گزارش پروژه دوم – قسمت اول (تابلو های رانندگی)

سروش حیدری :: 96222031

Main idea and a general explanation:

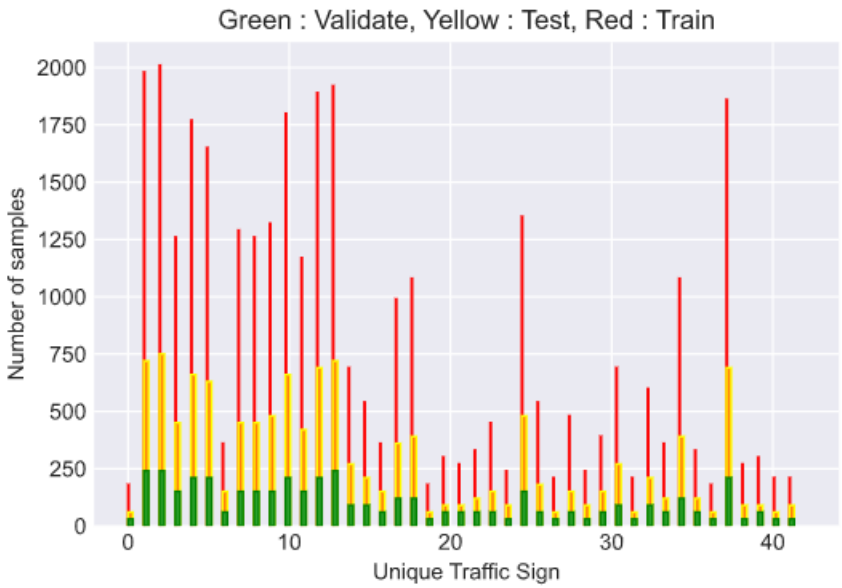
The main idea was to try to improve out convolutional network from raw pictures with data augmentations and other techniques like noise injections, though the basic raw data itself was appropriate enough to give us around 90-95 percent Accuracy but we'll try to improve that even it be just less than 5 percent

The steps are as shown below :

Yet before we hop into the training process first let us have a look at what's the distribution of our data in train, test and our validate sections :

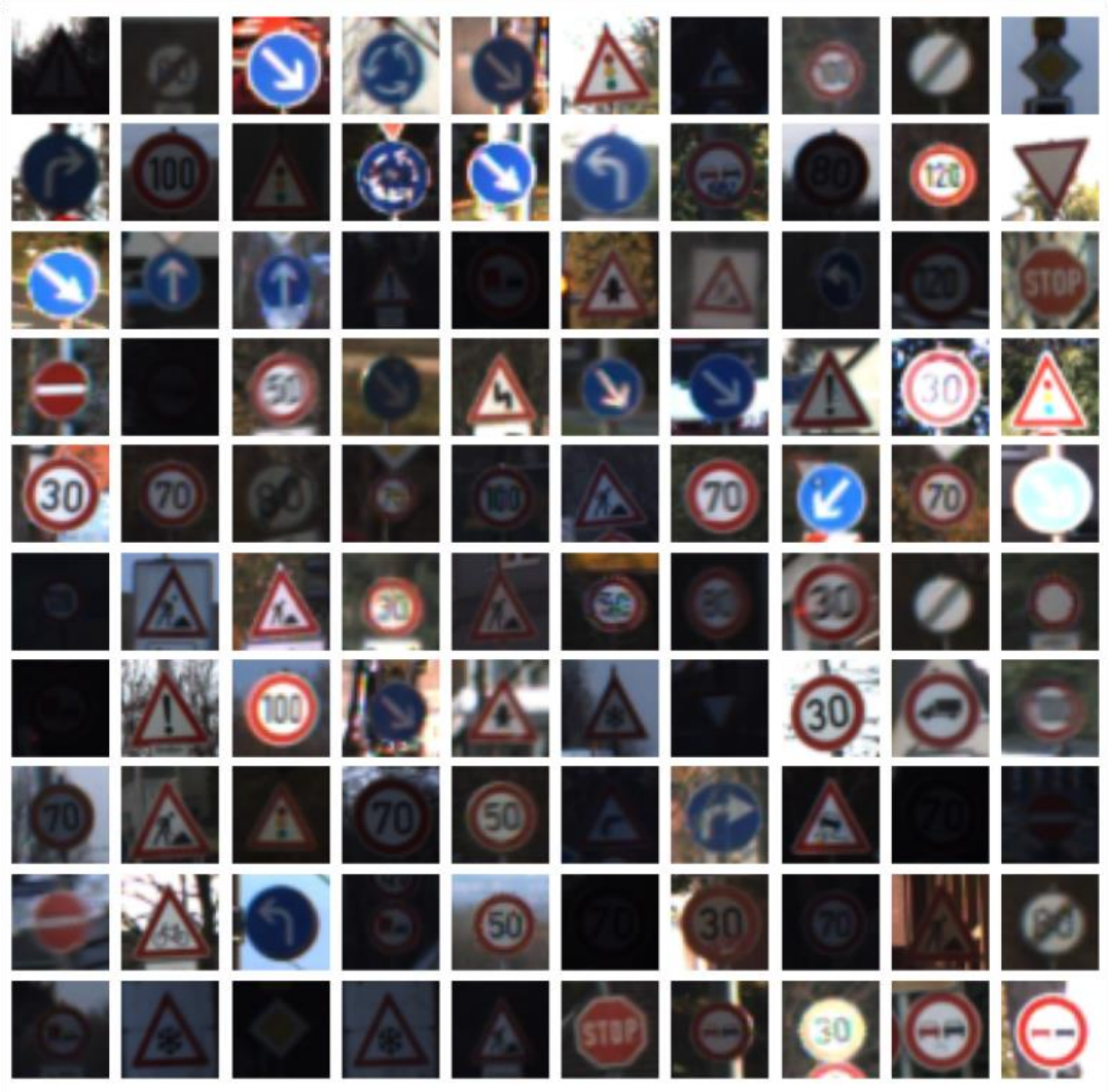
This charts shows us how many of each traffic sign is present in the data set

