Documentation Deep Learning project
Generated Image Classification

Dragomir Elena Alexandra (507)

1. Purpose

The task is to classify deep generated images into 100 classes. The training was performed on 13000 RGB images of shape 64 x 64 and validation on 2000 images.

2. Models

a. CNN models

The first model approached was a simple CNN architecture, composed of sequences of convolutional layers, batch normalization and ReLU activation function. At the end there is a dropout layer. During experiments I changed the optimizer, the number of sequences, values of filters, kernels, strides, paddings and dropout. Below there is an example of the described model for the following values:

6 blocks, filters = [64, 64, 128, 128, 128, 128], kernels = [7, 5, 5, 5, 5, 5], strides = [2, 2, 1, 1, 1, 1], paddings = [3, 3, 1, 0, 1, 0], dropout = 0.5

```
Input (3 channels)
|
|---- Conv2d(3, 64, kernel=(7, 7), stride=(2, 2), padding=(3, 3)) ---- BatchNorm2d ---- ReLU
|---- Conv2d(64, 64, kernel=(5, 5), stride=(2, 2), padding=(3, 3)) ---- BatchNorm2d ---- ReLU
|---- Conv2d(64, 128, kernel=(5, 5), stride=(1, 1), padding=(1, 1)) ---- BatchNorm2d ---- ReLU
|---- Conv2d(128, 128, kernel=(5, 5), stride=(1, 1)) ---- BatchNorm2d ---- ReLU
|---- Conv2d(128, 128, kernel=(5, 5), stride=(1, 1), padding=(1, 1)) ---- BatchNorm2d ---- ReLU
|---- Conv2d(128, 128, kernel=(5, 5), stride=(1, 1)) ---- BatchNorm2d ---- ReLU
|
|---- Flatten
|---- Dropout(0.5)
|---- Linear(3200, 100)
```

b. Resnets

The next model is an adaption of Resnet18 with a customizable number of layers, blocks per layer, filters and strides. During training we will test optimizers Adam and SGD, fine tune learning rate, momentum, and perform random search over the number of filters, strides and number of blocks for each layer. The architecture that reached the best validation accuracy is shown below:

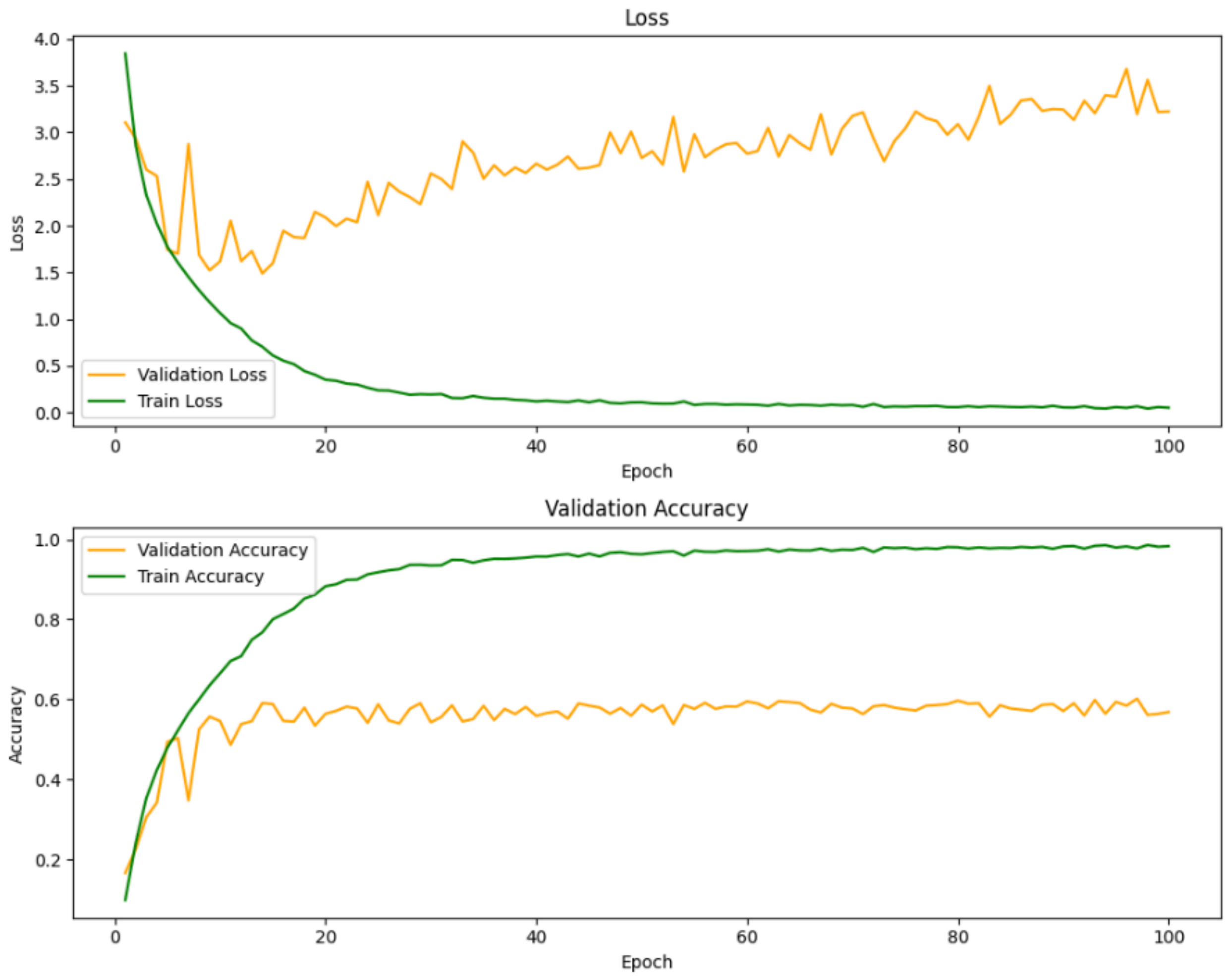
```
Input (3 channels)
    |
    |---- Conv2d(3, 64, kernel=(3, 3), stride=(1, 1), padding=(1, 1)) ---- BatchNorm2d ---- ReLU
    |
    |---- Blocks Group 0:
    |    |---- Block 0: Conv2d(64, 64) ---- BatchNorm2d ---- ReLU
    |    |---- Block 1: Conv2d(64, 64) ---- BatchNorm2d ---- ReLU
    |
    |---- Blocks Group 1:
    |    |---- Block 0: Conv2d(64, 128, stride=(2, 2)) ---- BatchNorm2d ---- ReLU
    |    |        |---- Downsample: Conv2d(64, 128, kernel=(1, 1), stride=(2, 2))
    |    |---- Block 1: Conv2d(128, 128) ---- BatchNorm2d ---- ReLU
    |
    |---- Blocks Group 2:
    |    |---- Block 0: Conv2d(128, 256, stride=(2, 2)) ---- BatchNorm2d ---- ReLU
    |    |        |---- Downsample: Conv2d(128, 256, kernel=(1, 1), stride=(2, 2))
    |    |---- Block 1: Conv2d(256, 256) ---- BatchNorm2d ---- ReLU
    |
    |---- AdaptiveAvgPool2d(output_size=(1, 1))
    |
    |---- Linear(in_features=256, out_features=100)
```

