Inspiration

In a previous hackathon, we wanted to implement image classification into our model, but had zero idea on how to do it, and ended up abandoning it, due to the time constraints given. When this datathon came around, we really wanted to pick a project that used image classification, so we could better use it for future projects. We got the idea as we were randomly talking with our friends at lunch one day, and it was the perfect project for us.

Dataset

The dataset we are using is the famous GTRSB, or the German Traffic Sign Recognition Benchmark dataset. This dataset consists of over 50,000 pictures, split into 40,000 test images, and the remaining 10,000 images split half and half into testing and validation data. We downloaded this data in a format known as “pickle”(.p) which is used in python to easily be able to store huge amounts of data, and is simple to extract.

Learning Problem and Model

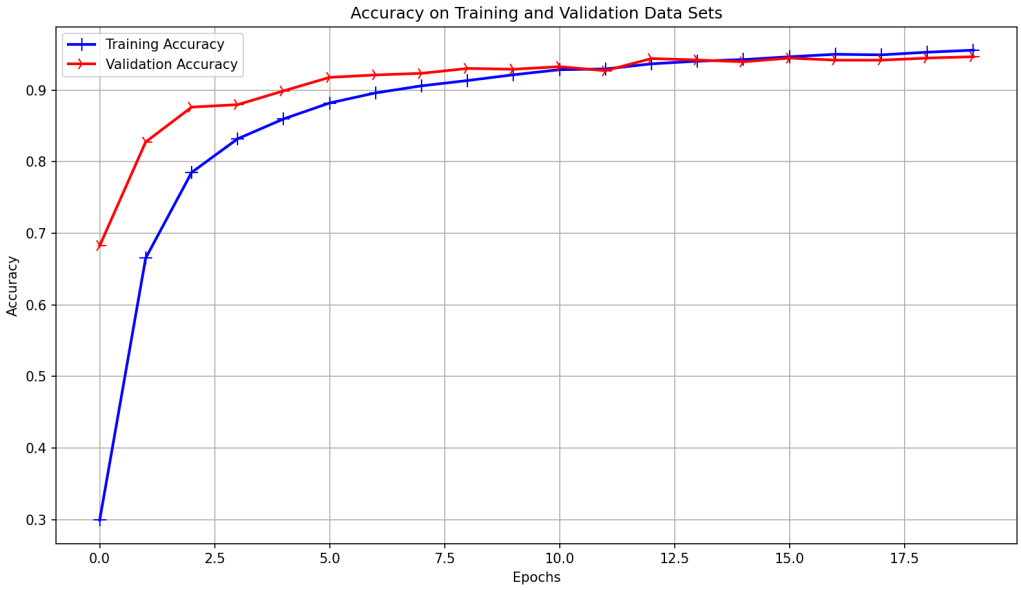
Our problem is a supervised problem, since all of our data has been labeled beforehand, and our job is just to classify the images, and classification falls under supervised learning. The model we are implementing is a CNN (Convolutional neural network) mode, which is specifically used for image recognition. How it works is first there is convolution, which puts the image through certain filters, which look for specific features within the image. Then there is pooling which simplifies the number of parameters by down-sampling, which decreases the number of factors, which makes the learning process faster. This is repeated through until the features are identified. After this the output is flattened, and fed through a neural network, which then gives a probability for each output. We will later show the architecture for our model.

Observations and Limitations

One major thing we noticed while trying out the model was that color, or lighting had zero effect on any of the results. We knew that this was the case due to the fact that we grayscale it, which helps us find the importance of grayscaling. Other observations we found with the model was that it was really good detecting images that were flipped, or upside down signs. For example, one of the signs called “double bound” was opening left or right, and the mode detected it anyway. Also, another example is the yield sign, when it was upside down, the model still predicted it. This was probably the variety of the data in the dataset we had. There were limitations though. The main limitation was that a difference in shape, font, or angle causes the mode to predict incorrectly. This can be fixed by altering the array of the picture, but due to the time constraints we were not able to do this.

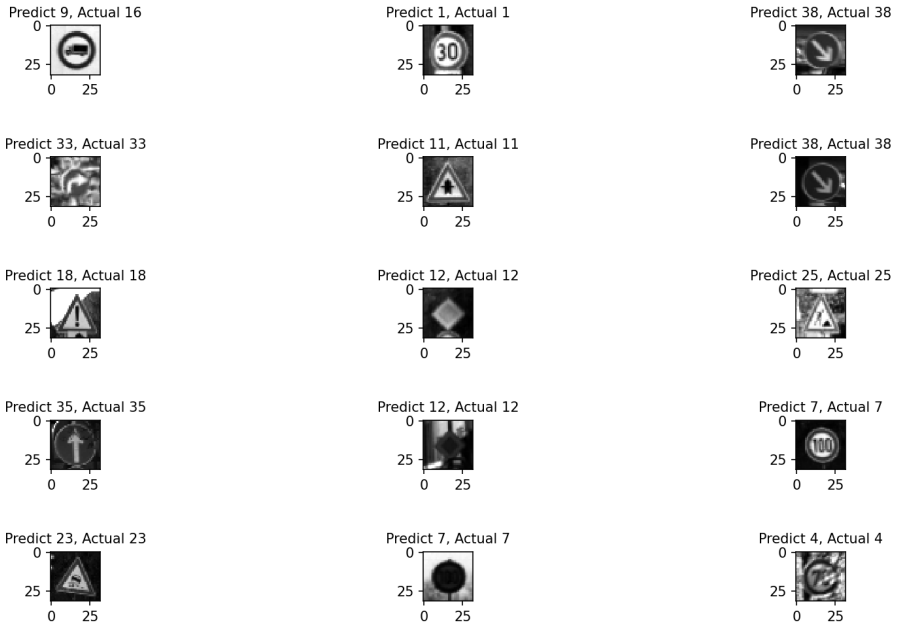
Visualizations

Here is a graph that shows the accuracy over epochs from the training and evaluation data

