Introduction

This document has been written for being a brief report of the project code realized for a PXL-Vision task assignment. Many details about its realization have been voluntary omitted, so that these pages cannot represent an exhausting and standalone explanation of the realized job.

Formalization of the Problem

The assigned task is about (a 21-classes) classification problem related to traffic signs images. The dataset has been furbished by PXL and it present a "natural" division (2 separate folders) between images collected from roads in Germany (20370 samples) and Belgium (1490 samples). This separation has been maintained for all the experiments. In particular, German images have been used for training and validation of the network, Belgian ones exclusively for testing. This kind of choice has the obvious disadvantage of being the most adversarial one from a performance point of view[1], but, at the same time, is able to reflects a more realistic scenario and so, according to me, it has been considered as a mandatory choice. Any other choice in this sense, would mean partially missing the real capability of the model, and with it, the ultimate objective of any machine learning solution.

In addition, as element of potential variability have been conducted 3 (K) different splitting of German images between training and validation datasets (K-fold) for (partially[2]) measuring the variation of performances moving the these two element and so how much a lucky/unlucky choice of these sets could influence the overall performance of the system on test set.

Then, there's a second additional natural division in the dataset we cannot ignore and it's fundamental for the next analysis. All the images samples contained in both the dataset (German and Belgium) are NOT independent one which other. This could complicate the *real* generalization measurement of the model. The dataset has probably been obtained capturing frames of different video with a certain

[1] It's clear that merging all images together and, after that, operating a separation between validation and test set, could bring to higher classification accuracy. The same would be if we'd use Belgium images directly as a validation set (partially justifying this choice with the absence of any training hyperparameters tunings (like learning rate scheduling for example).

[2] K=3 represent a too small number of iterations for that type of analysis. In each case, it's at the same time important for better communicate the idea of approaching to this kind of problem.

sampling rate. So, inside each folder, there are samples really similar between them, which represent exactly the same signal in the same location, only temporally separated in the original video captured. This mainly bring to two different problems:

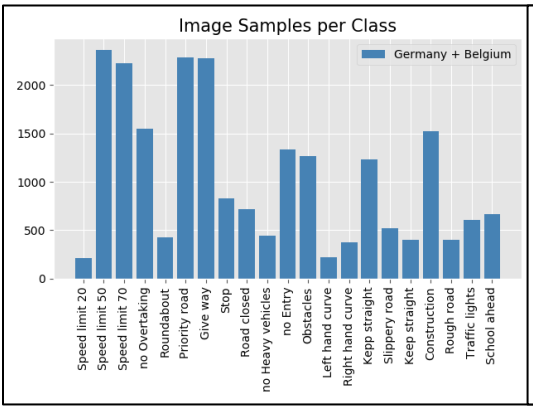
- The samples are NOT *iid* (independent and identically distributed) but just *id*
- There are images of different size, that means we should operating a resizing to a fixed dimension for training them as input unless to face additional problems (for using different input sizes we'd be forced to use a unitary batch size during training, to avoid batch normalization and so on…)

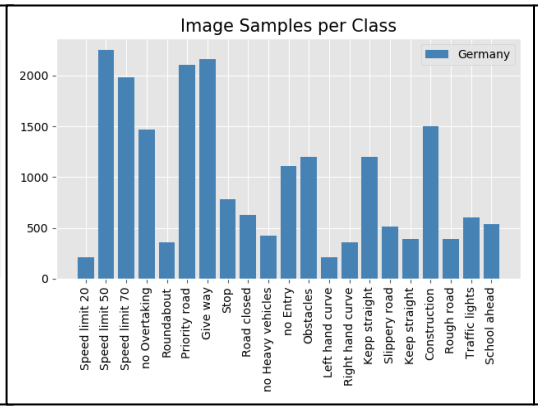
About the first point, the (K) random separations between training and validation have been not based on intra-class *frame* random selection but on intra-class *video* random selection. In particular, the video belonging of each frame can be easily inferred by the suffix name[3]. Doing so, we could consider as *iid* all the frames of different videos. Just like before, even this kind of choice has for sure a negative impact on absolute performances (on validation). On the other side, it should bring to a slightly positive impact on test set performance because of the major challenging of the validation set[4].

By an overview about classes numerosity distribution over samples, the dataset appears as clearly unbalanced.
It's opportune to say that we do not have the original video but just a selection of extracted frames. So, for correcting the data skew, the most natural way in this case would have been extract again all the samples with an adaptive sampling rate, generating, in this way, a perfectly balanced dataset. Indeed, the solution adopted for this kind of problem has been considering a term of penalization during training for most representative class in the data distribution. In particular, it has been assigned a weight for each class according to its numerosity[5]. Others, more complex solutions have not been necessary.

Let's take a look to data the distributions.

[3] A frame name example: 01165_00001. The suffix part (01165) seem to represent the number video, 00001 the number of the frame This is clearly has been inferred just by viewing the images. So, there isn't any official note about that.
[4] This variable has not been tested. On validation this has not brought to any accuracy overhead, like we'll are going to discover soon in these pages, it' reasonable there is been just a late on training convergence of the network (a parity of all other factors).
[5] **class_weight** argument in keras functuion model.fit()





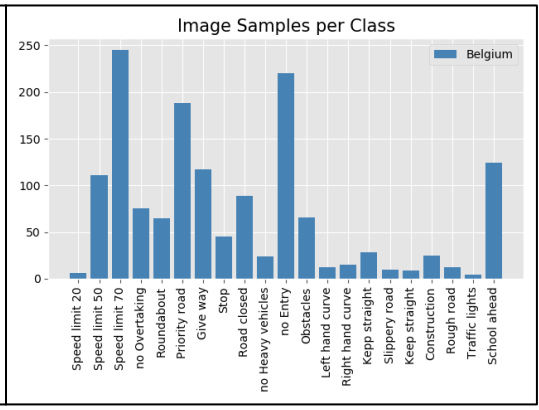
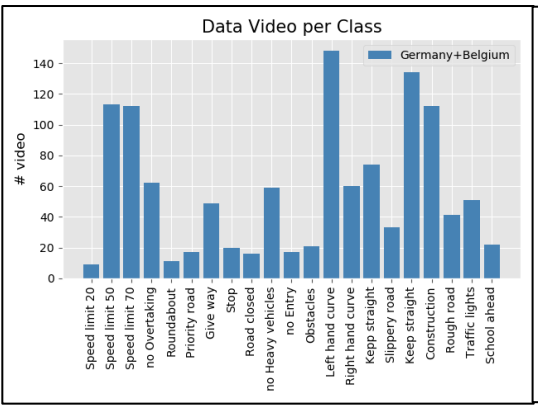


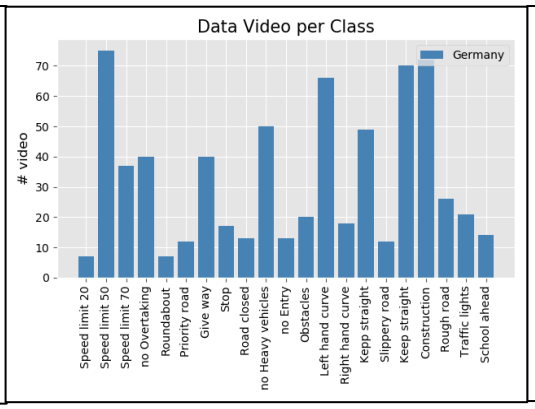
Image samples per class distribution:

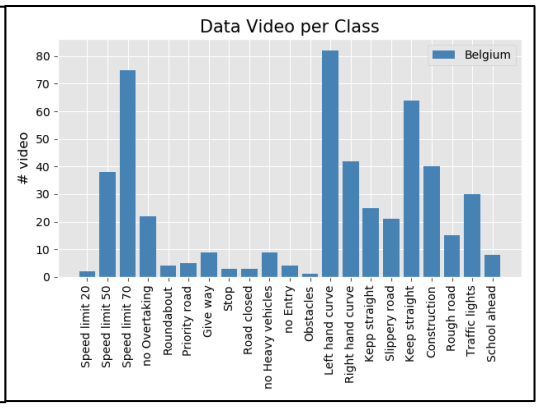
Min. value: 216
Max value: 2361
Mean value: 1041
Std value: 723

Min. value: 210
Max value: 2250
Mean value: 970
Std value: 675

Min. value: 4
Max value: 245
Mean value: 71
Std value: 71







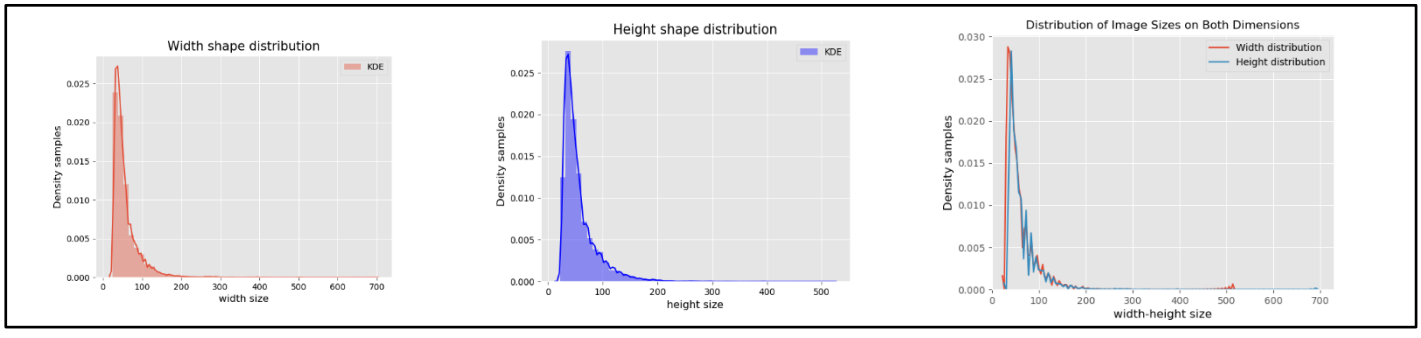
Data video samples per class distribution:

Min. value: 9
Max value: 148
Mean value: 56
Std value: 42

Min. value: 7
Max value: 75
Mean value: 32
Std value: 22

Min. value: 1
Max value: 82
Mean value: 24
Std value: 24

The dataset presents another issue. Images have really different sizes. So, it's even important trying to understand the distribution of the database according to the dimensions of the single images (both on width and height).



Width/Height dimensions samples distribution:

Min. width value: 22 (3 samples) - Min. width value: 517 (1 sample) - Mean. width value: 57 - Std. width value: 33
Min. height value: 25 (32 samples) - Max. height value: 695 (1 sample) - Mean. width value: 57 - Std. width value: 36
Mode width value: 35 - Mode height value: 33

By above figures we clearly infer two things:

- Both the two variables (width and height) have approximately the same values, so that all the images have a quite regular square shape, meanly differing for just a few pixels.

- The width/height samples distribution has a density peak (KDE plots) at around 50/60 pixels for both the variables (both the modes are about at 35 pixels instead).

*By this overview seem to be fair choosing as standard input size of network a resized **64**x**64** pixel image.*

At the end, as preprocessing, I've used an adaptive histogram equalization (CLAHE) for increasing the sharpness of edges and apparently solve the greater part of over/under brightness. I've not evidence of the impact of this transformation, because it has been applied in each training session.

The Model(s).

The key point of the task is for sure the creation of the model for classification. More in detail, it's more correct to speak in the plural form. There's a double requirement about this point, in fact. Firstly, trying to get the maximum numerical performance. Secondly, designing a new more efficient model solution for the portability on potential low power calculations hardware. For this reason, we are going to (really briefly) introduce two different neural network architectures. In a next section of this document will directly compare performances between the two models and computational power requirements.

Resnet VS Resnet.

