Computer Vision 1

Final Project Part 2

Michiel Bonnee	Axel Bremer	Ruben van Erkelens	Tim de Haan
11883936	11023325	12065064	11029668

Introduction

In this part of the project a convolutional neural network is built, that has the task of classifying images. In the previous part of the project we classified the images using handcrafted features. Because we use a convolutional neural network, the handcrafting of the features can be skipped since these are learned in the network as well.

We use the STL-10 dataset, the data consists of 10 classes of images. In this project we only use 5 of these classes. The image classes we use are : 1. airplanes, 2. birds, 3. ships, 4. horses, 5. cars. The pre-trained network however is trained on 10 classes. This network has to be updated to classify for these 5 classes instead of the full 10.

This project also has some incremental tasks we performed. First, we analyzed a pre-trained network architecture. Second, we did some preprocessing of the data we used as input. Third, we updated the network architecture to deal with our new data. And finally, we tweaked the hyperparameters.

3 Tasks

3.1 Understanding the Network Architecture

Network structure

1. The pattern we can see is as follows: First a convolutional layer, second a Relu layer, third a max pooling layer. This pattern is repeated 4 times. The network ends with a single softmax loss layer, which calculates the loss using the softmax function**[softmax function reference] don't know of hier echt een reference voor is**. This pattern is used in most convolutional network architectures.

2. The biggest size and most parameters can be found in the second to last layer. It uses 64 layers of size 64x64 which ends up in 64 x 64 x 64 = 192 parameters.

3.2 Preparing the Input Data

Data structure

All 6500 images are loaded in and formatted as specified in the assignment. They are resized to a size of 32x32 to fit the weights of the first layer of the pre-trained network.

3.3 Updating the Network Architecture

Updating the model

We changed the output layer from a 64 to 10 layer to a 64 to 5 convolution layer to accommodate for the fewer number of classes. The loss layer is still a softmax layer.

3.4 Setting up the Hyperparameters

Hyperparameter optimization

When running a convolution neural network there are multiple hyper parameters to be taken into account. The learning rate, weight decay, the batch size at which to process the data and the number of epochs the network will train.

In this report we only experimented with changing the batch size and the number of epochs. The different batch sizes we checked are 50 and 100. The number of epochs are set to 40, 80 and 120. The results of these hyperparameters are displayed in table 1. We can clearly see that a batch size of 50 yields better results than a batch size of 100. This can be due to the fact that a batch size of 100 makes the gradient generalize over too many points. This causes the model to underfit and perform worse. However when given long enough (120 epochs) we can see that the accuracy almost meets the one with batch size 50 ran for 120 epochs.

Every one of the models fine-tuned for 5 classes does however outperform the pre-trained network. The accuracy of the 10 class network on the 5 class dataset is 69.25. This is to be expected since the pre-trained network is trained to classify 10 different image classes, therefore it has to generalize a lot more than the networks trained for 5 different classes.

The best hyperparameters are a batch size of 50 ran for 120 epochs. The rest of the experiments will be done using the network trained with these parameters.

Fine-tuned accuracies

	40	80	120
50	80.08	77.70	82.45
100	75.75	76.85	81.97

Table 1: Accuracies of the fine-tuned network based on the batch size and epoch length.

4 Experiments

4.1 Feature Space Visualization

The method used for visualizing the feature space of the networks is t-sne by Laurens van der Maaten [1]. This reduces an n-dimensional feature vector to 2 dimensions to visualize them.

In figure 1 we show the second to last layer of the pre-trained and the fine-tuned networks. This is a 64 dimensional feature vector which is used to classify the images into their classes. We observe that for the fine-tuned network the classes are separated way better than with the pre-trained network. This shows us again that the fine-tuned network performs a lot better on our data than the pre-trained one. The data in the fine-tuned figure is more separable than the data in the pre-trained figure.

This visualization also shows us which classes the networks tend to mix up between them. For example, airplanes and ships look alike according to the network. This can be seen in figure 1, planes and ships are clustered together. This could be due to the fact that both planes and ships are both man made metal objects and often have a similar background, the sky and the sea are both greyish blue. That could also be the reason that some birds are in the plane cluster and vice versa. They are both

silhouettes against the sky as background. Birds and horses are both animals and are therefore closer together than to the other classes.

