Student: Radu Madalin-Cristian

Group: 507

Generated Image Classification
Deep Learning

1. Introduction

Objective

The task is to train deep classification models on a data set containing images generated by deep generative models. For each test image, we have to predict the class label.

The evaluation metric is the macro F1 score, which is given by the mean of the F1 scores computed for each class.

Overview of approaches

For this task, I chose to implement a simple neural network and a convolutional neural network that has an architecture similar to the ResNet 34 but scaled down.

2. Data preprocessing and augmentation

The Data

We are given 20000 images, of which 13000 are reserved for the training set, 2000 are reserved for the validation set and 5000 are reserved for the test set. The training set and validation set each come with their respective labels, while we will be judged based on the predictions on the test set.

Data alteration

I loaded the images and labels into numpy arrays and randomly shuffled them to introduce some chaos into the input data. I also normalized the images and one-hot encoded the labels so that the networks can have proper outputs.

In order to prevent overfitting and to increase the f1 score on the validation set, I augmented the images by using an ImageDataGenerator from tensorflow with which I randomly rotated, shifted, zoomed and horizontally flipped the images. These actions are performed randomly on the images given to us.

3. Model architectures

As stated before, I tried to implement a simple neural network (an MLP) and a convolutional network with a scaled down resnet architecture.

- For the MLP, I decided to start with a simple architecture consisting of these layers:
    - Flatten layer -> to bring the images to one dimension.
    - Fully connected layer with 24 neurons
    - Fully connected layer with 48 neurons
    - Dropout Layer for regularization
    - Fully connected layer with 100 neurons and softmax activation function as the output layer
- For the convolutional neural network, I started from a ResNet architecture and I scaled it down. This architecture consists of multiple ResNet blocks which have the following structure:
    - 3x3 convolution layer
    - Batch normalization
    - ReLU activation
    - 3x3 convolution layer
    - Batch Normalization
    - Layer in which we either add a 1x1 convolution with the input to the block to the output of the last layer or we add the input directly to the last layer

The architecture is the following:

- Input layer
- 7x7 convolution layer
- Batch normalization
- ReLU activation
- Pooling layer
- For nb_filters in [64, 128, 256]:
    - Resnet block with skip
    - Resnet block.
- Fully connected layer with 32 neurons
- Pooling layer
- Output layer

The MLP model had very bad performance: 0% precision and recall, but the convolutional neural network had very promising performance: around 30% f1 score on the validation set after 20 epochs. It's easy to say that these models can be improved. To do that, first I did some hyperparameter tuning with grid search and then I altered the architectures.