(For the sake of simplicity and better visualizations the signs are shown with their ID and not their name)



1 – our first network has no data Augmentation, our data are colored thus our input channel is (32, 32, 3)

Lets take a peek at our data :



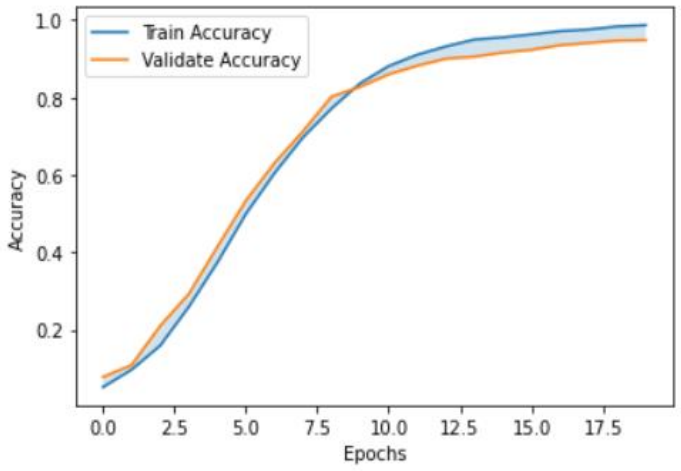
As we see there are various light conditions in our raw data we'll have a network trained by this data just to have a measure of improvement for late works

The hyper params and our network structure for this purpose is shown below :

```
epoch_num = 20
input_shape = (32, 32, 3)

network.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Conv2D(64, (3, 3), activation='relu'))
network.add(layers.MaxPooling2D((2, 2)))

network.add(layers.Flatten())
network.add(layers.Dense(500, activation='relu'))
network.add(layers.Dense(43, activation='softmax'))

network.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = network.fit(x, y, validation_split=0.2, epochs=epoch_num, batch_size=64)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_2 (Dense)              (None, 500)               1152500
_________________________________________________________________
dense_3 (Dense)              (None, 43)                21543
=================================================================
Total params: 1,193,435
Trainable params: 1,193,435
Non-trainable params: 0
_________________________________________________________________
```

And the output we got out of it would be :



