Computer Vision 1 - Final Project Part 2
CNNs for Image Classification

Wolf, Florian - 12393339 (UvA)
flocwolf@gmail.com

Biertimpel, David - 12324418 (UvA)
david.biertimpel@protonmail.com

Lindt, Alexandra - 12230642 (UvA)
alex.lindt@protonmail.com

Fijen, Lucas - 10813268 (UvA)
lucas.fijen@gmail.com

October 5, 2019

Introduction

In this part of the final project we explore image classification with convolutional neural networks (CNNs). Specifically, we make use of a pretrained CNN and fine-tune it so that it can be used to classify data samples of the *STL-10* dataset. In the course of this task, we analyze the architecture of a CNN and examine the accuracy of our fine-tuned network for different hyperparameters used in the fine-tuning process. Subsequently, we qualitatively and quantitatively evaluate the features extracted by the pretrained and the fine-tuned network. Finally, we compare the performance of the fine-tuned CNN with the *Bag-of-Words* (BoW) approach from the first part of the final project.

3 Training CNNs for Image Classification

3.1 Understanding the Network Architecture

The provided pretrained network consists of 14 layers, which are structured as follows:

$$
\begin{aligned}
&Input \\
&ConvLayer \rightarrow MaxPool \rightarrow ReLU \quad (1x) \\
&ConvLayer \rightarrow ReLu \rightarrow AveragePool \quad (2x) \\
&ConvLayer \rightarrow ReLu \rightarrow ConvLayer \\
&Softmax
\end{aligned}
$$

The input entering the network is of size $32 \times 32 \times 3$. For each Pooling layer in the network, the spatial dimension of the input is halved, which results in the input gradually shrinking to 4×4 while traversing the Pooling layers. The depth of the input is mapped to 32 by the first Conv layer and is further increased to 64 by the third Conv layer. The softmax layer in the end of the network maps to a 10-dimensional output vector representing the 10 classes of the *CIFAR-10* data set.

The fourth Conv layer further shrinks the spatial dimension of its input volume from 4×4 to 1×1. This effectively turns the subsequent Conv layer into a fully connected layer. Therefore, we can also write the $ConvLayer \rightarrow ReLu \rightarrow ConvLayer$ component of the network as $ConvLayer \rightarrow ReLu \rightarrow FC$, where FC represents the fully-connected layer.

We further observe the pattern $ConvLayer \rightarrow ReLu \rightarrow Pool$ in the first 10 layers of the network. This is because $ConvLayer \rightarrow MaxPool \rightarrow ReLu$ is equivalent to $ConvLayer \rightarrow ReLu \rightarrow MaxPool$, since the maximum element is selected by the Pooling layer regardless of the ReLu transformation.

A Conv layer's number of parameters can be determined by taking into account the size of the kernel, the depth of the input volume as well as the Comv layers's number of filters [1]:

$$\#parameters = (K_H \times K_W \times D \times F) + F,$$

where K_H and K_W are the height and width of the kernel, D is the depth of the input volume and F is the number of filters of the Conv layer. Adding F corresponds to the bias term introduced by each filter. In the given architecture the fourth Conv layer (the 11^{th} layer in the network) has the most parameters $[(4 \times 4 \times 64 \times 64) + 64 = 65600]$ and also consumes with 256 kilobytes most memory (layer with biggest size).

3.2 Preparing the Input Data

To preprocess the input data we implemented the `getIMDB` function. For this purpose, we take the *STL-10* data set that was also used for training the *Bag-of-Words* approach in the first part of this exercise. After resizing the input image to $32 \times 32 \times 3$, we normalize it and convert it to single precision. Afterwards we fill the variables `images.data`, `images.labels` and `images.set` such that each observation in the training and test data corresponds to its allocated label. In this process we only adopt observations of the classes *airplane*, *bird*, *ship*, *horse* and *car*. Finally, we include the variable `meta` which holds further information necessary for training. The size of the data end up being $[32 \times 32 \times 3 \times 6500]$.

3.3 Updating the Network Architecture

In the following, we want to continue training the pretrained network with our prepared data. This makes sense as the *CIFAR-10* (the data set the pretrained network is trained on) is very similar to the *STL-10* data set we use. Furthermore, training a network from scratch is computationally expensive and hardly possible in our case as the subset of the *STL-10* data set we use is likely too small. Therefore, we use transfer learning to fine-tune the weights of the pretained network with our training data. However, the *CIFAR-10* data sets contains 10 classes, while we only use 5 classes for our purposes. In order to make the pretrained network compatible with our training data, we need to adjust the size of the fully connected layer before the softmax layer. Since the output of the preceding layer does not change, the input size of the fully connected layer remains at 64 (`NEW_INPUT_SIZE=64`). Finally, to match our 5 classes we change the output size of the layer to 5 (`NEW_OUTPUT_SIZE=5`). The weights of this new layer are randomly initialized.