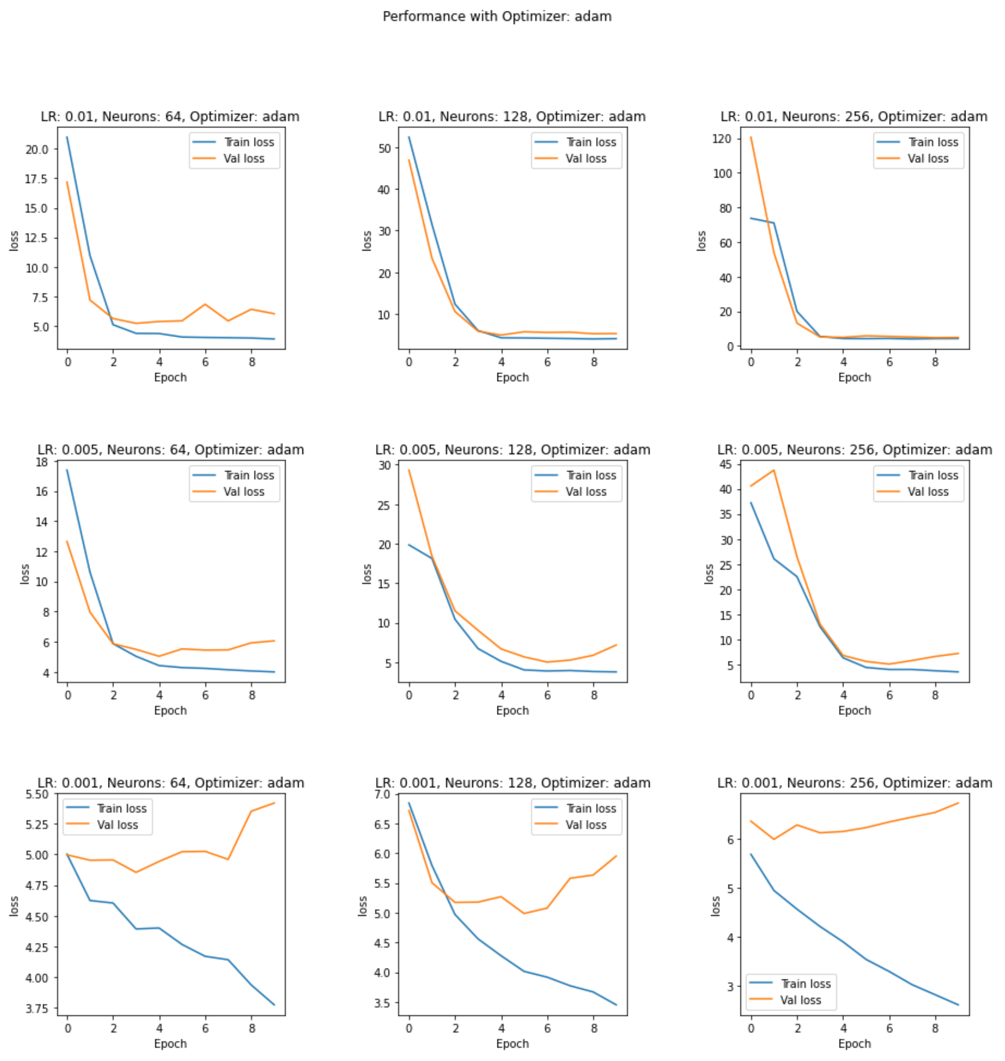
4. Hyperparameter Tuning

a) Hyperparameter tuning for MLP

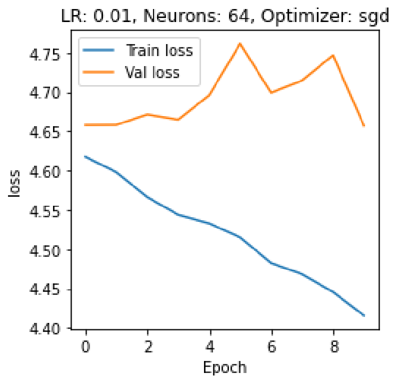
For the MLP, I chose to do a grid search over the number of neurons, the learning rate and the optimizer. The possible values for these hyperparameters are:

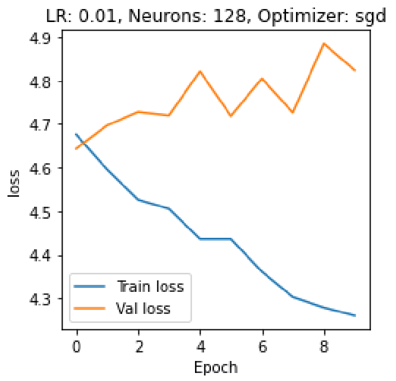
- Neurons = [64, 128, 256]
- Learning rate = [0.01, 0.005, 0.001]
- Optimizer = ['adam', 'sgd', 'rmsprop']

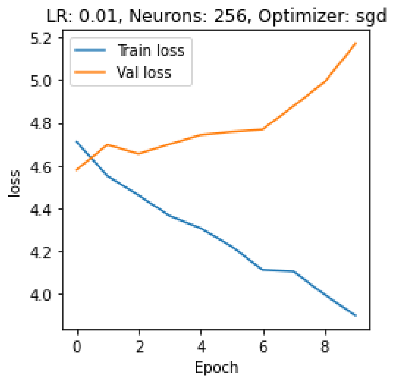
These are the results:

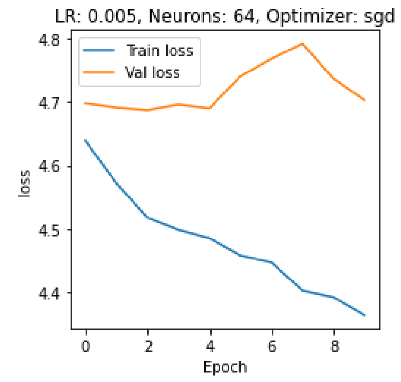


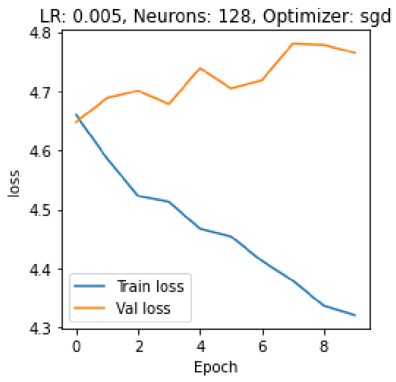
Performance with Optimizer: sgd

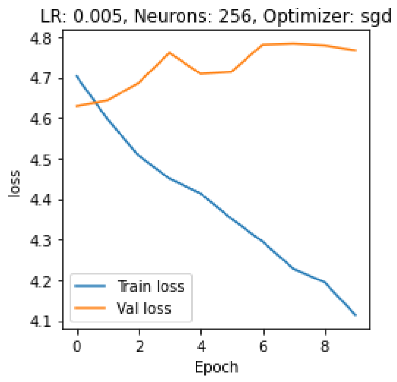


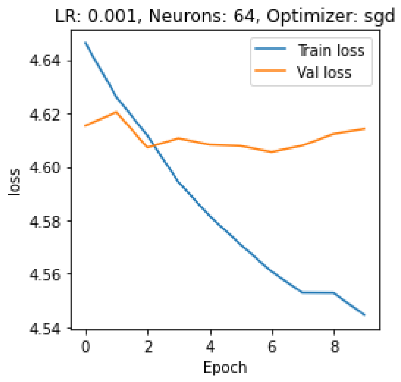


