

EFFICIENT CONVOLUTIONAL NEURAL NETWORKS FOR THE USE OF GESTURE RECOGNITION AND CONTROL

Jordan Graham-Bryan

Contents

[Abstract 2](#_Toc38556984)

[Introduction 2](#_Toc38556985)

[Literature Survey 3](#_Toc38556986)

[Proposed Framework 7](#_Toc38556987)

[Discussion 7](#_Toc38556988)

[Results 7](#_Toc38556989)

[Method 7](#_Toc38556990)

[Conclusion 7](#_Toc38556991)

**This report will assume that you have at least a beginner’s notion to python, as deep learning isn’t a beginner’s course or project**

# Abstract

# Introduction

Deep learning (DL) is a huge area, so big that it would be impossible to cover everything in this report. Terms such as ‘data science’, ‘machine learning’, and umbrella terms such as ‘artificial intelligence’ will be examined and why all these different areas are so important .Unlike machine learning (ML), deep learning does not need structured data. Deep learning networks use and reply upon Artificial neural networks (ANN) which use data you introduce, such as images of dogs and cats to train itself a similar way to which the neurons in your brain work. You train the model around these data sets (the more you have the better) and the model can depict key features and differences between the two types of animals. This report will give example to the power of deep learning by using a specific artificial neural network called a convolutional neural network.

Deep learning as of late has received a lot of attention over the past decade as its uses as become more apparent. Deep learning has been around for quite some time, it was invented in the 1950’s. The reason why it’s become popular recently is because of two ‘ingredients’ which are needed in order to make a model which is accurate. The two ‘ingredients’ are data and processing power (that of the graphics card is incredibly faster). As of recent years, data has been stockpiled by big companies like Facebook and google. Google have billions of images and hundreds of data sets which are free to use to train your own neural network. This report will focus mainly around a single specific type of deep learning called convolutional neural network, which are highly affective for image processing and recognition. There are many different types of machine learning and deep learning algorithms which each have their own uniqueness and effectiveness for different needs. For example, an artificial neural network called Q-learning has completely taken over the gaming industry. It learns in a very particular way which enables it to become better than the best players in the world, within any gaming genre. A company called OpenAI has shown the world what deep learning can achieve. Games such as StarCraft and dota which are dominated by people who plays 12 hours a day 7 days a week have been bested by these deep learning models. They did this by training a neural network by having their models play against itself repeatedly for millions of years (of course the games were sped up significantly) until it knew the game inside and out. Dota is a completely complex game, currently involving 117 different and unique heroes (characters), each having 3 or more different basic abilities with a unique ultimate ability, each hero can purchase 6 out of 145 separate items which give the hero special stats or/ and affect the game in some way. With all these variables put in play, on average a player will take 600 hours to become relatively ‘decent’ with game knowledge and mechanics (keyboard and mouse control). But with deep learning we can teach an algorithm to learn a game as complex as this, not only to learn but to become better than humans which have trained every day for years on end.

Q-Learning and convolutional neural networks are very different, especially the way they use their data. This will be analysed and discussed in thoroughly in the literature review. Convolutional neural networks can in fact be used to play games, if you have a game like dota or league of legends both of which have billions of variables to win a game, a neural network which can pick up on specific pixels such as if an enemy is running to you that might signal the neural network that you are likely going to get attacked, so run away or fight back.

The use of high-end general-purpose graphics processor units have enabled these deep learning models to sniff through millions upon millions of images, text, audio, or video files many times faster than that of the fastest processor on the market. This is because graphics cards are amazing at doing one thing at a time, i.e. displaying graphics or deep learning, this is due to the fact that a single graphics processing unit may have thousands of cores, where as a CPU usually maxes out at about 12 (even though CPU cores are faster, due to the amount of cores in a graphics card it doesn’t matter, it is more than made up for with the larger number of cores and faster memory as long as the operations can be parallelized, which is the process of giving each core the same amount of data or throughput, thus maxing out the threshold of how much data can be processed at once).

Why the need for different artificial neural networks?

The reasoning behind the different kinds of neural networks are because each different layer may perform different kinds of transformations on the input, examples will be given to this later in this report. Some layers are better suited for certain tasks than others. An example would be recurrent layers will be used and more suited towards speech recognition.

