Introduction to Machine Learning

Assignment 2: Report

Irek Nazmiev, SE-2

Outline

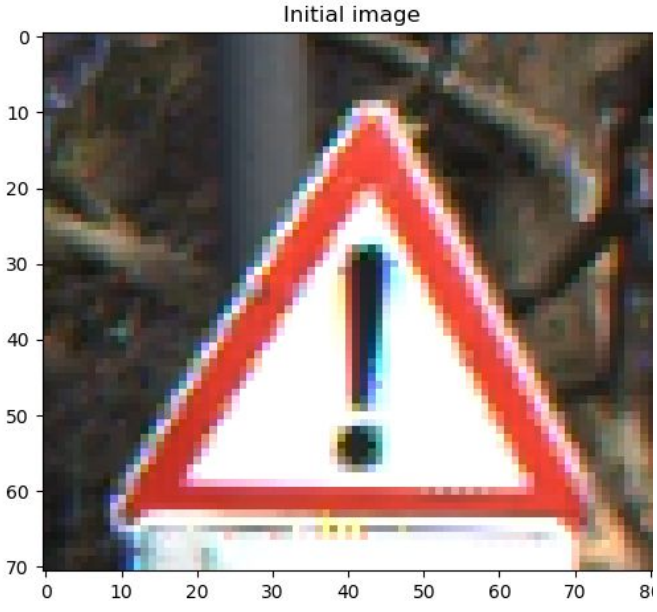
1. Reading images from training dataset
2. Images transformation to square shape
3. Images transformation to the same size
4. Splitting data into training and validation sets
5. Augmentation of images from training set
6. Images normalization
7. Building image set matrix
8. Train the model
9. Model evaluation
10. Experiments
11. Conclusion

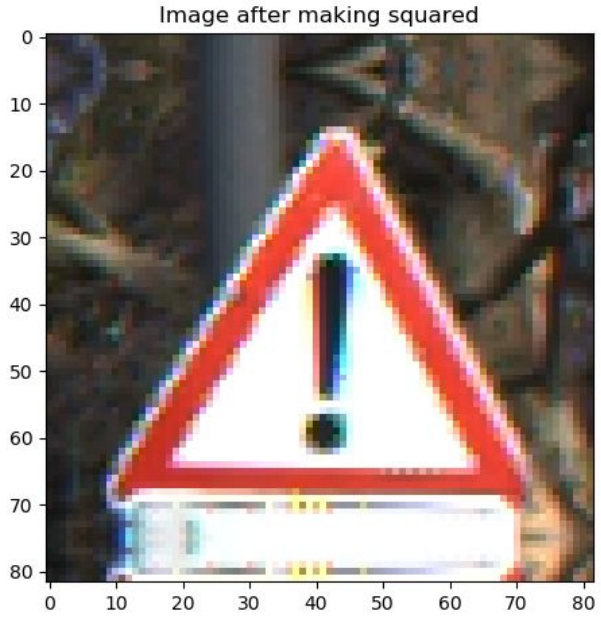
Reading images from training dataset

The provided script was edited in order to satisfy requirements stated in the program and embedded into main python file. Now it's not necessarily to be 43 classes, the amount of classes and class names are read from training data directory. Also the list of images information is returned containing each image's width, height, class and track - these values are needed for further program work.

Images transformation to square shape

Using class PadIfNeeded of Albumentations library, we add padding to the least dimension of each image. Also, the border_mode=4 is used by default, which is cv2.BORDER_REFLECT - it fills borders with colors similar to closer ones. This decision is better than adding black borders and increases model's accuracy on 0.5-2%, because the final image looks more natural and close to real-life ones. Black border are less unlikely to appear in photos used for prediction.





Images transformation to the same size

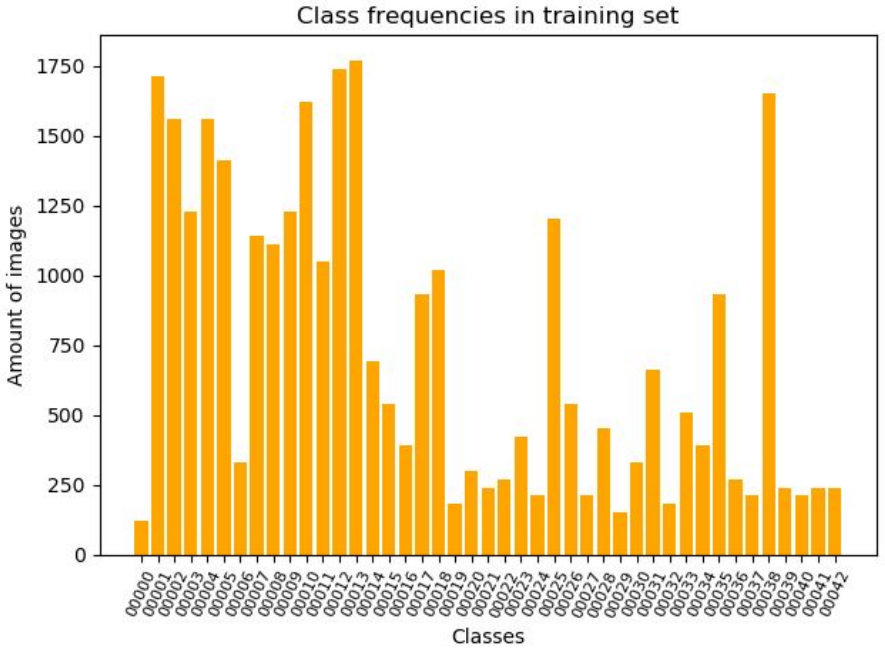
For transforming images to the same size, the cv2 library is used. Function resize gets image and change its size to the specified values.

Splitting data into training and validation sets

The goal is to split the both images and images information data into 2 sets in proportion of 4 to 1 - training set and validation set. Training set will be used for teaching the model while validation set - for validating the model. Each time the data should be shuffled in case of making unique both sets, splitted proportionally from one data set.

First, we need to take into account the fact stating that images are separated not only in classes, but in tracks too (tracks are sets of photos taken from different distances), and we should keep images from one track either in training set or validation set. This have to be made, because pictures from one track are quite similar. That's why being in both validation and training sets, pictures of one track will lead to the score of validation to be larger than it should be in reality.

In the program, unique tracks are generated by merging class and track names and excluding replicated ones. Then we get a set of unique tracks over all classes. The proportion of these unique tracks is taken, and the whole image set is split according to these tracks.



If we look at the bar chart of classes frequency in training set, it's obvious that they spread not evenly. For better model teaching we need to have more various images of each class (ideally, we need to balance all classes). This can be done by augmentation.

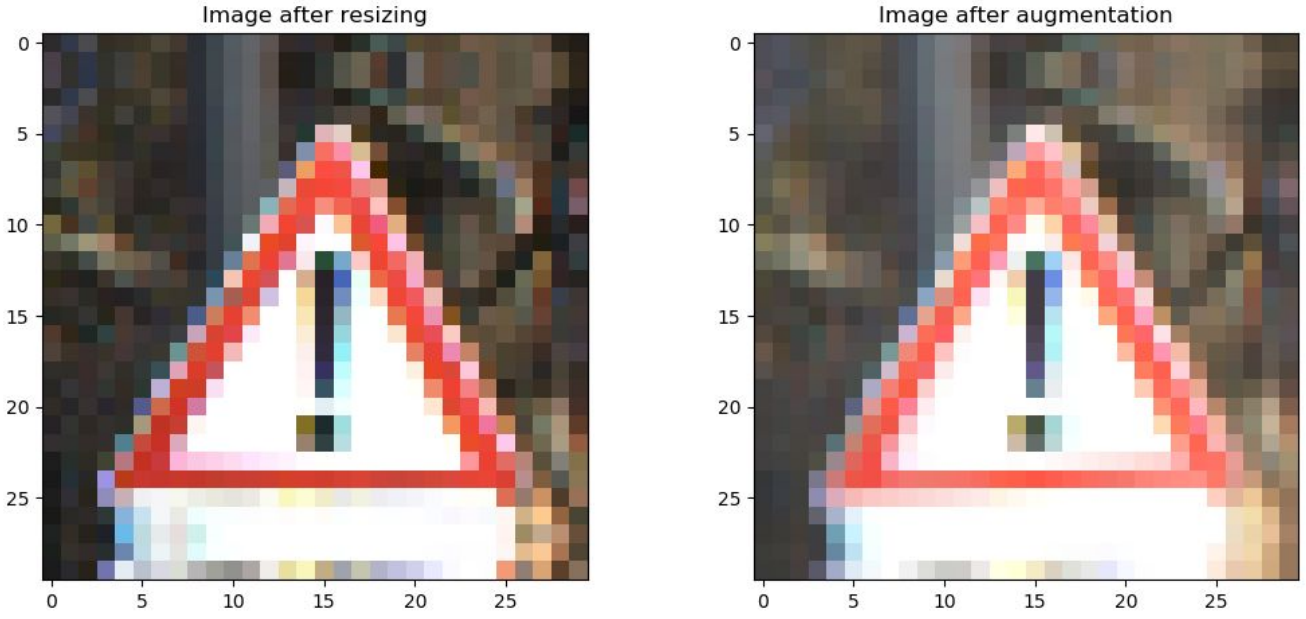
Augmentation of images from training set

From augmentation techniques presented in given resources (1, 2) I highlighted for myself the following 3 ones:

1. Rotation
2. Brightness
3. Contrast

These methods are most likely useful in case of traffic signs recognition. The accuracy of recognition is straightforwardly dependent on the quality of training images. The photos from dataset are taken from different distances, that’s why cropping technique is not included in the list above.

