1 Introduction

In part 1, the Bag-of-Words model makes use of hand-crafted features to train classifiers. In this task, we use a data-driven approach to perform image classification task. In this assignment, we use two different neural networks to perform classification and explore how the hyperparameters settings affect the model performance.

2 Datasets

CIFAR-10 The CIFAR-10 dataset[6] consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Figure 1: Visualization of the CIFAR-10 dataset.

STL-10 The STL-10 dataset[5] is an image recognition dataset for developing unsupervised feature learning, deep learning, self-taught learning algorithms. It consists of 96x96 pixels images and contain 10 categories, each with 500 training images and 800 test images. It is inspired by the CIFAR-10 dataset but with some modifications. In particular, each class has fewer labeled training examples than in CIFAR-10, but a very large set of unlabeled examples is provided to learn image models prior to supervised training. The primary challenge is to make use of the unlabeled data (which comes from a similar but different distribution from the labeled data) to build a useful prior.

3 CNNs for Image Classification

3.1 Neural Networks

Neural network is a set of neurons organized in layers. Each layer contains many neurons and they are fully connected with neurons in previous layer(As shown in fig 2). These neurons take the input from the previous layer, multiplies it by it's weights, and then passes the sum through the activation function to the other neurons(see fig 3). The *TwolayerNet* class in the experiment uses two-layer network with fully connected layers.

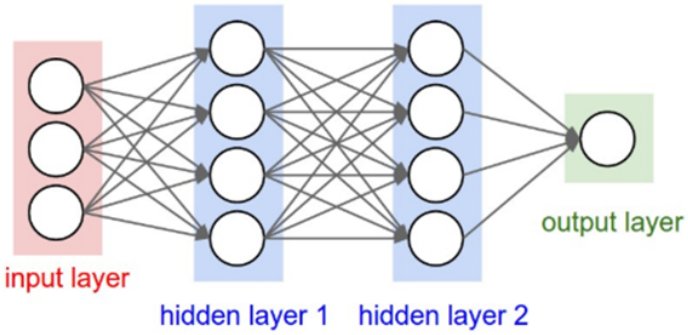


Figure 2: Two Layer Neural Network[2]

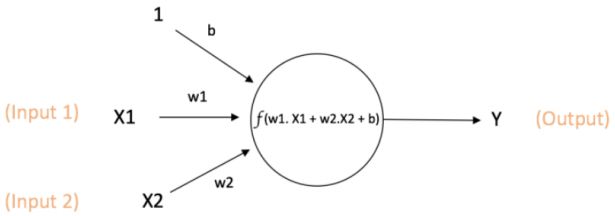


Figure 3: Computation in A Single Neuron[4]

3.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are generally used in computer vision. They are formed by Convolutional Layer, Pooling Layer, and Fully-Connected Layer. In the forward propagation, the input goes through Convolutional Layer, Pooling Layer and repeats the previous steps until it goes to Fully-Connected Layer to produce the final result. A CNN architecture example is shown as fig 4.

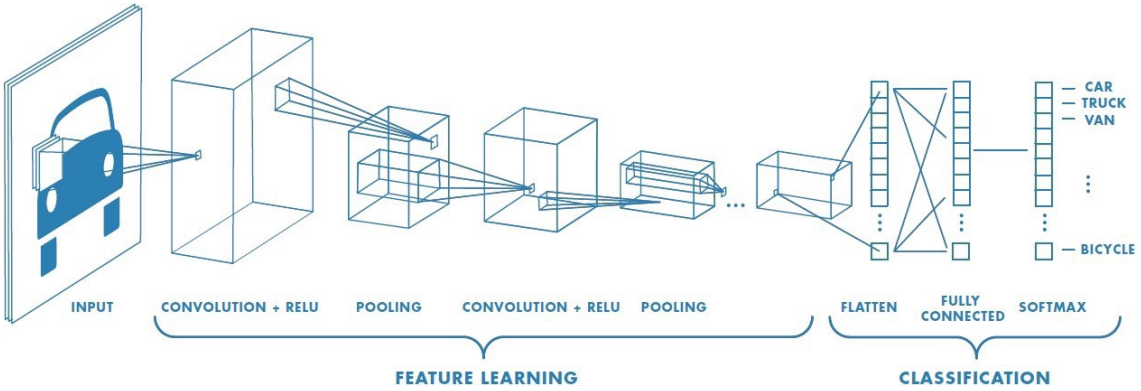


Figure 4: CNN networks architecture[1]

