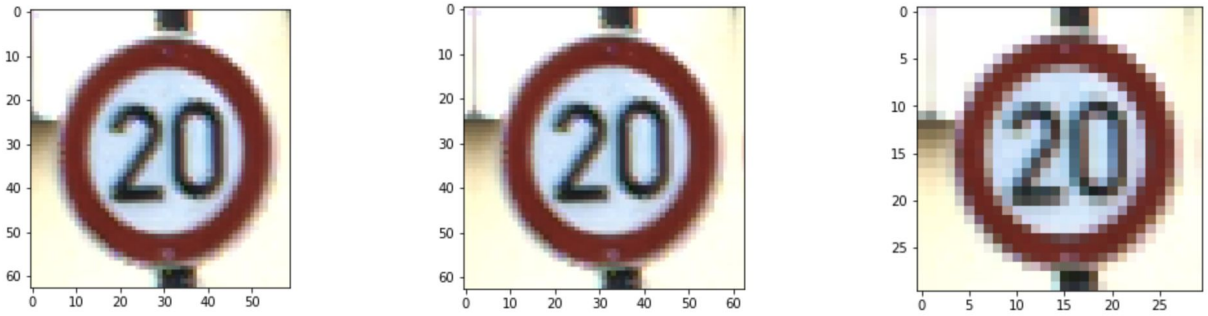
Machine Learning

GERMAN TRAFFIC SIGN RECOGNITION

NIKITA LOZHNIKOV

Preprocessing

Original | Padded | Padded and Resized



The padding was done using `skimage.util.pad` function that allows adding padding easily by providing a tuple that represents, which col/row you want to add. It also allows to use a `reflection` mode that enables padding with the nearest pixels instead of zeros.

The resizing was performed with the usage of `skimage.transform.resize` package that provides a very neat functionality like “antialiasing” for undersized images and “range preserve” for oversized ones.

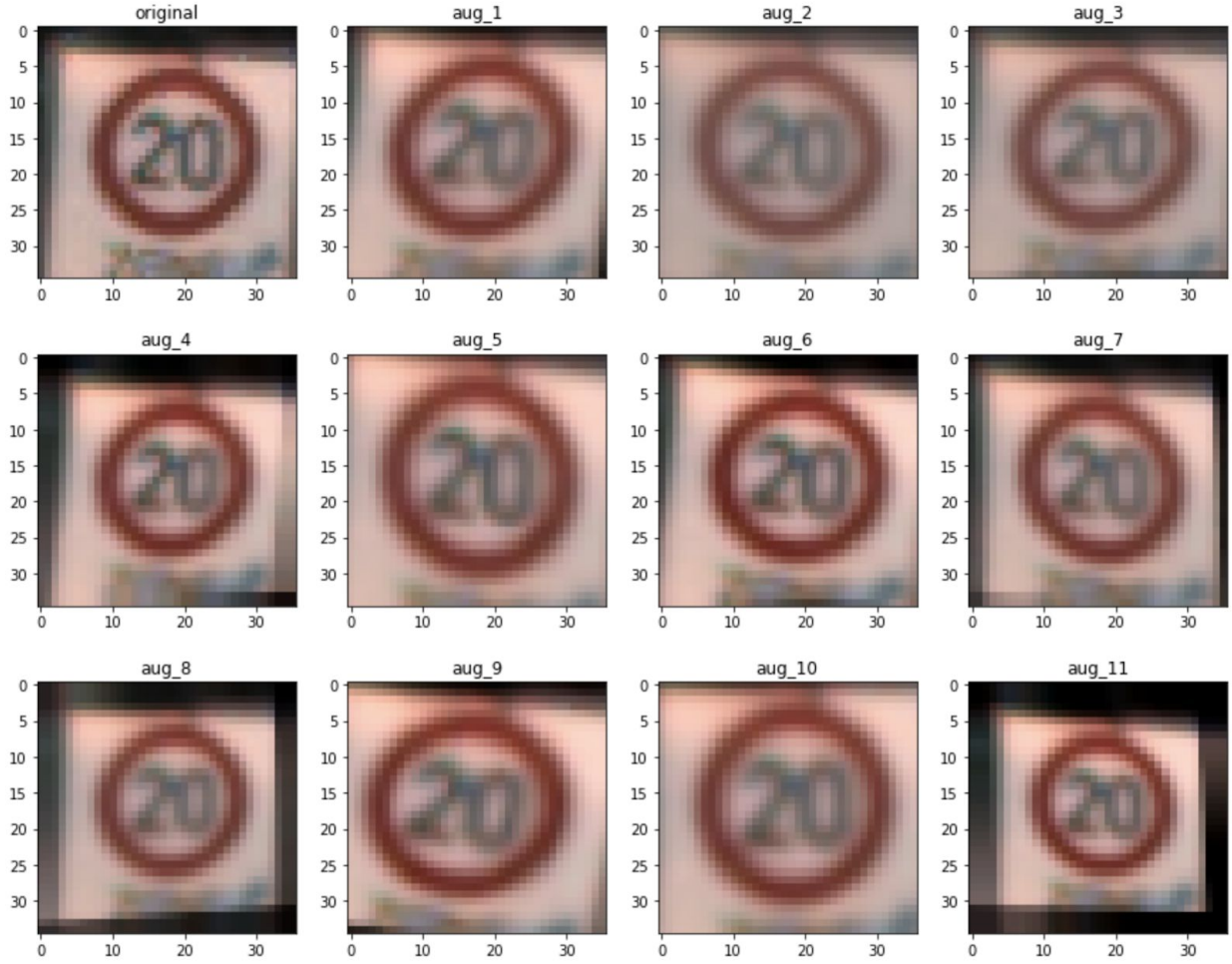
The normalization of the image (0..1 pixel scaling) is enabled by default `numpy` arrays.