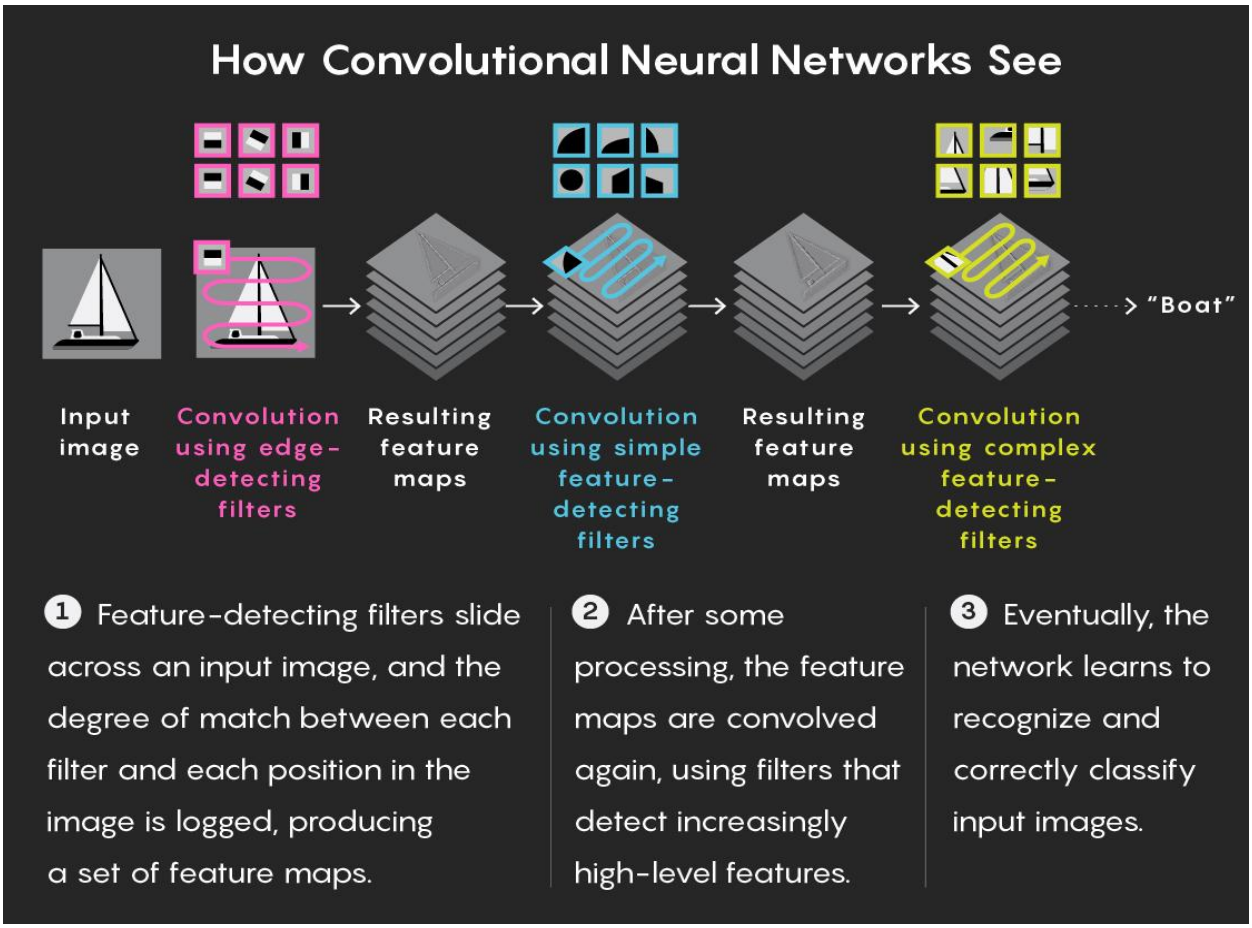


As you notice, the accuracy goes closer and closer to 100% over the number of epochs, showing how our model’s accuracy increases over the number of epochs, for the Training and Validation Data.

Here is another visualization with some of the different pictures, which has their predicted vs actual class



Here's an infographic on how a CNN Model works:



As you can see, first there is a layer that uses filters to detect features in the image, which results in feature maps. Then different filters are used again, to detect simple features. Then finally, complex features are being detected. This is done a certain number of times until the CNN model can tell what the image is. There are more intricacies, mainly involving simplifying and decreasing the number of data points, using other layers.

Google Colab Notebook Link: https://colab.research.google.com/drive/12fSCw_OwsIr9rU3jl3CziWF6tTeFDhC

Here you can see the code, and an in-depth explanation of what it does. It's also in the drive folder.