The deeper network I used for maximum performances is a classical *Resnet50* pretrained on *Imagenet* with 4 additional layers:

- A *fully-connected* layer with 1024 neurons
- A *dropout* layer with probability of 0.5
- A *fully-connected* layer with 512 neurons
- A *dropout* layer with probability of 0.5

The second (more efficient) model is another a residual architecture, directly inspired by the first one. The idea has been not just using another, less deep, *Imagenet* pretrained model, like MobileNet, or making a too shallow architecture or even trying to pruning/quantize the first model. I've indeed tried to customize the first one, maintaining approximately the same depth but with, at the same time, an exponential reducing of the Resnet 50 huge number of parameters and GigaFLOPs required for a single inference. Just in a few words, principal changes introduced in new architecture (Resnet_custom) in comparison to Resnet 50 are the following:

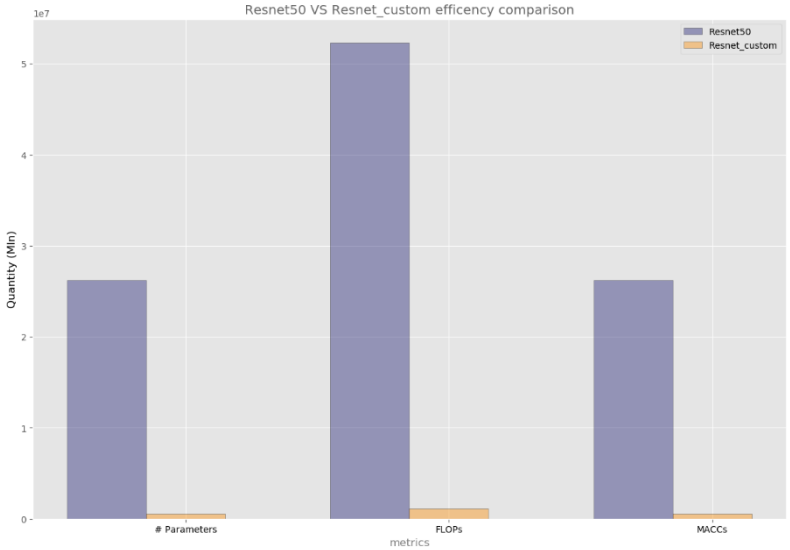
- First convolutional layer in traditional Resnet has a kernel size of 7x7 with a stride of 2 (originally designed for 224x224x3 input size). I changed it to 3x3 with a stride factor of 1.
- I removed the first max-pooling layer after the previous convolution

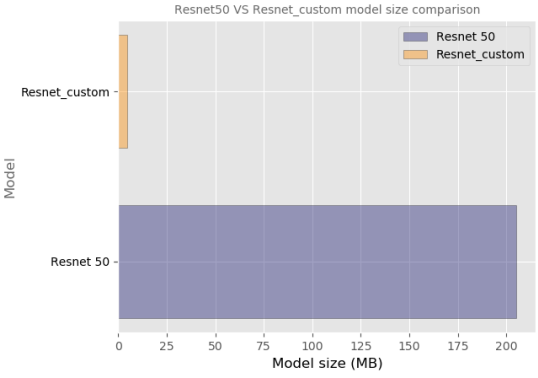
- I changed the number of filters used. Resnet50 starts with 64, the new one starts with 16.
- Resnet50 has divided in 4 stages. I've removed the last one, that is even the smallest one containing only 3 blocks (1 conv block and two identity ones). At the same, being the last stage, it involves the major number of filters (between 512 and 2048). This cut has "only" saved 9 layers on network depth and halved the number of parameters. In addition, it has had a great impact on reducing gradient vanishing during backpropagation phase and accelerating training convergence. So, the total numbers of layers for this Resnet-custom is exactly 41.
- I've added a *dropout* as last layer just before the output one. I've not added any *fully-connected* layers like in previous architecture.
- 1-Channel (grayscale) image as input: negligible impact on number of parameters (only on first convolutional layer), deeper impact on number of operations
- No pretraining (but this is not a bonus point)

Model(s) efficiency: images are worth one thousand words

Let's start to go deep in task realization approaching, for now, a model comparison just based on efficiency, we'll instead compare performances in next section.

There's not much to say about this, the efficiency saving related to the custom Resnet is huge. Let's take a look to the next two figures.





Efficiency metrics detail values:

Resnet 50	Number of parameters: 26.22 Mln. - FLOPs: 52.28 Mln. - MACCs: 26.19 Mln. - Model size: 205 Mb.
Resnet custom	Number of parameters: 556 K. - FLOPs: 1090 Mln. - MACCs: 530 K. - Model size: 4.7 Mb.

For a complete analysis we should consider RAM occupancy during inference[6] and average time for a single prediction on a same machine. This point has not been investigated in this work.

Model(s) Performances: Free lunch?

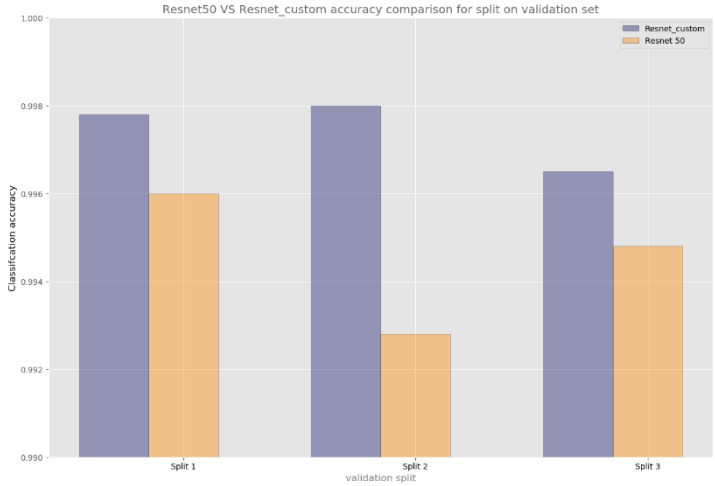
At now, we know we can just say to be able to save a huge quantity of resources selecting the customized model, but we do not know at which price in terms of classification performances. We're finally ready to discover if, for this specific task, we have just gained (or not) a free-lunch for today.

As anticipated, I've used 3 different splits between training and validation and trained both the models 3 times (K-folder analysis, K=3)[7].

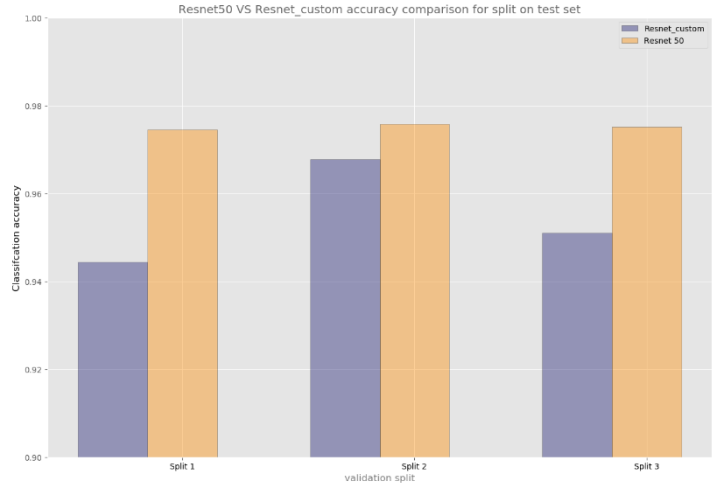
For all the splits, on **validation**, both the models have reached stunning levels of *accuracy*, so close to 100%, not showing any performance gap between them. There's been, instead, a clear gap in the training time required for convergence, so that Resnet 50 has been able to reach its maximum level of accuracy in about 10/15 epochs, the smaller model in 50/70 epochs. The deeper model has been trained in each split for 50 epochs and a fixed learning rate of $1e^{-4}$, the other one for 100 epochs and $1e^{-3}$ as learning rate. SGD has been used for both, with categorical cross-entropy as cost function. Bath size settled to 16.

Here the results obtained, in terms of accuracy, on validation set:

[6] Model size value, expresses is Mb, is just able to capture the RAM memory occupancy on model loading.
[7] Train and validation splitting have been done on videos reference, not on singles images as already said. Textual files containing relative path of images have been uploaded on Github repository.



Everything looks good in this barplot, we incredibly have more accuracy on the custom model. Looking now at results on testing set the situation appears as follow:



The takeaway by this brief comparison should be the following:

- On test set accuracy scores appear as to be upset. Now the deeper model performs better on each split showing so, a greater generalization capability.
- Resnet 50, on test set, show an incredible low variance about accuracy score, almost performing equally on all the splits. The same is not true for Resnet custom, that appear to be more sensible to the choice of training set.
- No free-lunches for today!

- More in general, the situation, for now, does not look so dramatic, even considering the efficiency saving of the smaller model.
- We are just looking to overall accuracy in these graphs, we must to go deep with this analysis taking into consideration the confidence score of each prediction and trying to evaluate the two models moving from the native **argmax** thresholding on values returned by softmax activation function. In other words, it's important to check **if** and **how** this accuracy numbers would change applying no more the classical threshold of $\left(\frac{1}{num\ of\ classes}\right)$ but an higher one for measuring how much confident has been the model for producing its predictions.
- The comparison analysis must take into consideration the other canonical metrics, i.e. *precision*, *recall*, *F1-score*
- It's important to take in mind acting these comparisons we're moving at the same time two factors of the networks, the *model complexity* and the *pretrained state*. So, it's possible the gap showed is duty (almost in part) to the random initialization of network and not (at all) to the different architecture. This would even explain the performance variance revealed (only) by custom Resnet on the different splits. So, for practically proving this, we should train a Resnet 50 without loading Imagenet weights for all the same splits and verify the presence of the gap here showed with the smaller network.

  At now, the remining time for completing this task is not enough, so no real evidence of what I've said is contained in these pages. Anyway, I'm personally quite confident about the validity of this hypothesis.

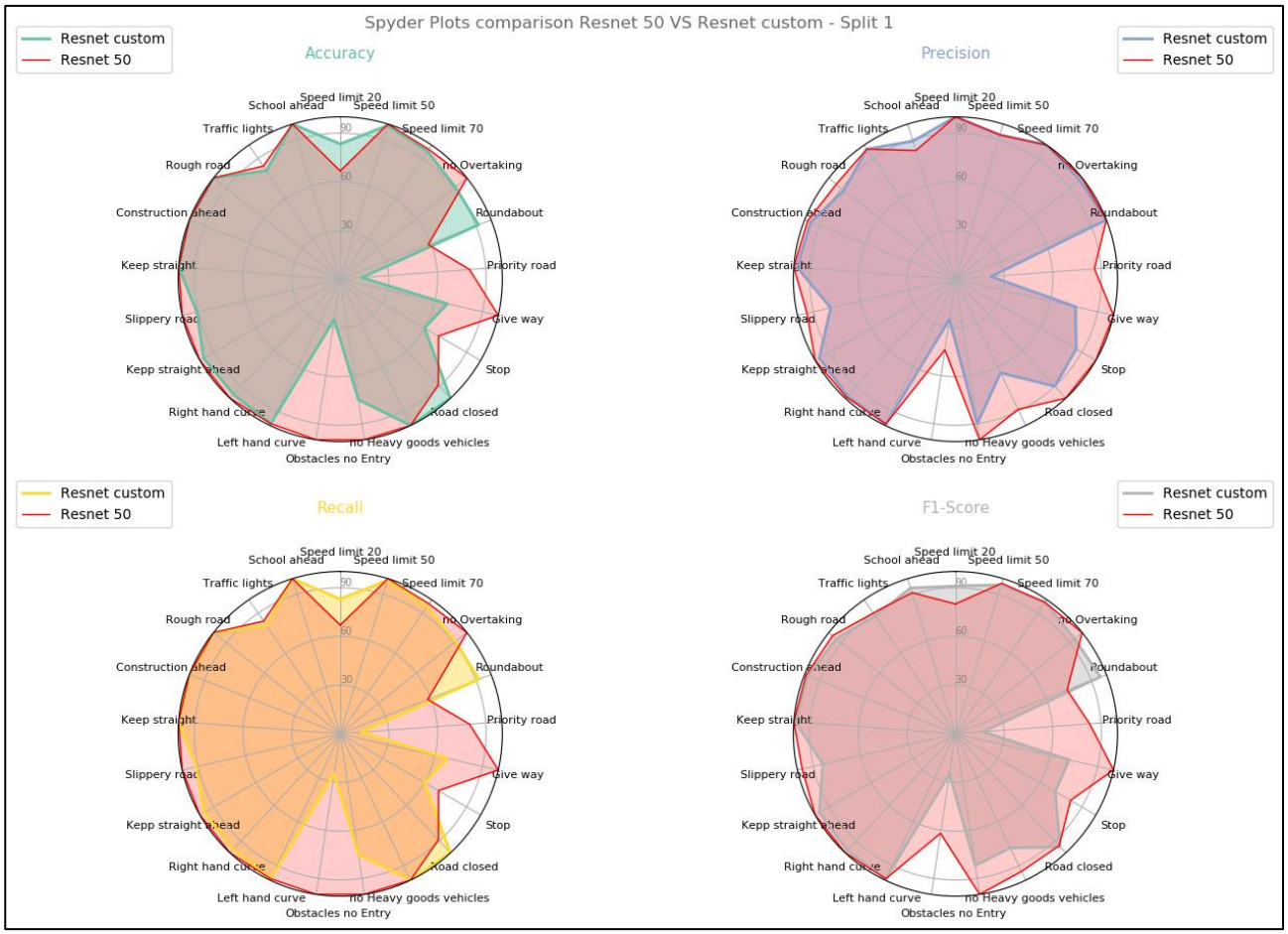
Beyond accuracy.

