BBM418 Computer Vision Laboratory

Programming Assignment 3
Image Classification with Convolutional Neural Networks

Mehmet Giray Nacakcı

21989009

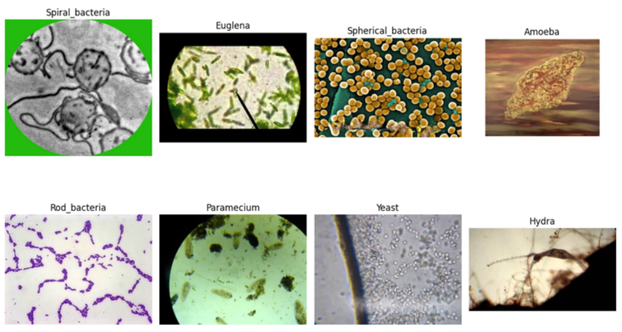
Department of Computer Engineering
Hacettepe University
Ankara, Turkey

b21989009@cs.hacettepe.edu.tr

Overview

In this assignment, the aim was to classify images of microorganisms using Convolutional Neural Networks. I only implemented the PART 1.

1 Dataset and Preprocessing



Images in the dataset are generally bigger, yet, for having a fixed size to feed to the network, and to have less computational time, images are resized to 3 x 224 x 224. Then, normalized.

I tried to use the highest quality resampling function (`PIL.Image.LANCZOS`) to preserve fine-grained details, without an antialiasing filter.

Since the categories in the dataset has 72, 75, 75 , 76, 85, 86, 152, 168 images,
for balance and homogeneity (for less bias towards any class);
each class is partitioned as 50 train, 10 validation, 10 test images.

Partitions are shuffled both before and after this partition, such that mini-batches will have homogenous many images of each class.

2 Custom CNN Architecture

My custom CNN Architecture was formed based on two aims:

1. the requirement of using 6 convolutional layers.
2. training process taking as little time as possible
   currently: takes around 700 seconds for 80 epochs.

Architecture is as follows:

```
# image size: 3 x 224 x 224
Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=1, padding=1)
ReLU()
# image size: 16 x 224 x 224

MaxPool2d(kernel_size=2, stride=2)
# image size: 16 x 112 x 112

Conv2d(in_channels=16, out_channels=16, kernel_size=3, stride=2, padding=1)
ReLU()
# image size: 16 x 56 x 56

Conv2d(in_channels=16, out_channels=16, kernel_size=3, stride=2, padding=1)
ReLU()
# image size: 16 x 28 x 28

#

Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1)
ReLU()
# image size: 32 x 28 x 28
Conv2d(in_channels=32, out_channels=16, kernel_size=3, stride=1, padding=1)
ReLU()
# image size: 16 x 28 x 28

# Residual Connection

Conv2d(in_channels=16, out_channels=16, kernel_size=3, stride=1, padding=1)
ReLU()

OPTIONAL:  Dropout (dropout_probability)  # drops out previous layer

# image size: 16 x 28 x 28
MaxPool2d(kernel_size=2, stride=2)
# output size: 16 x 14 x 14 = 3136
Flatten()

Fully_Connected_Layer (inputs = 3136 , outputs = 8)
```

Residual Connection

When residual connection starts, it can be considered as the network branches into two, then merges. Residual connection's values are directly (identity) (element-wise) added to the parallel layers' output, since same size.

Dropout

Dropout layer randomly (different for each forward pass) sets some of its inputs to zero during training. This helps regularization and thus reduces the risk of Overfitting, increases generalization ability. It is generally recommended to dropout towards the end, closer to the fully connected layers, so that, network will be forced to be more adaptive and rely less heavily on specific features or neurons.

3 Choice of Hyper-Parameters

Loss function of choice is CrossEntropyLoss, since it is popular in CNN networks. It aims to maximize the probability of the correct class among the output layer prediction outputs.

Activation functions add non-linearity to otherwise linear network layers, thus add complexity so that models can get complex enough to learn complex patterns.
There are many activation functions, yet, ReLU is the most popular and robust for Computer Vision applications, since it preserves gradients without shrinking them.

```
ReLU(x) = maximum(0, x)
```

`Torch.optim.Adam(learning_rate)` is a popular optimization algorithm, just like gradient descent. Adaptive Movement Estimation means it automatically adapts learning rate for each weight.