In the convolutional layer, the input data will go through a convolution function and activation function. In the convolution function, a sliding window called filter or kernel is applied to the input volume, which will produce a result of convoluted features. Its computation is shown in fig 5. The result acts as input to the activation function (it's a linear function called ReLu) to produce the output of the convolutional layer. Following the Convolution layer is a pooling layer. The pooling layer helps reduce the spatial size of the representation to reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The pooling layer operates in a similar way as convolution. The filters slide the input volume of this layer and calculate the average (Average Pooling) or maximum value (Max Pooling) of the selected window.

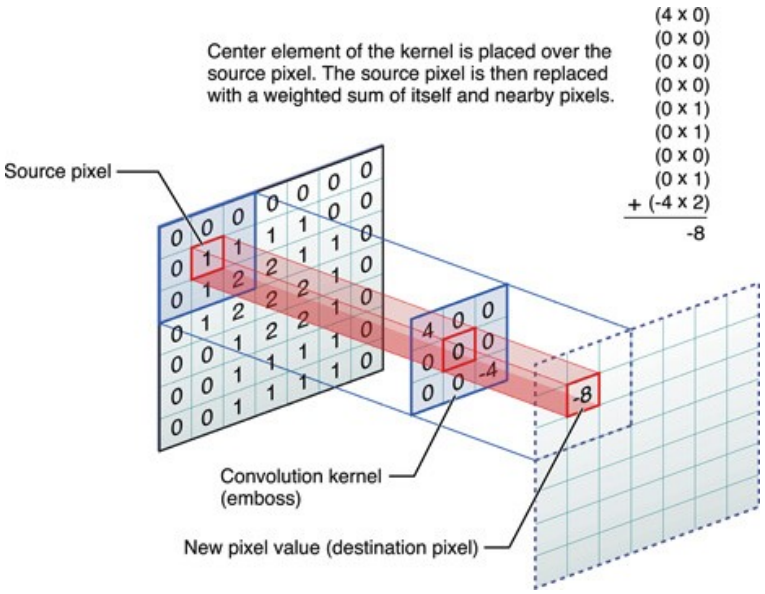


Figure 5: calculation with filter on CNN[3]

3.3 LeNet-5

LeNet-5[7] is a convolutional neural network structure. It comprises 7 layers, not counting the input, all of witch contain trainable parameters(weights)(see fig 6).

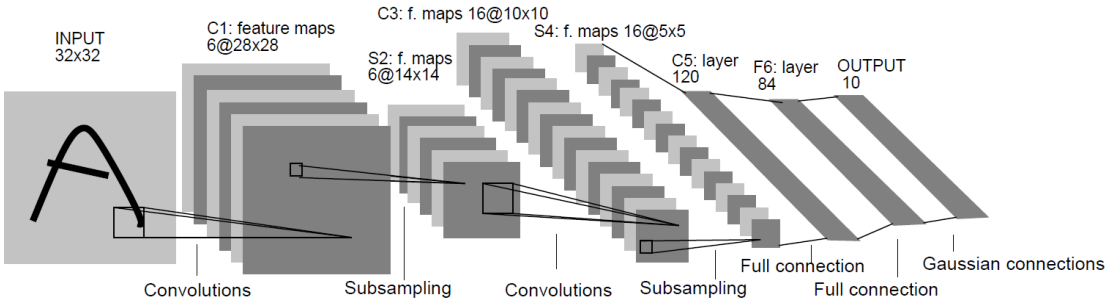


Figure 6: Architecture of LeNet-5[7]