As anticipated, in this final section we'll going to take into consideration other two points, i.e. the introduction of other well-known metrics other accuracy and the applying of several higher thresholding values on the probability vector returned by the softmax function in the output layer.

The overall objective of this last section is trying to validate the performance results numbers obtained until now and the quite good trade-off between efficiency and performances.

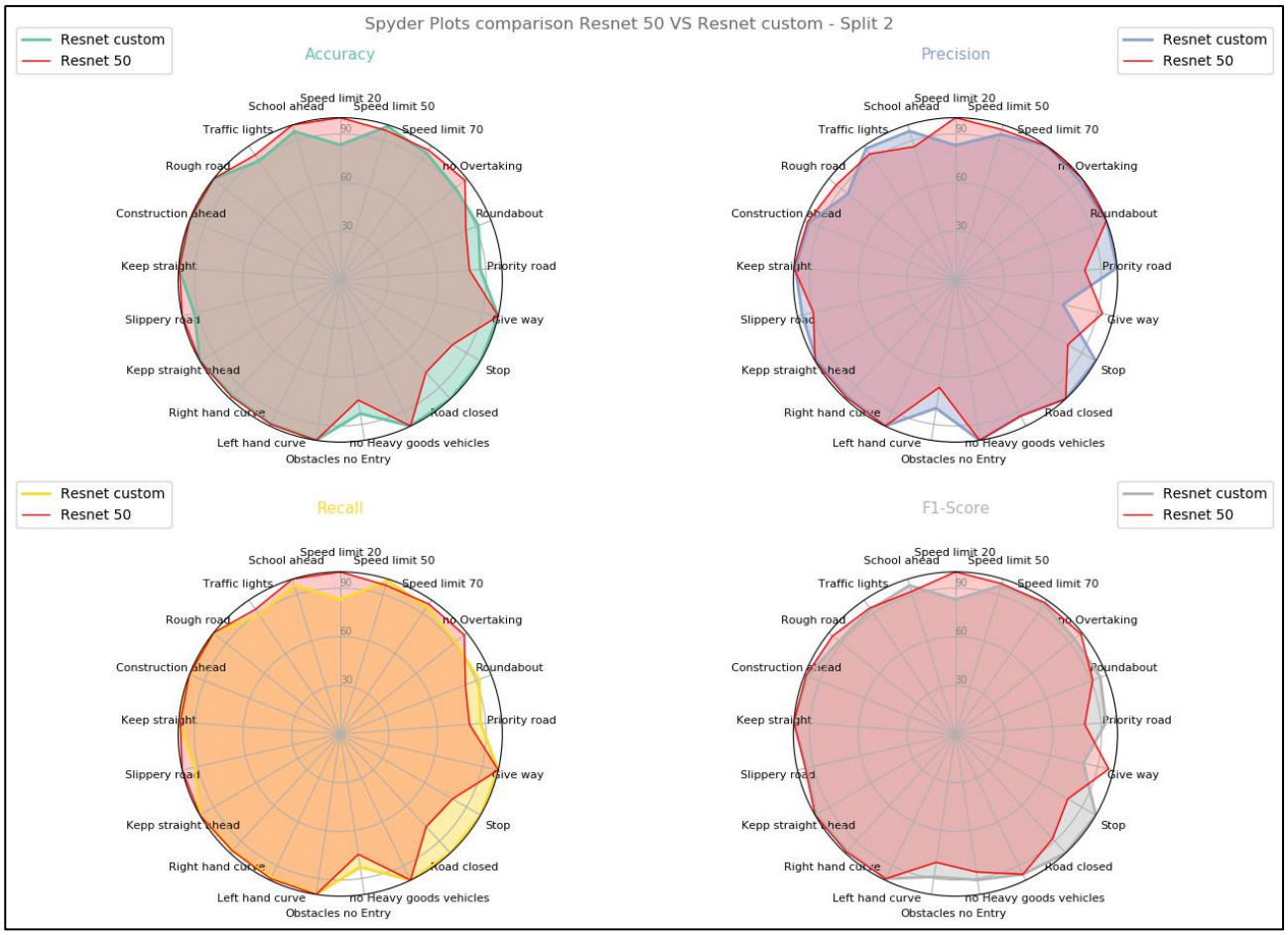
About metrics, the situation is visually summarized by the 3 following spyder plots, divided, here again, for each split.

As extreme synthesis, data confirms Resnet 50 performs generally better if compared with Resnet_custom, especially on split 1. On split 2, the choice of the customized Resnet would seem to be as globally preferable. On split 3, instead, there is no gap between the two models[8].

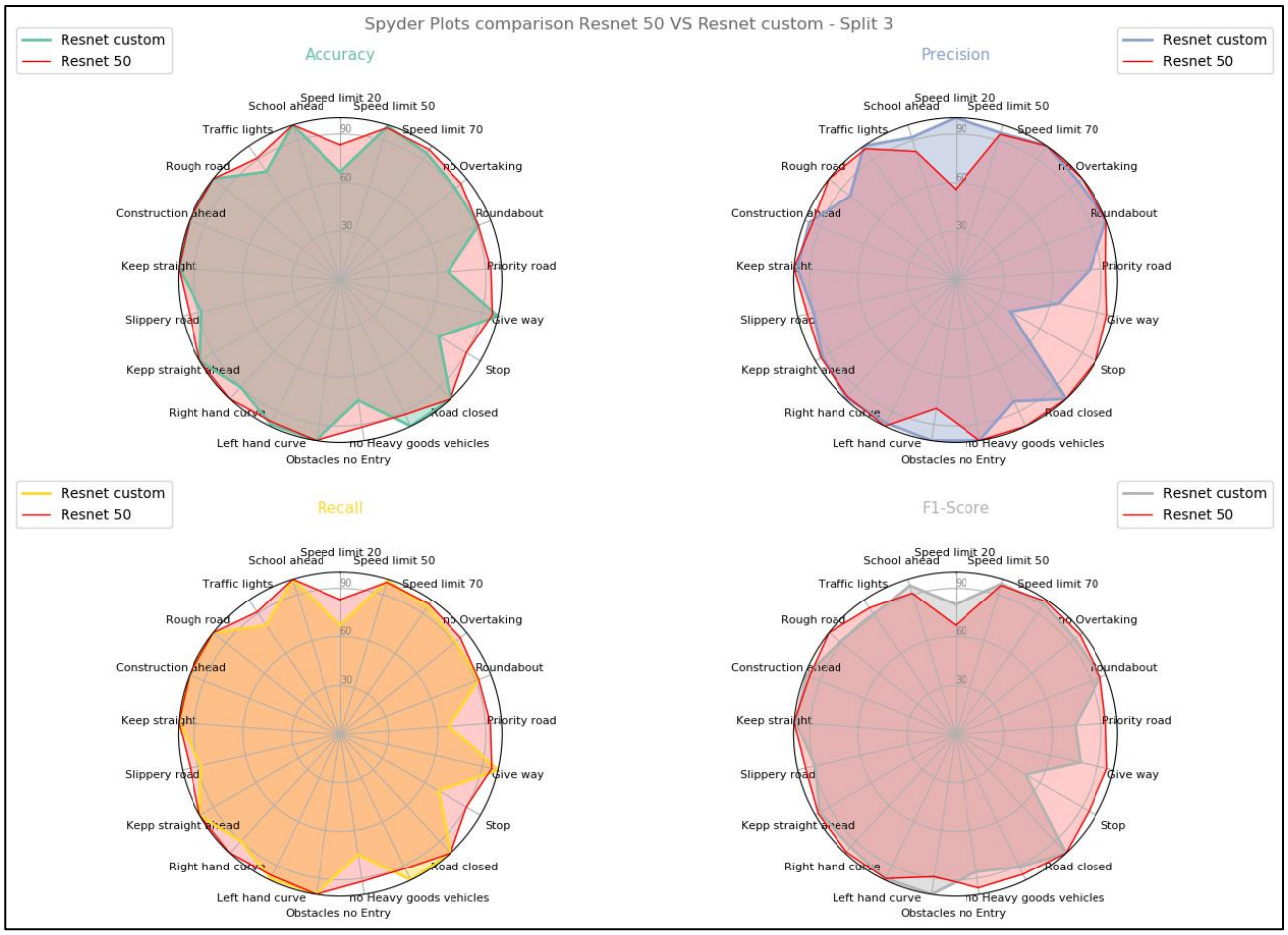


Spyder plot about performance comparison split 1

[8] For the sake of honesty, all these statements should be validated with specific statistics tests.



