Project 1 - report

Tomasz Krupiński, Piotr Bielecki

February 2024

Abstract

In this report, we delve into the exploration and evaluation of various neural network architectures and hyperparameters for image classification tasks, focusing on CINIC-10 dataset. Our investigation encompasses a comparative analysis of different network architectures, including ResNet50, ResNet100, and a custom-designed convolutional neural network (CNN). We scrutinize the impact of altering hyperparameters associated with training the training process and regularization techniques. Additionally, we investigate the effects of data augmentation techniques, incorporating both standard operations and advanced methods. Through rigorous experimentation and analysis, we determine the efficacy of each configuration in enhancing model performance.

Furthermore, the research extends to the implementation of ensemble learning techniques, including hard and soft voting, as well as stacking. By combining the strengths of individual models, the ensemble approach aims to further boost classification accuracy and resilience to variations in data. Our findings not only shed light on the influence of different configurations for neural network architectures and hyperparameters but also underscore the effectiveness of ensemble methods in further improving classification accuracy.

Contents

1 Introduction **3**

2 Instruction of the application **3**

3 Theoretical introduction **3**

3.1 Data Augmentation 3

3.2 Pretrained ResNet 4

4 Experiments **4**

4.1 Hyper-parameters 5

4.2 Further training 6

4.3 Data augmentation 7

5 Conclusions **9**

1 Introduction

Our task is to test and compare different network architectures, from which at least one has to be a CNN model. Also, we should research the influence of hyper-parameters for training and for regularization process. We will also carry out experiments on the impact of data augmentation techniques. Then, at the end, we will try to increase our score by using different ensemble methods.

We have chosen four different architectures to experiment on for this project.

1. Own implementation of CNN (keras/tensorflow)
2. Pretrained ResNet50
3. Pretrained ResNet101
4. ResNet50

Those models allow us to use and compare different NN architectures. Especially interesting will be whether the use of pretrained NN is beneficial to achieving high accuracy, and if so, then whether smaller networks are sufficient.

2 Instruction of the application

The application is written in jupyter notebooks, available on our github repository:
`https://github.com/bieleckipiotr/Deep-Learning`

3 Theoretical introduction

3.1 Data Augmentation

Data augmentation is the process of artificially generating new data from existing data, primarily to train new machine learning (ML) models. ML models require large and varied datasets for initial training, but sourcing sufficiently diverse real-world datasets can be challenging because of data silos, regulations, and other limitations. Data augmentation artificially increases the dataset by making small changes to the original data.[1]

Data augmentation is incredibly useful when it comes to limited datasets - in the case of this study, our training dataset contains 90k observations, which is not a small amount, but its not large either. We used data augmentation primarily as a source of additional training data. Besides well known, standard augmentation techniques like rotation or flipping, we decided to use one quite recently developed technique called CutMix introduced by Sangdoo yun et al. in their paper from 2019 called *CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features*. The authors claim their technique is advantagious over other, less advanced regional dropout techniques, which *remove*

[1]AWS data augmentation page

informative pixels on training images by overlaying a patch of either black pixels or random noise. Such removal leads to information loss and inefficiency during training. In the original paper, the researchers compare CutMix with other advanced augmentation techniques: *By making efficient use of training pixels and retaining the regularization effect of regional dropout, CutMix consistently outperforms the state-of-the-art augmentation strategies*

3.2 Pretrained ResNet

ResNet50 and ResNet101 are variants of the ResNet (Residual Networks) architecture, each differing in the number of layers and model complexity.

ResNet50 consists of 50 layers, making it shallower compared to ResNet101. Despite its relatively smaller depth, ResNet50 demonstrates strong performance in various image recognition tasks. It strikes a balance between model complexity and computational efficiency, making it widely adopted in both research and practical applications.

On the other hand, ResNet101 boasts a deeper architecture with 101 layers. By incorporating more layers, ResNet101 can potentially capture more intricate features from input images, leading to enhanced representation learning capabilities. This increased depth often results in improved performance, particularly in tasks that demand a high level of feature abstraction.

Both ResNet50 and ResNet101 architectures inherit the fundamental principles of ResNet, such as residual learning and shortcut connections, which enable effective training of very deep networks. These architectures serve as invaluable tools for tasks like image classification, object detection, and semantic segmentation, empowering researchers and practitioners to tackle complex visual recognition challenges with remarkable accuracy and efficiency.