Using Mini-Batch optimization, in each epoch, the training set is not used as a whole at once. An optimization (weight update) run is done for each batch (equal partition) of the set separately, one after another.
This approach reduces computation time and memory usage, and also allows the model to converge in less many epochs.
With batch_size = 40, our 50 x 8 = 400 image Training set is trained as 10 batches, and with batch_size = 20, as 20 batches.

With Learning Rates higher than 0.002, even though validation loss and test loss decreased, validation accuracy and test accuracy almost always stayed around at 12.5% throughout 80 epochs, which is the random probability for 8 classes, and did not exceed 15%. Thus, I experimented with smaller learning rates.

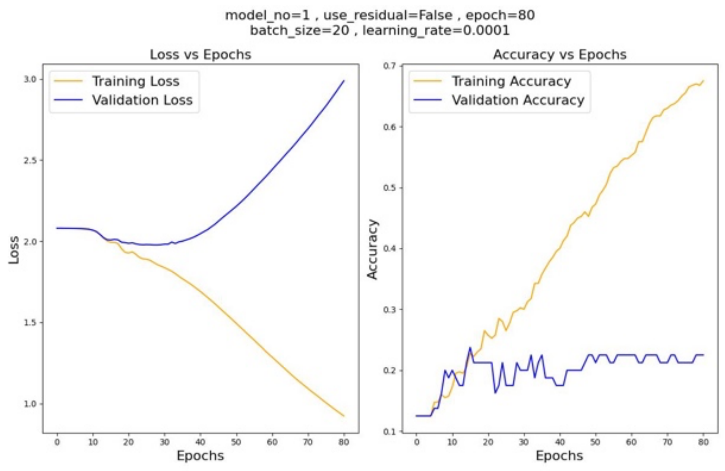
4 Results

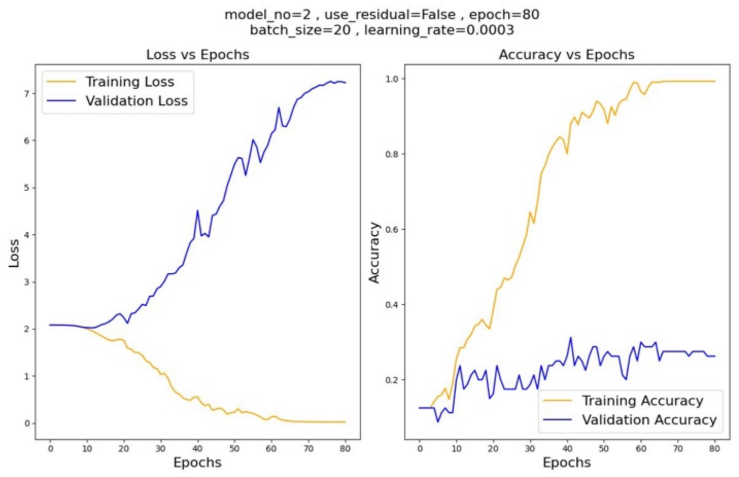
Loss is calculated as the CrossEntropyLoss of the dataset partition after training epochs.

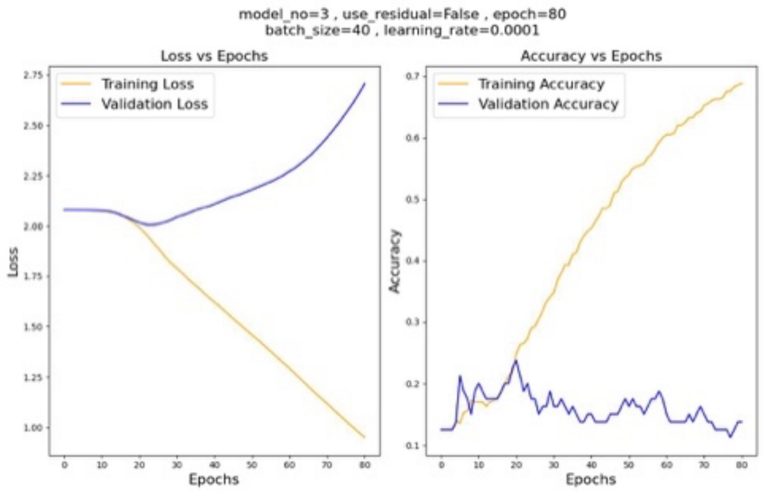
Accuracy is calculated as the ratio of how much of the actual classes of input images and the classes predicted by the network match. After each epoch, for Training and Validation partitions.