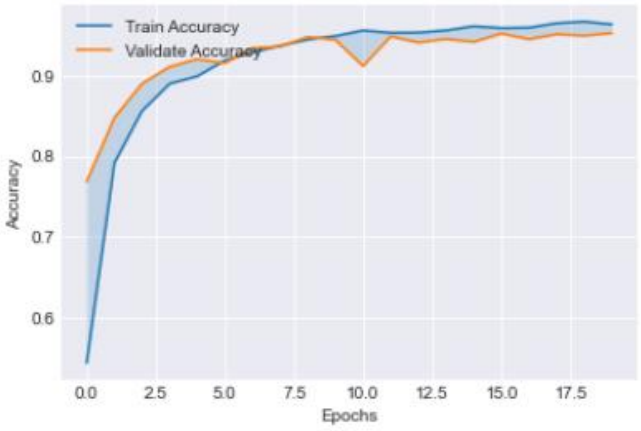
We can see even without dropout layers our validate and train accuracy have very little inconsistency, one thing we can learn from this fact is that :

Having all of our data in a certain angle and distance (simply having the data with very little differences in their angle of shape) could greatly improve the generalization of the network

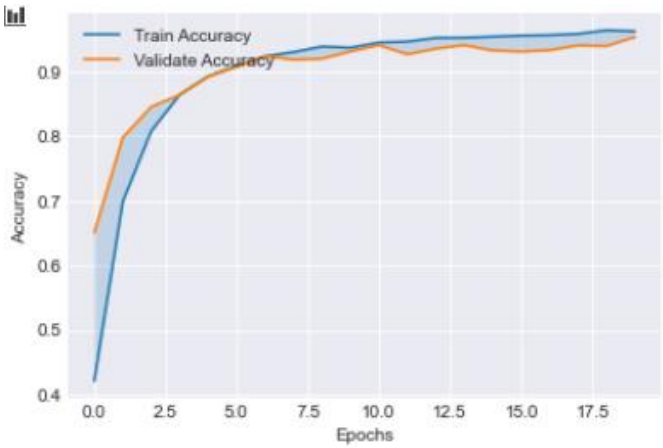
(this is the default version we're gonna work on later)

There were other variations with the same inputs and different hyper params

There's some of the shown below :



convolutional layer sizes = (64, 128)



convolutional layer sizes = (32, 64)

There are not any difference with them(little to none !)

---

2 – our next step is to greyscale our data, we'll see how that does affect our accuracy

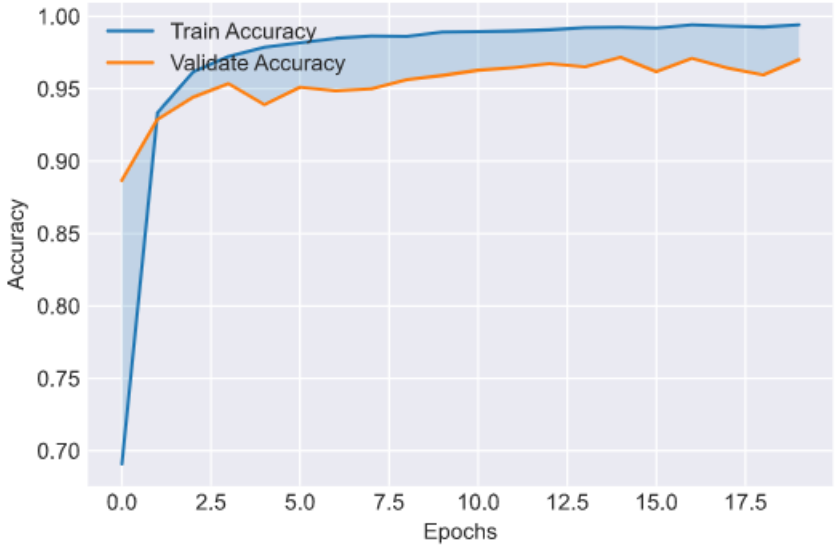
Lets first see the data in greyscale

Its notable to say that the picture above is a normalized version of our greyscaled data (the greyscale and normalized versions aren't really different we see the 3 versions below)

And I trained the network on the default hyper parameters stated above just with the difference of our input data his time we'll use the normalized version of our data

We can see our training accuracy is near 100 percent though I didn't use a drop our layer (and I should have!)