Spyder plot about performance comparison split 2

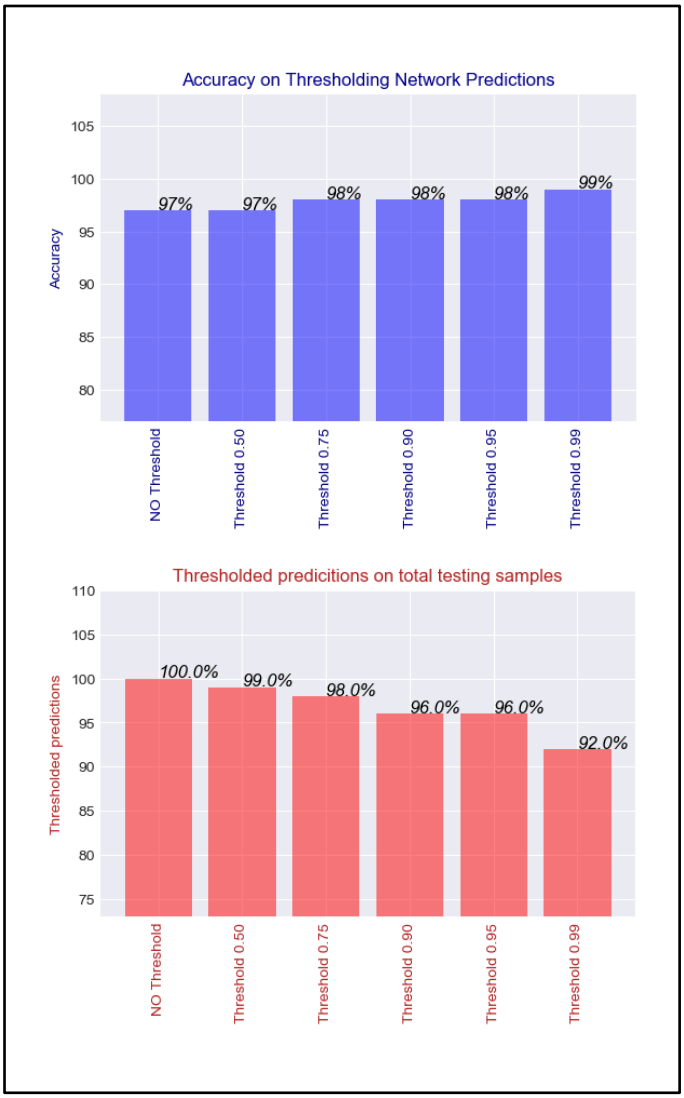


Spyder plot about performance comparison split 3

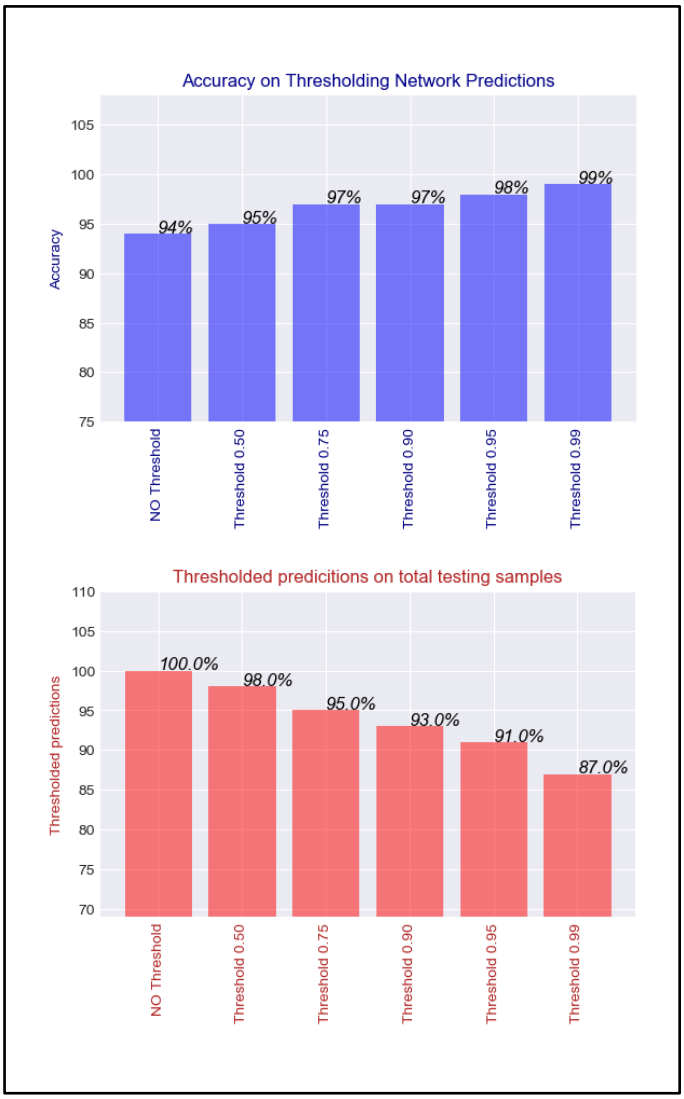
About thresholding, the overall idea is trying to infer how lucky the network has been on the prediction about test set. This is even, as said, mainly the reason for which I've done K training and K different performance evaluation.

More in detail, I'm going to apply **5** different level of thresholding, *0.5, 0.75, 0.9, 0.95, 0.99* to the output immediately returned by the network, so that, instead of just predicting the class with greater probability score (that in this case would be at least about 5%), I'm going to accept the class prediction of the network only in case of score greater than the established value, otherwise the sample will not be classified. This is in practice, a wide way to analyse the loss on test set predictions. Network could produce a great loss for each prediction but being able in any case to correctly classify the sample. A network that has been really "lucky" in its predictions, increasing the threshold value, will be able classify only a few numbers of samples, because all the other one has been predicted with a too low confidence score. So, it's possible the deeper network would be able to predict, in average, with a

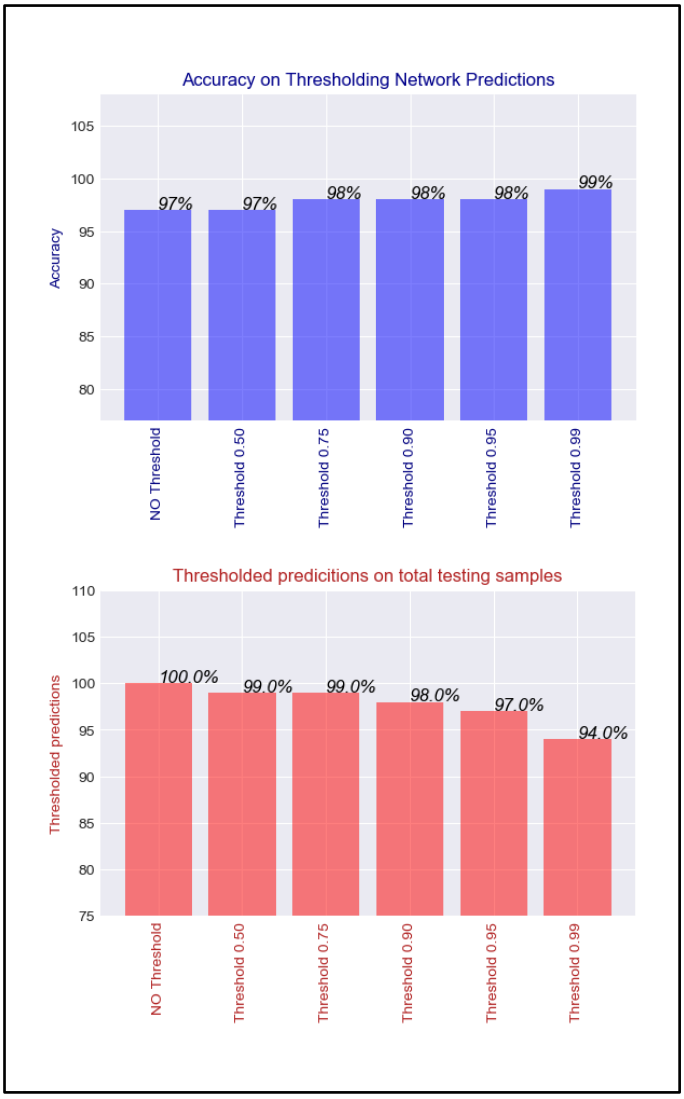
consistently greater score respect to the smaller one and so, this kind of analysis would bring to a stretching of the measure performances we've seen until now revealing new scenarios.



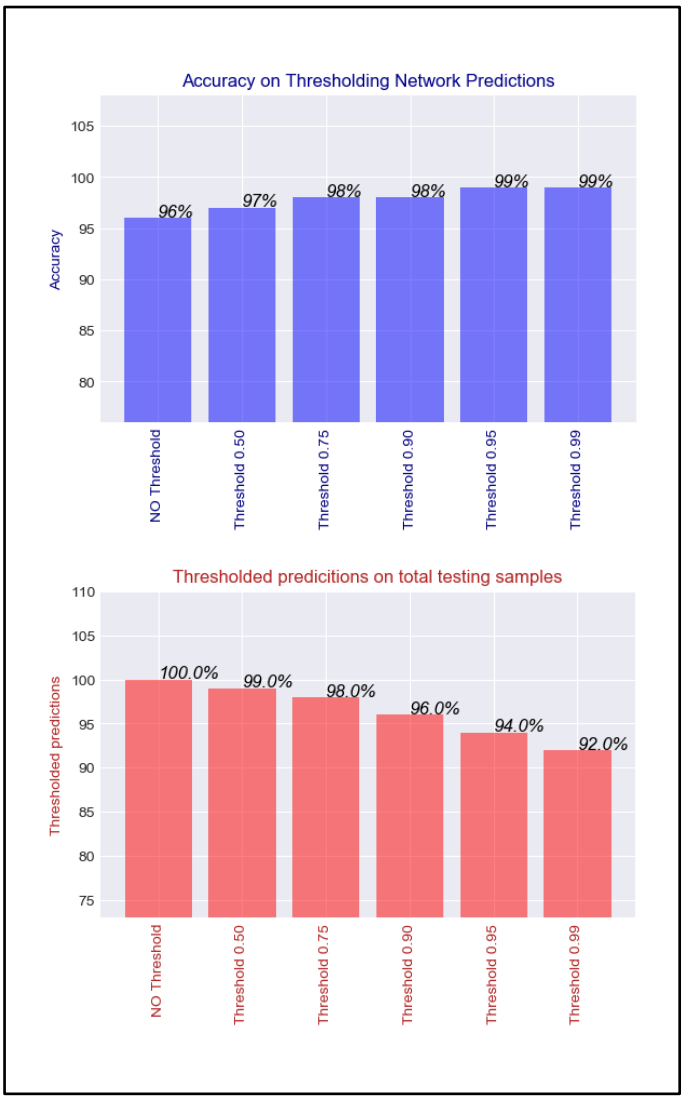
Resnet 50 – split 1



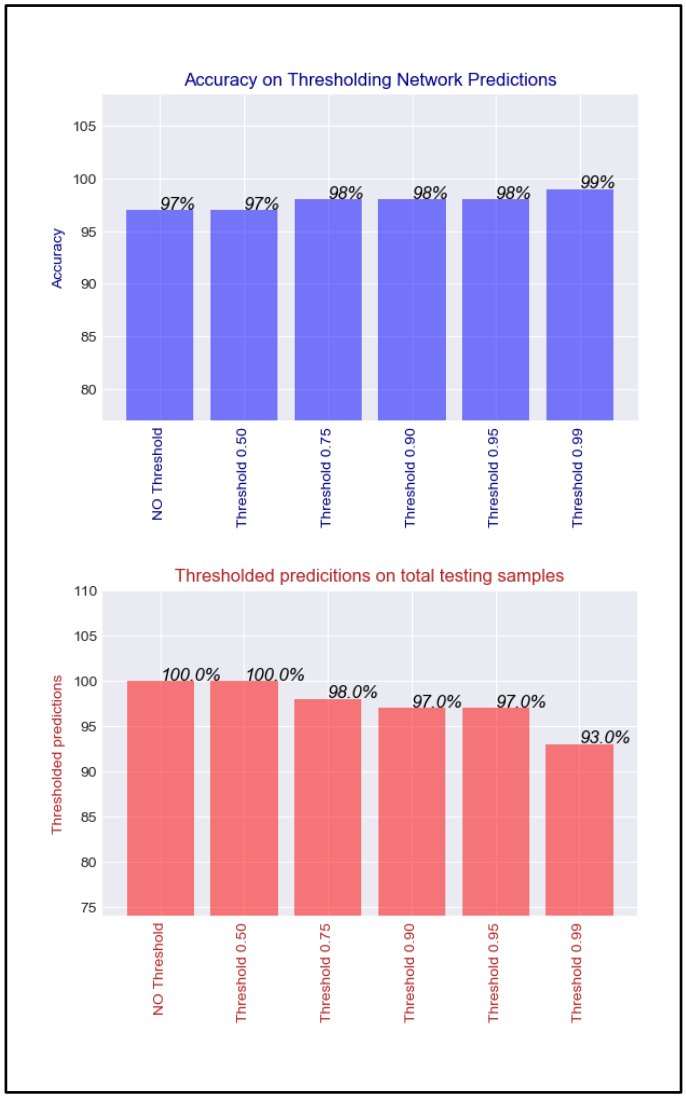
Resnet custom – split 1



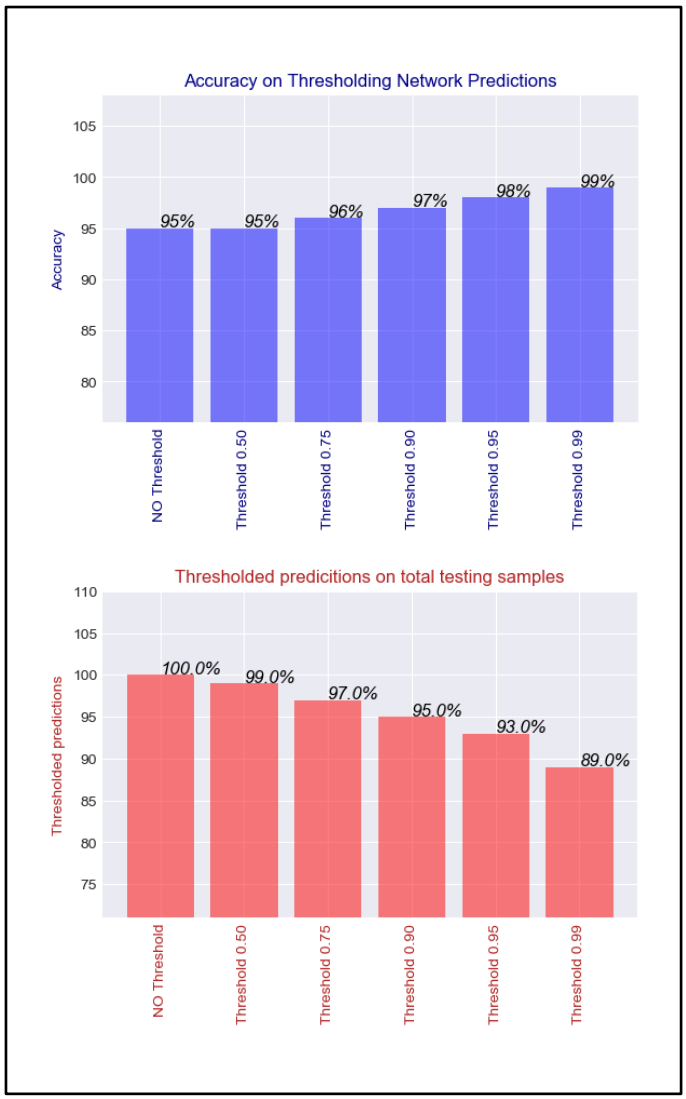
Resnet 50 – split 2



Resnet custom – split 2



Resnet 50 – split 3



Resnet custom – split 3

The barplots show, for both the models, a really good tradeoff between classification accuracy and level of confidence. In particular, in each box, we could appreciate, as expected, on the upper graph, an increase of accuracy with the increasing of threshold. This is because I'm discarded by the classification all the sample for which the model is not sufficiently confident in its prediction. At the same time, as expected, on the bottom graph we could see a progressive decrement of the number of samples survived to the applied threshold value. The good news is that, with a **99**% of threshold, on all split and on both models, we are able to predict, **at least**, the 89% of total samples.

