Deep Learning

Assignment 2

Felix Boelter

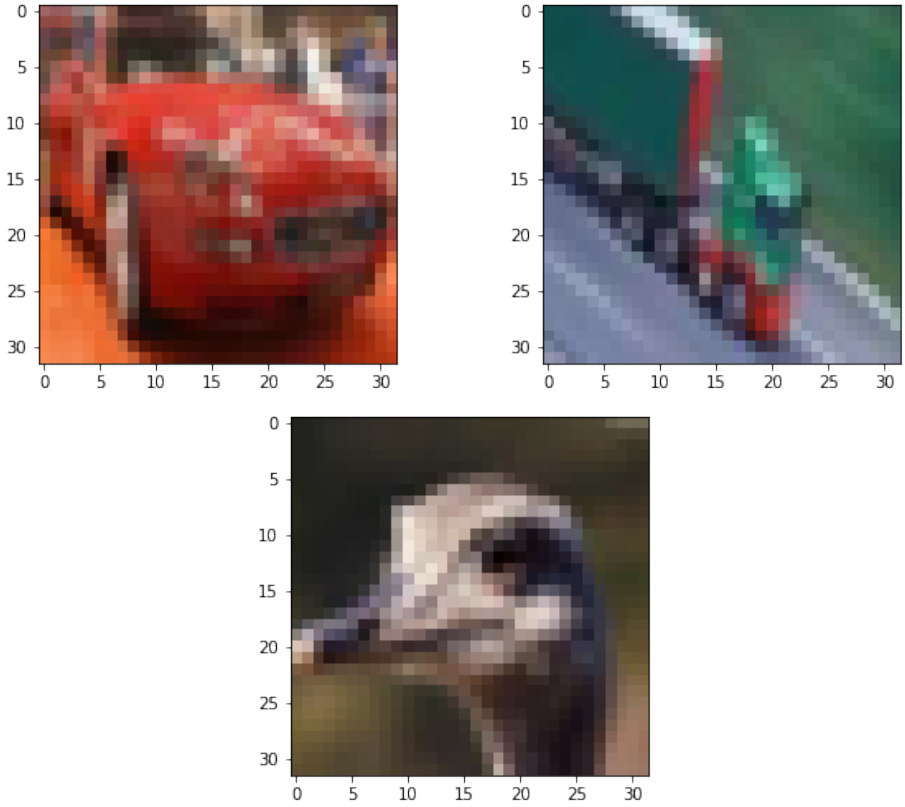
November 19, 2020

1 Image Classification Using ConvNets

1.1 Dataset

Visualize a few Images

Figure 1: Selection of Images from the Training Set



Normalize the values of the Images

To normalize the values of the images the mean and standard deviation were taken from the unnormalized images by concatenating the values, this gave me a mean of 0 and standard deviation of 1 for my data.

Mean Values			Standard Deviation Values		
0.4913159	0.48209456	0.4464672	0.2470339	0.24351034	0.26159197

Table 1: Normalization values for the transform argument

Split dataset into training and validation

To split the dataset into training and validation I sliced the dataset into the first 49000 images for the training set and sliced the last 1000 for the validation set.

Use of a sampler for the dataloaders

Using the SubsetRandomSampler, I used the indices creating for splitting the dataset into training and validation to get my random samples from the dataset, I then fed those elements into the both my dataloaders using the sampler parameter argument.

1.2 Model

Implementing the convolutional neural network

To create the first Convolutional Layer with 32 filters and a 3 x 3 window size, I used a 2D Convolution with in channels 3, output channels 32 and the kernel size of 3.

For the second Convolutional Layer with 32 filters and a 3 × 3 window size, I used the same 2D Convolution with in channels of 32, output channels of 32 and the kernel size of 3.

For the first Maxpooling layer with a 2 × 2 window size, I used a 2D Maxpooling with a kernel size of 2 and a stride of 2 which is set by default.

For the third Convolutional Layer with 64 filters and a 3 × 3 window size, I used the 2D Convolution with in channels of 32, output channels of 64 and a kernel size of 3.