3. Experiments

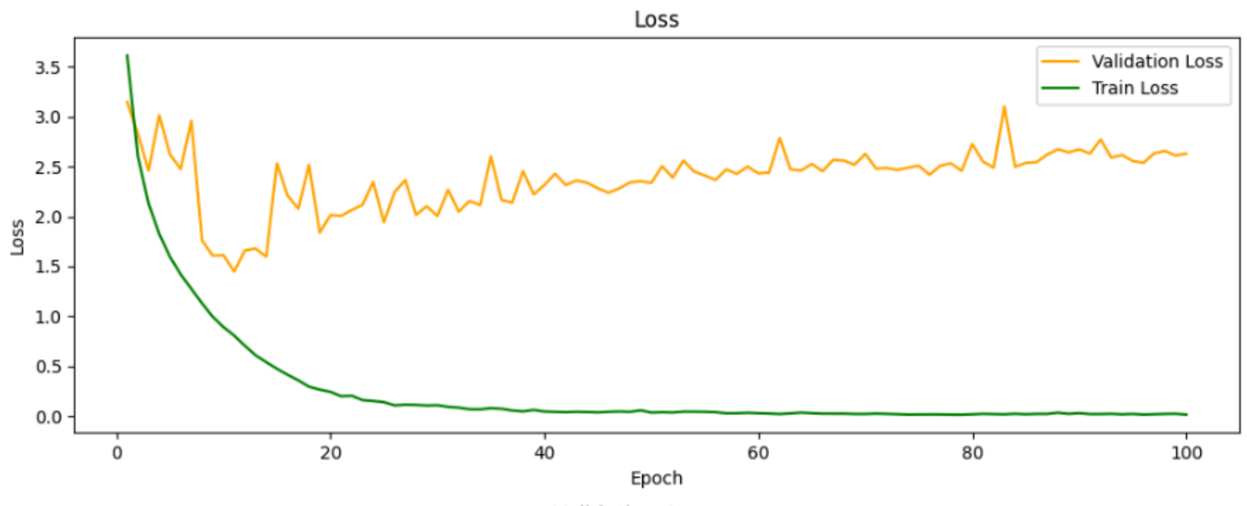
3.a. CNN models

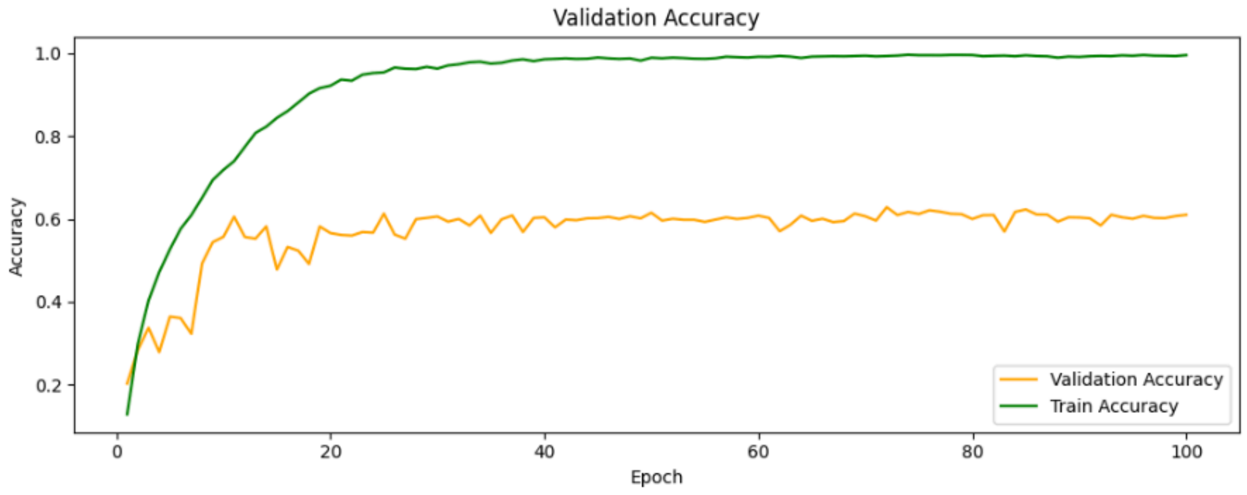
All training sessions had batch_size=32.

- 6 blocks, filters = [64, 64, 128, 128, 128, 128], kernels = [7, 5, 5, 5, 5, 5], strides = [2, 2, 1, 1, 1, 1], paddings = [3, 3, 1, 0, 1, 0], dropout = 0.5, Adam optimizer with lr=0.001, 100 epochs. **Max val accuracy: 60.20%** (epoch 97)



- 6 blocks, filters = [64, 64, 128, 128, 128, 128], kernels = [7, 5, 5, 5, 5, 5], strides = [2, 2, 1, 1, 1, 1], paddings = [3, 3, 1, 0, 1, 0], dropout = 0.5, SGD optimizer, lr = 0.01, momentum = 0.8, 100 epochs. **Max val_acc = 62.85%** (epoch 72)





- 6 blocks, kernels = [7, 5, 5, 5, 5, 5], strides = [2, 2, 1, 1, 1, 1], paddings = [3, 3, 1, 0, 1, 0], Adam optimizer and perform random search over the combination of filters (with the condition of keeping them in ascending order), the dropout and the learning rate

Ranges for the hyperparameters:

- Filters in [64, 128, 256] - arrays of 6 in ascending order
- Dropout in (0.3, 0.8)
- Learning_rate in (0.0001, 1)

There were trained 14 combinations of these hyperparameters over 15 epochs each and evaluated using the maximum accuracy on the validation data (correlated with the accuracy on training data). The results on epochs 5, 10, 15 and the maximum accuracy were as follows, sorted in descending order in regards of maximum validation accuracy:

Hyperparameters **Filters, dropout, lr**	**Val_acc / acc** **epoch 5**	**Val_acc / acc** **epoch 10**	**Val_acc / acc** **epoch 15**	**Maximum** **val_acc**

filters=(64, 64, 64, 256, 256, 256)
dropout=0.8 lr=0.001
	31/34	53/52	59/63	59

filters=(128, 128, 256, 256, 256, 256)
dropout=0.68 lr=0.0018
	42/41	54/60	53/73	57

filters=(64, 128, 128, 128, 128, 128)
dropout=0.3, lr=0.0001
	46/61	48/92	54/97	56

filters=(64, 64, 64, 64, 128, 256)
dropout=0.63 lr=0.0006
	27/48	53/65	55/20	56

filters=(128, 128, 128, 128, 256, 256)
dropout=0.3 lr=0.0018
	41/47	51/68	51/85	55

filters=(64, 64, 64, 128, 128, 128)
dropout=0.3 lr=0.0127
	25/24	40/41	50/50	50

filters=(64, 128, 128, 128, 256, 256)
dropout=0.8 lr=0.012
	10/5	15/9	15/13	21

filters=(64, 64, 64, 128, 128, 256)
dropout=0.8 lr=0.020
	1/1	1/1	1/1	1

filters=(64, 256, 256, 256, 256, 256)
dropout=0.63 lr=0.0335
	1/1	1/1	1/1	1

filters=(64, 64, 256, 256, 256, 256)
dropout=0.57 lr=0.0335
	1/1	1/1	1/1	1

filters=(64, 64, 256, 256, 256, 256)
dropout=0.8 lr=0.0885
	1/1	1/1	1/1	1

filters=(128, 256, 256, 256, 256, 256)
dropout=0.8 lr=0.6158
	1/1	1/1	1/1	1

filters=(64, 128, 128, 256, 256, 256)
dropout=0.57 lr=0.0206
	1/1	1/1	1/1	1

filters=(64, 64, 64, 64, 256, 256)
dropout=0.57 lr=1.0
	1/1	1/1	1/1	1

Now, the same hyperparameters, but with SGD optimizer and momentum in (0.1, 0.99):