In particular, even in this scenario, Resnet 50 performs well, continuing to classify a bigger number of samples for each threshold value and in each split, but the gap with the custom Resnet is absolutely affordable. It's then confirmed, as for prediction accuracies, the major variance of custom Resnet even prediction scores.

Split 2 appears be the better one for test set prediction. Both the models, in fact, have their performance peak in the second split. In particular, Resnet 50, has been able to predict with an accuracy score of 99% the 94% of the data. Resnet custom, instead, always on split 2 and with a score of 99%, the 92% of total amount of data. These are really comfortable results.

Part 2

This second section of the document, as required, has been dedicated to the second part of the assigned task. Inside dataset there are some samples that appears as too dark or too illuminated, with intensity values narrowed or stretched out. This is a trivial consequence of the fact the dataset has been collected sampling a video captured while driving. So, frequent change of light can easily bring the lens camera out of focus. This is a typical problem of *image processing*.

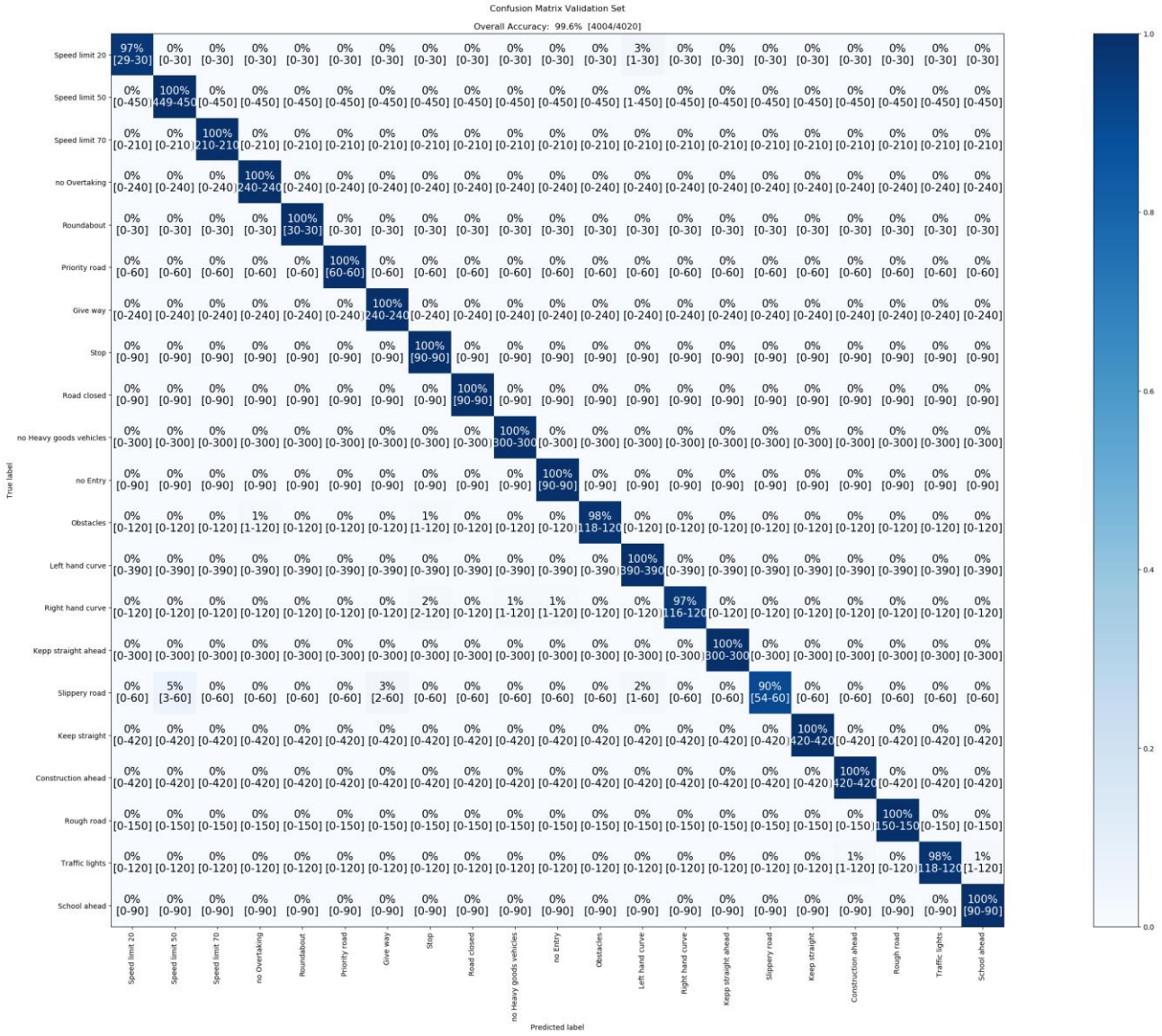
The task consists on the classification accuracy "correction" of classes number 12 and 13. More in detail, probably even thanks to the CLAHE transform used as preprocessing, these classes have never represented a particular problem, so this second part of the problem has not been cared as the first one.

Taking a look to the confusion matrix in appendix, about these 2 classes, we could know the worst case is represented by 90% for *Right hand curve*, on split 3, but it's clearly not duty to a specific problem in data images but a general performance drop duty to that particular split and model. In all other cases, the minimum accuracy for both the classes and the models has been 97%.

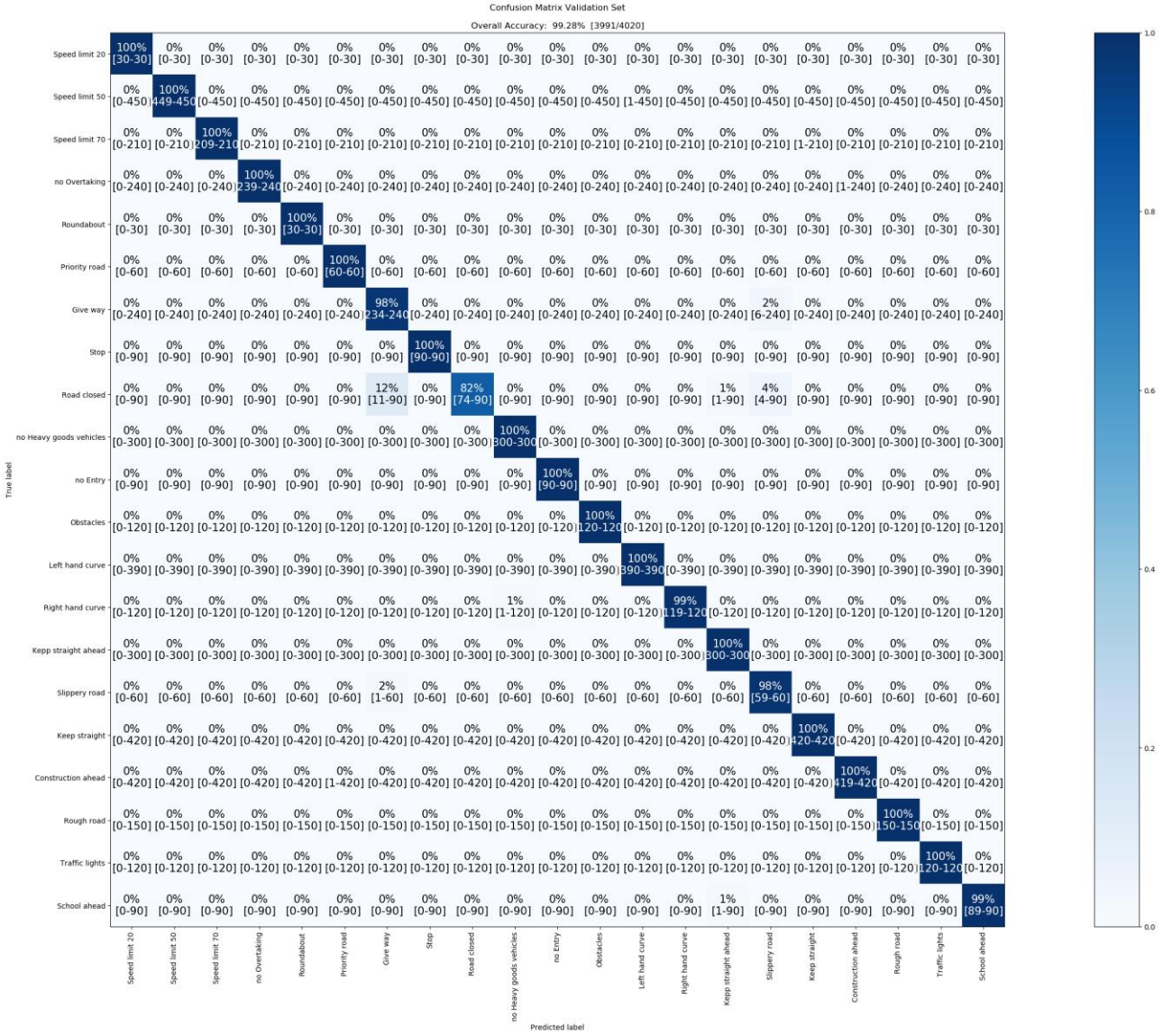
APPENDIX

This section contains all the confusion matrix for both the models and for each of the three splits considered.

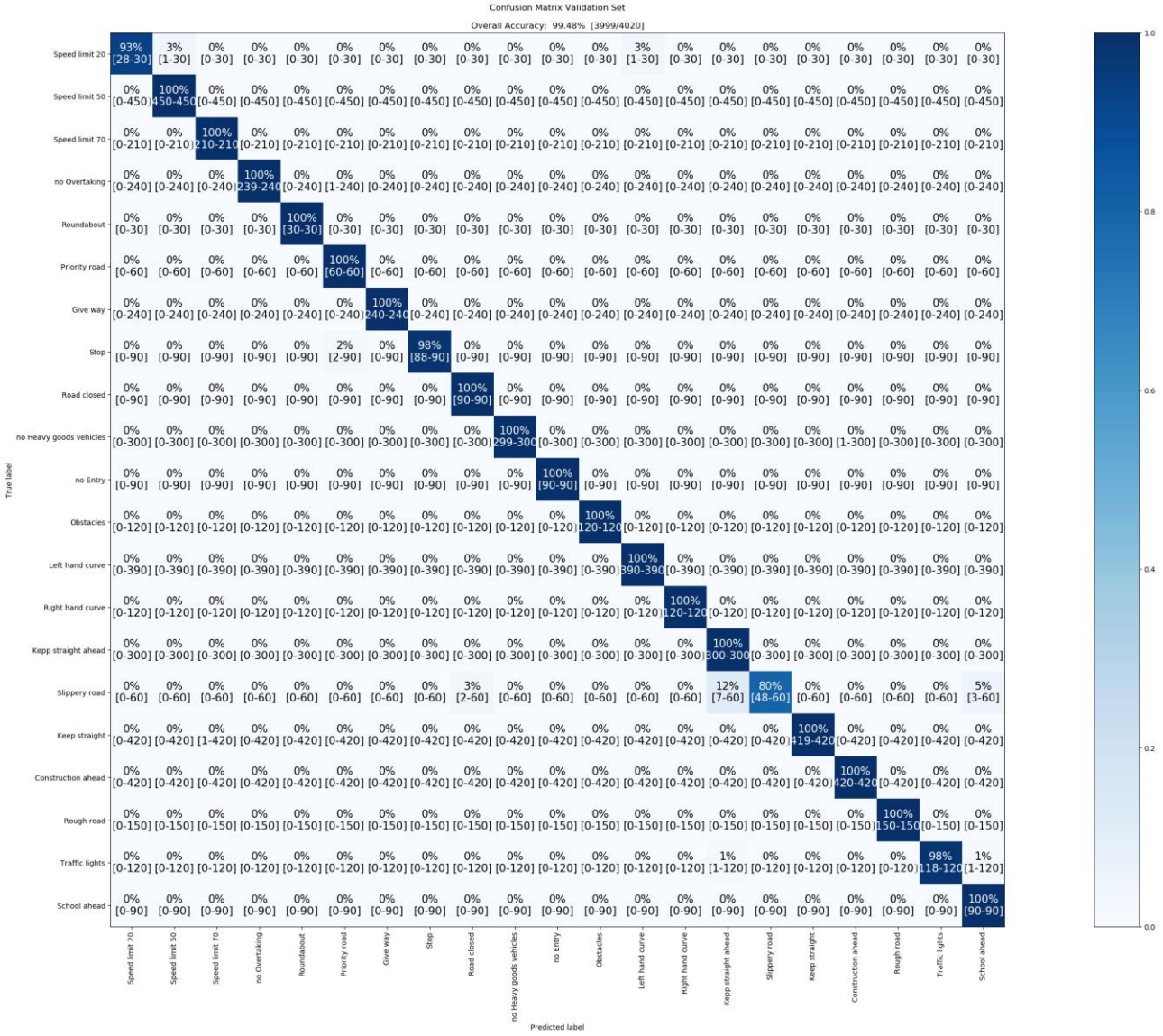
Confusion matrix for validation set (Resnet 50 and custom Resnet)