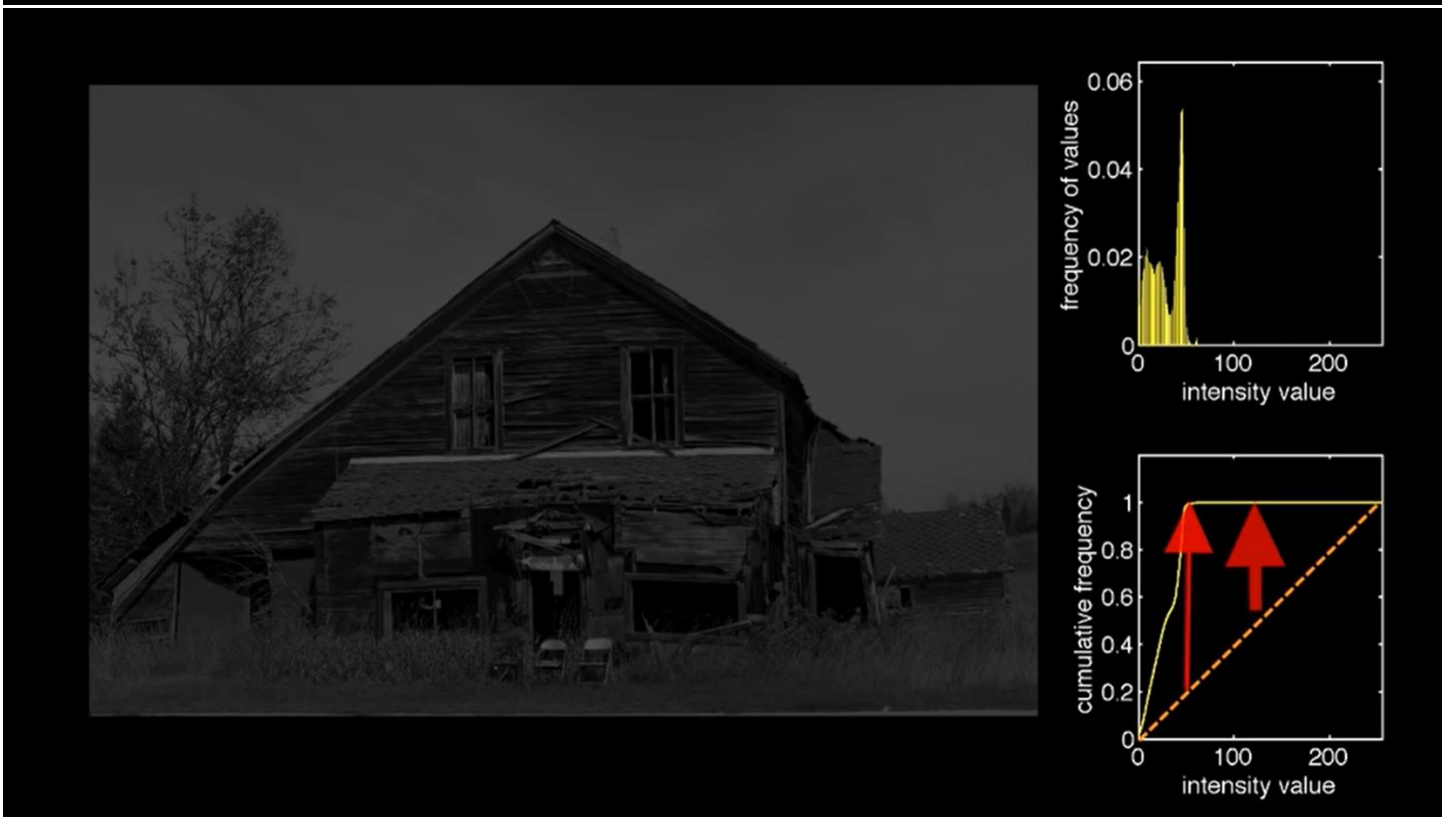
With a dropout layer we'd have around 97 percent accuracy of both validation and training data



Some of the data in the normalized version we saw above we're almost unusable due to their extreme darkness or luminance for this purpose we'll use a technique called "histogram equalization" there's more detail on the topic below :

In histogram equalization a chart of frequency per each unique intensity value is created meaning that we'll calculate the intensity of each and every pixle then a chart of frequencies of every unique intensity is calculated

Theres 2 pictures here to demonstrate the chart of each (before and after equalization) below :



The second picture is before doing the equalization if we look at the chart we'll see the amount of pixles with high intensities are a lot more than of the lower intensities after the equalization(pic 1) the chart is almost like a straight line

Here is some samples of our data after the equalization

So much better now !

