# Deep Hallucination Classification
Mihai Matei 511DS

The project was implemented with the latest tensorflow 2.4 and their own keras implementation using a 1070ti Nvidia graphics card. Pandas was used to pre-process the dataset. My own custom ***matmih*** library was used for plotting, statistics and model building.

The optimizer used was SGD since it seems to work best with images. If you search Google's Model Garden github you will find that all of their models are trained using this optimizer. The optimizer was started with a large 0.1 learning rate but with large BATCH size 512 and the models contained a ReduceLROnPlateau callback that reduced the learning rate by a factor of 5 when the validation loss did not decrease by 0.001 for 4 epochs.
The models were trained for a maximum of 150 epochs with early stopping after 20 epochs of no validation loss decrease and the epoch with the largest validation accuracy was saved and used.
The loss used was sparse crossentropy loss – the default loss used for multiclass classification. The loss was weighed with the dataset class distribution.
Additional data, saved weights can be found at https://github.com/glypher/ml_competition

## CNN Model and model search history

I have tried many approaches to finally implementing my own custom CNN model, described later in the document. First I did try to do transfer learning with Google's MobileNet_v2 and Resnet50 trained on ImageNet and upscale the training images to 244x244 but I couldn't achive more than 0.69 accuracy on the validation search. Then, since the data is 32x32, a multiple layer dense network did also perform in the range of 0.68 percent validation accuracy which lead me to believe that big CNN models do not really help on this specific dataset so I did search what was a best model for classification on a similar size and resolution dataset such as CIFAR10.
The model found which performed very well on the dataset contained VGG style Convolution layers (2 Convolution layers of equal number of filters) with 3x3 filter size (since our 32x32 is quite small increasing the filter size did not help) followed by MaxPooling of size 2x2 and Dropout. Additional dense layers were added to help remove some of the inherent correlation of the Convolution layers output prior to feeding it into a softmax classification layers. All internal layers use RELU activation which seems to work best with Convolution layers for this case as it is positive only when the convolution filter "activates". The model had multiple hyperparameter configurations, including number of cnn layers, number of filters for each layer, number and size of dense layers, optimizer used and many more. This allowed me to easily do a hyperparameter grid search using my own matmih library. Finally an inverted piramid type of increasing CNN layers size and dense size was found to have the best accuracy (around 0.71 on the validation set):

Input (None, 32,32)
64 3x3 Convolution + BatchNorm + RELU
64 3x3 Convolution + BatchNorm + RELU
MaxPool2D 2x2 + Dropout 0.3
128 3x3 Convolution + BatchNorm + RELU
128 3x3 Convolution + BatchNorm + RELU
MaxPool2D 2x2 + Dropout 0.3
256 3x3 Convolution + BatchNorm + RELU
256 3x3 Convolution + BatchNorm + RELU
MaxPool2D 2x2 + Dropout 0.3
512 3x3 Convolution + BatchNorm + RELU
512 3x3 Convolution + BatchNorm + RELU
MaxPool2D 2x2 + Dropout 0.3

GlobalAveragePooling2D
128 Dense + BatchNorm + RELU + Dropout 0.3
256 Dense + BatchNorm + RELU + Dropout 0.3
512 Dense + BatchNorm + RELU + Dropout 0.3
8 Dense + Softmax

## Data augmentation and Pretraining the Model on CIFAR10

To achieve better results the model was pretrained using CIFAR10 using 2 different augmenation methods: AutoAugment from Google's official model garden (https://arxiv.org/abs/1805.09501) and a random noise + flips. Both methods optained a validation accuracy of around 0.91 on CIFAR10 default split which is quite close to the 0.99 accuracy of the state of the art models https://paperswithcode.com/sota/image-classification-on-cifar-10.