# Literature Survey

**Python:**

python is a high-level programming language which has taken over the world. Python can be installed in seconds, it is lightweight and has thousands of libraries which add functionality and enable a programmer to create a program which can do anything. Python does everything that java does, with fewer lines of code, which is why over the past 3 years python has taken over java as the most popular language in the world/

**Virtual environments:**

A virtual environment is a place in which ensures that dependencies (or libraries) are kept separate from each other. For example, say someone is designing a neural network with OpenCV but they need to use an early version which coincides with their current version of TensorFlow, but on another project they are using OpenCV on its own and they need to the newest version, if they install both versions, python will automatically use the newest version of OpenCV causing problems for the persons neural network project. A Virtual environment enables this person to have 2 ‘separate’ environments with their own libraries and dependencies. Another benefit of a virtual environment is that if the creator somehow breaks his program such as installing the wrong dependencies or unable to uninstall multiple versions of the same library ( a common issue with tensorboard and TensorFlow), then they can delete this virtual environment and start again, where as if you installed via pip to your machine, worst case scenario would be a pc refresh.

**Pip:**

PIP is a package installer. By opening up the command prompt (cmd), you can install multiple dependencies and libraries for which you can use in when you open python through cmd. You can also upgrade or uninstall them from this terminal as well. It is easy to use, and to install.

**Anaconda:**

Anaconda is a free open source distribution of the most used programming languages in deep learning (R and python). Once downloaded you can create a virtual environment with ease. PIP can be used in the terminal, to install the dependencies needed such as TensorFlow and Keras. Anaconda is very popular because once you create your virtual environment, you can use several different text editors and applications to use python with, down to your custom favourite. One of the most popular is Jupyter notebook.

**Text editors, Integrated development environments (IDE’s) and Jupyter notebook:**

Python once downloaded can be used in any command prompt or terminal, this is usually fine for any experience programmer, who knows how to debug their code and does not need an application to provide help for them. However, typically you would want something which offers more support. You would not use a text document to write a report? Text editors are one up from terminals, a well-known example is notepad++, once you add the ‘PATH’ to the notepad file, any dependencies installed through cmd will then enable you to add it to your python file in notepad++. A huge downside to using text editors are that you can’t check if you make a mistake along the way unless you run your code, such as spelling or grammar mistakes. There are ways around this such as installing python addons, to check your code but usually it is easier to install and use an IDE.

integrated development environments are the best applications to use when programming, for several reasons. Each IDE (integrated development environment) are different in some way shape or form, but they all have the same sort of gimmick and that is to provide the programmer with basic help. For example, checking grammar (one capital letter where it should be will mess up your entire code). The most popular IDE for python is PyCharm. PyCharm is easy to use and every time you create a new python folder, it automatically creates its own virtual environment, not also that but it allows the user to install which version of the library they want with a fully integrated user interface, instead of having to use the terminal.

Jupyter notebook takes a slightly different route when it comes to IDE’s. Jupyter notebook is a web application which allows you to write your code line by line and allows you to test it to make sure there are no errors along the way. When writing big programs such as those which are created for neural networks, this can enable less confusion when writing it. For example, when wanting to see what an image or data set will look like, with some altercations (such as changing the picture resolution) you would have to wait until the end of a program for it to be printed out on the console. Whereas, Jupyter notebook prints it out for that specific line of code, making it a lot easier for the programmer(s) to design something which is easy to read and understand.

**Deep Learning**

Deep learning is a specific term which relates to machine learning but limited to the many forms of neural networks. A neural network in its simplest form is a series of algorithms which set out to recognize correlations in a set of data. It works the same way our neurons in our brain works, interconnecting to form an idea or image, or in this case a result. Their uses are endless, they can be used for image recognition and gaming, creating a perfect ‘player’ which can analyse (in real-time) images of the screen to detect enemies resulting in an action to occur. Neural networks current uses are varied, in electronics they are used in multiple different cases such as: analysis of chip failures, machine vision, voice synthesis and prediction of the code sequence. Even in manufacturing they have multiple uses such as product design and analysis, chemical product design analysis, project building and many more.

**TensorFlow**

TensorFlow is a very popular mathematics library which is used a lot in the field of deep learning. Since it adds a lot of value to how the programmer decides to build his or her model, it also enables a lot of other different packages to be installed which run on top of itself such as Keras or tensorboard (which is installed with TensorFlow automatically).

**Tensorboard**

This library comes pre-installed with TensorFlow, it enables your machine learning or deep learning algorithm, when it processes the data you input, to be displayed on your local machine, which you can access via your web browser, it gives you valuable information which you can change and you transform your model in a way which best suits you. It is a valuable tool which provides the measurements needed during the workflow from a machine learning model. It allows you to track experiment metrics like accuracy and loss, giving you a visualising method (through graphs)