The base layers of ResNet are set to be untrainable, to preserve the already trained weights.

To classify images into 10 classes, we have added two layers onto the ResNet50 and ResNet101 architectures. First, there is a dense layer, with the output of 1000 data points. It has changeable activation fucntion, and kernel regularization. Then, to make predictions, at the end, there is a dense layer with the output of 10 data points. The activation function is softmax, as it is commonly used in the purpose of making classifications. There is no regularization applied.

4 Experiments

We have done different kinds of experiments:

- Hyper-parameters tuning
- Data augmentation

In every section, there is a description of activities done and the results.

4.1 Hyper-parameters

The goal of the experiments in this section were to achieve the best possible accuracy on the training set. To do that, we have done two things. First of all, we have applied a grid search of different hyper-parameters with a small number of epochs, to try finding the most prospering architectures. Then, we have chosen the best ones and increased the number of epochs with the intention of achieving better results.

The hyper-parameters chosen for the gridsearch were as followed:

1. Own implementation of CNN (keras/tensorflow)

training	**regularization**
optimizer	L1, λ = [0.001, 0.01, 0.1, 0.3, 0.5]
dropout rate	L2, λ = [0.001, 0.01, 0.1, 0.3, 0.5]

2. Pretrained ResNet50

training	**regularization**
optimization algorithm [SGD, Adagrad, ADAM]	L1, $\lambda = [0.1, 0.3, 0.5]$
activation function [RELu, Linear, SoftMax]	L2, $\lambda = [0.1, 0.3, 0.5]$

3. Pretrained ResNet101

training	**regularization**
optimization algorithm [SGD, Adagrad, ADAM]	L1, $\lambda = [0.1, 0.3, 0.5]$
activation function [RELu, Linear, SoftMax]	L2, $\lambda = [0.1, 0.3, 0.5]$

The gridsearch results, for the ResNet networks gave us following answers:

RestNet	Optimization	Activation function	Regularization type	Regularization parameter	Accuracy
50	Adagrad	relu	l2	0.1	0.516589
101	Adagrad	relu	l2	0.1	0.509022
50	Adagrad	linear	l2	0.1	0.506867
101	Adagrad	linear	l2	0.1	0.498456
50	Adagrad	relu	l2	0.3	0.493600
50	Adagrad	linear	l2	0.3	0.492378
101	Adagrad	relu	l2	0.3	0.487256
50	Adagrad	linear	l2	0.5	0.482678
50	Adagrad	relu	l2	0.5	0.481533
101	Adagrad	linear	l2	0.3	0.481100
101	Adagrad	relu	l2	0.5	0.472811
101	Adagrad	linear	l2	0.5	0.472022
50	SGD	relu	l2	0.1	0.436200
50	Adam	relu	l2	0.1	0.432344
50	SGD	linear	l2	0.1	0.430867
...	...	...	...	...	...

Therefore, we have decided to use both ResNet50 and Resnet101, with Adagrad as Optimization algorithm, relu as activation function, l2 as regularization type and $\lambda = 0.1$ as regularization parameter.

The results for own CNN gridsearch:

Optimizer	Dropout	Lambda	Regularizer	Best Accuracy
SGD	0.1	L2	0.001	0.547417
Adagrad	0.1	L2	0.100	0.531194
Adagrad	0.1	L2	0.500	0.525667
SGD	0.2	L2	0.001	0.521458
Adagrad	0.1	L2	0.300	0.519708
SGD	0.1	L2	0.010	0.518611
Adagrad	0.1	L1	0.001	0.516764

Table 1: Own cnn experiment results

4.2 Further training

After finding optimal parameters, we have tried to increase the accuracy by training the models on more epochs. Depending on the models, there was different amount of epochs used to train them.

NN name	**epochs**	**accuracy**
Pretrained ResNet50	50	0.5573
Pretrained ResNet101	50	0.5583
Fully trained ResNet50	20	0.6314
Own implementation	50	0.6163
Fully trained ResNet 50 + augmentations	20	0.83

4.3 Data augmentation

We decided to use four standard augmentation techniques as well as single advanced one.