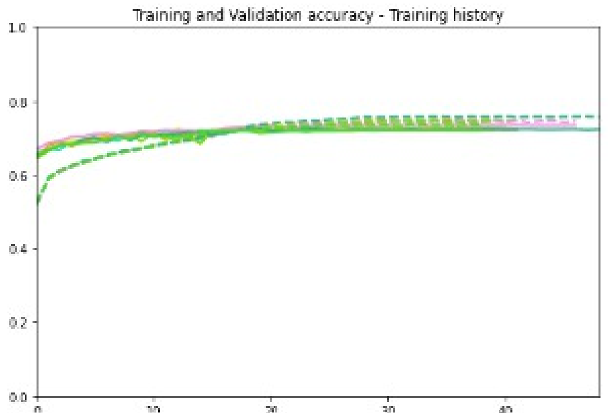
The data augmentation done using different augmentation images at every epochs and since AutoAugment takes a lot of time I used the parallel interleave approach for the Tensorflow Dataset, distributed on the number of CPU's available.

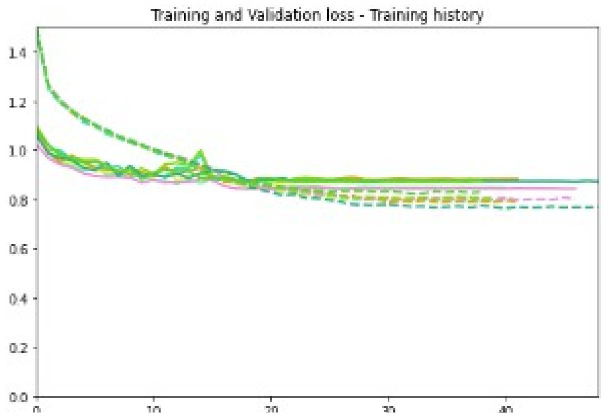
## Final Transfer Model

To make use of both available pre-trained models I build a composite model (please see mode_transfer.png in the archive) which has the same above model duplicated 2 times and their preclassification layer combined prior to the final softmax classification layer.

```
          CNNModel                          CNNModel
     CIFAR10 AutoAugment            CIFAR10 RandomNoise+Flips
           \                                  /
            \            Add(dense 512)      /
             ---------------------------------------------
                              |
                     8 Dense + Softmax
```

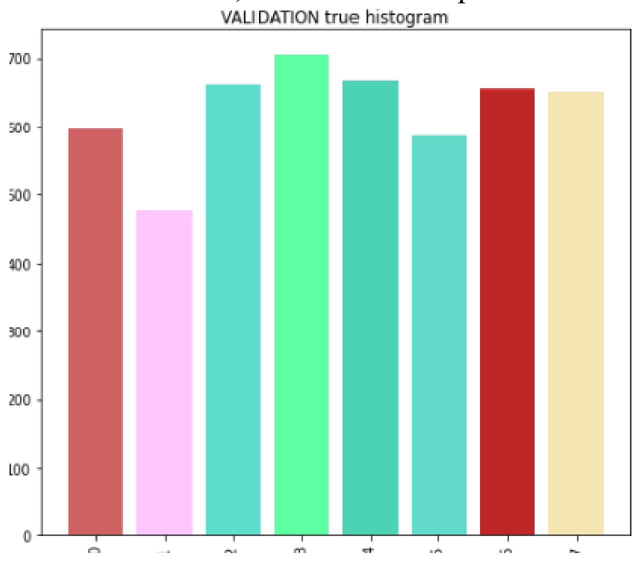
This model loaded with pretrained weights on the left and right part is then trained on our dataset. I train the model 8 times, first on the original train-validation split available in the competiton then on a 7 fold stratified split on all labeled data (train + validation).
The training was done with the same SGD optimizer with initial 0.1 learning rate and plateau learning rate reduction, batch size of 512 and sparse cross entropy weighted loss on the class distribution. Training as also done using new AutoAugment data augmentation at each step as described in the last paragraph.
The validation accuracy found was 0.7238 on the original train-validation split with 0.7331, 0.7278, 0.7256, 0.7340, 0.7246, 0.7262, 0.7214 respectively on the kfold split