3.4 Setting up the Hyperparameters

Before starting the transfer learning, we find appropriate hyperparameters for the training of the network. We accomplish this both with empirical exploration and with a grid search approach.

We first empirically set the hyperparameters for `lr_prev_layers`), `lr_mew_layers`, the learning rate over the epochs and the weight decay. Then we perform grid search on different permutations of the batch size and number of epochs.

The hyperparameters `lr_prev_layers` and `lr_new_layers` function as additional weights on the learning rate applied during backpropagation. This allows the learning rate to be controlled individually

for each layer. Since we only want to fine-tune the already trained layers and train the new layer from scratch, we assess the default values of `lr_prev_layers = [.2, 2]` and `lr_new_layers = [1, 4]` as suitable. As for the learning rate over time, we decided to reduce it gradually over time. This decline is shown in Table 1. We did not change the weight decay as experiments with values away from the initial value of 0.0001 did not produce satisfactory results.

Decay of the learning rate over the epochs				
epochs in %	1-30	31-55	56-80	81-100
learning rate	0.05	0.005	0.0005	0.0001

Table 1: Describes the decay of the learning rate over the epochs. The epochs are given in percent to consider changing learning rates

After setting these hyperparameters, we use the grid search to find the optimal combination of batch size and number of epochs. For this purpose we successively set the batch size to 50 and 100 and the number of epochs to 40, 80 and 120. For each permutation we fine-tune the pretrained network with the new training data and report the accuracy on the test data. The results of the grid search can be found in the Table 2.

	Batch size	
# Epochs	**50**	**100**
40	0.86	0.82
80	0.86	0.86
120	0.87	0.86

Table 2: Describes the resulting accuracies of the fine-tuned models in the grid search process.

When looking at the results of the grid search, we immediately notice that most of the achieved accuracies are very similar, being either 86.0 or 87.0 percent. The only model with a considerably lower performance is the one that was trained over 40 epochs with a batch size of 100 having an accuracy of only 82.0 percent. Since the performance of the other models is similar, we choose the model that was trained over the smallest number of epochs, as it is the least susceptible to overfitting. In Figure 1, the training process of this model is illustrated.

4 Experiments

To examine the performance of the fine-tuned CNN we perform three experiments. In the first one, we visualize and compare the features extracted by pretrained and fine-tuned CNN. The second experiment is twofold: We first compute the accuracy of the fine-tuned CNN and then train two Support Vector Machines (SVMs) on the classification of feature vectors extracted by fine-tuned and pretrained CNN respectively. The accuracies of both SVMs can then be considered a measure of *how representative* the feature vectors extracted by fine-tuned and pretrained CNN are. Finally, we compare the performance of the fine-tuned CNN to the Bag-of-Words based image classification, which was performed in the first part of this project.

4.1 Feature Space Visualization

For visualizing the space of features extracted by our network, we consider the activations of its second-last layer (i.e the last convolutional layer before the softmax activation). These activations are

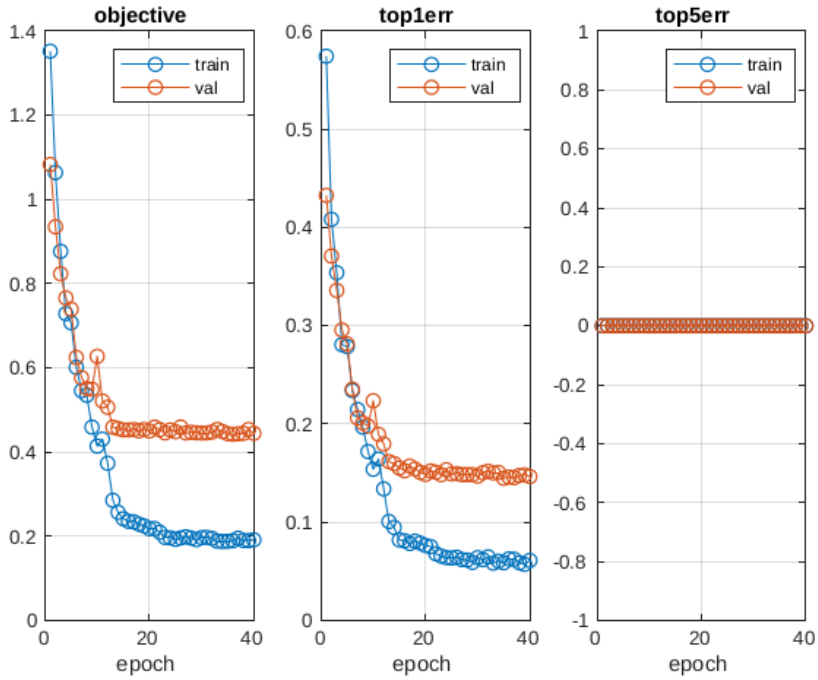

Figure 1: Illustration of the training process of the best performing model determined by the grid-search. The training objective, top-1 error and top-5 error are shown.

a 64-dimensional vector that represents higher level features of the respective input image. Therefore, we refer to these vectors as feature vectors and to the space of all possible feature vectors as feature space. To compare the features spaces of pretrained and fine-tuned network, we insert all test images into both networks and obtain the corresponding feature vectors from the second-last layer.