**Tensorboard**

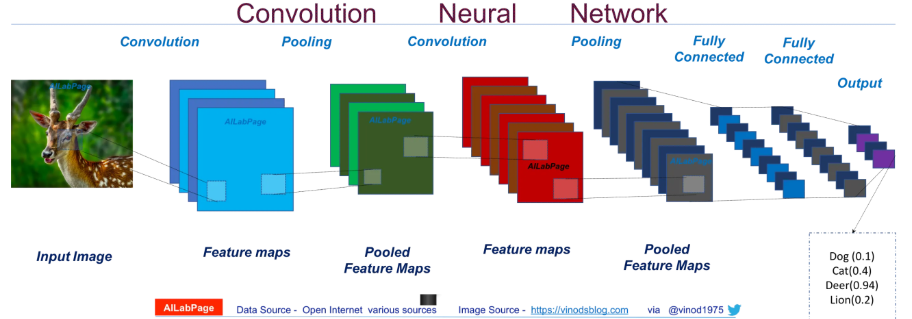
This library comes pre-installed with TensorFlow, it enables your machine learning or deep learning algorithm, when it processes the data you input, to be displayed on your local machine, which you can access via your web browser, it gives you valuable information which you can change and you transform your model in a way which best suits you. It is a valuable tool which provides the measurements needed during the workflow from a machine learning model. It allows you to track experiment metrics like accuracy and loss, giving you a visualising method (through graphs)

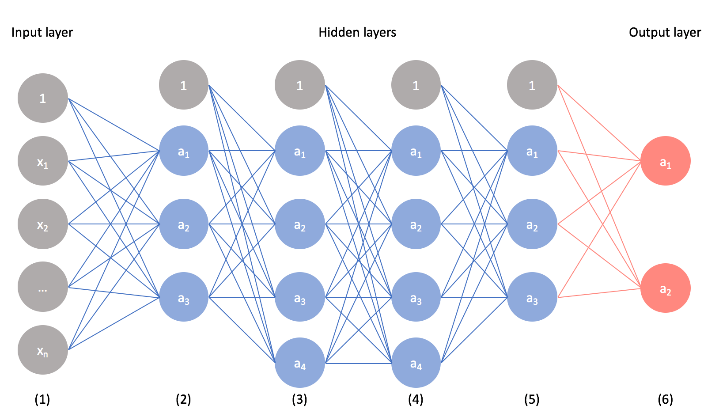
**Neural networks**

A Neural network is broad term which covers many different types of similar algorithms for deep learning, but each one has a more suited definition towards a specific problem. Two examples of neural networks are convolutional neural networks and Q-learning. Convolutional neural networks are by far the most popular for multiple reasons, such as ease of learning and use and the fact that there is not a lot of complex mathematics involved whereas with other deep learning techniques there are.

**Convolutional Neural Networks (CNN’s)**

Image recognition has become so popular over the last decade, allowing CCTV to recognize someone in a low quality video, allow a car to drive itself based on what a camera picks up in its front and rear view or even to play a game. CNN’s are the most known neural networks (not by name but by its use in image recognition).

Picture 1a. How a neural network learns.

The CNN algorithm takes an image and assigns weights and biases (variables which the algorithm uses to differentiate the image from others and identify similarities between this image and other ones) to the image. Compared to other classification algorithms, that of the convolutional neural requires less pre-processing. The image is fed through a series of connections (or neurons which are inter-connected) to one and other, they are often referred to as nodes or neurons as of the way which they represent the neurons in our brain.

The beauty of using CCN’s over other types of artificial neural networks is that they can be run on any device, from high end computers, to phones, to microprocessors such as the raspberry pi

**Input layer**

The first set of notes in a convolutional neural network are collectively called ‘input layer’. This is where the input data is given to the algorithm, therefore the number of nodes is completely dependent on how you input the data. For example, it may only take one single node instead of several, then passed onto the next set of nodes. Or if you are analysing multiple different sets of data/ input images then you may have several nodes in the input layer. in order to understand this layer, you must know and understand what ‘features’ are.

**Features**

A feature is something which the model will use in order to classify an image, so if you are training a model on dogs, one feature might be something in the image which all dog images have. What makes convolutional neural networks the best at what it does is that this artificial neural network assigns features to the training data you use without human interaction. This is known as unsupervised learning. Now although convolutional neural networks can be done using supervised learning, such as manually going through the data training set and labelling each image, telling the model what it needs to look for in images in order to classify the image.

**Hidden layer**

Once the input layer has decided what to do with the data, it is then sent to the next layer(s), commonly referred to as the hidden layer, as in input format. Every node is interconnected to the next following node, this makes it so that data can be sent to the corresponding node in the next layer. For example, input a picture of a dog input a dog/cat classifier, this would require the input layer to collect the data and then sent it to the next layer, but to its corresponding node. For example, one node may be used classify the image via tongue (as dogs tend to have their tongues out of their mouth more), then this node will determine whether or not to send it the next layers set of nodes, depending on colour of fur, shape of paw, size of the animal, etc….