The input is a 32x32 pixel image. Layer C1 is a convolutional layer with 6 featrue maps. Each unit in each feature map is connected to a 5x5 neighborhood in the input. The size of the feature map is 28x28 which prevents connection from the input from falling off the boundary. C1 contains 156 trainable parameters, and 122,304 connections. Layer S2 is a sub-sampling layer with 6 feature maps of size 14x14. Each unit in each feature map is connected to a 2x2 corresponding feature map in the C1. The four inputs to a unit in S2 is added, then multiplied by a trainable coefficient, and added to a trainable bias. The result is passed through a sigmoidal function. The 2x2 receptive fields are non—overlapping, therefore feat tire maps in S2 have half the number of rows and column as feature maps in Cl. Layer S2 has 12 trainable parameters and 5,880 connections. Layer C3 is a convolutional layer with 16 feature maps. Each unit in each feature map is connected to several 5x5 neighborhoods at identical locations in a subset of S2's feature maps. Layer C3 has 1,516 trainable parameters and 151,600 connections. Layer S4 is a sub-sampling layer with 16 feature maps of

size 3x5. Each unit in each feature map is connected to a 2x2 neighborhood in the corresponding feature map in C3, in a similar way as C 1 and S2. Layer S4 has 32 trainable parameters and 2,000 connections. Layer C5 is a convolutional layer with 120 feature maps. Each unit is connected to a 5x5 neighborhood on all 16 of S4's feature maps. Layer C5 has 48,120 trainable connections. Layer F6 contains 84 units and is fully connected to C5. It use average pooling and has 10,164 trainable parameters. Layer F7 is a fully connected softmax output layer with 10 possible values corresponding to the digits from 0 to 9.

3.4 Hyperparameters

Model hyperparameters are the properties that govern the entire training process. They determine how many images to be processed at each step, how much the weights of the network will be updated,how many iterations will the network run until convergence. In the experiment, we adjust hyperparameters Learning rate, Batch size, Number of epochs, optimizer and transform function to see how it affect the performance of our models.

4 Transfer Learning

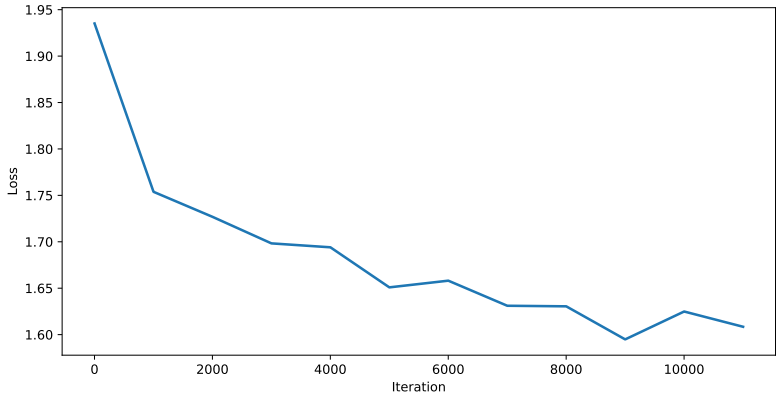
Transfer learning[8] is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. In practice, very few people train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pretrain a ConvNet on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the ConvNet either as an initialization or a fixed feature extractor for the task of interest. The following are three major transfer learning methods. The first strategy is to use ConvNet as fixed feature extractor. Take a ConvNet pretrained on ImageNet, remove the last fully-connected layer, then treat the rest of the ConvNet as a fixed feature extractor for the new dataset. The second strategy is to not only replace and retrain the classifier on top of the ConvNet on the new dataset, but to also fine-tune the weights of the pretrained network by continuing the backpropagation. It is possible to fine-tune all the layers of the ConvNet, or it's possible to keep some of the earlier layers fixed (due to overfitting concerns) and only fine-tune some higher-level portion of the network. The final strategy is to use pretrain models. Since modern ConvNets take 2-3 weeks to train across multiple GPUs on ImageNet, it is common to see people release their final ConvNet checkpoints for the benefit of others who can use the networks for fine-tuning. For example, the Caffe library has a Model Zoo where people share their network weights.

