**UNIVERSITY OF HERTFORDSHIRE**

**Department of Computer Science**

**Modular BSc Honours in Computer Science**

**6COM1053 - Computer Science Project**

**Final Report**

**April 2021**

**TITLE OF PROJECT**

**Understanding of neural network vision on a universal language.**

**Author's initials and surname**

**D.L.G. SOLBE**

18040764

**Supervised by:** Yongjun Zheng

ABSTRACT

Nowadays, the ability of computers to identify objects such as symbols, among other subjects, is vital in the use of machinery and day to day life for humans. One such facet is images to read universal visual language, such as street signs commonly used on cars.

This project aims to research computer vision and then produce an algorithm to test and better understand how these computers view images. This report will include image filtering and a MobileNet neural network with insight into feature maps.

Acknowledgements

Firstly, many thanks to the supervisor Yongjun Zheng for his help and supervision of this project. Yongjun Zheng has always given detailed replies and help.

Secondly, thanks to the Lonely Yogs for their many helpful ideas for this project and outside of it.

Moreover, thank you to my family, brother and mother, who aim for the best for us all.

Contents

1. Introduction for chapter one

This chapter goes over the overview of the project, including the problem and solution this paper aims to use, along with the main aims this paper's artefact seeks to achieve. A report structure shall also be included.

### 1.1 problem overview

Humans use languages to convey meaning, such as streets signs, known as a "universal visual language" (On The Road Trends, 2020). However, as reliance on computers is utilised more, how will computers understand this visual language? This understanding can be seen with modern cars such as Mercedes-Benz "Traffic Sign Assist" (Mercedes-Benz ECQ), which uses maps and cameras to inform the user of the road max speed; however, this introduces the issue of too much trust. What if the user of these cars is found speeding because the car misread the information on the sign? Would the user or car produce be the one at fault? This question becomes even more prudent with users trusting technology and its use without understanding it, as seen in Kiran, A.H. and Verbeek, P.P., 2010.

This issue comes under the umbrella of understanding how these algorithms work to produce the results this report needs for modern technology. To move forward, some risk must be taken, and it is only in understanding this risk that it may be accepted and used wisely.

This problem of: "How do computers understand human language such as traffic signs?" requires insight by this report utilising a proposed solution using the A.I. field with a narrow focus on traffic signs.


An excerpt from 'The Highway Code' (2021) showing some common English traffic signs

### 1.2 solution overview

By reading such works as Sajjad, K.M., 2010, Cilimkovic, M., 2015 and others, with more critical details in this project's chapter 2: literary review, it may be seen that computers often use neural networks to understand images making use of such modules as TensorFlow. Using similar methods like this to make a neural network for traffic signs will provide a base for this project to gain results.

However, these works do not include how a neural network will identify images. TensorFlow provides a method to produce what the neural network maps see an image as, using Inceptionism, called Deep Dream. These models give this project its hypothesis by giving us tools in which this report can find its answers.

Hypothesis: 'algorithms can identify patterns of languages and use these patterns to predict which data type they belong to.'

To produce these results, the following steps will be taken:

- The neural network will be trained with Mykola (2018) images in a balanced dataset using TensorFlow, OpenCV and MobileNet in python.
- The network will be checked and if past a point of 90% correct images it will be ready. 80% is selected due to the results from "average of the result is up to 90%" (Kothari, J.D., 2018) which is rounded down due to less skill and equipment.
- Deep dream will be run on each different image type. Each image type will have other key points on their maps, proving the network doesn't copy the same key points while tested on each subject to remove the 10% to 20% error risk.
- These results will be checked for a defined pattern. If found, they will be marked as a successful way to see computer vision. These patterns will identify the key points and should be easy to see, such as the example image below from DeepDream Alexander Mordvintsev (2015).
- Hypothesis will be supported if the network can 1: guess the image and 2: key points can be identified in the given output or unproven by these results of it is unable to guess the correct results with or without key points.

Image from Alexander Mordvintsev (2015) showing a deep dream image with clear patterns of weights in different angles and the arm connected to it.

### 1.3 aims

For the project to succeed, there must be a clear purpose and method of delivery of the achievements. The purpose of the project is to improve understanding of computer vision and create a practical neural network. In understanding computer vision, this report wishes to achieve a way to produce work that gives reliable results, measure those results and set achievable project goals. This report has produced guides to follow, such as reference work to gain understanding without plagiarising and produce work that others understand and can repeat. It also has the following goals:

Overall project goals:

- Produce reproduceable work including results and artifacts. To achieve this a selection of artifacts image and guesses results should be produced with code and notes of the working artefact in the final report.
- To produce a goal for artifacts and back it up with evidence. This can be seen with the 80%+

pass rate of guessing by the artefact which will be tested separately from training and results given in final report.
- To identify artifacts understanding of inputted image. As seen with use of checking feature Map and checking for patterns with use of high pass rate artifacts.
- To research different methods and use them to produce an effective artefact. This can be seen done within the literary review by identifying the use of MobileNet and reproducing an artefact with its use however the effectiveness will be proven by producing a reliable artefact.
- To produce correct references and state their importance. This can be seen throughout the work with quoting and Harvard references.

Artefact goals:

- Produce a filter to reduce image noise using OpenCV and changed to same size.
- Produce a dataset from Mykola (2018), filtered and refined to 1) reduce total amount of different signs, 2) remove copied/repeated images, 3) remove low quality images.
- Sort dataset into different random sets
- Use TensorFlow to produce MobileNet network then save model to file with trained weights
- produce image of feature Map layers
- a manual image input to test new untrained images on by hand
- produce a basic interface

Report goals:

- To produce work that is understandable by others. This can be achieved by checking work to others and having others overview it such as my supervisor
- produce a guide to pass on how to achieve a similar Artefact
- reflex the question of the paper in all work such as with links to it within the images from the artefact
- state if the question is answered within the work meaning whether it was proven or not with reason why
- provide real world use
- question future research based on results

### 1.4 report structure

The following structure for the report can be seen here:

- Title page which includes such details as name, course, university, project title, etc.
- Abstract to introduce the idea to a new reader.
- Acknowledgements to thank those who have helped this report

1. Introduction

This chapter introduces the paper with an introduction to the problem and solution this paper aims to use along with the main aims this paper's artefact aims to achieve. A report structure shall also be included, much like this current writing.

2. literature review

This chapter will go over the research needed to complete this report, including work done before and after creating the artefact. Most research is based on neural networks and other image detecting systems.

3. Methodology

This chapter will cover the planning and methods used to produce this work. This includes the artefact building, testing and later evaluation plan, and how research and this report writing were conducted..

4. system design

This chapter will go over the working of the artefact in more critical detail, including flow charts of training the model and the use, including the user interface. This overview will clearly describe how the MobileNet model works and feature maps and how these tools help answer the question.

5. Analysis Results

This chapter will state the final system and how its outputs can be compared and analysed to find results for this report's question. This output includes feature maps and the accuracy of guessed results.

6. Conclusion

This chapter covers limitations of the artefact, how results of the final question were answered, the results of the aims and conclude the report.

Appendix A

This selection will hold extra data and charts such as flow charts and outputs from the artefact.

2. Literature review

## 2.1. chapter Introduction

This review will research computer-based image recognition with deeper focus on universal visual language of street signs to solve the issue of how computers understand human languages. This will be in order of existing image recognition applications, algorithms used in image data and data passing, Technology used in the above topics, and Functions of Technology with a focus on greater results..

### 2.2. excluded texts

To gain a factual base the results will be limited to paper such:

- cited ten times or more by other sources showing other documents find this information reliable. This proves that others find this information to be true.
- related to the subjects as to exclude information that is misleading and unneeded. This is for computer learning is such a big field this review needs to focus or will make the review unusable.

#### 2.2.1.

The project is predicted to be street sign, as seen below in "fig 3" (Stallkamp, J., et al 2011), reading software with a focus on being readable for many. It should take images and output from a list a definitive result.


**Fig. 3.** Traffic sign classes

### 2.3. research

### 2.3+. Existing application

To find known pitfalls and methods research has been done on image recognition with computers:

#### 2.3.1 Image recognition background

images contain information however for computers this can complicate the work needed, therefor this report will simplify the data. This noise comes from image format and background, like foliage or walls.

Images come in many forms like differing resolution which results in images that are deformed or lack details. These effects the results of the image recognition tool and it is recommend by Sajjad, K.M., 2010 to use a high-resolution photographic camera or Infrared (IR) camera.

Issues can be improved on by editing the image before it is used by an Image recognition tool. Removing backgrounds is one way this is done, using examples by Sajjad, K.M., 2010 this report can see that below:

1. Reduce image noise by "Highlighting characters... Suppressing background." (Sajjad, K.M., 2010) by turning the image to black and white.
2. Then threshold of the pixel based on a pre-done based information. This means of a pixel doesn't have a given value it is changed. Like turned black if not matching or white if are.
3. component algorithm is used. This collects the white pixels if they match together, this
4. makes a collection. For example, letters and other "blobs" of white.
5. A "special algorithm" (Sajjad, K.M., 2010) removes other blobs. It is unclear in this case what this algorithm is however for the review it is not need to understand.

Fig 2 to 6 from K.M., 2010 show this:


Fig. 2. Captured Image (Original image)


Fig. 3. Localization (Thresholded image)


Fig. 4. Connected Components (Blobs)


Fig. 5. Segmented blobs


Fig. 6. Classified Blobs

showing how to improve the image for the tool however some issue remains. If an image is not of high quality, it can still result in a bad result as thresholding may not be able to pick out pixels from other details.

Gaussian Blur is another method. Sajjad, K.M., 2010 method mayn't work on more complex items but this method "rid of as much noise as possible while losing a minimum of

information" (Gedraite, E.S. and Hadad, M., 2011). Blurring the image using the average of a pixels nearby neighbours that pixel gains the greatest scale while its neighbours receive smaller scale. Often used with a "finite-state machines" (Waltz, F.M. and Miller, J.W., 1998) resulting in high yield, low loss data. The image below "fig 25" (Liu, Y.Q., et al., 2020) shows this effect in use by changing the "generalized Gaussian (GG) function" (Liu, Y.Q., et al., 2020):

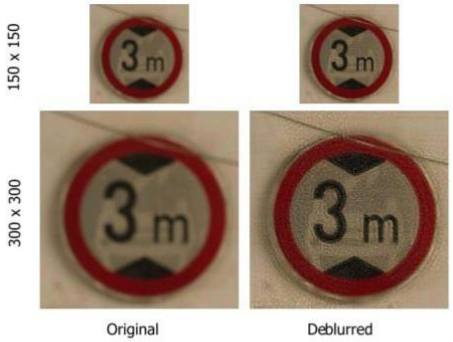



Fig. 25. Comparison of deblurring results produced by Zhang *et al.* [28] on images of sizes 150 × 150 and 300 × 300 pixels.

This smooths the image removing more details. It benefits from reducing the complexity however doesn't give as binary results as Sajjad, K.M., 2010 method.

temporal filtering will blur images making changes in movement or light fewer. This tool is used in "motor imagery" (Thomas, K.P., et al 2008) and is used in video.

### 2.3.2 Image recognition with DNN

Processing large variable data such as images this report need algorithms to determine the mathematical value of the image then logically connect that to output. One way of doing this is with a neural network (NN).

A NN can be seen as "layered network which have a layer of input unit … intermediate layers … output units" (Rumelhart, D.E , et al, 1986), in these layers are nodes. The input takes data such often in an, hidden then uses compare the data. with this output the computer can understand using 'activation ((weights (w) * score (P)) + bias)' a pattern can be seen

resulting in "when $\sum n \quad wi \; Pi \geq \theta$ (image of formulaic to reduce risk of document errors $\sum_{i=1} w_i P_i \geq \theta$ )where θ is the threshold" (Rojas, R. 1996) sends a value to the next layer, this is without Bias as it is not always needed but can be used as an constant

for linear function. Then the output takes this pattern recognition and shows what the computer believes it to be. seen from Cilimkovic, M., 2015 image below can be shown as a model of nodes and how they can interact. The lines are the weights with the circus being the nodes:

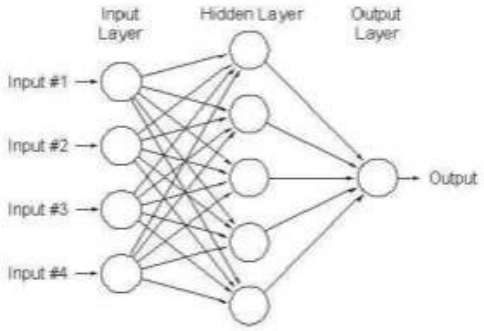



Figure 2: Simple Neural Network

calculating the weights, this report use a back-propagation algorithm. The weights at first are set at a value that likely is not the most effective but this report change this to gain a more correct output compared to input. This algorithm can "compute how the error will be affected by changing these states and weights" (Rumelhart, D.E, et al, 1986).

Deep neural networks (DNN) have had "successful application to … computer vision" ( Liu, W, et al, 2017). This is done in different ways but Liu, W, et al, 2017 talks about "layer-wise- greedy-learning… RBM … DBN … autoencoder". DNN is built on NN however they often give better results or other benefits such as less labelled input data with make it less work for a human by use of different Algorithms.

### 2.3.3. Image recognition of traffic signs

Traffic signs are used in computer vision (CV) and by looking at these projects this review will gain a better insight into different issues.

Dataset, which is explained as the input data which become subgroups in the case of Geronimo, D., et at, 2013 “Detection set … Classification set … Recognition set”. Subsets are used:

1. Candidates’ generation which finds similarities between items such as “distinctive colours and geometric shapes of the signs” (Geronimo, D., et at, 2013)
2. Binary classification which filters an item to present it with a label. This can be done with “histograms of oriented gradients, Haar-like features… support vector machines” (Geronimo, D., et at, 2013)
3. Clustering with the above tools can produce a system to list identify items. Clustering could then put identified items into groups.
4. Recognition system can then focus the item more using the same tools. Such as a sign in a ‘shape’ group to an “(“stop,” “give way,” etc.)” (Geronimo, D., et at, 2013)

The datasets are “divided into training and testing” (Geronimo, D., et at, 2013). Computers are very good at recognising patters and need to be tested with unused data to check they’re not learning the wrong thing e.g., weights.

Used in “Deepsign” (Li, D., et al, 2018) which uses clustering and recognition system seen in the below from Li, D., et al, 2018 “fig 2”:

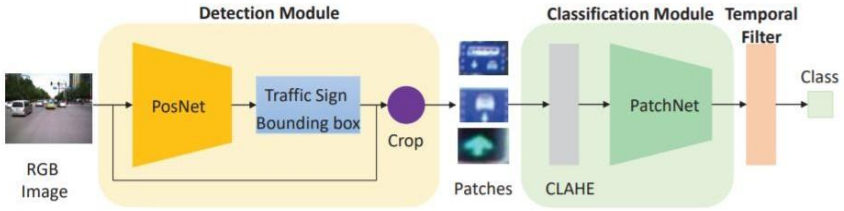



Fig. 2. DeepSign system architecture. The position network (PosNet) which is an object detection CNN first detects the traffic signs in an RGB image. Then after cropping the detected region, the image patches are converted to gray images and processed by contrast limited adaptive histogram equalization (CLAHE) and sent to the patch net (PatchNet) for classification. Finally, the temporal filter is employed to improve the recognition result.

Seen working where sign W11 are recognized with clustering however in (a) a backwards sign is recognized, P05. (b) uses filtering to then remove P05 then in the "(a)…(b)" below from:


Fig. 4. Traffic sign recognition without (a) and with (b) temporal filtering. The temporal filter can utilize the contextual information and filter out the false positive result PO5 at frame t-1.

Another way is with "Spatial Pyramid Pooling (SPP) Network" (Tai, S.K., et al, 2020) which splits up images so they don't need a fixed size then connect the split images to a NN. Letting us use different images.

### 2.3.4. Sub-summary

The project will filter images to reduce workload using thresholding shade to do this as it is simple to understand which is important to understand how computer understand human signs. To understand computer understanding of human signs, the project will focus on recognition system.

The project will use machine learning to achieve this. This choice comes down to a layer-wise- greedy-learning DNN as it should be at a level that is more understandable while being effective if the project uses a wide dataset to avoid such issues as overfitting and local minimum. My project may extend to video so it will benefit from temporal filtering and locate objects with clustering.

### 2.3.5 Algorithm and models

Understanding the logical, mathematical workings that may reduce computer risk and improve speed are needed and explained:

- component can connect vertices in an undirected graph. an Algorithm can be used to detect groups such as “blobs” (Sajjad, K.M., 2010).
- back-propagation algorithm is an “teacher-based supervised learning approach” (Liu, W, et al, 2017). It gives accurate results and works by calculating the cost from the input and the NN output then changes the weights however it stays in “local optima” (Liu, W, et al, 2017) not the global optimum.
- layer-wise-greedy-learning means learning will be done pre-training of NN, making it DNN. It takes features of input then feeds the next node a compact, labelled data resulting in reduced over-fitting from low inputs and “local minimum” (Liu, W, et al, 2017) is better resulting in quicker rates of converges.
- Support Vector Machine (SVM) are based on machine learning with “high accuracy” (Zhang, Y. and Wu, L., 2012) and overfitting concours are lessened. It works by breaking images into a histogram of colours, texture descriptor and shape descriptor. Then runs a PCA before using the “kernel trick” (Zhang, Y. and Wu, L., 2012) to split data. Then this report train the SVM with data and use a DAG-SVM table to save complexity and give results. We then use a test dataset to check its results. An issue is that it can “only handles binary classification problems” like set labels (Noble, W.S., 2006).
- artefactPrincipal component analysis (PCA) organizes data such that “largest variation come first” (Zhang, Y. and Wu, L., 2012) and other data changing effects
- Deepdream uses a layering of map-layer filters on an image which is passed into a neural network as explained by Alexander Mordvintsev (2015)

YOLOv3 seen below in fig 3 image from Tai, S.K., et al, 2020. This is a detection algorithm that benefits from working in real-time:

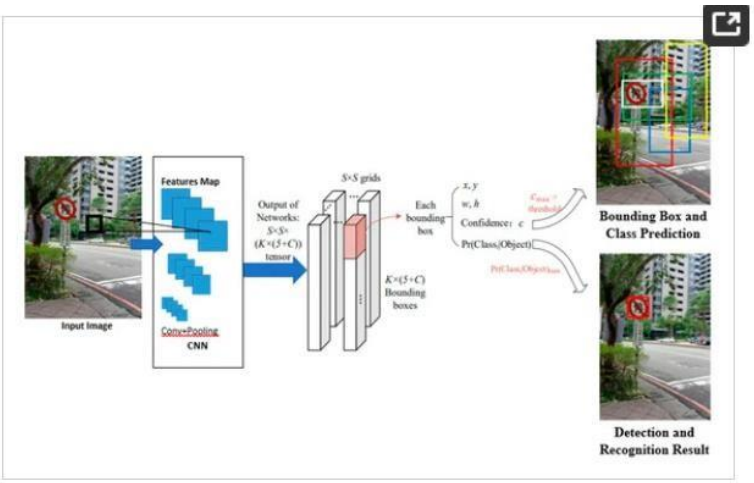



**Figure 3.** Yolo V3 SPP architecture.

### 2.3.6 Technology

This review has researched these technologies used to produce the underlying tools used in testing below:

- Python is a code which benefits from its libraries, pre-made code that can be used, which can be seen used in Sajjad, K.M., 2010 using "OpenCV… for languages … Python" and by Abu, M.A, et al, 2019. using "Python … with TensorFlow framework". It also benefits from an easy readability which will help keep it understandable.
- C / C++ is code which is much faster than Python while also letting users control memory which is important when dealing with data. This can be seen used in OpenCV and is used in CV such as in Baggio, D.L., 2012 "code in main() function".
-

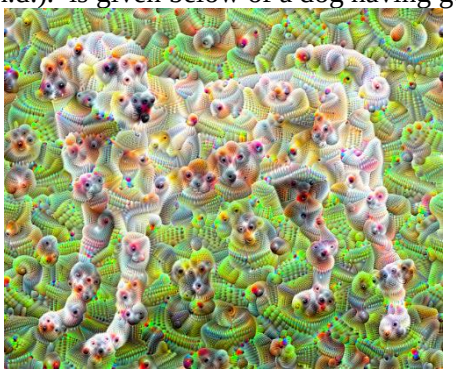
### 2.3.7. Sub-summary

The system is not time or complexity limited to such a degree so the project will use Python to code the project as it offers more readability while being a code, many are familiar with, and it offers a menagerie of libraries.

### 2.3.8. Function

A deeper look into the tools that are used to produce greater results and expand the project seen below:

- OpenCV is a C Library can be used with C and Python. For use with image recognition, it has "five hundred optimized algorithms" (Sajjad, K.M., 2010) which can be used to benefit the AI and are used in many works while also used in "OpenCV GUI"( Baggio, D.L., 2012)
- TensorFlow Library for python which can be used to make many AI including NN and DNN such in "usage of Tensor Flow" (Kothari, J.D., 2018). It adds the benefit of a reliable, state-of-the-art model on which this report can base the work letting us skip many mundane tasks so this report can better understand the core issues.
- MobileNet Library is used to make DNN for mobile as well as embedded devices such as a pretraining then used with the Tensor flow like within Kothari, J.D., 2018 "configured with MobileNet". It acts as a NN with must be trained and tested but can output a good result with fewer resources.
- Deepdream is a python library used to produce images from a neural network by "visualizes the patterns" (TensorFlow. (n.d.).) learned by the network in each layer such as feature Map. An example from TensorFlow. (n.d.). is given below of a dog having gone through this system.



Deepdream dog from TensorFlow. (n.d.).

### 2.3.9. Sub-summary

This project will use libraries in the code to gain a more accurate score as many do not have deep experience with DNN and filtering. MobileNet and TensorFlow work well together and as such the project will use them to make a DDN and OpenCV as a filtering tool. These tools could be used to expand the project to include webcams with the use of OpenCV and other such useful tools for real-

world testing while using just TensorFlow could give a greater understanding by being more detailed.

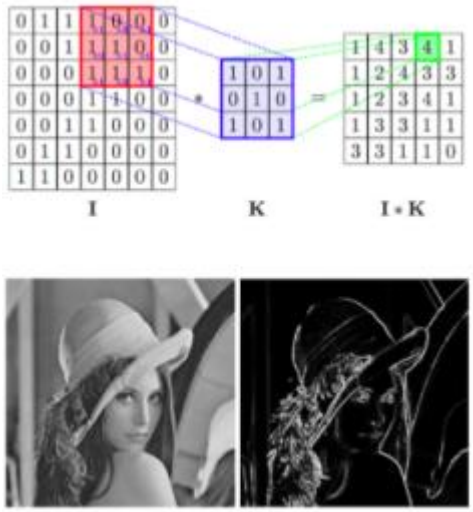
### 2.4. Chapter summary

This chapter has looked over many tools, past applications and key technologies that have been used, in doing so this report have narrowed our project down to thresholding shade filtering recognition DNN, using layer-wise-greedy-learning Algorithm, using of Python and its libraries to gain the most benefits while also learning of pitfalls such as using the wrong image recondition like clustering and limited datasets. This set should give the project a strong base to produce an image recognition system for street signs which is also readable making are problems addressable where they may not have been before while also giving way to expanding the project in the future.

### 2.5. EXTRA RESEARCH

During the production of the artefact some extra research was required to understand how the system worked and how this result was produced. This extra research is included here.

During the creation of the artefact, it was found that the input array from the MobileNet did not work with that of deepdream. This resulted in a secondary method to view the systems outputs. One such method was the use of feature maps which are made use of inside CNN “Convolutional Neural Networks” ,Persson, S. (mars 2018), models which MobileNet is. These feature maps are produced to form “particular features of interest”, Persson, S. (mars 2018), and then are measured against weights to decide the systems result. They’re made by taking the input image and applying a matrix filter over it. This filter then acts to produce key features, which is why the name is feature maps, which the network uses. A image of one can be seen below.




An example of feature maps being produced by a neural network from Persson, S. (mars 2018)

While producing the artefact some extra libraries had to be used to enhance or use the tools. This was to reduce human error, speed up work and ensure the tool worked. This included python’s libraries of random which will let the code choose random variables, more can be explained in later chapters or from their documentation random (2022). Also included is the pythons’ libraries of OS which will let the code edit and open system/ user files, more can be explained in later chapters or from their documentation OS (2019). Third is the use of shutil, a command which lets the code end the current running application of itself, more can be read here in its document shutil (2010). Lastly the use of mathplot was used to produces charts of the systems output. These charts used mathplot to not only

produce them but also to give a pop-up menu, which more can be read about in the documentation matplotlib.org. (2022).

In addition to using these tools another tool was used. The use of OpenCV was already decided however to reduce the risk of running outdated system an upgrade was made to OpenCV 2 which changed the way the code will be produced but reduce the risk or running different versions of pythons for the same result. More can be read about it in it's documentation, docs.opencv.org. (n.d.).

This concludes the extra research needed for the system and the methods of using them.

# 3. Methodology

## 3.1 chapter introduction

This chapter will go into methodology details, the how of each main part. These parts are research, dataset, artefact, and testing. Then included will be its purpose in the report and to justify it. All research was conducted from documents, records, case studies, and research with a minimum of 10 works citing them to ensure the research was reliable, meaning this project's work can be reliable.

## 3.2 methodology of research

To gain a greater understanding of the topic, this project conducted research. This research was based on the knowledge that a large amount of image processing today is done by A.I. or algorithms that detect basic shapes. The first research was on methods of computer image reading.

Already made computer image reading software was the first to gain information on how it can be produced to a high level and how these models work to achieve results. This research would prove helpful in creating an artefact as it would provide working methods and ways to improve outcomes. The information on model results would provide evidence of how the artefact's final results for this report were produced and the method used to achieve a result. A quantitative and confirmatory research focus was used to find models with a high success rate for their results and had confirmatory codes or methods. Once a paper was found, it was checked quickly to ensure all the above points were found and based on computer image identification. A secondary focus on traffic signs or langue was used over other papers as it focused more on this report's subject. Such research that was found using this methodology was Sajjad, K.M., 2010 and its use of filtering to improve its model's results which can be used for this reports artefact. Another was Li, D. et al., 2018, which provided a detailed case study that describes the different types of computer vision and the workings of neural networks such as data sets. This provided information on already existing systems and the method to find them. This new knowledge also leads to further information, such as neural networks.

Further research of neural networks and introductions to feature maps could provide a method of producing an answer to the problem. With fundamental research on models complete, a way to answer the problem of this report was needed to be researched. This quantitative research would look at how neural networks functions to determine different objects and then how to show those functions to understand them. This would be used to answer the problem by showing how one type of function works in the report and would give the needed steps of the artefact. This can be seen with Cilimkovic, M., 2015 with a greater insight into layers which will provide a backbone to the explanation of the report's findings. Another example is TensorFlow. (n.d.). This shows an example of feature maps leading to important research for the artefact output and explanation of the results. This shows how further research will be used in this report and how the objective of the artefact was found. This leads to how to produce these objectives with the artefact.

Methods of researching how to make this reports artefact along with its, explained above, objectives. To achieve the main aims of the artefact, research on tools and methods to build it is required. This research is in addition to the already found information such as filtering using OpenCV Sajjad, K.M., 2010 and known skills such as coding with languages such as Python or the C family. This use of pre-existing information is to build on already done research and skill to save time for the project and building of the

artefact. New research can be seen with known Python knowledge through research of python libraries for neural networks such as Kothari, J.D., 2018. This method of research will provide the ways to achieve the goals of the artefact later with a reliable and repeatable code that many can use.

These were the main methods of research used to gain understanding and tools for the report and artefact. This benefits the report by adding evidence and support and displaying the future work needed to be achieved in the project.

### 3.3 methodology of the dataset

Research on neural networks showed the importance of having access to a large and balanced dataset for this project's artefact to train from. From research such as by Kothari, J.D., 2018 and Li, D. et al., 2018 which go over details of datasets, some goals could be set such as having:

- over five different sign types for the artefact to train with
- over 200 different images for each image type
- images have a high-quality
- balanced amount of images

these goals were ideas backed up by research and will be explained in more detail in chapter four. As this project is academic, the images for the dataset must be given permission to be used for the academic purpose to ensure the ethical reason of not stealing others' work without permission.

Using these goals as guides, a qualitative search of research papers stating these points was conducted. Such papers as Stallkamp, J, et al., 2011 used a "lifelike dataset of more than 50,000 traffic signs" from the GTSRB "German traffic sign recognition benchmark" from Mykola (2018). This dataset achieved all set goals apart from a balanced number of images; however, due to its larger size, this could be achieved by editing the dataset.

The method of setting goals, researching papers using datasets and checking datasets goals worked to achieve a dataset that could be used for this project's artefact; however, in doing so, some extra work would be required to achieve all goals.

A method to improve this in the future could be to produce a new dataset by taking pictures from the real world. This guarantees the goals of the dataset while potentially providing more real-world examples such as extra noise from plants; however, this would likely come at a greater cost of time to produce a dataset compared to the one already produced. Either method can work for the training of the artefact.

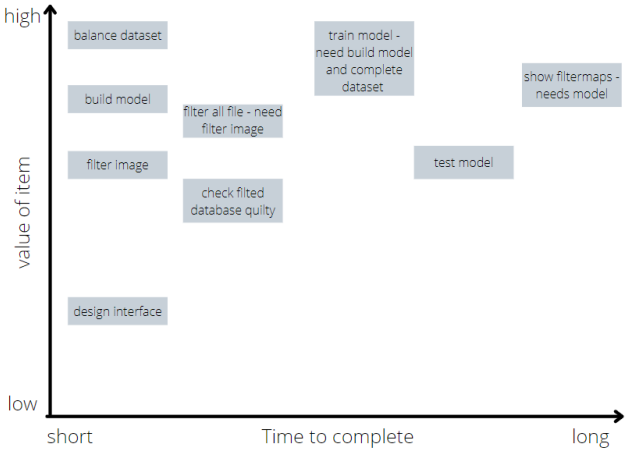
### 3.4 methodology of artefact

To attain results to answer the question of this report, an artefact with the set goals from the research will be produced with the tools and code also from the research. To do this, an Agile development methodology was used to design it. This method was chosen due to a small team developing the artefact, which an agile method lets developers take on different parts of the system such as coding, testing, and planning to mean this model matches this project's workforce.