In technical terms, each connection from one unit to another, will have its own unique weight which is a number between 0 and 1 (w = 0.3). Weights are what represent the strength between the connection of each unit. When you first receive an input, to the input layer, this input of data is then passed onto the next unit via the connection, and the input will be multiplied by the weight assigned to this connection. A weighted sum is then computed with each of the connections that are pointing to this neuron, the sum is then passed to an activation function which transforms the result to a number between 0 and 1. (output = activation(weighted sum of inputs)). Once the sum is received from the transformation from the activation function, this will then be passed onto the next neuron in the next layer. this process, while not overly complicated but can take some time getting used to, is repeated until it gets to the output layer. This also means that the weights for each process be change continuously in efforts to reach optimised weights from each connection as the model continues to learn from the data.

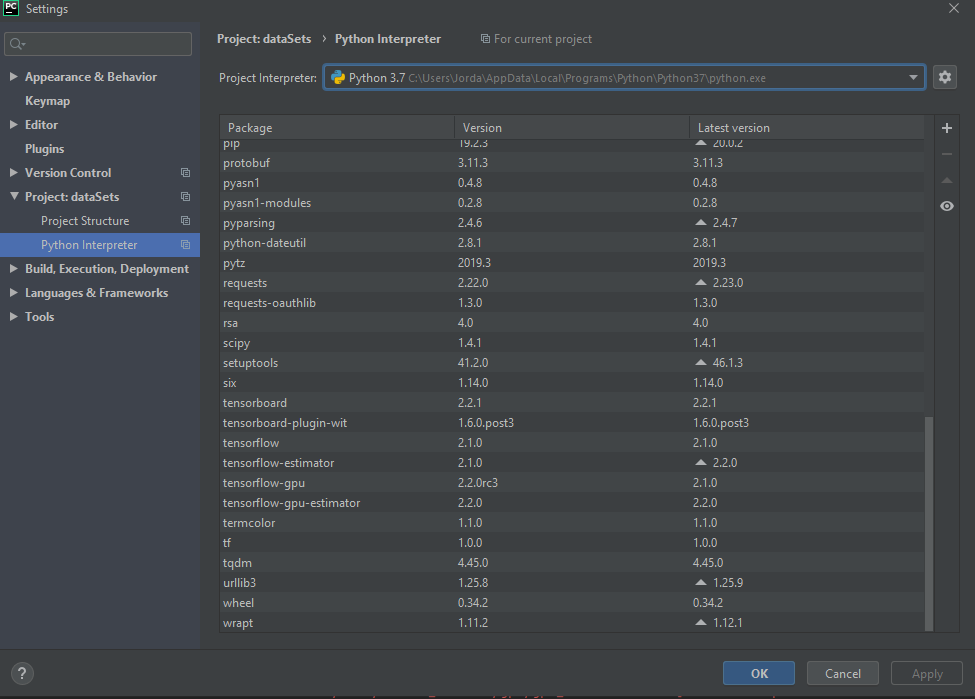
**Output layer**

The output layer has any number of nodes, it is dependent on the number of classifications your model runs. For example, if it is a cat and dogs’ classifier it will have 2 nodes in the output layer, representing which one the image is. If it is a hand gesture recognition, then you would have ‘x’ number of nodes corresponding with the amount of gestures you want to the model to recognise. You can of course add to the model if you want it to classify more images.

# Proposed Framework

This section of the report will detail how I trained and tested my CNN and how the code works. Firstly, I needed an IDE or text editor to choose in order to create my virtual environment. So, I used PyCharm for this project as its incredibly easy to use and download the dependencies and libraries which are needed to create a convolutional neural network. After installing PyCharm, when you create a new folder, you can choose whether to create a new virtual environment or you can choose an existing environment, which is very helpful when you have any pre-required libraries that are quite large and take time to download

**The zip file which this report was sent in, will contain the code with comments, detailing every single line. This report will detail only sections of the code, for better understanding please open the python files to see.**

Once you have all the libraries installed you can double check in the settings and under python interpreter, which shows the version of python you are using (which is helpful if you are using libraries which only work with a specific version of python, such as TensorFlow 2.0 which only works with python 3.7), it also shows all the packages/libraries which you have installed too. As in the picture to the right.

The first step in programming a CNN is loading the data in a way which the model (which you later create) can understand. I used my own custom dataset of hand gestures. There are quite a few different datasets of gestures out there, but instead I decided to take around 800 images of me doing two types of gestures. The thumbs up gesture and the ok gesture.