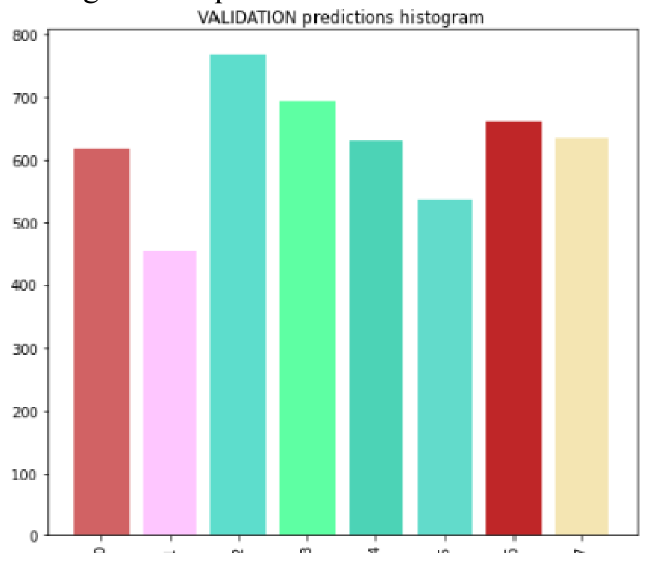


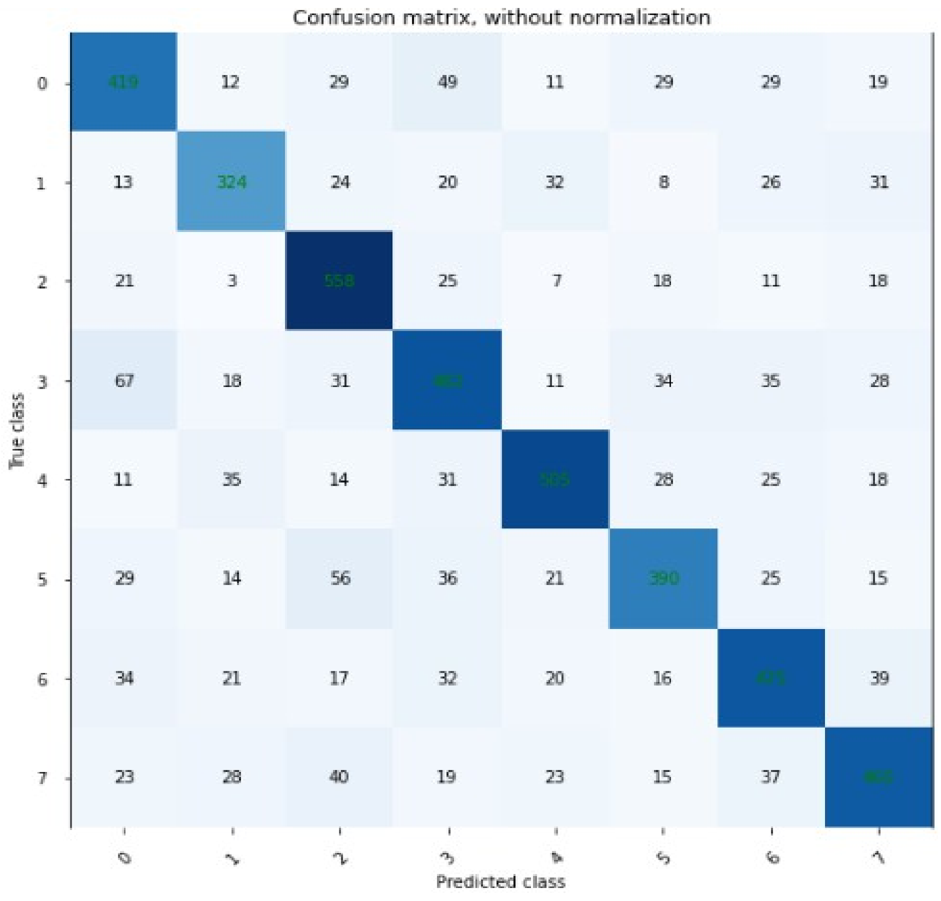


**<u>Original Validation Set metrics</u>**

As specified the original validation accuracy for the transfer learning model was **0.7238.** Confusion matrix, ROC curve and predicted class histogram was plotted as well.