The benefits of this method were that it allowed the software to be released in parts meaning small scale testing and design were done before being joined to the main system later. Compared to a waterfall method in which it may be all wholly written and then tested, large-scale error testing and design changes would likely have to be made. It also had the increased benefit that if a new tool was being used, it was often tested and could practice apart from the primary system, often saving time by not having to run the whole system on a known not working code and reducing the risk of losing the main code while editing the new tool such as an input to a library code.

An issue with agile was that it reduced the amount of documentation for the report, such as past issues. This is an issue due to a common issue not being reported in this document and exploration for change to the code not being highly detailed.

The development of the artefacts was broken up into parts as suggested by an agile method which can be seen in the below chart. These were put to be done from top left to the bottom right, and the parts placement is done with what is most important based on time to complete and the value of the item.




This method was used to get the most critical parts done first while ensuring the time would be kept. The use of deepdream was expected however was changed to show a feature map. This is part of the method; instead of taking too long to produce a working deep dream function, a substitute was used instead. This benefited the artefact by meaning other features such as a design interface were able to be produced along with the showing of filtered maps compared to a single function of deepdream.

These agile methods produced an artefact following the set goals with minimal issues. Setting steps, knowing the order, and how to produce all the needed goals with no overuse of time.

In future work, an added step of documentation during the artefact production would be advised to better understand issues and solutions to the task. This will benefit from a final detailed report if an increase in time to produce the final artefact.

### 3.5 methodology of testing

Testing the work during building and after the building is vital to ensure a factual report. All work needed to be tested to show what is expected to ensure no mistakes or false variables are given, which could discredit any findings found. Such testing includes the functions of the artefact to ensure the model will function and is repeatable, checking feature Map details to prove they are displaying correct results to ensure no false positives or negatives and testing the artefact guess function to prove the real-life application of such a code.

Testing of artefacts is done throughout the development and at the end. This is to ensure all the functions give out correct results and work with other functions, such as the filter working with the guess function to ensure the neural network will receive the same filtered image that it was trained on; else, the level of correctness may be significantly lowered. Tested on the individual's function is to ensure it produces the correct inputs and outputs. It is then added to the main code to be tested with other functions inputting and outputting to each other. These connected functions testing ensures that there is not an issue passing that data around, such as not returning a string value; if such a case did happen, a correction could be made knowing the issue exists within the connections of the functions. An interface is then added and tested. This interface is a way of testing as it limits the inputs that can be used, so no more testing will be needed for the user side once it is tested. It will be manually tested by inputting all different types of input available and in a different order, including incorrect inputs, to ensure this will not cause a bug to form, which could later damage any findings.

Testing of feature Map is essential to answer this report. These feature Maps should be tested to check for constants within them, such as repeating patterns, to understand how they work to process the image.

This should be done by taking different image types, including wrongly guessed images by the network, then taking them from different points in the model's layers. These layer feature Map images are then compared to each other and discussed by recognised patterns, and known knowledge will lead to a logical conclusion about their purpose, function, and further developments from other image types in the dataset. This deconstructing of the patterns will base this report's conclusion based on the found results.

Testing of guess is essential to understand if a balanced dataset was used. At the same time, being trained, the neural network will test the model to ensure it is improving and not stuck in a false positive. This testing shows that 92% of guesses are correct; however, testing needs to be done afterwards. This second testing is due to the test data coming from the training data, which can mean that if the data is imbalanced, it can lead to a false level of correctness. The images of the test set may also be all the same, which could cause the network to train to only one image. For example, every warning sign is the same, meaning it will train itself to produce a 100% correct guess on that image type while it only knows that one image.

These issues are rare due to the above testing of the data set; however, simple manual testing is done by passing three images per image type into the guess function of the artefact, which will predict the image. By checking these results to 92%, the machine predicted it would be clear to see if any of the above issues have happened by dramatically reducing incorrect guesses. These manual image checks should be the same image type as before; however, use new images in the training set to reduce the risk of the neural network having only learned of very few different images. Using wrong image types to test it will not result in any test data as there is no 'unknown' type, and it has to guess some type.

All these different tests will be fundamental in producing an accurate and logical result, which will help this report resolve the problem it aims to solve.

### 3.6 chapter summary

This chapter covers many vital parts of this report and then explains their purpose, including how and why they were done in such a way. The methodology of research, testing, and the artefact was done in the way that was thought best. However, improvements can always be made, as explained in some of the points given about ways to do things in future tests on subjects in the same methodology area. These explanations prove that these methods have effectively gained evidence and tools. This report contains highly reliable evidence and factual output from this report's artefact; however, it could be improved in further works.

4. system design

### 4.1 chapter introduction

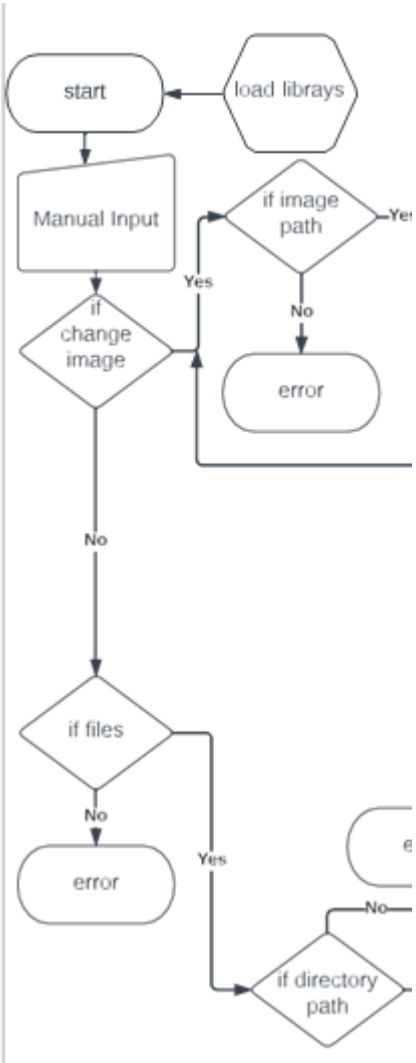
This chapter will go over the working of the artefact, including flow charts of training the model (which can also be found in the appendix) and the use of the model, which includes the user interface. This overview will clearly describe how the MobileNet model works and feature maps and how these tools help answer the question.

All used libraries and versions are listed in the appendix under ‘Artefact libraries and versions.

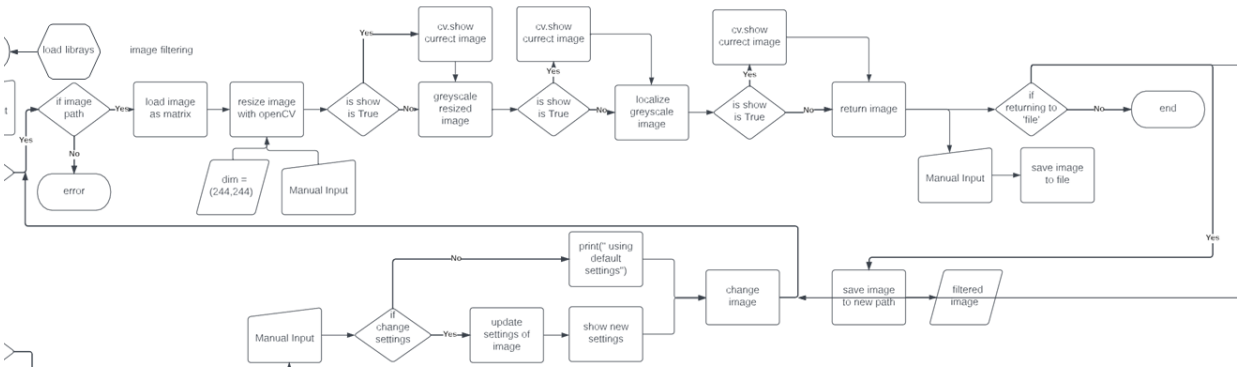
All code can be found in Appendix B

### 4.2 flow charts

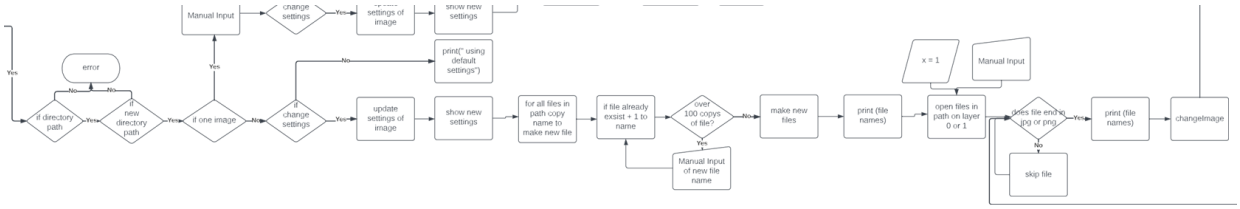
3 main models were produced for this system. Below are some flow charts to explain the basic method used by them to produce the output. The first is image filtering model.



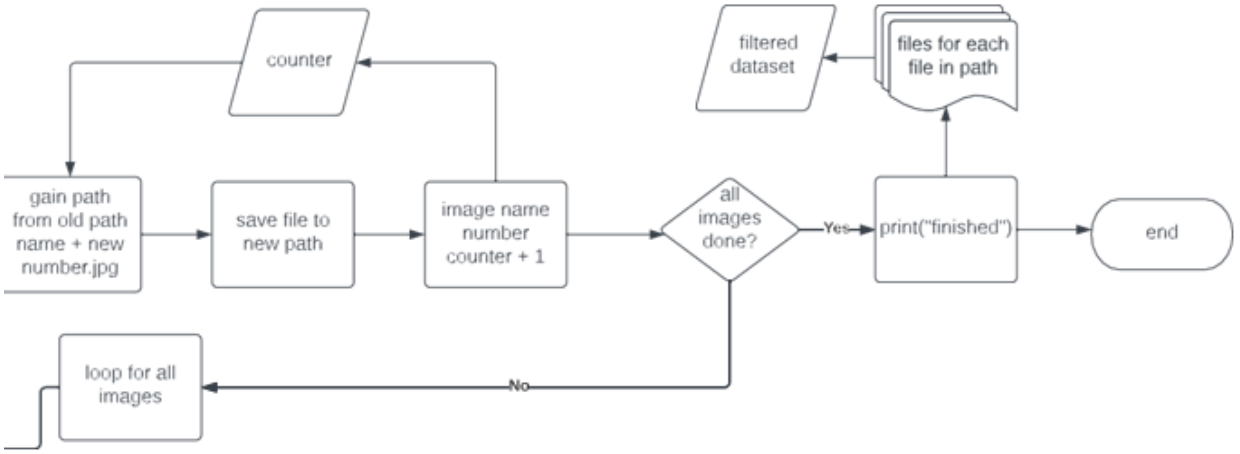

This starts by loading the model librarys first. Then once’s started it will run a single change image or the whole file. It can also give an error.




If change image is chosen it takes the images path, changes it to the given inputs or default values then saves it to a file. If it's apart of the change all files it will return back to that path or finish the code.

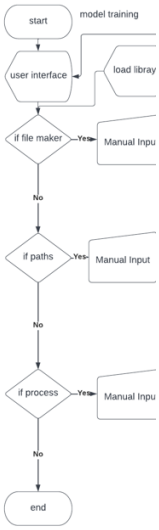



If change file is chosen then it checks and uses the user inputs. It then take a image or more than one image to the seen before change image line. Onces it returns it is then changed to a new file with it's new name.

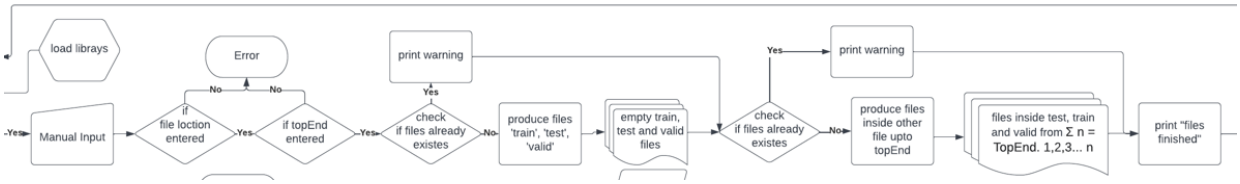



Here the file is named, and all images are sorted to their files. Once this is done it returns it ends the system.

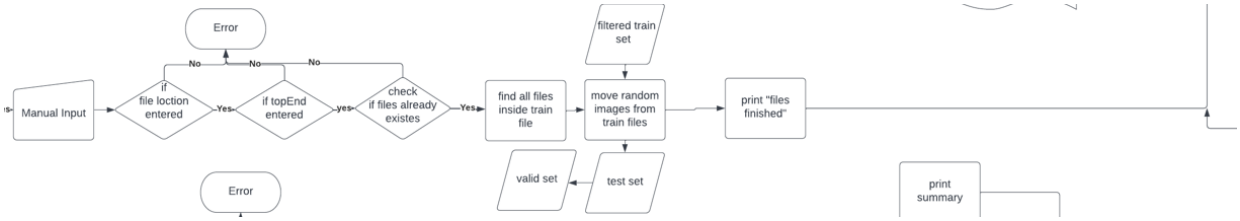
This is the end of the filter system. Next is the training model.



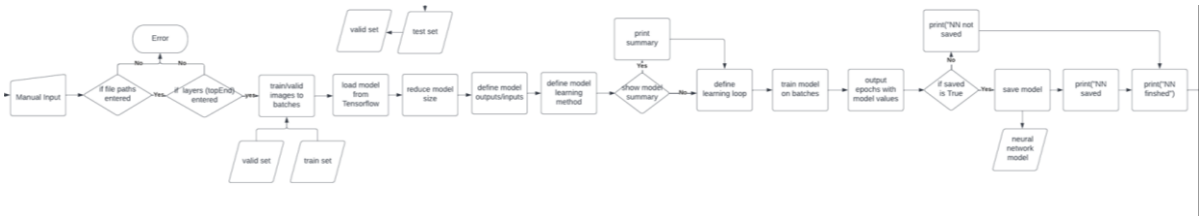

This model starts the same way and has a under interface to run the command. It ends if the exit is chosen.




This is the file maker. It makes new files for the user in place of humans doing it and causing human error. This is done checking for pre-existing file and then making them up to the users given amount. It then loops to the start.

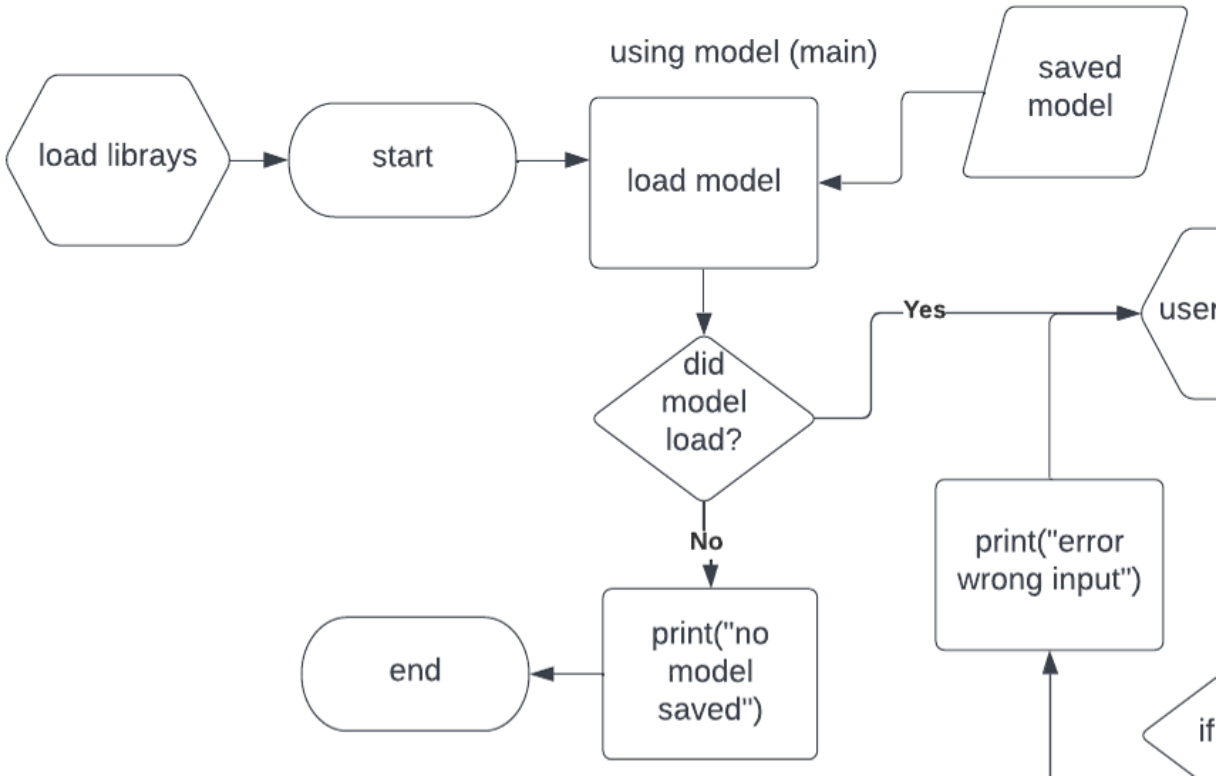



This is the paths option which checks if files are made then places randomly images from train to valid and test. It then loops back to the start

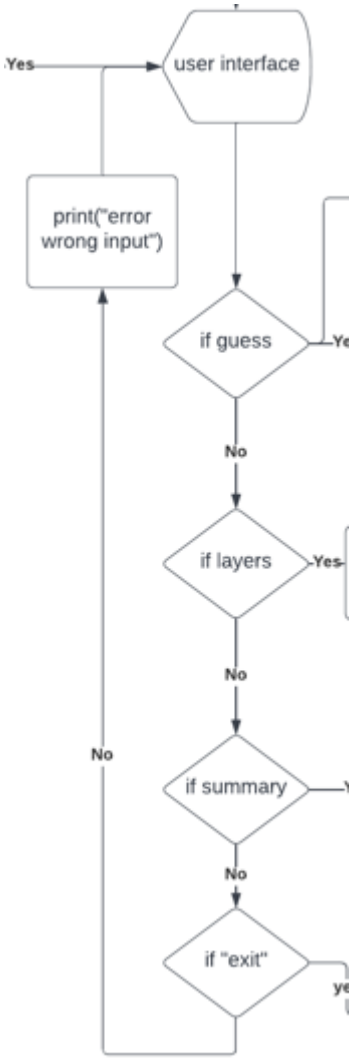



This is the process mode which will make and train the model. It starts by checking the files and then taking the files to turn them into batches to train the model. It can then save the model before looping back to the start.

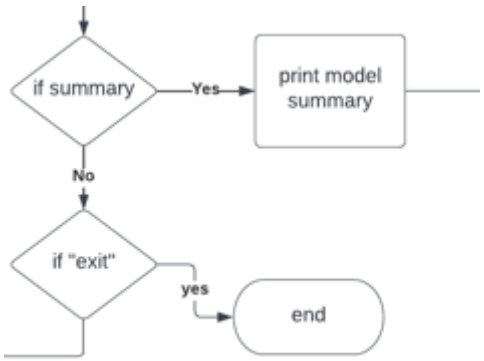
This completes the training model. Lastly is the main model.



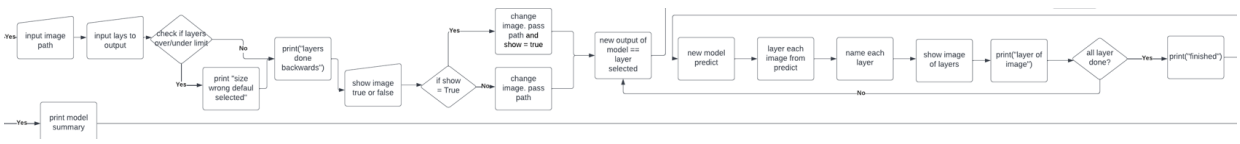

This starts as the others but also loads the model of the neural network before the user can do anything. If not saved it will end the system.



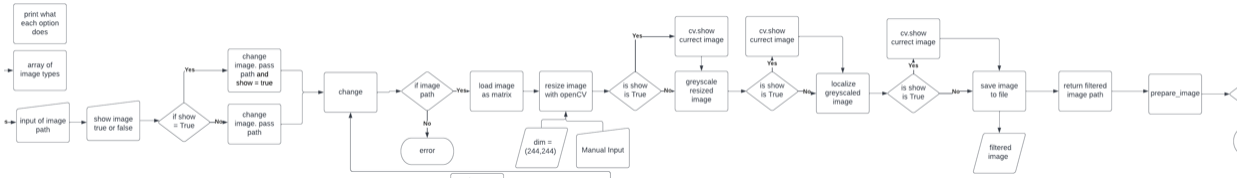

If it does load the user interface will show which will let the users pick their options or loop on itself.




If exit is selected it will end the program. If summary is shown it will give the user the summary of the model.

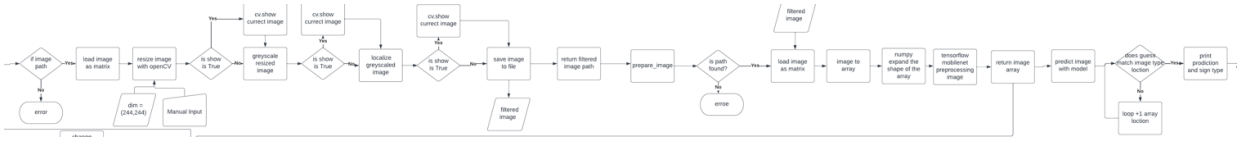


If layers is selected it will loop the user options to give each layer of the model with a loop of the feature maps as a chart to be shown to the user. It then loops back to the start.




If guess is selected it will find the image in the file and load it into the neural network to be processed. It will first filter the image to reduce the noise as it was trained on those kind of images.

#




It then takes the networks guess and gives it to the user along with what datatype that number connect to. It then loops back to the start.

### 4.3 Dataset

The design of the dataset is essential to reduce imbalance data, improve neural network training and play a large part in the overall data types.

The effect of imbalanced data on the neural network can cause the results to calculate a high accuracy when it often has a high rate of incorrect guesses. These incorrect guesses can happen when a set of data types is more prominent than other data types caused by the network, for example, only identifying that one data type with a high pass right, but due to it having 70% of the total training data, the network believes it is working at 70% accuracy when in fact all other data types are not being checked. This imbalance can also be caused by data sets having the same images repeated in which the network will only train to that image, such as if a random split is done between test and train, which would result in an incorrectly predicted level of accuracy done by the testing of the network.

This risk of incorrect accuracy can be checked by manual testing, which was done after the neural network model was trained; however, before this, the data set for this model was given rules in its production.

“the German traffic sign recognition benchmark” Mykola (2018) was the base dataset that will be edited. This dataset has 42 different image types; however, they all consist of different amounts of images, as file 24 has 270 images compared to file three, which has 1,410. A rule that is needed to reduce data imbalance is having a similar number of images in each file, as this reduces the above risk of incorrect model accuracy. To do this, the largest files were kept up to seven, and this is for a large amount of training data is still needed; however, only a small range of different image types was needed to demonstrate a working model. Seven was chosen as it was more significant than 2, there was a drop in the number of images per file apart from a few larger files, and some images were not chosen due to the signs being very alike and not showing a detailed variation which was believed to need in a demonstration. 7 was also chosen as more was not needed. A table of these files and their file amounts can be seen below, showing the wide range of amounts that can cause incorrect readings.

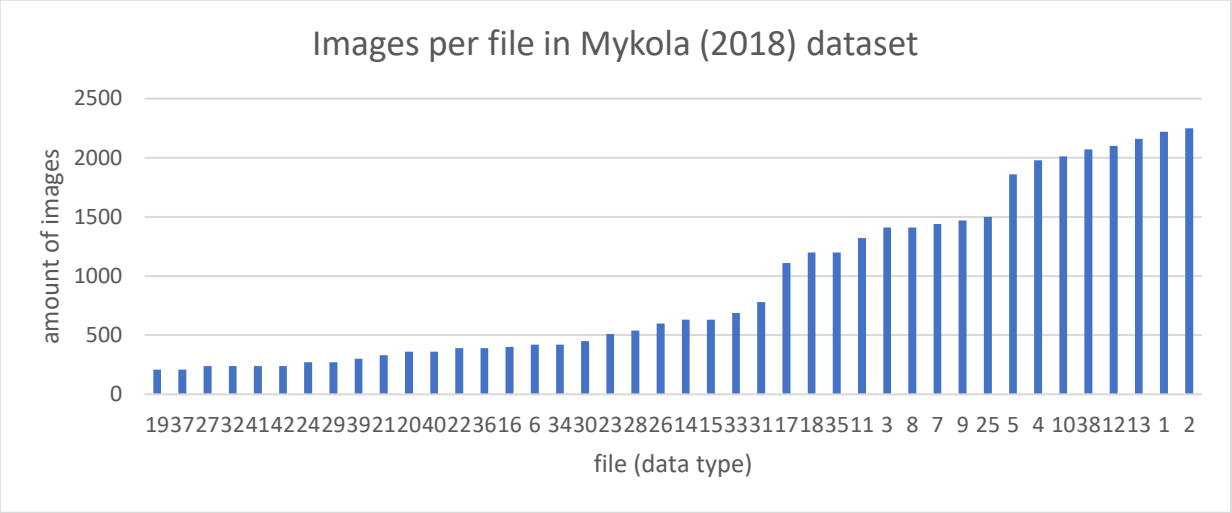



Table showing images per file from the Mykola (2018) dataset.

The new dataset uses files 1, 2, 3, 4, 11, 13, 17, 18. This changes the average from the old data set from 924.5 to 1659 which shows how the new data set while smaller in data type has a greater number of images in each file. This new dataset size can be seen below.

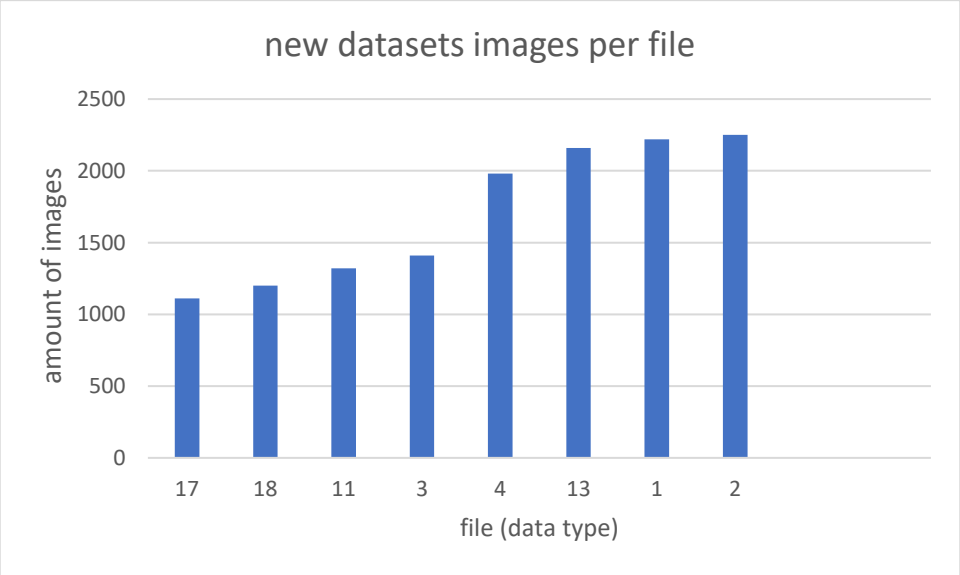



Table showing images per file from the new dataset which is taken from Mykola (2018) dataset.

Another way to reduce the risk of incorrect data is to randomly pick images from train data and move them too valid. This removes the risk of using the same images for testing and training while also removing human error in picking images. This can be seen with the flow chart of the model training random sort function in the appendix and with this code below:

```
def randomSort(location, fileTotal=10, SampleSize=30, testSampleSize=5):
    paths = {"trainPath": str,
             "validPath": str,
             "testPath": str}

    for z in range(0, fileTotal):

        onlyFiles = [f for f in listdir(location + "/train/" + str(z)) if
                     isfile(join(location + ("/train/" + str(z)), f))]  # finds all files in dict

        samples = random.sample(onlyFiles, SampleSize)
        print("samples", samples)
        for j in samples:  # select all random file to be moved
            shutil.move(location + "/train/" + str(z) + "/" + j,
                        location + "/valid/" + str(z))  # moves from train to test

        onlyFiles = [f for f in listdir(location + "/train/" + str(z)) if
                     isfile(join(location + "/train/" + str(z), f))]  # finds all files in dict again as some moved

        test_samples = random.sample(onlyFiles, testSampleSize)
        for k in test_samples:
            shutil.move(location + "/train/" + str(z) + "/" + k, location + "/test/" + str(z))

    paths["trainPath"] = location + "/train"
    paths["validPath"] = location + "/valid"
    paths["testPath"] = location + "/test"

    print("random sort done")
    return paths
```

This python code from the artefact shows how the path to the training set file is given, called location, then this algorithm first finds all the path inside such as files 1 to 25 then checks all the images and randomly picks up to the given amount, called sampleSize and testSampleSize, images to move to the pathway valid and test. Then it returns the paths it just used to move the images with.

This code randomly moves a set number of images from the train file to the test and valid files.

All these were used to balance the data on the neural network for the model, and in doing so, a new dataset was produced; however, these datasets are still not complete. Before splitting the training set-up, a filter was applied to each image and saved. This is to reduce the noise of the image in such a way that the main points of the image stay the same such as the sign, but other background issues are removed. This means the network does less work to understand an image, and it will improve the number of correct guesses.

This noise reduction is done by changing the shape of all images in the dataset to dimensions 244 and 244, which the mobile net model uses. A greyscale is applied, and then a localized threshold is done; this threshold will change unconnected pixels of the image to white and connected pixels to black at a selected group size of pixels. An example can be seen below.

Image of a no entry sign being filtered by the artefact (not to scale). The unedited image is the leftmost image, the middle image has been reshaped then greyscaled and the rightmost image has been thresholded. Image from Ingham, J. (2019).