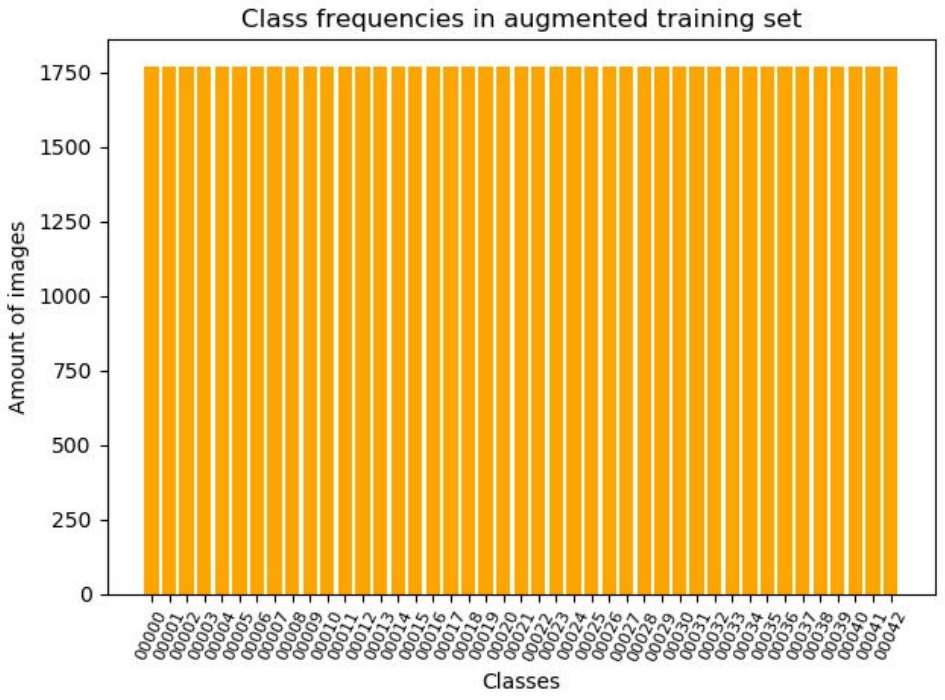
The photos may be taken during different weather conditions, at different times of a day and by different users. That's why these photos will most likely be taken with different brightness level (depending on the time of day), level of contrast (depending on camera specification, time of day) and degree of rotation (depending on user’s photo skills). Other methods connected to hue, saturation levels changing and others are quite farther from real-world cases.



On the example above, the contrast is less, the brightness is more, and there's a small rotation counter-clockwise. All images generated during augmentation process get each of 3 image processing effects with 100% probability, which increases the variety of a set. By doing small changes, the augmentation seems not to spoil the model, but improve, because new images look new and similar to the real-world ones.

For each class in the training set, the augmentation algorithm takes random image and processes it. This happens the necessary amount of times for class to be balanced with other classes (to get size equal to maximum size through all classes before augmentation).

After the augmentation, all classes in the training set are balanced and have an equal number of images each. Each class is supplemented with new images generated from existing ones (with small differences imitating real-life conditions that will help in better teaching the model in a future).



Images normalization

This stage is a specific preparation for data to be put into the model. Normalization is a representation of image's pixel as values from 0 to 1. This can be done by dividing each pixel's RGB value by 255. In the program, numpy array of image is just divided by 255, that performs the necessary action. This is done to both training and validation sets.

Building image set matrix

This is the final stage for data to be prepared for putting into the model. The program represents each image as 1-dimensional vector using numpy function ravel. As a result, we get matrix of 1D vectors representing the whole image set. Now data is prepared for being used in model teaching.

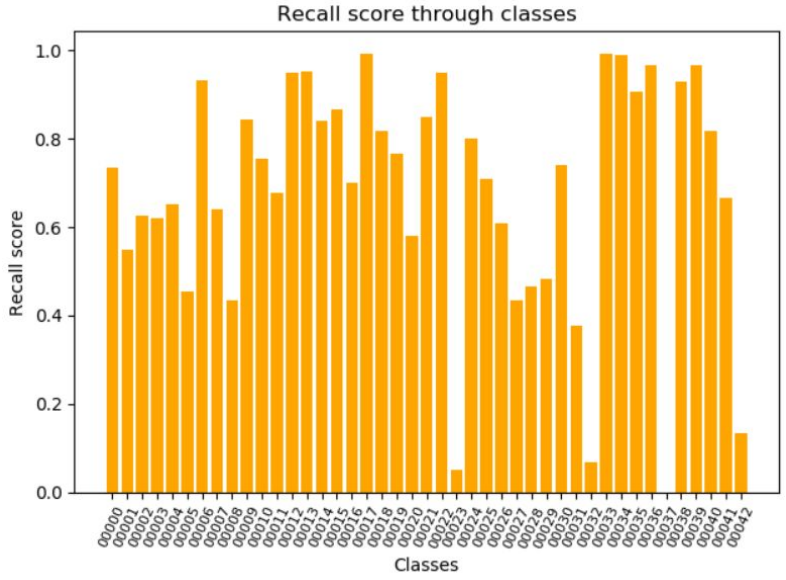
Train the model

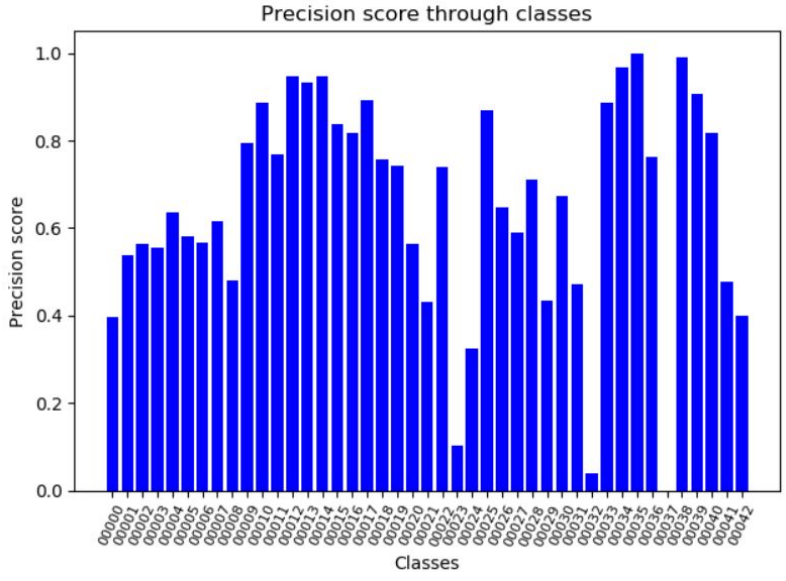
Random forest classifier from scikit-learn library is trained with parameter of 20 estimators, using prepared training set with predefined answer labels (containing correct class names). With image size 30x30, using augmentation and validation set for validating the model, the accuracy is 0.71. Using 200 estimators, the program resulted with total accuracy equal to ~0.8 which is a good result.