Imports and libraries are as followed:

**NumPy**: this package is usually imported for scientific computing, very useful for deep learning.

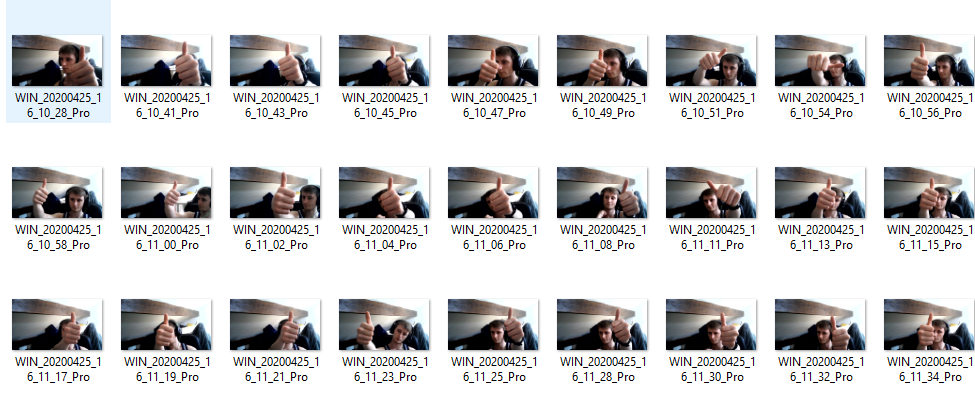
**Matplotlib**: Package which allows the user to plot data on graphs, particularly useful in deep learning

**OpenCV** (open source computer vision): a very popular library in deep learning and out of deep learning. A package for image processing and manipulation.

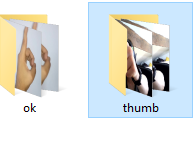
**TensorFlow**: perhaps one of the most known libraries in deep learning. TensorFlow in its simplest form is a maths library but offers a lot of functionality which is helpful in deep leaning. A good habit is to import TensorFlow as tf, this is because instead of typing out TensorFlow every time, you only need to type tf.

**Keras**: Keras is a neural network library, which can run on top of TensorFlow and other libraries in order to create an efficient neural network. It is user-friendly which means that the code is easy to understand and learn.

**My Data**

For my project I decided that I was going to make my own dataset, this is not necessary there are tones of data sets out there for whatever model you want to create. I did this by tediously taking 400 or so pictures of me doing one gesture, then 400 pictures of me doing another gesture. It doesn’t matter as long as youre sure that the data isnt corrupted.

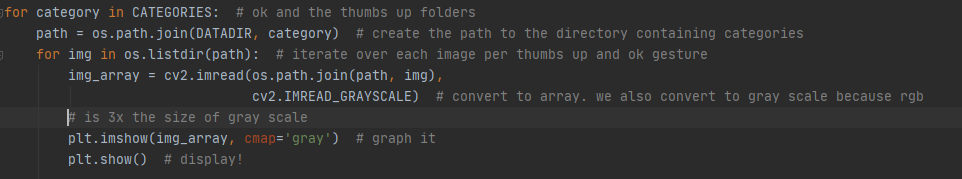
**Loading in data**

After importing these libraries in out code, we can now focus on importing our data. As stated, I have created a custom data set, this is what will be used for this specific neural network. The way in which this data will be presented to the neural network is very specific, you must call the location which the folders of your data is in and you should separate each classification you want. For example, I want to classify 2 different hand gestures, therefore I will create 2 different folders which I will put the two different types of gestures I want the neural network to know and differentiate.

As you can see with the way I have set out my data.

 Here is the parent folder, holding the testing data

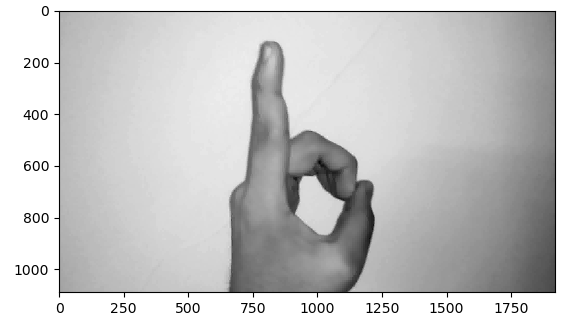
Here are the testing folders, labelled as categories so I can recall them later in the code.

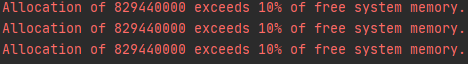
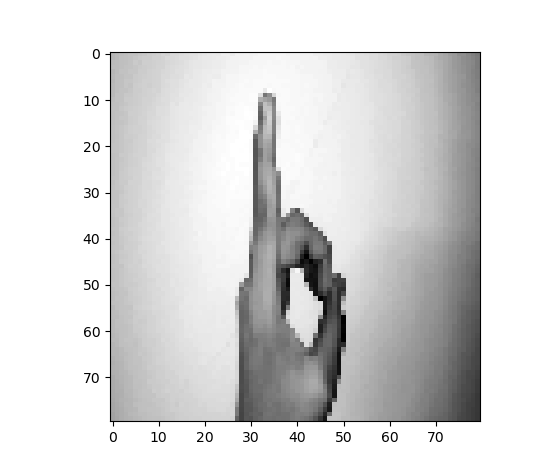