This noise reduction was used on the whole dataset which was achieved by the artefact code seen below and can be seen in the Flow chart of image filtering of file in the image filtering model and a single image filter in Model of main (using model) within the appendix. The code below from the artefact

shows how this works from the image filter model however a similar code found between lines 164 to 202 from the main model.

```
164  def changeImage(directory, show=False, dim=(244, 244), localizes=True):  # Change images to s
165      window_name = 'Image'
166
167      if not path.exists(directory):
168          raise Exception("cvImageChanger: findImage: no file exist, check location")
169
170      image = cv.imread(
171          directory)
172
173      resized = cv.resize(image, dim,
174                          interpolation=cv.INTER_AREA)
175
176      if show:
177          print("press 'ESC' to exit")
178          cv.imshow(window_name, image)  # shows image ?remove or make command based
179          i = cv.waitKey(0)
180          if (i == "ESC"):
181              cv.destroyAllWindows()
182
183      grey_image = cv.cvtColor(resized, cv.COLOR_RGB2GRAY)  # changes to grey
184
185      if show:
186          print("press 'ESC' to exit")
187          cv.imshow(window_name, grey_image)  # shows image ?remove or make command based
188          i = cv.waitKey(0)
189          if (i == "ESC"):
190              cv.destroyAllWindows()
191
192      if localizes:
193          image = Localize(grey_image)
194
195      if show:
196          print("press 'ESC' to exit")
197          cv.imshow(window_name, image)  # shows image ?remove or make command based
198          i = cv.waitKey(0)
199          if (i == "ESC"):
200              cv.destroyAllWindows()
201
202      return image
```

This method used here is a updated method seen from Sajjad, K.M., 2010 using the first 2 methods. These are changed however to use OpenCV 2 an updated code where some of the code syntax is changed meaning the codes needed to be updated as such, this includes the ‘Localize’ command seen below:

```
72  def Localize(img):
73      ret2, item = cv.threshold(img, 0, 255,
74                                cv.THRESH_BINARY + cv.THRESH_OTSU)
75
76      return item
```

This updated version of OpenCV was used to reduce the risk of unfixed bugs, faster programming and to work with the Python 3.9 used.

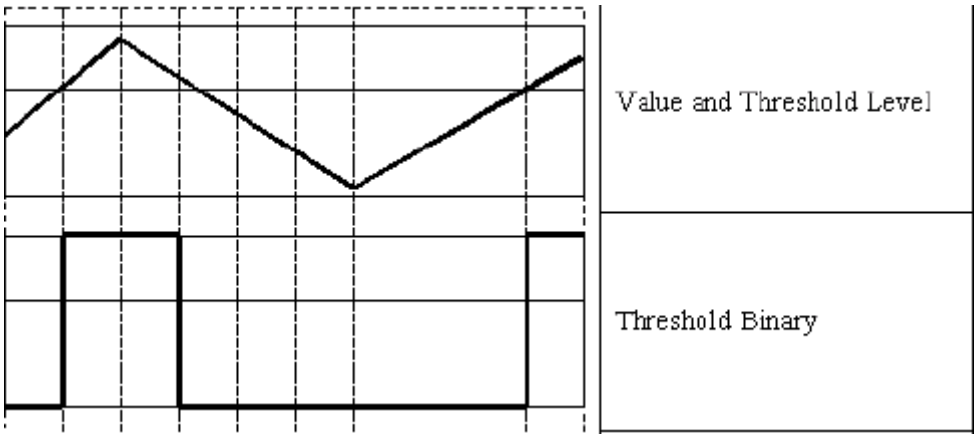
The algorithm used here works by checking the path to the directory exists in line 167 where if not found an error is raised with the message “cvImageChanger: findImage: no file exist, check location”. The directory is passed from where the code was called such as in guess() and layer() in the main code model and in files() in the filter model. If the path does exist it open the image with cv.imread as an array which is then resized by the OpenCV command cv.resize. the dimensions for the resize are 244 by 244 however these can be changed when the function is called.

The Boolean known as ‘show’ is false however can be changed when called. If true then if functions at lines 176, 185, 195 will let the code inside them run. This code is method used to show the image as it is changed starting with the unchanged image between lines 177 and 181, the resized and grey scale image

between lines 186 and 190 and finally the Localize thresholded image between lines 196 and 200. These functions by first showing the user a message of “press Esc to exit” then using cv.imshow with the current version of the image and window_name of ‘image’. This shows the image to the user however it needs a delay or would be removed quickly as the code moved on. A delay is caused by cv.waitkey which will wait forever with the use of 0 until a keypress from the user is used. If this keypress is an ‘Esc’ button it will use cv.destrotyAllWindows() to remove the cv.imshow image. This is repeated for all ‘if show:’ in the function. A future model may wish to remove this all or nothing function with a more detailed control however that was not need for this project.

The next command after resized is grey_image which uses the command cv.cvtColor. the resized image is given to the command along with the code RGB2GRAY which tell the command in which way to change the colour. This produces the grey image while maintain lots of details however this method may not work with non RGB types of images such as CMY OR SRGB so for future models a type tester could be used to check the image type before changing it. This change may reduce the speed but should improve the product.

The last command of this function is the Localize function which can be turned off when the function changeImage is called by changing the value localizes to false. This command call a different function then returns the image back to changeImage, this was done in order to make the command more readable however on future works could be kept without the need to call it. This command takes the grey scaled image and uses the cv.threshold command to change it. As seen used in docs.opencv.org. (n.d.) documentation this command uses the cv.THRESH_BINARY and cv.THRESH_OTSU to produce a filtered image. These filter types are used to inform the cv.threshold of the calculation needed to produce the correct filter types which are the OTSU algorithm which chooses the optimal threshold value and binary threshold which is the activin type meaning it can only be on or off or in this case black or white can be seen below:




Threshold binary from docs.opencv.org. (n.d.) showing when the value peaks the binary threshold is high or one while when it is not peaked the binary is low or zero.

The image is then returned to the called function. This is the end of the function and returns the now changed image array with the respaced, filtered image.

The main model use of this function has a few changes. This is due to the image being saved to a file in the function as seen below:

```
22      name = os.path.basename(os.path.normpath(directory))
23      currectDic = os.getcwd()
24
25      if not path.exists(currectDic + "/run_images"):
26          os.mkdir(currectDic + "/run_images")
```

First it gets the name of the image file from the given directory using the os.path.basename(os.path.normpath(directory)) commands. These two commands work together to do this with norm path first normalizing the path such as removing extra slashes or dots and then the base name command will take the last part of the path such as ‘image.PNG’ from ‘home/images/image.PNG’.

Then this code loads the path that the python file is located at and names it as currectDic using the os.getcwd() command. It then checks the checks if that location has a file inside it called run_images. If it doesn’t a new file is made. This file will be used to store filtered images inside of it. For future improvements a way to access this file from inside the code could be produced reducing the need to run the same code again also there is no proception from replacing a image with the same name.

Then at the end of the code instead of a returned image array it is saved to this fold run_image using cv.write under the name gotten from line 22. This image item is the filtered image. A return command then returns a path of the location of run_image file and the name of the image saved. This can be seen below:

```
65      cv.imwrite(currectDic + "/run_images" + "/" + name, item)
66      return (currectDic + "/run_images"), name
```

This is the completed image filtering code used by the artefact. However, to change the whole dataset code will be used to reduce human error and save time. This algorithm can be seen in the image filter flow chart model found in the appendix.

This system to filter all images then save them to a different file was used to reduce the need to filter images repeatedly before being used in training and due to the way TensorFlow batches work the images would need to be filtered beforehand. This algorithm was also designed so if the model had different requirements than first understood, from the research, a simple change of command could be given however due to the number of settings presented and their use by others a user interface was added to reduce complexity of use.

This code was split into groups which will be known as 1. One image, default value 2. One image, changed value 3. File of images. These groups will be explained below with examples.

The basic of all the groups was a list of values called ‘settings’ and calling the function changeImage() which is spoken about above which will filter a image. These settings would be used in place of the functions arguments, the information that is passed into them, the settings would be used.

```
9           settings = [False, (244, 244), True]  # defaults settings
```

This list on line 9 is a default setting layout. This first False stands for changeImage show which will make the code not show the image to the user, the second value of (244, 244) is the value for dim in changeImage and will let the image to reshaped to that size and lastly the final True is for localizes in changeImage which will let the image be threshold or if changed will keep it just respaced and grayscale. This can be seen then the function is called as seen in the image below on line 43.

```
43                  q = changeImage(directory, settings[0], settings[1], settings[2])
```

However each group then does things differently. This code will be broken up to produce a more readable report.

For group 1, One image and default value, to be used it requires the argument ‘one’ to be True, by default it is False as this is not a function used by this report.

```
7   def files(directory, newDirectory, layer=1, one=False):
8       if one:  # one image for system to change
9           settings = [False, (244, 244), True]  # defaults settings
10          print("current settings: directory = ", directory, "show=", settings[0], "dim=", settings[1], "localizes=",
11                settings[2])
12          i = input("change settings? (Y)")
13          i = i.lower()
14          if i == "y" or i == "yes":  # change settings for one image
```

This ‘one’ is checked by an if statement on line 8. The setting is then set to their default values on line 9 before being shown to the user in order to reduce the need to check the code. A option to change the image is then presented and if a ‘Y’ or ‘YES’ is entered then the if statement on line 14 will let the, change it. For this group any other input such as ‘no’, ‘n’, etc then they will continue with the default settings.

```
45          else:  # default settings for one image
46              print("running default image ")
47              a = changeImage(directory)
48              cv.imwrite(newDirectory, a)
```

Due to the if statement not running an else statement on line 45 is used to call the function image change. Before this is done a message informing the user of the default settings being used on a image is used on line 46, then the function is called without any added augments as they’re the same as the default changeimage() arguments. Then the function returns the image it is saved as ‘a’ which is then saved to the newDirectory given on line 7 when the function files is called. This is done with cv.write and saves ‘a’ to this location.

This finishes the first group. the next group is 2 with one image and changed values. This method starts the same as the first group 1 however at the input on line 12 a yes or y is given. This runs the if function on line 14.

```
14      if i == "y" or i == "yes":  # change settings for one image
15          for x in range(len(settings)):
16              print("type new input in same format as: ", settings[x])  # loop the different settings
17              b = input()
18              if type(settings[x]) is tuple:  # changes size of image
19
20                  try:
21                      eval(b)
22                  except:
23                      print("wrong input value or no value. !using default!")
24                  else:
25                      if len(eval(b)) == 2:
26                          print("layout correct, changing values to", eval(b))
27                          settings[x] = eval(b)
28                      else:
29                          print("wrong value size. !using default!")
30
31
32              elif b.lower() == "true" or b.lower() == "false":  # chages show, and localizes settings
33                  b.capitalize()
34                  settings[x] = bool(b)
35                  print("correct value, changing to:", b)
36
37              else:
38                  print("value input wrong or no input. !using default!")
39
40          print("new settings: directory = ", directory, "show=", settings[0], "dim=", settings[1], "localizes=",
41                settings[2])
42          print("running")
43          q = changeImage(directory, settings[0], settings[1], settings[2])
44          cv.imwrite(newDirectory, q)
```

This algorithm starts with a loop that list the total value of the settings array e.g. 1, 2, 3. Then it prints this setting and asks the user to input a new value in the same format as it was written such as for setting[2] which at default is (244,244) it would need a input like (100, 4). then the code checks the setting value so setting[x] where x = 1 would be for the show setting and would be False if in default. If it is the second value, it would be for dim with is done in a tuple which is checked on line 10.

A try command is then used on line 20. This is used to reduce this risk of the code being misused which could cause it to stop working or damage the work. Eval(b) is used to check the value is valid. If the value is wrong or there is no value entered by the user a warning message is sent to them, this warning would not stop the program but would keep the setting the same value. In future works a more forgiving menu could be produced but it is not needed for this report. However is eval(b) does work then the else command is run in which the length of the b is found and if equal to two then a message saying it is corrected if given the setting[2] is changed. If the length of the b is wrong, then a message informing the user that it is wrong is sent and default values is used.

If the if command on line 20 is wrong a second check is done for True or False values. This is done by checking the string value of b and if ‘true’ or ‘false’ then if it is it is changed to have b start with a capital then changed to a Boolean value and saved as setting[x]. a future check could be the value of the setting to insure a Boolean is not put in place of the tuple. If this elif statement on line 32 also fails, then a warning message is sent that a wrong input was given and the default will be used.

Once this is all done and the loop is finished it prints the new settings to the user and a message to inform them it is running. It then sends these need setting as aruments to the changeImage() and saves the returned filtered image with cv.imwrite.

This completes group 2. Group 3 File of images which works a lot like group two however the ‘one’ argument is false. It starts at like 49 with an else command connected to the first if on line 8. It changes after the message to the user of running.

```
84          print("running")
85
86      filentitles = {}
87
88      for file in os.listdir(directory):
89          print(directory + "/" + file)
90          filename = os.fsdecode(file)
91
92          if os.path.isdir(directory + "/" + file):
93              print("a", file)
94              if directory == newDirectory:
95                  filentitles[filename] = 0
96
97              if filename in filentitles:
98                  print("two file found with same name. input new name. old name:", filename)
99                  k = input()
100                 while True:
101                     if k in filentitles:
102                         print("all ready named file", k, ". choose new name")
103                         k = input()
104
105                     else:
106                         break
107
108                 filentitles[k] = 0
109                 apath = newDirectory + "/" + k
110                 os.mkdir(apath)
111
112             else:
113                 filentitles[filename] = 0
114                 apath = newDirectory + "/" + filename
115                 try:
116                     os.mkdir(apath)
117                 except FileExistsError:
118                     print("file already named that. made copy")
119                     copyAmount = 0
120                     while True:
121                         try:
122                             os.mkdir(newDirectory + "/" + filename + str(copyAmount))
123
```

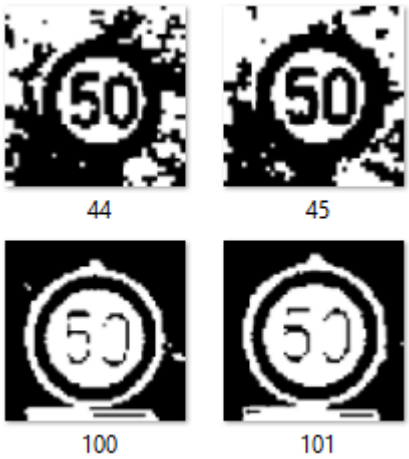
This code firstly goes through each file of the given path and add their name to a dictionary if it's new with the value 0 this can be seen between lines 92 and 95, this only happens if the new path is the same as the old path. Then if the file name already exists in the dictionary, it asked for a new name for that file to be named from the user between lines 97 to 106. Then a new file is made after what the user wants to call it and it's added to the dictionary.

If is doesn't already exist then a new file named the same as the old one is made. This has some safety check to ensure the files are not called the same thing, if they are the name is changed to have a number on the end to 100. This can be seen between 112 and 122 above and 124 to 132 below. If over 100 files are named this it auto stops for to many copies of that file have been made. This was put in place to make the process much easier for a person to do quickly however the risk or the system thinking it is a virus or damaging a computer means a limit was put in place.

```
124                         except:
125                             copyAmount += 1
126                             if copyAmount > 100:
127                                 raise FileExistsError("warning: over 100 copy of", (newDirectory + "/" + filename), ". auto stopping")
128
129
130                         else:
131                             print("copy made")
132                             break
133
134         print(filentitles)
135
136         if layer == 1:
137             for names in filentitles:
138                 counter = 0
139
140                 for file in os.listdir(directory + "/" + names):
141                     if file.endswith((".jpg", ".png")):
142                         print(file)
143                         newImage = changeImage(os.path.join((directory + "/" + names), file), settings[0], settings[1],
144                                                settings[2])
145                         holder = (newDirectory + "/" + names + "/" + str(counter) + ".jpg")
146                         cv.imwrite(holder, newImage)
147                         counter += 1
148
149
150
151         elif layer == 0:
152             for file in os.listdir(directory):
153                 if file.endswith((".jpg", ".png")):
154                     print(file)
155                     newImage = changeImage(os.path.join(directory, file), settings[0], settings[1], settings[2]
156                                            )
157                     holder = (newDirectory + "/" + "copy.jpg")
158                     cv.imwrite(holder, newImage)
159
160     print("finished")
```

Once these files were made it would show them all to the user on line 134 to reduce confusion. Then depending on the layer it will check each file inside the folder. If the found file is a jpg or png file it will show it's name and filter it using changeImage() with the chosen settings. The new image will then be saved to the newly made file inside the new path. The name for the image will start with 1 and increase every loop. The layer works by selecting how many files it must open. For one layer it opens all the files inside the file with the names just saved in filentitles in the old path. However, this only works if the old files have the same name as what was made. Layer 0 works by opening all files in that file given. This is done between lines 136 and 158. This will cause all images inside these files to be filtered and then saved to a new folder with the same or user chosen named files inside.

An example of these images inside the files can be seen below:

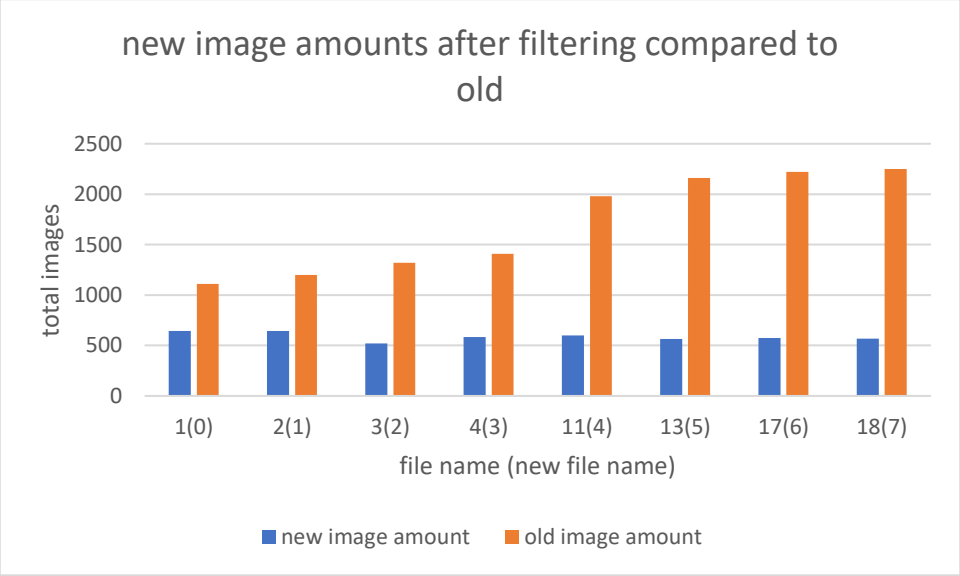



Filtered images of 50 mph signs.

All groups finish with a print function informing them of completion on line 160. This is the final line of the code for filtering images in file groups.

In the future to make this a more readable it could be split into different functions which can reduce the time it takes to make edits to the code. Another benefit could be a checker to check that the file locations it has been given do exists on the system as this could save errors later when no file is found, or no image saved. All these features let the training data be changed before the dataset is used to train the neural network model to an accepted level.

Due to some of the quality of the database images such as very low resolution some images were removed. This was measured to a human tester being able to guess what the sign was from the image, if a human couldn't understand a langue, then it's unlikely to be reliable to learn from. The final values can be seen below:




This did produce a more balanced dataset however with less practice data with an average of 587 images per file. In the future a quality check of data should be done however doing one after has worked in this case.

### 4.3 Training of model

The model of a neural network requires many steps to gain a high value of accuracy with its guesses. These steps include a balanced dataset as done above but also to choose a model which has the correct inputs and outputs, to create a validation set with randomly picked images, to provide it with batches, to give the network model values such as learning rate, loss and metrics and also to define the training amount.

Also, to improve the speed a user could use this system other features such as a user interface (UI) is provided. This is to make the use of the system far smoother.

The first function is seen below called file_maker(). This function is used to produce or check for files known as train, valid, test. These files and the included files inside with the labelled data should already have been produced in the filtering system.

```
def file_maker(location, topEnd):
    os.chdir(location)
    try:
        os.mkdir("train")
        os.mkdir("valid")
        os.mkdir("test")

    except FileExistsError:
        print("warning file already found")

    for num in range(0, topEnd):
        try:
            os.mkdir("train/" + str(num))
            os.mkdir("valid/" + str(num))
            os.mkdir("test/" + str(num))

        except FileExistsError:
            print("warning file", num, " already found")
            pass

    print("files finished")
```

This can be seen done first by saving the current file location to the location given by user. It then tries to make the files ‘train’, ‘test’ and ‘valid’ however if these files already exists then an error is given to the user and the system is stopped. If this except is not raised it then tries to produce a file inside the just made files from 0 up to the given number from the user, called topEnd. if a file already with that number is found a message informs the user of such and moves onto the next number.

Once finished a message informs the user that it’s done. in future works this file_maker() may wish to be removed as it’s made redundant from the filter functions see before and this report states when using the model training that a dataset it needed.

More work on the files is needed however which is not redundant. To reduce human error of copying files and to improve the speed of the system a way to randomly sort images into different file was produced.

```
def randomSort(location, fileTotal=10, SampleSize=30, testSampleSize=5):
    paths = {"trainPath": str,
             "validPath": str,
             "testPath": str}

    for z in range(0, fileTotal):

        onlyFiles = [f for f in listdir(location + "/train/" + str(z)) if
                     isfile(join(location + ("/train/" + str(z)), f))]  # finds all files in dic

        samples = random.sample(onlyFiles, SampleSize)
        print("samples", samples)
        for j in samples:  # select all random file to be moved
            shutil.move(location + "/train/" + str(z) + "/" + j,
                        location + "/valid/" + str(z))  # moves from train to test

        onlyFiles = [f for f in listdir(location + "/train/" + str(z)) if
                     isfile(join(location + "/train/" + str(z), f))]  # finds all files in dict

        test_samples = random.sample(onlyFiles, testSampleSize)
        for k in test_samples:
            shutil.move(location + "/train/" + str(z) + "/" + k, location + "/test/" + str(z))

    paths["trainPath"] = location + "/train"
    paths["validPath"] = location + "/valid"
    paths["testPath"] = location + "/test"

    print("random sort done")
    return paths
```

This functions called randomSort() first makes a list of file paths for each of the main files in use train, test, valid (ttv) but each is only given a place holder string. Then a loop is stared from 0 to the default value of fileTotal which is 10 or can be changed by the user. This is the number of datatypes so for this report's dataset would be set to 8. For each loop the images in that value file are found and stored to onlyFiles. A random sample up to the given sampleSize which can be changed by the user is taken from this onlyFiles. These samples are then shown to the user. Then code then moves these samples from the training file to the valid file. This is then repeated for the test file with the testSampleSize being default 5. This is done for every file in the dataset train file.

Once finished the paths of the ttv are saved. Then the user is messaged that it's done, and it returns the paths to where it was called from in the code. For future works a model with more selection should be used. This should include a file checker to ensure no issues of lost files if none is found or to show which folders are missing. The file moving and random selection could be moved into a different function to reduce the number of lines therefore reducing the complete to a reader however this is not critical and was there for not added to save on time. This data will be used by the model to produce a high guess rate while removing human mistakes.

The next function is called process() which introduces the model and it's defined variables. This is done with the use of moblieNet from the Tensorflow library which is then trained with the dataset produced as seen above. The model can be edited and changed to produce a higher level of success and also to process data differently which may also alter the results of its training and therefor later accuracy.

```
def process(fileL, layers=10, saved=False, savedName="my_model"):
    tBatch = ImageDataGenerator(
        preprocessing_function=tf.keras.applications.mobilenet.preprocess_input).flow_from_directory(
        directory=fileL["trainPath"], target_size=(224, 224), batch_size=10)
    vBatches = ImageDataGenerator(
        preprocessing_function=tf.keras.applications.mobilenet.preprocess_input).flow_from_directory(
        directory=fileL["validPath"], target_size=(224, 224), batch_size=10)

    mn_mobile = tf.keras.applications.mobilenet.MobileNet()

    x = mn_mobile.layers[-6].output  # run tests

    out = Dense(units=layers, activation="softmax")(x)
    untrained_m = Model(inputs=mn_mobile.input, outputs=out)

    for layer in untrained_m.layers[:-2]:
        layer.trainable = False
    untrained_m.compile(optimizer=Adam(learning_rate=0.0001), loss="categorical_crossentropy", metrics=["accuracy"])

    x = input("view summary? (Y/N)")
    x.lower()
    if x == "y":
        untrained_m.summary()

    untrained_m.fit(x=tBatch,
                    steps_per_epoch=len(tBatch),
                    validation_data=vBatches,
                    validation_steps=len(vBatches),
                    epochs=25,
                    verbose=2,
                    )

    if saved:
        untrained_m.save("saved_model/" + savedName)
        print("NN saved")
    else:
        print("nn not saved")
```

The function for process()takes the augment fileL, which should be a dict containing the paths to each saved location of the randomly spilt data, layers which is the same as above an should be the same as the amount of datatypes which for this report is 8. Saved is a boolean function which is set to False as default and savedName is a string which is set to ‘my_model’ as default.

The first line produces batches known as tBatch from the train dataset. This uses the command ImageDataGenerator which randomly changes the images which expands the size of the dataset but also lets it learn unseen data, data which has never been learnt before because it hasn’t been seen. This ImageDataGenerator doesn’t do this as the dataset has already been setup with plenty of images however it does use pre-processing function which applies the command it’s given to every image used.

This command for preprocess_input is from the TensorFlow, karas and MobileNet library which alters the image given to it to fit the format the model requires. This can be adding another dimension to produce batches. More details can be found TensorFlow Core v2.8.0(N/A).

The images for these are then processed by flow_from_directory(). This is needed for imageDataGenerator to function for it dictates in this reports case the directory, where the images are coming from, the target_size which is the size of the image and will be resized if needed however the filter will have already done that, and the batch_size which sets the size of each batch to be run.

This model setting such as batch_size was randomly picked due to the use of a new tool however it worked to the amount it needed. In future works it should be tested with other sized batches in order to find a higher level of succeed with the neural network however to size time this method was not taken.

This was then repeated but on the valid dataset. More research has suggested that this is not needed however this is more of a suggestion as if a batch does miss a valid image, it will likely be used in a later batch or later epochs. This should be investigated more for the future but was not changed due to higher results and time constrains.

These commands can be seen between lines 69 and 73 for both train and valid data. It was not used on a test path as this will be done with human management but a automated checker could be added.

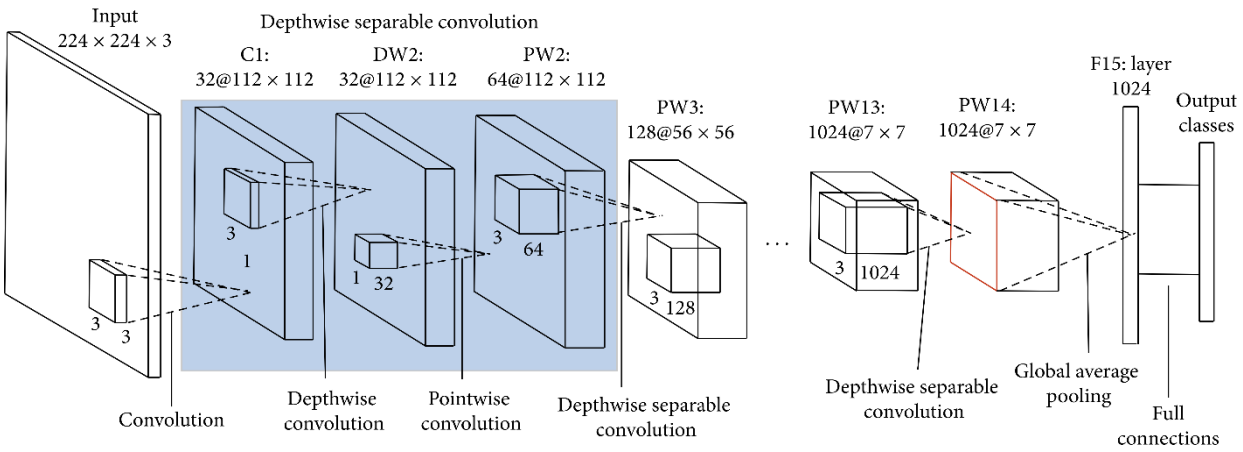
To summarise these steps the code first:

- Takes a batch of 10 from the datasets train or valid.
- Check and change the size if needed to 224, 224.
- Pre-process the images to fit the MobileNet model input.
- Get saved to tBatch or vBatch.

The next command is used to load the model from MobileNet. This can be seen on line 75. This is the full MobileNet model which is loaded from TensorFlow and saved under the name of ‘mn_mobile’. Then on line 77 the output is selected at -6 which changes when the output ends.

Due to this model using CNN, convolutional neural networks, it reduces the image but without removing any of the details. This works by taking the pixel values, RGB in this case, and fitting it in a matrix which adds up all the values. This new matrix formed from the old one is known as the convolved feature which is then used in the next layer of the neural network. When the layer gains this it can output a score based on different feature it can detects such as in this case the shape of a sign which will place it in a class.

A pooling layer is also used but this acts as more of a way to save power when data is being processed. An average pooling is used in this case which takes the average of a matrix of pixels and then shrinks all values to that in the next layer.




This can be seen here in this diagram from Wei Wang, al et, 2020 of the MobileNet model v2 which is the same used in this report.

These details are important to understand line 77. This command is removing the last 6 lines and putting the output layer closer to the start. This doesn’t remove the important layers that are needed for the model to function such as the layers before the global_average_pooling. This edit needed to be made due to a new output being made. This model is used for other image recognition however needs to be changed to suit this reports work. These layers are however saved to ‘x’ for later use.