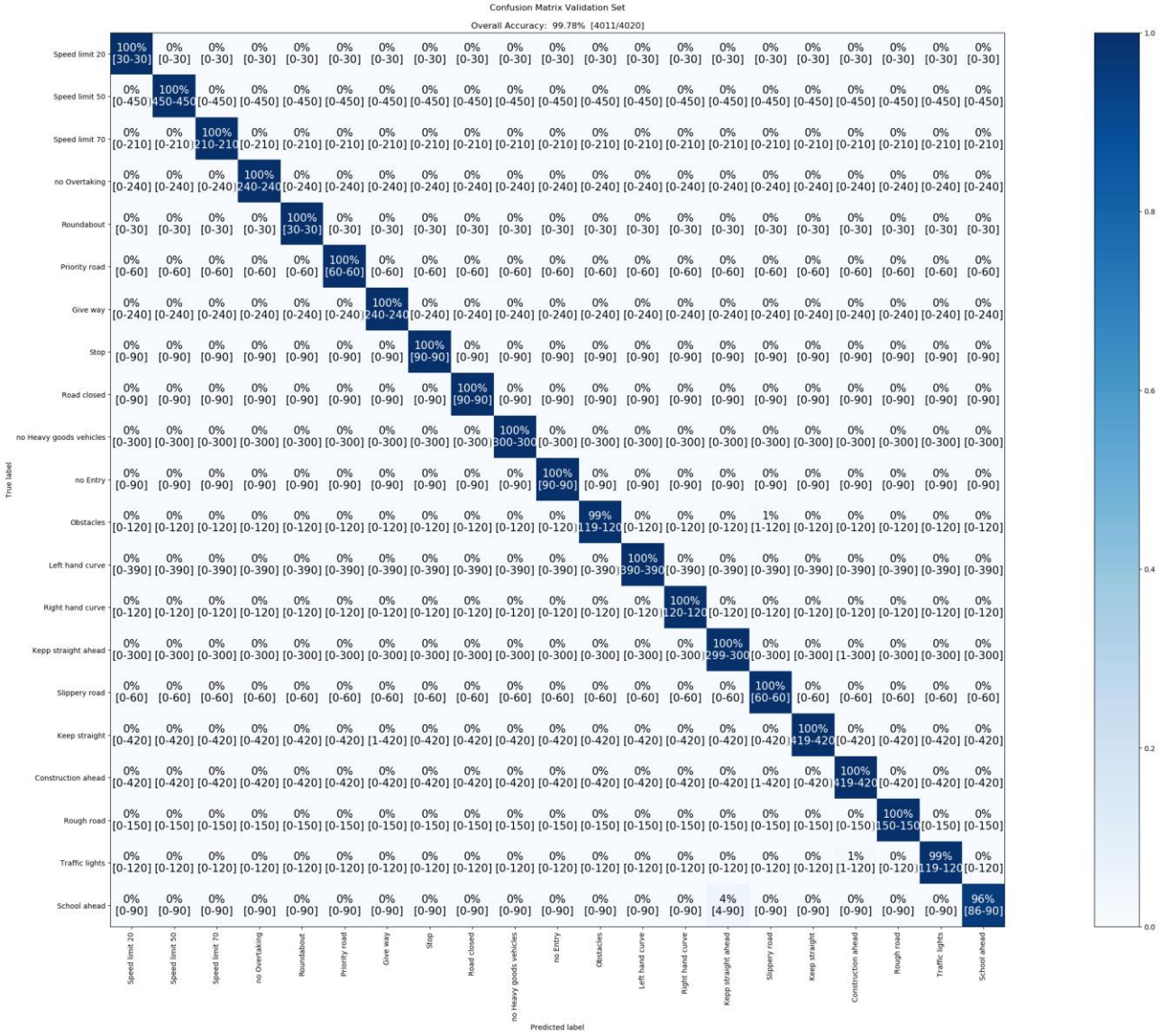
Confusion matrix for Resnet 50 on split 1



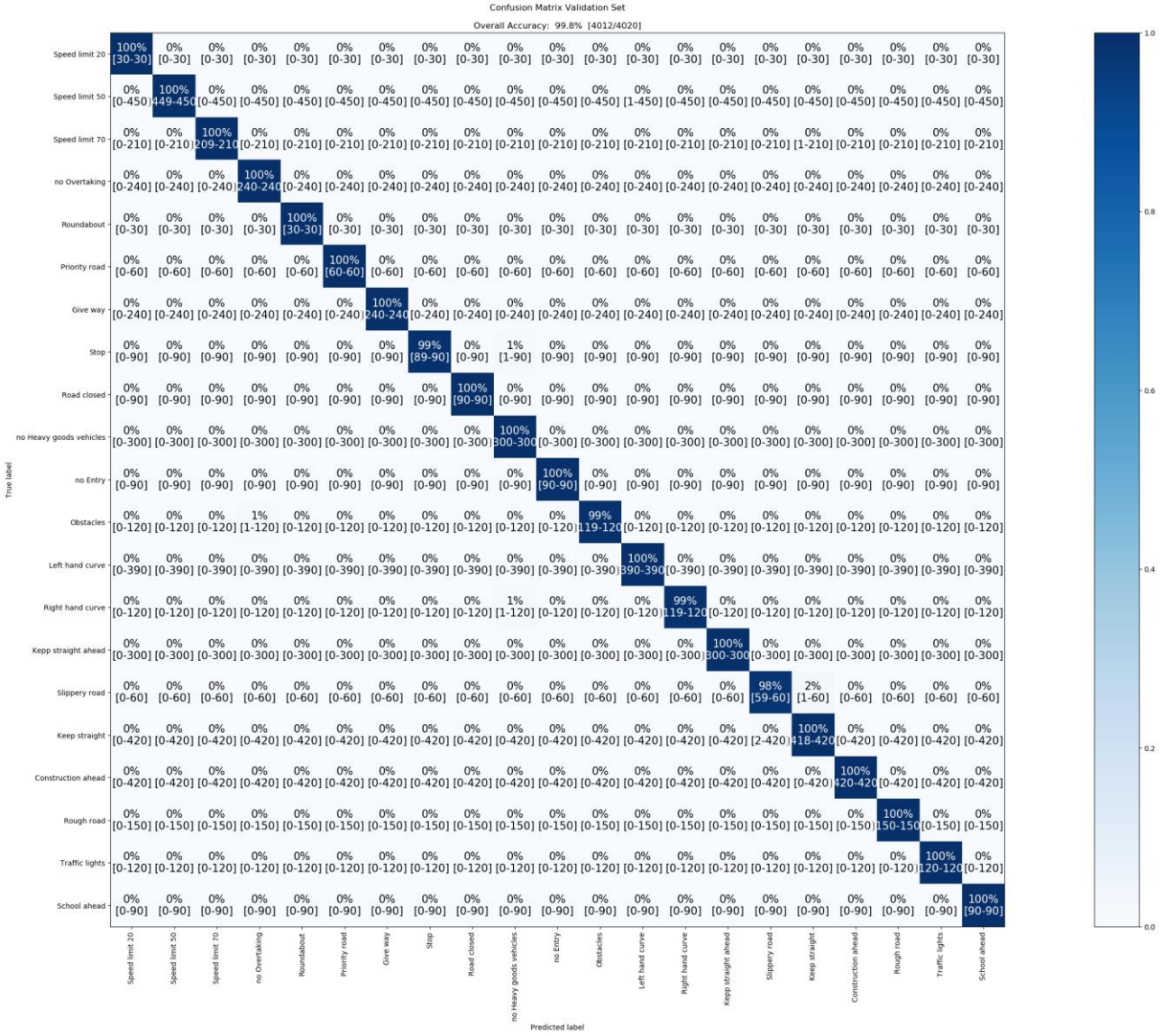
Confusion matrix for Resnet 50 on split 2



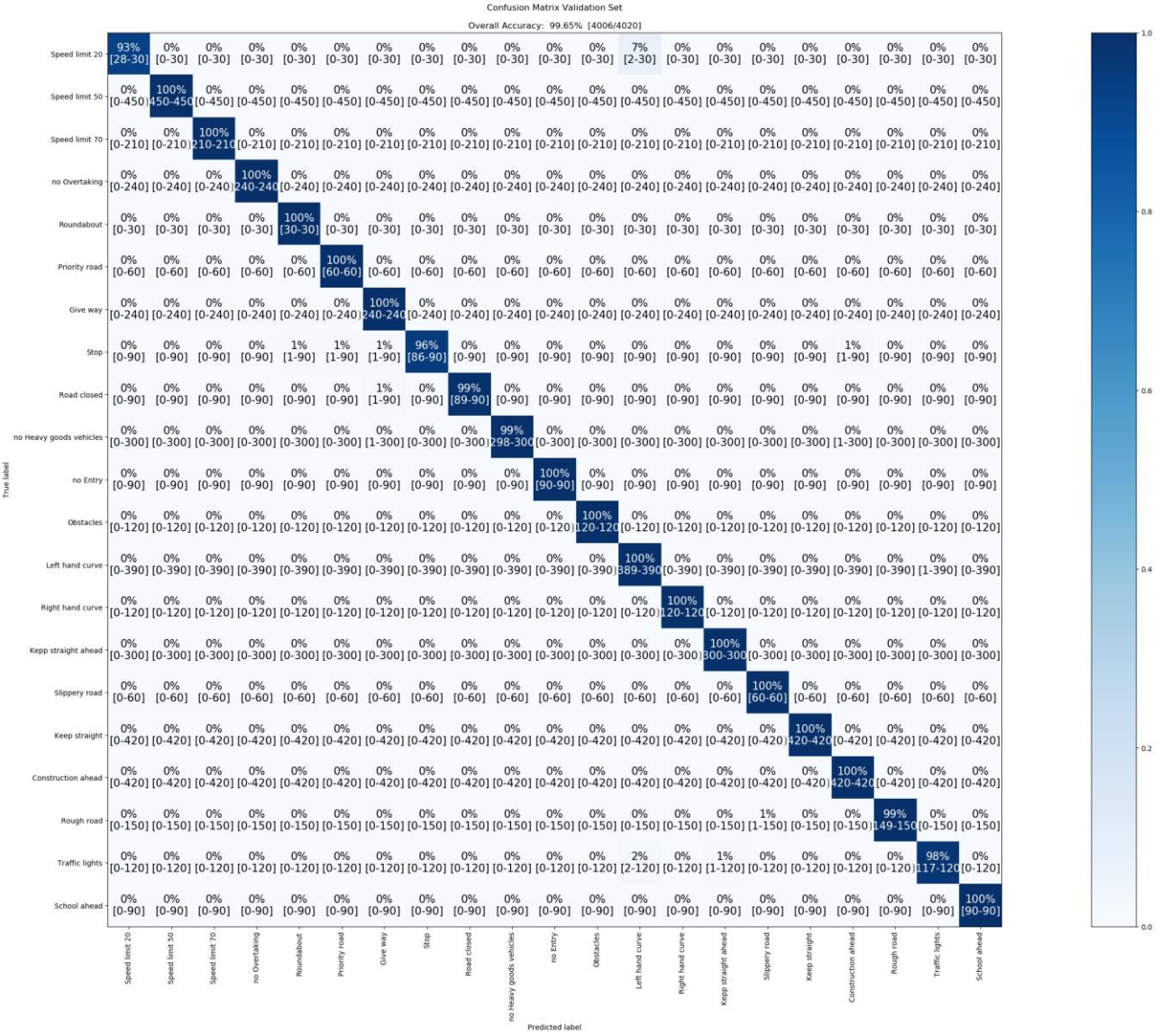
Confusion matrix for Resnet 50 on split 3