For the fourth Convolutional Layer with 64 filters and a 3 × 3 window size, I used the 2D Convolution with in channels of 64, output channels of 64 and a kernel size of 3.

For the second Maxpooling Layer with a 2 × 2 window size, I used a 2D Maxpooling with a kernel size of 2 and a stride of 2 which was set by default.

For the first Fully Connected Layer with 512 units, I used a Linear transformation with 1600 (64 × 5 × 5) in features and 512 out features.

I then had to use a second Fully Connected Layer and applied a Linear transformation with 512 in features and 10 out features, as the model needs to classify images using 10 labels.

The Softmax was applied to the tensor by using the CrossEntropyLoss which uses a LogSoftmax.

Use of ReLUs

The ReLUs were applied after each convolutional layer and the first fully connected layer. To get the data into the fully connected layer, it was reshaped into a tensor of shape (-1, 64 × 5 × 5) which was ran through the first fully connected layer, where the output of this layer was ran through the second fully connected layer, which gave us 10 out features which can be used in the Softmax function.

1.3 Training

Implementing the Training Pipeline

When creating the training pipeline, I monitored the training loss and accuracy every 100 steps and the validation accuracy every epoch. Furthermore, to get reproducible results, I set both the PyTorch seed and the NumPy seed to 0.

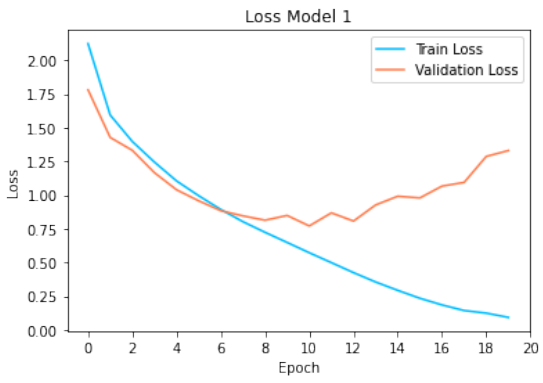
Training the model

	Epochs	Batch Size	Momentum	Learning Rate	Optimizer
Model 1	20	32	0.9	10^{-3}	SGD

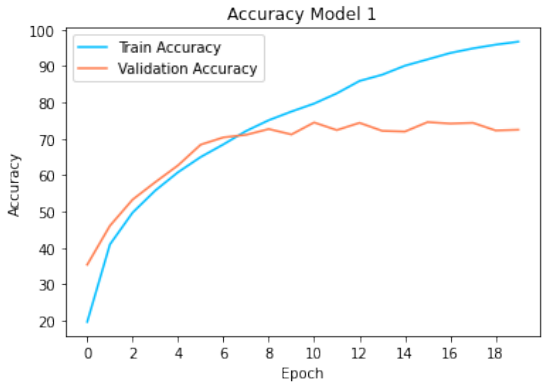
Table 2: Model hyperparameters

	Accuracy	Loss	Epoch
	Validation	Validation	Validation
Model 1	74.6%	0.9791406	16

Table 3: Model performance



(2.1)



(2.2)

Figure 2: Evolution of Training and Validation Loss/Accuracy for Model 1

In Figure (2.2) the training accuracy goes up to 100% while the validation accuracy stabilizes at around 70-75%. In Figure (2.1), it can clearly be seen that the model is overfitting the data, as the training loss is decreasing to zero while the validation loss is increasing.

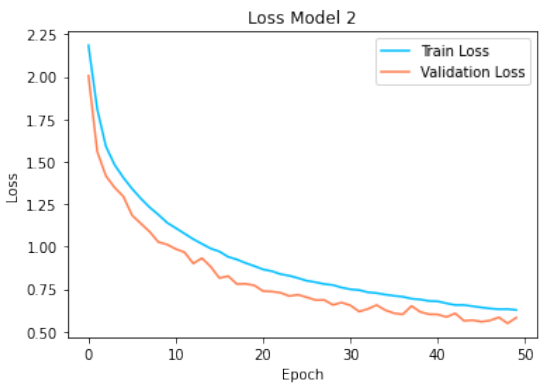
Using Dropout for model regularization

	Epochs	Batch Size	Momentum	Learning Rate	Optimizer	Dropout
Model 2	50	32	0.9	10^{-3}	SGD	0.5