This means adding a new output layer with is done on line 79 and 80. First the layer detailed need to be made with command Dense from TensorFlow which makes a dense layer meaning all the input are fully

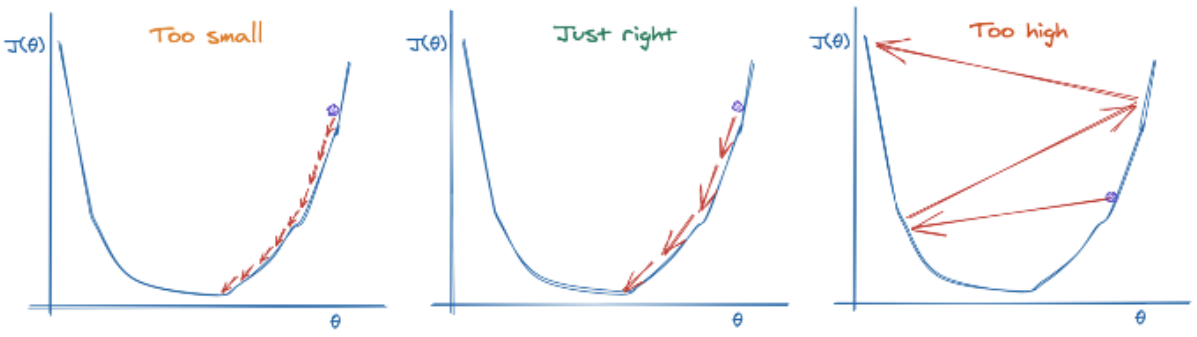
connected to it, this is used to accept any outputs from the pooling layer which will be weighted based on which class the neural network layers believe it is. The augments for this are the units which will be based on a number in this case 8, the layers augment, which will be the output it gives based on class. It will also be given the activation 'softmax'. The 'x' is also added at the end to ensure meta data from the Output of the model is given to the new layer.

The activation softmax which means the values that are given are treated such that the largest value is changed to 1 while all overs are given a 0. This ensures a single output is given based on the neural networks best classification, also known as best guess as to which classes it belongs to.

Line 80 shows the command model being used which is done to make a new model based on the mn_mobile net and the new dense layer just made. This command connects them as the mn_mobile as the input to the output of the dense layer.

Then between lines 82 and 83 the layers are lopped. This locks the last layers from being trainable. This works by locking the values the nodes are given. The value of a neural network is that the layers can be trained to new data then finding ways to improve on that value with back propagation which uses maths to change the weight value of nodes with gives greater results. For this model the last 2 have been set to false which mean their values cannot be changed which results in a greater value of accuracy. This gain is because the model has already been trained on other datasets when this report loaded it and so the end layers are already suited to do their task however the method for choosing these layers to freeze is up to the programmer so it may be worthwhile for future works to test it. With a accuracy of 97% this was not an issue for this report.

On line 84 it set the model using the command compile which use a loss, metrics, and optimizer which configures the model for training. These methods are what are used to set the weight values when it is being trained. "The purpose of loss functions is to compute the quantity that a model should seek to minimize during training." Team, K. (n.d.) and the use of the categorical_crossentropy is used in this report. This is for it is used with a multi-class classification model like this one with 8 output labels and it uses a method of giving an output label, like in this case 30 mph, a binary value which is then changed to a category from these values. Metric is then given as accuracy; this is used to find the performance of the model in use. This is a very simple way to measure it as it can do so by checking if the label, the guess, is equal to the data type. This is given as a percentage and the machine uses this to check how much it is improving or not improving. Then lastly is the optimizers which find the gradient which is the way that the model knows how to change the weights to improve the results. Adam is used as it stands for the Adam algorithm which is good for "computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters" Kingma et al., 2014. It uses a learning rate of 0.0001 which is the size of the changes made which if too small can cause slow learning and if too big can cause the learning rate to jump massively. An example graph is given below.




A chart showing the effects of learning rate on a system from Chen, B. (2020).

This is all used to train the model however before training beings a user can choose between lines 86 and 89 if they wish to see the summary. This is a simple if statement which if ‘y’ is inputted then the summary of the new model is shown, known as untrained_m. this is a simple command which shows the user the different layers of the network, an example of this reports summary can be seen below.

```
conv_dw_13_relu (ReLU)       (None, 7, 7, 1024)        0
_________________________________________________________________
conv_pw_13 (Conv2D)          (None, 7, 7, 1024)        1048576
_________________________________________________________________
conv_pw_13_bn (BatchNormaliz (None, 7, 7, 1024)        4096
_________________________________________________________________
conv_pw_13_relu (ReLU)       (None, 7, 7, 1024)        0
_________________________________________________________________
global_average_pooling2d (Gl (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 8)                 8200
=================================================================
Total params: 3,237,064
Trainable params: 8,200
Non-trainable params: 3,228,864
_________________________________________________________________
```

The end of this reports summary where different layers can be seen. As can be seen the MobileNet model has been edited to remove the last 6 layers up to the global_average_pooling2d and a dense layer with 8 nodes added.

This model is then trained using the fit command on line 91 with arguments from lines line 91 to 97. This is the command that trains the network to generate a higher accuracy. This is done by slicing the dataset given to it into new batches then iterating over the images for the whole data set for each epochs. The data for training is given as x from the tbatch made before with the length of tbatch being used to tell the systems how many images it needed to send into the system, this is then repeated with the validation_data and vbatch however this data will be just used to check the value of which it is learning not improve the learning, checks that over fitting is not happening. Then the epochs is selected, this will be the amount of times the model is looped. 25 was chosen as around 20 epochs the model stops improving greatly and starts lowering its accuracy sometimes. This lowered score is not taken to the next layer and only the best model is kept so this doesn’t negatively affect the model. The next 5 do have some small increase of 2% to 3% which is why it is set to 25 epochs. In the future a lowering of the Adam optimizer with greater epochs maybe used for a greater accuracy however this is only minor improve that this report doesn’t need. Verbose is what is used to show the details od how the model is doing. The values of 0 shows nothing, 1 shows a progress bar and 2 like in use shows the epoch value. Values 1 and 2 also include the values of loss, accuracy from the training data, val_loss and val_accuracy which is from the valid data. An example from training the model used fot this report can be seen below.

```
Epoch 1/25
2022-04-06 21:08:59.985874: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185]
471/471 - 39s - loss: 1.4682 - accuracy: 0.4769 - val_loss: 0.8942 - val_accuracy: 0.7292
Epoch 2/25
471/471 - 38s - loss: 0.7691 - accuracy: 0.7490 - val_loss: 0.6286 - val_accuracy: 0.8042
Epoch 3/25
471/471 - 39s - loss: 0.5999 - accuracy: 0.8105 - val_loss: 0.5170 - val_accuracy: 0.8458
Epoch 4/25
471/471 - 37s - loss: 0.5093 - accuracy: 0.8407 - val_loss: 0.4451 - val_accuracy: 0.8875
```

Epochs 1 to 4 with values from training this report model. With a starting accuracy of 4766 with is about 48% rounded up.

```
471/471 - 35s - loss: 0.1534 - accuracy: 0.9672 - val_loss: 0.1505 - val_accuracy: 0.9583
Epoch 23/25
471/471 - 35s - loss: 0.1477 - accuracy: 0.9704 - val_loss: 0.1477 - val_accuracy: 0.9667
Epoch 24/25
471/471 - 36s - loss: 0.1421 - accuracy: 0.9709 - val_loss: 0.1415 - val_accuracy: 0.9625
Epoch 25/25
471/471 - 35s - loss: 0.1371 - accuracy: 0.9745 - val_loss: 0.1384 - val_accuracy: 0.9667
2022-04-06 21:24:25.486222: W tensorflow/python/util/util.cc:348] Sets are not currently co
NN saved
nn finished
```

Epochs 22 to 25 with values from training this reports models. With final epoch 25 having the accuracy of 97%.

From this epoch data this report can understand that this model has improved and therefore been trained successfully. The accuracy and val_accuracy are very near each other's value showing that the dataset and training have likely been very balanced. If they were not alike it would show an issue with training has happened. The loss values show how many were done incorrectly on a scalar value and by having it lower means less is being lost which is good.

Once this model is trained it can be used to predict a sign from the data types however this is not done in this code section while the time it takes to train a model can take a while which is not very useful. To save time from retraining a model a saved model can be made. This is done between line 99 and 103. If when the function is called saved is chosen as a option then the model will save to the given name passed to it. The code then informs the user of saving it or it would say not saved if the user chose to not save it. The user will then in informed that the program has finished on line 105.

The test data is not used here and will be used to test the model by human input.

To make this tool easier to use a inter interface was added. This is a basic text input user face.

```
162 ▶  if __name__ == "__main__":
163        main()
```

This code here is used to inform the code to run the main function which is the user interface.

```
108 def main():
109     while True:
110         print("")
111         selection = input("please select 'file_maker' to produce images files,"
112                           " 'random sort' randomly spilt images to different files, "
113                           "'rprocess' to randomly split images and process model,  "
114                           "'process' to process/train model"
115                           "'exit' to end: ")
116         selection.lower()
117         if selection == "file_maker" or selection == "f":
118             a = input("please input file location")
119             b = input("please image types amount")
120             file_maker(a, int(b))
121         elif selection == "random sort" or selection == "r":
122             a = input("please input file location")
123             b = input("please image types amount")
124             path = randomSort(a, int(b))
125             print("paths:" + str(path))
126         elif selection == "rprocess" or selection == "rp":
127             a = input("please input file location")
128             b = input("please image types amount")
129             paths = randomSort(a, int(b))
130             print("image done. processing model")
131             q = input("save model? (True/False)")
132             q.capitalize()
133             if q == "True":
134                 s = input("save model name? ")
135                 process(paths, int(b), saved=bool(q), savedName=s)
136             else:
137                 process(paths, int(b), saved=bool(q))
138         elif selection == "process" or selection == "p":
139             c = {"trainPath": str, "validPath": str}
140             a = input("please input path for training")
141             c["trainPath"] = a
142             a = input("please input path for valid")
143             c["validPath"] = a
144             b = input("please image types amount")
145             q = input("save model? (True/False)")
146             q.capitalize()
147             if q == "True":
148                 s = input("save model name? ")
```

This is the main() which runs the user inface. It largely uses if and input commands to run the function seen above. It starts with a while True loop which lets the user input anything and not end the program by mistake. The a string is sent to the user to inform them of the options they can use which can then be type into a input. The input is lowered so that when the string is compared it will be the same even if capatized. A if, elif and else check is then done, first for the options ‘file_maker’, then ‘random sort’, then ‘rprocess’, then ‘process’, then ‘exit’. If these all fail the else is used to inform the user of a issue and the loop restarts. A short hand letter can also be used to use that if which is more for cases where it’s used many times and so it will save users time.

The file_maker selection on line 117 takes the users dataset path and the amount of datatypes before calling file_maker() seen above with those inputs. This is used to make the files of ttv.

The random sort selection on line 121 also takes the path and amount of data types. It then uses the randomSort() function with the given values. Once finished it also give the user the paths to all fine locations. This randomly sorts the images between ttv.

The rprocess on line 126 uses two function and takes the same two inputs of file path and data type values. It then calls the randomSort() function but instead of giving the paths it uses them to run process() function. This also takes two more inputs of if the user wants to save the model and the name such a model could be given when saved to a file location. If save is not used then these new inputs are not given to the function as they're not used.

Process on line 138 works like rprocess but with already randomly sorted data. The user will need to input both the paths for train and valid. These will then be added to a dict as that what is needed as an input to the function process(). It also ask the same save question as rprocess..

An exit mode is also given which is the correct way to end the system, on line 153. This command breaks the while true loop which then lets the final command of quit() be run on line 159. This command ends the program.

The else option is given on line 155. Once they're done it exits back to the while true loop.

The user interface could use more options to change settings however this was not given as it was not needed by the report and also to reduce the amount any user would need to input and need to understand about how the system worked.

This is all the command of this model of the training. To summarise the steps:

*User interface to select function seen below -*

- *Makes files ttv if needed but should be done in the filtering model*
- *Can randomly sort images between ttv with randomsort()*
- *Can randomly sort these images then train the model without saving it or choose to save it*
- *Or can train the model without also randomly sorting image but needs to give the path manually. This can also be saved or not saved if chosen by the user.*
- *Exit can be selected to end the system*
- *An error for misspelt commands*

*Ends program.*

This gives a trained model with about 97% accuracy and a high likely hood of not being over fitted due to the similarly with val_accuracy. While some improves could be given a slight insight such as this report can give most user the ability to make such a model without many issues. While this model is useful in providing a method to show it works it is not very practical and doesn't answer the question of this report, for that this report need something to use the trained neural network.

### 4.4 Model system (Main)

To use the neural network, it requires another system to run data though it. These data points such as images are needed to understand the practical use of the system along with ways to gain answers to the reports question of how these machines understand images. This Main model should achieve this and a explained of how so will be given below.

The first command of this main model is to load the neural network.

```
9   while True:
10      try:
11          new_model = tf.keras.models.load_model("./saved_model/model_2")
12          break
13
14      except:
15          print("error finding saved model. please ensure model is in the same file location in /saved_model/ for new use under name 'model_2'.")
16          a = input("please enter model path of put exit to end program: ")
17          if a == "exit" or a == "Exit":
18              exit()
19          else:
20              new_model = tf.keras.models.load_model(a)
21              break
22
23
```

This can be seen done between line 9 and 21. This method is more user friendly as it first checks for where the file, containing the neural network, should be for this report and loads it as ‘new_model’ before ending the while loop. If not found however it instead asks the user to exit the program or give a new path to the model. If this model is not found, then the program will send an error message but it is then it breaks and lets the system carry on. It is loaded from the files using load_model which is a TensorFlow command so it will load it as if it was just trained and no editing or extra training will be needed. this is a method of using the tools already made.

Another tool that is required again is the filtering and pre-processing of the image before it’s trained.

```
24  def changeImage(directory, show=False, dim=(244, 244),
25                  localizes=True):  # Change images to selected size and grey scale.
26      window_name = "Image"
27
28      name = os.path.basename(os.path.normpath(directory))
29      currectDic = os.getcwd()
30
31      if not path.exists(currectDic + "/run_images"):
32          os.mkdir(currectDic + "/run_images")
33
34      if not path.exists(directory):
35          raise Exception("cvImageChanger: findImage: no file exist, check location")
36
37      item = cv.imread(
38          directory)  # finds image, currently using test image replace original = cv.LoadImageM("image.jpg")
39
40      resized = cv.resize(item, dim,
41                          interpolation=cv.INTER_AREA)  # replace thumbnail = cv.CreateMat(original.rows/ 10,
42
43      if show:
44          print("press ESC to exit")
45          cv.imshow(window_name, item)  # shows image ?remove or make command based
46          i = cv.waitKey(0)
47          if (i == "ESC"):
48              cv.destroyAllWindows()
49
50      grey_image = cv.cvtColor(resized, cv.COLOR_RGB2GRAY)  # changes to grey
51
52      if show:
53          print("press ESC to exit")
54          cv.imshow(window_name, grey_image)  # shows image ?remove or make command based
55          i = cv.waitKey(0)
56          if (i == "ESC"):
57              cv.destroyAllWindows()
58
59      if localizes:
60          item = Localize(grey_image)
61
62      if show:
63          print("press ESC to exit")
64          cv.imshow(window_name, item)  # shows image ?remove or make command based
```

```
65          i = cv.waitKey(0)
66          if (i == "ESC"):
67              cv.destroyAllWindows()
68
69          print(name)
70
71      cv.imwrite(currectDic + "/run_images" + "/" + name, item)
72      return (currectDic + "/run_images"), name
73
```

```
75  def Localize(img):
76      ret2, item = cv.threshold(img, 0, 255,
77                                cv.THRESH_BINARY + cv.THRESH_OTSU)
78
79      return item
```

This is the code for filtering the image which has been seen before from the image filtering model. This is needed again as the image passed to the program maybe not from the filtered folders and as such need to be filtered for the first time. It does have some small differences to the first code such as line 28 which gets the name if the file while line 29 gets the current path of where the code is running.

It then uses this need information to check for the file 'run_images'. If not found a new one will be made in the current path. It then also changed near the end with lines 69 to 72.

This is because it's saving the image to the file run_images instead of returning it to act as a way of not having to filter many images all the time and as some of the program has been made to take images from the file in different ways so just passing the image array may cause issue later on. This can be seen done with cv.imwrite to save it to the current path in the run_image file with the name of passed image name from the path. This was done on line 28. This is the new filted image which is saved.

It then returns the name of the image with the path to the run_images. As this is a file that should always exist due to the way the model works this method of passing back the path shouldn't be needed however due to the limits of time this will not be changed in the report but in the future it should be. The limits of how this is done has some benefits of under not being able to greatly affect the way the code works and reduce the risk of, for example, detecting the model or path of the image. This limiting is from the user interface.

The user interface is the fun function to run called main(). This is due to when the code is started it run the following command as seen before.

```
188 ▶  if __name__ == "__main__":
189        main()
```

This means that if the python code is called 'main' it will run the function main(). This is a way to make sure the code will start with that code and also is just apart of how python works.

```
def main():
    while True:
        print("")
        selection = input("please select 'guess' to input an image and guess which sign it is,"
                          " 'layer' to pick a image and see how each layer breaks it down, "
                          "'summary' to see total model summary or 'Exit' to end program: ")
        selection.lower()
        if selection == "guess" or selection == "g":
            guess()
        elif selection == "layer" or selection == "l":
            layers()
        elif selection == "summary" or selection == "s":
            summ()
        elif selection == "exit" or selection == "e":
            break
        else:
            print("error, option not selected. try again.")
            print("")

    quit()
```

This main() function works like the other user interface seen before with it being text based and having a while true loop. This code starts by having a endless loop In order to reduce issues of user selection the wrong order or being able to run a function then once done they can choose another option. It then shows the user the selections they can choose. A if, elif and else command chain are then used to check the user's response with a shorthand letter also being checked for quick use of multiple functions. These selections are:

- Guess which run the guess() function
- Layer which runs the layer() function
- Summary which runs the summary() function
- Exit this will break the loop and let the quit() command run ending the program
- Else will run if none of these are given and tell the user to select a different option.

All these function are run without any other inputs needed in this main() function as each function was designed more with multi use in mind and request the needed information later.

The first function of guess is designed for a user to be able to pass an image into the neural network and reply what image type it believes it to be.

```
def guess():
    signs = ["30 mph", "50 mph", "60 mph", "70 mph", "junction", "Warning (empty)", "no entry",
             "Warning (!)"]  # correct names?

    LOC = input("please input full address of image: ")
    show = input("show image? True/False ")
    show.capitalize()
    if len(show) == 4:
        bool(show)
        currectDic, name = changeImage(LOC, show)  # C:\Users\gamin\OneDrive\Desktop\final project
    else:
        currectDic, name = changeImage(LOC)
    tool = prepare_image(currectDic, name)

    predictions_single = new_model.predict(tool)
    # print(predictions_single)

    for i in signs:
        if np.argmax(predictions_single) == signs.index(i):
            print("this sign is: ", np.argmax(predictions_single), ". Also know as: ", i)
```

This guess() function start off by loading a list of the different sign types. This is saved as ‘signs’ and will be used later to label the data. As the image types for this report were labelled as numbers and not their sign name this report found that to reduce the need for an external key that list what each number corresponds to an internal label system should be added to the code itself.

The next lines from line 98 to 106 deals with the user input and filtering of the image. It starts with two input commands for the pathway to where the image is stored and if the user want to show the image as it’s being filtered. The show filtered image input is then changed to have the first letter capitalized and the rest lower case as this is the format needed to turn it from a string value to a Boolean. A Boolean is what is expected to be passed to the filter image code seen above which is why this is done. a if command is used to check if the string ‘show’ is 4 letter and if it is so then the code will pass this Boolean to the change image or if not the changeImage() function is still called but without the show Boolean value added to it. This is for a ‘true’ is four letters long and so it will only run if it’s in the correct value while false is five letter long as anything else not equal to 4 won’t run the function with the Boolean. This is not the best method and should be changed in future codes to change the Boolean first after checking it’s spelt correctly then there is no need for the if command however was level of checking was not needed in this report.

The changeImage() function then returns the value of the path and name of the new filtered image. These details are then given the function prepare_image() which changes the image to an array that suits the MobileNet network model. This array is then passed back as ‘tool’. This is done on line 106.

This image is then passed to the neural network to guess the image. This is done on line 108 sing the command predict. This command sends the single image to the neural network, runs it through the network as it would have done with training however with only one image and no back propagation this process is comparatively and based on the computer very quickly. This prediction will return what the network believes the image is which is saved as ‘predictions_single’. This could be returned to the user as it’s data type label but to save extra work on the user end as said above it is passed to a for loop.

This loop works by looping all the different item inside ‘signs’ which is the name of the sign. If this name’s index location, where it is in the list, is the same as the predicted number then print this number with the sign’s name to the user. This works as the data types id number is the same as the name of the sign’s location in the signs list for example 0 is the same as 30 mph while 1 is the same as 50 mph.

This code is what was used to test the test images from the dataset. This is done by running it and changing the different image each time. The results from this can be seen below.

| | | predicted | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| actual | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 1 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 2 | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 |
| | 3 | 1 | 0 | 0 | 5 | 0 | 0 | 0 | 0 |
| | 4 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 |
| | 5 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 |
| | 6 | 0 | 1 | 0 | 0 | 0 | 0 | 5 | 0 |
| | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |

This shows a 93% accuracy with these tests however such results could be due to other factors which will to explained later in this report.

This is the end of the guess function. It has no returns and would send the user back to the interface loop. Here other options can be selected.

Another option is the summ() function which uses the seen before command of summary.

```
116     def summ():
117         new_model.summary()
```

This shows the trained neural network layout as seen before in the training model image. This is the second last command with the last being the layer() function.

The layer function was originally intended to produce a deepdream image which would have provided the layers of feature Map on top of the original image. This would have provided a clear way to view how the network see different parts of the image however due to using MobileNet this method did not work. As seen with the pre-process command the array for MobileNet cannot be anything but a set size which did not work with the deepdream model. Different models may work but due to the constrain of time that this report is under the model could not be changed. A models that does work with deepdream are InceptionV3 seen used in a TensorFlow example found online at https://www.tensorflow.org/tutorials/generative/deepdream [accessed 08/04/2022].

This caused more research into the feature Map which the deep dream would have used to view the image with. The new method was to find and show the feature Map then check for patterns inside of them which on further research seemed to provide more data points and control than the deepdream option while producing a way to answer this reports question.

Feature Map are the apart of the neural networks method to recognise patterns and will be used later to answer this reports question however currently using this information it can be determined that the network in use also uses these maps and can be accessed. This is done below.

```
def layers():
    print("total layers: " + str(len(new_model.layers) - 3))
    a = input("input first layer: ")
    if int(a) < 2:
        print("too low, changed to 2")
        a = "2"
    b = input("to layer: ")
    if int(b) > len(new_model.layers) - 3:
        print("too high changed to " + str(len(new_model.layers) - 3))
        b = str(len(new_model.layers) - 3)
    if (int(a) - int(b)) != 0:
        print("notice: layers done backwards")

    LOC = input("please input full address of image: ")
    show = input("show image? True/False ")
    for i in reversed(range(int(a), int(b) + 1)):
        print("layer - " + str(i))

        model_cut = tf.keras.Model(inputs=new_model.inputs, outputs=new_model.layers[i].output)
        if len(show) == 4:
            bool(show)
            currectDic, name = changeImage(LOC, show)
        else:
            currectDic, name = changeImage(LOC)
        img = prepare_image(currectDic, name)

        nodeImages = model_cut.predict(img)

        amountSize = nodeImages.shape[3]
        if amountSize > 100:
            user = input("amount of images over 99. recommend a sample instead. press Y for a smaller sample.")
            user.lower()
            if user == "y":
                amountSize = 64
                print("smaller size chosen")
            else:
                print("layering of plot will be done. warning lag may happen due to lots of images. ")
```

```
            size0 = amountSize // 7
            size = amountSize % 7
            layer = 1

            fig = plt.figure(figsize=(10, 7))
            for images in range(1, amountSize):
                fig.add_subplot((size0 + size), 7, images)
                plt.imshow(nodeImages[0, :, :, layer - 1])
                plt.axis('off')
                #plt.title("MapFilter = " + str(i) + "." + str(layer))
                layer += 1

            print("image of layer: " + str(i))
            plt.show()

    print("finished")
```

The layer() function is used process an image in the trained model but removing the dense layer to produce the output as one of the other layers, these layers are given by the user. It then loops the given layer and shows the feature Map using matplotlib due to each layer having more than one feature Map.

To produce these results it first require the user to input the details of the first layer they wish to access. For this model the limit is 2 to 85 due to the input and output layer not having a feature map. If the user inputs a too small layer then it will default to the lowest of 2. Then it will ask for the highest layer that excluding the last 3 layer due to them not being able to be processed. If the user chooses a too high layer then it's set to the highest layer minus three. This removes the issue of any non-combatable layers being used which would stop the program. The program does the layers from highest to smallest for example between 2 and 10 it would go 10, 9… 2, therefore a message informing the user of this is given.

The next information needed it the image path which would be processed as to activate the feature map. While some feature maps can be viewed without an image being processed this was not achieved in this report due to how the arrays were produced. Future projects should work towards this however from a human view point these un-active maps can look odd or not make sense.

Due to the image needing to be filtered as the neural network was trained on filtered images it also asked the user if they want to see the image being filtered as seen before. However due to this being in the loop it shows the image and filters the image many times. This is not very efficient and in future versions should be moved to another selected or save the image array as a viable and have a true/false if state to check if it is the first time it has been filtered. It could also make use of the filtered image folder however this folder doesn't have any checks and can overwrite work without warning which is an issue.

Once the user has given this information the system then loops within the range of the first two values for layers. This has a plus one due to the way loops work and wouldn't loop the last layer if not done.

On line 136 it then informs the user of which layer they're on. This layer cutting to get this layer's output is then done on line 138 using the TensorFlow model command seen used in the training model. This makes a new model with the inputs of the trained neural network but with the output being the selected layer which is given from the loop.

Between lines 139 and 144 are the same image filtering and preparing of the image seen inside the guess function. On line 147 is a simulate predict seen before however now with the cut model.

AmountSize is the variable which holds how many outputs, feature maps, the model outputted will contain. For this report there can be between 64 and up to and beyond 1024 as this value. A layer value is also made on line 149 which will be used in conjunction with amountSize to work out the layering of the images.

Due to many featured maps being produce per layer a warning system was included which can give a limited sample size if over 99 images are found. This is very important as showing 1024 images can be a lot of work for the computer. This is done with a simple if command and a input which askes the user to make the choose. In the future a way to split these images should be produced however for the time given for this project this was not doable. This is done between lines 150 and 157.

The code them splits up the number of imagers per row and columns. This is done by having each row be 7 long then having as many columns as the amountSize dived by 7 has. This also include often a non-full row which is worked out from the remainder of the amount size this is also where layer is set to 1 to show the first layer for every loop. This is done on line 159 to 161.

A new figure is made which the images will be shown on using matplotlib. This is done with the figsize 10 and 7 however the user can change this with the use of the pop-up menu. Then a loop for all the images inside the feature map is run. Per each loop a new sub plot is added from line 165 with the column location being size0 plus size which stands for the divided amountSize plus remainder, the row

done automatedly by the plot with a limit of 7 along and the loops current number know as ‘image’ added last, kind of like an ID. Then the subplot adds the feature map for that ‘image’ by using imshow command. This command adds the image to the newly made subplot and get the image by plotting the array from the output of the neural network, changing it to a new image by having the 4th element be called layer and adding to it cheap loop. The axis is turned off as it’s not needed. then the layer adds one to itself. This is done between lines 164 and 169. This method could be improved by replacing layer with the loop made number, but this method works and is not limited by memory space to such a degree.

It also has the issue that large amounts of images cause the data to over lay. This can cause issue of the data being confusing however at that level a sample of data should do. This could be improved by splitting the data up between the plots however to do this took too much time and cause many issues with the displaying of feature maps. This issue included the risk that some images would be missed meaning the level of work needed could still be ineffective in the long run.

Once this plotting loop is finished it run it informs the user of the layer it’s showing and then using plt.show() command shows the charts.

It will then loop every asked layer and once finished will print ‘finished’ to show it is done. then it returns the user interface loop.

To sum up all this model’s key points:

- Filters images and prepares them.
- User interface.
- network can guess the image.
- can show networks layers.
- can exit program.

### 4.7 running requirements

To run this program some requirements are needed. for the training and use of the network a CPU was used however it can be added to the code to let a GPU be used for greater speed, but the GPU must be able to run CUDA 8.0.

The computer specs used to produce this work was:

- Processor: AMD Ryzen 5 3600X 6-Core processor 3.80GHz
- Installed RAM: 16GB
- Samsung 22inch curved monitors
- Logitech wireless keyboard and mouse
- ADATA SX8200PNP DISK
- AMD Radeon RX 5600 XT GPU

The software that was used to produce this was:

Microsoft Windows:

- Edition Windows 10 Pro
- Version 21H1
- Installed on 07/12/2019
- OS build 19043.1348
- Experience Windows Feature Experience Pack 120.2212.3920.0

All used libraries and versions of libraries, codes and anaconda listed below in the appendix. The key points Code langue: Python 3.9, Conda version : 4.10.3.

Pycharm is the system used to develop the code in and version 2021.3 was used in this project.

This use of software and hardware is used due to its availability and ability to run program.

