DL Project 1 report

Jedrzej Ruciński, Adam Czerwoński

March 2024

Contents

1 Introduction

The goal of this project is to create an image classification model for identifying 10 different classes of photos coming from the CINIC 10 data set. The data consists of three sets, training, validation, and test, each of which contains 90000 32 by 32-pixel images For this deep learning task, we focus on using Convolution Neural Networks (CNNs) and explore their different architectures and hyperparameters. Moreover, we consider different data augmentation techniques to increase the volume of our training set. The project will be carried out in Python, with the use of TensorFlow for creating the models.

1.1 Data

As mentioned before, due to the data volume being very large, we use TensorFlow batches for storing it in Python. These batches are of size 32, meaning they contain 32 images which are stored in a `numpy ndarray` of size $(32, 32, 3)$ representing the height, width, and color of the pictures. To improve the training process we also scale the values of pixels to a $0 - 1$ range.

Below we may see an example image from each of the 10 classes in CINIC 10. For this visualization, the images have been rescaled to a larger size to improve clarity.

Figure 1: CINIC-10 examples for each class.

2 Residual Model

The first model we used for this task is a variant of the ResNet (Residual Network) model [He et al., 2015]. ResNet is famous for its use of residual blocks, which helps to mitigate the vanishing gradient problem and allows for training much deeper neural networks effectively.

We use standard residual blocks which are created by the `residual_block` function. Within each block, two convolutional layers are applied with batch

normalization and ReLU activation after each convolution. The residual connection skips one or more layers and directly adds the input to the output of the second convolutional layer within the block. This allows the network to learn residual mappings, which are more straightforward for optimization.

The final network is created by the `create_resnet_model` function which combines three residual blocks with pooling layers. To prevent overfitting, we also added dropout layers with a dropout rate of 0.2, at the end of each residual block. Finally, for the training process, we use two callback methods:

1. *Early stopping* - which stops model training after validation loss does not decrease after a number of epochs.
2. *Reduce learning rate on plateu* - which reduces the optimizer's learning rate when the validation loss plateaus.

The first training run for this neural network was done on 20 epochs.

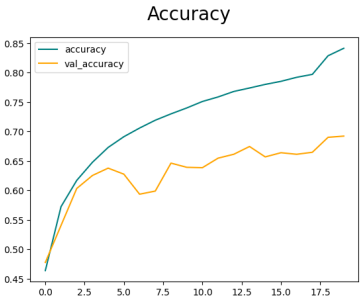

Figure 2: Accuracy over epoch iterations for resnet

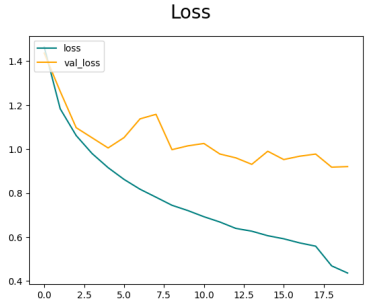

Figure 3: Loss function epoch iterations for resnet

This network was also able to achieve an accuracy of 0.6916222 on the testing data

3 Hyperparameter Influence

This section of the report dives into the process of conducting a hyperparameter grid search for optimizing a CNN architecture. Through experimentation and evaluation, we aim to identify the hyperparameter configuration that maximizes the network's performance metrics.

We divide this hyperparameter tuning into two parts. One regarding the training of the model, and another regarding regularization that is applied in the output layer of the network.

These tests will be conducted on our handmade ResNet model, created by functions `create_resnet_model` and `residual_block`. For the regularization part, we perform Elastic Net regularization [Zou and Hastie, 2005], written in the python class `ElasticNetRegularization`.

3.1 Training hyperparameters

For the training parameter grid search, we consider two parameters.

1. *Number of residual blocks*, which just corresponds to how many of these blocks we implement in this network. For this parameter, we test out values $1, 3$, and, 5
2. *Kernel Size*, which influences the convolutional layers inside the residual blocks. Kernel size corresponds to the dimensions of the square kernel, or filter applied in the layer. The values we test are $2, 3, 4$, and, 5.

First, let us take a look at the individual impact these parameters have on the validation accuracy of the network.