Model evaluation

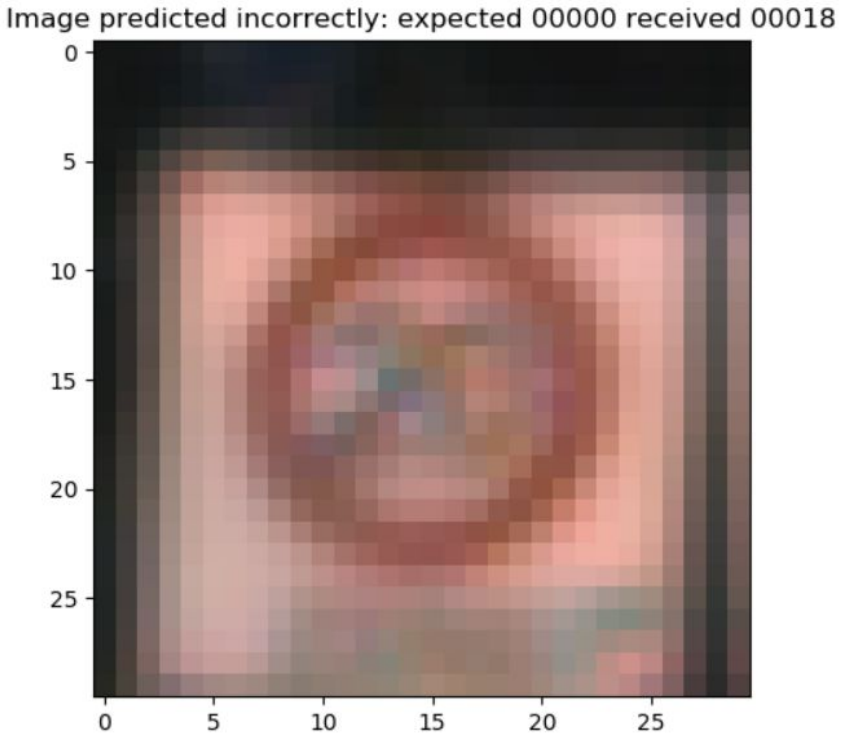
Model was validated using validation set (which is just 20% of initial data set). As a result, with 20 estimators the total accuracy is 0.713154533844189. Total time spent: 199.4316s.

Also, the **recall** and **precision** of each class separately:





Obviously, some images were misclassified. In this model such example is the following image:



Experiments

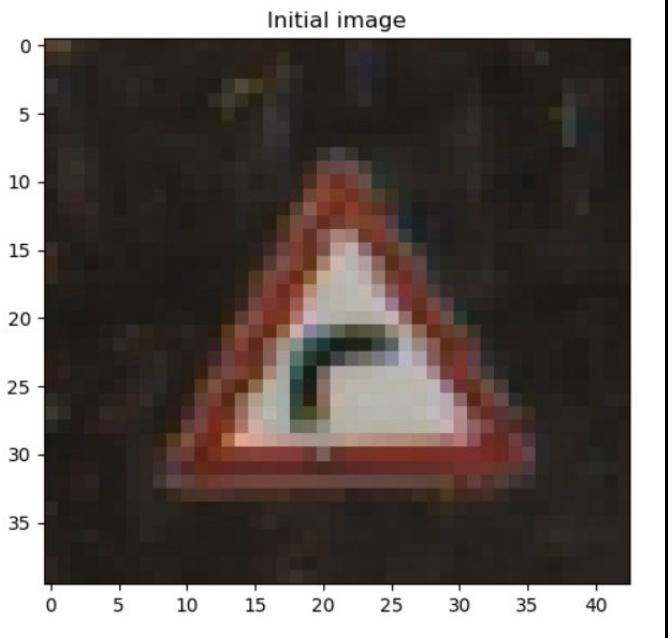
The model described above used augmentation technique, evaluated using validation set on images of 30x30. The program also provides experiments with test data for following cases:

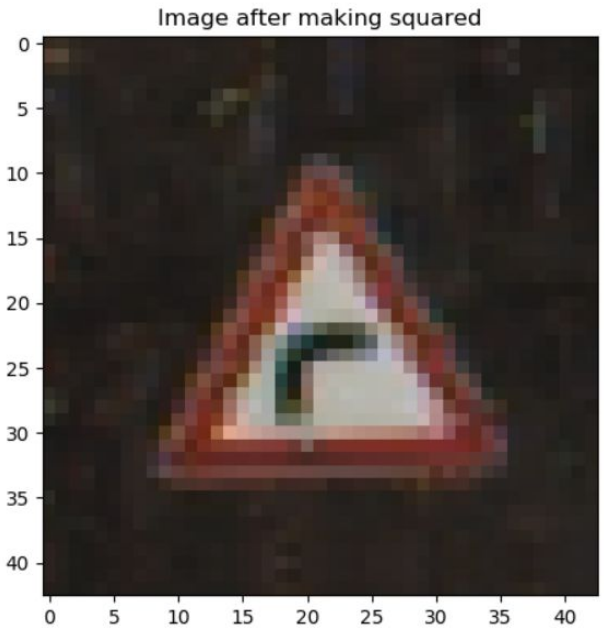
- 30x30 images, without augmentation
- 30x30 images, with augmentation
- 35x35 images
- 40x40 images
- 45x45 images
- 50x50images

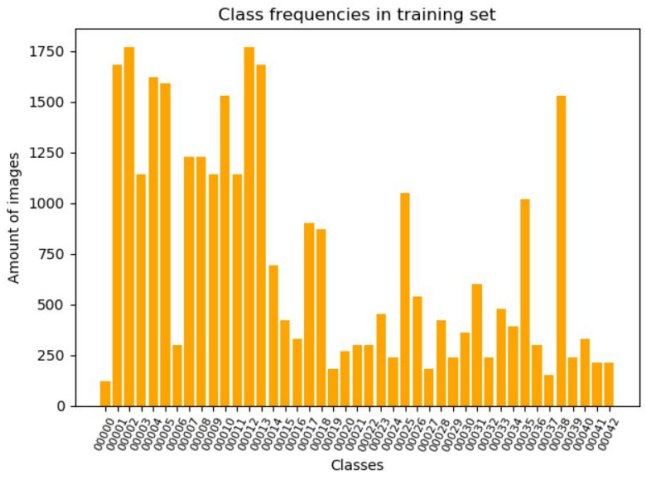
After all experiments, the graphs of time and accuracy depending on image size are plotted.

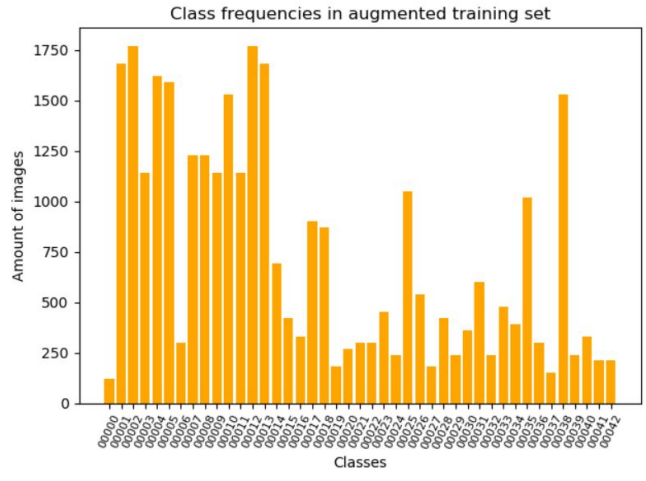
30x30, without augmentation, test set

- Model's accuracy: 0.6809184481393508.
- Total time spent: 119.8809s.
- Without augmentation the accuracy is least possible. Moreover, the accuracy is less than in both models using images of the same 30x30 size (using validation and test sets for evaluation). By received result, we can see that augmentation is necessary to use technique for achieving good model accuracy. Carefully chosen augmentation technique may critically influence on model's behaviour, because some of them can cause to more realistic cases, but others - not. Augmentation techniques chosen according to the situation, lead to better model accuracy.