For minimum running requirements come from the use of systems and langue's which is how these limits have been found:

| Requirements | minimum | recommended |
|---|---|---|
| Operating system | Linux- Ubuntu 16.04 or Windows 8 | Or later models |
| RAM | 5GB | 8GB |
| Disk spaces | 8.5GB | 10GB |
| CPU | Modern CPU | Multi-core CPU |
| Monitor resolution | 1024x768 | 1920x1080 |
| GPU | WSL2 via Windows 10 19044 | |
| pip | 19 or later | May need manylinux2010 |

The limits for individual system can be seen below:

for running python 3:

| Requirements | minimum | recommended |
|---|---|---|
| Operating system | Linux- Ubuntu 16.04 or Windows 7 | Linux- Ubuntu 17.10 or Windows 10 |
| RAM | 2GB | 4GB |

Details of running requirements from Instructions to install Python 3. (n.d.).

for running anaconda:

| Requirements | minimum | recommended |
|---|---|---|
| License | Free use and redistribution | |
| Operating system | Windows 8 or newer, 64-bit macOS 10.13+, or Linux, including Ubuntu, RedHat, CentOS 7+, and others. | |
| System architecture | Windows- 64-bit x86, 32-bit x86; MacOS- 64-bit x86; Linux- 64-bit x86, 64-bit aarch64 (AWS Graviton2 / arm64), 64-bit Power8/Power9, s390x (Linux on IBM Z & LinuxONE). | |
| Disk spaces | 5GB | |

Details of running requirements from Instructions to docs.anaconda.com. (n.d.).

for running PyCharm:

| Requirements | minimum | recommended |
|---|---|---|
| RAM | 4GB | 8GB |
| CPU | Modern CPU | Multi-core CPU |
| Disk Space | 3.5GB | SSD drive with 5GB |
| Monitor resolution | 1024x768 | 1920x1080 |
| Operating system | Windows 8, macOS 10.14, Linux that supports Gnome, KDE , or Unity DE | 64-bit version of Windows, macOS, or Linux |

Details of running requirements from Instructions to PyCharm Help. (n.d.).

for running TensorFlow:

| Requirements | minimum | recommended |
|---|---|---|
| python | 3.7 | 3.10 |
| Operating system | Ubuntu 16.04 , macOS 10.12.6 , Windows 7 | Or later OS's |
| GPU | WSL2 via Windows 10 19044 | |
| pip | 19 or later | May need manylinux2010 |

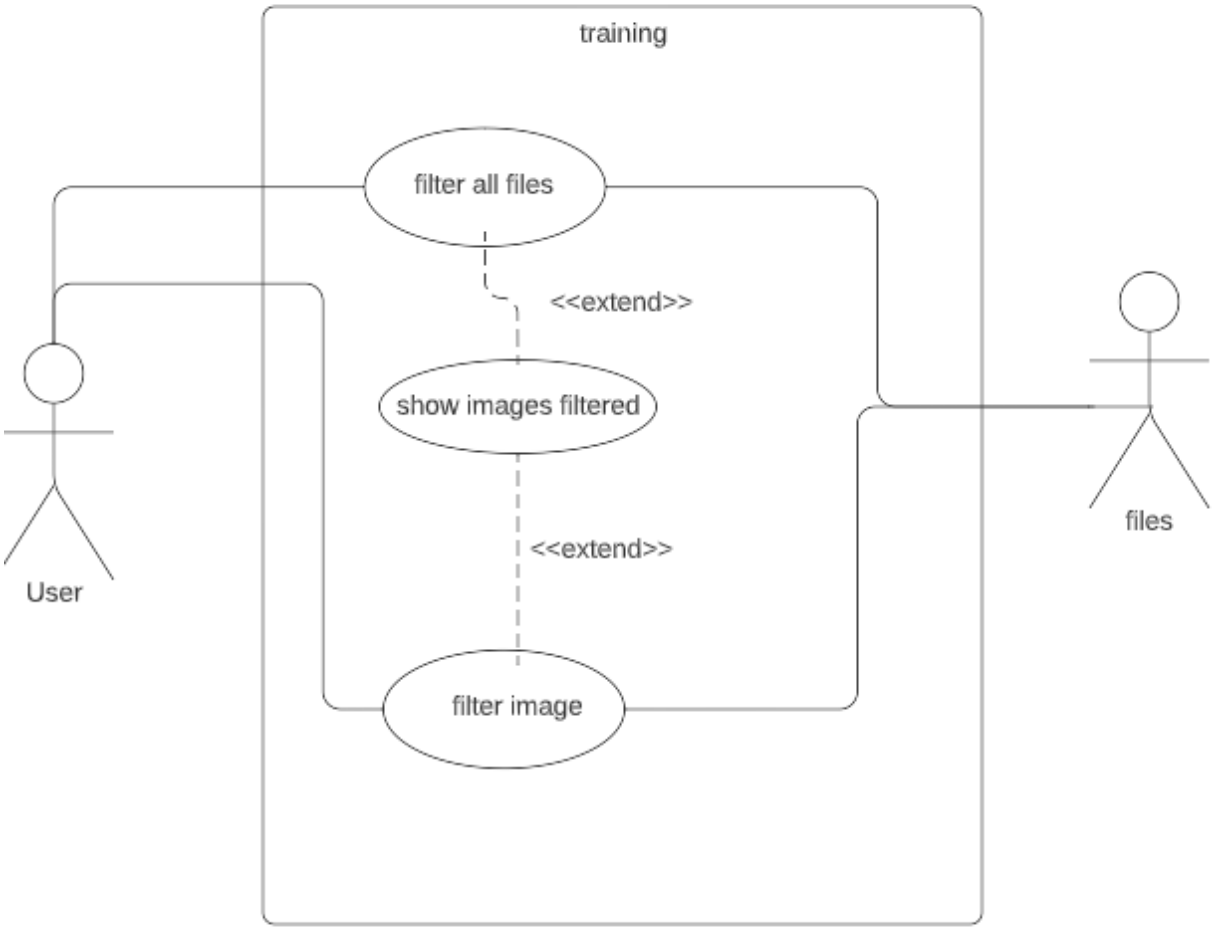
Or use google colab.

Details of running requirements from Instructions to TensorFlow. (n.d.).

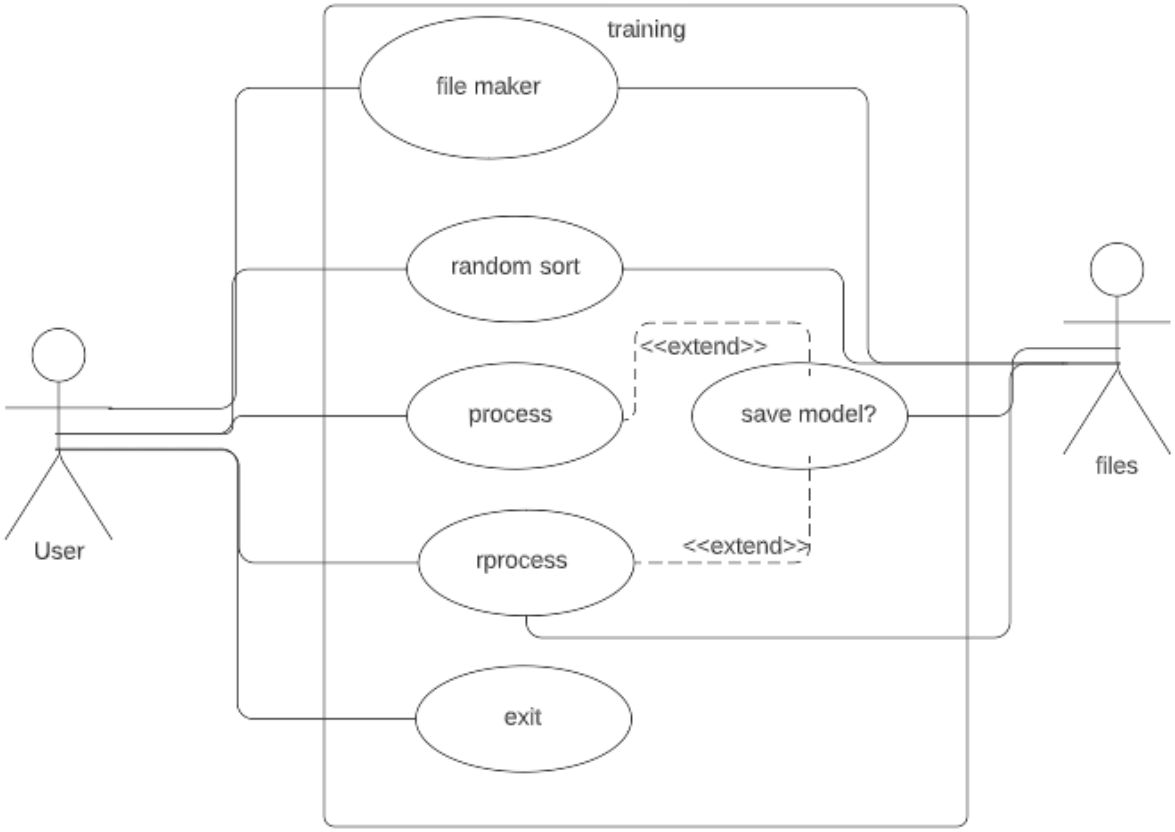
### 4.8 user side

With the use of a user interface and pop out windows to shows data it is important to show what these results look like not just from the code side but also the users. For this the use of the program PyCharm was used to display console prompts, this should have no effect on how the code outputs information apart from where it is displayed.

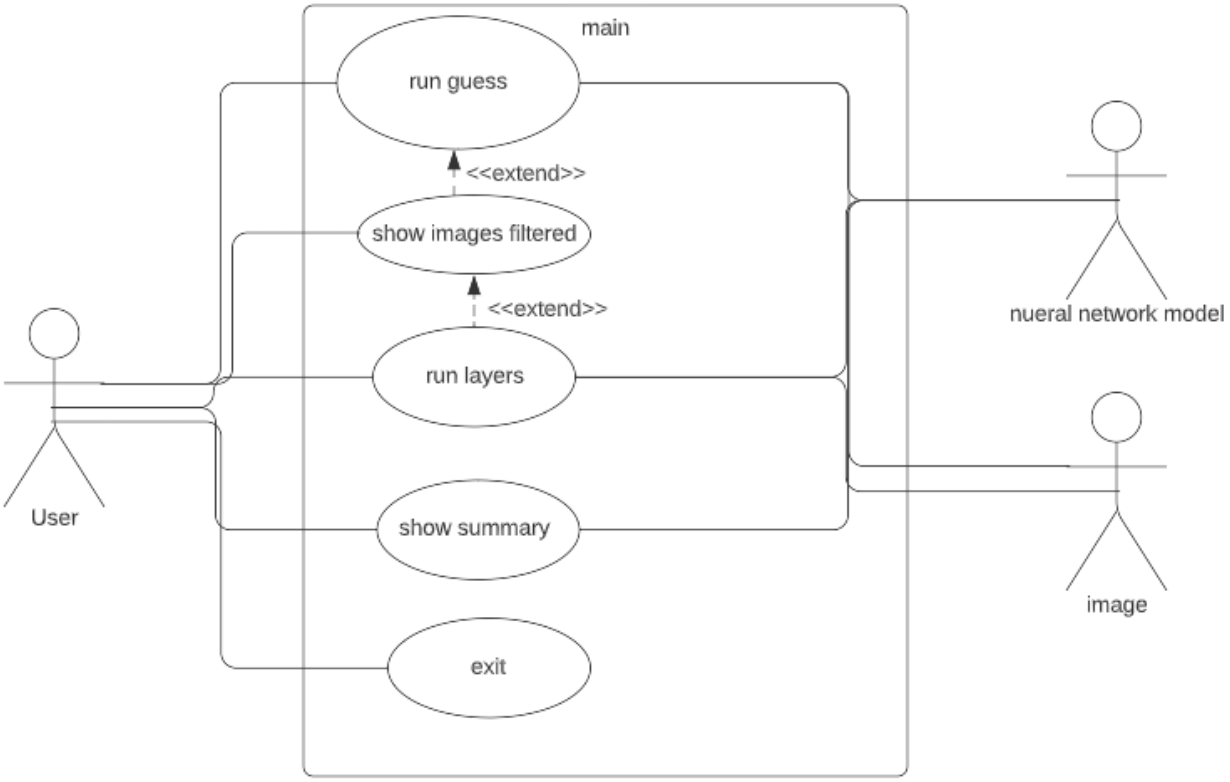
A short use case diagram has been included to show how these systems should work. First is filtering




This is a use case diagram for filtering. Filter has only two options which can be extended to show the filtering of images. It then gives the new file to the file actor. The next is a use case diagram for training the model.




This is the use case diagram for training. This model has two actors of user and files. It shows how all the system effect the file apart from the model training which may not effect the file if not saved. Last is the use case diagram for main.

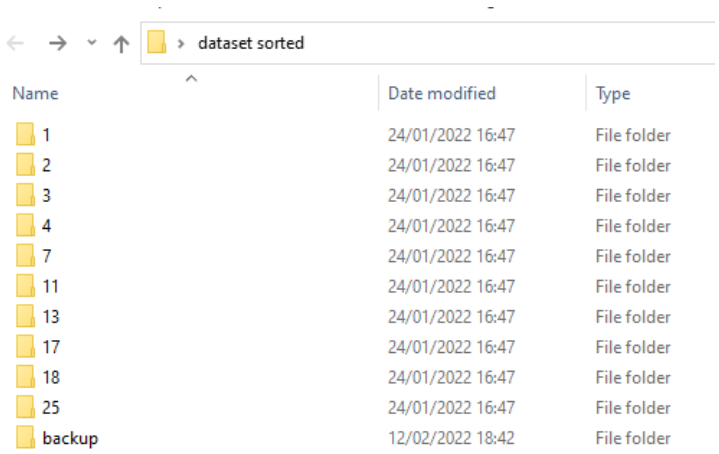



This is the use case diagram for main. In this diagram the actor of user is interacting with the system and can select 4 things. It can also extend the system to show the filtered image. It then connects to the other actors of the system such as the nueral network and the image that is stored in the system. This is all the basic models from the user view but now a more detailed breakdown of the user interface is given below using screen shots.

For image filtering model it needs one file with images and one without. It will then change all those images to filtered images in the new folder. This example also uses layer equal to one as it needs to enter each file going one layer deep.

```
212    files("C:/Users/gamin/Desktop/dataset sorted", "C:/Users/gamin/Desktop/test_file", layer=1)
```

Input needed to get moves and filter images.



The layout of a layer one file.

Once this is done running the code gives this output. Using the default settings by using a input ‘n’ seen in green. It then gives the name of each file path and file name. then a name of the files is given as a dict.

Then the name of each image changed is printed.

```
current settings: directory =  C:/Users/gamin/Desktop/dataset sorted show= False dim= (244, 244) localizes= True
change settings? (Y)n
C:/Users/gamin/Desktop/dataset sorted/1
a 1
C:/Users/gamin/Desktop/dataset sorted/11
a 11
C:/Users/gamin/Desktop/dataset sorted/13
a 13
C:/Users/gamin/Desktop/dataset sorted/17
a 17
C:/Users/gamin/Desktop/dataset sorted/18
a 18
C:/Users/gamin/Desktop/dataset sorted/2
a 2
C:/Users/gamin/Desktop/dataset sorted/25
a 25
C:/Users/gamin/Desktop/dataset sorted/3
a 3
C:/Users/gamin/Desktop/dataset sorted/4
a 4
C:/Users/gamin/Desktop/dataset sorted/7
a 7
C:/Users/gamin/Desktop/dataset sorted/backup
a backup
{'1': 0, '11': 0, '13': 0, '17': 0, '18': 0, '2': 0, '25': 0, '3': 0, '4': 0, '7': 0, 'backup': 0}
00001_00000_00000.png
00001_00000_00001.png
00001_00000_00002.png
00001_00000_00003.png
00001_00000_00004.png
00001_00000_00005.png
```

Once finished it then shows ‘finished’.

```
00007_00047_00029.png
finished
```

If run again then it will produce copys as files already have the given name.

```
file already named that. made copy
copy made
C:/Users/gamin/Desktop/dataset sorted/11
a 11
```

This will inform the user as seen above. The effect of this can be seen bellow in the file with each copy file having a 0 after it.

| Name | Date modified | Type |
|---|---|---|
| 10 | 11/04/2022 19:21 | File folder |
| 11 | 11/04/2022 19:12 | File folder |
| 13 | 11/04/2022 19:12 | File folder |
| 17 | 11/04/2022 19:12 | File folder |
| 18 | 11/04/2022 19:12 | File folder |
| 20 | 11/04/2022 19:21 | File folder |
| 25 | 11/04/2022 19:12 | File folder |
| 30 | 11/04/2022 19:21 | File folder |
| 40 | 11/04/2022 19:21 | File folder |
| 70 | 11/04/2022 19:21 | File folder |
| 110 | 11/04/2022 19:21 | File folder |
| 130 | 11/04/2022 19:21 | File folder |
| 170 | 11/04/2022 19:21 | File folder |
| 180 | 11/04/2022 19:21 | File folder |
| 250 | 11/04/2022 19:21 | File folder |
| backup | 11/04/2022 19:11 | File folder |
| backup0 | 11/04/2022 19:21 | File folder |

The default setting can then be changed. This can be seen below with the use of ‘y’ as the input as green.

```
change settings? (Y)y
type new input in same format as:  False
True
correct value, changing to: True
type new input in same format as:  (244, 244)
150,150
layout correct, changing values to (150, 150)
type new input in same format as:  True
False
correct value, changing to: False
new settings: directory =  C:/Users/gamin/Desktop/dataset sorted show= True dim= (150, 150) localizes= True
```

This then lets the user give different inputs for other values. Its looped by the order seen in the menu. It can then be seen with the new options. This changes to the show to true makes image pop up work, done with cv.imshow() with ‘esc’ being used to close them.


Unchanged image in image pop out.


Greyscaled and changed in size in image pop out.


Localized image in image pop out.

If the setting change is given wrong as seen below then the menu will use the default and tell the user.

```
change settings? (Y)y
type new input in same format as:  False
no
value input wrong or no input. !using default!
type new input in same format as:  (244, 244)
maybe
wrong input value or no value. !using default!
type new input in same format as:  True
yes
value input wrong or no input. !using default!
new settings: directory =  C:/Users/gamin/Desktop/dataset sorted show= False dim= (244, 244) localizes= True
```

This is all the user inface for this model apart from some very alike codes such as for only one image.

For the model training no extra code input is needed and can be run with the use of the user interface.

The first user input is for which mode to use. This mode below is the file maker. For this the path to the place the file will be made should be inputted and the amount of file. It then shows it's finished

```
please select 'file_maker' to produce images files, 'random sort' randomly spilt images to different files, 'rprocess' to randomly split images and process model,  'process' to process/train model'exit' to end: file_maker
please input file location
please image types amount
files finished
```

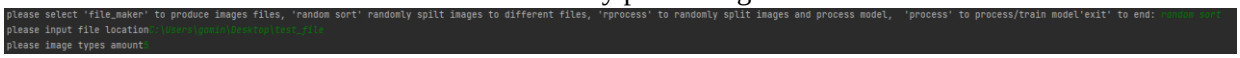
As seen below 3 files are made. Test, train, valid.

| | | |
|---|---|---|
| test | 11/04/2022 20:13 | File folder |
| train | 11/04/2022 20:13 | File folder |
| valid | 11/04/2022 20:13 | File folder |

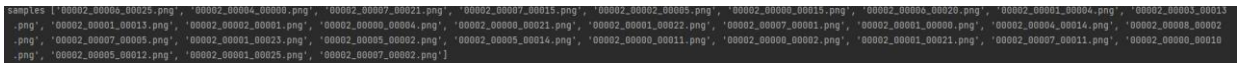
For each pf these files more files are made inside upto the amount given. This starts at zero however.

| Name | Date modified | Type | Si |
|---|---|---|---|
| 0 | 11/04/2022 20:13 | File folder | |
| 1 | 11/04/2022 20:13 | File folder | |
| 2 | 11/04/2022 20:13 | File folder | |
| 3 | 11/04/2022 20:13 | File folder | |
| 4 | 11/04/2022 20:13 | File folder | |

The next mode used is random sort which randomly picks images for the valid and test folders.



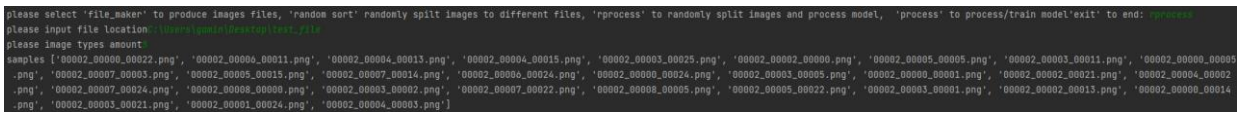

It needed the user to input where the test, train and valid files and the amount of datatypes that will be swapped around.



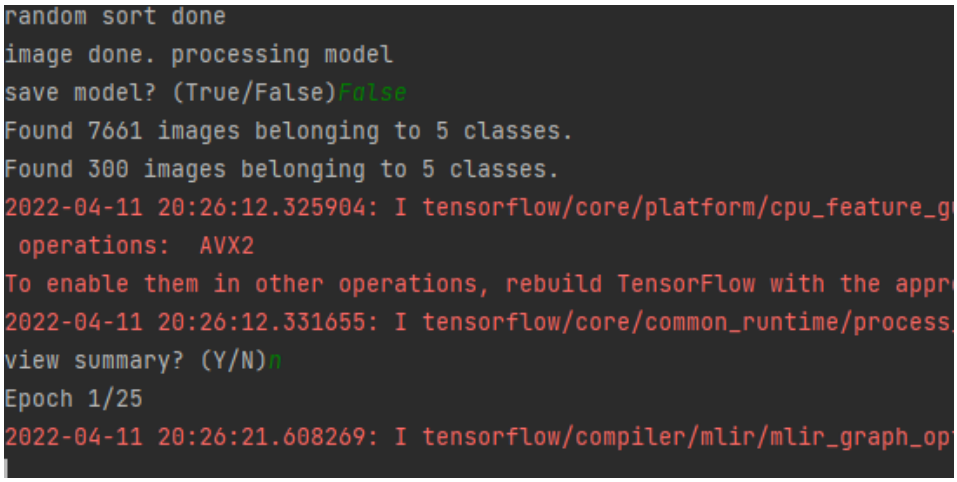

It then shows the user the files it will be moving to different folders.


Then is tells the user it's done before the user interface gives them the needed path data.

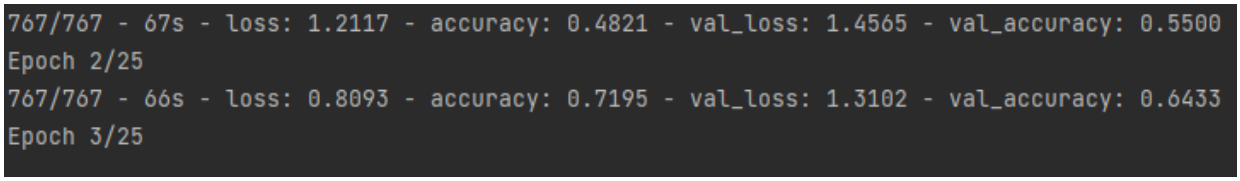
The next mode of rprocess uses this same starting method.




It then informs the user it has moved the folders before needing input on if the user wants to save the model or view the summary which in this case it is not. It then starts the training. The red text is from tenserflow informing the user that the GPU is not being used and the CPU is.




Then as the model trains it gives the user information on how well it's preforming.




If the model summary input is a 'y' then the gives a summary of what the model looks like as seen below:

```
view summary? (Y/N)y
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
```

Once the code is finished if the code was not saved it will show.

```
nn not saved
nn finished
```

However, if saved is given a second input is asked for the name. here the name del is made as this is just a test

```
random sort done
image done. processing model
save model? (True/False)True
save model name? del
Found 8445 images belonging to 5 classes.
```

Once finished it shows.

```
NN saved
nn finished
```

To show it's done and has saved.

This concludes the rprocess model. The process mode works a lot like this as it uses the same code however the user must give the paths themselves as seen below.

```
please select 'file_maker' to produce images files, 'random sort' randomly spilt images to different files, 'rprocess' to randomly split images and process model,  'process' to process/train model'exit' to end: process
please input path for training:C:\Users\gomin\Desktop\test_file\train
please input path for valid:C:\Users\gomin\Desktop\test_file\valid
please image types amount:3
save model? (True/False)False
Found 8445 images belonging to 5 classes.
Found 30 images belonging to 5 classes.
```

As seen above this is the same as seen before but with 2 inputs and the image will not be swapped. It could in the future just take the path both files are found in, but this is not needed at this time. This leads the final User interface tools such as seen below the 'no option selected'.

```
please select 'file_maker' to produce images files, 'random sort' randomly spilt images to different files, 'rprocess' to randomly split images and process model,  'process' to process/train model'exit' to end: help
error, option not selected. try again.
```

And the quit command. As it is a python code PyCharm informs the user that it finished with exit code 0.

```
please select 'file_maker' to produce images files, 'random sort' randomly spilt images to different files, 'rprocess' to randomly split images and process model,  'process' to process/train model'exit' to end: exit

Process finished with exit code 0
```

This concludes the training model and leads to the main model. This main model first loads the user interface.

```
C:\Users\gamin\anaconda3\envs\finalProject\python.exe "C:/Users/gamin/OneDrive/Desktop/final project/main.py"
2022-04-12 14:24:40.010976: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical
 operations:  AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-04-12 14:24:40.011842: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with default inter op setting: 2. Tune using inter_op_parallelism_threads for best performance.

please select 'guess' to input an image and guess which sign it is, 'layer' to pick a image and see how each layer breaks it down, 'summary' to see total model summary or 'Exit' to end program:
```

This shows the seen before warning of not using the graphics card and then shows the given options a user can input.

```
please select 'guess' to input an image and guess which sign it is, 'layer' to pick a image and see how each layer breaks it down, 'summary' to see total model summary or 'Exit' to end program: guess
please input full address of image: C:\Users\gamin\OneDrive\Desktop\final project\1113055.jpg
show image? True/False false
2022-04-12 14:37:21.877010: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
this sign is:  6 . Also know as:  no entry
```

This is the guess option. Once's given the address of an image it will filter it which can be used but not in this example as it has been seen before.

It will then output what the neural network believes the sign to be. In this case it is correct and got the sign no entry.

```
please select 'guess' to input an image and guess which sign it is, 'layer' to pick a image and see how each layer breaks it down, 'summary' to see total model summary or 'Exit' to end program: summary
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
conv1 (Conv2D)               (None, 112, 112, 32)      864
```

The next mode is summary which just shows the summary of the model as seen before.

```
please select 'guess' to input an image and guess which sign it is, 'layer' to pick a image and see how each layer breaks it down, 'summary' to see total model summary or 'Exit' to end program: layer
total layers: 85
input first layer: 2
to layer: 2
please input full address of image: C:\Users\gamin\OneDrive\Desktop\final project\1113055.jpg
show image? True/False false
layer - 2
image of layer: 2
```

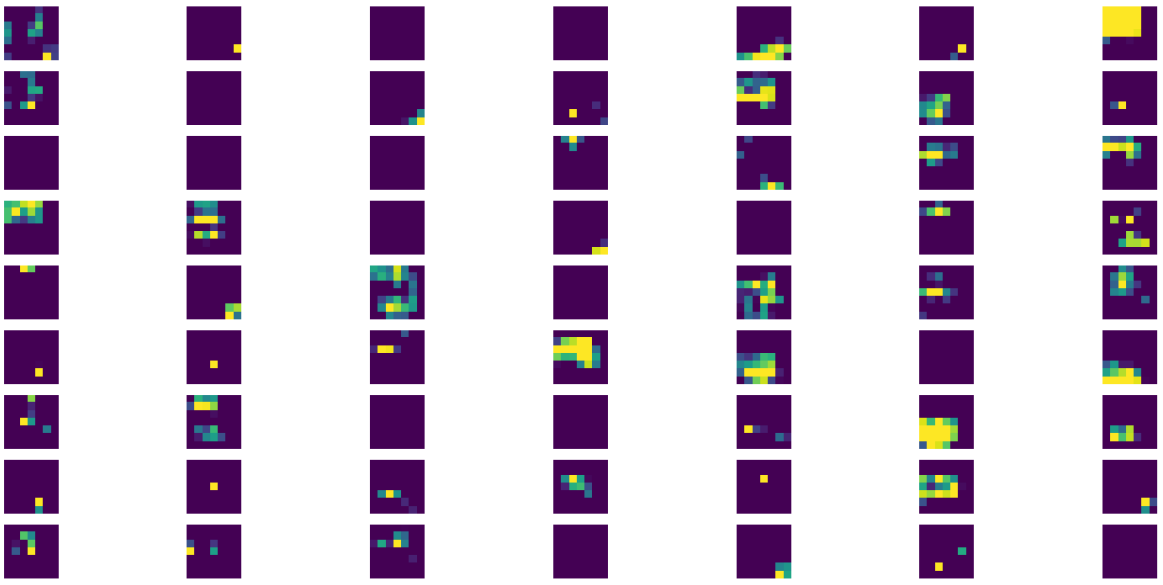
The next command is the layer command. This takes the input of the lowest layer that wants to be checked such as here 2 which is the lowest. Then it takes the highest up to in this case 85. This example uses 2 to 2 so will only show layer 2. It then needs the image to process before asking if the user wants to see the filtering. It then prints the current layer it is showing.

It also then gives a pop out menu of all the feature images. This is layer 2 with a no entry sign. Once closed the next layer will show.

If the wrong layer amounts are given, then is uses the default largest and smallest layer possible. In this case 1 changes to 2 and 100 changed to 85. Some may find it odd that it only goes to layer 85 however this is due to the last layers not having the same kind of output as these feature maps.

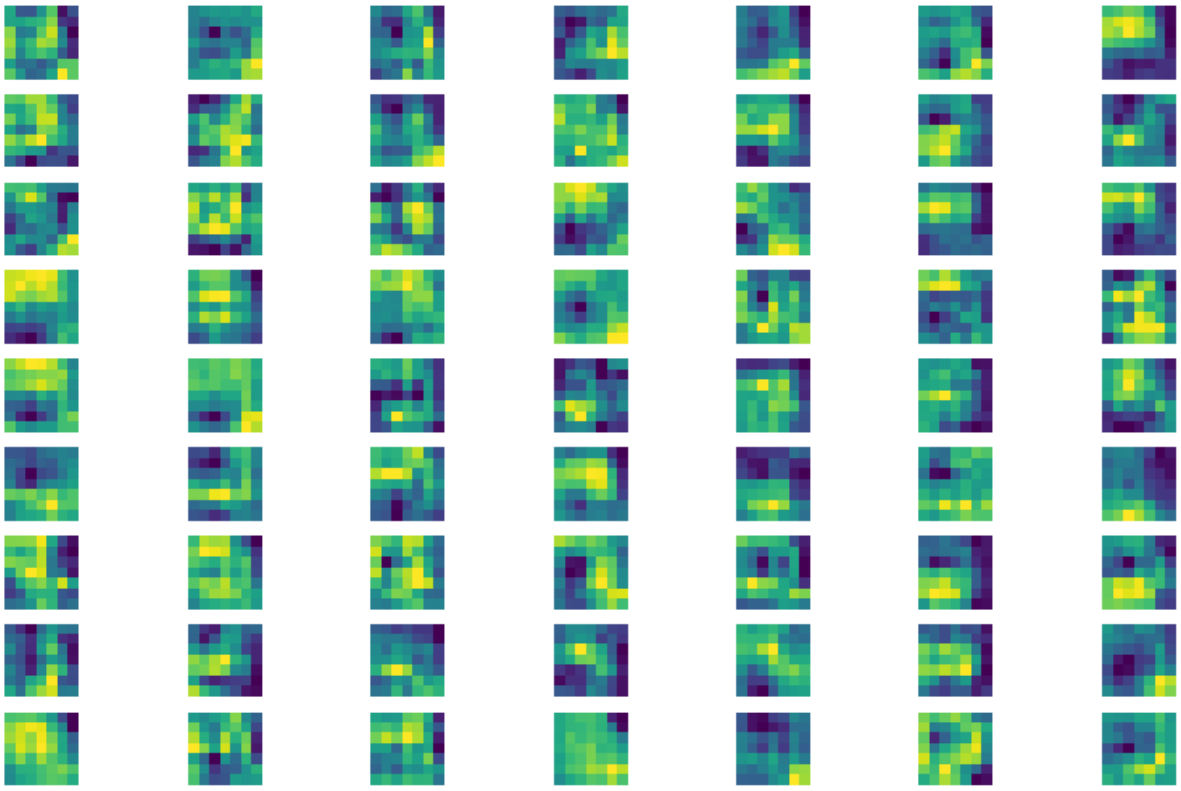
```
total layers: 85
input first layer: 1
too low, changed to 2
to layer: 100
too high changed to 85
notice: layers done backwards
please input full address of image: C:\Users\gamin\OneDrive\Desktop\final project\1116055.jpg
show image? True/False
layer - 85
2022-04-12 14:48:19.404603: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] No
amount of images over 99. recommend a sample instead. press Y for a smaller sample.y
smaller size chosen
image of layer: 85
```

This output also tells the user that it's done backwards so would start with the largest layer as seen with it starting with layer 85. As layer 85 has 1024 images which can be hard to read a option of only 99, a sample size, is given. This image of 99 can be seen here.



```
layer - 84
amount of images over 99. recommend a sample instead. press Y for a smaller sample.y
smaller size chosen
image of layer: 84
```

Once the layer 85 is closed it shows the out for layer 84. As this also has very large amount of image a 99 size sample can also be taken.



