**Ex3 Report**

**Part 1:**

In this part I was asked to visualize the data. Using matplotlib, I plotted 4 examples per class:



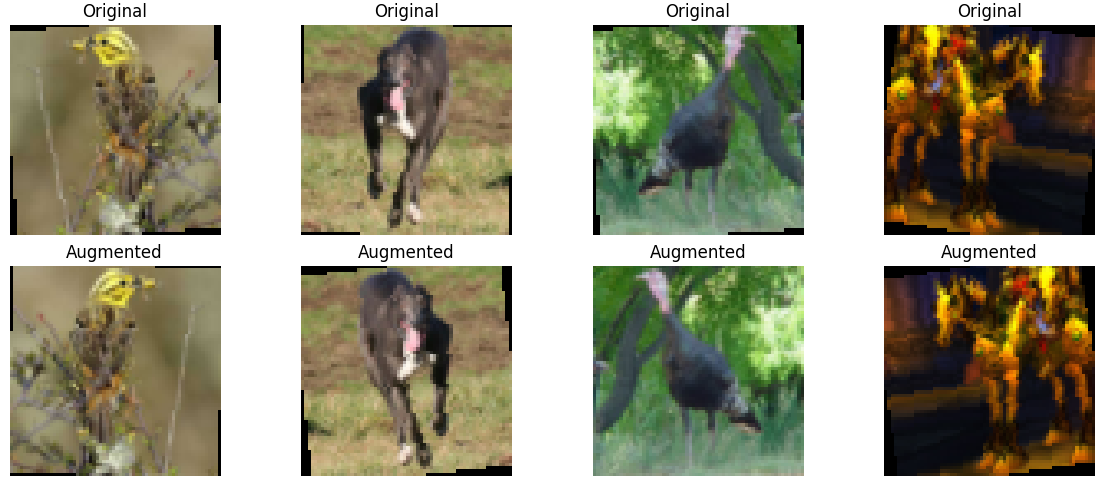
**Part 2:**

In this part I was asked to use several architectures to train a model on the data.

To do so, I use these augmentations:

* Random horizontal flip
* Random rotation of up to 10 degrees to the right

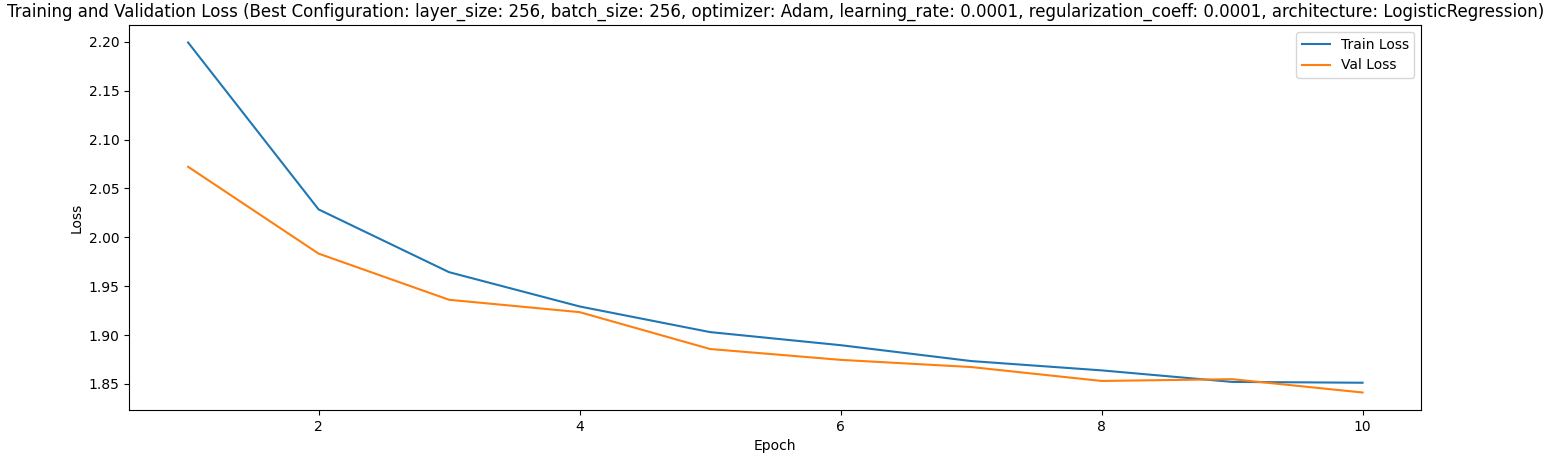
Here are some examples for images before and after augmentation:



I ran the training with 50 different configurations, 10 for each architecture, which were sampled randomly from these options:

* Layer size: 32, 64, 128, 256, 512
* batch size: 32, 64, 128, 256
* optimizer: Adam, Stochastic Gradient Descent
* learning rate: 0.1, 0.01, 0.001, 0.0001
* regularization: 0.1, 0.01, 0.001, 0.0001

here are the plots of my best models’ loss and accuracy:  
Logistic Regression:



תמונה שמכילה טקסט, קו, צילום מסך, עלילה

התיאור נוצר באופן אוטומטי

Full-Connected NN:

תמונה שמכילה טקסט, צילום מסך, קו, עלילה

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט, קו, צילום מסך, עלילה

התיאור נוצר באופן אוטומטי

CNN:

תמונה שמכילה קו, טקסט, צילום מסך, עלילה

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט, קו, עלילה, צילום מסך

התיאור נוצר באופן אוטומטי

Fixed MobileNet:

תמונה שמכילה טקסט, קו, לוח כתיבה לבן, צילום מסך

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט, קו, עלילה, צילום מסך

התיאור נוצר באופן אוטומטי

Learned MobileNet:

תמונה שמכילה טקסט, קו, עלילה, צילום מסך

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט, קו, עלילה, תרשים

התיאור נוצר באופן אוטומטי

These are my insights about the hyperparameters:

* High layer size allows the model to capture more complex patterns in data and might cause overfitting. Low layer size might cause underfitting the same way.
* Lower batch size might lead to slower convergence, but also to better performance on unfamiliar data.
* Higher learning rate might lead to faster convergence but can also skip optimal solutions.
* If the regularization coefficient is too low, the model would learn the training data too closely. If it’s too high, the model would be too simple to capture the required patterns.
* Optimizer: Adam tends to converge more quickly which can cause missing the optimal solution, while SGD tends to converge more slowly compared to Adam, but it can lead to finding better solutions.

Number of parameters:

Network 2:

I chose a hidden layer size of 256, which gives us,

Network 3:

**Instructions on how to run the code:**

The code contains two files:

* Part1.py, which visualizes the data.
* Part2.py, which does the training

For part 1, you can simply run the code in Part1.py, for example using:

py Part1.py

and this will plot the 4 examples of images per class.

For part 2, you need to run Part2.py. you will be prompted to choose between 3 options:

1. Train on different configurations
2. Train once on a chosen architecture
3. Plot examples of augmented images

If you choose option 1, the program will start the training of 50 different configurations (10 for each architecture), 10 at a time (using multithreading). This process will take a lot of time and can be stopped by giving ‘y’ as an input. If you stop the program in the middle of the processing, it will show the results of the models that has been processed so far.

If you choose option 2, you will be prompted to choose an architecture, and afterwards it will train one model according to the architecture and default hyperparameters.

If you choose option 3, examples of images and their augmented versions will be plotted.