**Final Project Report – RandAugment  
Vlad Shtompel 314614397**

**Abstract**

In this work, I evaluated the algorithm ‘RandAugment’ as proposed in [1].

Data augmentation is a powerful technique to increase the generalization capability and reduce overfitting of deep learning models. However, it is not obvious which augmentations should be applied for a certain task, nor at which frequency or strength. In recent years, many different augmentation policies were proposed, such as ‘AutoAugment’ (AA) [2].

However, these policies usually require a second optimization procedure to select the optimal augmentation strategy for the task.

In the paper [1], the authors propose a simple augmentation strategy which performs about as good as, and sometimes better than, methods such as AA but with a much simpler optimization procedure, with a considerably smaller search space.

**Algorithm**

Key idea:

* Define a list of K transformations which make sense for the task at hand (*some* domain knowledge is required)
* Define a linear scale M of strength for each transformation, say 0 to 10.
* Now you have 2 parameters to optimize, M and N – number of transformations to choose out of possible K.
* Finally, at training time, to each sample apply N random transformations with strength M.

The advantage or RA is that its very straightforward to implement and optimize, only roughly 102 search space size, and it allegedly performs quite well.

The downside, is that a more thorough algorithms can likely perform better (as evident even by the RA paper authors).

As a baseline comparison, I used a simple random flip + random resized crop augmentation policy.

**Data[[1]](#footnote-1)**

Due to the nature of the algorithm, I had to work with image data (classification).

Finding 20 image datasets which were of reasonable size proved to be quite difficult. I ended up following Prof. Lior Rokach’s suggestion of picking 4-5 datasets and splitting them to artificially create 20 datasets:

[VGG-Flowers](https://www.robots.ox.ac.uk/~vgg/data/flowers/102/index.html) x 1 (102 category, no split)

[The Oxford-IIIT Pet Dataset](https://www.robots.ox.ac.uk/~vgg/data/pets/) x 1 (only the cats, 12 categories, no split)

[Stanford Dogs Dataset](http://vision.stanford.edu/aditya86/ImageNetDogs/) x 6 (120 categories, randomly split into 6 datasets of 20 each)

[CINIC-10](https://paperswithcode.com/dataset/cinic-10) x 2 (10 categories, split into 2 datasets of 5 each, randomly discarded 90% of the images per class. Left with 2,700 samples per class)

[CIFAR-100](https://www.cs.toronto.edu/~kriz/cifar.html) x 5 (100 categories, randomly split into 5 datasets of 20 each)

[COIL-100](https://www1.cs.columbia.edu/CAVE/software/softlib/coil-100.php) x 5 (100 categories, randomly split into 5 datasets of 20 each)

After some cleaning up and organizing, I ended up with 20 datasets as described above with a total of more than 125,000 images(!)

**Method**

Since I was required to compare 2 algorithms, each across 20 datasets, each with a 10-fold cross-validation loop, each with a 3-fold inner cross-validation loop with 50 hyperparameter trials...

Needless to say, this was an absurd number of models that I had to train, which was only made worse by the fact that I had to do it on images.

Some concessions had to be made:

\* I used a relatively small model (ResNet-18)

\* I used a pretrained model and only replaced the last layer for each dataset.

\* I only trained for 2 epochs per inner CV fold and 10 epochs per outer CV fold.

\* I used [optuna](https://optuna.org/) to search for hyperparameters and early pruned unpromising trials.