As seen from this image from layer 84.

```
amount of images over 99. recommend a sample instead. press Y for a smaller sample.n
layering of plot will be done. warning lag may happen due to lots of images.
```

If no is selected then it will showing all the images but this can be slow as each image is an movable chart and as seen below makes It very hard to read.

This is all images from layer 82. As can be seen not very easy to read.

```
please select 'guess' to input an image and guess which sign it is, 'layer' to pick a image and see how each layer breaks it down, 'summary' to see total model summary or 'Exit' to end program: exit
error, option not selected. try again.
```

On the user inface if something is not a command it askes the user to rewrite it.

```
please select 'guess' to input an image and guess which sign it is, 'layer' to pick a image and see how each layer breaks it down, 'summary' to see total model summary or 'Exit' to end program: exit

Process finished with exit code 0
```

And exit command works the same as before.

This is all the user interface that can be used and their outputs. These are very important as while the code can provide a practical understanding of what is happening the actual use of the models will provide far more useful data points for the project overall. These points can provide far greater insight into the problem this project wishes to solve.

### 4.9-chapter summary

In this conclusion, all the code used for this project, the outputs, future improvements, and reasoning as to why that path was chosen. All these points need to be explained to make the findings of this report repeatable and show how this project could change for an even better result in future works using current or different paths, which can only be done if the first method, this artefact, is understood. This artefact, as seen, consists of three main models working together but which focus on different points of the task. These models can be seen broken down as:

1. Filtering model:
   1.1. Filter an image to remove noise.
   1.2. Send whole files though filtering to speed up users ease and remove human error.
2. Train model:
   2.1. Make ttf files
   2.2. Randomly sort files between ttf to reduce human error.
   2.3. Rprocess to do random sort and train the model.
   2.4. Process to just train the model.
3. Use model (main):
   3.1. Use trained network to guess the name of a sign.
   3.2. Show the feature maps of each layer.
   3.3. Show model summary.

While all these features achieved the needed goals, some of the researched and requested tools were not usable such as DeepDream. This tool caused problems due to it not working with the MobileNet model; however, it was replaced using feature maps. This has given a far more reproducible result and greater levels of data which is of greater use to this report.

Many improvements could be made to the system to improve the results, which were not done due to time limits; however, some of the main improvements are discussed here. More significant testing of the model variables should be conducted to gain even greater accuracy with the neural network. This is for an even more practical real-world model. To change the model to one that shows feature maps without needing an image to be processed would give even more evidence to the results of this report and, as such, would be helpful for future research in this area. Finally, to make the user interface more user friendly with more options to change settings and less strict inputs, which set the setting to default if wrong. This is to save users time and provide a more real-world application.

Overall, the primary goals for this artefact have been achieved, which is likely to provide clear evidence for the conclusion, which others can repeat in this chapter. As such, this chapter and artefact have been successful in their task and will be used within these reports to understand the question.

# 5. Analysis Results

## 5.1 chapter introduction

To answer this report's question of 'how do computers understand human language such as traffic signs', this report has produced an artefact as seen in chapter 4; however, the result from this model requires further insight and explanations as to their meaning.

At the current moment, based on current research seen in chapter 2, the hypnosis is that the model can identify images by recognising common patterns of pixels between different parts of the data types. This hypnosis is due to research and the basic overview of the output seen in 5.2.

## 5.2 system outputs

The artefact produced many different results which can be viewed; however, first, the artefact itself requires an overview due to its accuracy. This overview is an essential step due to the predicted accuracy being near 97%; however, when the test data was used, only 92.5%.

| | | predicted | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| actual | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 1 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 2 | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 |
| | 3 | 1 | 0 | 0 | 5 | 0 | 0 | 0 | 0 |
| | 4 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 |
| | 5 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 |
| | 6 | 0 | 1 | 0 | 0 | 0 | 0 | 5 | 0 |
| | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |

This chart is showing the predicted test data by the neural network.

This result is not an issue as both scores are above the wanted level; however, the 5% difference would suggest that the model might be overtrained, which could produce biased results. In order to check this, new images from a different source than the dataset were used to check the model.

| | | predicted | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| actual | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 2 | 1 | 0 | 5 | 0 | 0 | 0 | 0 | 0 |
| | 3 | 0 | 0 | 0 | 5 | 0 | 0 | 0 | 0 |
| | 4 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 |
| | 5 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 |
| | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 |
| | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |

New chart of predictions using Tabernik, et al, (2019) images as a data input.

As seen from the new model, 92% accuracy was gained from this method. This result would suggest that it is not the wrong model but the test data. This result could be due to the test dataset coming from the TSRD, which used low-quality images, which would cause the network to get it wrong.

This idea comes from the second test data from Tabernik et al. (2019), being so high while the images were hand-picked like the train and valid data, which has the unreadable human signs removed, used to train the model. It is also due to the data in the first chart showing that the most mistakes are in sign datatype 0 and sign datatype 1. This insight is important as these signs are the speed signs which have a lot in common with other signs that the network believes it is. An example can be seen below.

A low-quality image from the test dataset under the dataset name one is 30mph signs on the right compared to a no entry sign image on the left.

As can be seen, both images have a circler shape and image inside. As the image is filtered, colour has no impact on the network's understanding of the image. Therefore, a low-quality image could confuse the system due to the similarities.

While this may prove that this data is safe to use for this report's example, it also suggests that it could be poorly equipped for real-world use. All images inside the dataset are taken during good conditions, so while a good quality camera could remove the risk of low-quality images, other issues such as weather, mud, or poor quality of signs due to rust/age. For this report, though, it does not matter, however, and as such, this project can use this evidence to work towards its question.

This result can also be used to push toward the hypothesis that when an image loses its fine details due to being low quality, the neural network can still identify the more significant features, such as the pattern of the shape. This is seen with low-quality images in the first dataset being correlated by the network to other similar shaped signs. The neural network uses the average pooled network features in the second last layer to combine all this data so it can combine the larger shapes such as the circle and 'empty' inside pattern to no entry. This testing could help prove the hypnosis that patterns affect the network.

On the other hand, it could just be that the neural network must answer. As its dense layer uses a binary output and has no 'unknown' output, it could be that the network is just giving the highest of low odds guess. For example, if it was not a binary out but percentage biased, it could be shallow such as 20% for data type 6, which would show it does not believe it is that type. It does not do this as it is binary, which means that it takes the most likely reply and gives that as an answer, with that being one and everything else being 0. As the network must give one of the data types, it may just be giving a bad guess which could show that it has nothing to do with the hypnosis or even disprove it by showing that clear patterns are not used by the system.

This concept, however, is hard to prove with just this data due to it having low rates, 8% from the first dataset and 2% from the second, so testing using feature maps and low-quality images should be done to check these factors.

In conclusion, this has shown that for this project, this model will be very suitable and that even from the known errors found in the system, there is still information that can be gained that can alter these reports' final findings.

The other outputs from the system include the correct outputs of 30 mph, 50 mph, 60 mph, 70 mph, junction, Warning (empty), no entry and Warning (!) signs which should also be tested. This is due to some signs like the first five being alike while others have different patterns, such as warnings being upside-down triangles. This can be checked for how the feature maps different process patterns.

If it is found that these feature maps have different reactions to the different patterns and shapes, it would likely point towards the hypnosis that computers use patterns to recognise these images. This could be seen with different feature maps focusing on different parts of the images while some being unresponsive to these images. However, if no change can be found, it would suggest the system is not using the patterns to guess, with a high degree of accuracy, what these images are and relies on a different method, therefore disproving the current hypnosis.

This, however, does pose two problems. It may be that the feature maps themselves show these changes in a way a human could not understand, making it hard to prove if this proves or disproves the current hypnosis. The second issue is that features such as the weighting done in training the node may affect the outcome in a way the feature maps do not display. This could mean that even if a feature map shows a clear sign of the network recognising patterns, it will be hard to prove that a different fracture is not also affecting the outcome. Therefore, multiple tests should be done and condensed into a chart to see if this happens multiple times. This will show a correlation between correct guesses and feature maps recognising the patterns or disproving these feature maps producing a role in the high accuracy of the work.

To conclude on correctly guessed data types, they will be tested for differences and simulates to understand how those differences are or are not used by the network.

### 5.3 analyzing feature maps

The primary use of the artefact for this report, which has been reliable, is to produce feature maps of the neural network. These feature maps will be reviewed to find the needed details as spoken about in 5.2 while expanding on what these could mean for the reports question and computer vision.

It can be seen from the different layers that the images are lower quality and more minor because of the convolution filter, which can produce points of interest. The convolution filter takes an image and reduces it based on a matrix value that can change the neural network views the images. This can be seen below from layer 2 using a no entry sign as an input.

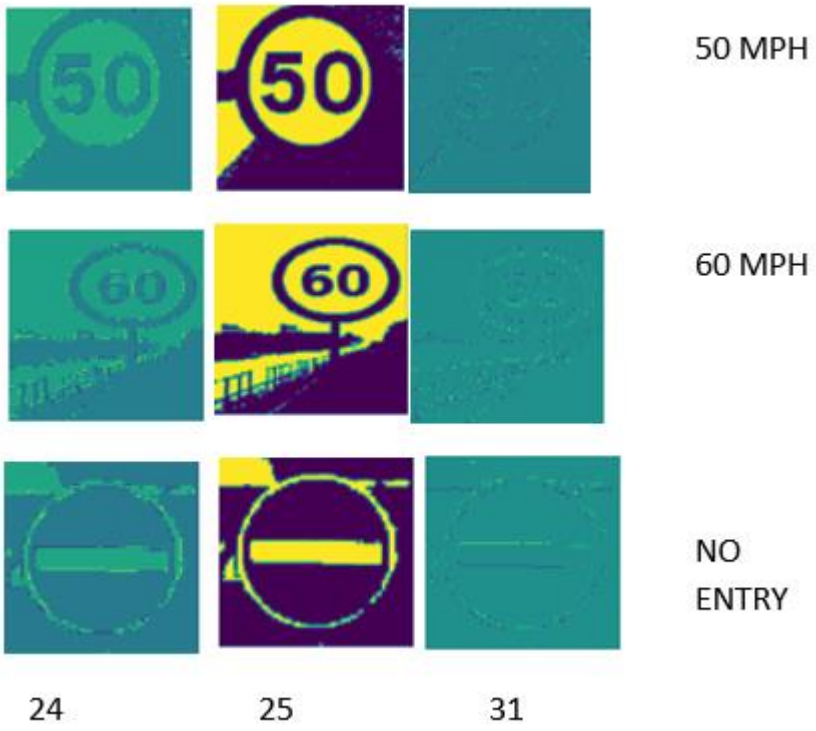
After being passed in, this first image has no highlights at all due to convolution.

This second image shows the sign with the inside highlighted in black as the convolution has selected this.

This image can be seen to have only the very edge highlighted by the convolution.

These images all contain different highlights of the image passed to them. This shows that the network likely benefits from highlighting different vital features to find different patterns within them better. This is seen multiple times on different layers where these and other convolutional filtered images will have sections highlighted, such as layers 8, 15, 22 and other layers using conv2D and DepthwiseConv2D methods seen in the summary. This persistent pattern of filtering the images with convolutions shows that the neural network requires these highlighted details to function effectively. The weights for the neural networks are trained and therefore weighted on these filtered images meaning the network uses highlighted features to conduct images. This works with the hypnosis that the neural network uses patterns to identify images as these highlights quickly show familiar patterns.

These common patterns can be seen with different signs below as the filter is the same for all images passed to it, then the different highlights can be seen in effect on different signs as done below.



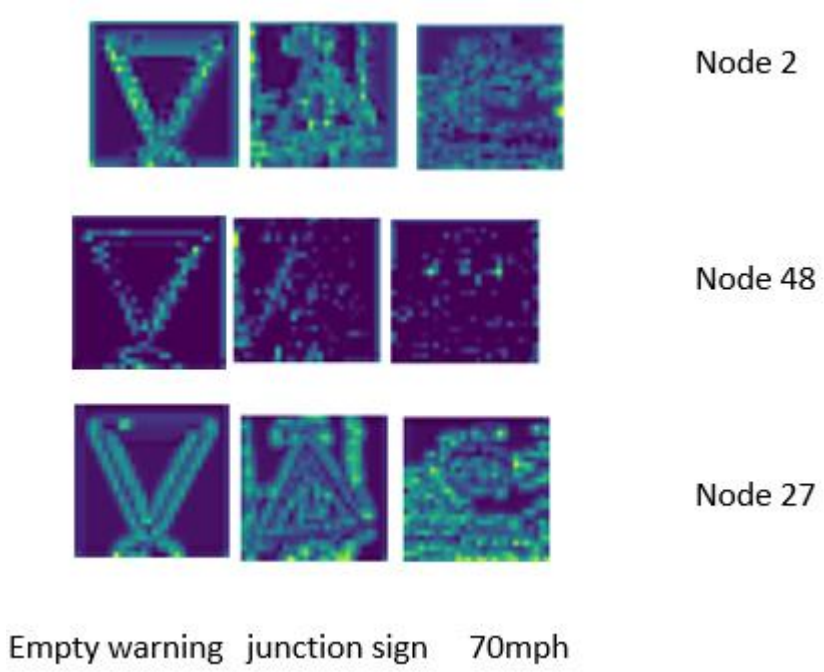

These images are taken from layer 2 all with the same filtering from nodes 24, 25, 31. As can be seen the images from node 24 have a general highlight of the whole image. Node 25 shows a highlighting of solid black objects while node 31 shows just the edges of shapes. This filtering can show how the neural network breaks down the image into key patters such as solid numbers like ‘50’ and ‘60’ while not showing the solid bar in the no entry which are key patterns the network would need to find the different between those three signs. It can also be seen in the node 31 with the outlines that shapes such as circles stand out far more then background shapes like in 60 mph sign. These are all patterns the neural network could pick up on and likely the main reason for doing so.

However, this could also mean that the layers simply change the data to reduces the memory of the computer which is needed to run the neural network. If network works off a weight that only uses the highlighted edges of the image such as in node 31 when having extra pixel complexity, such as in node 24 which is less smooth, could increase the work needed from the network. This can be seen with higher quality images having a far greater pixel value which increase complexity. Using O notation such as linear or greater with a very large pixel size could increase the time and power a computer needed to do these tasks.

This is unlikely due to the issue that even if the details are remove the pixel count is still the same suggesting that it won’t save on space or processing. Also seen within the image is colour which was not introduced by the image as that was filtered from a black and white image. If they want to reduce complexity was programmed into the filtering of the program, then added colour compared to a binary black or white is in effect. This is due to colours taking up a greater space than just black and white. The final point as to why this is likely not a way to reduce processing is due to the program producing more images. As seen in later layers with over 1024 images compared to the one inputted that pixels size likely doesn’t affect the size as even if the later images are smaller they still produce far from different images. All these reasons suggest that the second reason, that this filtering is done to reduce complexity, is likely incorrect or a by-product of the true purpose.

Therefor for this key component of the neural network it's expected this supports the hypothesis because convolution filtering is done to find patterns due to its highlighting of key parts of signs in the feature maps.

the feature map can show the ways that the neural network understands which image belongs to which data type by seeing how different pixels are chosen between layers. As each feature map is used on each image some will be weighted to a particular data type, image of sign. This results in the images being very clear while in other being very unfocused or blank. This can be seen in layer 35.

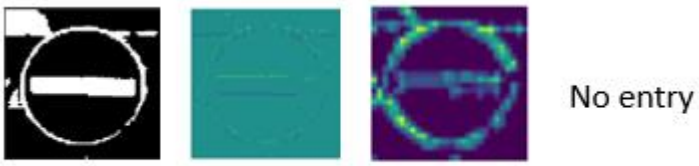



These are image from layer 35 from the same nodes listed to their sides with the data type listed below them. As can be seen these images can differ greatly. On node 2 the images contain most details however the 70mph sign is removed from its image with only a background compared to a easy to see outside of the two other signs and a clear inside of the junction sign.

Node 48 is very hard to see for all images due details being removed by the neural network however the outside of the empty warning sign can be seen along with one edge of the junction sign. The 70mph is almost complete gone.

Lastly the node 27 has all signs outlined including the 70-mph sign. the junction sign is the clearest with all edges and inside seen however the 70 mph is lacking its inside shape of the '70' while the warning sign is missing it's top line with only one cube highlighted.

This could lead to the same conclusions as explained before where convolution has been applied to the images. This would keep in line with the same hypnosis not expand on it in any meaningful way. This would explain why they do not look like the original images that were given to the input and with such things as the highlights edges as seen before.

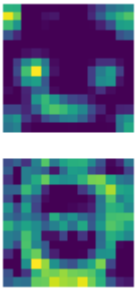



This is an example showing the same image at different layers. At layer 0 it's filtered by not used by the neural network yet, layer 2 is the first convolution image and layer 35 is batch normalized. As can be seen the white edges are highlighted in the 2 and then 35 layer which supports the idea that this is just another way for the network to highlight images with small changes.

However, this is unlikely due to some feature maps having different key points highlighted on layer 35. This is most noticeable on node 45 where most of the 70-mph sign is lost while the general pattering of the outside of the empty warning is still visible. This can also be seen where the warning sign loses it's top while those pixels are highlighted on the junction sign on node 48. Node 2 shows a lack of circus even though most of the background is highlighted and the other two sign show detail. All these different details on the same node were not seen on layer 2. On layer 2 each node gave the same highlight however these nodes on layer 35 seem to select only parts of the system to highlight compared a general overall filtering.

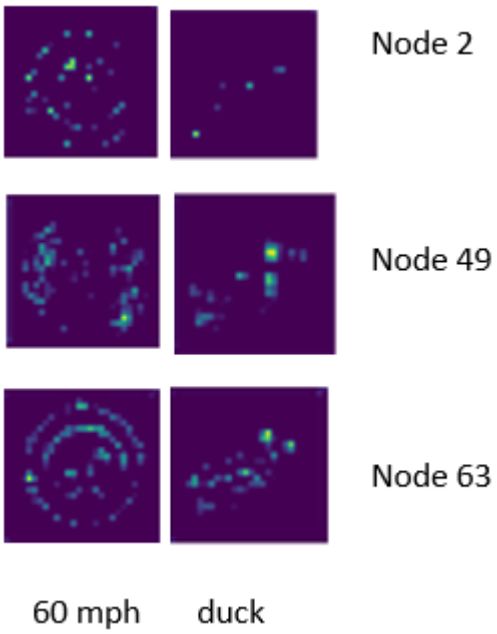
Another more likely option is that this feature map is high-lighting different parts the image based on patterns. While this report know that convolution filtering has been applied to these images as this is a later layer than layer 2 and others it would seem something else is causing a more focused select of the images. It can be seen that the feature maps being applied to the image is done to highlight parts that fit a pattern such as node 48 selecting sides of triangles and node 27 not accepting straight lines on the warning sign. this supports the hypnosis that the neural network uses patterns to identify images. This selective pattern selection was not seen in layer 2 where a general filter was applied to it. This selective mapping likely comes from weight changes to the nodes where different pixel patterns are highlighted if they match or excluded if not, it then likely goes on to use this to guess the image based on these patterns. During training it is likely these map matrixes were changed to suit the new datatypes common patterns it was being trained on which is why it works on signs.

This repeats on layer such as 69 and other which include the term 'BatchNormaliz' in the name however due to shrinking of image size it can become hard to find details using a human understanding. It can still be seen however such as In layer 69.



As can be seen from these two node feature maps from layer 69 of a no entry sign some maps still exclude such as the top sign missing 60% of the round shape, background and inside bar while the lower node has a full circle with bar.

This effect of only some patterns being seen can be seen even more clearly with a wrong sign. using a duck from Jeremy B (2018) as the input with was guessed to be 60 mph compared to a 60 mph sign this report can see this selective pattern.




While all images seen in this example lack details the 60mph sign has a lot more of itself then that of the duck in the same node. This is likely because the duck lacks many of the common patterns seen in the signs. This is most clear in node 63 where the ‘60’ can be seen while only a small amount of the ducks back can be seen. The reason the duck turns up at all is likely because of background noise in training and doe to the different locations sign could be in making it so the neural network needs to have the feature maps check for all kind of pixel placement.

In conclusion it is likely that these signs have been under the same seen before effect of convolution filtering however it also shows the differences between node highlights due to the nodes corresponding to different patterns between feature maps and data type images.

A test of an incorrect sign type was recommended in the last chapter to find anything else out about how the network decided on the final label. An image of a duck which is not sign was given to the network and got the result of a 60mph sign. to check if this held any details on how the network may use patterns to recognise this it was compared to a correctly guess 60mph sign.


This shows nodes that do and don't look alike between a duck which the neural network things is a 60-mph sign and a 60-mph sign. the images to the left and right are from the same node while horizontally from the same image of duck or sign.

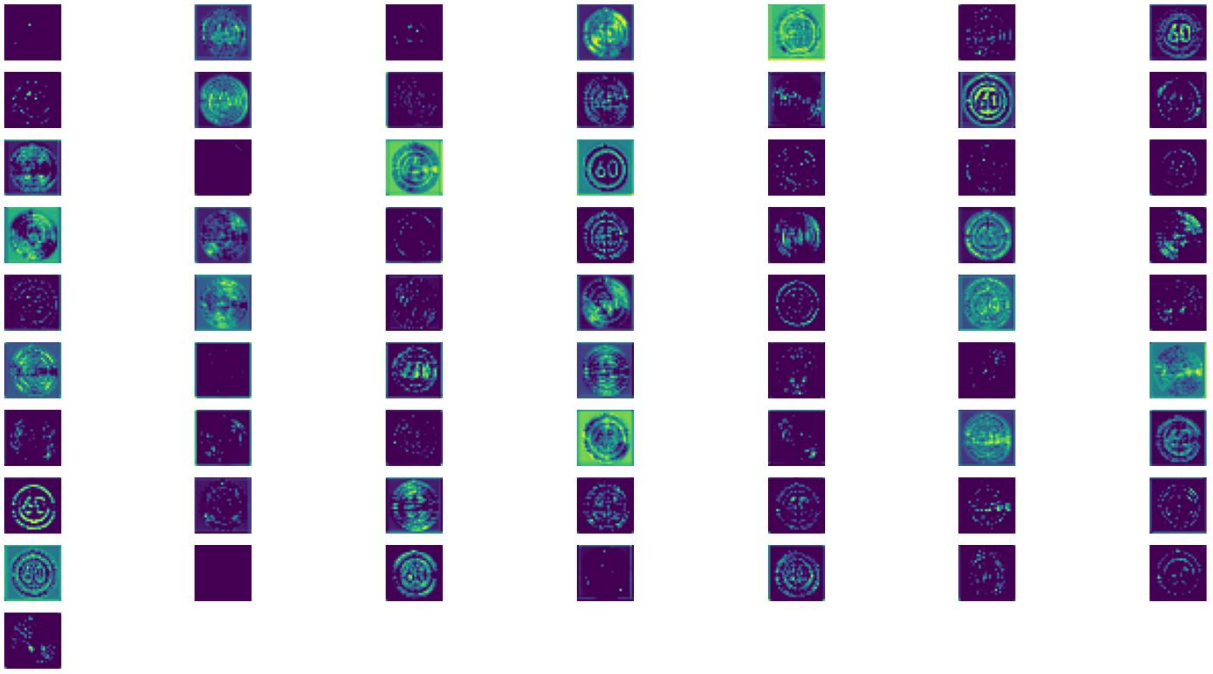
This may suggest that as it is still guessed 60-mph even when the images with lots of detail do not correlate (not alike) it's still getting a high pattern detection because of the lesser detailed feature maps (alike). This could mean that images with less details carry the same weight as those with lots of detail, with many with more feature maps being of the less detailed, or that feature maps of lesser detail carry more weight. This is for the network responded with 60-mph even when the images do not look alike.

However, this hard to know for sure and require more testing in the future for it could also be that, as previously stated, a binary output was needed, and 60 mph was the largest of small values. It could be these alike feature maps are what gave it higher score than other data types but it's hard to check with such little data from these feature maps.

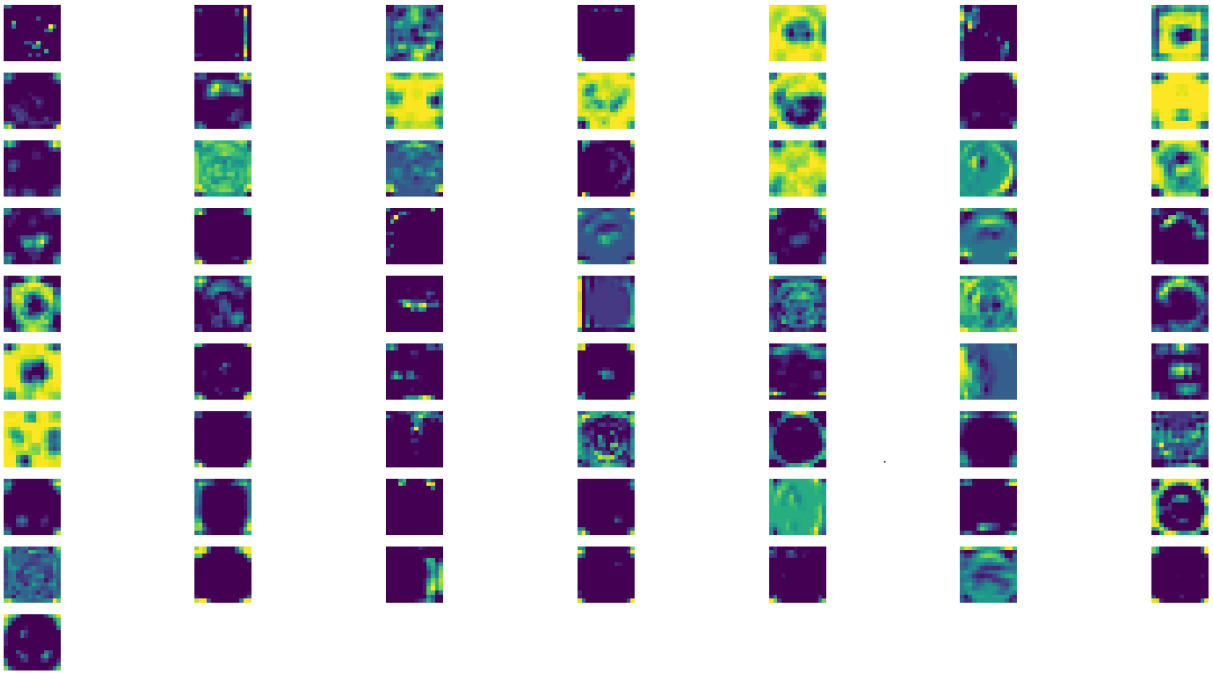
This doesn't go against the hypnosis but does add to it suggesting that all nodes and feature maps carry weight including that of low details which may affect the result.

Batch normalized images can be seen increasing in noise as it increases in layers. This can be seen as it changes later.

Layer 35 with use of 60-mph sign



Layer 69 with use of 60-mph sign



This can show how many layers of filtering can greatly alter the images. This could suggest that details that humans fail to pick up are used by the neural network to understand which sign this is. These details are likely patterns as even 69 layers deep and having the image filtered many times some key features such as circles can be seen in both. This may also explain why issues with the accuracy are often with thinking circle signs are different circle-based signs rather than a warning sign which has the key pattern of an upside-down triangle. This supports the hypnosis that patterns are used however not in a way which was not explained before.

### 5.4 chapter summary

This chapter has gone over the details of the system outputs, accuracy of those results and a deeper analysis as to how this impact the overall reports question. It has shown that this data can be usable for this report, and has been, however the system may not be used in practical road environment within a car however could be used as a training tool. this basic overlook also gave way to the idea that patterns are recognized by the network which is why signs that look alike are often mistaken by the network as a same shaped sign.

This gave the hypnosis the model can identify images by recognising common patterns of pixels between different parts of the data types.

It was then debated in 5.3 using feature map outcomes which showed that the hypnosis based on that evidence is likely correct. This however also leads to more research being suggested to find how the weights are used to measure these feature maps however for this report this information is not needed as while they play a part of this evidence from other reports like Cilimkovic, M., 2015 and Persson, S. (mars 2018) show how weights effect these kinds of models.

# 6 report conclusions

## 6.1 chapter introduction.

This chapter will go over the conclusion of this report and come to a final evaluation of how this report was conducted. This will fist focus on, given each chapter, a conclusion to this reports question of how computers will understand visual language.

Then what future reports can do to improve on this.

## 6.2 conclusion of report

This report has focused and done research on the current methods in which computer vision can recognise images and visual langue's such as traffic signs. This has been built on with materiel such as feature maps from a neural network, a common way to identify items within image, artefact. These feature maps were looked at in detail and debated if they fit the hypnosis, which this report has found they have.

