Comuputer Vision Final Project Part 2

Maximilian Schlögel UvA
Anca-Diana Vicol VU
Benjamin Kolb UvA
Philipp Lintl UvA

1 Introduction

Part two of the final project deals with the application of a Convolutional Neural Networks (CNN) for image classification. The underlying dataset divides images into 10 classes, whereas our version is a subset of 5 classes. We draw on a technique called *Transfer Learning*, which relies on a model, that has been trained on another, possibly larger dataset. In the report, we compare the accuracy of our fine tuned CNN and a classifier, that once takes the original pretrained and once the finetuned features as input.

2 The Tasks

The pretrained network `pre trained model.mat` was initially trained on the `CIFAR-10` dataset. This consists of $32 \times 32 \times 3$ images of 10 categories. According example images are depicted in Figure 1-5.

Figure 1: Example image of class horse

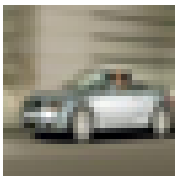


Figure 2: Example image of class car

Figure 3: Example image of class bird

Figure 4: Example of class airplane

Figure 5: Example image of class ship

The dataset subject to our analysis is `STL-10`, which originally contains images of size $96 \times 96 \times 3$. Though assigning 10 categories, we limit the data to five categories only. Namely: *airplanes, birds, ships, horses* and *cars*.

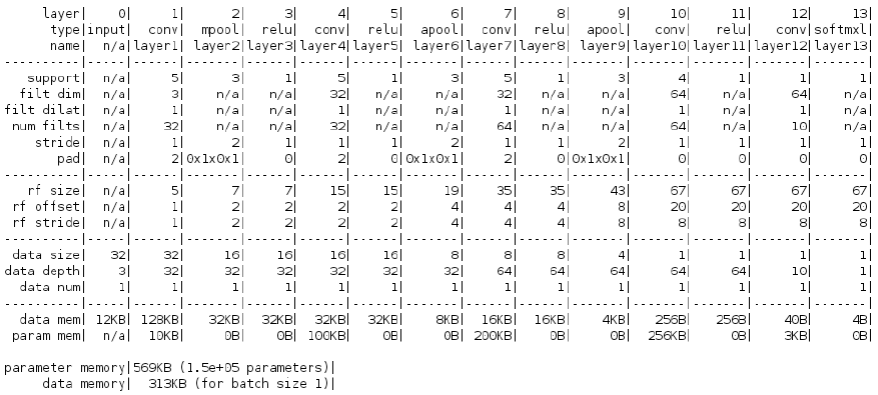
2.1 Convolutional Neural Networks

CNNs are neural networks, that learn filters in order to extract features, which then are concatenated to serve as input to a standard classifier such as a Support Vector Machine (SVM) or Multi Layer Perceptron (MLP). In several so called convolutional layers, weight sharing and local connectivity reduce the number of parameters and allows for robust filters to be learned.
In our task, we are given such a CNN with the following structure:

2.2 Understanding the Network Architecture

The architecture of the displayed network follows the general structure as such. The model contains convolutional (conv) layers, which are downsampled via average and max pooling and activated via non-linear activation function RELU. The conv layers are followed by one fully connected layer on top. Regarding the pooling of the convolutional layers, it is noticeable, that at first maxpooling in

Preprint. Work in progress.



layer	0	1	2	3	4	5	6	7	8	9	10	11	12	13
type	input	conv	mpool	relu	conv	relu	apool	conv	relu	apool	conv	relu	conv	softmxl
name	n/a	layer1	layer2	layer3	layer4	layer5	layer6	layer7	layer8	layer9	layer10	layer11	layer12	layer13
support	n/a	5	3	1	5	1	3	5	1	3	4	1	1	1
filt dim	n/a	3	n/a	n/a	32	n/a	n/a	32	n/a	n/a	64	n/a	64	n/a
filt dilat	n/a	1	n/a	n/a	1	n/a	n/a	1	n/a	n/a	1	n/a	1	n/a
num filts	n/a	32	n/a	n/a	32	n/a	n/a	64	n/a	n/a	64	n/a	10	n/a
stride	n/a	1	2	1	1	1	2	1	1	2	1	1	1	1
pad	n/a	2	0x1x0x1	0	2	0	0x1x0x1	2	0	0x1x0x1	0	0	0	0
rf size	n/a	5	7	7	15	15	19	35	35	43	67	67	67	67
rf offset	n/a	1	2	2	2	2	4	4	4	8	20	20	20	20
rf stride	n/a	1	2	2	2	2	4	4	4	8	8	8	8	8
data size	32	32	16	16	16	16	8	8	8	4	1	1	1	1
data depth	3	32	32	32	32	32	32	64	64	64	64	64	10	1
data num	1	1	1	1	1	1	1	1	1	1	1	1	1	1
data mem	12KB	128KB	32KB	32KB	32KB	32KB	8KB	16KB	16KB	4KB	256B	256B	40B	4B
param mem	n/a	10KB	0B	0B	100KB	0B	0B	200KB	0B	0B	256KB	0B	3KB	0B