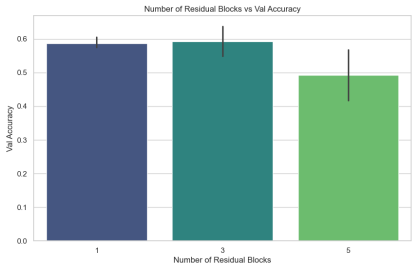

Figure 4: Validation accuracy distribution for each number of residual blocks

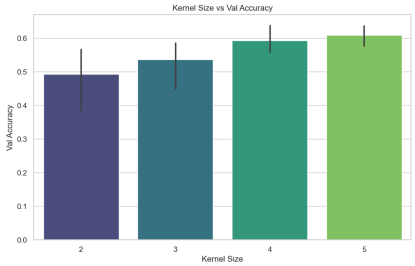

Figure 5: Validation accuracy distribution for each kernel size

From these plots, we may infer that for our data and our ResNet model, either one or three residual blocks are the optimal value. Along with this, we see that larger filter, or kernel, sizes result in higher accuracy. Let us now see how a combination of these parameters influences validation accuracy.

The heatmap confirms that a combination of a lower number of residual blocks, along with higher kernel sizes results in the best results on the validation data.

3.2 Regularization hyperparameters

For this grid search, we explore three different parameters that were used for the network to reduce overfitting. The first two, *alpha* and *lambda* are parameters passed when initializing the `ElasticNetRegularization` class. They control the balance between the l_1 and l_2 losses, which form our regularization loss.

$$l_{reg} = \lambda * (\alpha * l_1 + (1 - \alpha) * l_2) \tag{1}$$

The final hyperparameter is the dropout rate passed to each dropout layer in the network. The grid is then as follows:

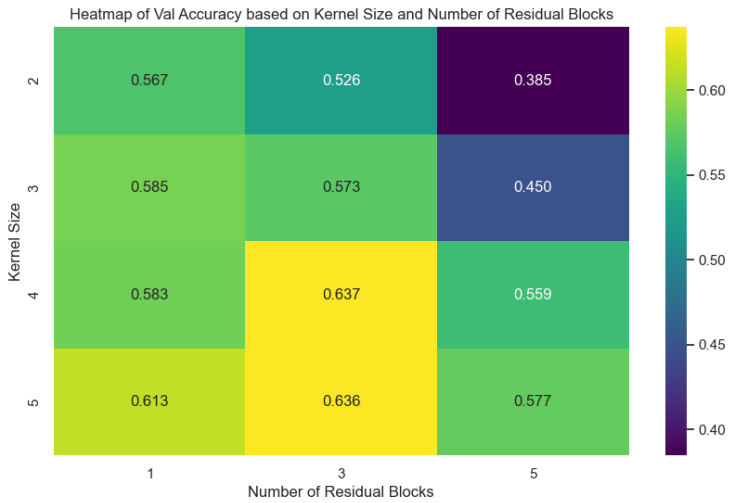

Figure 6: Heatmap for hyperparameters regarding the training process.

- $\alpha \in 0.2, 0.5, 0.8$
- $\lambda \in 0.001, 0.01, 0.1$
- $droprate \in 0.1, 0.3, 0.5$

Below we may see visualizations of how these parameters impacted the validation accuracy of our network. Firstly, let us once again analyze individual hyperparameter influences on the model.

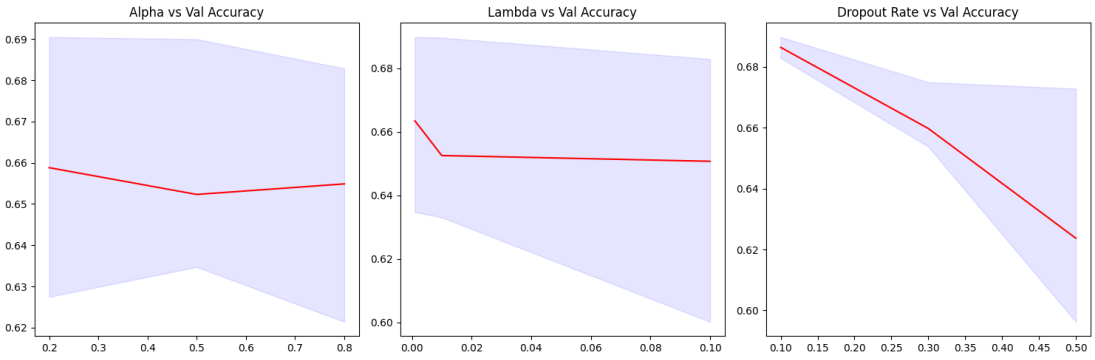

Figure 7: Mean val accuracy values along with interquartile ranges for each parameter.