This hypnosis was based on research as seen in chapter 2 and a basic over look at how on a high accuracy neural network incorrect guesses were labelled. It was then further examined in chapter 5 using feature maps to show how patterns were affected by the machine and what this meant.

this evidence from the artefact helped conclude that algorithms can identify patterns of languages and use these patterns to predict which data type they belong too. This however is not the complete method of how algorithms like neural network work as some also require weights as can be seen from processing a random image produced a output from the neural network. However, using the evidence and research this hypothesis of pattern prediction can be understood as a way that algorithms understand langue's.

so how computers understand visual language is by finding patterns within images. This is done using feature mapping which highlights different parts of images for the algorithm to pick up on. It can do a general highlight which shows different parts of image such as edges but applying the same filter to all data type. It can also highlight different parts in a special way depending on the data type for example the sides of an upside-down triangle and not circles. This recognising of shapes is trained into the algorithm which can be done manually or as seen in this example by an artificial intelligence however needs to find general pattern first as all images can have differences such as quality and background noise. To find this pattern is as said before by this filtering.

To summarise, an algorithm uses filters to find common patterns. There are different filters. This filtered image is then compared to a seen before pattern or weight based on pixel location which lets the

algorithm guess to a high degree what data type it is. This is the explanation to the question of this report.

### 6.3 future reports

This report was designed to answer the now explained question of how algorithms understand images however it has also brought up work for future reports. This includes ways to improve this report and other questions which could be investigated.

Some ways to improves this are spoken about in chapter 4 which states ways to improve the artefact. This includes the greater testing of the model variables, change model to one that shows feature maps and to improve the user interface. All of these methods are spoken about more in chapter 4 but all are used to improve the results of the current report but couldn't be done due to the limit of time.

Another way to improve it would be to make a heat map of each feature map compared to different datatypes as this could give a way to view what the feature maps are checking for. This was not done in this report as it would take a large time sink.

Another way would be further research into how weights or patterns are recognised by machines. This could be done with researching how neural networks learn new patterns and groupings or how manual algorithms can be programmed.

It could also be done by training a neural network but taking measurements of how it groups pixels or shapes in an image change every epoch as the network improves its accuracy. This would likely show the network grouping pixels into common shapes of the test data however this is just an untested hypothesis.

These methods and research would likely give a greater understanding of the topic while also proving more evidence for or against this report's method and conclusion.

This report was unsuccessful in using the deepdream tool however feature maps have shown to be a far more detailed and controllable tool in to use. For future projects the use of feature maps is recommend over deepdream.

### 6.4 limitations

This report has been limited in some ways influence the overall work. The main limits are the time and pre-existing skills.

The limits of time have caused some tools such as deepdream to have not been used. While this report values the use of feature maps over that of deepdream using both would have only added to the report. The reason this was not done was because MobileNet was not able to run TensorFlow however other models of neural networks were. If time was given a new model could have been made which worked with deepdream. This adds the benefits of having deepdream to back up or disprove current work while also having an extra model to compare the feature maps too. This limit of time did not let this, and other improvements written about in chapters 6.3 and 4.

The limit of pre-existing skill causes time from the project and limited the tools that could be used. As to use the tools such as OpenCV and TensorFlow in the artefact had to be learnt while the artefact was being produced it cost time from the project which could have been used to produce other methods as talked about. However, it also meant that tools such as a heat map or current artefact tools are not as useful as they could have been. This increases the minimum running requirement of the project while also taking time from the users as the artefact can take longer than is needed to run a process.

### 6.5 Project aims

The project had overall aimed to be covered which will be written about whether they were successful and why.

Overall project goals:

- Produce reproduceable work including results and artifacts. To achieve this a selection of artifacts image and guesses results should be produced with code and notes of the working artefact in the final report. This has been achieved with the use of chapter 4 which details the needed parts to make it reproduceable.
- To produce a goal for artifacts and back it up with evidence. This can be seen with the 80%+ pass rate of guessing by the artefact which will be tested separately from training and results given in final report. This 80%+ aim was overachieved with a 90%+ pass rate and with repeatable outputs proved to be effective.
- To identify artifacts understanding of inputted image. As seen with use of checking feature Map and checking for patterns with use of high pass rate artifacts. This is achieved and seen within chapter 4 and 5 with the details of the output showing how the artefact understood images input.
- To research different methods and use them to produce an effective artefact. This can be seen done within the literary review by identifying the use of MobileNet and reproducing an artefact with its use however the effectiveness will be proven by producing a reliable artefact. This was achieved with research and even further with extra researched in place of deep dream.
- To produce correct references and state their importance. This can be seen throughout the work with quoting and Harvard references. This was done till the end and was achieved by doing so.

Artefact goals:

- Produce a filter to reduce image noise using OpenCV and changed to same size. This was done and seen in chapter 4 in the filter model and later in the main model.
- Produce a dataset from Mykola (2018), filtered and refined to 1) reduce total amount of different signs, 2) remove copied/repeated images, 3) remove low quality images. This was done and used to train the model.
- Sort dataset into different random sets. This was done with the training model seen in chapter 4.
- Use TensorFlow to produce MobileNet network then save model to file with trained weights. This is done and seen in chapter 4.
- produce image of feature Map layers. Done in place of deep dream and done. Seen in chapter 4.
- a manual image input to test new untrained images on by hand. This is done and seen in chapter 4.
- produce a basic interface. This was done for main and training, seen in chapter 4.

Report goals:

- To produce work that is understandable by others. This can be achieved by checking work to others and having others overview it such as my supervisor. This was done for first chapters and spell checked to insure it was achieved.
- produce a guide to pass on how to achieve a similar Artefact. This is done in chapter 4.
- reflex the question of the paper in all work such as with links to it within the images from the artefact. This is done in chapter 5 and 6 and was achieved.
- state if the question is answered within the work meaning whether it was proven or not with reason why. Done in chapter 5 and 6 and was achieved.
- provide real world use. This is done in chapter 4 and 5 with real world uses given.
- question future research based on results. This is done in chapter 4, 5 and 6 and was achieved.

### 6.6 chapter final

This chapter has covered how the report could be improved, such as more model testing and why it was not due to the limitation imposed on the work. It also concluded the report's question using the research and artefact, which produced a working hypothesis. Finally, the aims were covered in which most were achieved. This ends this report.

Appendix A

**Artefact libraries and versions**

All used libraries and versions of libraries, codes and anaconda listed below:

- Code langue: Python 3.9
- Conda version : 4.10.3
- Conda-build version : 3.21.4

Copy from anaconda environment:

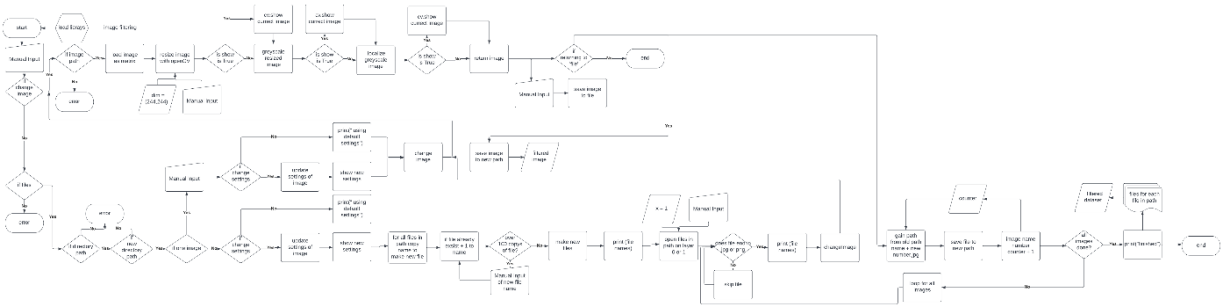
```
# Name                    Version                   Build  Channel
_tflow_select             2.3.0                       mkl
abseil-cpp                20210324.2          hd77b12b_0
absl-py                   0.15.0              pyhd3eb1b0_0
aiohttp                   3.8.1            py39h2bbff1b_0
aiosignal                 1.2.0               pyhd3eb1b0_0
astor                     0.8.1            py39haa95532_0
astunparse                1.6.3                    py_0
async-timeout             4.0.1               pyhd3eb1b0_0
attrs                     21.4.0              pyhd3eb1b0_0
backcall                  0.2.0               pyhd3eb1b0_0
blas                      1.0                         mkl
blinker                   1.4              py39haa95532_0
brotli                    1.0.9               h8ffe710_6    conda-forge
brotli-bin                1.0.9               h8ffe710_6    conda-forge
brotlipy                  0.7.0            py39h2bbff1b_1003
ca-certificates           2021.10.26          haa95532_4
cachetools                4.2.2               pyhd3eb1b0_0
certifi                   2021.10.8        py39haa95532_2
cffi                      1.15.0           py39h2bbff1b_0
charset-normalizer        2.0.4               pyhd3eb1b0_0
click                     8.0.3               pyhd3eb1b0_0
colorama                  0.4.4               pyhd3eb1b0_0
cryptography              3.4.8            py39h71e12ea_0
cycler                    0.11.0              pyhd8ed1ab_0    conda-forge
dataclasses               0.8                 pyh6d0b6a4_7
decorator                 5.1.1               pyhd3eb1b0_0
flatbuffers               2.0.0               h6c2663c_0
fonttools                 4.29.1           py39hb82d6ee_0    conda-forge
freeglut                  3.2.1               h0e60522_2    conda-forge
freetype                  2.10.4              h546665d_1    conda-forge
frozenlist                1.2.0            py39h2bbff1b_0
gast                      0.4.0               pyhd3eb1b0_0
giflib                    5.2.1               h62dcd97_0
google-auth               1.33.0              pyhd3eb1b0_0
google-auth-oauthlib      0.4.1                    py_2
google-pasta              0.2.0               pyhd3eb1b0_0
grpcio                    1.42.0           py39hc60d5dd_0
h5py                      3.6.0            py39h3de5c98_0
hdf5                      1.10.6              h7ebc959_0
icc_rt                    2019.0.0            h0cc432a_1
icu                       68.1                h6c2663c_0
idna                      3.3                 pyhd3eb1b0_0
importlib-metadata        4.8.2            py39haa95532_0
```

```
intel-openmp              2021.4.0         haa95532_3556
ipython                   7.31.1           py39haa95532_0
jasper                    2.0.33           h77af90b_0    conda-forge
jedi                      0.18.1           py39haa95532_1
joblib                    1.1.0            pyhd3eb1b0_0
jpeg                      9d               h2bbff1b_0
keras-preprocessing       1.1.2            pyhd3eb1b0_0
kiwisolver                1.3.2            py39h2e07f2f_1    conda-forge
libblas                   3.9.0            1_h8933c1f_netlib    conda-forge
libbrotlicommon           1.0.9            h8ffe710_6    conda-forge
libbrotlidec              1.0.9            h8ffe710_6    conda-forge
libbrotlienc              1.0.9            h8ffe710_6    conda-forge
libcblas                  3.9.0            5_hd5c7e75_netlib    conda-forge
libclang                  11.1.0           default_h5c34c98_1    conda-forge
libcurl                   7.80.0           h86230a5_0
liblapack                 3.9.0            5_hd5c7e75_netlib    conda-forge
liblapacke                3.9.0            5_hd5c7e75_netlib    conda-forge
libopencv                 4.5.1            py39h27d8466_0    conda-forge
libpng                    1.6.37           h2a8f88b_0
libprotobuf               3.14.0           h23ce68f_0
libssh2                   1.9.0            h7a1dbc1_1
libtiff                   4.2.0            h0c97f57_3    conda-forge
libwebp                   1.2.2            h57928b3_0    conda-forge
libwebp-base              1.2.2            h8ffe710_1    conda-forge
lz4-c                     1.9.3            h8ffe710_1    conda-forge
m2w64-gcc-libgfortran     5.3.0                    6    conda-forge
m2w64-gcc-libs            5.3.0                    7    conda-forge
m2w64-gcc-libs-core       5.3.0                    7    conda-forge
m2w64-gmp                 6.1.0                    2    conda-forge
m2w64-libwinpthread-git   5.0.0.4634.697f757       2    conda-forge
markdown                  3.3.4            py39haa95532_0
matplotlib                3.5.1            py39hcbf5309_0    conda-forge
matplotlib-base           3.5.1            py39h581301d_0    conda-forge
matplotlib-inline         0.1.2            pyhd3eb1b0_2
mkl                       2021.4.0         haa95532_640
mkl-service               2.4.0            py39h2bbff1b_0
mkl_fft                   1.3.1            py39h277e83a_0
mkl_random                1.2.2            py39hf11a4ad_0
msys2-conda-epoch         20160418                 1    conda-forge
multidict                 5.1.0            py39h2bbff1b_2
munkres                   1.1.4            pyh9f0ad1d_0    conda-forge
numpy                     1.21.2           py39hfca59bb_0
numpy-base                1.21.2           py39h0829f74_0
oauthlib                  3.1.1            pyhd3eb1b0_0
olefile                   0.46             pyh9f0ad1d_1    conda-forge
opencv                    4.5.1            py39hcbf5309_0    conda-forge
openssl                   1.1.1m           h2bbff1b_0
opt_einsum                3.3.0            pyhd3eb1b0_1
packaging                 21.3             pyhd8ed1ab_0    conda-forge
parso                     0.8.3            pyhd3eb1b0_0
pickleshare               0.7.5            pyhd3eb1b0_1003
pillow                    8.4.0            py39hd45dc43_0
pip                       21.2.4           py39haa95532_0
prompt-toolkit            3.0.20           pyhd3eb1b0_0
protobuf                  3.14.0           py39hd77b12b_1
```

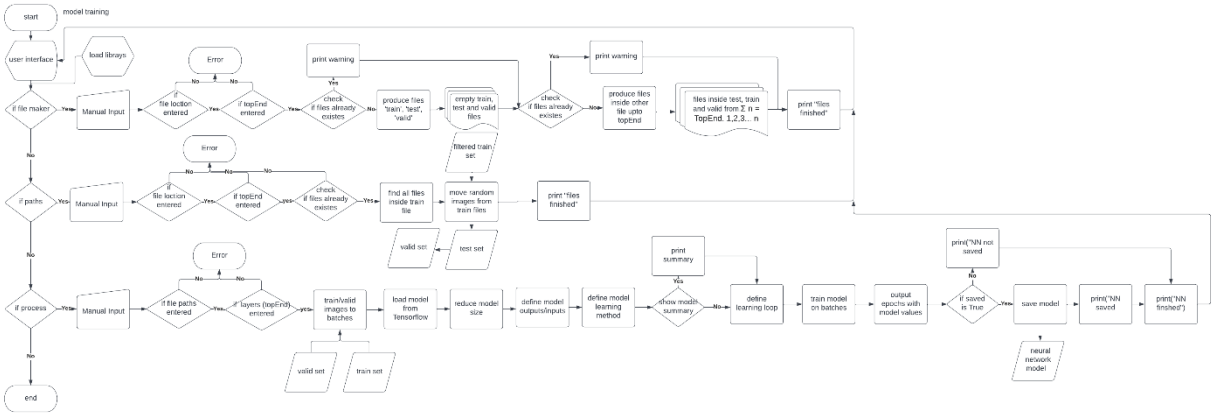
```
py-opencv                 4.5.1           py39h832f523_0    conda-forge
pyasn1                    0.4.8             pyhd3eb1b0_0
pyasn1-modules            0.2.8                    py_0
pycparser                 2.21              pyhd3eb1b0_0
pygments                  2.11.2            pyhd3eb1b0_0
pyjwt                     2.1.0           py39haa95532_0
pyopenssl                 21.0.0            pyhd3eb1b0_1
pyparsing                 3.0.7             pyhd8ed1ab_0    conda-forge
pyqt                      5.12.3          py39hcbf5309_8    conda-forge
pyqt-impl                 5.12.3          py39h415ef7b_8    conda-forge
pyqt5-sip                 4.19.18         py39h415ef7b_8    conda-forge
pyqtchart                 5.12            py39h415ef7b_8    conda-forge
pyqtwebengine             5.12.1          py39h415ef7b_8    conda-forge
pyreadline                2.1             py39haa95532_1
pysocks                   1.7.1           py39haa95532_0
python                    3.9.0              h6244533_2
python-dateutil           2.8.2             pyhd8ed1ab_0    conda-forge
python-flatbuffers        1.12              pyhd3eb1b0_0
python_abi                3.9                  2_cp39    conda-forge
pyyaml                    6.0             py39h2bbff1b_1
qt                        5.12.9             h5909a2a_4    conda-forge
requests                  2.27.1            pyhd3eb1b0_0
requests-oauthlib         1.3.0                    py_0
rsa                       4.7.2             pyhd3eb1b0_1
scikit-learn              1.0.2           py39hf11a4ad_1
scipy                     1.7.3           py39h0a974cb_0
setuptools                58.0.4          py39haa95532_0
six                       1.16.0            pyhd3eb1b0_0
snappy                    1.1.8               h33f27b4_0
sqlite                    3.37.0              h2bbff1b_0
tensorboard               2.6.0                    py_1
tensorboard-data-server   0.6.0           py39haa95532_0
tensorboard-plugin-wit    1.6.0                    py_0
tensorflow                2.6.0           mkl_py39h31650da_0
tensorflow-base           2.6.0           mkl_py39h9201259_0
tensorflow-estimator      2.6.0              pyh7b7c402_0
termcolor                 1.1.0           py39haa95532_1
threadpoolctl             2.2.0              pyh0d69192_0
tk                        8.6.11              h8ffe710_1    conda-forge
tornado                   6.1             py39hb82d6ee_2    conda-forge
traitlets                 5.1.1             pyhd3eb1b0_0
typing-extensions         3.10.0.2             hd3eb1b0_0
typing_extensions         3.10.0.2           pyh06a4308_0
tzdata                    2021e               hda174b7_0
unicodedata2              14.0.0          py39hb82d6ee_0    conda-forge
urllib3                   1.26.7            pyhd3eb1b0_0
vc                        14.2                h21ff451_1
vs2015_runtime            14.27.29016         h5e58377_2
wcwidth                   0.2.5             pyhd3eb1b0_0
werkzeug                  2.0.2             pyhd3eb1b0_0
wheel                     0.35.1            pyhd3eb1b0_0
win_inet_pton             1.1.0           py39haa95532_0
wincertstore              0.2             py39haa95532_2
wrapt                     1.13.3          py39h2bbff1b_2
xz                        5.2.5               h62dcd97_1    conda-forge
```

| | | | |
|---|---|---|---|
| yaml | 0.2.5 | he774522_0 | |
| yarl | 1.6.3 | py39h2bbff1b_0 | |
| zipp | 3.7.0 | pyhd3eb1b0_0 | |
| zlib | 1.2.11 | h8cc25b3_4 | |
| zstd | 1.5.0 | h6255e5f_0 | conda-forge |

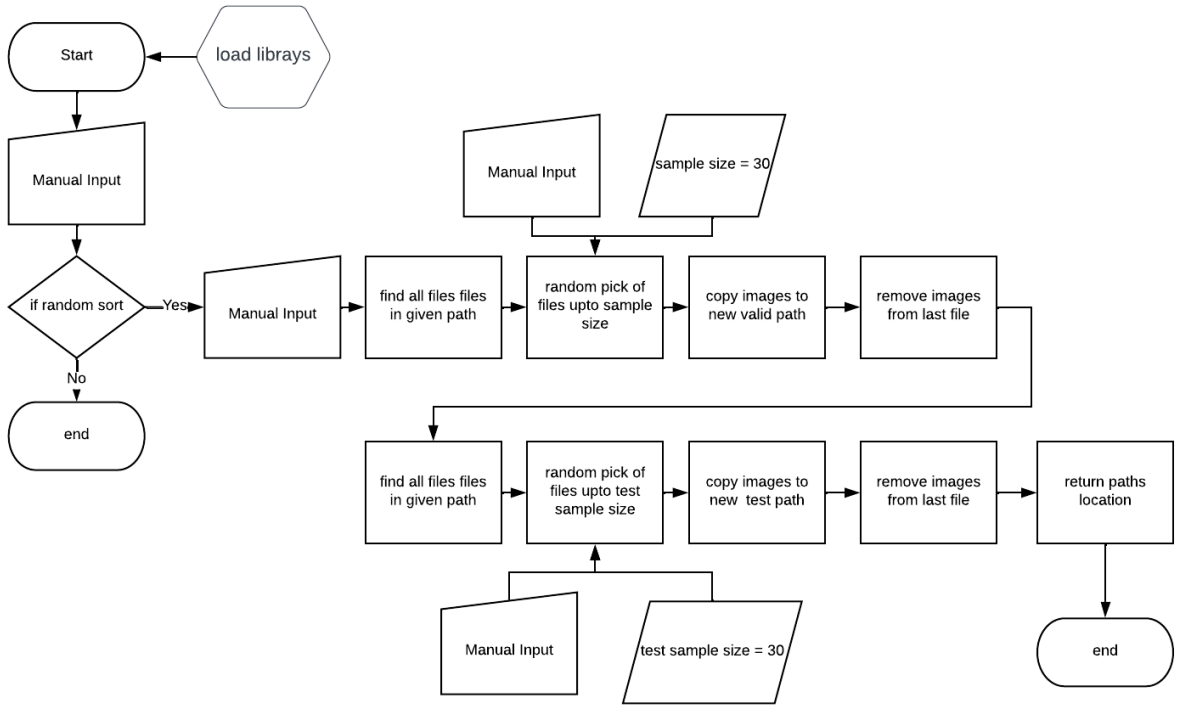
**Flow chart of image filtering**



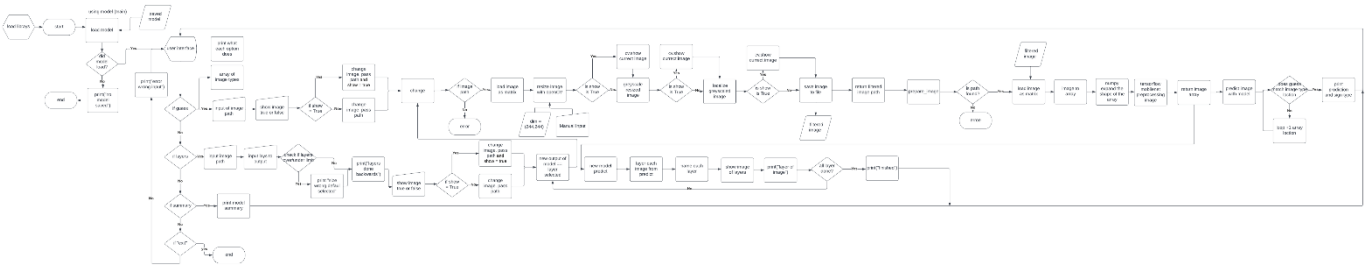
**Flow chart of model training**




**Flow chart of model trainings random sort function**




**Model of main (using model)**



**Summary of neural network layers**

Layer (type) Output Shape Param #

1. input_1 (InputLayer) [(None, 224, 224, 3)] 0
2. conv1 (Conv2D) (None, 112, 112, 32) 864
3. conv1_bn (BatchNormalization (None, 112, 112, 32) 128
4. conv1_relu (ReLU) (None, 112, 112, 32) 0
5. conv_dw_1 (DepthwiseConv2D) (None, 112, 112, 32) 288
6. conv_dw_1_bn (BatchNormaliza (None, 112, 112, 32) 128
7. conv_dw_1_relu (ReLU) (None, 112, 112, 32) 0
8. conv_pw_1 (Conv2D) (None, 112, 112, 64) 2048
9. conv_pw_1_bn (BatchNormaliza (None, 112, 112, 64) 256
10. conv_pw_1_relu (ReLU) (None, 112, 112, 64) 0
11. conv_pad_2 (ZeroPadding2D) (None, 113, 113, 64) 0
12. conv_dw_2 (DepthwiseConv2D) (None, 56, 56, 64) 576
13. conv_dw_2_bn (BatchNormaliza (None, 56, 56, 64) 256
14. conv_dw_2_relu (ReLU) (None, 56, 56, 64) 0
15. conv_pw_2 (Conv2D) (None, 56, 56, 128) 8192
16. conv_pw_2_bn (BatchNormaliza (None, 56, 56, 128) 512
17. conv_pw_2_relu (ReLU) (None, 56, 56, 128) 0
18. conv_dw_3 (DepthwiseConv2D) (None, 56, 56, 128) 1152
19. conv_dw_3_bn (BatchNormaliza (None, 56, 56, 128) 512

20. conv_dw_3_relu (ReLU) (None, 56, 56, 128) 0
21. conv_pw_3 (Conv2D) (None, 56, 56, 128) 16384
22. convpw3bn (BatchNormaliza (None, 56, 56, 128) 512
23. convpw3relu (ReLU) (None, 56, 56, 128) 0
24. convpad4 (ZeroPadding2D) (None, 57, 57, 128) 0
25. convdw4 (DepthwiseConv2D) (None, 28, 28, 128) 1152
26. convdw4bn (BatchNormaliza (None, 28, 28, 128) 512
27. convdw4relu (ReLU) (None, 28, 28, 128) 0
28. convpw4 (Conv2D) (None, 28, 28, 256) 32768
29. convpw4bn (BatchNormaliza (None, 28, 28, 256) 1024
30. convpw4relu (ReLU) (None, 28, 28, 256) 0
31. convdw5 (DepthwiseConv2D) (None, 28, 28, 256) 2304
32. convdw5bn (BatchNormaliza (None, 28, 28, 256) 1024
33. convdw5relu (ReLU) (None, 28, 28, 256) 0
34. convpw5 (Conv2D) (None, 28, 28, 256) 65536
35. convpw5bn (BatchNormaliza (None, 28, 28, 256) 1024
36. convpw5relu (ReLU) (None, 28, 28, 256) 0
37. convpad6 (ZeroPadding2D) (None, 29, 29, 256) 0
38. convdw6 (DepthwiseConv2D) (None, 14, 14, 256) 2304
39. convdw6bn (BatchNormaliza (None, 14, 14, 256) 1024
40. convdw6relu (ReLU) (None, 14, 14, 256) 0
41. convpw6 (Conv2D) (None, 14, 14, 512) 131072
42. convpw6bn (BatchNormaliza (None, 14, 14, 512) 2048
43. convpw6relu (ReLU) (None, 14, 14, 512) 0
44. convdw7 (DepthwiseConv2D) (None, 14, 14, 512) 4608
45. convdw7bn (BatchNormaliza (None, 14, 14, 512) 2048
46. convdw7relu (ReLU) (None, 14, 14, 512) 0
47. convpw7 (Conv2D) (None, 14, 14, 512) 262144
48. convpw7bn (BatchNormaliza (None, 14, 14, 512) 2048
49. convpw7relu (ReLU) (None, 14, 14, 512) 0
50. convdw8 (DepthwiseConv2D) (None, 14, 14, 512) 4608
51. convdw8bn (BatchNormaliza (None, 14, 14, 512) 2048
52. convdw8relu (ReLU) (None, 14, 14, 512) 0
53. convpw8 (Conv2D) (None, 14, 14, 512) 262144
54. convpw8bn (BatchNormaliza (None, 14, 14, 512) 2048
55. convpw8relu (ReLU) (None, 14, 14, 512) 0
56. convdw9 (DepthwiseConv2D) (None, 14, 14, 512) 4608
57. convdw9bn (BatchNormaliza (None, 14, 14, 512) 2048
58. convdw9relu (ReLU) (None, 14, 14, 512) 0
59. convpw9 (Conv2D) (None, 14, 14, 512) 262144
60. convpw9bn (BatchNormaliza (None, 14, 14, 512) 2048
61. convpw9relu (ReLU) (None, 14, 14, 512) 0
62. convdw10 (DepthwiseConv2D) (None, 14, 14, 512) 4608
63. convdw10bn (BatchNormaliz (None, 14, 14, 512) 2048
64. convdw10relu (ReLU) (None, 14, 14, 512) 0
65. convpw10 (Conv2D) (None, 14, 14, 512) 262144
66. convpw10bn (BatchNormaliz (None, 14, 14, 512) 2048
67. convpw10relu (ReLU) (None, 14, 14, 512) 0
68. convdw11 (DepthwiseConv2D) (None, 14, 14, 512) 4608
69. convdw11bn (BatchNormaliz (None, 14, 14, 512) 2048
70. convdw11relu (ReLU) (None, 14, 14, 512) 0
71. convpw11 (Conv2D) (None, 14, 14, 512) 262144
72. convpw11bn (BatchNormaliz (None, 14, 14, 512) 2048
73. convpw11relu (ReLU) (None, 14, 14, 512) 0
74. convpad12 (ZeroPadding2D) (None, 15, 15, 512) 0

75. convdw12 (DepthwiseConv2D) (None, 7, 7, 512) 4608
76. convdw12bn (BatchNormaliz (None, 7, 7, 512) 2048
77. convdw12relu (ReLU) (None, 7, 7, 512) 0
78. convpw12 (Conv2D) (None, 7, 7, 1024) 524288
79. convpw12bn (BatchNormaliz (None, 7, 7, 1024) 4096
80. convpw12relu (ReLU) (None, 7, 7, 1024) 0
81. convdw13 (DepthwiseConv2D) (None, 7, 7, 1024) 9216
82. convdw13bn (BatchNormaliz (None, 7, 7, 1024) 4096
83. convdw13relu (ReLU) (None, 7, 7, 1024) 0
84. convpw13 (Conv2D) (None, 7, 7, 1024) 1048576
85. convpw13bn (BatchNormaliz (None, 7, 7, 1024) 4096
86. convpw13relu (ReLU) (None, 7, 7, 1024) 0
87. globalaveragepooling2d (Gl (None, 1024) 0
88. dense (Dense) (None, 8) 8200

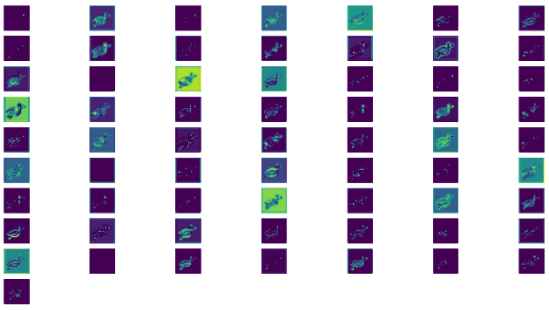
**TTV**

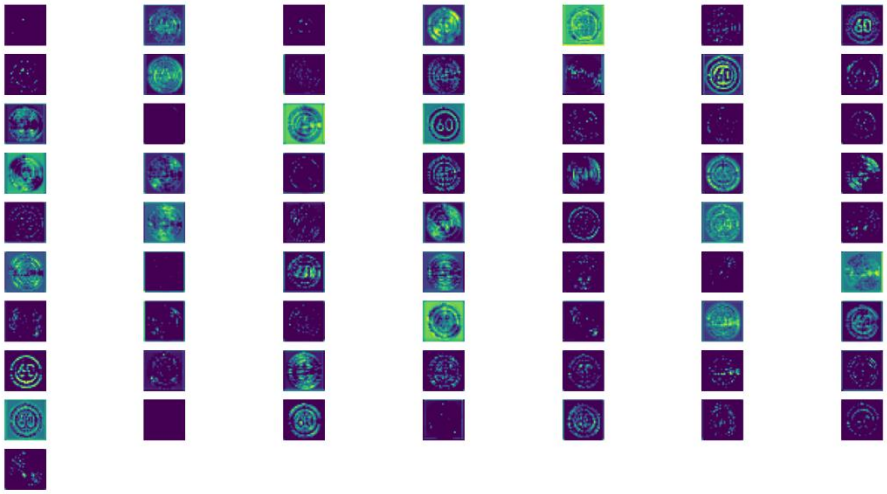
TTV stands for train, test and valid. This is shorthand for the file's names often used by the model.