Hyperparameters **Filters, dropout, lr**	**Maximum val_acc on 15 epochs**

filters=(128, 256, 256, 256, 256, 256) dropout=0.46
lr=0.0018 momentum=0.61
	60

filters=(64, 64, 128, 128, 128, 256) dropout=0.35
lr=0.0029 momentum=0.79
	59

filters=(64, 64, 128, 128, 128, 256) dropout=0.35
lr=0.0078 momentum=0.80
	57

filters=(64, 64, 64, 128, 128, 256) dropout=0.8 lr=0.0006
momentum=0.75
	56

filters=(64, 128, 128, 128, 128, 128) dropout=0.3 lr=0.0545 momentum=0.33	54
filters=(64, 64, 64, 64, 64, 256) dropout=0.3 lr=0.0014 momentum=0.6	54
filters=(128, 128, 128, 128, 256, 256) dropout=0.8 lr=0.0545 momentum=0.47	52
filters=(128, 128, 256, 256, 256, 256) dropout=0.52 lr=0.0006 momentum=0.38	52
filters=(64, 128, 128, 256, 256, 256) dropout=0.46 lr=0.0885 momentum=0.56	50
filters=(64, 64, 128, 128, 128, 128) dropout=0.57 lr=0.0545 momentum=0.24	48
filters=(64, 64, 64, 64, 64, 256) dropout=0.3 lr=0.0004 momentum=0.1	39
filters=(128, 128, 128, 256, 256, 256) dropout=0.63 lr=0.0001 momentum=0.1	29
filters=(64, 64, 128, 128, 256, 256) dropout=0.35 lr=0.0001 momentum=0.24	29
filters=(128, 128, 128, 128, 128, 256) dropout=0.57 lr=0.0001 momentum=0.1	27
filters=(64, 64, 64, 64, 64, 64) dropout=0.63 lr=0.0001 momentum=0.14	13
filters=(64, 64, 64, 256, 256, 256) dropout=0.68 lr=1.0 momentum=0.38	1
filters=(64, 64, 64, 128, 256, 256) dropout=0.46 lr=1.0 momentum=0.75	1
filters=(128, 128, 128, 128, 128, 256) dropout=0.68 lr=0.0885 momentum=0.94	1
filters=(64, 64, 64, 64, 64, 128) dropout=0.3 lr=0.6158 momentum=0.52	1

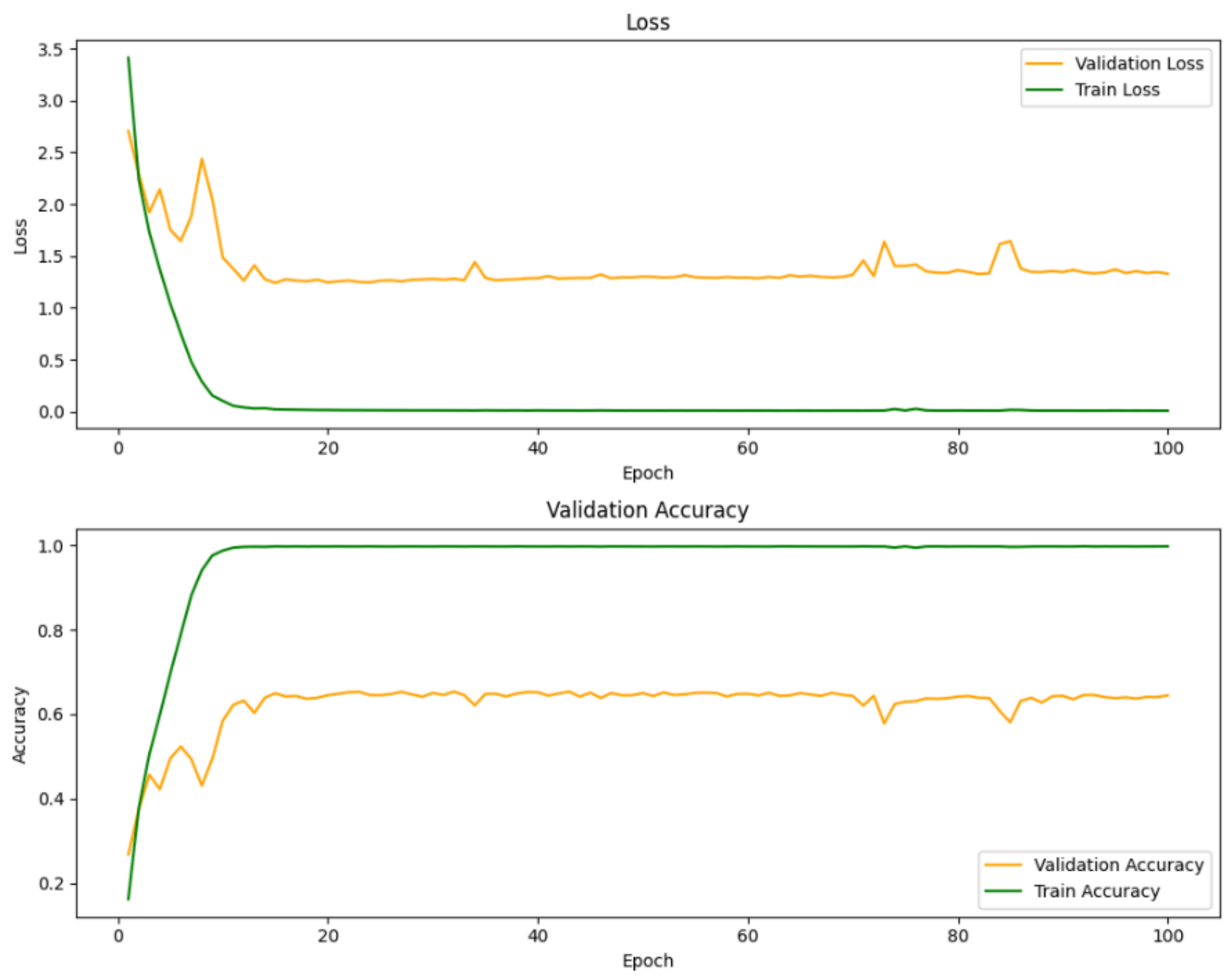
We can observe now that the first 3 combinations restrict the range for momentum and learning rate, therefore we could perform another random search, but with learning_rate in (0.01, 0.001) and momentum in (0.6, 0.85).

Hyperparameters **Filters, dropout, lr**	**Maximum val_acc on 15 epochs**
filters=(128, 256, 256, 256, 256, 256) dropout=0.46 lr=0.0020 momentum=0.74	63
filters=(128, 128, 256, 256, 256, 256) dropout=0.52 lr=0.0016 momentum=0.67	62
filters=(128, 128, 128, 256, 256, 256) dropout=0.63	61