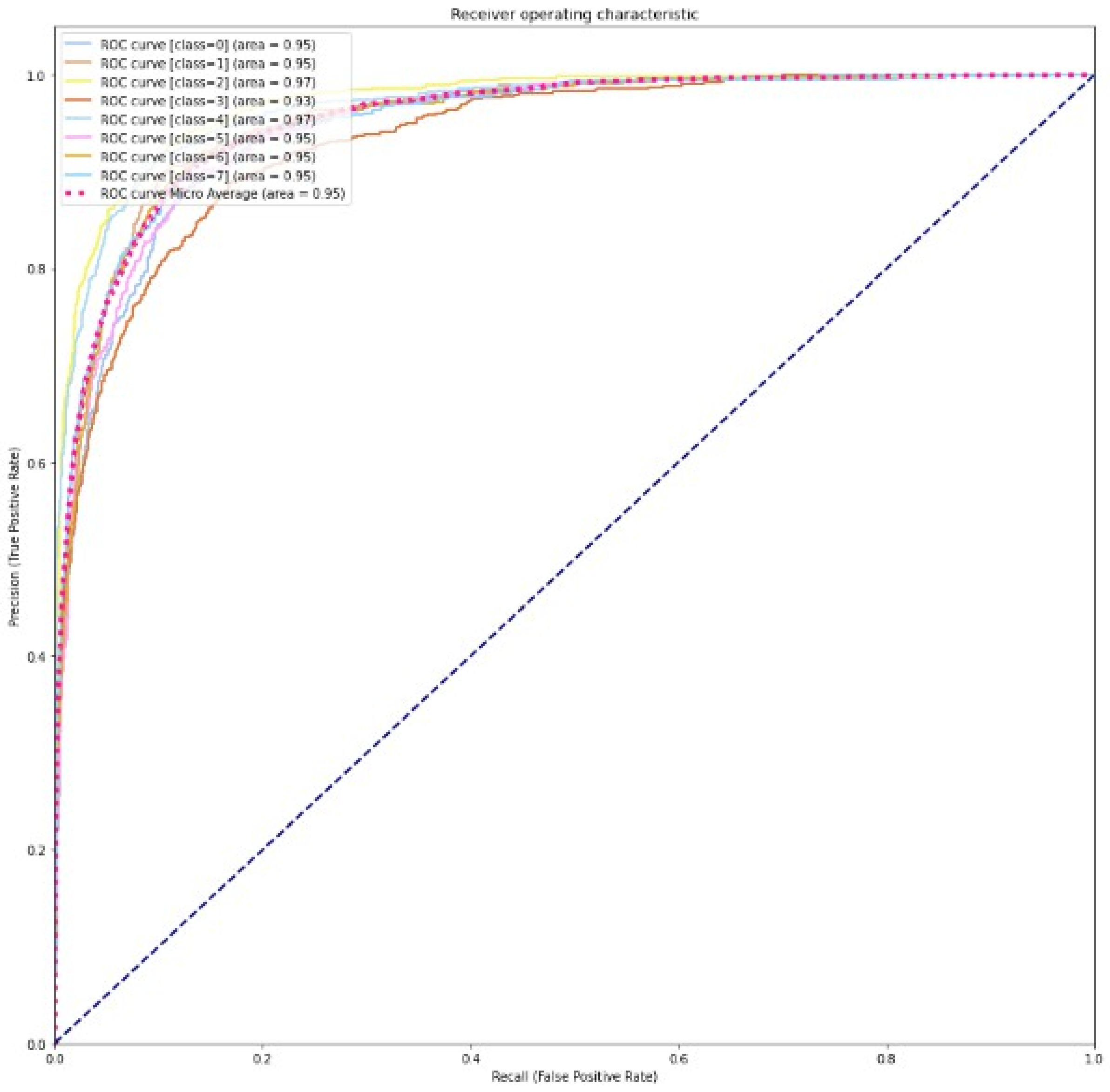






| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 419 | 12 | 29 | 49 | 11 | 29 | 29 | 19 |
| 1 | 13 | 324 | 24 | 20 | 32 | 8 | 26 | 31 |
| 2 | 21 | 3 | 558 | 25 | 7 | 18 | 11 | 18 |
| 3 | 67 | 18 | 31 | 482 | 11 | 34 | 35 | 28 |
| 4 | 11 | 35 | 14 | 31 | 505 | 28 | 25 | 18 |
| 5 | 29 | 14 | 56 | 36 | 21 | 390 | 25 | 15 |
| 6 | 34 | 21 | 17 | 32 | 20 | 16 | 471 | 39 |
| 7 | 23 | 28 | 40 | 19 | 23 | 15 | 37 | 465 |

The histogram and confusion matrix shows that amomg all, class 3 is one with the larger misspredictions. This can be seen from the ROC curve as well since each curve is a probability curve of the model's binary discrimination ability of that class agains all other. For example class 3 has the lowest value of 0.93 AUC (area unde the curve).