Confusion matrix for Resnet custom on split 1

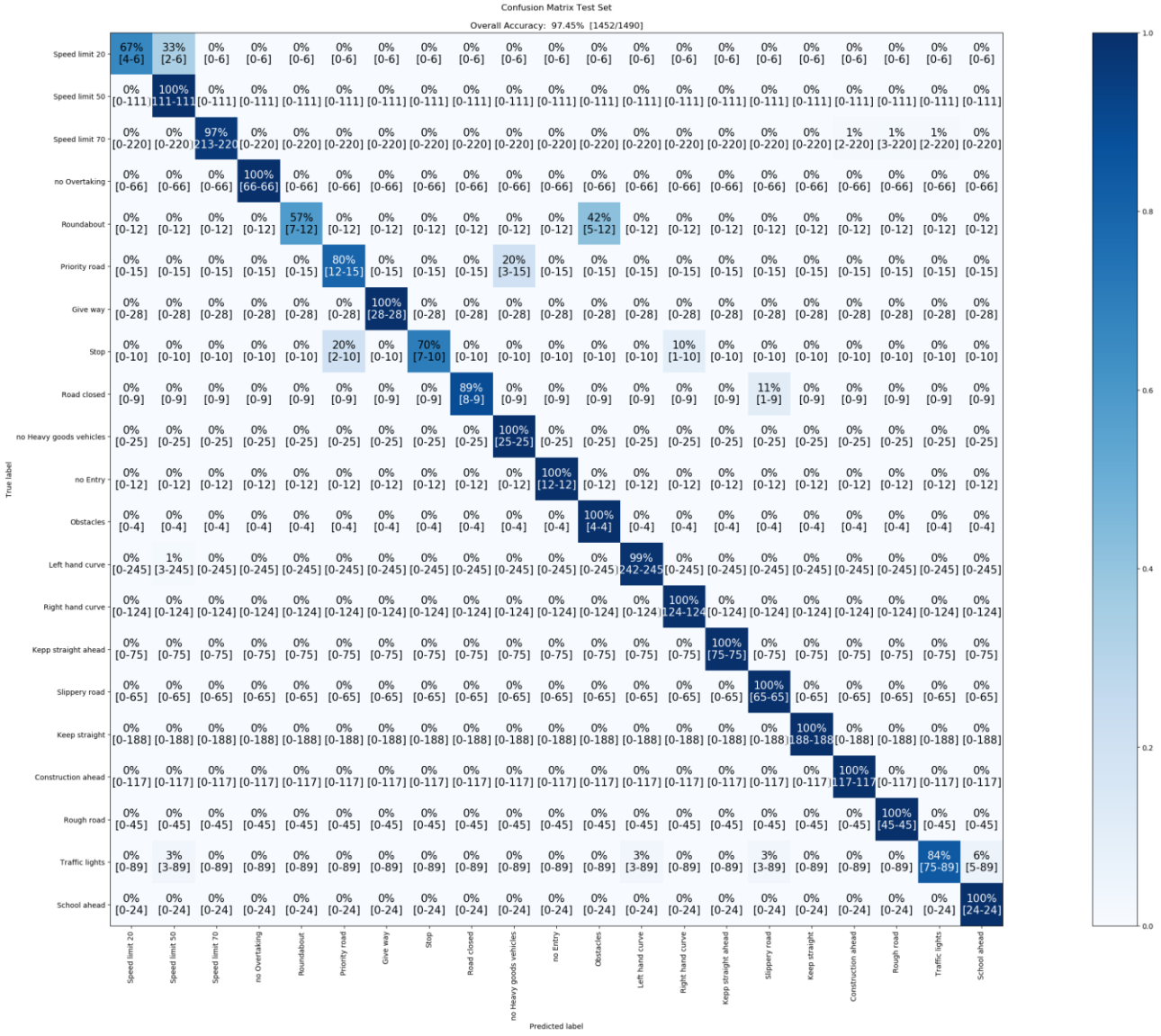


Confusion matrix for Resnet custom on split 2

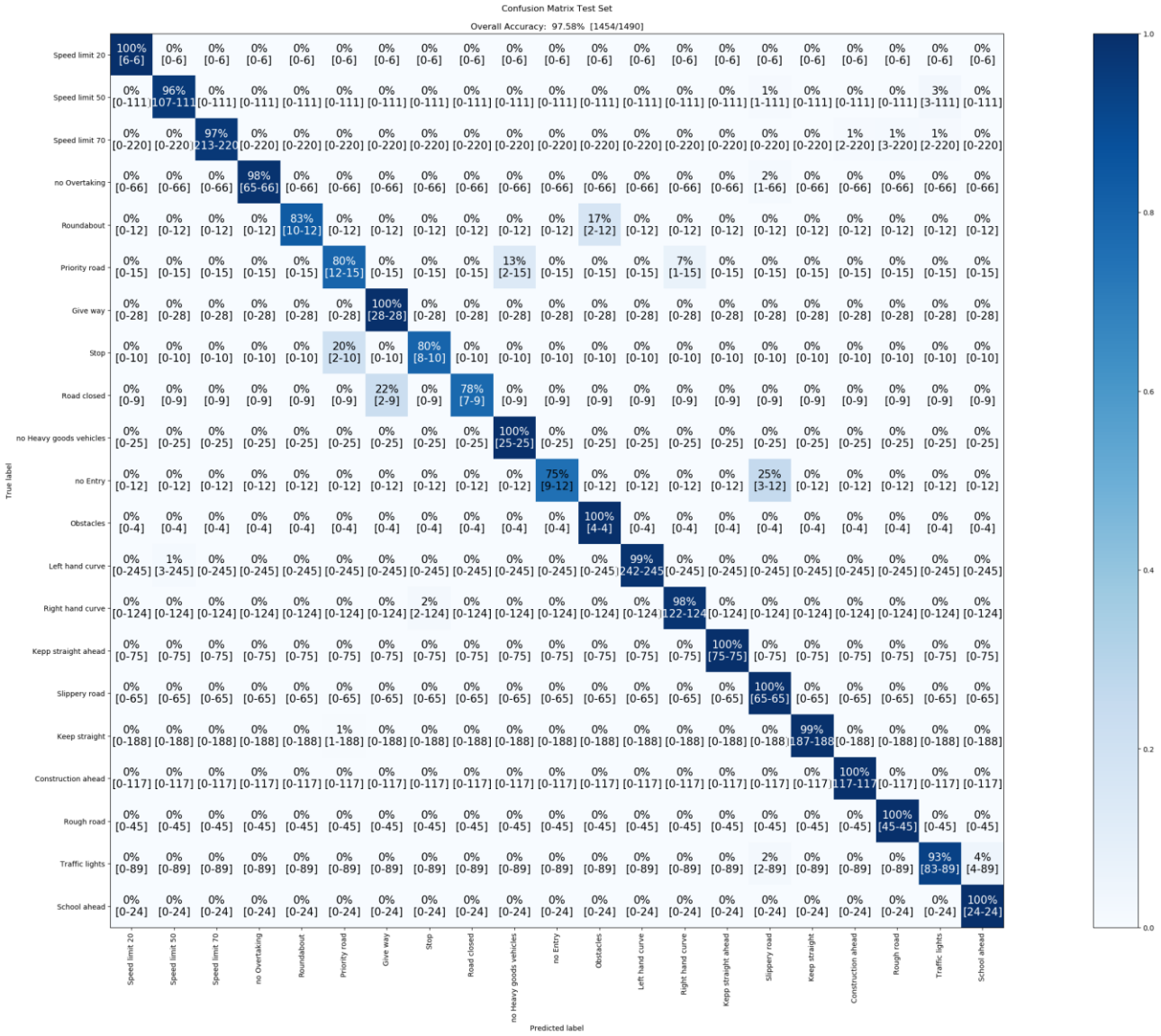


Confusion matrix for Resnet custom on split 3

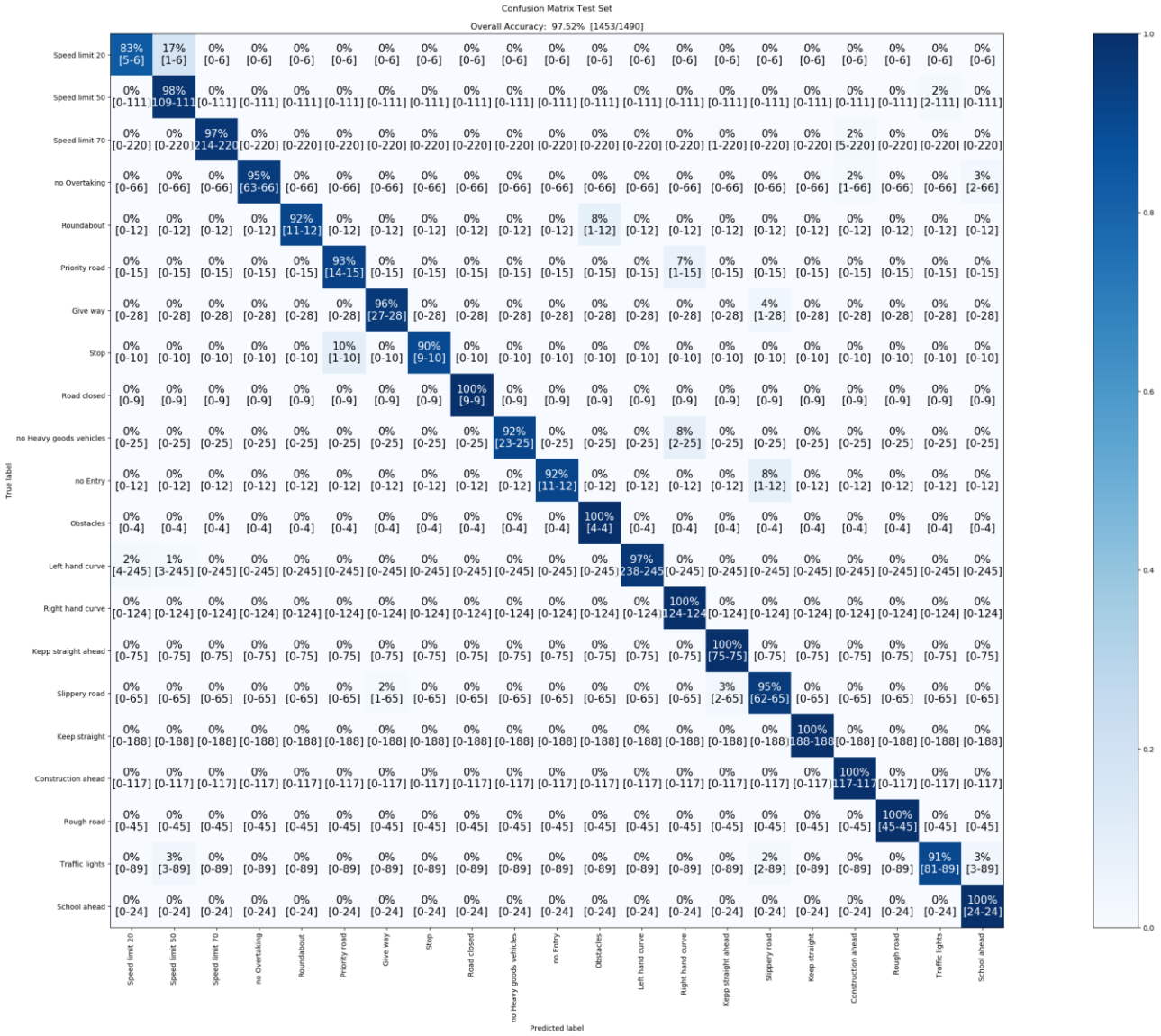
Confusion matrix for test set (Resnet 50 and custom Resnet)