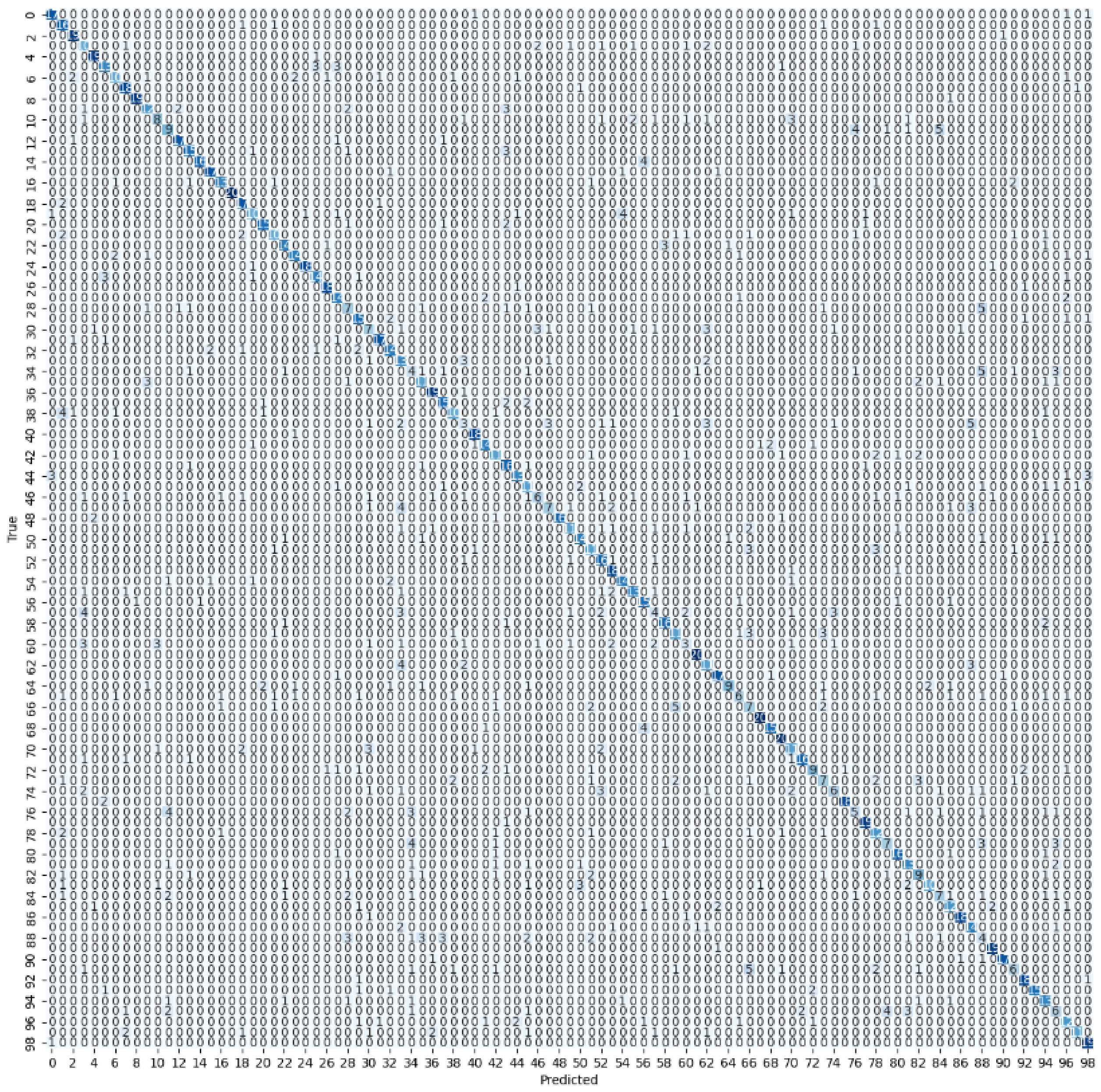
lr=0.0011 momentum=0.6	
filters=(64, 64, 128, 128, 128, 256) dropout=0.35 lr=0.0029 momentum=0.79	60
filters=(128, 128, 128, 128, 256, 256) dropout=0.8 lr=0.0048 momentum=0.70	60
filters=(64, 128, 128, 256, 256, 256) dropout=0.46 lr=0.0054 momentum=0.73	60
filters=(64, 64, 64, 128, 128, 256) dropout=0.8 lr=0.0016 momentum=0.78	59
filters=(64, 64, 64, 256, 256, 256) dropout=0.68 lr=0.01 momentum=0.67	59
filters=(64, 64, 128, 128, 256, 256) dropout=0.35 lr=0.0011 momentum=0.63	59
filters=(64, 64, 64, 256, 256, 256) dropout=0.57 lr=0.0020 momentum=0.79	59
filters=(64, 128, 128, 128, 256, 256) dropout=0.57 lr=0.0023 momentum=0.79	59
filters=(128, 128, 128, 128, 128, 256) dropout=0.68 lr=0.0054 momentum=0.83	58
filters=(128, 128, 128, 128, 128, 256) dropout=0.57 lr=0.0011 momentum=0.6	58
filters=(64, 64, 64, 64, 64, 256) dropout=0.3 lr=0.0014 momentum=0.6	57
filters=(64, 64, 128, 128, 128, 128) dropout=0.57 lr=0.0048 momentum=0.63	57
filters=(128, 128, 128, 128, 128, 256) dropout=0.57 lr=0.0020 momentum=0.66	57
filters=(64, 128, 128, 128, 128, 128) dropout=0.3 lr=0.0048 momentum=0.66	56
filters=(64, 64, 64, 128, 256, 256) dropout=0.46 lr=0.01 momentum=0.78	56
filters=(64, 64, 64, 64, 64, 64) dropout=0.63 lr=0.0011 momentum=0.61	56
filters=(64, 64, 64, 64, 64, 128) dropout=0.3 lr=0.0088 momentum=0.71	55

Now we will train the combination with the best results:

- 6 blocks, filters = [128, 256, 256, 256, 256, 256], kernels = [7, 5, 5, 5, 5, 5], strides = [2, 2, 1, 1, 1, 1], paddings = [3, 3, 1, 0, 1, 0], dropout = 0.46, SGD optimizer, lr=0.0020 momentum=0.74, 100 epochs. **Max accuracy: 65.35%** (epoch 43)



Below there is a plot with the heatmap obtained from the confusion matrix:



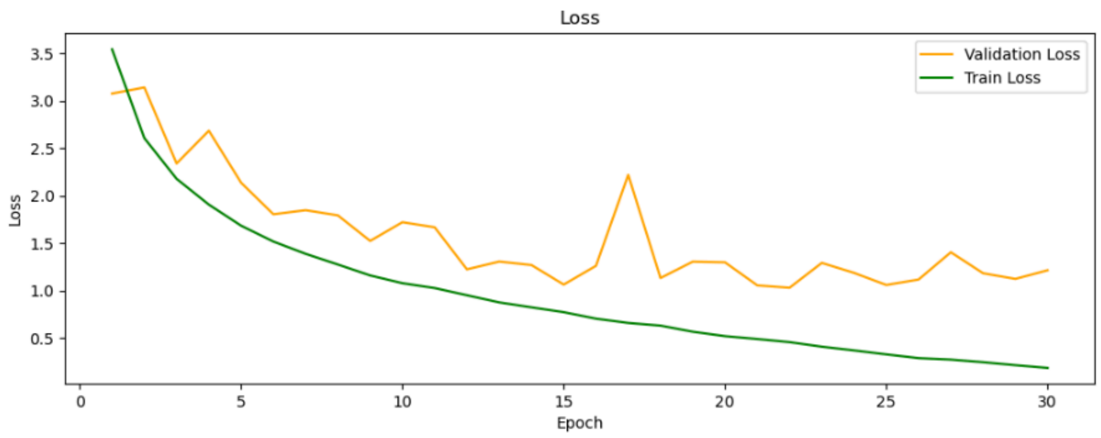
3.b. Resnet based models

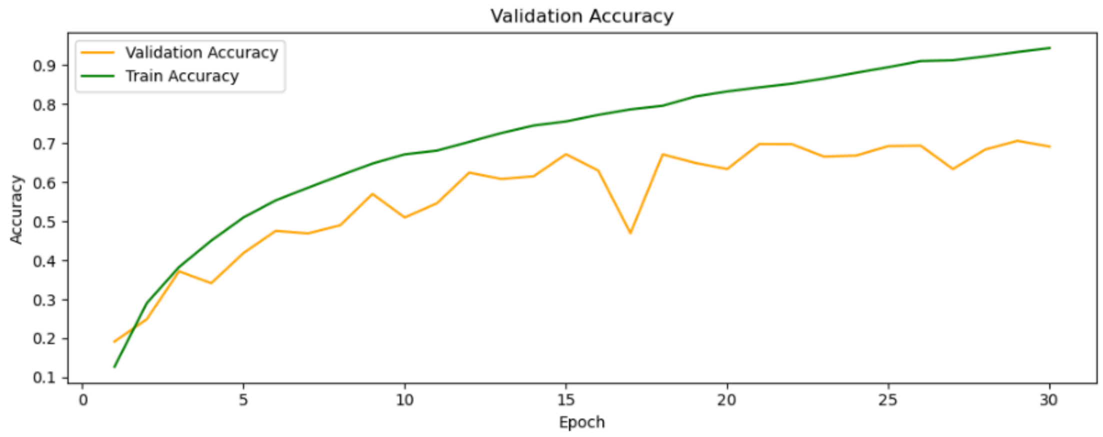
The architecture initiates with a convolutional layer featuring 64 filters, a kernel size of 3, a stride of 1, and padding of 1. This configuration ensures that the convolutional operation does not alter the dimensions of the input. Subsequently, multiple layers follow, each containing a variable number of ResNet blocks. These blocks adhere to the ResNet18 type and consist of two 3x3 convolutions with variable number of features, taking on values of 64, 128, 256, or 512. Additionally, each ResNet block incorporates batch normalization and the Rectified Linear Unit (ReLU) activation function. The network further incorporates a global average pooling layer, which computes the average value for each feature map across the spatial dimensions. Finally, a fully connected layer is employed for classification, predicting the output among the 100 classes.

All training sessions were with batch_size=32.

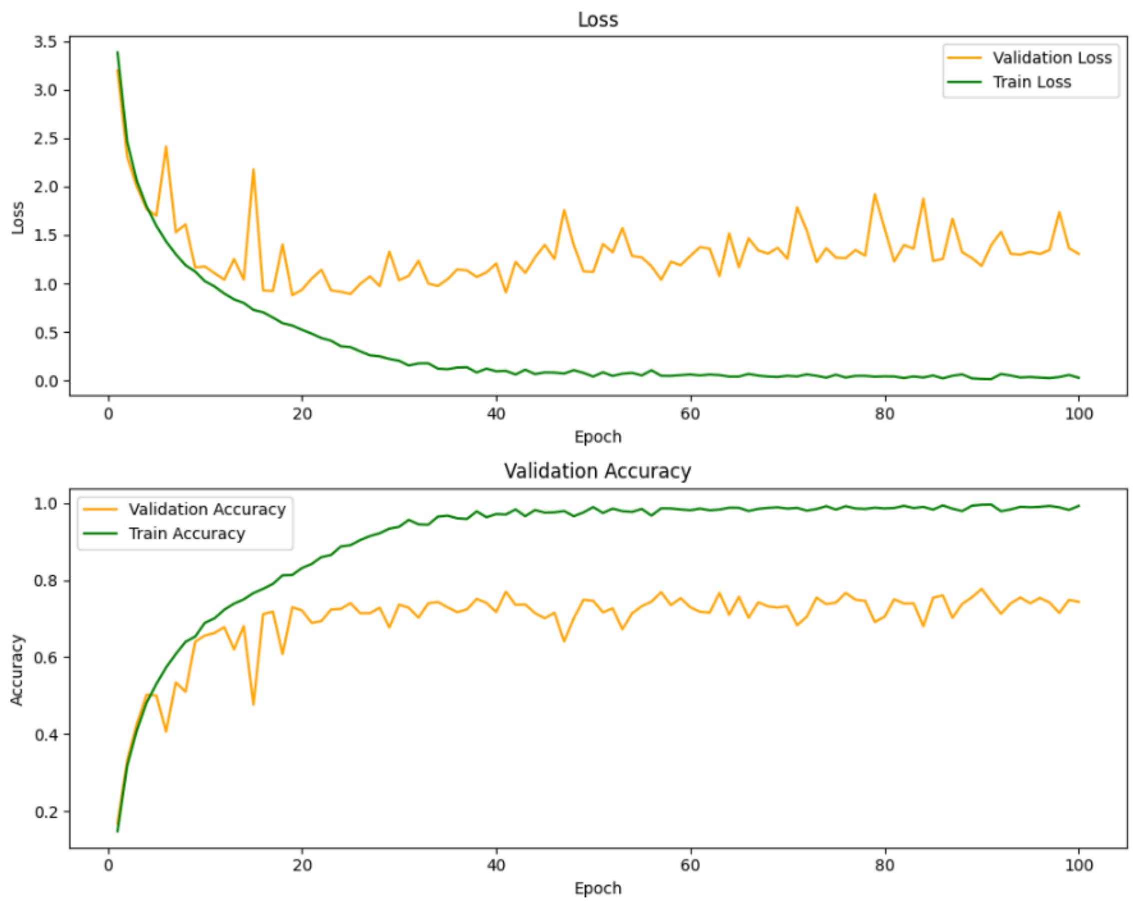
The first approach in training was to start with 2 blocks in each layer, filters of size 64, 128 and 256 and strides 1, 2 and 2 with both Adam and SGD optimizers.

- 3 layers, nr_blocks = [2,2,2], filters=[64,128,256], Adam, lr = 0.001, 30 epochs, Max val_acc = 70.62% (epoch 29, train_acc=93.39%) -> F1 score 69.22% on test data