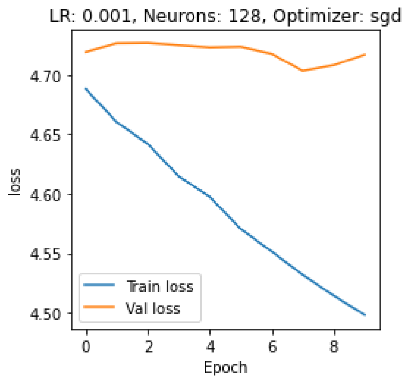


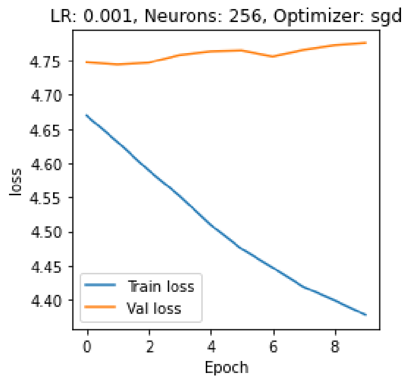


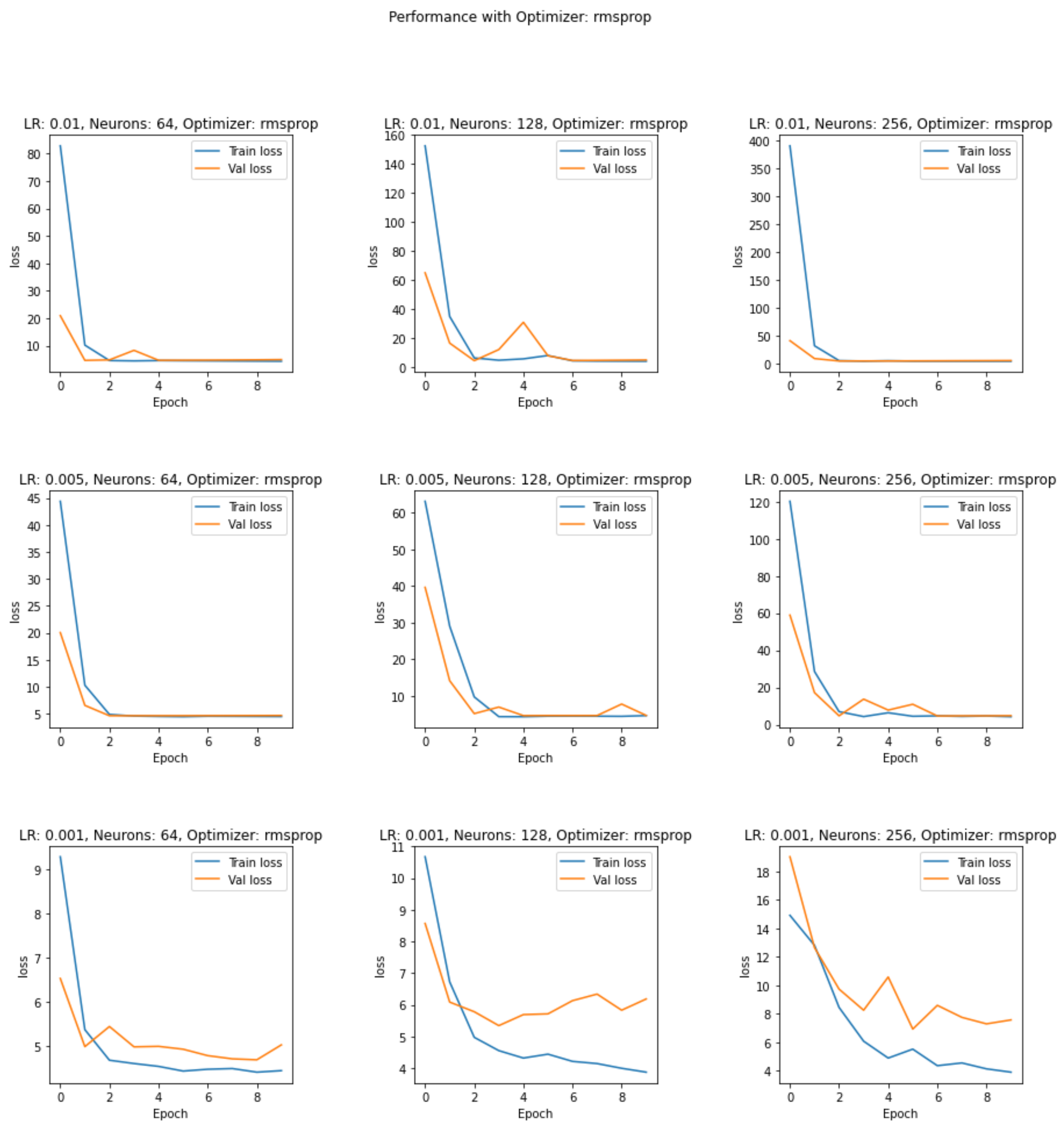








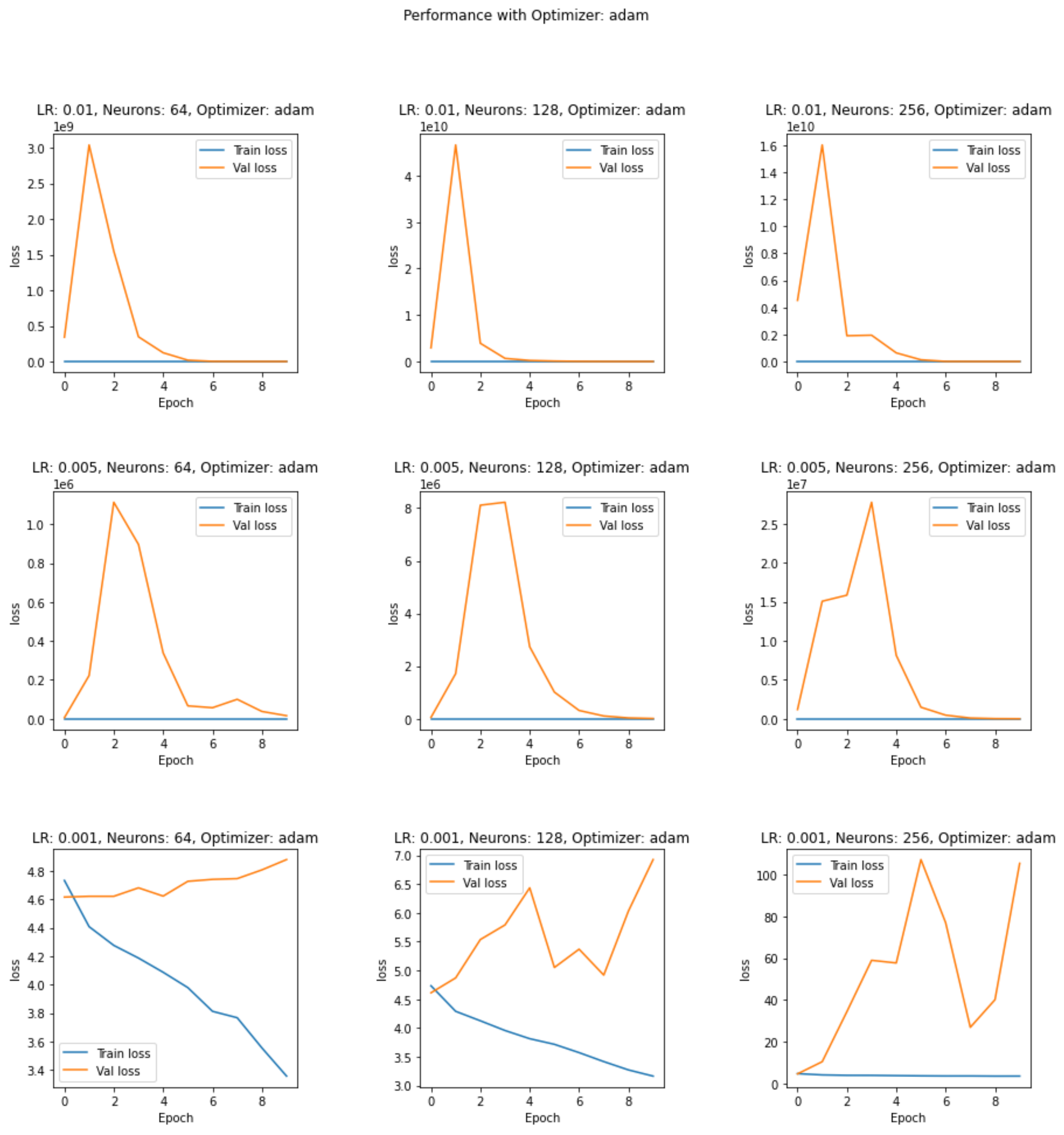


