neural network, then I start adding filters to the dataset that simplify the images.

For this project I utilized a few different libraries for both image processing and machine learning. The most general library I used is numpy, a mathematics library focused on many applications for mathematics in computer science, although I primarily use it in my code to manipulate matrices. Numpy is also a requirement for a few of the other libraries. For image processing I use opencv2 for python. OpenCV is a computer vision library and includes many useful functions such as the ability to apply grayscale, blur, and several other functions I utilize in my program to simplify the dataset for the neural network. I also use large sections of the Kreas library. Kreas is a machine learning library that boasts a large number of applications, but most importantly it was the backbone the application of machine learning in this project.

The code that I used is broken up into multiple different programs each serving a different purpose. I will go over each of these programs and explain their purpose, On the way I will clarify different functions that I will use, and mention if such a function was used in a previously mentioned program.

The programs Binary_Image_Classification Video Viewer, Frame parser and Image Scrambler are all simple programs used to make it easier for me to create the dataset I used for the rest of the programs. I will leave brief descriptions on the Github pages for each of them explaining them in more detail, but they are not interesting enough on their own to warrant much discussion.

Before we get into the machine learning part of this machine learning project, I will get into the computer vision part of this computer vision project. The program that goes by the name Binary_Image_Clasification_Pre_Filterer is there to add a filter onto a test and validation set before it is used to train a neural network. It starts with two variables i and k both set to zero but. The i variable is used to track which image we are looking for and the k variable changes what directory we are in. If k is even it is the std or standard directories (for images that do not have a pose in it), and if k is odd then it is in the pose directories (for images that do.) If k is 1 or less we are looking at the test images and if it is strictly greater than 1 we are looking at images in the validate set. Either way this is just to ensure all directories are reached. The max_count is the largest enumeration of each type of image (pose or standard) that we will run into and is configured differently if it is looking for pose images or standard images. Either way the program sorts through each image it finds, applies the selected filters, then writes them to the appropriate place in the output. In the version I have right now it has the filters of a grayscale filter followed by a median blur and a laplacian for the purposes of edge detection. This section of code changed as I added or removed filters to attempt to optimize the filters used.

The last program I will be talking about is the Binary_Image_Classification program itself. It starts by defining classifier as a Sequential from kreas.models. This refers to the type of

method it uses internally to represent the network. Next we add 2d convocation to that network. The shape of the convocation changed often through development as I tinkered with the neural network. I flatten the classifier and add several different layers, most importantly ending with a sigmoid with a single unit to make the network output only a 0 or a 1 to determine if it is detecting the pose or not. I compile the classifier with the adam optimizer which is a stochastic gradient descent, meaning as new data points are added it optimizes based on the gradient of changes. The loss function is binary cross entropy, something I truthfully don't understand too well, however showed the best results. I also have it display accuracy by adding it as a metric.

The image generators in the next part of the code are there to create even more images from the limited dataset available. They will flip, rotate, scale, and blur the images and add them as additional data points. After all is said and done, I fit the neuralnetwork to the set. I use 5 epochs because each epoch takes a long time to complete, but one is not reliable enough. Afterwords I save the classifier as a .h5 so I can re-use it later without needing to compile everything again. It is important to note this only saves the weights, which is why in the final test program there is some code taken directly from this program that is largely unchanged. They create the same neural network, it just needs to load the weights for each node.

Dataset:

Creating the dataset was the bulk of the work on this project as I had come across many problems when making it. First of all, this is an image dataset that I am creating using video. The idea is that the program shouldn't need previous frames to identify the pose, and to make it easier to get a larger dataset with what little I have, I would record the situation the program is designed to recognize, then separate the video into the individual frames. Because of this, After seperating the video into various frames, I would need to manually sort which frames contained the pose I am attempting to identify and which do not.

One of the biggest issues I ran into while creating the dataset is ensuring that I set it up in a way that is consistent. When I first made the dataset, there were quite a few problems that ended up meaning I decided to scrap the old dataset and make a new one. First of all, when I recorded the dataset, I did so in two parts, and there were slight changes in the background for each part. Not only that, but because of the color of the background, it became difficult to differentiate different key areas for the pose (such as my arms) and the background. To solve these problems I ended up wearing clothing that contrasts with the color of the background, and moving some of the objects to minimize the amount of background clutter.