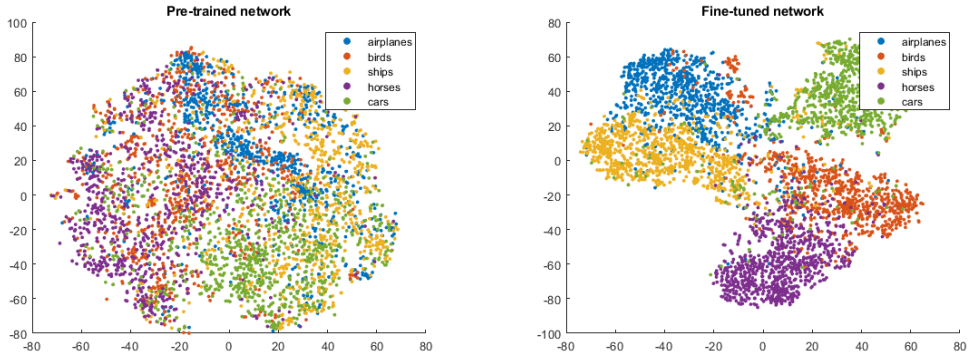

Figure 1: The t-SNE 2D feature map for the networks' 2nd to last layer.

4.2 Evaluating Accuracy

In table 2 the accuracies for all the different hyperparameters can be found. We found that the SVM and the CNN of the fine-tuned features both gave the same accuracy. They are consistently better than the pre-trained network. Which is plausible because of the reasons given in 3.4. Every convolutional neural network we trained was far better than any of the bag of words models we trained in part 1 of this project. The best accuracy we found was for the RGB-sift method using 4000 words. It was an accuracy of 28.23%, which, in a case with 5 classes is only marginally better than random.

The features trained by the CNN turned out to be much more meaningful than the sift features we have found in part 1. We tried to visualize the histogram created by the bag of words model using the t-SNE algorithm. This can be seen in figure 5. It is a random cloud of points, not separable at all. This confirms that these features are less meaningful than the ones found by the CNN.

batch size	number of epochs	CNN: fine-tuned	SVM: pre-trained	SVM: fine-tuned
50	40	0.8	69.25	80.08
50	80	0.77	69.25	77.70
50	120	0.82	69.25	82.45
100	40	0.76	69.25	75.75
100	80	0.77	69.25	76.85
100	120	0.81	69.25	81.97

Table 2: Accuracies of the fine-tuned network based on the batch size and epoch length.

4.3 Bonus: Freezing Early Layers

An experiment we ran was to not update the layers in the new network except for the final layers. This way we save the features found by the pre-trained network and only try to classify using these features. It was however in vain, both the networks had an accuracy of 68.88, which is even less than the normal pre-trained network. In figure 2 the learning process of these networks can be observed. From this we can infer that it is not a smooth loss function. It takes a long time to find the weights that come close to minimizing the loss function. A batch size of 100 elongates this time even more due to the generalization that comes from the bigger batch size.

Conclusion

In this project we used a pre-trained convolutional neural network, trained to find features images of 10 classes, and fine-tuned it for 5 classes. We tried different hyper parameters and found that the best ones are a batch size of 50 and to run it for 120 epochs. The fine-tuning improved the model by 13%. It also vastly outperformed our previously trained bag of words model. It did so by more than 50%, reaching a high of 82.45%. From this we can conclude that the features found by CNN's are often more meaningful than 'handmade' features.