1. CutMix,
2. Rotations,
3. Flipping (only vertical),
4. De-colorization,
5. Added noise

Experiment results:

augmentation	ResNet50 accuracy	ResNet101 accuracy
None	0.52	0.51
Rotation	0.47	0.45
Vertical flip	0.52	0.51
Decolorization	0.47	0.47
Gaussian Noise	0.53	0.52
CutMix	0.52	0.52

The results are not conclusive as to whether data augmentation actually helps the training process, unfortunately we did not keep track of the validation accuracy, which could have pointed us to a regulatory feature of these augmentations. in figures below, examples of the impact of used augmentations is presented.

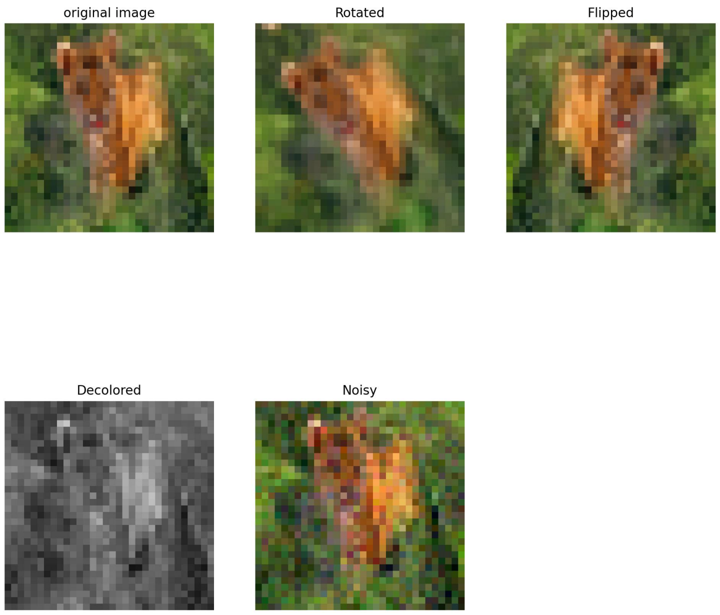

Figure 1: standard augmentations

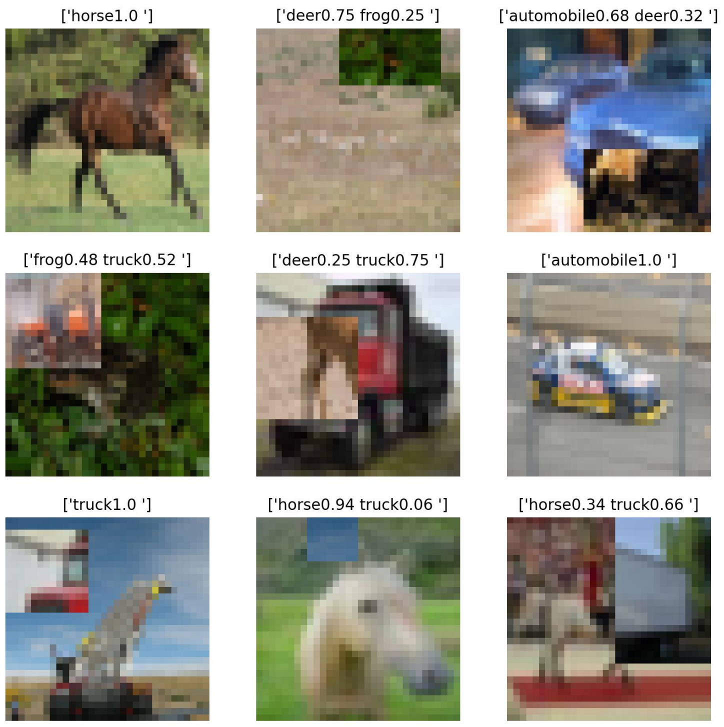

Figure 2: cutmix augmentation

5 Conclusions

We have done different types of experiments to try to achieve the best model possible. Both the hyperparameter tuning, and data augementation increased the accuracy of the models, as we have predicted. The best model by a landslide was the ResNet50 architecture, with tuned in hyperparameters. Unfortunately, training every layer increased training time per epoch at least threefold. The most surprising part, was that the pretrained network did not work as good as fully trained ones. Therefore our final conclusion is that even when a network is pretrained on similar dataset, there should be done some interference if the actual dataset is different.