## <u>**Ensemble Model**</u>

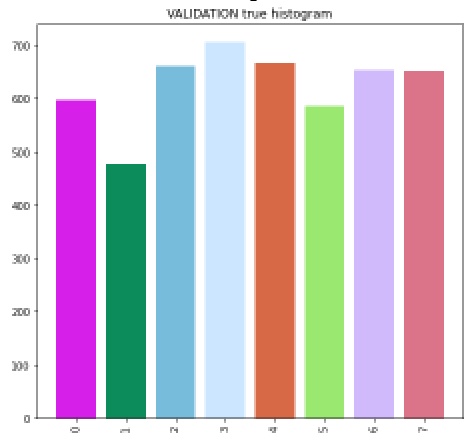
To further improve the final overall test accuracy an ensemble model was done using the predictions of all the above 8 trained models. Two different kind of ensemble model prediction selection was used:

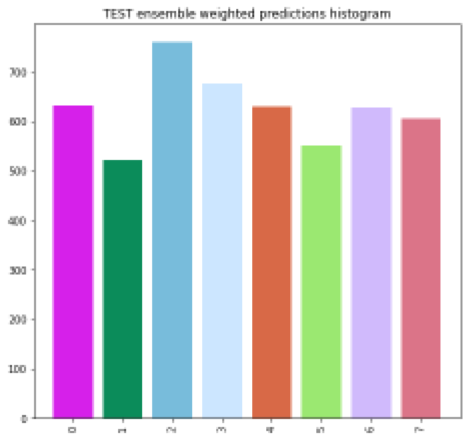
- max prediction – majority voting using all of the prediction of each of the model
- trained weighted prediction – Trained weights on each of the model class prediction and model importance by optimizing the results on all of the data accuracy.

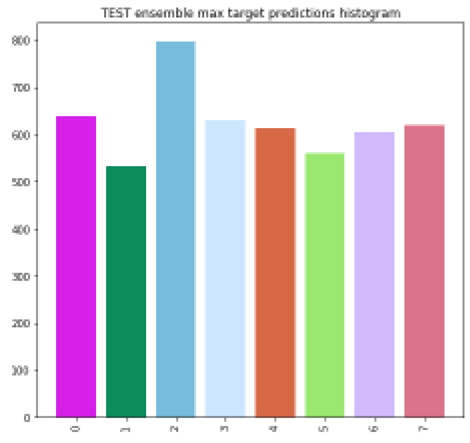
All combinations of the trained models in the final ensemble model (combination of 2, 3, .. 8 models) was tested on all of the data with both max prediction and weighted prediction. The model using weights for original split and folds 1,2,4,6 was found to have the best accuracy of 0.91 on all of the data (this is big because each of the model sees different part of the data).
Also all of the 8 models was used as well in the prediction.
Max prediction seemed to be better in most cases than trained weighted prediction.

## Test Data Results

The 2 above chosen ensemble models were submitted in the competition and obtained 0.722 and 0.723 accuracy respectively. Also the predicted test class histogram was plotted for each model to compare it to the validation histogram since I assumed that their distribution should be the same:







## Conclusion

Overal I am not happy with my results. Bigger CNN models have a tendency to overfit the data since they have larger capacity. I did not try the smallest deployment of the official vision models and assumed that a smaller model is better since it will have a larger regularization.

Additionally I have to say that I had only a limited intuition on what to do and how to add the layers and was more of a trial by chance that a guided strategy on what to do. But to my defense I did not really find anything online and even on some of the mode papers (VGG, ResNet) on how/what to do. If I had a lot of time I would plot the distribution of the data input and output at each layer to check how each layer decorrelates the data, the weights statistics at each layer and so on.