5 Experiment and Results

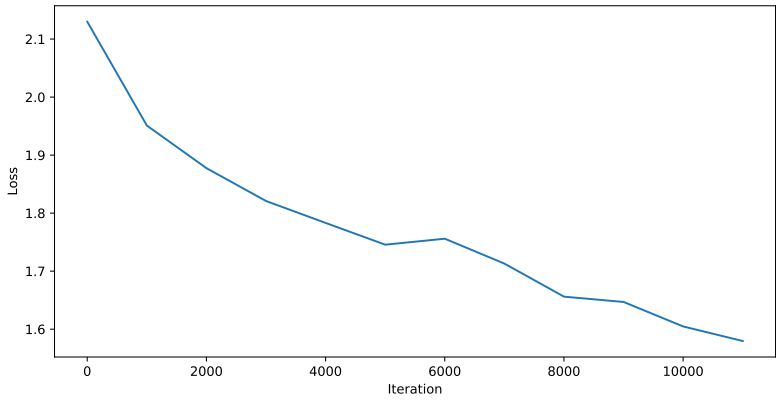
5.1 Implementation of TwoLayerNet and ConvNet

Our implementation of the convolutional network is very similar to the LeNet one. We replaced the slightly more complex pooling layers with average pooling and used a simple tanh activation instead of the parametrized one from LeNet. For the fully-connected network in this part of the assignment, we used 50 hidden units.

After having implemented both networks, we ran each one for one epoch on the CIFAR-10 training set. For the optimizer we used stochastic gradient descent (SGD) with a learning rate of 0.001, momentum of 0.9 and batch size 4. We chose the cross-entropy loss function to measure the performance of our networks. The loss curves for both networks are shown in Figure 7. As we can see, the loss is mostly decreasing with each iteration, i.e. the performance on the training set is getting better. We can also see though that at some iterations the loss is getting bigger. This is because SGD uses an estimate of the true error and therefore not all updates result in an overall improvement. When comparing both loss curves it is noticable that the curve for the two layer network already flattens at the end which indicates that it is approaching convergence, while the curve for the convolutional network is still very steep at the end. This suggests that the convolutional network has more potential than the two layer net and that it can still be trained further to improve performance. This was expected since in general CNNs work better for image processing tasks.



(a) Two-layer fully connected network



(b) Convolutional network

Figure 7: Loss curves for the CNN and fully-connected network for one epoch on the CIFAR-10 training set.

5.2 Implementation of Dataset Class

In this part of the assignment we implemented our own dataset class using the PyTorch wrapper. We load the raw data with the `pickle` package and store the images and their corresponding labels in two separate Python lists. In order to be able to apply `torchvision.transforms.ToTensor`, which transforms Numpy array to Torch tensors, we had to store the images in the $H \times W \times C$ format. This dataset class could then be used in the same way as the built-in dataset from PyTorch. To show this, we trained the same convolutional network from the previous part with the same parameter settings for one epoch using our custom class. The resulting learning curve is shown in Figure 8. As expected, the result doesn't differ very much from the curve in Figure 7(b) since the method of how the data is loaded doesn't influence the optimization process. The minor differences are due to the stochasticity of the SGD optimizer.

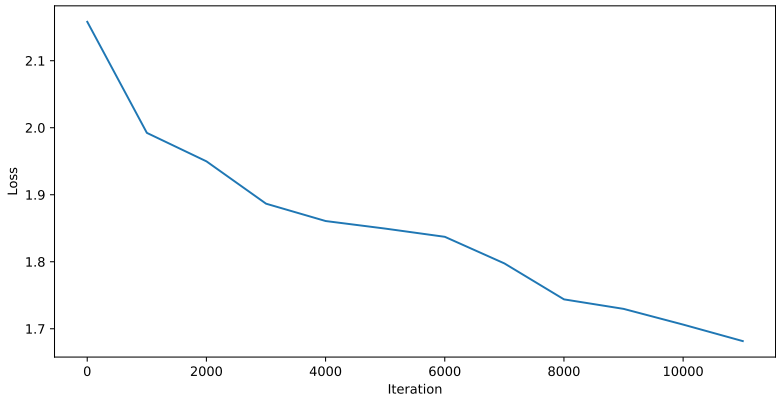


Figure 8: Loss curve for the CNN using our custom CIFAR-10 dataset class.

5.3 Setting up the hyperparameters

In this section we try to experiment with both architectures in order to get the most performance out of them. We rework the general architectures and also try out several hyperparameter setting.

First, we updated the architecture of our convolutional network by replacing the average pooling layer with a max-pooling layer. Furthermore, we also replaced the tanh activations with ReLU activations. The ReLU function is a more popular choice for most modern day deep learning projects since it suffers less from the vanishing gradient problem and requires much fewer computational resources compared with the tanh activation.