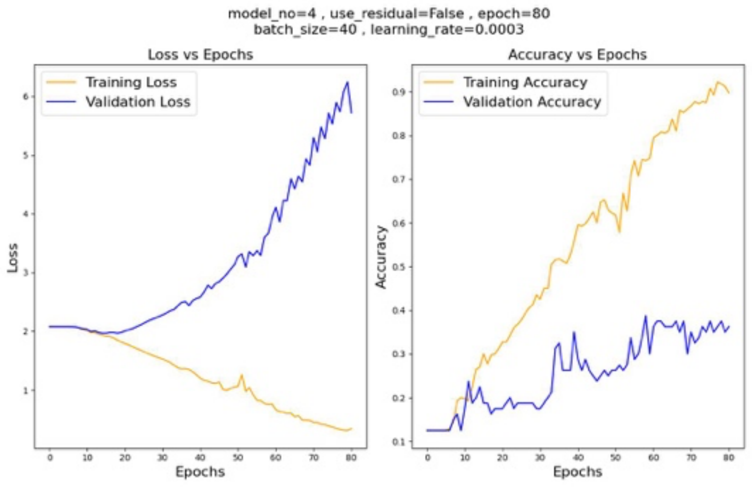
Accuracy = Count of matching correct predictions / Count of all Predictions

Models of 12 different settings of Hyper-Parameters and architectural connection preferences are trained, and evaluated below.