And the results are :

Sadly I forgot to take a shot of the plot though the percents are as shown below and we can see a slight improvement over our validate accuracy reaching almost 97.5 percernt

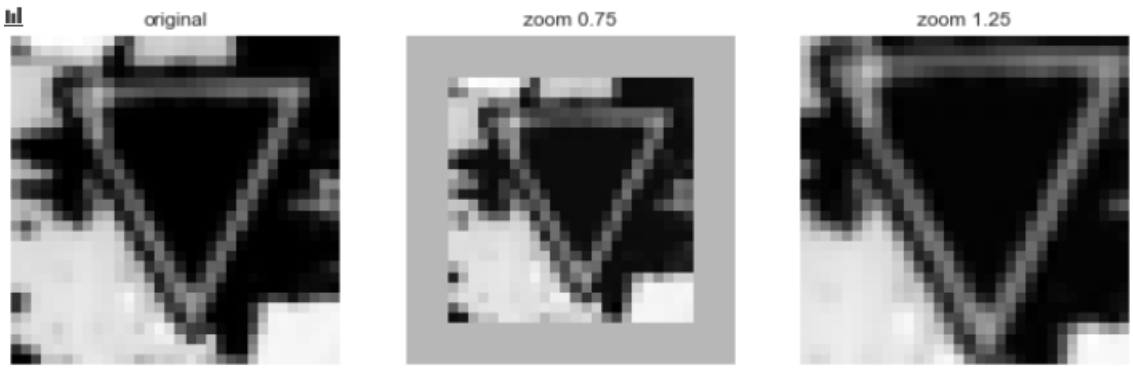
```
Epoch 20/20
544/544 [==============================] - 32s 60ms/step - loss: 0.0256 - accuracy: 0.9912 - val_loss: 0.0863 - val_accuracy: 0.9748
Model: "sequential_4"
```

So far all we've done is to change the data and not augmenting anything to it

Here we'll have data augmentation with 2 steps : first having two different zoomed version (1.25 and 0.75)

And second : augmenting noised images and also a custom filter which will be discussed late)

Adding zoomed versions of our data :

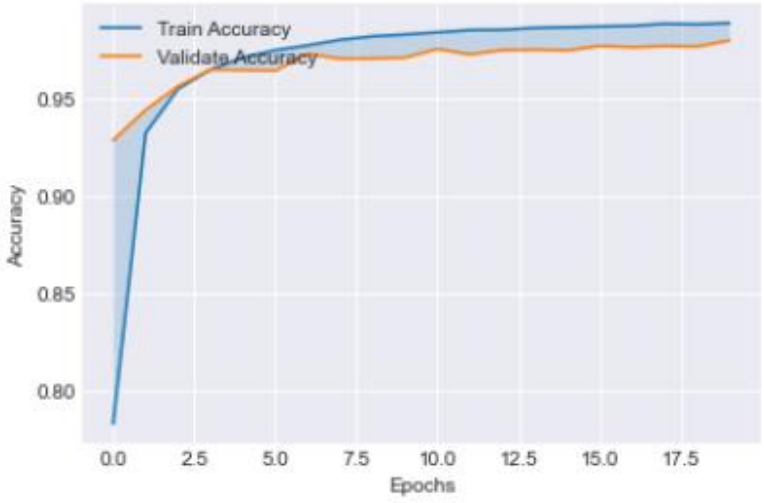


This is a quick demonstration of what we'll have for each of our pictures, this means we'll now have 104379 pictures to train on and 13230 pictures to validate (for some of the signs which already had a high number of samples we wont do this augmentation so we'll get a more distributed data

In the samples we had of the original data we could see that almost all the sign had sizes in this 3 versions so we do expect to have an improvement after this augmentation

We're seeing the validate accuracy of 98 percent which has improved by 0.5 percent in respect to out previous version (very little but still something 😊)

Also the convergence of our train accuracy has improved too meaning having zoomed pictures let our network understand the patterns faster !