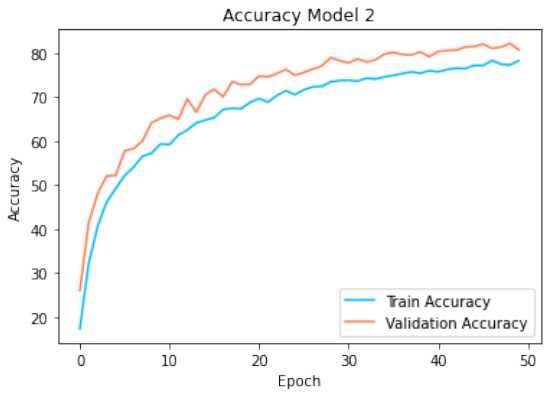
Table 4: Model hyperparameters

	Accuracy	**Loss**	**Epoch**
	Validation	Validation	Validation
Model 2	82.2%	0.5478038	49

Table 5: Model performance



(3.1)



(3.2)

Figure 3: Evolution of Training and Validation Loss/Accuracy for Model 2

The number of epochs were increased as dropout slows down convergence. A dropout layer was added after each max-pooling and the first fully-connected layer.
In Figure 3.2 we can see that the accuracy has improved a lot by adding dropout. There has been an increase in accuracy of 7.6%. In Figure 3.1 we can see that it is still slightly overfitting, but raising the number of epochs could be beneficial as the loss is still decreasing.

Hyperparameter Settings

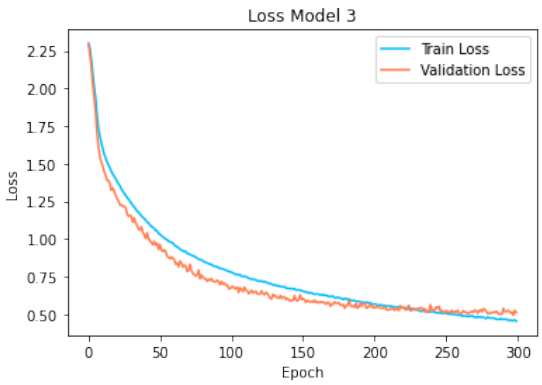
For the hyperparameter settings, the models were all trained using the SGD optimizer.

	Epochs	Batch Size	Momentum	Learning Rate	Dropout
Model 3	300	32	0.5	10^{-3}	0.5
Model 4	300	64	0.9	10^{-3}	0.5
Model 5	50	32	0.9	10^{-3}	0.2
Model 6	300	32	0.9	10^{-3}	0.8
Model 7	50	128	0.5	10^{-2}	0.5

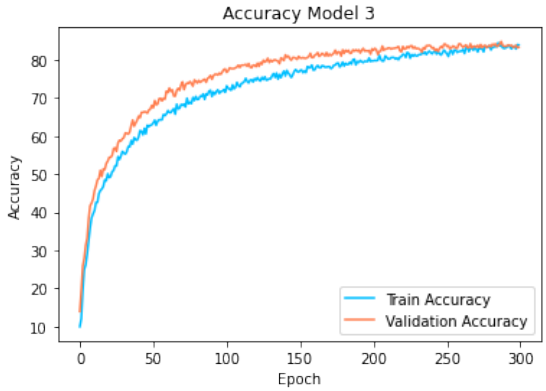
Table 6: Hyperparameter settings for the Models

	Accuracy	**Loss**	**Epoch**
	Validation	Validation	Validation
Model 3	84.8%	0.49940664	288

Table 7: Model performance



(4.1)