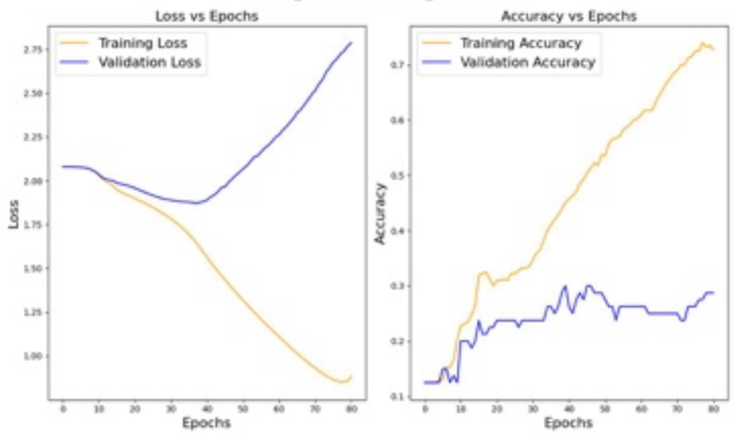

model_no=1 , use_residual=False , epoch=80
batch_size=20 , learning_rate=0.0001


model_no=2 , use_residual=False , epoch=80
batch_size=20 , learning_rate=0.0003

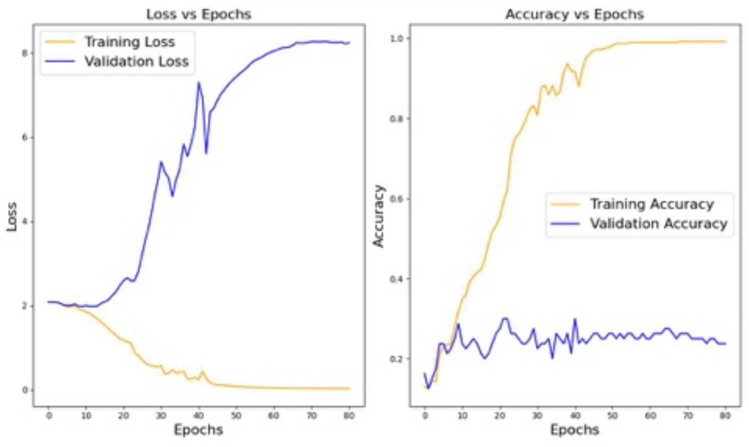

model_no=3 , use_residual=False , epoch=80
batch_size=40 , learning_rate=0.0001


model_no=4 , use_residual=False , epoch=80
batch_size=40 , learning_rate=0.0003

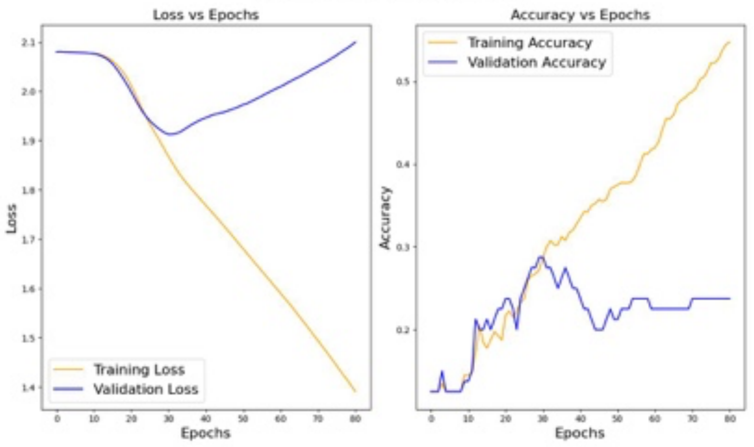
model_no=5 , use_residual=True , epoch=80
batch_size=20 , learning_rate=0.0001



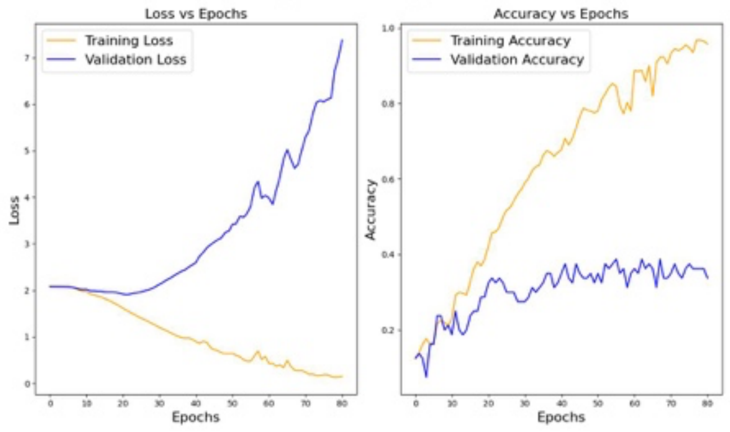
model_no=6 , use_residual=True , epoch=80
batch_size=20 , learning_rate=0.0003



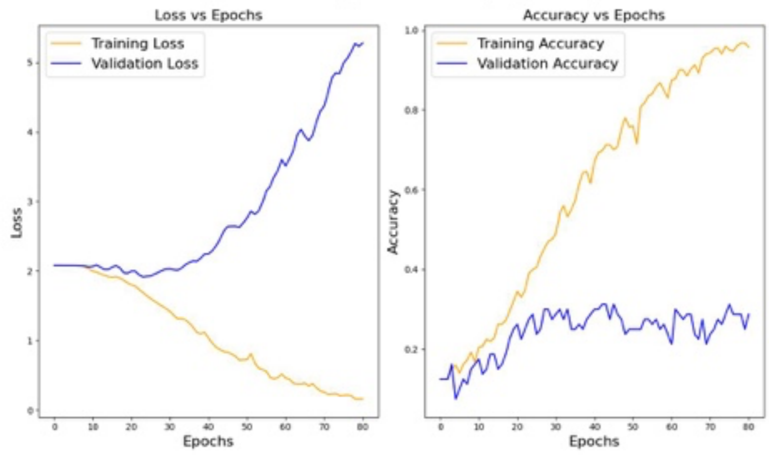
model_no=7 , use_residual=True , epoch=80
batch_size=40 , learning_rate=0.0001



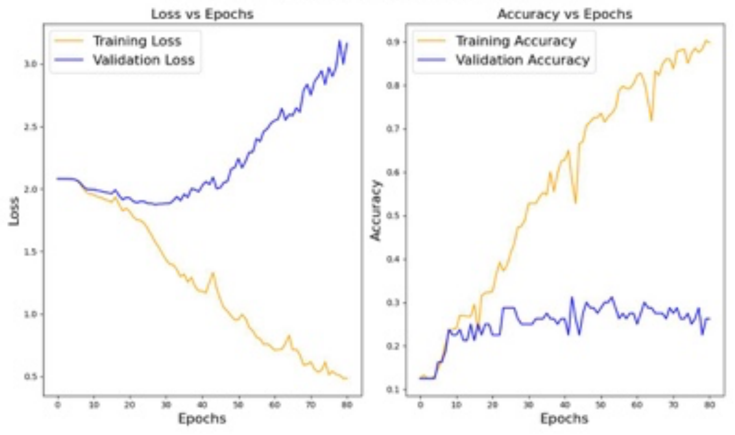
model_no=8 , use_residual=True , epoch=80
batch_size=40 , learning_rate=0.0003