In the end my setup looks something like this: I start by placing a camera on the top of my monitor for my desktop computer. The background is mostly plain, as I have removed everything from the walls behind me. Next I set the camera recording and work on my computer for about thirty minutes to an hour. Occasionally I will perform the pose I am attempting to recognize, making sure I am at least mostly on shot for the camera. After I have finished recording, I take the footage and watch it back, pausing and marking down the frames that I can

identify the pose in. The camera I am using records at thirty frames per second in 720p, so the images are quite clear at that resolution, and every minute of footage is about 1800 images. Over the course of half an hour, thats 54,000 images. Most of those images are nearly the same, so when I sort the images into pose or no pose, I generally only include every 100th frame, meaning our dataset is instead of 54,000 images, only 540. I do this to prevent overfitting, as if two frames that are identical that end up in both the validation set and the testing set would be bad for the network.

Evaluation:

To setup my project, I started by getting a proper camera and a place. I knew that by the nature of the project, the location of the camera and the quality of the camera needed to be consistent. My original plan was to record myself in front of my computer, occasionally performing a pose to the camera. After placing the camera and recording myself (mostly doing schoolwork) for about an hour to an hour and a half, I ran through the footage as discussed in further detail in the section above.

The first dataset ended up with around 1,600 images. 375 of which were identified (by me) to be the pose I was attempting to identify. After I had finished this, I attempted to run it through a neural network without adding any filters on the images at all. After adjusting the network until I ended up with a result I found acceptable, the network was able to perform at about 77% accuracy, according to the validation set. My next step was to add filters onto the images to attempt to increase the accuracy of the network.

I started with a grayscale filter with a median blur. The idea being that to recognize a pose, the image doesn't really need color or sharp images, it just needs a general idea of the shape of the person in the image. It is important to note that because the poses I have chosen are mostly arm positioning poses I could get away with this. If I was using poses that included finger positioning it may not be as simple. This filter had no significant change to the accuracy of the network.

My next filter, one that I was anticipating would have more success was to add a laplacian filter to perform basic edge detection. The idea being that since it only needs the shape to identify the pose, everything else doesn't matter. When I applied this filter to the images and

ran it through the neural network, the accuracy dropped significantly, and it didn't take long for me to discover why.

A significant problem I discovered was that while I was easily able to identify my arm position in the images, filters and neural networks would have a difficult time with it since my arms are nearly the same color as the background color of the images. Recognizing this, I decided to create a new database, this time wearing a dark jacket that contrasts with the background.

My second dataset ended up having even fewer images than the previous one. A meager 700 images. Not many, but it would have to do. This dataset I was more vigilant about performing the pose often, so the a larger proportion of those images was the proper pose than previously. With such a small dataset however, now more than ever was I worried about the issue of overfitting.

After running the images through the neural network, I was able to get an accuracy of 98% without adding any filters, and while the grayscale filter appeared to lower the accuracy further, adding the edge detection still read an accuracy of 98%. I was skeptical, however. The accuracy of 98% was done on a validation set consisting of frames taken from the original recording. I wasn't satisfied with the result and decided to add one final test to be sure the accuracy I was reading was correct.

I recorded one last time, probably about 5 minutes. I didn't worry about overfitting since this was just going to test the neural networks, not actually change them. Same camera, same location, this time use every frame to test rather than every 10th or every 100th. My results can be found below. You will notice that I did not include the grayscale filter, this is because even

when using the standard validation set, its accuracy was a meager 60%, so I didn't have much faith in it.

There were 4158 images in total, out of those 105 are the pose we are looking for.

Neural Network Type	False positive count	False Negative count
No Filters	36	102
Edge Detection	2	16

Because of this we can see that the true accuracy of the two networks is quite different. Without any filter, the network appears to perform horribly. It missed almost every image that contained the pose, and incorrectly identified the pose in a non-insignificant amount of images that did not contain the pose. When applying the filters to only include edges, the neural network performed quite well. With only 2 frames incorrectly identified as containing a pose and only missing 16 that did. Having said that, this still means it misses 15% of the images that contain a pose, but with the massive number of frames that did not contain a pose and only misidentifying 2, that is a result I can live with.

Conclusion

The biggest take-away from this is that small datasets can still create neural networks that function in limited circumstances if the proper steps are taken to maximize the amount of useful information the neural networks receive. When I started the project I anticipated that I may be able to expand the circumstances that the neural network functioned for by adding filters, however in the end creating a functional neural network even for a restricted number of circumstances was more challenging than I had originally thought.

If I was going to do this project again, I would try to increase the size of the dataset as well as attempt other methods of computer vision, perhaps including something to determine depth or identify the person in the frame to isolate them from the background.

References

Institution of Mechanical Engineers. Jun 07, 2011. “Kinect team scoop the £50,000 MacRobert Award.” Retrieved Apr. 23, 2018 (http://www.imeche.org/news/news-article/kinect-team-scoop-the-50-000-macrobert-award).

Venkatesh Tata, becominghuman.ai. Dec 13, 2017 “Simple Image Classification using Convolutional Neural Network — Deep Learning in python.” Retrieved Apr. 23, 2018 (https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8).