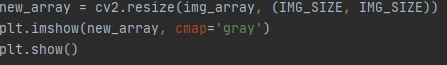
Now that the data has been imported into our code, we can now do as we please with it. Here I create the for loop which creates the path to the directory containing the categories. In order to make sure the code is **efficient**, this loop will read the images in Gray scale, this isn’t necessary but when dealing with larger data sets which contain thousands of images it makes your code more efficient because RBG coloured photos are 3x the size of grey scale. Also, this data will be displayed in a graph.

Then to show the image we just print it:

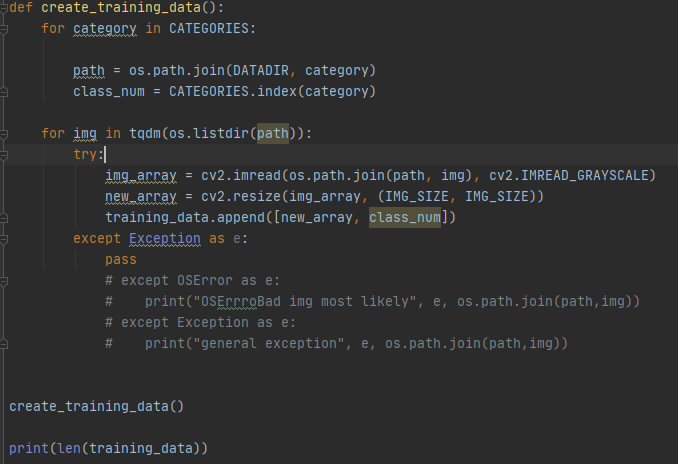
Print(img\_array)

Print(img\_array.shape) There is one problem with this photo, as indicated with the numbers on the y and x axis, the image is 1000 x 1800 pixels, this is fine when its just a few pictures. However, when dealing with deep learning you tend to need more than just a ‘few’ pictures. If we leave the images at their original size, this will cause the program to exceed the memory which is allocated to the model. As shown here.

the way in which this can be dealt with is by changing the image size, we want it extremely lower than what it is now because the model needs to load all of these images in. this process is done by changing the value we give the IMG\_SIZE, I changed the image to 80x80. Not only does this resize the image but it also lowers the quality. The really good quality which deep learning has, especially with CNN’s is that the images don’t have to be high quality, as long as the model can make out what is in the picture, i.e. see distinctive features, then this should be absolutely fine for the training set. Then if we print the image out after we manipulate the size and resolution:

The image is more than noticeable. It is clearly and an ok symbol, the distinctive features of the fingers are what the model will apply its weights and biases to. This has only changed the shape and resolution for one image, now we need to create a new an array for all images to be this resolution.

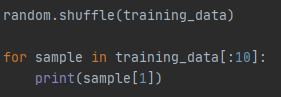
Now we have out data in a format which can be understood, what we need to do now is to train the model on the data we have now made acceptable for the model. We need to define a class for the training data and then create a for loop which reads the data(images) in the file path in the format we have specified.

here we create the training data class, then create the for loops for reading the data and recalling the array which we made to read through all the images in grey scale and 80x80 resolution to ease the strain on our graphics card. the exception is there to ensure that any data which Is broken (i.e. unreadable, such as read-only or maybe the image is corrupted) is passed, and it does not stop the program from running, moving onto the next image. The reason why the exception is commented out on my code is because I know my custom data set does not have any corrupted images which would cause the program to stop, but if you are using someone else’s data set you might want to add this to your code to ensure that errors don’t stop the program from running.

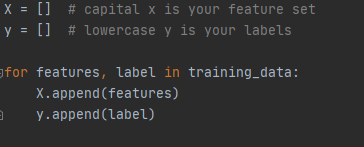
After this we recall the class ‘create-training-data()’ and we print out the result as shown below:

this number represents how many images it trained on. The reason why there is 2 different bars on 100% is because there are two categories which we defined for the model to know what it is classifying.

Another important aspect when preparing the data to the model is shuffling. If you show the model images of a cat a thousand times, this will add create very strong biases and weights to your neural network for cats, thus confusing it when you show it data of another animal. Shuffling ensures it is given a mixture of what you are training it on, this is very important otherwise it will assume everything is cat (or whatever you wanted it to classify on).