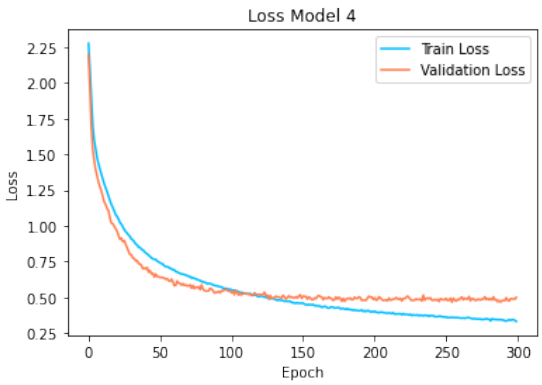
(4.2)

Figure 4: Evolution of Training and Validation Loss/Accuracy for Model 3

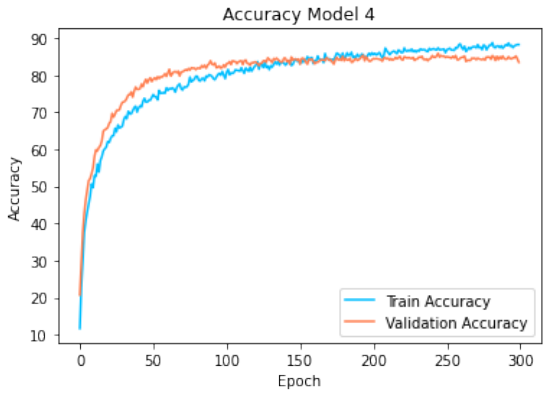
For model 3, when lowering the momentum to 0.5, the convergence took longer so the number of epochs had to be increased. The best accuracy was found at epoch 288 with an accuracy of 84.8% which is an increase of 2.6% in model 2 (Table 5).
In Figure 4.1, the model still shows slight signs of overfitting.

	Accuracy	**Loss**	**Epoch**
	Validation	Validation	Validation
Model 4	85.9%	0.48535138	245

Table 8: Model performance



(5.1)



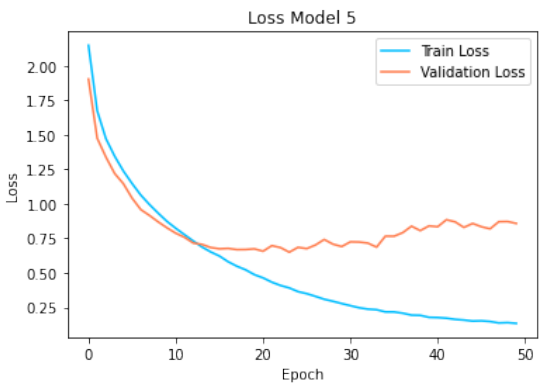
(5.2)

Figure 5: Evolution of Training and Validation Loss/Accuracy for Model 4

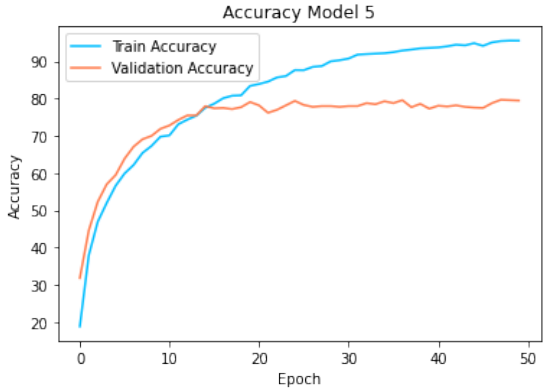
The validation accuracy has improved by 1.1% from model 3 (Table 7). The model stabilized and didn't get a higher validation accuracy for 55 further epochs and the training accuracy went up to 90%.