parameter memory|569KB (1.5e+05 parameters)|
data memory| 313KB (for batch size 1)|

Figure 6: Network Architecture of pretrained CNN model

combination with relu activation is applied, whereas in the remaining convolutional layers, average pooling is drawn on. The latter layers pool after the activation part. This makes sense as maxpool does not change with negative values. Average pooling on the other hand does, which is why the order is changed.
The receiptive field size increases monotonically from 5 up to 67, similarly for the stride. This is in line with the design of convnets which cover low level features in the first layers and semantic features in the last ones. Therefore, smaller recepteive fields learn smaller or in other words more low level features.
The number of filters increases from 32 in the first conv layer to 64 and then 10 in the fully connected layer. The last number is in line with the number of classes.
Filter dimension ranges from 3 in the first to 64 in the final conv layer, which is also in line with the notion of covering low level features in the first layers up to higher level and semantic level features in the end.
Most parameters and biggest size are also looked at. To be more specific, largest memory consumed by parameters can be found in conv Layer 10 with 256 KB. This is caused by the number of filters as well as dimension being 64, whereas the incomming data from previous layers is of depth 64.

3 Preparing the Input Data

3.1 Setting up the Hyperparameters

While looking into hyperparameters tuning we experimented with *number of epochs, batchsize* as advised in the assignment sheet. However, as the given default values resulted in `NA` values for the objective, we chose to also tune the *learning rate previous model, learning rate new layers, weight decay*, as well as the decrease of learning rate in time.
We encountered several scenarios. For too small learning rate values, the overall prediction error converged at around 30% with a difference of 20 % between train and test images, which we consider as overfiting. Too large learning rates on the other hand yielded similar results. Both scenarios are distinguished by a quick drop on error, as well as a large gap between training and test error early on as exemplary seen in Figure 9. We therefore settled in between 0.01 and 0.05 with a slightly higher learning rate for the added layers, as the amount of training data is limited and therefore does not allow to fine tune the pretrained network properly. Furthermore, increasing the weight decay from 0.0001 to 0.0075, as well as allowing a higher learning rate for more epochs decreased the amount of overfitting and the overall prediction error.
In general, a lack of computational means and time prevented us from presumably finding a model with a tighter gap between train and test set, as well as lower prediction error in total. After trying several parameter settings, we found no substantial difference and as there was no other performance expectation than better than random, we settled for the values depicted in Table 1. A general observation from most suitable settings can be formulated as such:
The final prediction error on the test data seems to range from 25-18%. For settings yielding test errors beneath 20%, the training error drops very low (5%), which is an indication of overfitting, as

the training images are learned ”by heart”. The test error decreases as well, which is why we chose such a setting nonetheless (Figure 10). In general, crossvalidation should be drawn on to find proper hyperparameters independent of training / test allocation. Furthermore, inproper settings can lead to no learning at all, as depicted in Figure 8, which is caused by either a too little or too high learning rate on the new layers.
Proper tuning and additional adjustments would be subject to the bonus section. After adjusting the respective parameters, we tuned for the number of epochs and the batch size with the according result:

Hyperparameter	Chosen value
LR pretrained model	[0.02, 0.2]
LR new layers	[0.05, 0.5]
weight decay	0.0075
time adjustment	0.1*0.5*LR twice each 30 epochs, than 20 and 10.

Table 1: Used hyperparameters: LR = Learning rate

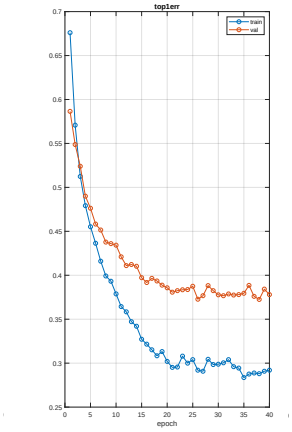


Figure 7: High test error

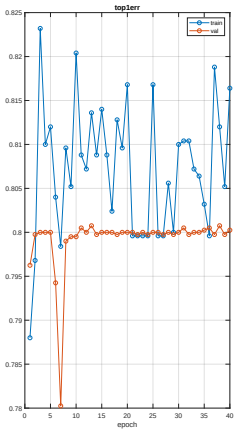


Figure 8: Learning rates too high/low.