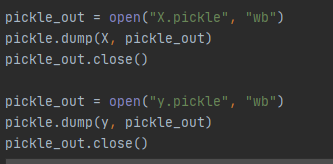
This screen capture of the shuffling data. The first line shuffles the data, and then we make a for loop. The loop shows us how it assigns value to 10 images (10 is specified, this can be more or lower). If it is a thumbs up, it will be assigned a 0, if it is an ok gesture it will be assigned a 1. If you are training a neural network to classify more then 2 different objects or gestures, then the categories will assign an incrementing number (i.e. thumbs up = 0, ok gesture = 1, high five =2, etc…).

Now this data is shuffled and manipulated into a format which our model can understand we can now package it into the variables which we will use right before we feed into our neural network. We do this with **x** and **y** variables. Generally, **x** is for your feature set and **y** is for your labels, this can be expressed by this:

We define the variable for x and y, and we append them to features and labels in a for loop.

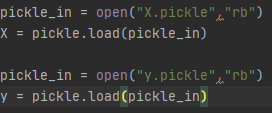
Unfortunately, as of yet we cannot pass a list to the neural network. Because of this we need to make sure our **X** value is a NumPy array:

# Now that the data is in a format which we need, the next step is to ensure that we save our progress. A way of saving progress is using the **PICKLE** package which we imported. You can use numpy.save as another way of saving and loading your data but this is what will be used for now.

We are just ‘dumping’ the **X** and **y** values (features and labels) into the pickle load function so that we can recall it in our next step of making a convolutional neural network.

**Creating the model**

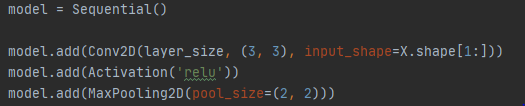
This is the next step in the creation of our convolutional neural network. We have formatted the data in a way in which we deem fit for it to be prcoessed. Now we must create the mode for our neural network. We need to make a new python file, in the same directory (to maintain our current virutal enviroment with all of the same libraries and dependencies).

Firslty, after we make sure that we have imported the right libraries (if you are using jupyter notebook you must do this, if you are using pycharm, it will do it for you as long as you create the new python file in the same directory as the loading\_data.py file is in), we must import our data which we saved via pickle:

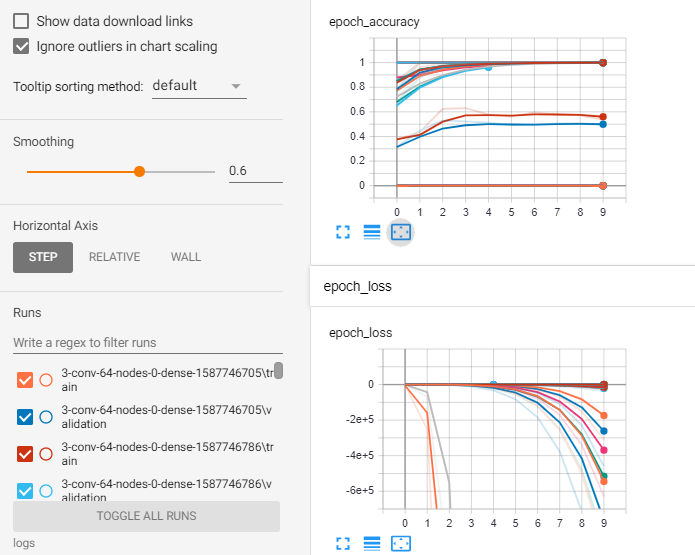
Secondly, before we feed data into the neural network we must normalise the data. We must scale the data. Therefore, in our case we are using images as our data, which means the minimul value for pixel data is 0 and the max is 255. There are some keras.utils functions which do this but for now we can just do

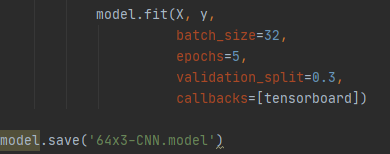
For our model, it will be a simple sequential model, so we need to define the model as sequential. Then we will use the model.add syntax, this is so ensuring that the model is correctly defined. We then need to create it as a 2D convolutional model and how many nodes in the layer. now we must add an activation function (an activation function has been discussed earlier in this report), in this case we will use the relu ( rectify linear) activation. Then we will use the max pooling technique. Max pooling basically ensures that the image data is lowered, similar to compression, enabling the model to take in all the data at a very fast pace whilst also being efficient in how it applies its weights and biases to the image also, you must also specify the pool size, in this case I have made it a 2x2.

Now, as this is a deep learning project, it needs more than one layer. It must contain some sort if middle/hidden layer. All that is needed is the same exact code, without defining the input (‘input\_shape’).