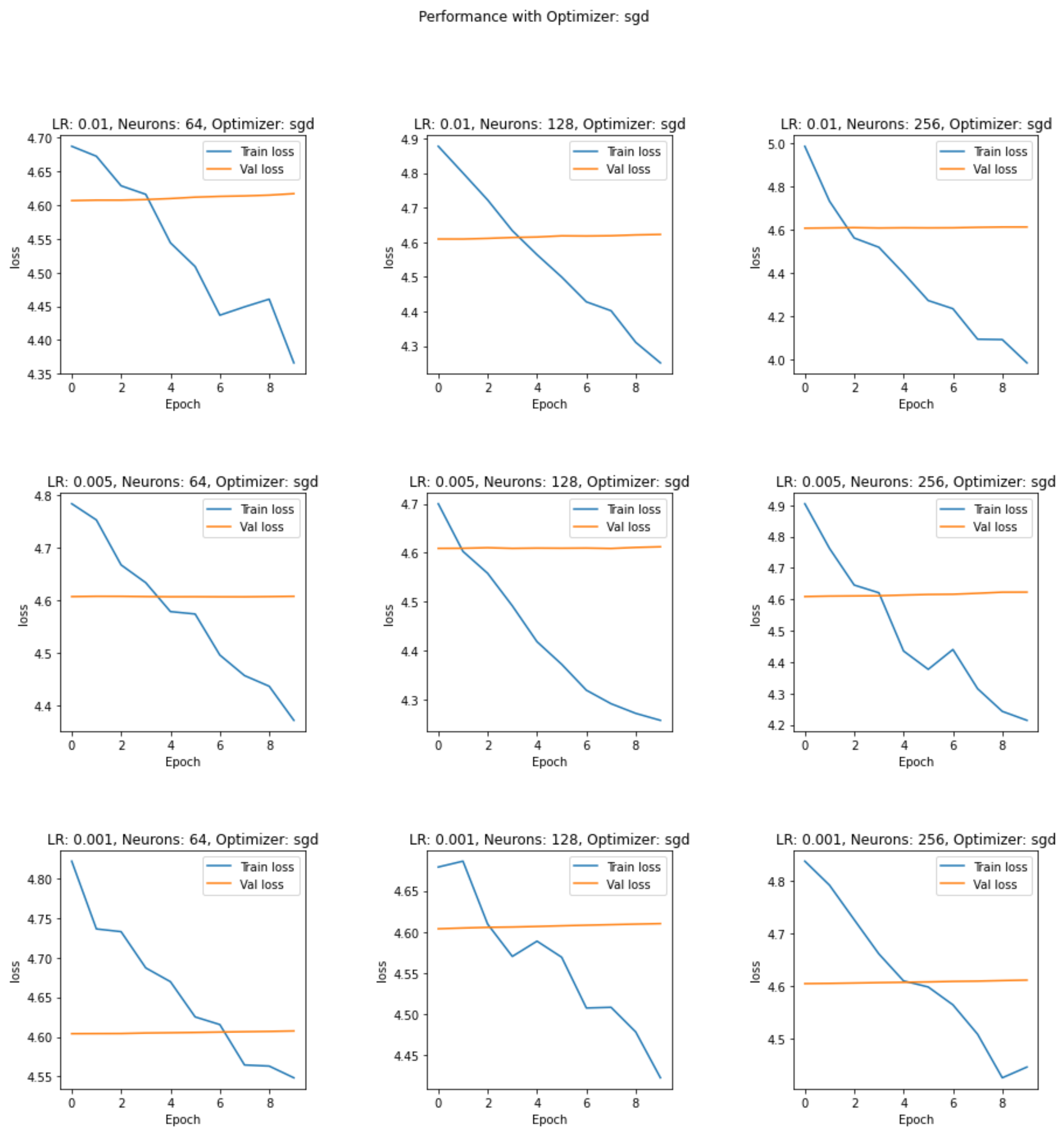


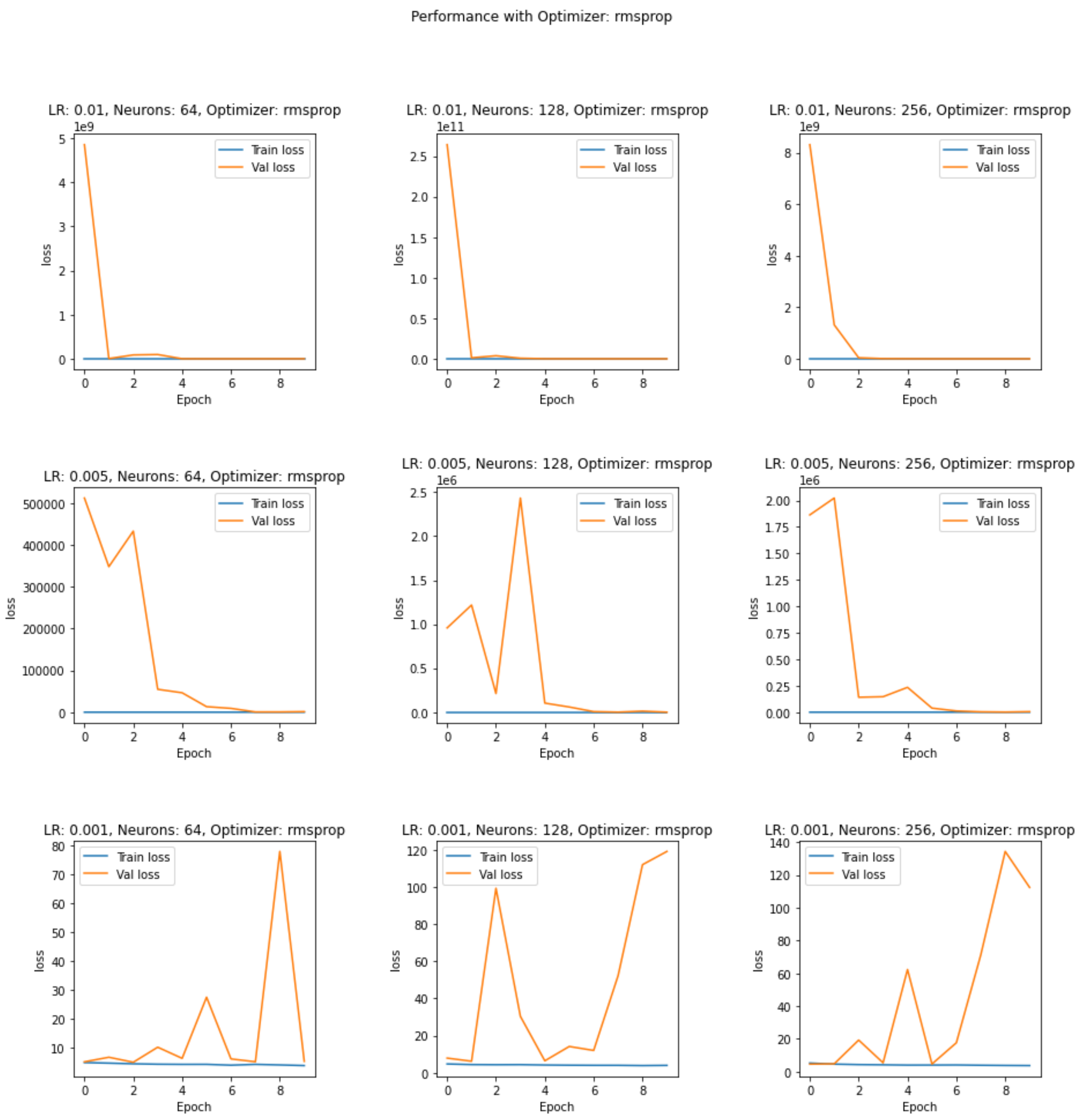
It is plain to see that the validation loss is locked at about 5, which means that the architecture is blocking the improvement. It is pretty obvious that we need to change the architecture in order to improve the performance. On that note, we will continue with the convolutional neural network from now on.