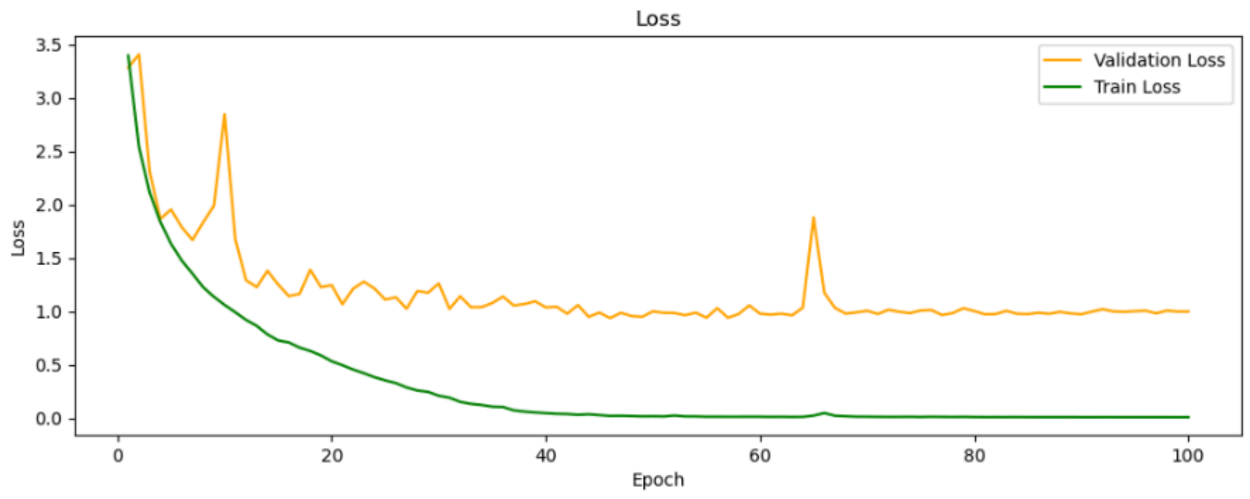


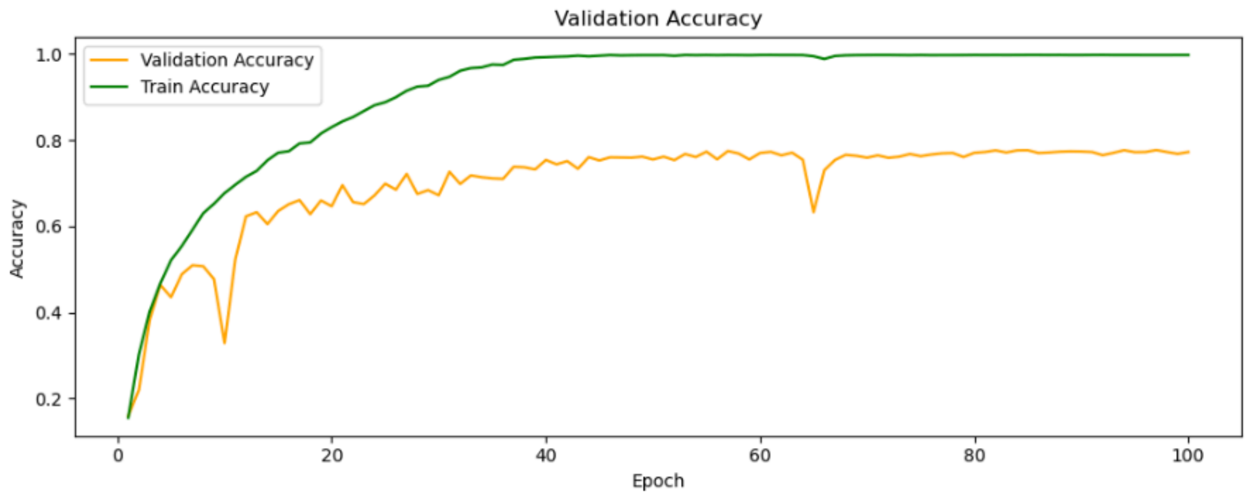


- 3 layers, nr_blocks = [2,2,2], filters=[64,128,256], Adam, lr = 0.001, 100 epochs, Max val_acc = 77% (epoch 90, train_acc=99%)



- 3 layers, nr_blocks = [2,2,2], filters=[64,128,256], SGD, lr = 0.001, momentum = 0, 300 epochs, Max val_acc = 68% (epoch 290, train_acc=99.7%)
- 3 layers, nr_blocks = [2,2,2], filters=[64,128,256], SGD, lr = 0.01, momentum = 0, 80 epochs, Max val_acc = 74.46% (epoch 80, train_acc=99.72%)
- 3 layers, nr_blocks = [2,2,2], filters=[64,128,256], SGD, lr = 0.01, momentum = 0.9, 100 epochs, Max val_acc = 77.69% (epoch 100, train_acc=99.73%)





On the test data, this model reached an f1 score of 77.6%.

Next I tried to slowly vary the number of blocks on 1 or more layers and/or momentum.

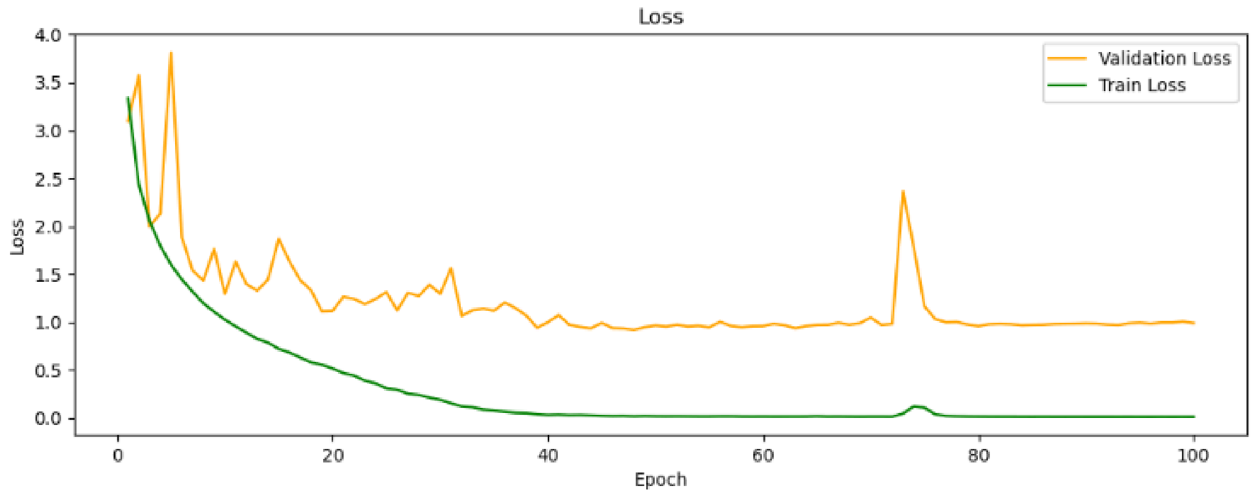
- 3 layers, nr_blocks = [3,2,2], filters=[64,128,256], SGD, lr = 0.01, momentum = 0.5, 80 epochs, Max val_acc = 76.12% (epoch 79, train_acc=99.76%)
- 3 layers, nr_blocks = [3,3,3], filters=[64,128,256], SGD, lr = 0.01, momentum = 0.95, 80 epochs, Max val_acc = 75.42% (epoch 80, train_acc=99.74%)

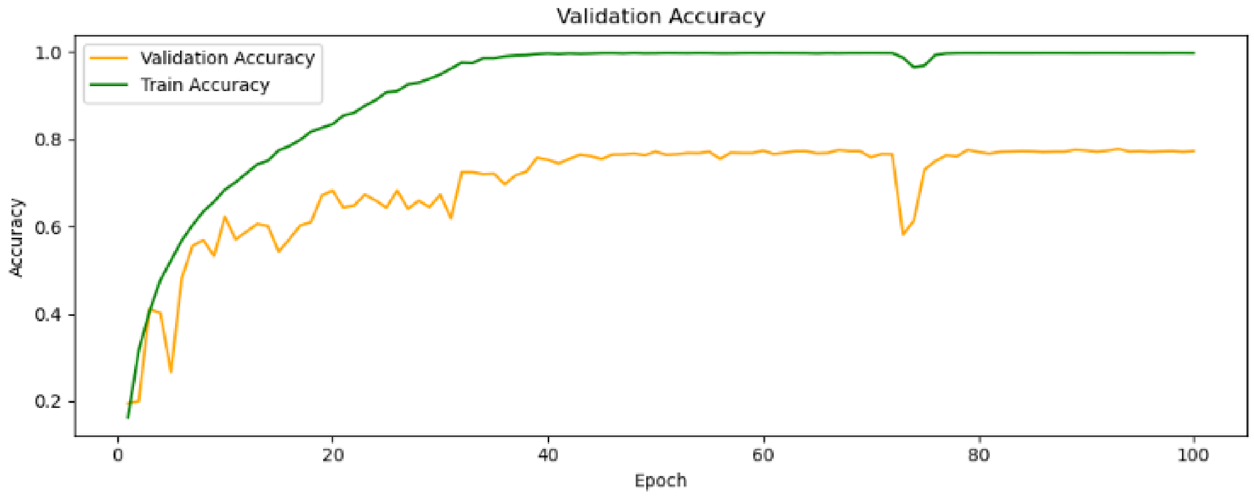
Next I performed a random search for the learning rate and momentum of the SGD optimizer when there are 3 layers, nr_blocks = [2,2,2], filters=[64,128,256]. We will have 10 experiments on 15 epochs each with random values for these 2 hyperparameters in the following ranges:

  - Momentum in (0.1, 0.99)
  - Learning_rate in (0.0001, 1)

Hyperparameters lr, momentum	**Maximum val_acc on 15 epochs**
lr = 0.0545, momentum = 0.38	69.46
lr = 0.0127, momentum = 0.52	68.65
lr = 0.0885, momentum = 0.1	67.62
lr = 0.2335, momentum = 0.52	67.23
lr = 0.0127, momentum = 0.56	65
lr = 0.0018, momentum = 0.38	54.62
lr = 0.006, momentum = 0.7	51.12
lr = 0.006, momentum = 0.75	50.62
lr = 0006, momentum = 0.28	37.35
lr = 0.001 , momentum = 0.7	25.96