**Feature map:**

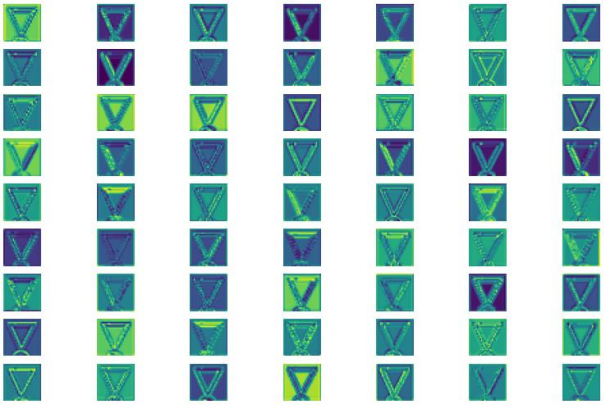
Duck image layer 35 (64 sample)

Layer 24 60mph (64)



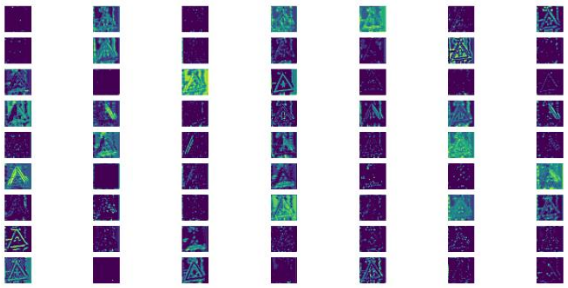


Layer 34 of empty warning (64)

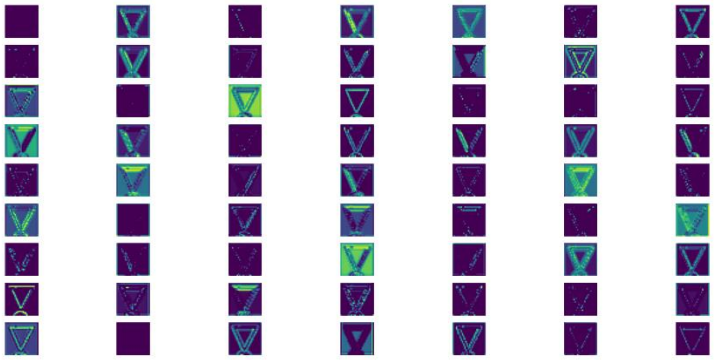


Layer 24 of 70mph (64)

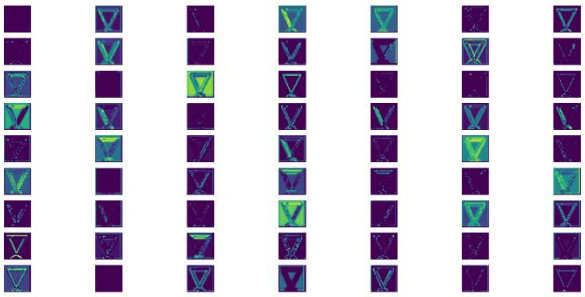
layer 35 of junction sign (64)



layer 35 of warning signs empty (64)



layer 36 of warning sign empty(64)



Appendix B

**Filtering model**

```
from os import path
import os
import cv2 as cv



def files(directory, newDirectory, layer=0, one=False):
    if one:  # one image for system to change
        settings = [False, (244, 244), True]  # defaults settings
        print("current settings: directory = ", directory, "show=", settings[0], "dim=", settings[1],
"localizes=",
              settings[2])
        i = input("change settings? (Y)")
        i = i.lower()
        if i == "y" or i == "yes":  # change settings for one image
```

```
        for x in range(len(settings)):
            print("type new input in same format as: ", settings[x])  # loop the different settings
            b = input()
            if type(settings[x]) is tuple:  # changes size of image

                try:
                    eval(b)
                except:
                    print("wrong input value or no value. !using default!")
                else:
                    if len(eval(b)) == 2:
                        print("layout correct, changing values to", eval(b))
                        settings[x] = eval(b)
                    else:
                        print("wrong value size. !using default!")


            elif b.lower() == "true" or b.lower() == "false":  # chages show, and localizes settings
                b.capitalize()
                settings[x] = bool(b)
                print("correct value, changing to:", b)

            else:
                print("value input wrong or no input. !using default!")

        print("new settings: directory = ", directory, "show=", settings[0], "dim=", settings[1],
"localizes=",
              settings[2])
        print("running")
        q = changeImage(directory, settings[0], settings[1], settings[2])
        cv.imwrite(newDirectory, q)
    else:  # default settings for one image
        print("running default image ")
        a = changeImage(directory)
        cv.imwrite(newDirectory, a)
else:
    settings = [False, (244, 244), True]
    print("current settings: directory = ", directory, "show=", settings[0], "dim=", settings[1],
"localizes=",
          settings[2])
    i = input("change settings? (Y)")
    i = i.lower()
    if i == "y" or i == "yes":
        for x in range(len(settings)):
            print("type new input in same format as: ", settings[x])
            b = input()
            if type(settings[x]) is tuple:
                if b is None:
                    pass
                try:
                    eval(b)
                except:
                    print("wrong input value or no value. !using default!")
                else:
                    if len(eval(b)) == 2:
```

```
                    print("layout correct, changing values to", eval(b))
                    settings[x] = eval(b)
                else:
                    print("wrong value size. !using default!")


            elif b.lower() == "true" or b.lower() == "false":
                b.capitalize()
                settings[x] = bool(b)
                print("correct value, changing to:", b)

            else:
                print("value input wrong or no input. !using default!")

        print("new settings: directory = ", directory, "show=", settings[0], "dim=", settings[1],
"localizes=",
              settings[2])
        print("running")

    filentitles = {}

    for file in os.listdir(directory):
        print(directory + "/" + file)
        filename = os.fsdecode(file)

        if os.path.isdir(directory + "/" + file):
            print("a", file)
            if directory == newDirectory:
                filentitles[filename] = 0

            if filename in filentitles:
                print("two file found with same name. input new name. old name:", filename)
                k = input()
                while True:
                    if k in filentitles:
                        print("all ready named file", k, ". choose new name")
                        k = input()

                    else:
                        break

                filentitles[k] = 0
                apath = newDirectory + "/" + k
                os.mkdir(apath)

            else:
                filentitles[filename] = 0
                apath = newDirectory + "/" + filename
                try:
                    os.mkdir(apath)
                except FileExistsError:
                    print("file already named that. made copy")
                    copyAmount = 0
                    while True:
                        try:
```

```
                    os.mkdir(newDirectory + "/" + filename + str(copyAmount))

                except:
                    copyAmount += 1
                    if copyAmount > 100:
                        raise FileExistsError("warning: over 100 copy of",(newDirectory + "/" +
filename), ". auto stopping")


                else:
                    print("copy made")
                    break

    print(filentitles)

    if layer == 1:
        for names in filentitles:
            counter = 0

            for file in os.listdir(directory + "/" + names):
                if file.endswith((".jpg", ".png")):
                    print(file)
                    newImage = changeImage(os.path.join((directory + "/" + names), file), settings[0],
settings[1],
                                           settings[2])
                    holder = (newDirectory + "/" + names + "/" + str(counter) + ".jpg")
                    cv.imwrite(holder, newImage)
                    counter += 1



    elif layer == 0:
        for file in os.listdir(directory):
            if file.endswith((".jpg", ".png")):
                print(file)
                newImage = changeImage(os.path.join(directory, file), settings[0], settings[1], settings[2]
                                       )
                holder = (newDirectory + "/" + "copy.jpg")
                cv.imwrite(holder, newImage)

print("finished")


def changeImage(directory, show=False, dim=(244, 244), localizes=True):  # Change images to selected
size and grey scale.
    window_name = 'Image'

    if not path.exists(directory):
        raise Exception("cvImageChanger: findImage: no file exists, check location")

    image = cv.imread(
        directory)

    resized = cv.resize(image, dim,
                        interpolation=cv.INTER_AREA)
```

```
    if show:
        print("press 'ESC' to exit")
        cv.imshow(window_name, image)  # shows image ?remove or make command based
        i = cv.waitKey(0)
        if (i == "ESC"):
            cv.destroyAllWindows()

    grey_image = cv.cvtColor(resized, cv.COLOR_RGB2GRAY)  # changes to grey

    if show:
        print("press 'ESC' to exit")
        cv.imshow(window_name, grey_image)  # shows image ?remove or make command based
        i = cv.waitKey(0)
        if (i == "ESC"):
            cv.destroyAllWindows()

    if localizes:
        image = Localize(grey_image)

    if show:
        print("press 'ESC' to exit")
        cv.imshow(window_name, image)  # shows image ?remove or make command based
        i = cv.waitKey(0)
        if (i == "ESC"):
            cv.destroyAllWindows()

    return image


def Localize(img):  # https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html - Otsu's
Binarization
    ret2, th2 = cv.threshold(img, 0, 255,
                             cv.THRESH_BINARY + cv.THRESH_OTSU)  # cvThreshold(image,
binary_image,128,255,CV_THRESH_OTSU)

    return th2



files("C:/Users/gamin/Desktop/dataset sorted", "C:/Users/gamin/Desktop/test_file", layer=1)


#a = changeImage("C:/Users/gamin/Desktop/final project/test3.JPG")

#cv.imwrite("C:/Users/gamin/OneDrive/Desktop/final project/TEST3.JGP", a)
```

**training model**

```
import tensorflow as tf
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
import os
import shutil
```

```
import random
from os import listdir
from os.path import isfile, join


def file_maker(location, topEnd):
    os.chdir(location)
    try:
        os.mkdir("train")
        os.mkdir("valid")
        os.mkdir("test")

    except FileExistsError:
        print("warning file already found")

    for num in range(0, topEnd):
        try:
            os.mkdir("train/" + str(num))
            os.mkdir("valid/" + str(num))
            os.mkdir("test/" + str(num))

        except FileExistsError:
            print("warning file", num, " already found")
            pass

    print("files finished")


def randomSort(location, fileTotal=10, SampleSize=30, testSampleSize=5): #30 5
    paths = {"trainPath": str,
             "validPath": str,
             "testPath": str}

    for z in range(0, fileTotal):

        onlyFiles = [f for f in listdir(location + "/train/" + str(z)) if
                     isfile(join(location + ("/train/" + str(z)), f))]  # finds all files in dict

        samples = random.sample(onlyFiles, SampleSize)
        print("samples", samples)
        for j in samples:  # select all random file to be moved
            shutil.move(location + "/train/" + str(z) + "/" + j,
                        location + "/valid/" + str(z))  # moves from train to test

        onlyFiles = [f for f in listdir(location + "/train/" + str(z)) if
                     isfile(join(location + "/train/" + str(z), f))]  # finds all files in dict again as some moved

        test_samples = random.sample(onlyFiles, testSampleSize)
        for k in test_samples:
            shutil.move(location + "/train/" + str(z) + "/" + k, location + "/test/" + str(z))

    paths["trainPath"] = location + "/train"
    paths["validPath"] = location + "/valid"
    paths["testPath"] = location + "/test"
```

```
    print("random sort done")
    return paths


def process(fileL, layers=10, saved=False, savedName="my_model"):
    tBatch = ImageDataGenerator(
        preprocessing_function=tf.keras.applications.mobilenet.preprocess_input).flow_from_directory(
        directory=fileL["trainPath"], target_size=(224, 224), batch_size=10)
    vBatches = ImageDataGenerator(
        preprocessing_function=tf.keras.applications.mobilenet.preprocess_input).flow_from_directory(
        directory=fileL["validPath"], target_size=(224, 224), batch_size=10)

    mn_mobile = tf.keras.applications.mobilenet.MobileNet()

    x = mn_mobile.layers[-6].output  # run tests

    out = Dense(units=layers, activation="softmax")(x)
    untrained_m = Model(inputs=mn_mobile.input, outputs=out)

    for layer in untrained_m.layers[:-2]:
        layer.trainable = False
    untrained_m.compile(loss="categorical_crossentropy", metrics=["accuracy"],
optimizer=Adam(learning_rate=0.0001))

    x = input("view summary? (Y/N)")
    x.lower()
    if x == "y":
        untrained_m.summary()

    untrained_m.fit(x=tBatch,
                    steps_per_epoch=len(tBatch),
                    validation_data=vBatches,
                    validation_steps=len(vBatches),
                    epochs=25, #25
                    verbose=2,
                    )

    if saved:
        untrained_m.save("saved_model/" + savedName)
        print("NN saved as", savedName)
    else:
        print("nn not saved")

    print("nn finished")


def main():
    while True:
        print("")
        selection = input("please select 'file_maker' to produce images files,"
                          " 'random sort' randomly spilt images to different files, "
                          "'rprocess' to randomly split images and process model,  "
                          "'process' to process/train model"
                          "'exit' to end: ")
        selection.lower()
```

```
        if selection == "file_maker" or selection == "f":
            a = input("please input file location")
            b = input("please image types amount")
            file_maker(a, int(b))
        elif selection == "random sort" or selection == "r":
            a = input("please input file location")
            b = input("please image types amount")
            path = randomSort(a, int(b))
            print("paths:" + str(path))
        elif selection == "rprocess" or selection == "rp":
            a = input("please input file location")
            b = input("please image types amount")
            paths = randomSort(a, int(b))
            print("image done. processing model")
            q = input("save model? (True/False)")
            q.capitalize()
            if q == "True":
                s = input("save model name? ")
                process(paths, int(b), saved=bool(q), savedName=s)
            else:
                process(paths, int(b))
        elif selection == "process" or selection == "p":
            c = {"trainPath": str, "validPath": str}
            a = input("please input path for training")
            c["trainPath"] = a
            a = input("please input path for valid")
            c["validPath"] = a
            b = input("please image types amount")
            q = input("save model? (True/False)")
            q.capitalize()
            if q == "True":
                s = input("save model name? ")
                process(c, int(b), saved=bool(q), savedName=s)
            else:
                process(c, int(b))

        elif selection == "exit" or selection == "e":
            break
        else:
            print("error, option not selected. try again.")
            print("")

    quit()


if __name__ == "__main__":
    main()
```

**main model**

```
import numpy as np
import tensorflow as tf
import cv2 as cv
from os import path
from tensorflow.keras.preprocessing import image
```

```
import os
import matplotlib.pyplot as plt

while True:
    try:
        new_model = tf.keras.models.load_model("./saved_model/model_2")
        break

    except:
        print(
            "error finding saved model. please ensure model is in the same file location in /saved_model/ for new use under name 'model_2'.")
        a = input("please enter model path of put exit to end program: ")
        if a == "exit" or a == "Exit":
            exit()
        else:
            new_model = tf.keras.models.load_model(a)
            break


def changeImage(directory, show=False, dim=(244, 244),
                localizes=True):  # Change images to selected size and grey scale.
    window_name = "Image"

    name = os.path.basename(os.path.normpath(directory))
    currectDic = os.getcwd()

    if not path.exists(currectDic + "/run_images"):
        os.mkdir(currectDic + "/run_images")

    if not path.exists(directory):
        raise Exception("cvImageChanger: findImage: no file exist, check location")

    item = cv.imread(
        directory)  # finds image, currently using test image replace original = cv.LoadImageM("image.jpg")

    resized = cv.resize(item, dim,
                        interpolation=cv.INTER_AREA)  # replace thumbnail = cv.CreateMat(original.rows/ 10, original.cols / 10, original.type) cv.Resize(original, thumbnail)

    if show:
        print("press ESC to exit")
        cv.imshow(window_name, item)  # shows image ?remove or make command based
        i = cv.waitKey(0)
        if (i == "ESC"):
            cv.destroyAllWindows()

    grey_image = cv.cvtColor(resized, cv.COLOR_RGB2GRAY)  # changes to grey

    if show:
        print("press ESC to exit")
        cv.imshow(window_name, grey_image)  # shows image ?remove or make command based
        i = cv.waitKey(0)
        if (i == "ESC"):
```

```
            cv.destroyAllWindows()

    if localizes:
        item = Localize(grey_image)

    if show:
        print("press ESC to exit")
        cv.imshow(window_name, item)  # shows image ?remove or make command based
        i = cv.waitKey(0)
        if (i == "ESC"):
            cv.destroyAllWindows()

        print(name)

    cv.imwrite(currectDic + "/run_images" + "/" + name, item)
    return (currectDic + "/run_images"), name


def Localize(img):
    ret2, item = cv.threshold(img, 0, 255,
                              cv.THRESH_BINARY + cv.THRESH_OTSU)  # cvThreshold(image,
binary_image,128,255,CV_THRESH_OTSU)

    return item


def prepare_image(currectDic, name):
    if not path.exists("./run_images"):
        raise Exception("prep_image: findImage: no file exist, check location")

    img = image.load_img(currectDic + "/" + name, target_size=(224, 224))
    # img = PIL.Image.open(currectDic + "/" + name, (224, 224))
    # print(img)
    img_array = image.img_to_array(img)
    img_array_expanded_dims = np.expand_dims(img_array, axis=0)
    return tf.keras.applications.mobilenet.preprocess_input(img_array_expanded_dims)


def guess():
    signs = ["30 mph", "50 mph", "60 mph", "70 mph", "junction", "Warning (empty)", "no entry",
             "Warning (!)"]  # correct names?

    LOC = input("please input full address of image: ")
    show = input("show image? True/False ")
    show.capitalize()
    if len(show) == 4:
        bool(show)
        currectDic, name = changeImage(LOC, show)  # C:\Users\gamin\OneDrive\Desktop\final
project\1116055.jpg
    else:
        currectDic, name = changeImage(LOC)
    tool = prepare_image(currectDic, name)

    predictions_single = new_model.predict(tool)
    # print(predictions_single)
```

```
    for i in signs:
        if np.argmax(predictions_single) == signs.index(i):
            print("this sign is: ", np.argmax(predictions_single), ". Also know as: ", i)


def summ():
    new_model.summary()


def layers():
    print("total layers: " + str(len(new_model.layers) - 3))
    a = input("input first layer: ")
    if int(a) < 2:
        print("too low, changed to 2")
        a = "2"
    b = input("to layer: ")
    if int(b) > len(new_model.layers) - 3:
        print("too high changed to " + str(len(new_model.layers) - 3))
        b = str(len(new_model.layers) - 3)
    if (int(a) - int(b)) != 0:
        print("notice: layers done backwards")

    LOC = input("please input full address of image: ")
    show = input("show image? True/False ")
    for i in reversed(range(int(a), int(b) + 1)):
        print("layer - " + str(i))

        model_cut = tf.keras.Model(inputs=new_model.inputs, outputs=new_model.layers[i].output)
        if len(show) == 4:
            bool(show)
            currectDic, name = changeImage(LOC, show)
        else:
            currectDic, name = changeImage(LOC)
        img = prepare_image(currectDic, name)

        nodeImages = model_cut.predict(img)

        amountSize = nodeImages.shape[3]
        if amountSize > 100:
            user = input("amount of images over 64. recommend a sample instead. press Y for a smaller
sample.")
            user.lower()
            if user == "y":
                amountSize = 64
                print("smaller size chosen")
            else:
                print("layering of plot will be done. warning lag may happen due to lots of images. ")

        size0 = amountSize // 7
        size = amountSize % 7
        layer = 1

        fig = plt.figure(figsize=(10, 7))
        for images in range(0, amountSize):
```

```
            fig.add_subplot((size0 + size), 7, images + 1)
            plt.imshow(nodeImages[0, :, :, layer - 1])
            plt.axis('off')
            #plt.title("MapFilter = " + str(i) + "." + str(layer))
            layer += 1

        print("image of layer: " + str(i))
        plt.show()

    print("finished")


def main():
    while True:
        print("")
        selection = input("please select 'guess' to input an image and guess which sign it is,"
                          " 'layer' to pick a image and see how each layer breaks it down, "
                          "'summary' to see total model summary or 'Exit' to end program: ")
        selection.lower()
        if selection == "guess" or selection == "g":
            guess()
        elif selection == "layer" or selection == "l":
            layers()
        elif selection == "summary" or selection == "s":
            summ()
        elif selection == "exit" or selection == "e":
            break
        else:
            print("error, option not selected. try again.")
            print("")

    quit()


if __name__ == "__main__":
    main()
```

References:

Abu, M.A., Indra, N.H., Rahman, A.H.A., Sapiee, N.A. and Ahmad, I., 2019. A study on Image Classification based on Deep Learning and Tensorflow. International Journal of Engineering Research and Technology, 12(4), pp.563-569.

Baggio, D.L., 2012. Mastering OpenCV with practical computer vision projects. Packt Publishing Ltd.

Chen, B. (2020). Learning Rate Schedule in Practice: an example with Keras and TensorFlow 2.0. [online] Medium. Available at: https://towardsdatascience.com/learning-rate-schedule-in-practice-an-example-with-keras-and-tensorflow-2-0-2f48b2888a0c [Accessed 6 Apr. 2022].

Cilimkovic, M., 2015. Neural networks and back propagation algorithm. Institute of Technology Blanchardstown, Blanchardstown Road North Dublin, 15, pp.1-12.

docs.anaconda.com. (n.d.). Installation — Anaconda documentation. [online] Available at: https://docs.anaconda.com/anaconda/install/#requirements [Accessed 9 Apr. 2022].

docs.opencv.org. (n.d.). OpenCV: Miscellaneous Image Transformations. [online] Available at: https://docs.opencv.org/4.x/d7/d1b/groupimgprocmisc.html#ggaa9e58d2860d4afa658ef70a9b1115576a147222a96556ebc1d948b372bcd7ac59 [Accessed 30 Mar. 2022].

Gedraite, E.S. and Hadad, M., 2011, September. Investigation on the effect of a Gaussian Blur in image filtering and segmentation. In Proceedings ELMAR-2011 (pp. 393-396). IEEE.

Geronimo, D., Serrat, J., Lopez, A.M. and Baldrich, R., 2013. Traffic sign recognition for computer vision project-based learning. IEEE transactions on education, 56(3), pp.364-371.

Ingham, J. (2019). Shocker as drivers don't recognise no entry or bus only signs. [online] Express.co.uk. Available at: https://www.express.co.uk/life-style/cars/1116055/shocker-drivers-don-t-recognise-no-entry-or-bus-only-signs [Accessed 29 Mar. 2022].

Instructions to install Python 3. (n.d.). [online] Available at: https://www.shahandanchor.com/home/wp-content/uploads/Python-installation-instructions-1.pdf.

Jeremy B (2018) unsplash: duck. [online] Available at: https://unsplash.com/photos/LJSH3NOTLwc

Kingma, D.P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. [online] arXiv.org. Available at: https://arxiv.org/abs/1412.6980.

Kiran, A.H. and Verbeek, P.P., 2010. Trusting our selves to technology. Knowledge, Technology & Policy, 23(3), pp.409-427.

Kothari, J.D., 2018. A Case Study of Image Classification Based on Deep Learning Using Tensorflow. Jubin Dipakkumar Kothari (2018). A Case Study of Image Classification Based on Deep Learning Using Tensorflow. International Journal of Innovative Research in Computer and Communication Engineering, 6(7), pp.3888-3892.

Larochelle, H., Mandel, M., Pascanu, R. and Bengio, Y., 2012. Learning algorithms for the classification restricted boltzmann machine. The Journal of Machine Learning Research, 13(1), pp.643-669.

Li, D., Zhao, D., Chen, Y. and Zhang, Q., 2018, July. Deepsign: Deep learning based traffic sign recognition. In 2018 international joint conference on neural networks (IJCNN) (pp. 1-6). IEEE.

Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y. and Alsaadi, F.E., 2017. A survey of deep neural network architectures and their applications. Neurocomputing, 234, pp.11-26.

Liu, Y.Q., Du, X., Shen, H.L. and Chen, S.J., 2020. Estimating generalized gaussian blur kernels for out- of-focus image deblurring. IEEE Transactions on circuits and systems for video technology, 31(3), pp.829-843.

matplotlib.org. (2022). Matplotlib: Python plotting — Matplotlib 3.3.4 documentation. [online] Available at: https://matplotlib.org/stable/index.html.

Mercededes-Benz ECQ, Traffic Sign Assist, [ONLINE] Available at: https://www.mercedes-benz.co.uk/passengercars/mercedes-benz-cars/models/eqc/safety.pi.html/mercedes-benz-cars/models/eqc/safety/driving-assistance-gallery/traffic-sign [Accessed 5 MAR 2022]

mykola (2018). GTSRB - German Traffic Sign Recognition Benchmark. [online] Available at: https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign.

Noble, W.S., 2006. What is a support vector machine?. Nature biotechnology, 24(12), pp.1565-1567.

OS (2019). os — Miscellaneous operating system interfaces — Python 3.8.0 documentation. [online] Available at: https://docs.python.org/3/library/os.html.

Persson, S. (mars 2018). Examensarbete 30 hp 6 Mars 2018 Application of the German Traffic Sign Recognition Benchmark on the VGG16 network using transfer learning and bottleneck features in Keras. [online] Available at: https://www.diva-portal.org/smash/get/diva2:1188243/FULLTEXT02.pdf [Accessed 16 Apr. 2022].

PyCharm Help. (n.d.). Get started | PyCharm. [online] Available at: https://www.jetbrains.com/help/pycharm/2021.3/quick-start-guide.html [Accessed 9 Apr. 2022].

random (2022). random — Generate pseudo-random numbers — Python 3.8.2 documentation. [online] Available at: https://docs.python.org/3/library/random.html.

Rojas, R. (1996). Neural Networks - A Systematic Introduction. [online] Semantic Scholar. Available at: https: ww.semanticscholar.org/paper/Neural-Networks-A-Systematic-Introduction-Rojas/c585a68ec44ae0c64417e5fb29ea597a90fbd580 [Accessed 6 Nov. 2021]

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986). Learning representations by back- propagating errors. Nature, [online] 323(6088), pp.533–536. Available at: https://www.semanticscholar.org/paper/Learning-representations-by-back-propagating-errors-Rumelhart-Hinton/052b1d8ce63b07fec3de9dbb583772d860b7c769#references [Accessed 6 Nov. 2021].

Sajjad, K.M., 2010. Automatic license plate recognition using python and opencv. Department of Computer Science and Engineering MES College of Engineering.

shutil (2010). shutil — High-level file operations — Python 3.7.4 documentation. [online] Available at: https://docs.python.org/3/library/shutil.html.

Stallkamp, J., Schlipsing, M., Salmen, J. and Igel, C., 2011, July. The German traffic sign recognition benchmark: a multi-class classification competition. In The 2011 international joint conference on neural networks (pp. 1453-1460). IEEE.

Stallkamp, J., Schlipsing, M., Salmen, J. and Igel, C., 2011, July. The German traffic sign recognition benchmark: a multi-class classification competition. In The 2011 international joint conference on neural networks (pp. 1453-1460). IEEE.

Tabernik, Domen and Sko{\v{c}}aj, Danijel, (2019), IEEE Transactions on Intelligent Transportation Systems - Tabernik2019ITS. [online] Available at: https://prints.vicos.si/publications/369 [accessed 12 Apr 2022]

Tai, S.K., Dewi, C., Chen, R.C., Liu, Y.T., Jiang, X. and Yu, H., 2020. Deep learning for traffic sign recognition based on spatial pyramid pooling with scale analysis. Applied Sciences, 10(19), p.6997.

Team, K. (n.d.). Keras documentation: Losses. [online] keras.io. Available at: https://keras.io/api/losses/.

TensorFlow Core v2.8.0 , (N/A), tf.keras.applications.mobilenet.preprocessinput. https://www.tensorflow.org/apidocs/python/tf/keras/applications/mobilenet/preprocessinput [online] Available at:  [Accessed 6/04/2022].

TensorFlow. (n.d.). DeepDream | TensorFlow Core. [online] Available at: https://www.tensorflow.org/tutorials/generative/deepdream.

TensorFlow. (n.d.). Install TensorFlow 2. [online] Available at: https://www.tensorflow.org/install.

The Highway Code (2021) Traffic signs, Signs giving orders. [online] Available at: https://assets.publishing.service.gov.uk/media/58170307ed915d61c5000000/the-highway-code-trafficsigns.pdf.

Thomas, K.P., Guan, C., Tong, L.C. and Prasad, V.A., 2008, August. An adaptive filter bank for motor imagery based brain computer interface. In 2008 30th Annual international conference of the IEEE engineering in medicine and biology society (pp. 1104-1107). IEEE.

Waltz, F.M. and Miller, J.W., 1998, October. Efficient algorithm for gaussian blur using finite-state machines. In Machine Vision Systems for Inspection and Metrology VII (Vol. 3521, pp. 334-341). International Society for Optics and Photonics.

Wei Wang, Yutao Li, Ting Zou, Xin Wang, Jieyu You, Yanhong Luo, 2020, "A Novel Image Classification Approach via Dense-MobileNet Models", 2020. [online] Available at: https://doi.org/10.1155/2020/7602384 [Accessed 29 Mar. 2022].

Zhang, Y. and Wu, L., 2012. Classification of fruits using computer vision and a multiclass support vector machine. sensors, 12(9), pp.12489-12505.