The biggest impact on accuracy is observed with the change in the dropout

rate, however, accuracy also slightly decreases with larger lambda values. Below we also have a 3-dimensional grid showing how combined parameter values impact validation accuracy.

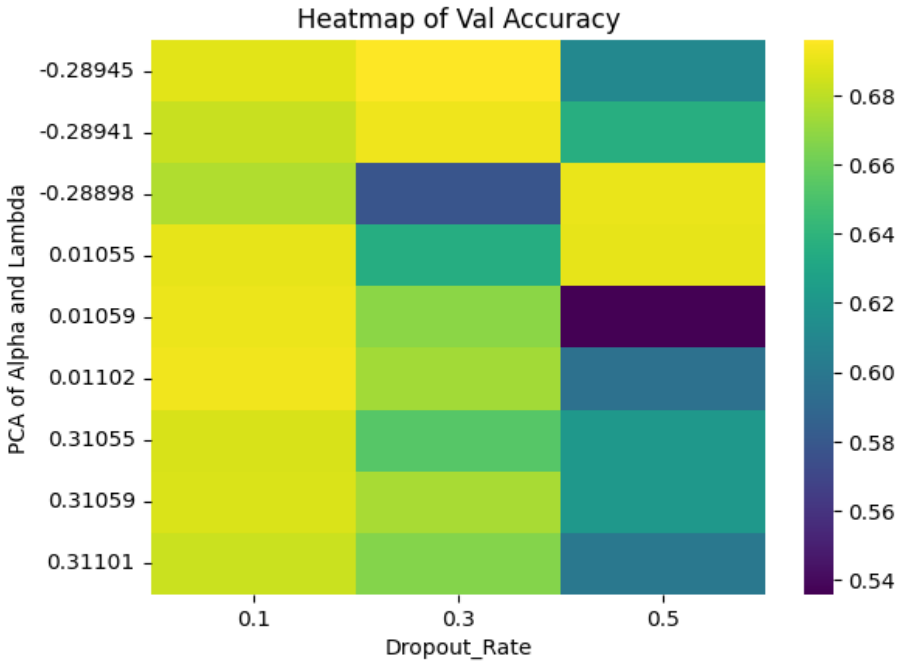

Figure 8: Combined parameter impact on validation accuracy.

4 Data augmentation

In this section we will investigate the influence of data augmentation on the accuracy of a simple model. Augmentation is a way of generating new data by manipulating the original one. It's classified as a regularization technique, because it is mainly used for dealing with overfitting. We tried the following augmentations:

- brightness - randomly increasing or decreasing brightness,
- flip - randomly flipping horizontally or vertically,
- rotation - randomly rotating to the left or to the right by less than 90 degrees,
- zoom in - randomly zooming in up to 50%,

- cut out - cutting random part of the picture.

Figure 9: Augmentations.

Firstly, we have trained a very simple model (3 convolutional layers, max pooling and three dense layers) on the original data. Then, for each type of augmentation, we have created a new model trained on the augmented dataset, which was twice as big. In the end, we got the following results:

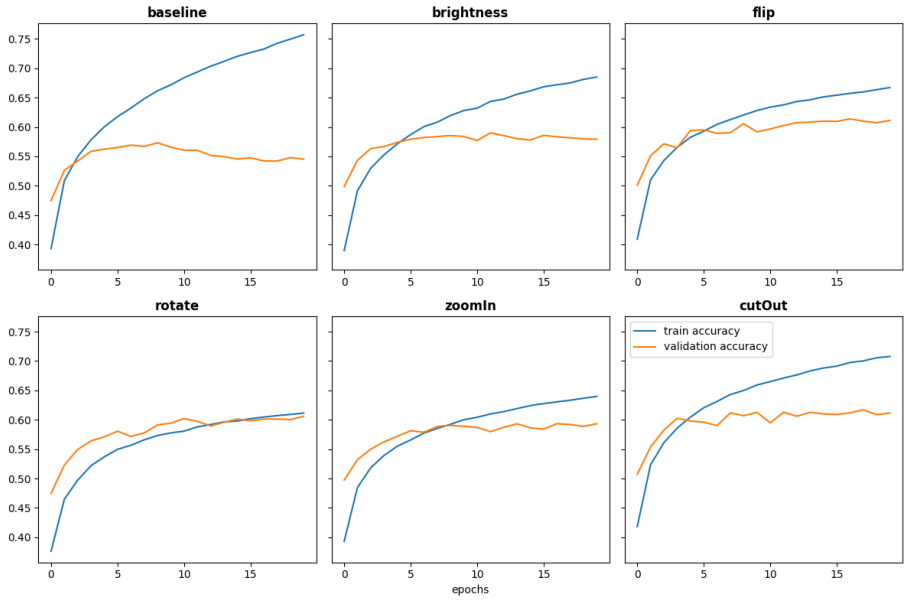

Figure 10: Accuracy over epoch iterations for each augmentation.

