Computer Vision 1
CNNs for Image Classification
Final Project Part 2

Gongze Cao, Kai Liang, Xiaoxiao Wen and Zhenyu Gao
Faculty of Science, University of Amsterdam

1 Introduction

In the second part of the final project, we mainly focus on using Convolution Neural Network (CNN) for the same five-class image classification task as in the Part 1. We begin with the provided pre-trained CNN which is trained on the CIFAR-10 dataset. As this dataset contains images of different classes and figure sizes than the dataset we are using (i.e. a subset of STL-10), we first pre-process our data to match the input of the pre-trained network and update the network architecture to classify five classes. Then, we experiment with several hyperparameters (e.g. batch size, epoch number and learning rate) to find the best fine-tuned model. After this, we visualize the feature spaces of the fine-tuned model and the pre-trained model to compare their performance. At the end, we evaluate on the accuracies of different classification strategies with different features (i.e. pre-trained SVM, fine-tuned SVM, fine-tuned CNN and SIFT+BOW SVM).

Throughout the assignment, we distribute our work fairly in terms of coding and writing the report.

2 Network structure

1. We can see patterns from the architecture of the network that it consists of the repeated combinations of the following layers:
   (a) Convolution layer for image filtering.
   (b) (Max- or Average-) Pooling layer for feature sub-sampling.
   (c) ReLU layer for applying non-linear activation.

   Besides, the network contains a input layer for processing and transforming input images and a output softmax layer for the classification task.

2. It can be inferred from the statistics of the architecture that layer 1 has the biggest size in the entire network (i.e. of size $32 \times 32 \times 32$), and that layer 10 has the most parameters (i.e. using $256KB$ for parameter memory), which can also be manually verified by computing for each convolution layer the parameters needed and seeing that layer 10 uses $4 \times 4 \times 64 \times 64 = 65536$ parameters in total.

3 Data structure

For convenience, the input passed to the pre-trained network is of a customized data structure containing the input data and necessary meta information about the dataset. This data structure is denoted as **imdb** under which there are two main parts, **images** and **meta**:

1. **images** includes the raw images **data** as a 4D tensor of shape (*im_height*, *im_width*, *num_channel*, *num_images*). The **labels** under **images** is an 1D tensor representing the labels of images, and **set** is an 1D tensor indicating whether the corresponding images are from the training set or the test set.

2. **meta** includes all meta information about the data set, which has two members **sets** and **classes**. **sets** contains the split names, and **classes** contains the names of the five classes of the dataset (i.e. airplanes, birds, ships, horses and cars).

Based on the data structure, we implement the `getIMDB` function which reads figures along with their information, and transforms them into the **imdb** structure.

4 Updating the model

For NEW_INPUT_SIZE, it should correspond to the output size of the previous layer (i.e. block 4), and for NEW_OUTPUT_SIZE, it needs match the possible number of classes of our classification task. Thus, we set the two parameters: NEW_INPUT_SIZE and NEW_OUTPUT_SIZE, to namely 64 and 5.

5 Hyperparameter optimization

To achieve better results, we experiment with different hyperparameters, which include:

1. Learning rate for previous layers (**lr_prev_layers**).
2. Learning rate for updated layers (**lr_new_layers**).
3. Weight decay (**weightDecay**).
4. Batch size (**batchSize**).
5. Number of epochs (**numEpochs**).

For **lr_prev_layers** and **lr_new_layers**, their original values are too large which always lead to unwanted overflow in the objective function, so we empirically decrease them to $[0.005, 0.03]$ and $[0.05, 1.0]$ respectively. Additionally, the default value for **weightDecay** is inappropriate which leads to over-fitting of training data easily, thus, we increase it to 0.001 for a greater decay. In terms of **batchSize** and **numEpochs**, we use grid search to find their optimal values. We set **batchSize** to $\{50, 100\}$ and **numEpochs** to $\{40, 80, 120\}$, and the results are shown from Figure 1a to Figure 1f.