Furthermore, we also changed our optimizer to Adam, which has the ability to automatically adjust the learning rate over time and is more robust than SGD. After that, we investigated three different hyperparameters: batch size, learning rate and the hidden layer units in both the two layer fully connected network and the classifier of the convolutional network. We sweeped through 20 different combinations of hyperparameters using random search and computed the validation and training accuracy after five epochs for the CNN and two epochs for the two layer net since the latter one seemed to converge faster. The results for the different hyperparameter setting are listed in Table 1 for the CNN and Table 2 for the two layer net. Overall, we were able to see that the gap between training and validation accuracy became larger when the number of hidden units increased, which is caused by overfitting due to the higher model complexity. Furthermore, we noticed that a small learning rate together with a small batch size leads to extremely poor results. This is because a small batch size leads to a worse error estimation per iteration, i.e. using a large learning rate in this case might cause a large update of the weights into the wrong direction. A large batch size with a small learning rate did also result in a worse performance when compared to a similar setup with a higher learning rate. That is caused by the slower convergence of the loss when using a smaller learning rate.

After examining the hyperparameters sweep and further tuning them a bit manually, we ended up with the setting (lr = 0.001, batch size = 32, hidden size = 54) for the CNN and (lr = 0.0009, batch size = 64, hidden size = 72) for the two layer net. In the next step, we investigated the optimal number of epochs by training both networks for 50 epochs and computing the validation and training accuracy after each one. The loss and accuracy curves can be seen in Figure 9 for the two layer net and Figure 11 for the CNN. While we can see that the loss curve decreases monotonically for both networks, the gap between the validation and training accuracy becomes increasingly larger. This is because the networks start to strongly overfit after a certain epoch, which even leads to a decrease in validation performance at some point. The best validation performance was achieved after around ten epochs in both cases.

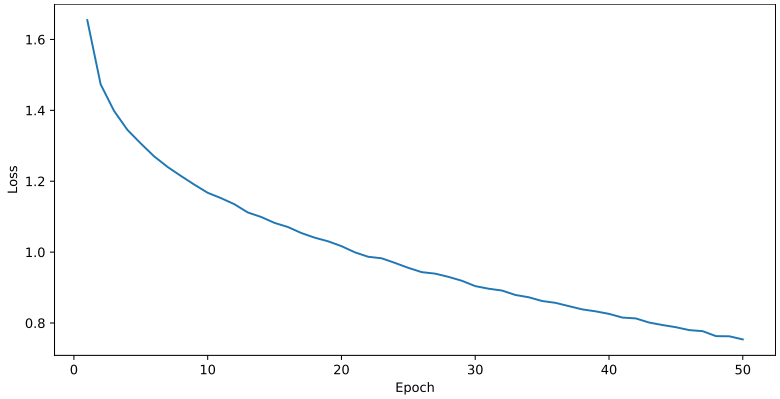
With the final hyperparameter settings we achieved a validation accuracy of 51 % with the two layer net and 57 % with the CNN. This further proves our initial claim that CNNs perform better then two layer nets in image processing tasks.

learning rate	batch size	hidden size	train acc	val acc
0.0060	8	54	0.100	0.100
0.0001	8	56	0.488	0.481
0.0001	8	71	0.493	0.493
0.0001	32	87	0.479	0.471
0.0018	8	75	0.542	0.514
0.0008	4	73	0.596	0.540
0.0061	4	57	0.184	0.181
0.0008	64	61	0.552	0.530
0.0010	32	54	0.579	0.545
0.0028	16	85	0.537	0.511
0.0029	4	81	0.414	0.414
0.0039	8	77	0.436	0.422
0.0001	64	50	0.439	0.441
0.0084	16	77	0.299	0.299
0.0001	32	91	0.484	0.478
0.0002	64	75	0.462	0.464
0.0019	32	54	0.596	0.541
0.0009	16	96	0.600	0.552
0.0021	64	50	0.586	0.544
0.0011	8	62	0.574	0.530