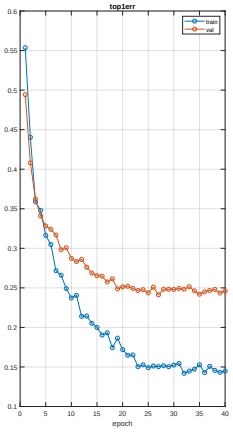


Figure 9: Improved but not end choice.

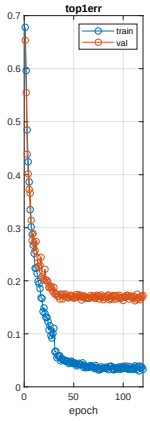


Figure 10: Final choice.

Parameters *number of epochs* and *batch size* are tuned for values $\{40, 80, 120\}$ of the former and $\{50, 100\}$ of the latter. The parameter settings are evaluated on our finetuned CNN itself, as well as a Supoport Vector Machine (SVM) model on top of the learned features of solely the pretrained and once the finetuned network. Results can be seen in Table 2

Batch size	**Number of epochs**	**CNN**	**PT-SVM**	**FT-SVM**
50	40	0.83	0.6908	0.8343
50	**80**	**0.83**	**0.6908**	**0.8343**
50	120	0.83	0.6908	0.8295
100	40	0.77	0.6908	0.7698
100	80	0.77	0.6908	0.7740
100	120	0.77	0.6908	0.7753

Table 2: Hyperparameter tuning for number of epochs and batch size.

Considering, our choice for learning rate values included tuning to some extent already, it is not surprising, that no performance difference between the number of epochs can be detected. Only, batchsize 50 outperforms batchsize 100 by around 6%, which is why we choose batchsize to be 50. Looking at Figure 10 reveals, that the algorithm seems to converge around 40 epochs, it is not atypical that more epochs do not significantly change the obtained weights. Especially, as the learning rates are decreased over time as well. The fact that no difference among the number of epochs is obtained it is more desireable to choose a lower number of epochs and therefore less computational

costs. That is, why our choice for the number of epochs is 80. We draw the conclusion, as Figure 10 does not yet seem converged at 40 epochs. We note, that for the choice of the learning rate values, it could have happened that we focused more on models trained on batchsize 50, which can distort the hyperparameter tuning.

4 Experiments

4.1 Feature Space Visualization

In order to visualize the feature space we used the tsne function on the results of the last convolutional layer. From our observations the feature matrix at this point is sparse which is not an appropriate input for the eig function that is used in tsne to compute the eigenvalues and eigenvectors. Thus we substituted eig for eigs in the script and chose initial dims to be equal to the small number of eigenvalues of the matrix (6).

In order to better understand the effects of finetuning we used the modified tsne function to reduce the dimensions of the feature space to 2 and plotted the results on the test set using both the pre trained and the tuned networks (Figures 11 and 12).

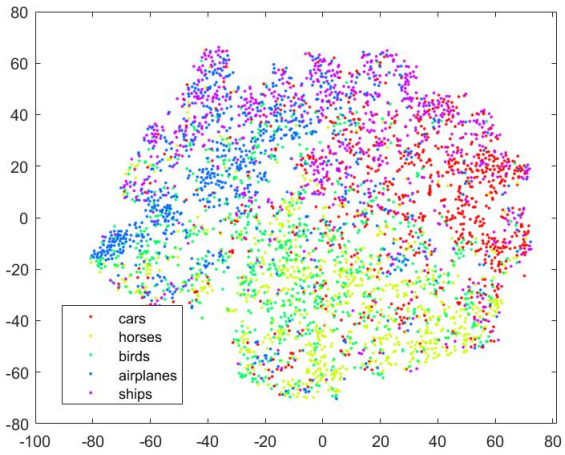


Figure 11: Tsne visualisation over the pretrained network features.