To visualize the obtained feature vectors per image class, we make use of the suggested `t_sne` method [2]. This dimensionality reduction method takes features with corresponding labels as input and reduces their dimensions to 2 while preserving as much spatial information as possible. The `t_sne` results for the features extracted by pretrained and fine-tuned network are displayed in figure 2. For the pretrained model, the features extracted from images of different classes are not that easy to distinguish, i.e. there is a considerable overlap of features of different classes. This means that the pretrained network tends to consider images of different classes to have similar high-level features. Since the final classification step is performed on these features, we can expect the pretrained classifier to perform not very well on our test data. In contrast, the fine-tuned network extracts more representative features from our test images. Features of the same class are clearly grouped together in feature space, overlap is almost only happening on the borders of such groups and in a few smaller areas between them. Therefore, we can expect the fine-tuned network to overall perform better on our test data than the pretrained network.

4.2 Evaluating Accuracy

In order to quantitatively evaluate the performance of the fine-tuned CNN, we measure not only its accuracy, but also its improvement during the fine-tuning process. We also assess the representation of the observations in the feature space learned by the CNNs. For this purpose, we use both the CNN pretrained on the *CIFAR-10* data set and the fine-tuned CNN as a feature extractor. For each image in the *STL-10* data set, we obtain its corresponding activations in the CNN's last hidden layer and treat them as a feature vector. We then train two linear SVMs, one with the features from the pretrained CNN and one with the features from the fine-tuned CNN. This results in three accuracies: that of the fine-tuned CNN, that of the SVM trained with the pretained features and that of the SVM

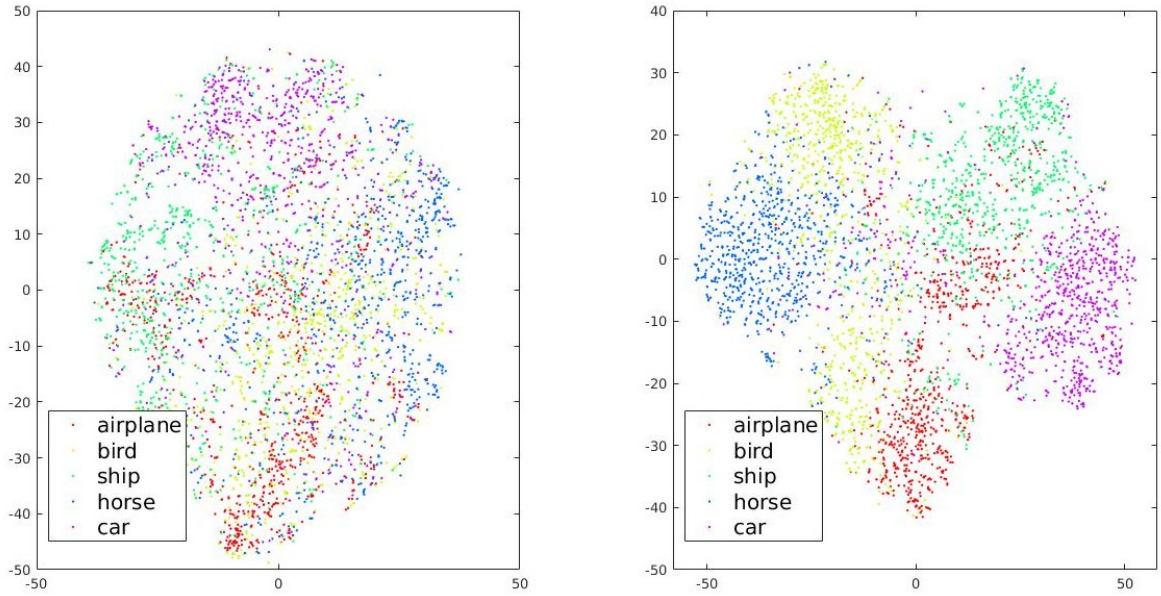

Figure 2: Space of features extracted from our test data with the pretrained network (left) and the fine-tuned network (right) by classes.