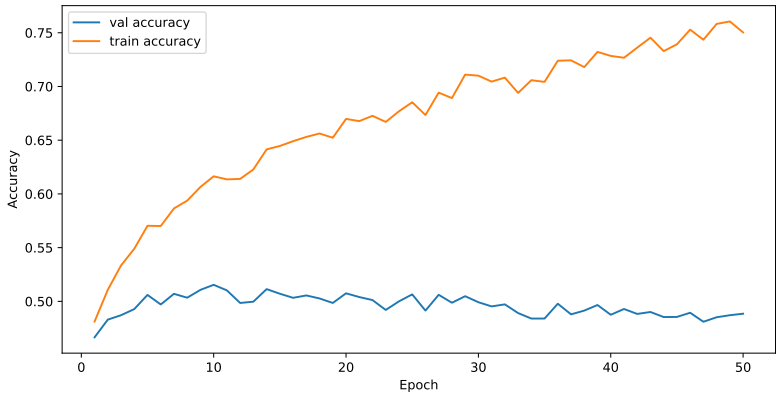
Table 1: Validation and training accuracy for different hyperparameter setups for the CNN.

learning rate	batch size	hidden size	train acc	val acc
0.0003	4	59	0.501	0.469
0.0005	16	64	0.498	0.471
0.0028	16	74	0.444	0.427
0.0054	32	73	0.406	0.388
0.0007	32	69	0.509	0.471
0.0002	8	65	0.508	0.479
0.0071	4	82	0.224	0.224
0.0050	16	94	0.362	0.360
0.0015	4	92	0.428	0.414
0.0003	32	67	0.507	0.479
0.0009	64	72	0.511	0.482
0.0004	4	53	0.499	0.468
0.0003	16	55	0.498	0.466
0.0100	64	72	0.364	0.362
0.0085	32	69	0.315	0.300
0.0003	65	66	0.496	0.473
0.0001	16	81	0.488	0.468
0.0003	16	96	0.514	0.482
0.0007	64	59	0.508	0.478
0.0032	4	58	0.342	0.323

Table 2: Validation and training accuracy for different hyperparameter setups for the two layer net.

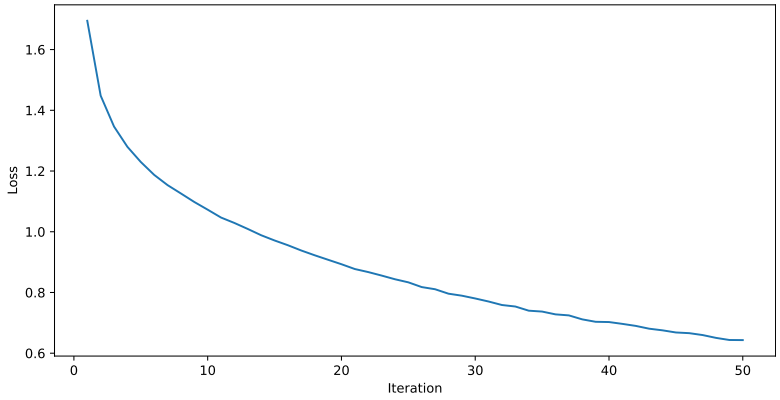


(a) Loss curve

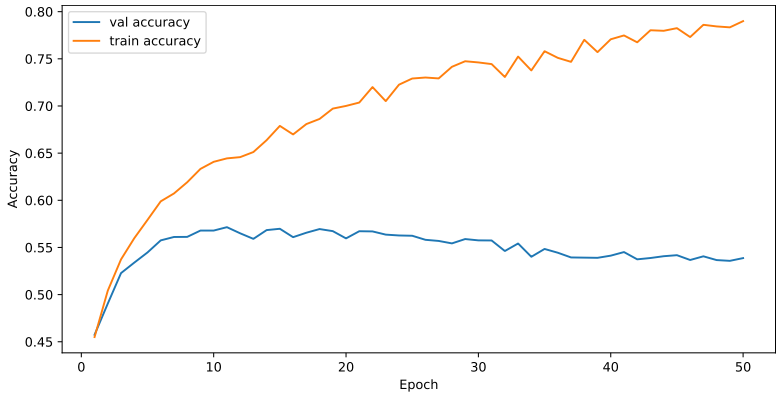


(b) Training and validation accuracy

Figure 9: Loss and accuracy curves for the two layer net after optimizing the hyperparameters.