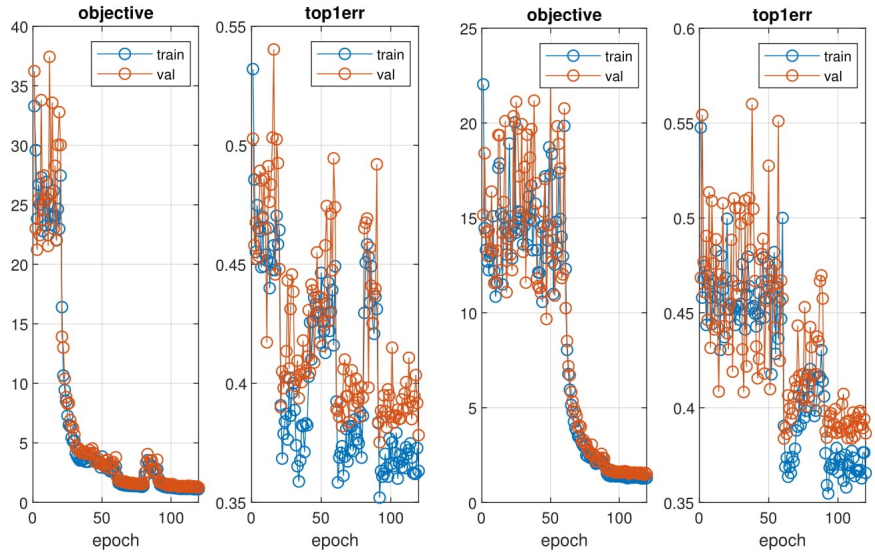

Figure 2: Error plots for the frozen layer networks with batch size 50 and 100 respectively.

References

[1] Laurens van der Maaten. Software | t-sne. https://lvdmaaten.github.io/software/#t-sne. Accessed: 2019.

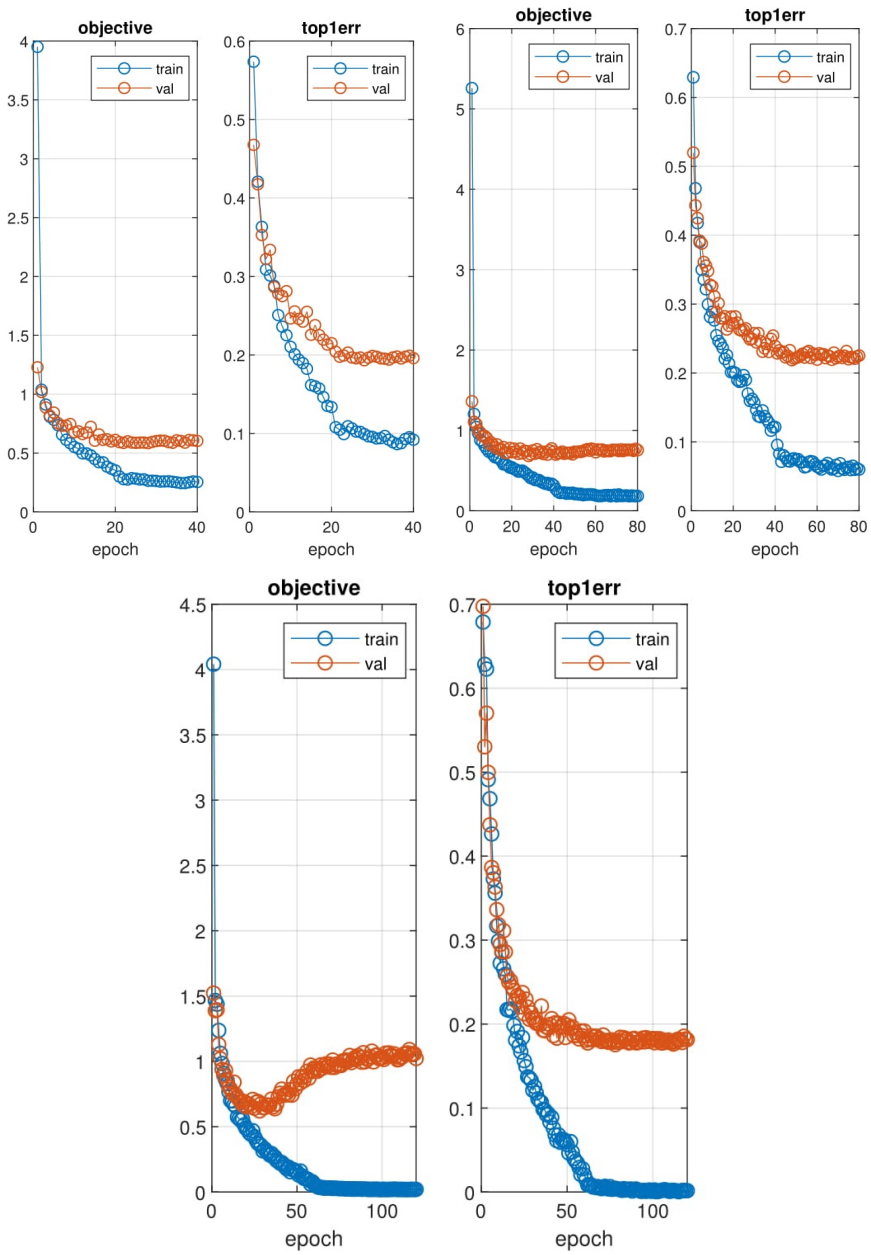

Figure 3: Error plots for batch size 50 trained for 40, 80 and 120 epochs respectively.