	Accuracy	**Loss**	**Epoch**
	Validation	Validation	Validation
Model 5	79.7%	0.8707671	48

Table 9: Model performance



(6.1)



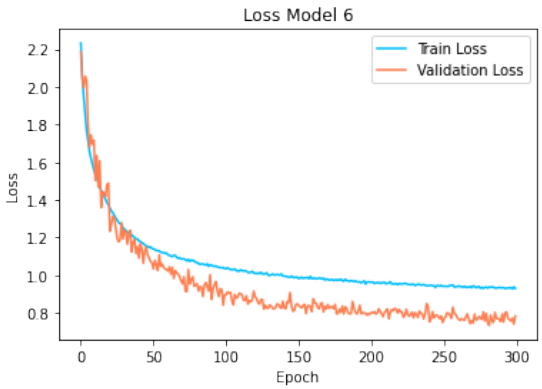
(6.2)

Figure 6: Evolution of Training and Validation Loss/Accuracy for Model 5

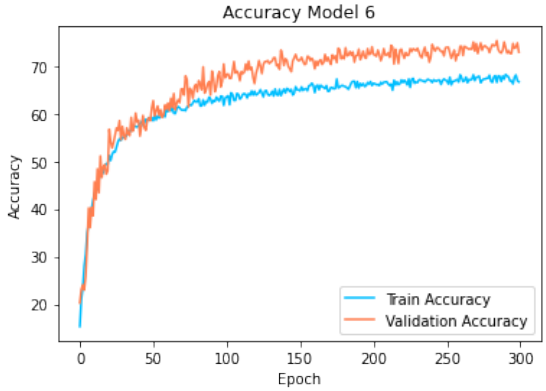
In this model when setting the dropout to 0.2, the model is overfitting, this can be seen in 6.1. the data and we get a drop in validation accuracy of 2.5% from model 2 (Table 5). The dropout probability is set too low and the model isn't dropping enough values from the neural network.

	Accuracy	**Loss**	**Epoch**
	Validation	Validation	Validation
Model 6	75.3%	0.7490999	285

Table 10: Model performance



(7.1)



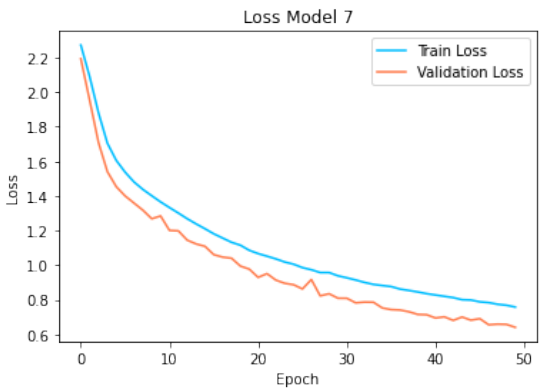
(7.2)

Figure 7: Evolution of Training and Validation Loss/Accuracy for Model 6

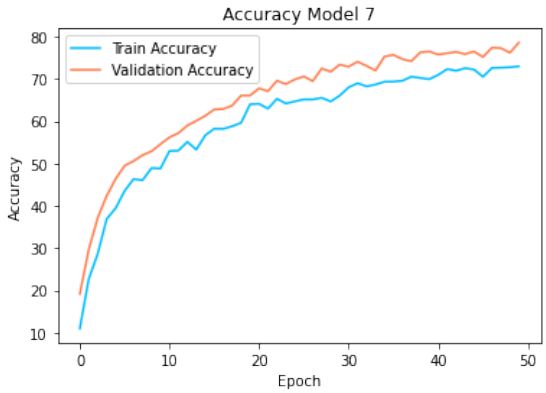
Model 6 is clearly underfitting, as the training and validation loss in Figure 7.1 aren't converging to zero. If the dropout is set too high the model can't learn enough and thus is going to underfit.

	Accuracy	**Loss**	**Epoch**
	Validation	Validation	Validation
Model 7	78.6%	0.63963145	50

Table 11: Model performance



(7.1)



(7.2)

Figure 8: Evolution of Training and Validation Loss/Accuracy for Model 7

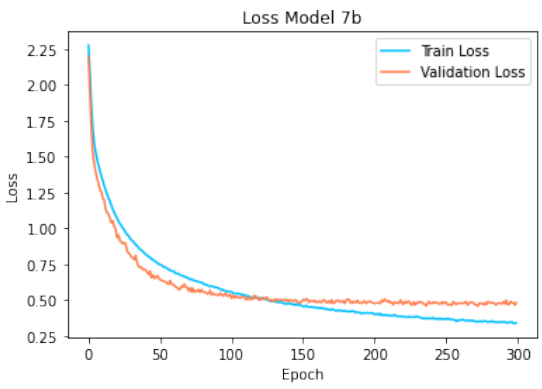
Model 7 looks promising as the highest validation accuracy was set on the last epoch of the experiment. The loss in Figure 7.1 is still decreasing and the accuracy in Figure 7.2 is still increasing. The experiment will be ran a second time with using a higher number of epochs.