(a) Loss curve



(b) Training and validation accuracy

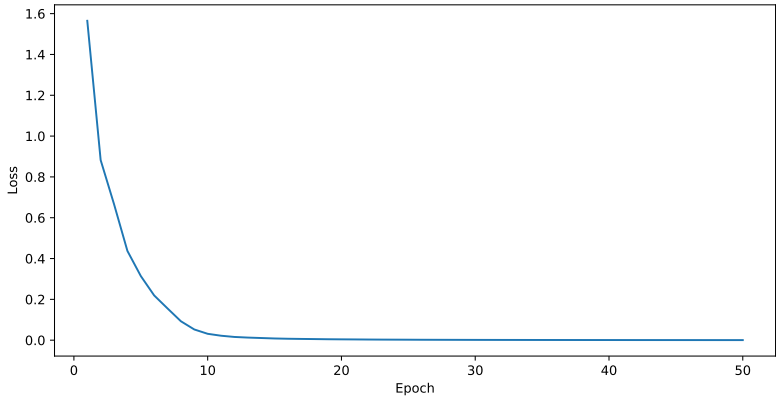
Figure 10: Loss and accuracy curves for the CNN after optimizing the hyperparameters.

5.4 Transfer Learning on STL-10

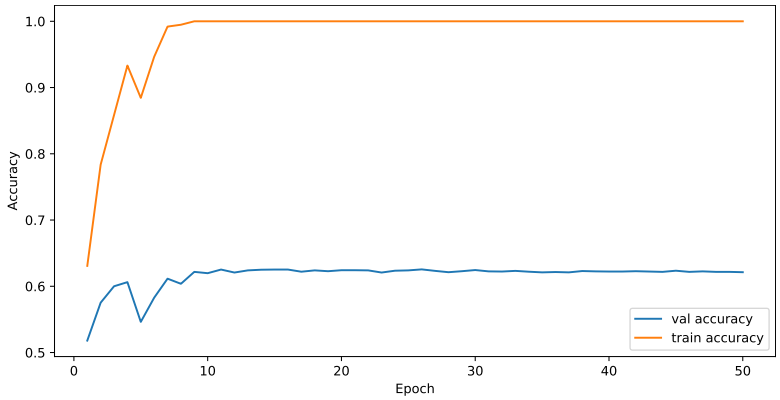
After working with CIFAR-10 dataset in the previous tasks, we now apply transfer learning to use our pre-trained network on the STL-10 dataset. We first implement a custom dataset class for the new dataset similar to how we did it in Section 5.2. While the STL-10 dataset consists of ten classes, we only use five of them here, which are airplane, bird, ship, cat and dog. Therefore, we need to extract those specific classes in our custom dataset class.

To classify the new samples, we use the pre-trained CNN feature extractor from the best model that we trained in Section 5.3. Since the STL-10 images are larger then the CIFAR-10 images and we only want to classify five classes now, we need to swap the classifier of the CNN. To work on the new dataset we replace the first fully connected layer with one that has an input size of $120 \cdot 17 \cdot 17$ and output of 57 and the second fully connected layer is replaced by one that has an output of size five. Furthermore, we also freeze the weights of our the feature extractor (i.e. all the convolutional layers) in order to prevent overfitting since the STL-10 training set is relatively small.

We train the model with the same setup as our best model from Section 5.3, i.e. we use the Adam optimizer with a learning rate of 0.001 and a batch size of 32. We run the training loop for 50 epochs and save the best model based on highest validation accuracy during this process. The resulting loss, training and validation accuracy curves can be seen in Figure 11. While we already observed overfitting in Section 5.3, it seems to be much stronger for this dataset. This is indicated by the large gap between validation and training accuracy and the loss being already zero after epoch 20. The reason why the overfitting is so strong here is because the dataset is so small. To improve the performance, more regularization techniques like dropout need to be applied. One further thing to notice is that the overall classification accuracy improved to 62 % from the 57 % with CIFAR-10 despite the strong overfitting. This is because we are only dealing with five classes now in contrast to the ten classes from CIFAR-10.