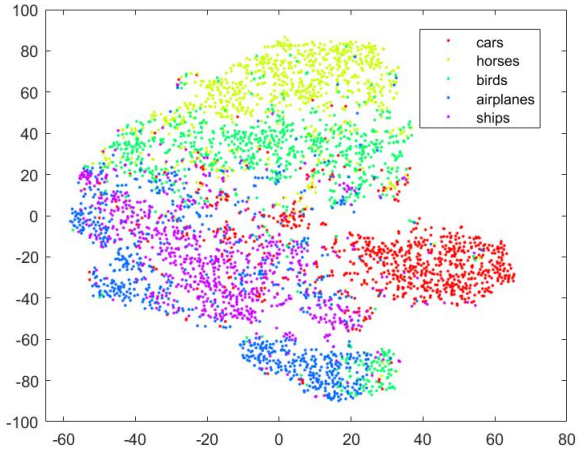


Figure 12: Tsne visualisation over the finely tuned network features.

As can be observed from the plots the features in the finely tuned network are much more separable leading us to believe that freezing the first layers of the network would not convey better results.

It is also interesting to notice that similar classes such as ships and airplanes or birds and horses are relatively close in the feature space.

4.2 Evaluating Accuracy

As we already provided the SVM performance on the learned feature is Table 2, we only need to compare it to the BOW results of part 1 of the final project. For the former, we can observe the following:
For both batchsize values, the accuracy of the CNN, that is finetuned on the pretrained model, is about equal to the respective performance obtained by a SVM, that is performed on the final layer of the fine tuned CNN, i.e the learned features. Performing the SVM on the features obtained by the pretrained model yields a noticeably lower accuracy. We therefore can reason, that finetuning the network improves the performance of a classifier, that is performed on the learned features. However, the amount of training images seem not to suffice to train the fully connected layers of the fine tuned CNN properly, as it should be able to outperform a standard classifier given enough data and

similarity to the dataset of the pretrained model.

5 Bonus section

All of the below calcuöated models drew on the parameter settings from above, batch size 50 and the number of epochs of 80.

5.1 Freezing early layers

One approach typical to Transfer Learning tasks is to **freeze** layers. As described above, Transfer Learning is the process of using a pre-trained model. This pretrained model then acts as a feature extractor, that does not require an enormous amount of data, as class related information from another data source is used. Depending on the similarity of images between our data source and the one from the pretrained model, as well as the amount of additional training data, it can be useful to also finetune the weights of the pretrained model. If, however, the domains and the images are very similar, it can also be more efficient, to not train the pretrained network, or in other words freeze the respective layers. In our case, the classes of our data set are exactly the same as in the CIFAR 10 source. Therefore, choosing not to backpropagate the errors from the task to the early layers, could save computational efforts. Additionally, the amount of training images is rather small. In this case, fine tuning could lead to overfitting. [2] That is, why the learning rates of the previous layer are set to zero and therefore the weights of the pretrained model are not adjusted in the backpropagations step. To guarantee comparability, we chose 80 epochs for batchsize 50 and left the remaining parameters unchanged. As seen in Table 3, freezing the layers of the pretrained model leads to a decrease in performance. More accurately, the CNN itself classifies only 60% of the test images, which is much lower than our fine tuned version from Table 1. In our opinion, there is a difference between the features detected from the pretrained data source the ones in our training data. So, the domains do not match that well, which is why fine tuning should be applied. Also, the final layers could be overfitting on the training images, without adjusting the weights of the previous layers. We furtherly noticed a much more instable behaviour of the error over the number of epochs (Figure 14).

Specific bonus part	**CNN**	**PT-SVM**	**FT-SVM**
Frozen layers (1)	0.60	0.6908	0.6908
Dropout (2)	0.79	0.6908	0.8103
More Fully connected Layers (3)	0.84	0.6905	0.833
Data Augmentation (4)	0.82	69.05	0.8103
(2) & (3) & (4)	0.82	69.05	0.8176

Table 3: Performance depending on the kind of adjustment.

5.2 Dropout

As seen in Figures 7-10, overfitting on the training data above the test set with a margin of minimally 10% seems unavoidable. This observation is typical to Deep neural nets, which are prone to overfitting, due to the large number of parameters. Another strategy to counteract overfitting is to randomly drop neurons with the respective connections during different training stages, which is called **dropout**. Not considering certain connections can overcome the problem of too similarly adopted units. [1] For our base CNN, we observed exactly that, as the obtained learned features only yielded a few eigenvalues for the tsne procedure. This means that there were several very similar units which is a sign of a high level of co-adaption. We added a dropout between the last convolutional layer and the first fully connected layer. The remaining parameters were set in accordance with the hyper parameter tuning. The results reveal a slightly lower accuracy compared to our final choice. However, looking at the progress over epochs shows a smaller degree of overfitting compared to the base CNN (Figure 13).