> *Make sure that images from the particular track end up being in either training or validation split, otherwise validation will be faulty (Think why?).* -- The answer is that there’re multiple very close pictures so if we split tracks we will effectively have leakage of the training samples into the test/validation data and thus our result will not be credible.

Augmentation

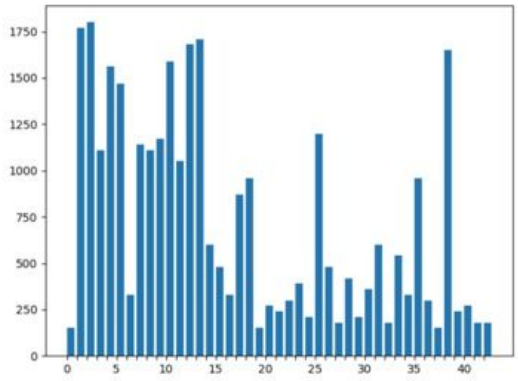
For this part, I used `imgaug` package that allows one easily define image transformation in a declarative manner, with probabilistic/affine/linear/perspective/else transformations. I decided to apply `Gaussian Blur` in 50% of the cases (randomly), then `Perspective Tranformation` - the intuition here is that the signs on the road can be squizzed due to the pillar being tilted. This is also where `Affine` transformations come in.

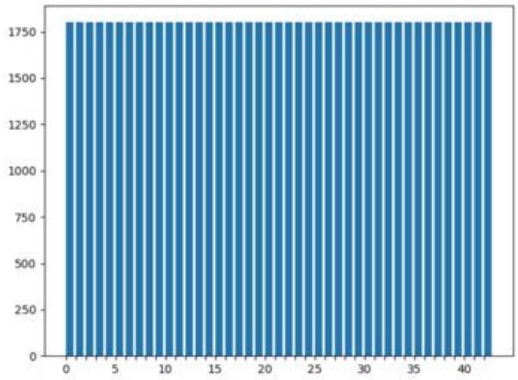
On the image below you could see what it looks like when I pass the same image to the transformation object 15 times (*top left image is the original, others - transformation results*).



I decided to avoid all (both vertical and horizontal) flips because on our case it may leed to extra ambiguity, for example, a red triangle, or an arrow sign do have different meanings when flipped, thus, this augmentation I considered harmful and decided to see how just a blur and perspective transformations will affect the result.

I also decided to introduce augmentation to all of the classes. I picked the most frequent class, multiplied its frequency by 1.5 and this was my final goal for the classes to be. E.g. if there're 3 classes with frequencies 150, 500 and 1000, then the total number of samples will be 4500. 1000 is the most frequent. Times 1.5 => 1500. And there will be 3 classes of 1500 elements.





This is what class distribution histogram looks like before the augmentation.

After the augmentation is was all painted in blue uniformly just as we have seen in the assignment.

Evaluation

Metrics

This is the result of training `RandomForestClassfier` of the whole Final_Training with mentioned above augmentation.

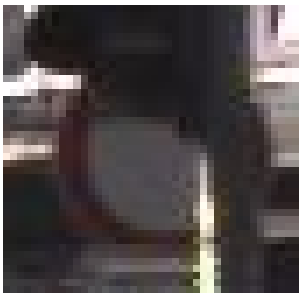
	Accuracy	**Precision**	**Recall**	**F1**
Micro	0.76	0.76	0.76	**0.76**
Macro	0.76	0.69	0.71	0.69
Weighted	0.76	0.77	0.76	0.76

In our case the imbalanced data issue we tried to negotiate with augmentation thus, *Micro* metrics are the same.[micro/macro/weighted explained]

Misclassified

True class 12

Predicted class 15



These are examples of the classes

These are the worst (f1-score < 50) performant classes for our model

	precision	recall	f1-score	support
0	0.40	0.17	0.24	60
19	0.41	0.60	0.49	60
20	0.30	0.63	0.41	90
21	0.66	0.37	0.47	90
23	0.43	0.43	0.43	150
27	0.05	0.02	0.03	60
30	0.46	0.37	0.41	150
32	0.58	0.32	0.41	60

Experiment and analyze!

Non-augmented data

With non-augmented data, there was a magnitude order less data so there was little chance that with this imbalanced data the model will be performant.

	Accuracy	**Precision**	**Recall**	**F1**
Micro	0.62	0.62	0.59	0.58
Macro	0.65	0.55	0.60	0.55
Weighted	0.70	0.63	0.59	**0.61**

Image size

I didn't make the charts, but form logs I recorded during the testing I found that

1. The bigger the size of the image is - the higher is the accuracy (but there's saturation)
2. The bigger the image is - the slower the algorithm works