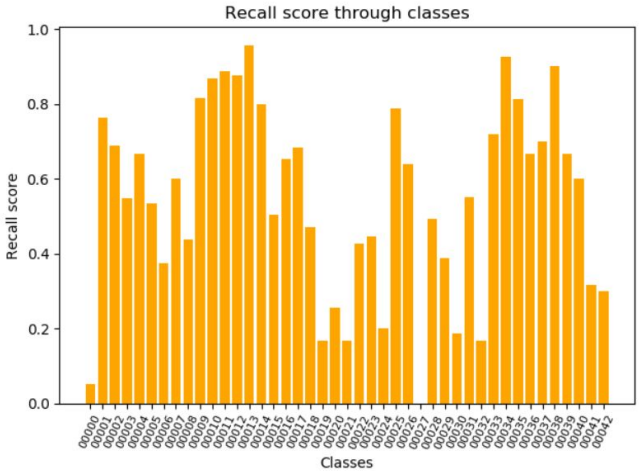


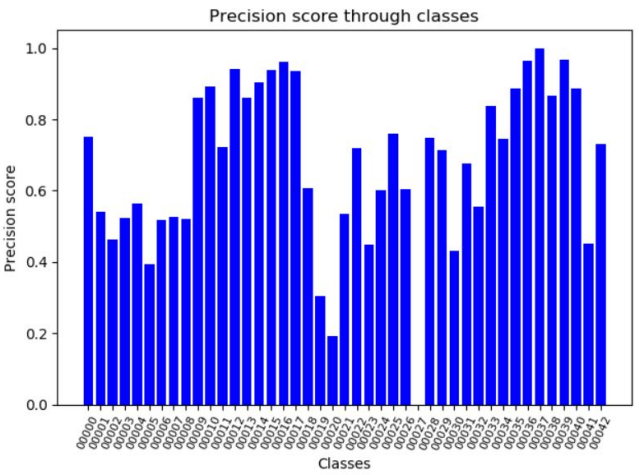


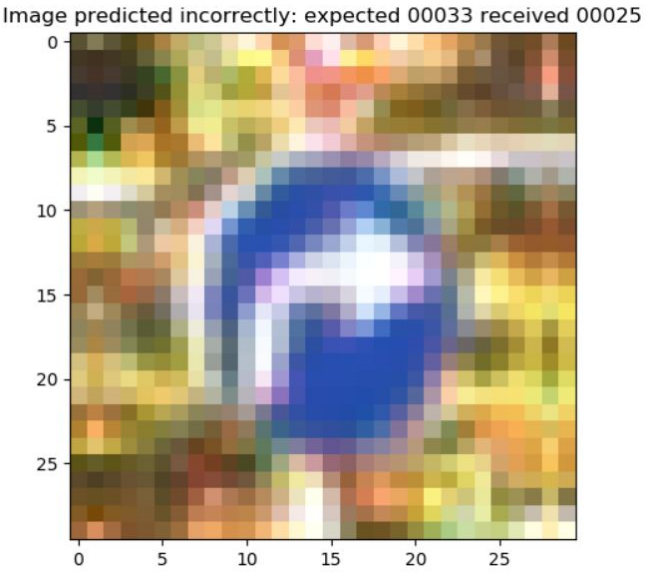




No augmentation at all - the same result

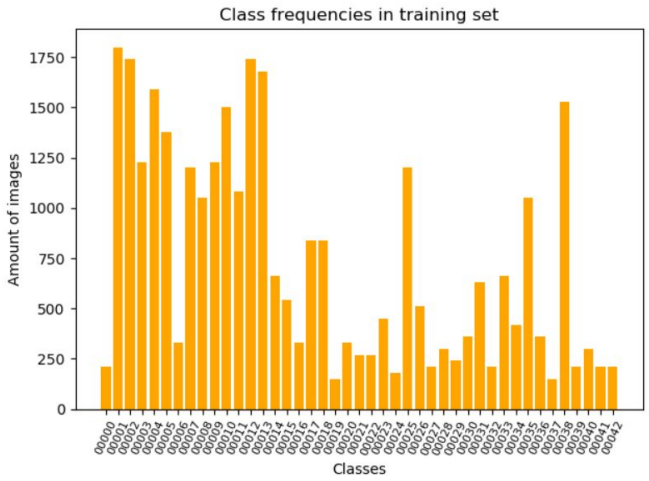






30x30, with augmentation, test set

- Model's accuracy: 0.7056215360253365.
- Total time spent: 192.0004s.


Class frequencies in training set
Amount of images
Classes

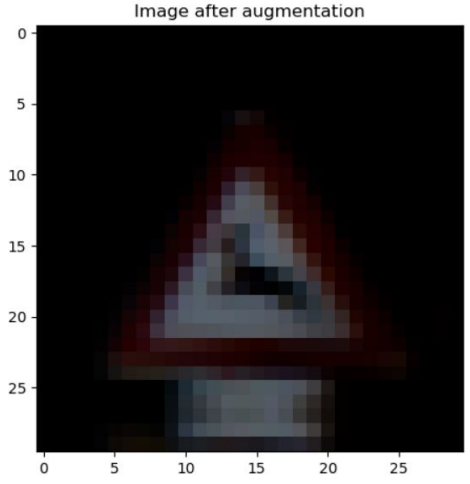
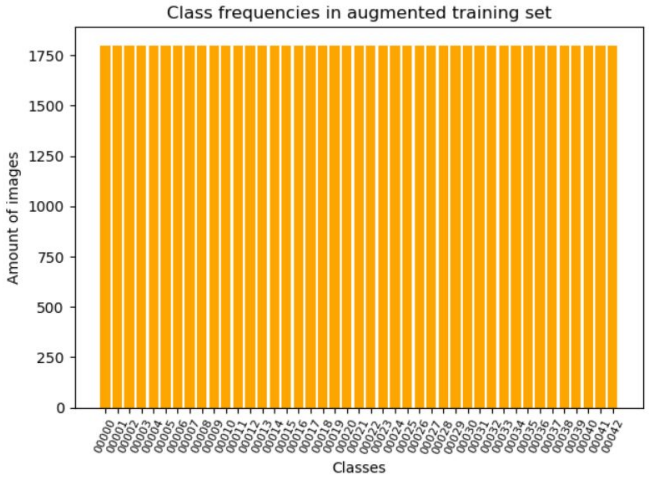


Image after augmentation


Class frequencies in augmented training set
Amount of images
Classes

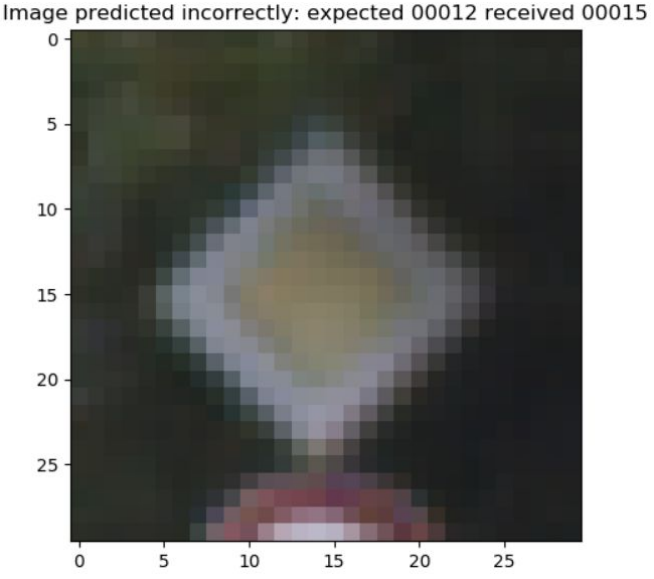
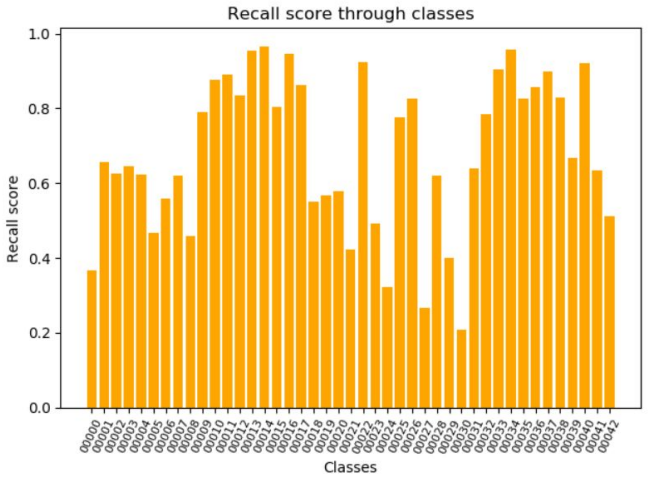
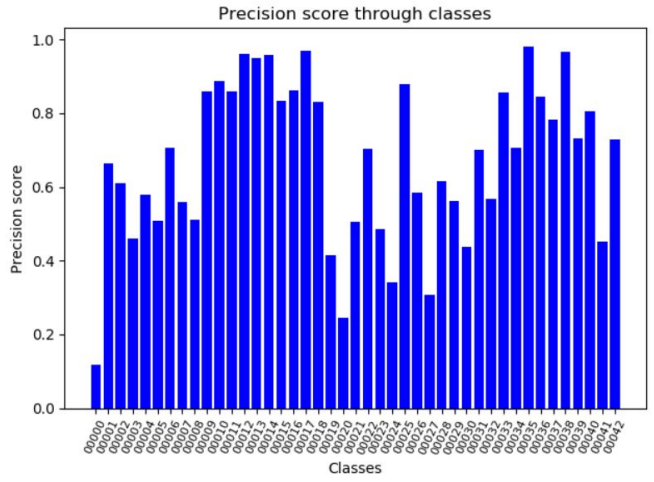