	Accuracy
Fine-Tuned CNN	0.860
SVM on Pretrained Features	0.621
SVM on Fine-Tuned Features	0.855

Table 3: Overall accuracies of fine-tuned CNN, the SVM trained on features extracted by pretrained CNN and the SVM trained on features extracted by fine-tuned CNN.

trained with the fine-tuned features. These accuracies are shown in Table 3.

Observing the accuracies, we see that the fine-tuned CNN performs similar to the SVM that was trained with the fine-tuned features (0.86 vs. 0.8547). This result is reasonable since the last layer of a CNN performs linear regression and both linear regression and a linear SVM learn a linear decision boundary. By training the SVM with the features from the CNN's last hidden layer, we essentially replaced one linear classifier with another and consequently cannot expect any substantial differences in performance. When we tailor the interpretation to our current implementation, we notice that the library *liblinear*, which we currently use for the SVM implementation, actually implements "*L2-regularized logistic regression*" with the default parameter setting. This is another reason why the results do not substantially deviate from each other.

Besides comparing the fine-tuned CNN and SVM against each other, we also observe that both perform significantly better than the SVM trained on the pretrained features. What already became apparent while visualizing the feature space in section 4.1 is now also reflected in the quantitative results. During the fine-tuning of the pretrained CNN, the existing feature space adapted successfully to the feature space defined by the *STL-10* data set. However, the feature space defined by the pretrained CNN does not sufficiently reflect the *STL-10* data set to be used for classification. We can therefore conclude that fine-tuning the CNN has significantly improved both the separation of the classes in the feature space and the performance of the models.

4.3 Comparison with the Bag-of-Words model

Finally, we compare our fine-tuned CNN with the BoW model we approached in the last part of this exercise. In Table 4, we see the accuracies of BoW models across different color spaces, SIFT types and vocabulary sizes. We already know from the first part of the final project that the BoW approach

Colorscale	SIFT Type	Vocabulary Size	Classifier Accuracy
Gray	**Regular**	**400**	0.808
		1000	0.802
		4000	0.8
	Dense	**400**	0.869
		1000	0.858
		4000	0.813
RGB	**Regular**	**400**	0.810
		1000	0.803
		4000	0.8
	Dense	**400**	0.874
		1000	0.865
		4000	0.813
Opponent	**Regular**	**400**	0.816
		1000	0.805
		4000	0.8
	Dense	**400**	0.881
		1000	0.8732
		4000	0.82655

Table 4: Accuracies for the different method combinations

yields better performance when using dense SIFT descriptors than when using regular keypoint SIFT descriptors. When comparing the fine-tuned CNN with these BoW models, we first notice that the BoW with regular SIFT descriptors performs considerable worse than the fine-tuned CNN. However, the BoW model using dense SIFT descriptors shows a similar performance and even outperforms the CNN in some cases. This is especially evident for the vocabulary size 400 category in both the RGB and opponent color space (0.874 and 0.881).

However, although the BoW model performs better in some instances, it also carries some disadvantages. First, constructing the vocabulary, building up image histograms and training the BoW model is more time consuming than the CNN setup, both in terms of implementation and run time. Furthermore, the performance of the BoW model is heavily dependent on the choice of the hyperparameters, such as the color space, vocabulary size and SIFT type. In turn, the optimal hyperparameters are also highly dependent on the respective problem and the given data set. Thus, the once optimal hyperparameter setting may radically change with the problem or the data set. In contrast, heuristics for the hyperparameters of a CNN, such as learning rate, batch size, or number of epochs, seem to be more generally applicable and often lie in similar ranges across different problems and data sets. Finally, it appears that improving the CNN, for example by adding an additional hidden layer or changing details in the network architecture, is more accessible, while in our example it is challenging to further improve the BoW performance.

Conclusion

Over the course of the second part of the final project we approached transfer learning with CNNs by fine-tuning a pretrained network for classification with the *STL-10* dataset. After finding satisfactory hyperparameters through empirical exploration and grid-search, we qualitatively explored the feature space of the pretrained and fine-tuned CNN by using t-SNE visualization. Afterwards, we quantitatively evaluated the fine-tuned CNN and compared it with two SVMs trained on features extracted by the pretrained and fine-tuned CNN. Considering the qualitative and quantitative results it is evident that the fine-tuning process was successful in learning a better representation of the feature space. This becomes especially obvious when looking at the accuracy difference of 23.4% between both trained SVMs. Further, we did not see a notable difference between the fine-tuned CNN and the SVM trained on its extracted features. Finally, when comparing the fine-tuned CNN with the BoW approach, we found the BoW model with the dense Sift descriptors to perform better than the fine-tuned CNN. However, we considered the BoW architecture less efficient and found little potential for possitble performance improvements compared to the CNN.

References

[1] Itseez. *The OpenCV Reference Manual*, 2.4.9.0 edition, April 2014.

[2] Laurens Van Der Maaten. Accelerating t-sne using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245, 2014.