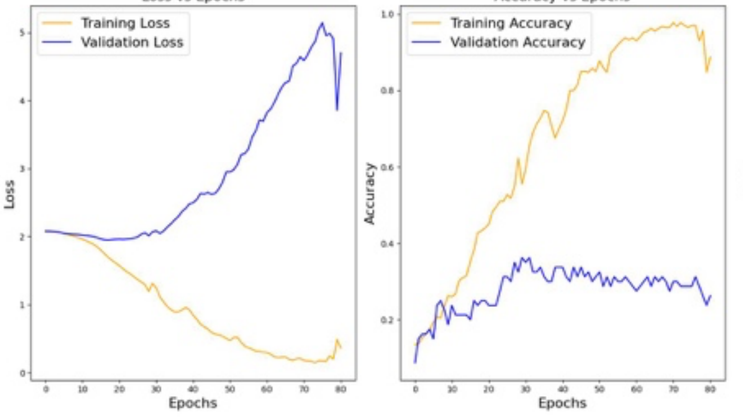
model_no=9 , use_residual=False , epoch=80
batch_size=40 , learning_rate=0.0003 , dropout_probability=0.2



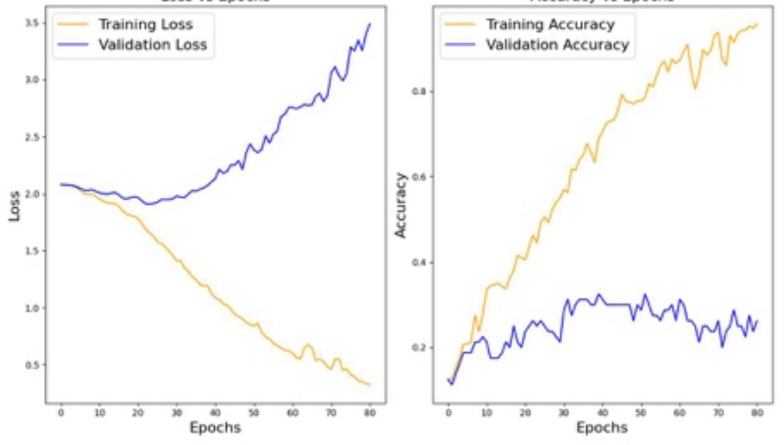
model_no=10 , use_residual=False , epoch=80
batch_size=40 , learning_rate=0.0003 , dropout_probability=0.5