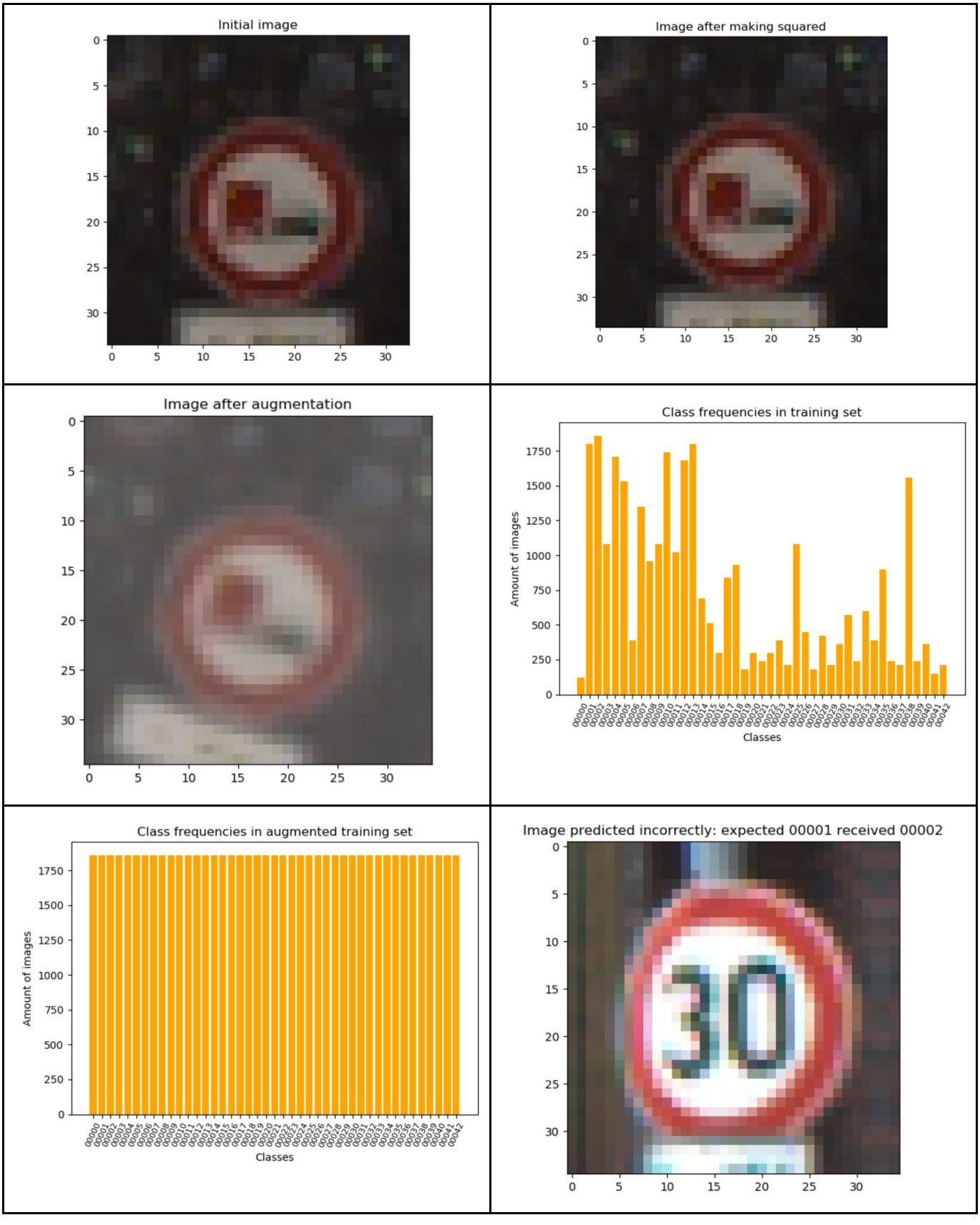
Image predicted incorrectly: expected 00012 received 00015

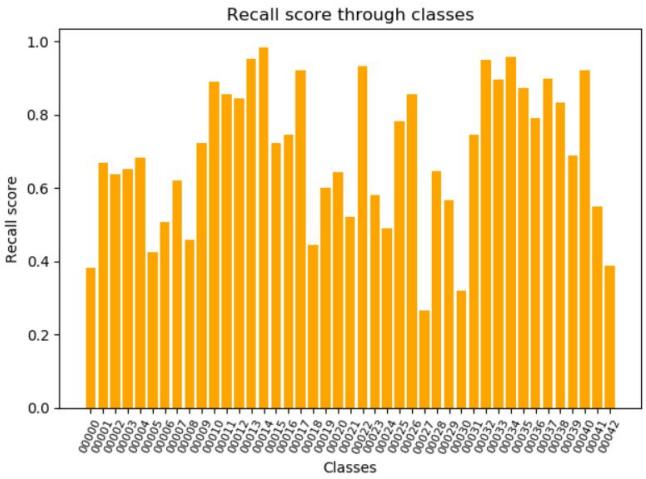

Recall score through classes
Recall score
Classes

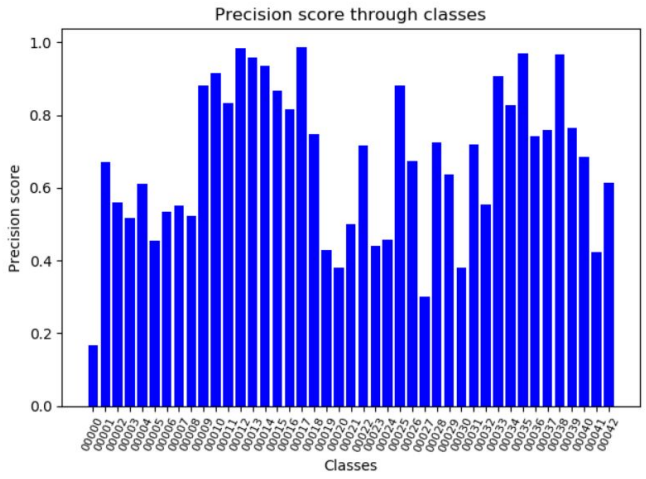

Precision score through classes
Precision score
Classes

35x35, with augmentation, test set

- Model's accuracy: 0.7188440221694379
- Total time spent: 221.9323s.