\* Worst of all was realizing I could set up the experiment better, but only half way through training. Sadly, I could not afford to rerun everything. (More on that in the discussion)

Still, it took nearly 2 weeks to train everything on 4 machines:

2 x Computers from work with GTX 1080TI (only partially available to me)

1 x Microsoft Azure VM with a Tesla V100 (only partially available to me)

1 x My own machine with RTX 3060

(I added a ‘GPU’ column to the results tables to help make sense of the train\infer times)

**Hyper-opt**

The parameters I chose to optimize were:

* Learning rate – [1e-5, 1e-3] (log-uniform):  
  LR is probably the single most important hyperparameter to optimize for any machine learning model or task. Finding an optimal LR instead of going with some default value can result in 10ths of points in metrics increase. Choosing a bad learning rate can prevent a model from converging at all.
* Optimizer – [Adam\SGD]:

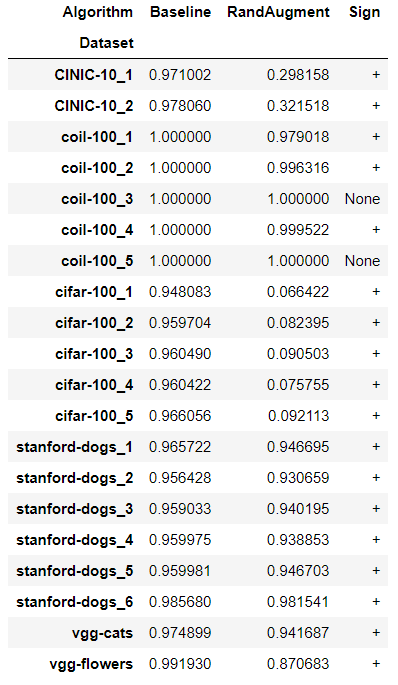
While Adam is considered quite a solid default, SGD still has the potential to out-perform in some cases. In addition, I added another parameter; amsgrad [True\False] and nesterov [True\False] for Adam\SGD respectively.

* Batch size – [32\128\256]:  
  Choosing a smaller batch size is known to have a regulatory effect on training, which helps the model generalize. However, for tasks with many classes it could be beneficial to have a larger batch size, such that most classes will be represented in each batch. To test these notions, I made 3 choices which represent ‘small’, ‘medium’ and ‘large’ sizes.

To score the different configurations, I used the F1 score since its very robust, takes into account both precision and recall and is generally considered a good metric for classification when we don’t have a preference (precision over recall or vice-versa)

To run the trials, I used optuna. The sampler was TPE (default) and in order to save time I pruned any trial that did not score in the 90th percentile on the first fold (out of 3).

Of course, the pruning was only enabled after running 10 full trials.



**Statistical test**

Since I only have 2 algorithms, I decided to use the sign test as an alternative to the Friedman test.

The null hypothesis for the sign test is that the difference between medians is zero.

I used the (median across 10 folds) of the area under the precision-recall curve to compare the algorithms.

The p-value for these results is obviously 0, so they are significant and the null hypothesis is rejected.

Also, there is no real need for a post hoc test.

**Discussion**

When considering which hyperparameters to optimize, I briefly considered to include M, N (the values for strength, and number of transforms) for RandAugment, but I quickly dismissed this idea under the assumption that it would not be ‘fair’ as in – the RA trials will have extra 2 parameters over the baseline model and therefore it will be considerably harder to find good values.

In hindsight, I think this was a mistake.

I chose the values 9, 2 (respectively for M, N) since this is what the authors of [1] claimed worked best for ResNet, thinking it should perform well for me. However, I failed to consider the difference in datasets between myself and what they used in [1].

It is clear from the results presented above, that the RA policy chosen does not fit datasets like cifar-100 (CINIC is also largely based on cifar). Had I made this realization earlier, I would have re-ran the training. But due to how costly the whole procedure was (in compute, and especially time) I could not afford to do so.

For some datasets, such as Stanford-dogs and coil-100 the policy that I chose seems to perform quite well, and I’m fairly confident that I could outperform the baseline policy had I tried to search for better M, N.

**References**

[1] Ekin D Cubuk, Barret Zoph, Jonathan Shlens, and Quoc V Le.

*RandAugment: Practical automated data augmentation with a reduced search space*.

[2] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. *AutoAugment: Learning augmentation policies from data.*

1. [Download link](https://drive.google.com/file/d/170xmCJGW3fpw0DBCRz-Fu-UFW22kzSa3/view?usp=sharing) [↑](#footnote-ref-1)