- 3 layers, nr_blocks = [2,2,2], filters=[64,128,256], SGD, lr = 0.054, momentum = 0.38, 100 epochs, Max val_acc = 77.81% (epoch 93, train_acc=99.75%)



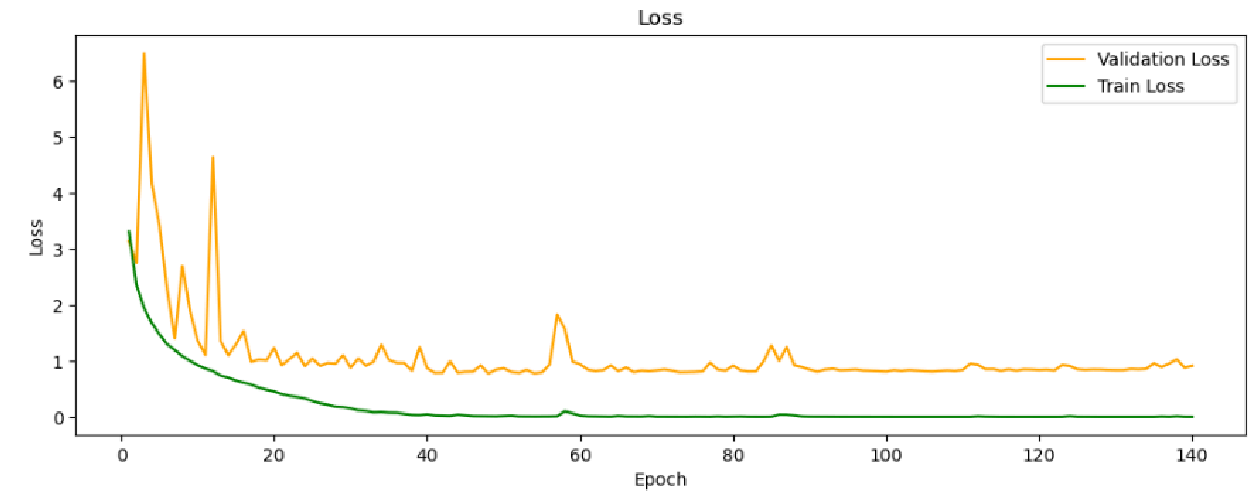


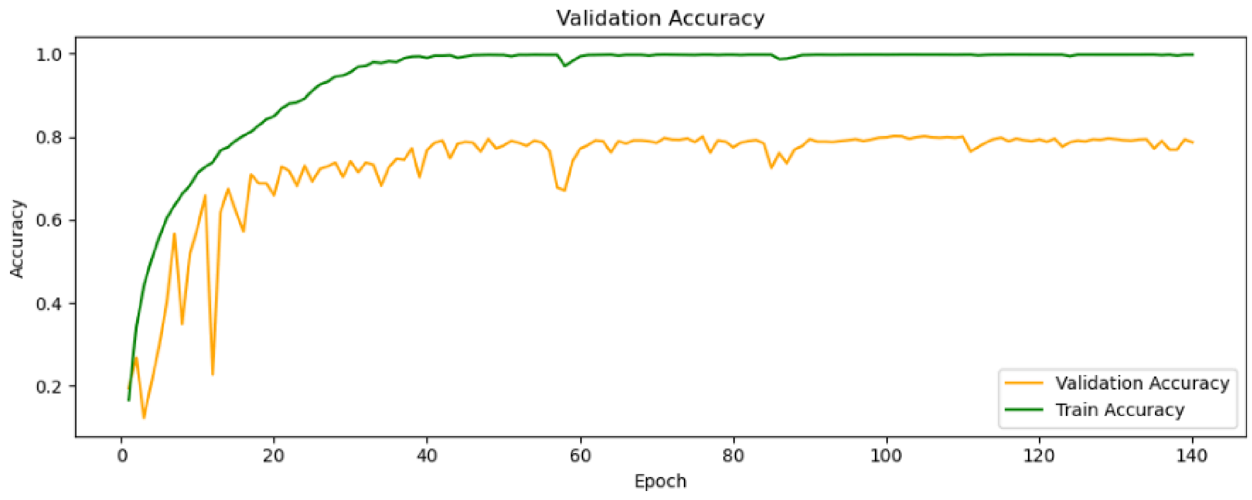
The first 5 results from the previous random search suggest us that we can restrict the range for learning rate between 0.01 and 1:

Hyperparameters lr, momentum	**Maximum val_acc on 15 epochs**
lr = 0.0263 , momentum = 0.75	73.2
lr = 0.1128, momentum = 0.56	69.95
lr = 0.0127, momentum = 0.7	69.7
lr = 0.1128, momentum = 0.52	68.6
lr = 0.026, momentum = 0.7	68.6
lr = 0.4832, momentum = 0.52	68
lr = 0.2976, momentum = 0.1	67.65
lr = 0.2335, momentum = 0.38	66.2
lr = 0.0263 , momentum = 0.28	65.95
lr = 0.0428, momentum = 0.38	65.75

Now we will train the model with the best hyperparameters:

- 3 layers, nr_blocks = [2,2,2], filters=[64,128,256], SGD, lr = 0.026, momentum = 0.755, 140 epochs, Max val_acc = 80.15% (epoch 101, train_acc=99.75%)





F1 score on test data: 79.015%
F1 score on validation data: 80.21%

In an attempt to optimize the model's performance, I undertook another iteration to fine-tune key parameters. This included adjusting the number of blocks for each layer, the number of filters, and experimenting with different strides. Despite these adjustments, the latest experiment did not outperform the previous iteration. Detailed results are outlined below:

Fixed hyperparameters: 3 blocks, lr=0.026, momentum=0.755

Hyperparameters to be tuned range:

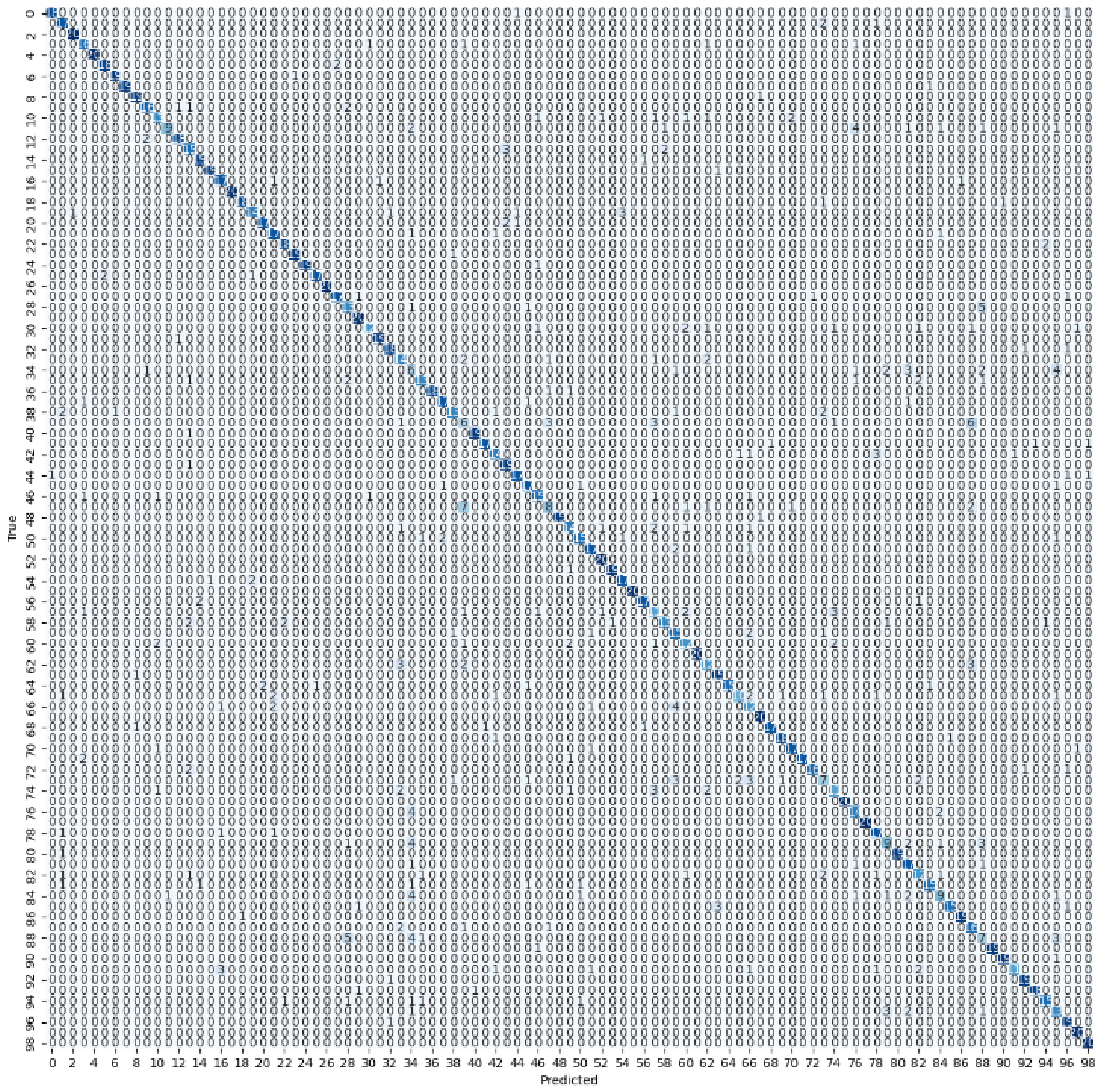
- nr_blocks_layer1: range(1, 6),
- nr_blocks_layer2: range(1, 6),
- nr_blocks_layer3: range(1, 6),
- strides: [1, 2, 3] - arrays of 3
- filters: [64, 128, 256, 512] - arrays of 3 in ascending order

Hyperparameters nr_blocks_layer1, nr_blocks_layer2, nr_blocks_layer3, strides, filters	**Maximum val_acc on 15 epochs**
blocks 4, 4, 4, strides=(2, 2, 1), filters=(64, 128, 128)	65.2
blocks 1, 5, 2, strides=(3, 2, 3), filters=(128, 128, 512)	65.15
blocks 5, 3, 5, strides=(3, 1, 1), filters=(512, 512, 512)	64.35
blocks 1, 1, 3, strides=(2, 3, 1), filters=(128, 128, 256)	63.95
blocks 1, 2, 5, strides=(2, 2, 2), filters=(128, 256, 256)	63.55
blocks 5, 2, 2, strides=(3, 3, 3), filters=(256, 256, 512)	63.35
blocks 2, 2, 3, strides=(3, 2, 2), filters=(256, 256, 256)	61.5
blocks 5, 2, 3, strides=(1, 1, 2), filters=(64, 64, 128)	60.25
blocks 2, 2, 2, strides=(3, 2, 3), filters=(64, 64, 64)	56.05
blocks 4, 3, 1, strides=(1, 1, 1), filters=(64, 128, 128)	55.45

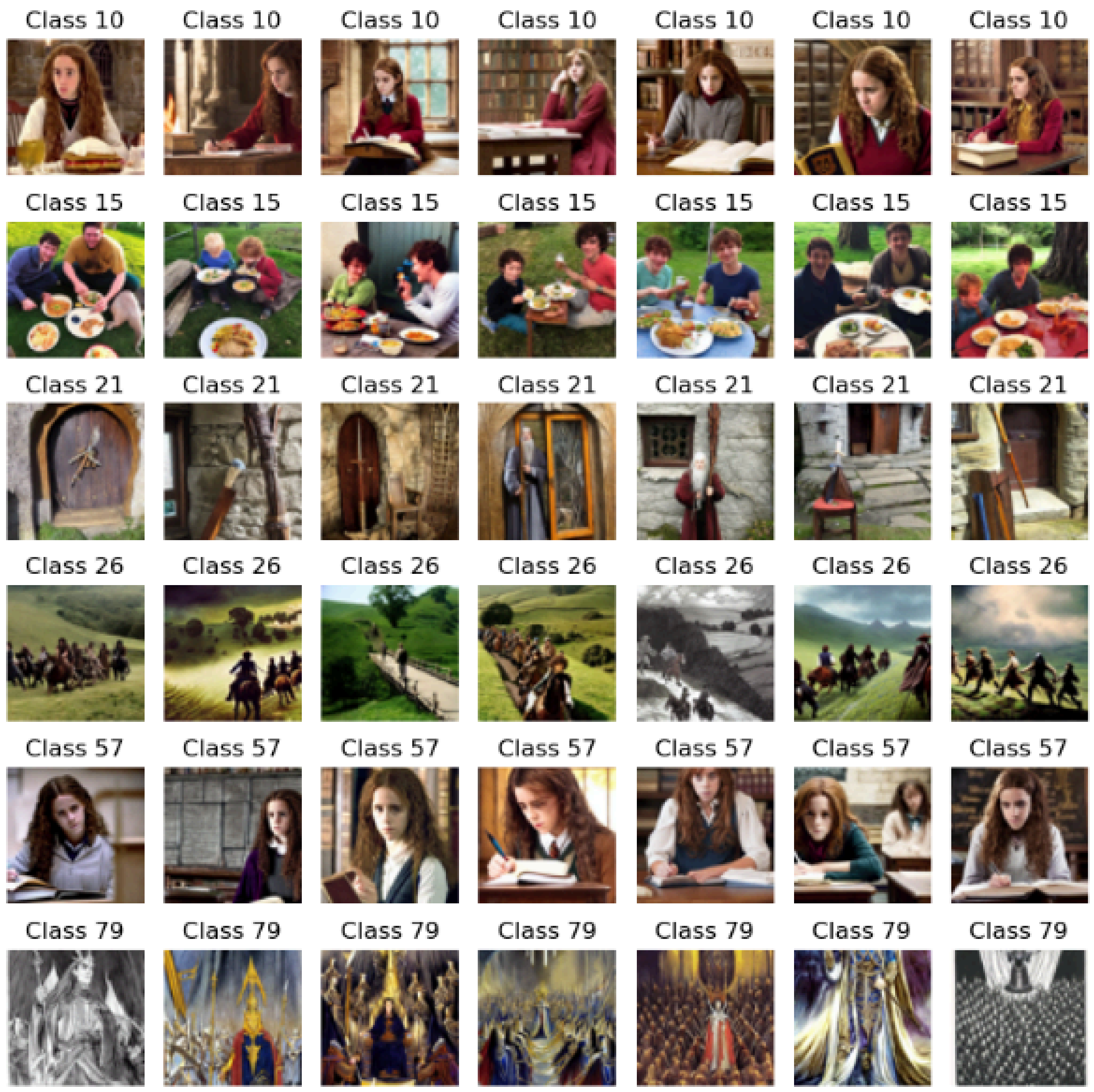
4. Final submission

As I specified earlier, the model with the best results was composed of 3 layers, nr_blocks = [2,2,2], filters=[64,128,256], SGD optimizer, lr = 0.026, momentum = 0.755, 140 epochs.

Below there is a plot with the heatmap obtained from the confusion matrix:



We can visualize the predictions on a few classes on the test data with the following plot. There were selected the first 7 images from a few classes that were labeled with the results from the final submission.


Class 10
Class 15
Class 21
Class 26
Class 57
Class 79