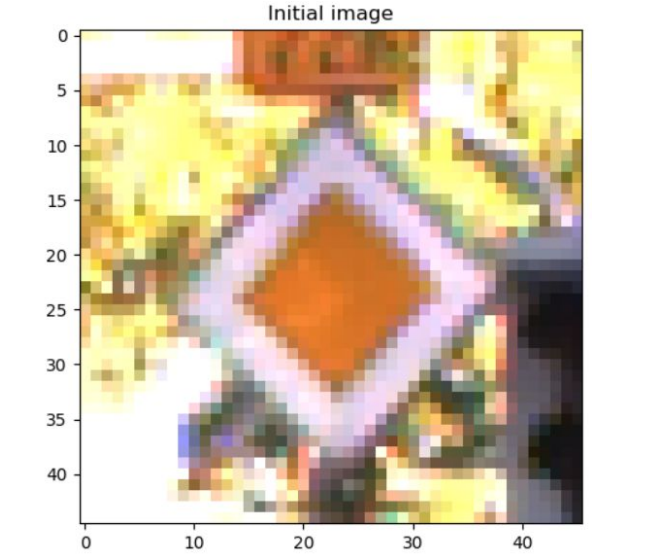


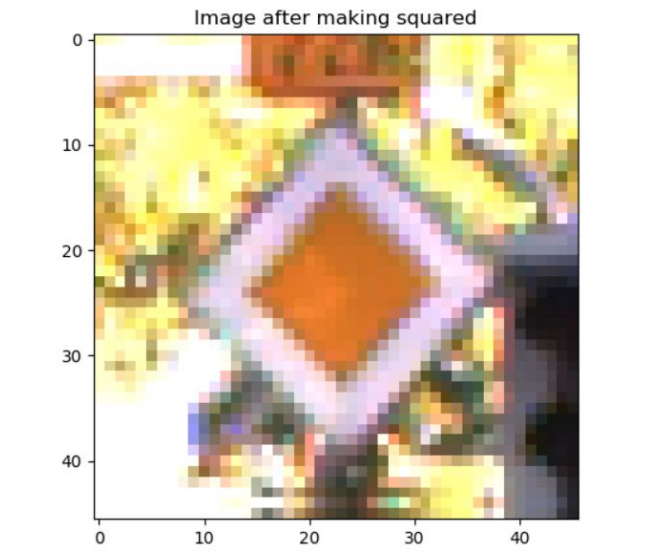


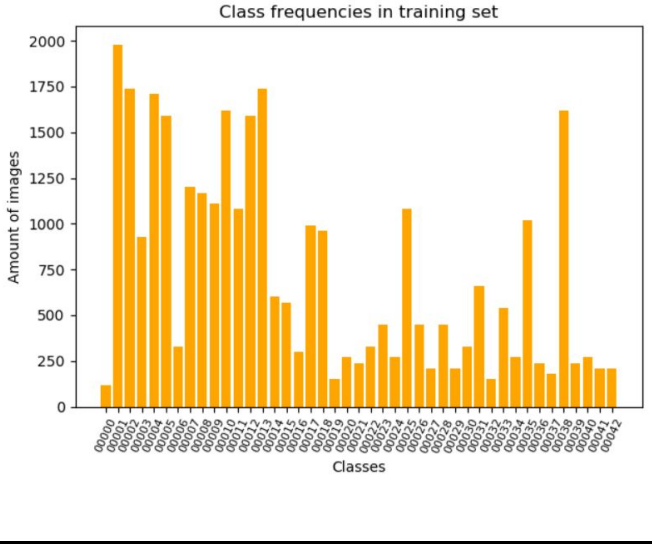


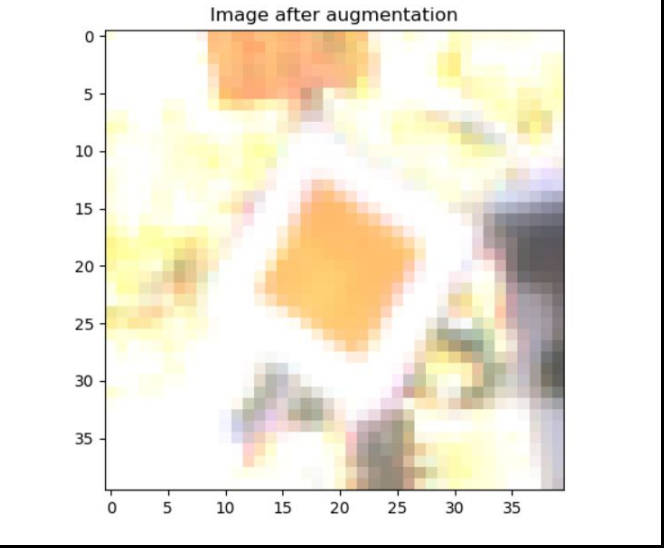
40x40, with augmentation, test set

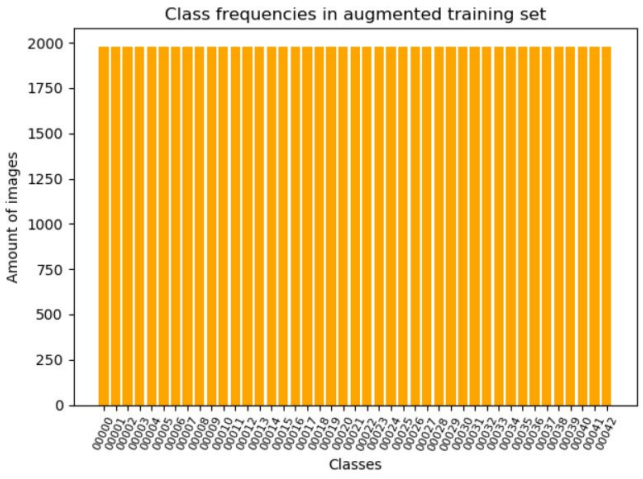
- Model's accuracy: 0.7262866191607285.
- Total time spent: 141.6202s.