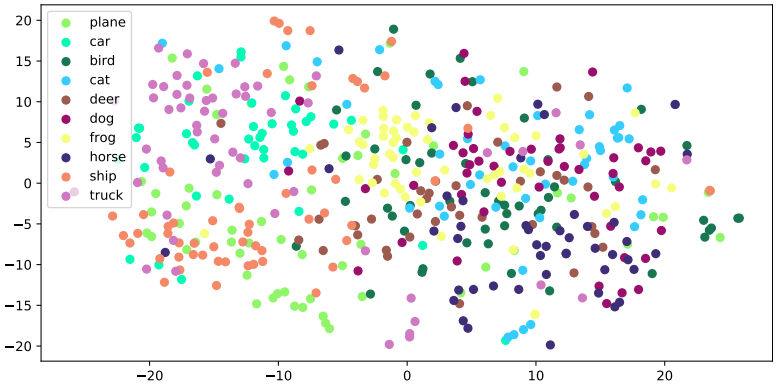
(a) Loss curve



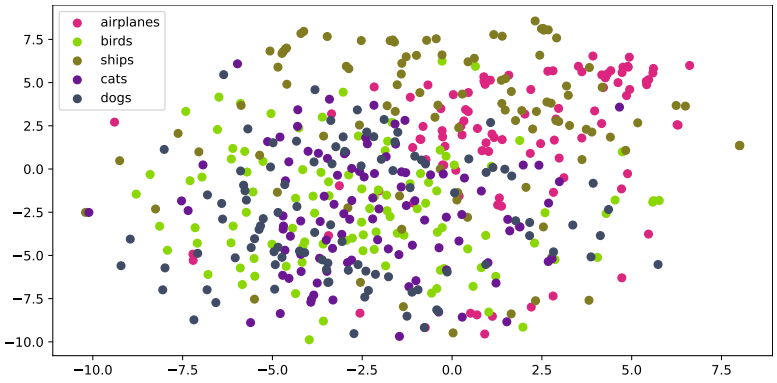
(b) Training and validation accuracy

Figure 11: Loss and accuracy curves using the CNN on STL-10.

Finally, we visualize the features of the network that was just trained on CIFAR-10 and the network that was used for transfer learning on STL-10. We map the feature space to a two-dimensional space by using t-sne, which is a dimensionality reduction algorithm. We use the built-in (TSNE) class from the sklearn package. The visualizations for 500 test samples from both datasets are shown in Figure 12. In both scatter plots we can see that there are visible clusters for each class. Even though samples from one class are grouped together they are not linearly separable from the other classes in this two-dimensional representation. This shows that our model still has plenty of uncertainty and is not able to correctly distinguish samples from different classes, which confirms our rather mediocre validation performance of 57 % for CIFAR-10 and 62 % for STL10. Furthermore, we can see that the clusters for the CIFAR-10 dataset are stronger mixed together while the clusters for the STL-10 dataset are slightly more compact, which also confirms our increase in performance of about 5 %.



(a) CIFAR-10



(b) STL-10

Figure 12: T-SNE visualizations of the features extracted by the CNN for the CIFAR-10 and STL-10 dataset.

6 Conclusion

In this part of the final lab project we gained hands-on experience on image classification using neural networks, especially convolutional neural network. We worked with two image datasets, namely CIFAR-10 and STL-10 and learnt how to apply transfer learning in order to use pre-trained networks. For the transfer learning, we saw how problematic it can be to have a too small dataset in form of the STL-10 set, which can lead to very strong overfitting and needs further improvement.

References

[1] A comprehensive guide to convolutional neural networks — the eli5 way. `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`.

[2] Introduction to neural networks. `https://www.i2tutorials.com/introduction-to-neural-networks/`.

[3] Multi-modal neural machine translation. `https://www.slideshare.net/SebastianRuder/multimodal-neural-machine-translation-iacer-calixto`.

[4] A quick introduction to neural networks. `https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/`.

[5] Coates, A., Ng, A., and Lee, H. An analysis of single-layer networks in unsupervised feature learning. G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15 of *Proceedings of Machine Learning Research*, JMLR Workshop and Conference Proceedings, pp. 215–223.

[6] Krizhevsky, A. Learning multiple layers of features from tiny images. Tech. rep., 2009.

[7] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[8] Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. CNN features off-the-shelf: an astounding baseline for recognition. *CoRR abs/1403.6382* (2014).