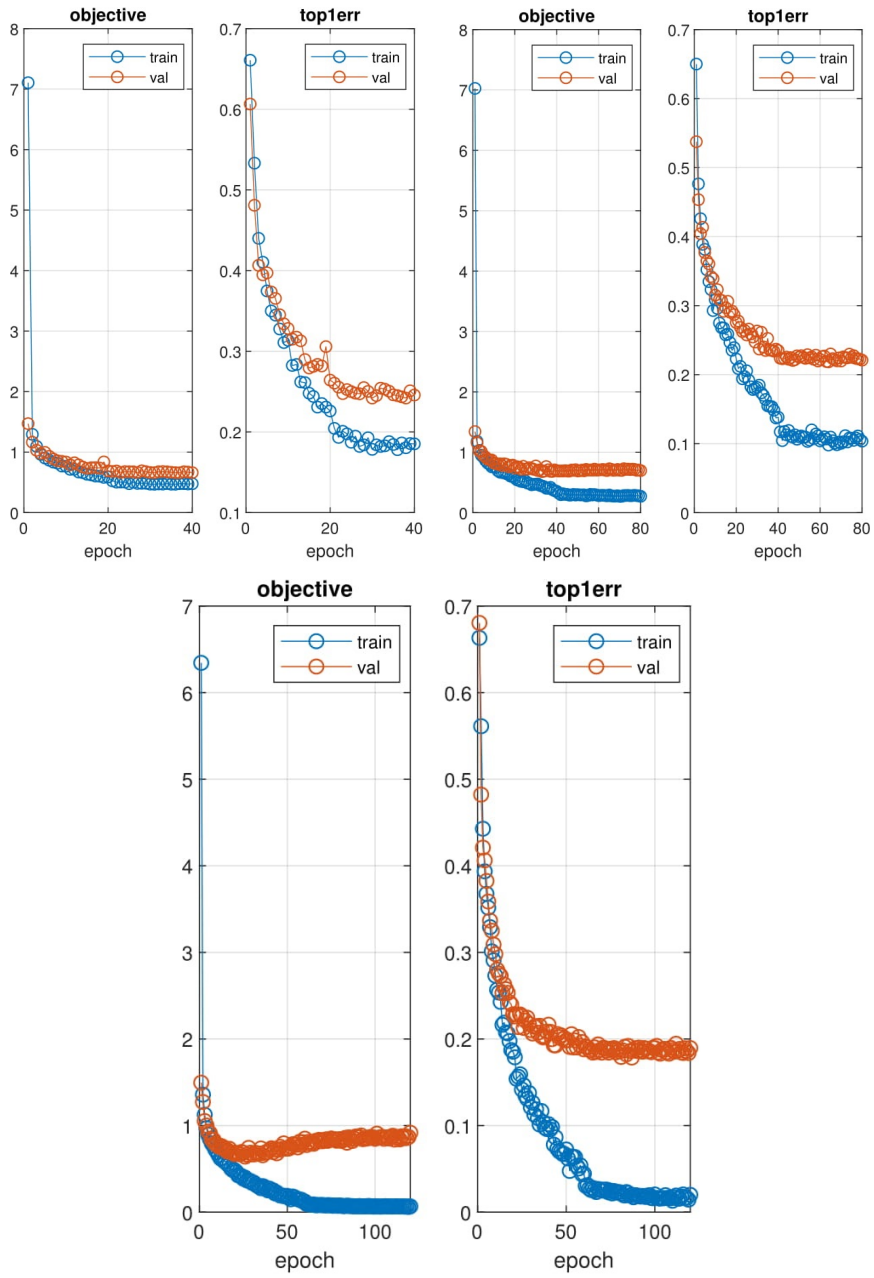

Figure 4: Error plots for batch size 100 trained for 40, 80 and 120 epochs respectively.

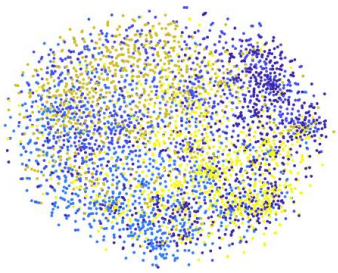

Figure 5: t-SNE visualization of the bag of words model histogram.