As expected, augmentation decreased overfitting and gave a little improvement in the accuracy.

5 Transfer learning

Transfer learning is a popular method for training neural networks that uses already pre-trained models. We not only copy the architecture of a recognised model but we actually use the same weights that were used on a different dataset. More precisely, we start with the same weights and then adjust them a little bit so that they are more tailored to our classification problem. This way we exploit the fact that the convolutional layers often times learn to detect general things like contour lines of an object.

5.1 Choosing the base model

Firstly, we had to choose the base model. We decided to use VGG19 model [Simonyan and Zisserman, 2014] which is a very simple but deep architecture consisting of convolutional layers and max pooling. On the top of it we have added three dense layers with 512 neurons each, then dropout with 0.4 rate and at the end dense layer for returning the predictions. Besides that we have used simple data augmentation.

5.2 Training

Having chosen our base model, we could start training it. Initially we only trained the top layer and the weights from VGG19 were frozen.

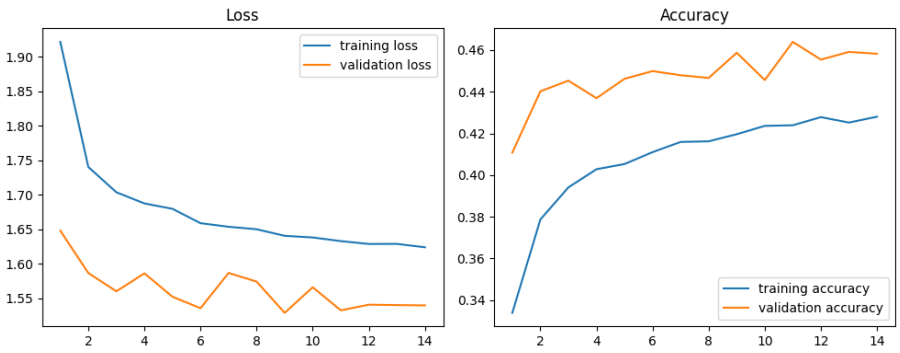

Figure 11: Accuracy and loss over epoch iterations for initial training.

The results aren't stunning but that was expected. Next step was to unfreeze VGG19 weights and train the whole model, however with very small learning rate (10^{-5}). This way we could preserve what the convolutional layers have learned on the previous dataset.

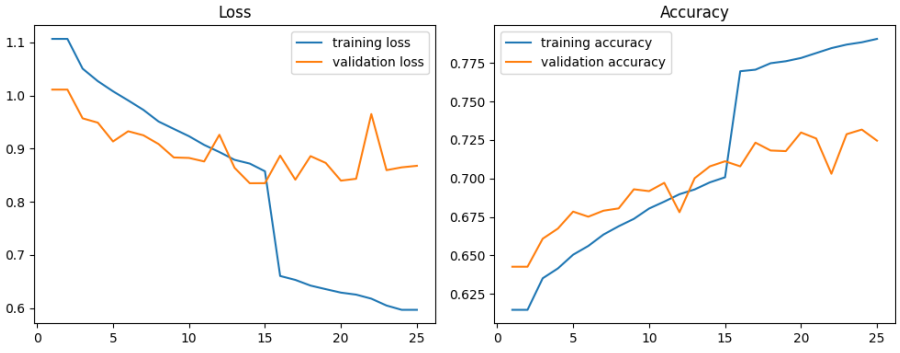

Figure 12: Accuracy and loss over epoch iterations.

We can see a significant improvement in the accuracy. After just the first epoch, we experienced a 15 percentage point increase. For the final test we used our model on the test set and got 72.48% accuracy which is the best result we have achieved throughout this project.

References

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition.

[Zou and Hastie, 2005] Zou, H. and Hastie, T. (2005). Regularization and Variable Selection Via the Elastic Net. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67(2):301–320.