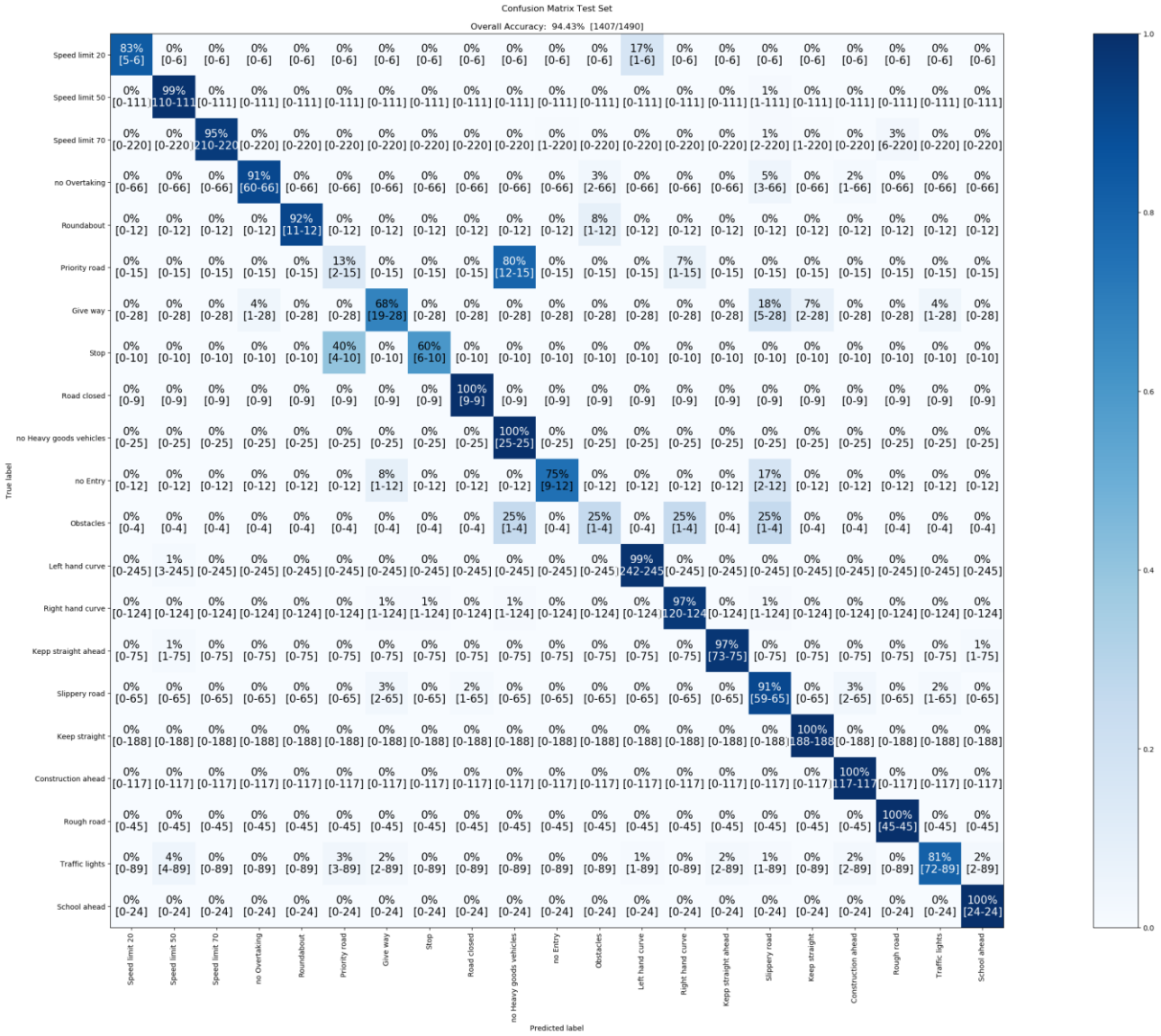
Confusion matrix for Resnet 50 on split 1



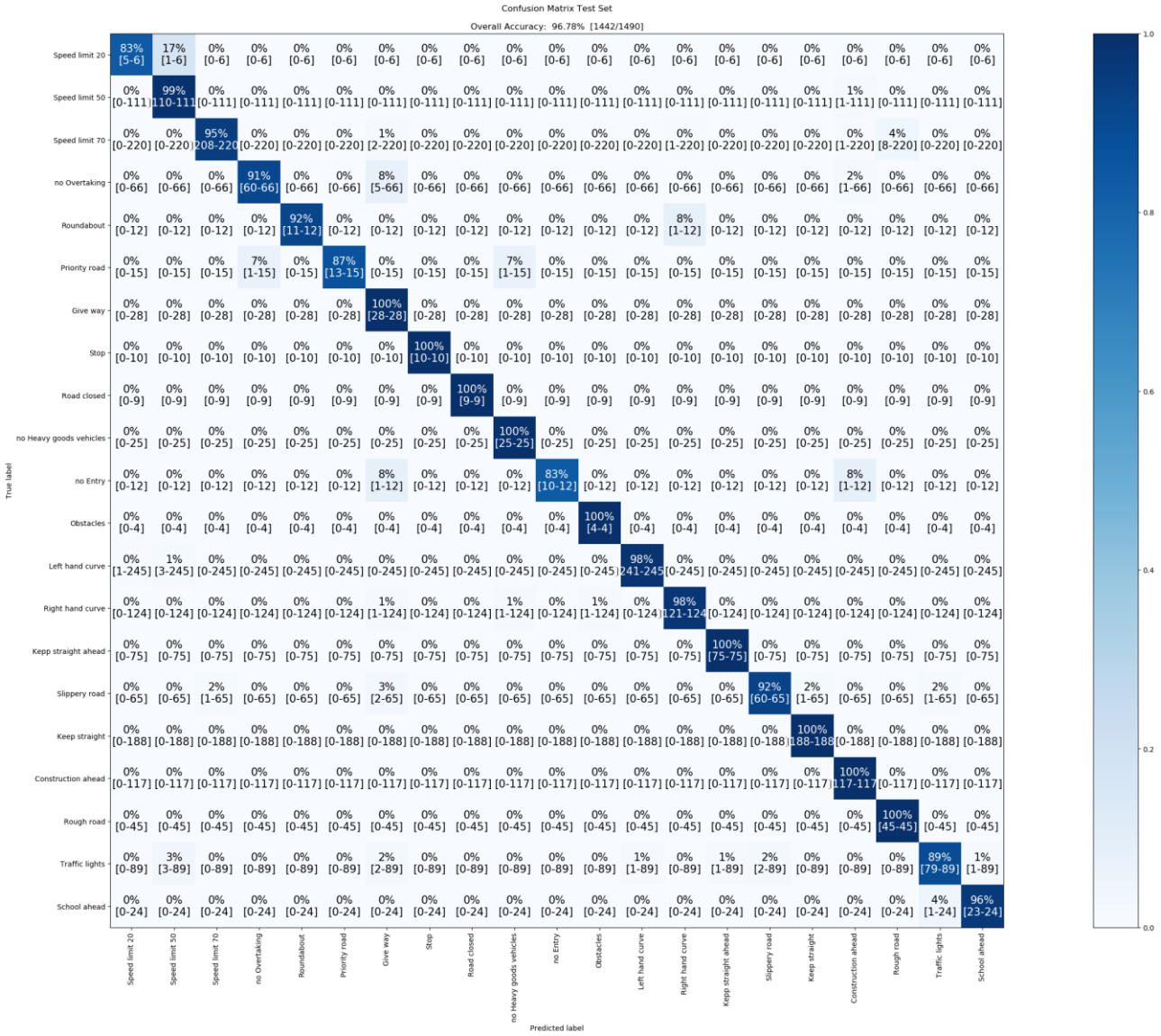
Confusion matrix for Resnet 50 on split 2



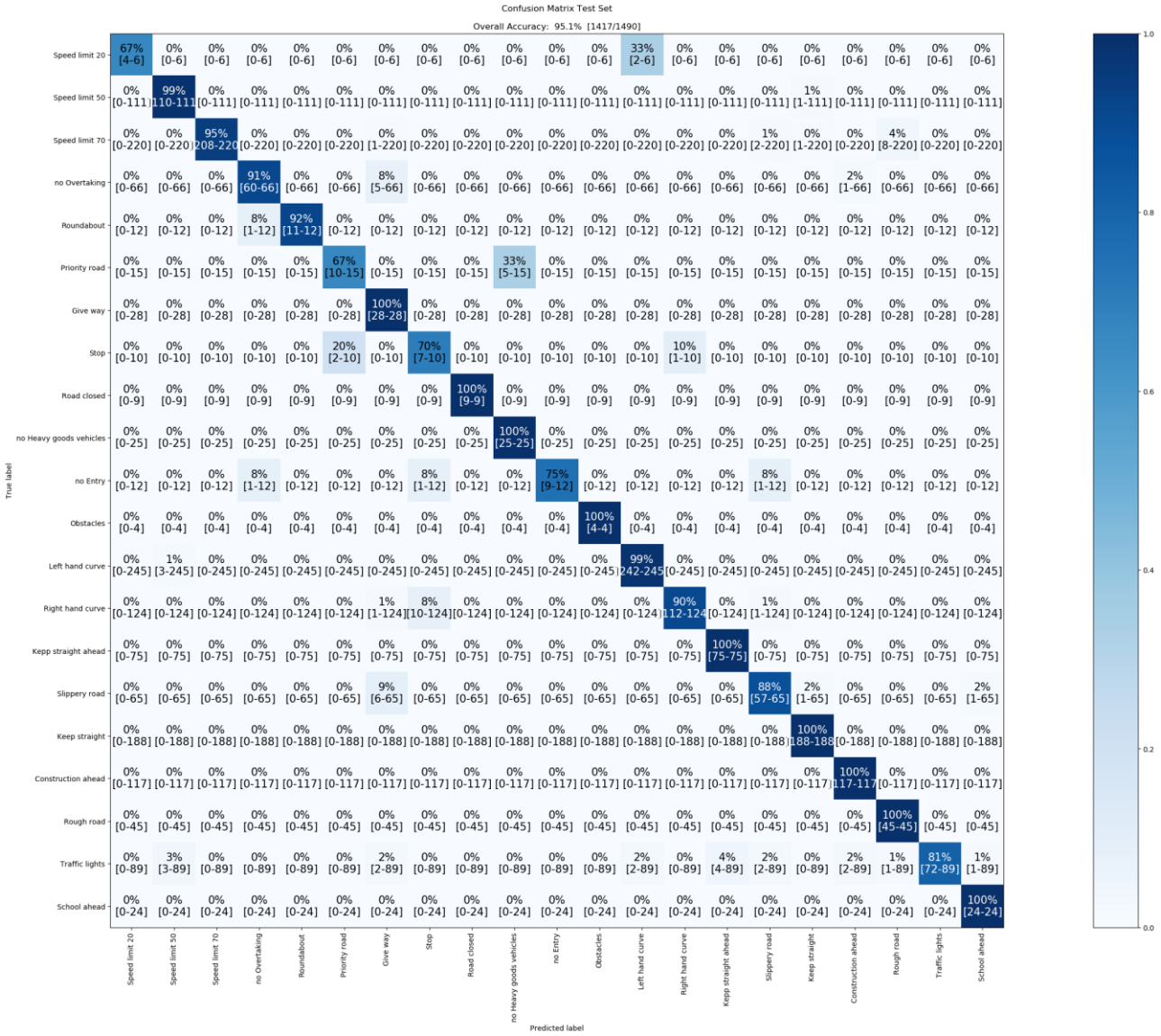
Confusion matrix for Resnet 50 on split 3



Confusion matrix for Resnet custom on split 1



Confusion matrix for Resnet custom on split 2



Confusion matrix for Resnet custom on split 3