5.3 Deeper Networks / more fully connected layers

Another approach to improve the classification result is to either **deepen** the architecture within the conv layers as within AlexNet or VGG or simply add **more fully connected layers**. Considering that the convolutional layers learn features and the fully connected layers represent a Multilayer perceptron (MLP) classifier on top, we decided to add two additional fully connected layers. This MLP can detect the learned features more granularly, if there are more layers. We therefore extended the basic architecture of two additional FC layers with 32 hidden units. We chose the half of 64 by a heuristic to not add too many parameters. We base this decision on the small amount of training images and in general only to see a trend. It opens up room for further parameter tuning regarding the number of additional layers as well as respective hidden units. We do not notice any significant difference to the results of our base CNN.

5.4 Data augmentation

In order to improve the ability to generalize better on unseen test data, or in other words learn more robust features, **data augmentation** can be applied. This describes transformations to the original images to artificially increase the number of images. More accurately, each image can be subject to transformations per epoch to add some variation with a certain probability. We chose to flip the images horizontally, add a minor rotation or zoom into the image slightly. We observe a slightly smaller accuracy with comparable degree of overfitting.

5.5 Combination of all approaches

As a final adjustment, we combine dropout, deeper networks and data augmentation. Though the progress (Figure 15) promised the best accuracy so far, the actual performance was slightly lower than the base CNN. However, given properly tuned learning rates, weight decay and time adjustment, all these adjustments could have made an impact. Considering the smaller degree of overfitting compared to the base CNN and the smaller error compared to the CNN solely applying dropout, applying all of the adjustments seem useful.

5.6 Concluding remarks

To sum up the bonus section, freezing layers is not successfull, dropouts reduce the degree of overfitting, Data augmentation does not seem to impact the performance, just the same as using more fully connected layers at the end. However, combining the aforementioned approaches except for the freezing of previous layers, results in a less overfitted but still well performing model. In general, maybe not tuning the learning rate as we did, would have yielded bigger differences in the discussed approaches.

Further explorations could be to tune the remaining hyperparameters properly and to try different pretrained models such as AlexNet and VGG.

6 Conclusion

7 Separation of work

References

[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, and Y. Bengio. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 1929.

[2] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3320–3328. MIT Press, 2014.

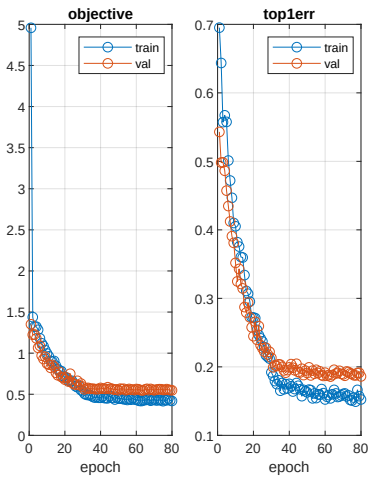


Figure 13: Dropout progression curve.

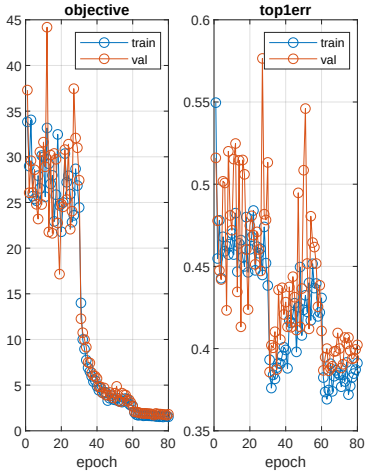


Figure 14: Pretrained layers frozen .

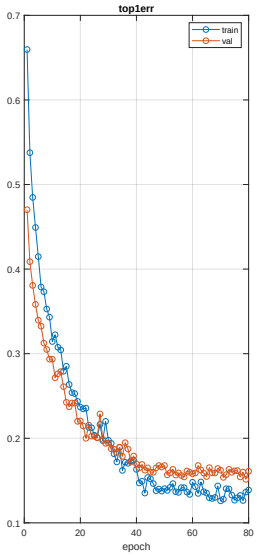


Figure 15: Applying dropout, data augmentation and more FC layers.