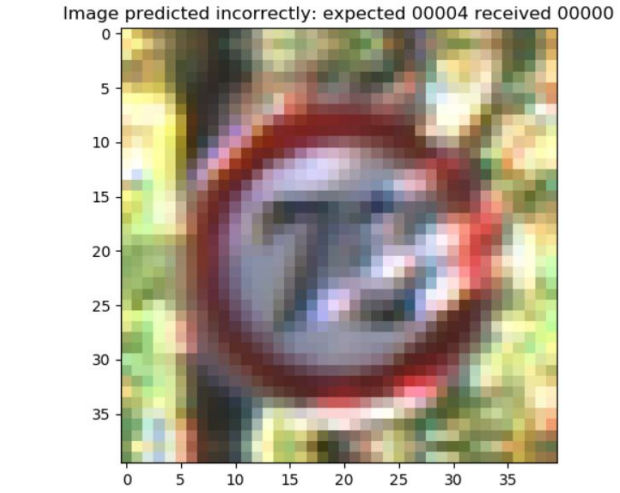


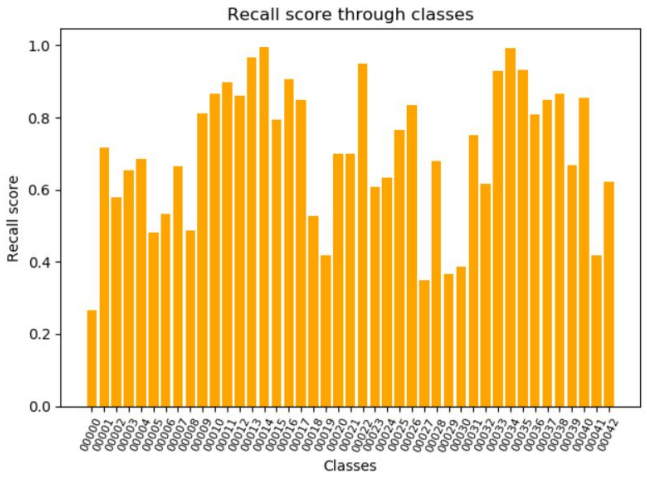


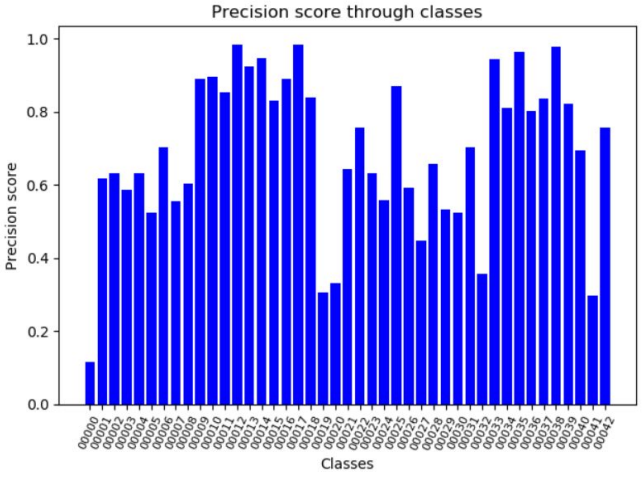






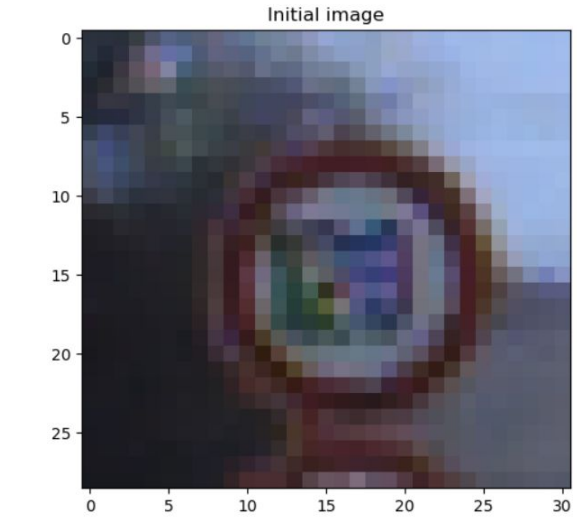


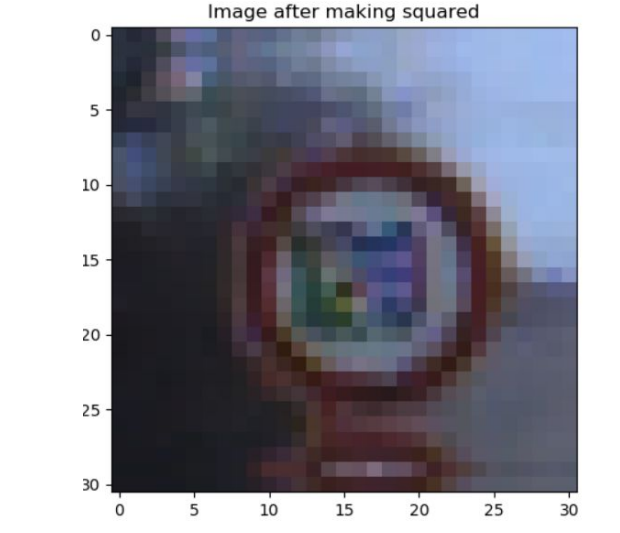


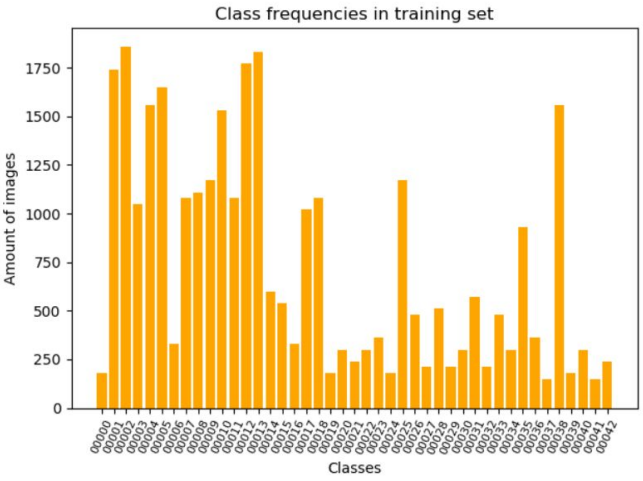


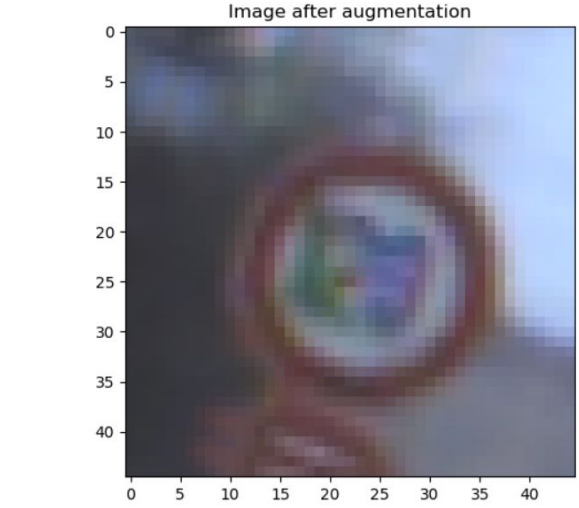
45x45, with augmentation, test set

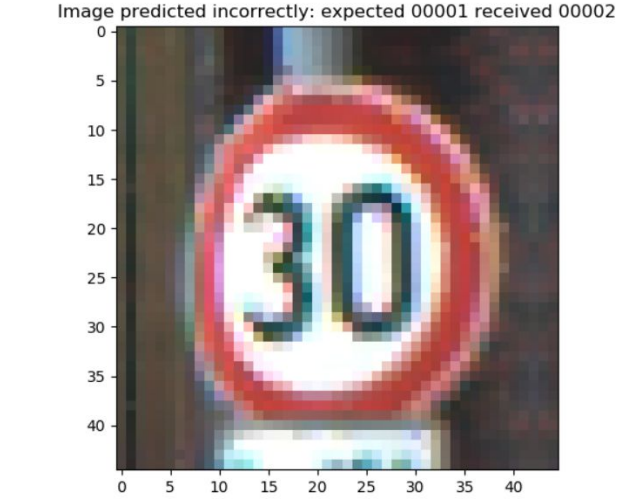
- Model's accuracy: 0.723119556611243.
- Total time spent: 113.1776s.

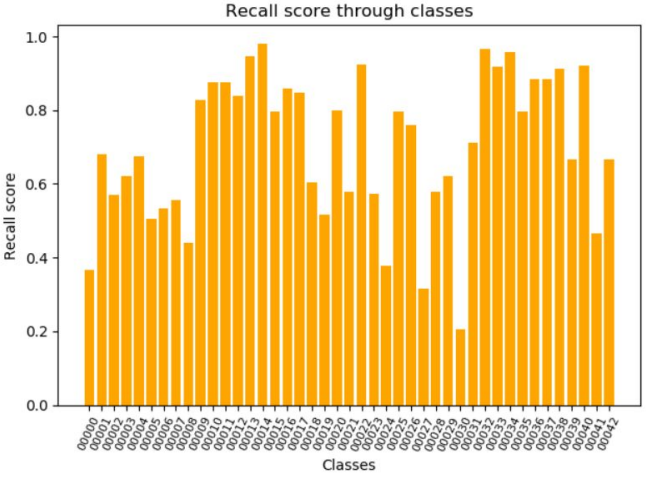


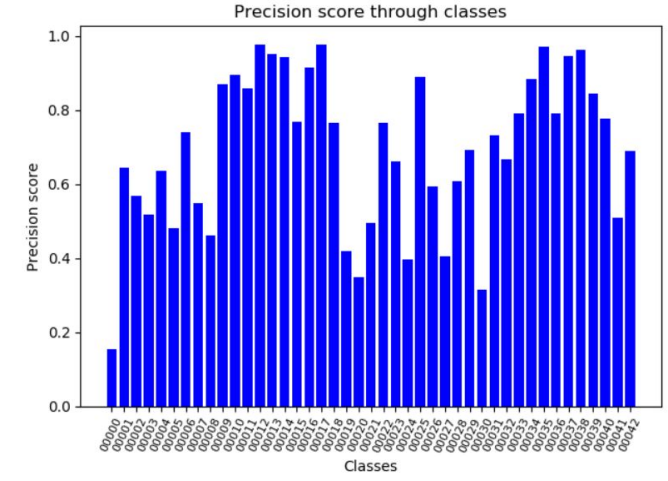






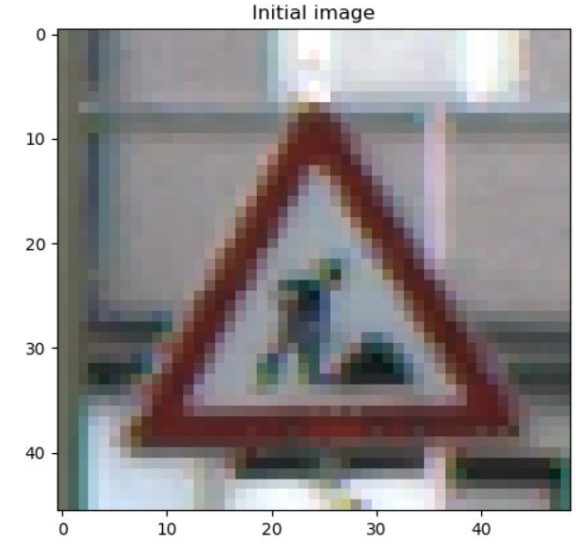






50x50, with augmentation, test set

- Model's accuracy: 0.7424386381631037.
- Total time spent: 171.9705s.


Initial image

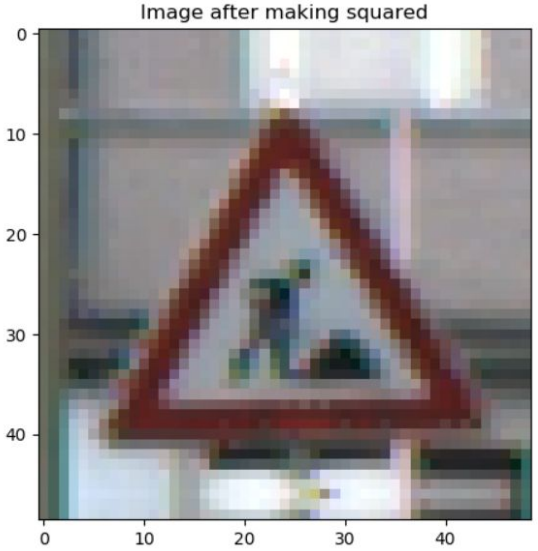
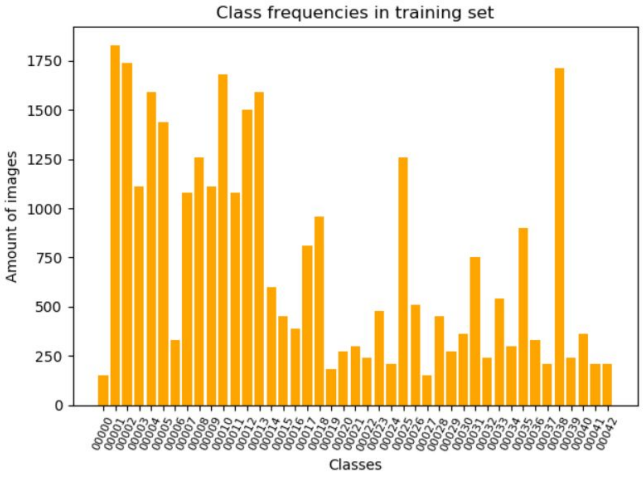