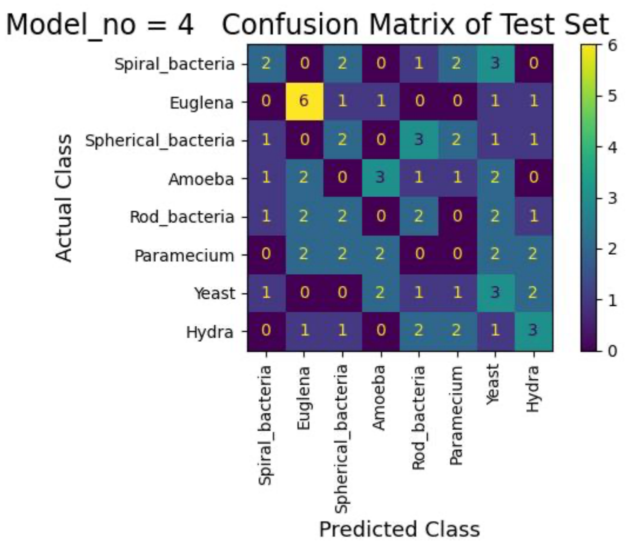


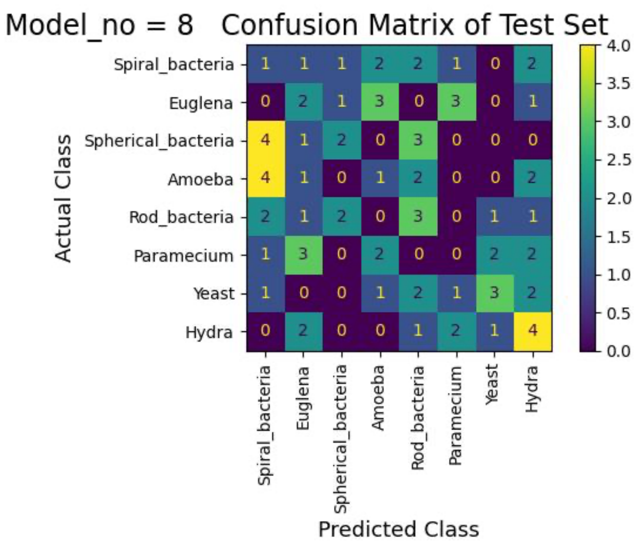
model_no=11 , use_residual=True , epoch=80
batch_size=40 , learning_rate=0.0003 , dropout_probability=0.2



model_no=12 , use_residual=True , epoch=80
batch_size=40 , learning_rate=0.0003 , dropout_probability=0.5







	After 80 Epochs					Earliest Best Epoch (Max Validation Accuracy)					Model Parameters			
	train loss	validation loss	train accuracy	validation accuracy	test accuracy	best epoch number	train loss	validation loss	train accuracy	validation accuracy	Residual connections?	Batch size	Learning Rate	Dropout probability
model 1	0.92	2.99	0.68	0.26	0.20	15	1.99	2.01	0.23	0.24	NO	20	0.0001	0
model 2	0.02	7.23	0.99	0.26	0.21	41	0.43	3.97	0.88	0.31			0.0003	
model 3	0.95	2.71	0.69	0.14	0.28	20	1.99	2.02	0.25	0.24		40	0.0001	
model 4	0.34	5.71	0.90	**0.36**	0.26	58	0.75	3.67	0.74	**0.39**			0.0003	
model 5	0.88	2.79	0.73	0.29	0.30	39	1.59	1.88	0.45	0.30	YES	20	0.0001	
model 6	0.02	8.24	0.99	0.24	0.23	21	1.14	2.65	0.59	0.30			0.0003	
model 7	1.39	2.10	0.55	0.24	0.26	29	1.88	1.92	0.27	0.29		40	0.0001	
model 8	0.15	7.37	0.96	**0.34**	0.20	72	0.37	4.14	0.89	**0.39**			0.0003	
model 9	0.16	5.28	0.96	0.29	0.29	42	0.90	2.36	0.70	0.31	NO	40	0.0003	0.2
model 10	0.48	3.16	0.90	0.26	0.20	42	1.25	2.03	0.58	0.31				0.5
model 11	0.36	4.70	0.89	0.26	0.24	29	1.31	2.07	0.56	0.36	YES	40	0.0003	0.2
model 12	0.33	3.50	0.96	0.26	0.24	39	1.13	2.10	0.70	0.33				0.5

Results Discussion

Early stopping is not applied, I wanted to see where the models can reach in 80 epochs, thus epoch_80 is not the best number for most models. If training process could be faster, I would have liked to explore higher epoch numbers to experiment with convergence.

First 8 models are trained, then, based on the hyper-parameters of the best two models of the first 8, which are model_4 and model_8, models with Dropout (model_9 to 12) are created and trained.

The best model among those Without Residual Connection is model_4,
The best model among those With Residual Connection is model_8.

Because both have the highest Validation Accuracy at their best epoch, and also their epoch_80 . Having high validation value in epoch_80 is important since it means there is a chance that it can go higher if we continue training for more epochs.

Models without Dropout perform better. This is probably because both Network and Dataset is small, thus Network already hardly learns any generalization about the classes, dropout becomes an obstacle to learning.

Thus, overall best models are model_4 and model_8 , and their confusion matrices are given above.

Dataset is very small, and no augmentation was applied to increase data and model robustness. Thus, training loss decreases nicely, yet, model memorizes the Training_set (reaching 0.99 training accuracy), it indicates overfitting. Validation loss increases since model is not generalizing.

It should be noted that, the resulting Loss and Accuracy values are not a great indication of the model, and it can be coincidence to have these values in a run, since validation and test sets are only 80 and 80 images. Average results of multiple trainings could be more meaningful.

While random accuracy of 8 classes is 12.5 %, reaching 40 % with model_4 and model_8 is a success.

For more Accuracy:

- Leaky ReLU could be tried to decrease vanishing gradients problem.
- Convolutional filter channels could be increased, to have more representation power.
- Input images could be resized to a larger size.
- Training could be continued for more epochs.
- …