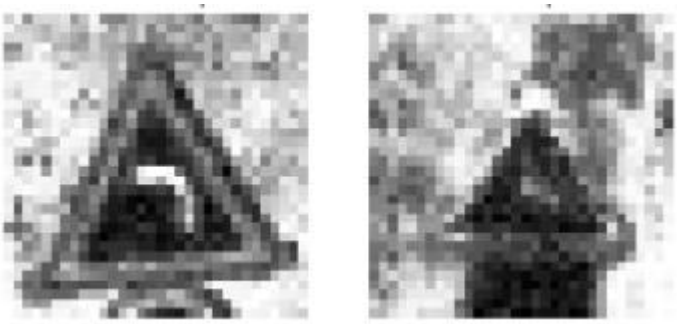

```
Epoch 20/20
1632/1632 [==============================] - 103s 63ms/step - loss: 0.0339 - accuracy: 0.9890 - val_loss: 0.0747 - val_accuracy: 0.9800
Model: "sequential_7"
```

On the second step of data augmentation we'll inject some noise on our data as well as a custom filter applied to the histo-equalized version of the normalized data this time we'll have around 170000 data (not exactly)

And an improvement on the results

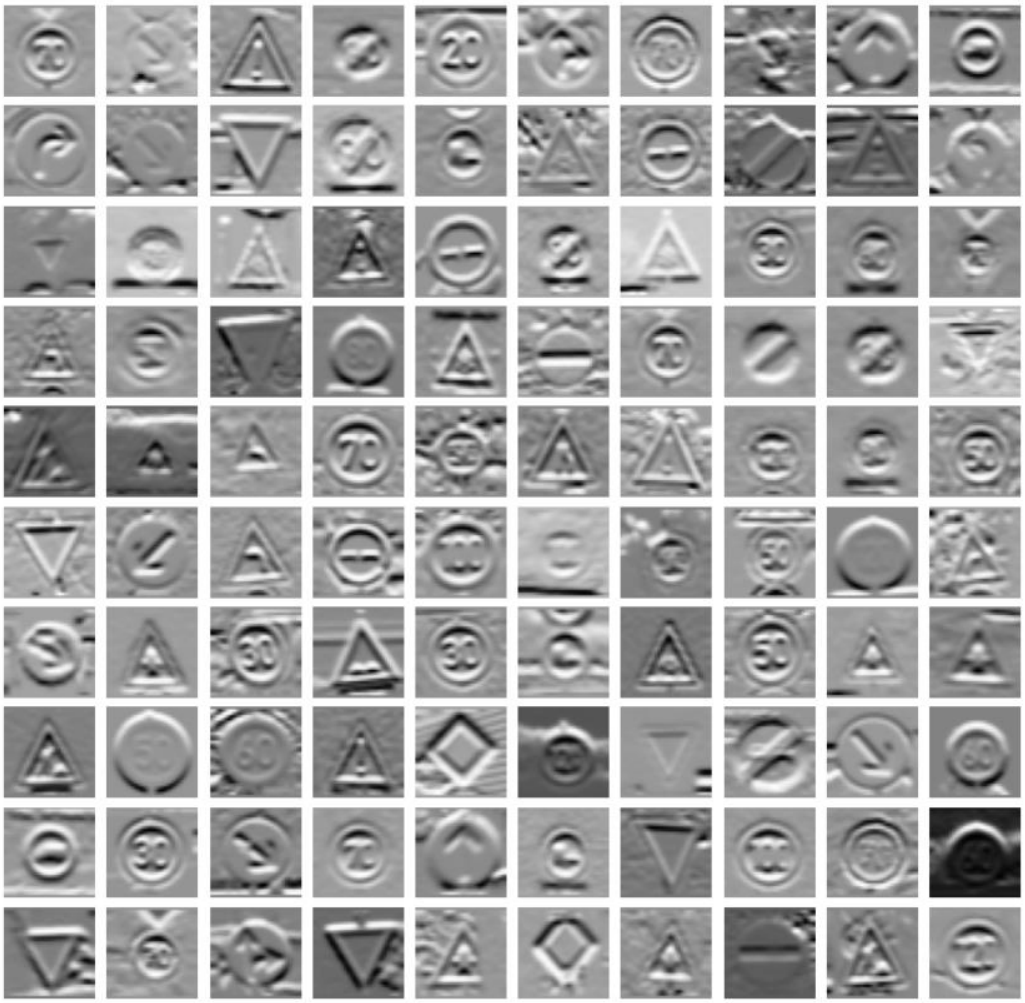
This is a sample of what our noised data might look like