Image after making squared


Class frequencies in training set
Amount of images
Classes

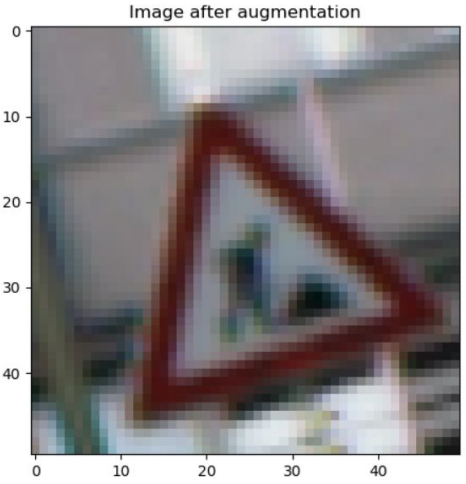


Image after augmentation


Class frequencies in augmented training set
Amount of images
Classes

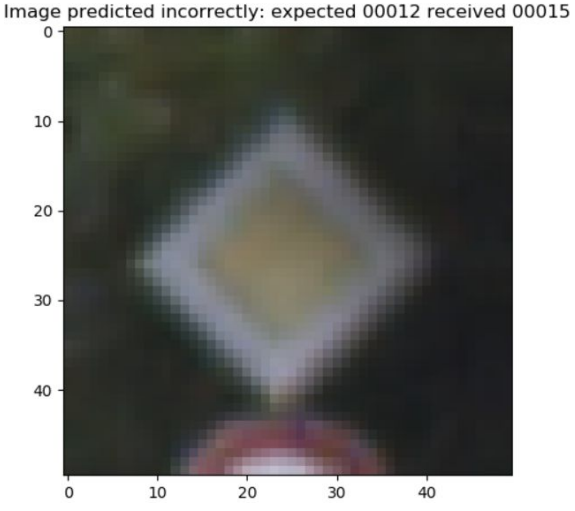
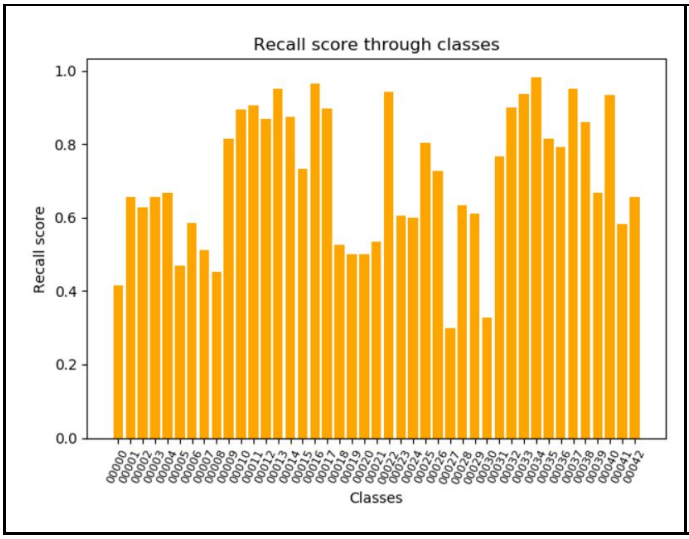
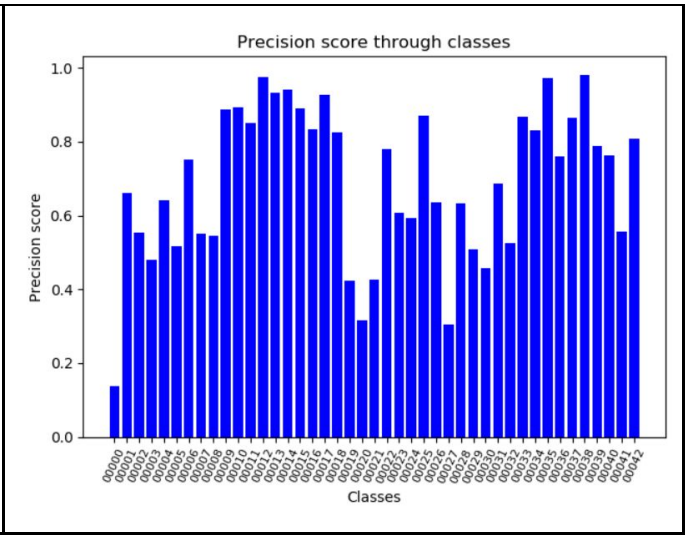


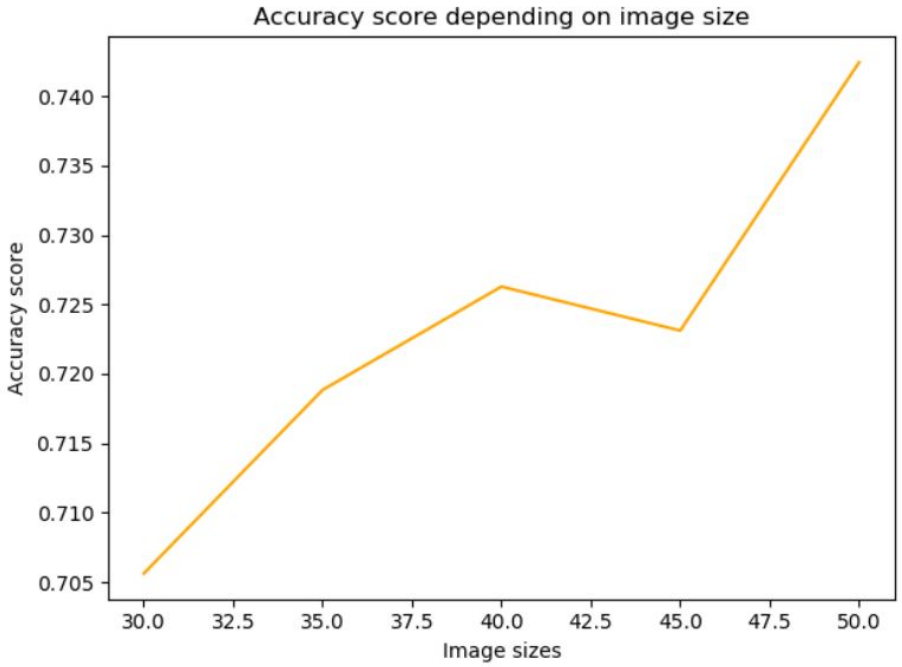
Image predicted incorrectly: expected 00012 received 00015



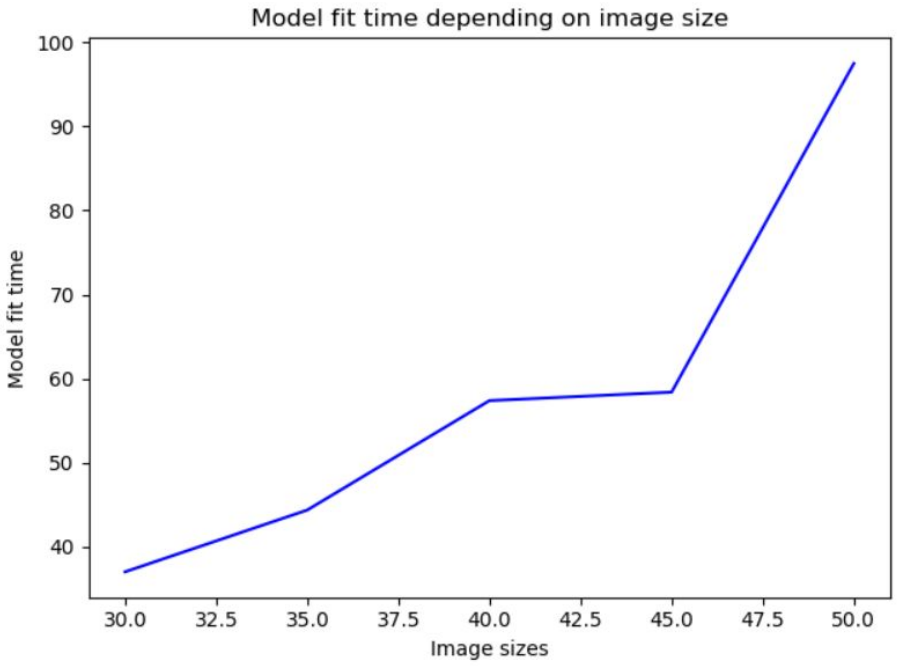


Conclusion

Accuracy score increases over image size growth. This happens because on larger images there's more information to teach from. There are more pixels with information, the image is more accurate, and augmentation is better to use.



By the way, by increasing the image size, we also increase the time for image learning by the model. The graph below represents abrupt time grow, and it's just the time for model fitting. Some other time is also spent on data processing (augmentation, preparation).


Model fit time depending on image size
Model fit time
100
90
80
70
60
50
40
30.0
32.5
35.0
37.5
40.0
42.5
45.0
47.5
50.0
Image sizes