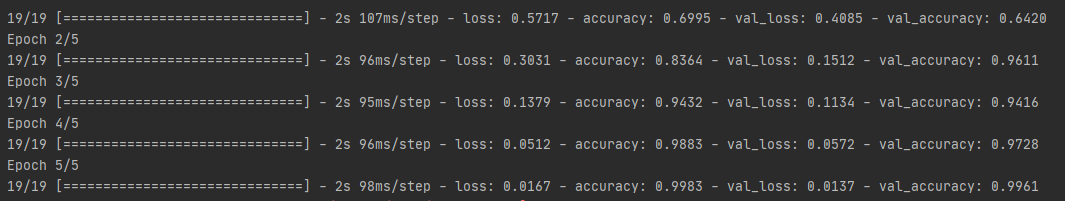
We need to flatten the data. Model.add(Flatten()) this is because CNN’s are two-dimensional. And we can then add the tensorboard function.

Here we can finally use tensorboard, we can change some of the numbers in the model we have created and in real-time we can check to see if we get better accuracy or see what our loss is via the graphs. Personally I wont be needing to look at this because I have already done this before but for those her attempting this as their first time they might want to take a look at this and see what they can change and improve.

now all that is left to do is specify the batch size (how many of the images in your data set do you want the model to try and learn at a time. Of course, you don’t want it to do all of them at the same time, so be careful with what you choose here, I have found that 32 is more than enough for me and my model. validation\_split is the out of sample data, 0.3 is 30%. Epochs is the amount of times the a deep learning model will go through the data, to ensure that is has learnt all it can from the data set. Contrary to popular belief if you make it run through the data too many times it may pick up things that it should not associate with what you are trying to make it learn. For example, if I made it look through my data set too many times, it would expect to see the same background, or my face in most of them. So you have to find a healthy balance in your deep learning algorithm to ensure that you do the right amount of epochs.

The model.save feature allows me to save the current model, and change what I want then run it again as a different model. This is so that if you mess things up, you can always return back to a version which ran properly.

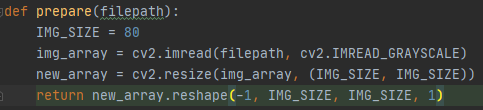
After running this, you will get the following in your terminal:



It displays everything you can see using tensorboard, but in plain text. As you can see my accuracy is quite high, this is because I have fine tuned my deep learning model.

**Testing your model**

Now after the hard work of creating your model and changing aspects to fit a design of your liking, you must create your last python file, which will recall the model which you want to reuse (the one which you saved via mode.save).



here you must define the data which you will prepare. The IMG\_SIZE should be equal to the same amount of which you trained your data on, this will ensure that any new images which you want to test your model on, will be changed to the same resolution and the same size as the one you used to create the data on.

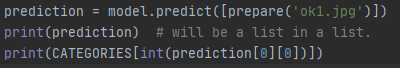
The img\_array is what we will use to read the image in the same grey scale, as we have only trained out model in grey scale due to making it simpler.

New\_array will accept the image and resize it using the img\_size syntax.

Return function will return the image with shaping that TensorFlow wants.



This will load in the model which we want.



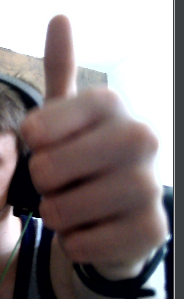
This is the last part of the testing model, the prediction function, will allow us to import the picture file (jpg or pgn etc…) and it will tell the model to run its data on that and return a value, either 1 or 0. 1 stands for one gesture, 0 stands for another. All this depends on the accuracy of your model.

# Results

This image of my hand doing an ‘ok’ gesture called ‘ok2.jpg’ is put through my model via the prediction = model.predict.

When I run this I get the following result:

The accuracy is fairly good for the ok gesture. Now what about the thumbs up gesture?



This is called ‘thumb1.jpg’ and when I run this, the model returns a value of 0, which is a thumbs up.

**What could have been done better?**

One aspect of this which I would have liked to import in my project would have been making the project work in real-time. What I mean by this is I would have liked to create a python file which used the camera and every second it would take a picture, if gestures were recognised then an outcome would have been performed. For example, using the ‘ok’ gesture, I would have liked this so control the brightness on the monitor. Similarly, I would have liked to have used the thumbs up gestures to open controls for volume, all in real-time. This would cause some problems such as memory issues. For myself it wouldn’t be too much of an issue since I have very high specifications and a very good graphics card (RTX 2080), but for those who are using machines that do not have very high specs, this would cause memory problems. Thankfully, there are ways of capping the amount of memory used in program by the GPU or processor.

There are a few bugs in my model, for example, even if there is no gesture present it will always assign a 0 or 1. I could fix this by adding an else if exception to my code.

**Having enough time I would have liked to have added this into my design but sadly, due to recent events of convid-19 this has made it difficult for me to get the recourses I needed in order to complete this project in time**

# Method

# Conclusion