For this purpose we used gaussian noise injection algorithm

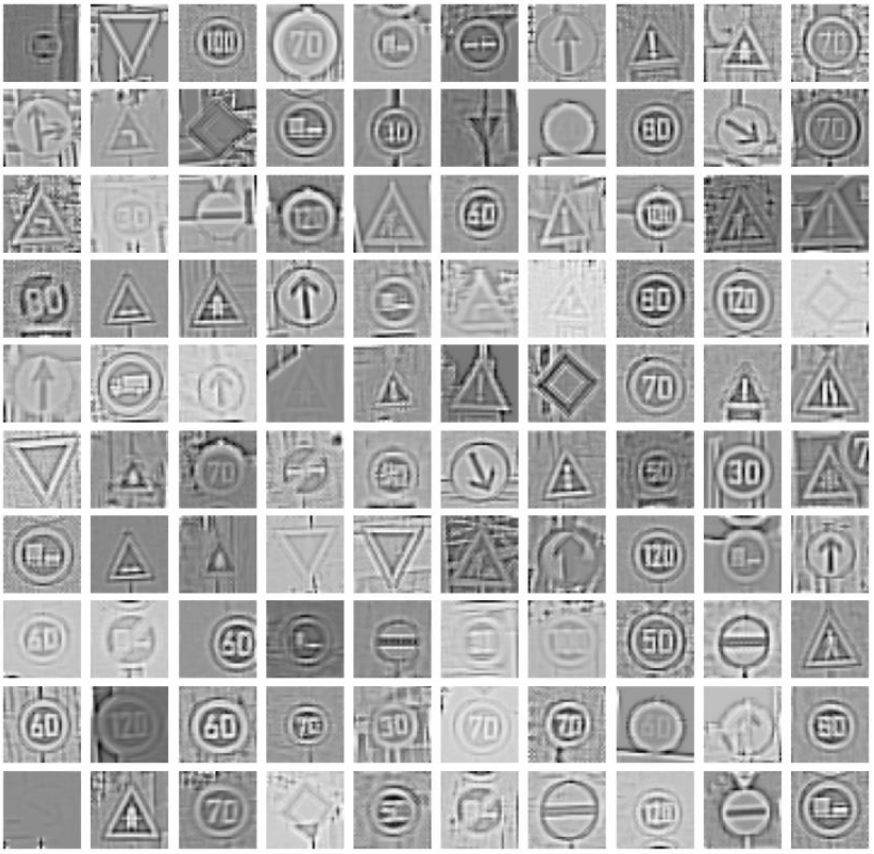


And also our custom filter which is done using the matrix below is :

```
([[ -1, -2, -1], [ 0, 0, 0], [ 1, 2, 1]])
```



There was another filter that would give us this data but they both resulted in the same accuracy so I just used the first filter and not both (we don't want to have too much data due to complexity of calculations



Though due to high amount of data augmentation which required a LOT of processing power and also

my fragile laptop which is at the brink of inner collapse :D , I couldn't train the network with the full range of 20 epochs and it just went only 7 epochs far

though we can see that with only "7" epochs we got the same accuracy as what we had before with 20 epochs

I call that improvement :D

```
Epoch 7/20
2719/2719 [==============================] - 177s 65ms/step - loss: 0.0649 - accuracy: 0.9800 - val_loss: 0.1001 - val_accuracy: 0.9701
```

So the best result we could end up with would be 99 percent of training accuracy and 98 percent of validating accuracy (if we don't count the last version in!)

(unfortunately my laptop froze and I couldn't take the test accuracy)