	Epochs	Batch Size	Momentum	Learning Rate	Dropout
Model 7b	300	128	0.5	10^{-2}	0.5

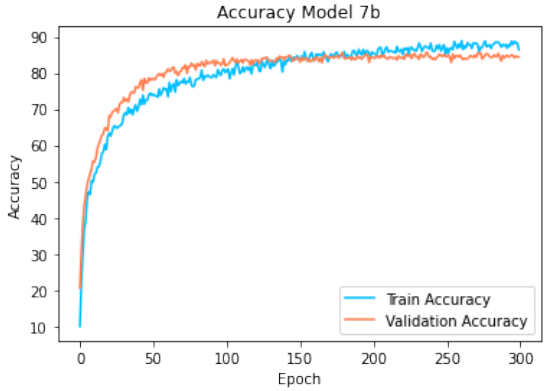
Table 12: Hyperparameter settings for the Models

	Accuracy	Loss	Epoch
	Validation	Validation	Validation
Model 7b	85.7%	0.45941567	237

Table 13: Model performance



(7.1)



(7.2)

Figure 9: Evolution of Training and Validation Loss/Accuracy for Model 7

In Table 13, we can see that the highest validation accuracy was 85.7% at epoch 237. The model starts stabilizing after epoch 237 and doesn't get a higher accuracy, this can also be seen in Figure 7.1 where the validation loss is staying at ≈ 0.5, while the training loss is decreasing.
Furthermore, both Figures look close to Model 4 (Table 8), which could be because of the increase in learning rate and the lowering of the momentum.

Test set accuracy

The test set with 10,000 images was ran on model 4 (Table 6), as it had the highest validation accuracy.

	Test Accuracy	**Validation** Accuracy
Model 4	83.01%	85.9%

Table 14: Model 4 test set performance

The model performed well on the test set with a 2.89% drop in accuracy.

1.4 Questions

1. What does the momentum parameter of SGD do?
In usual SGD learning takes very long, however using the method of momentum the learning is accelerated. This is done by accumulating an exponentially decaying moving average of past gradients where the momentum continues to move in their direction. The momentum hyperparameter is in the form of $\frac{1}{1-\alpha}$, where α is the hyperparameter that controls the velocity of the momentum.

2. What are the differences and similarities between a convolutional and a fully-connected layer?
A fully-connected and a convolutional layer are both used in classification. However a fully-connected layer is much more computationally expensive than a convolutional layer, as it connects to all neurons in the previous layer, whereas a convolutional layer is only connected to a few local neurons.
A fully-connected layer is used to compute weights and classify the images, however a convolutional layer is used for convolution which creates a 2-D feature map.

3. What is the softmax layer and why is it the last layer of the network?
The softmax layer is used to create a probability distribution over n possible values.
Additionally, when using a discrete variable with n values it produces a vector $\hat{\mathbf{y}}$ which sums up to 1 so that it represents a valid probability distribution.
Softmax is mostly used as the last layer of a network because it needs to represent the probablity distribution over n classes.

4. Why do we use the cross entropy loss function for classification?
The cross-entropy loss function greatly improves performance of models with a sigmoid or softmax output, which suffer from saturation and slow learning when using the mean squared error loss. As classification uses a softmax output, cross-entropy loss improves the performance compared to the MSE loss.

5. What does dropout do?
Dropout is a hyperparameter that prevents overfitting. It temporarily removes (drops out) units which are hidden or visible in the neural network and all their incoming and outgoing connections. Each unit is removed with a fixed probability p.