b) Hyperparameter tuning for CNN

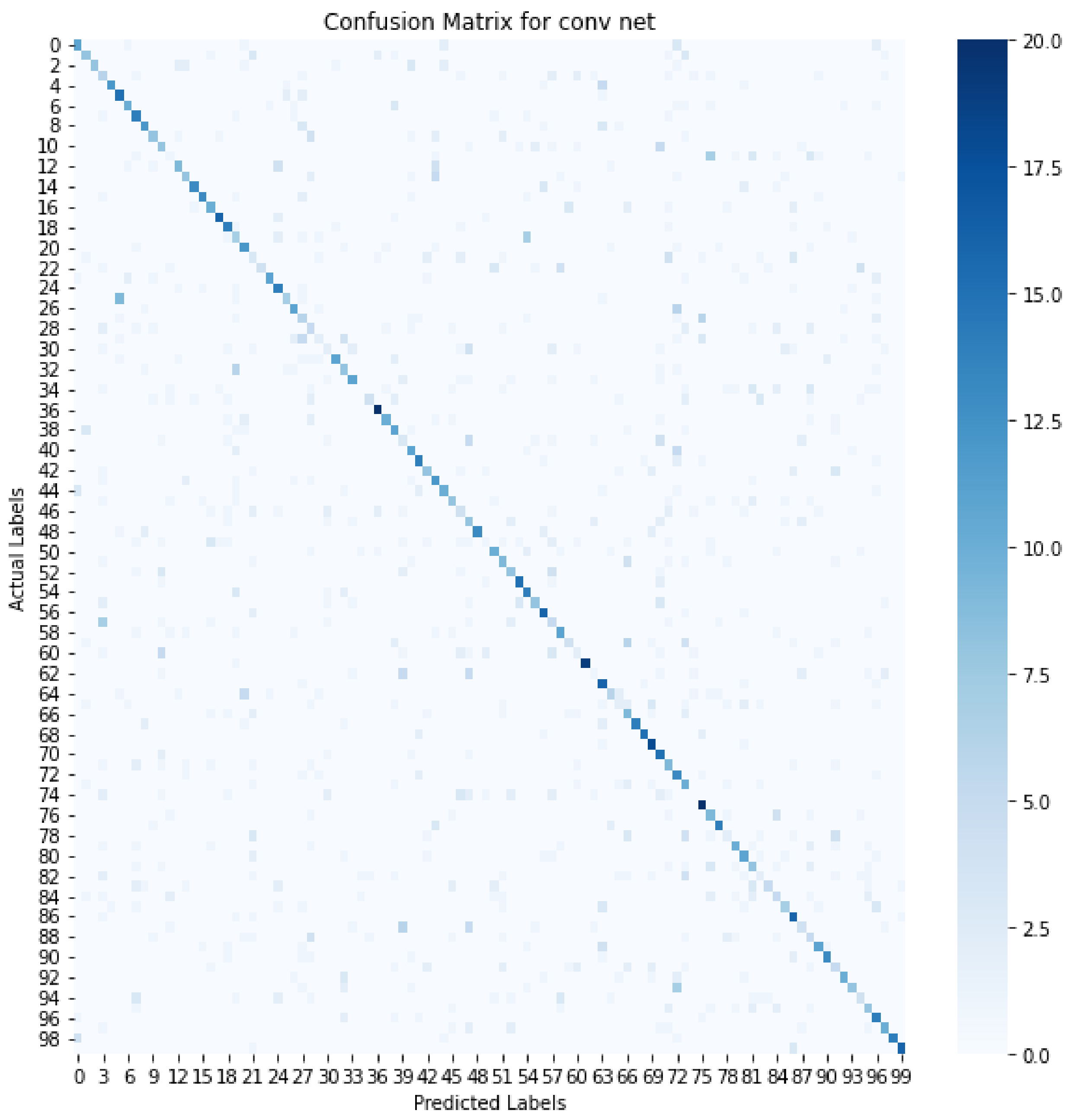
For this architecture, I chose to add a fully connected layer after each resnet block and I optimized that layer's number of neurons using grid search. I also did grid search over the learning rate and the optimizer. Here are the results:



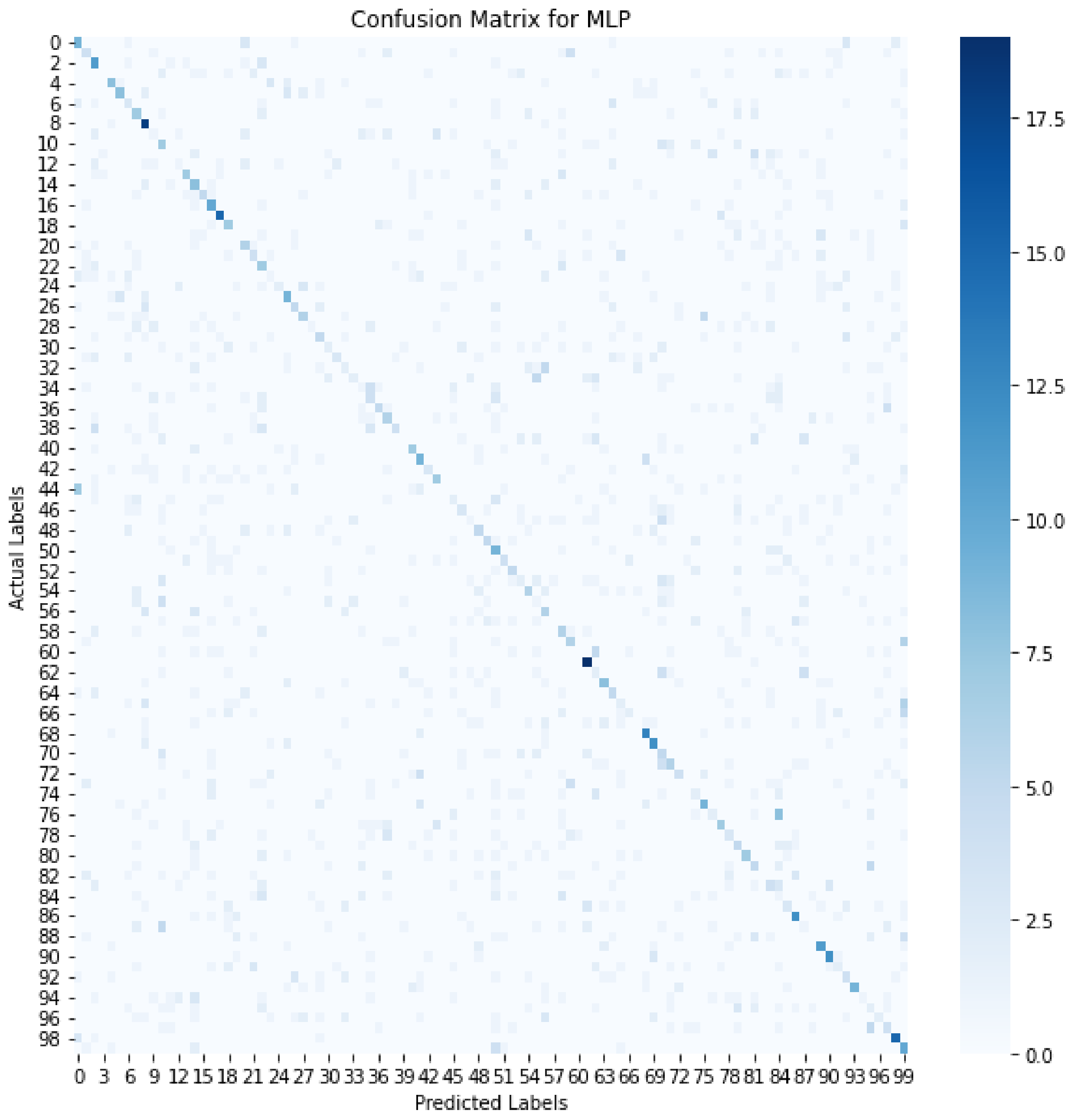

Performance with Optimizer: sgd
LR: 0.01, Neurons: 64, Optimizer: sgd
LR: 0.01, Neurons: 128, Optimizer: sgd
LR: 0.01, Neurons: 256, Optimizer: sgd
LR: 0.005, Neurons: 64, Optimizer: sgd
LR: 0.005, Neurons: 128, Optimizer: sgd
LR: 0.005, Neurons: 256, Optimizer: sgd
LR: 0.001, Neurons: 64, Optimizer: sgd
LR: 0.001, Neurons: 128, Optimizer: sgd
LR: 0.001, Neurons: 256, Optimizer: sgd
Train loss
Val loss
loss
Epoch



On first glance, it seems like RmsDrop and Adam are the most suitable, but if you look closely, you will see that the figures are scaled, and you can observe in the plots with the smallest learning rates that RmsDrop and Adam yield increasing validation loss. The validation loss still increases with SGD, but it is only a slight increase, and the performance is the most consistent, locking at about 4.6. The fully connected layer did not seem to affect the model's performance, while the learning rate only seemed to affect the time it takes to converge. Given these considerations, the final model is the same as the initial one, except I added a fully connected layer after **all** the resnet blocks and also added a dropout layer with 50% rate for regularization purposes. I trained this model for 20 epochs and obtained improved results. This is the confusion matrix of the model after training:



This model's validation f1 score is about 47% and you can see that some classes are very poorly predicted. Still, this performance is acceptable. The MLP's performance is much worse; it plateau-ed after 60 epochs at f1 score = 21%. These result do not indicate a valid model, but I will add the confusion matrix for this model regardless:



5. Conclusion

The results indicate that the best model is the convolutional one, which yielded an f1 score of 50% on the test data, but this model could vastly be improved. The best options for improvement are to either change the model's architecture or to do different data augmentation on the training data.

6. Bibliography

- https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8