As can be inferred from Figure 1, on the one hand, when keeping the batch size constant, increasing the number of epochs will generally *reduce* both the top 1 training error and the validation error after convergence. On the other hand, when keeping the number of epochs constant, increasing the batch size will *increase* both the top 1 training error and the validation error after convergence. Thus, the best hyperparameter settings for **batchSize** and **numEpochs** are namely 50 and 120.

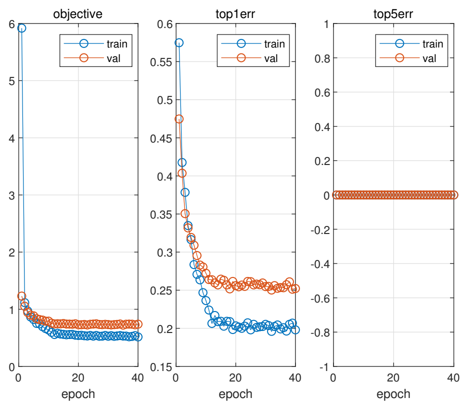
6 Visualization

To compare the difference of the pre-trained model with the fine-tuned model, we apply **t-sne** as a dimensionality reduction method to reduce the dimension of the feature space of the two models to 2 and visualize them using features extracted from their second to last convolution layer (i.e. layer 10) each. As shown in Figure 2, we can tell that the feature space of the pre-trained network is more random and blended compared to the feature space of the fine-tuned network, and that the feature space of the fine-tuned model is more discriminative which therefore means that the fine-tuned network performs better in feature detection.

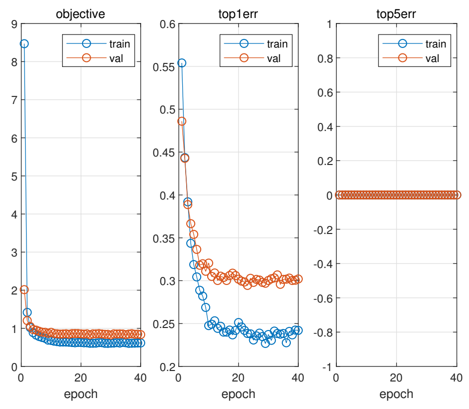
7 Accuracy

After extracting features from both the pre-trained and fine-tuned network, we then use these features separately for evaluating the accuracies of different classification strategies. The results are as follows:

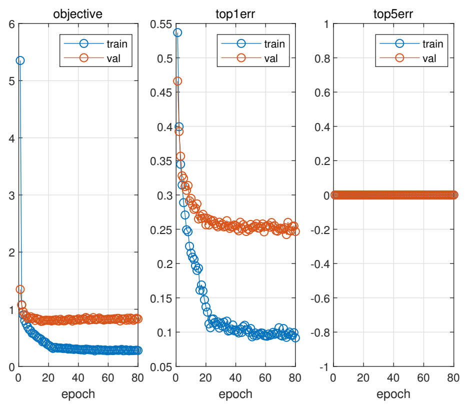
1. The classification accuracy of SVM using **pre-trained features** is: 69.33%.
2. The classification accuracy of SVM using **fine-tuned features** is: 77.42%.
3. The classification accuracy of **fine-tuned CNN** is: 76%.



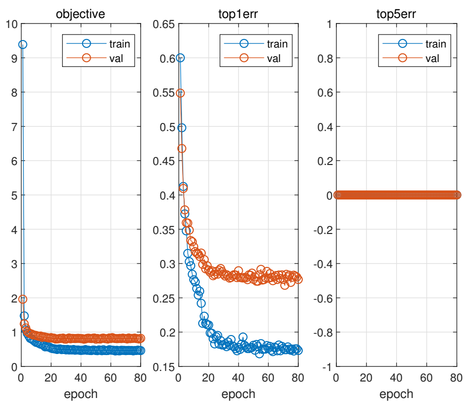
(a) ***batchSize*** $= 50$ & ***numEpochs*** $= 40$.



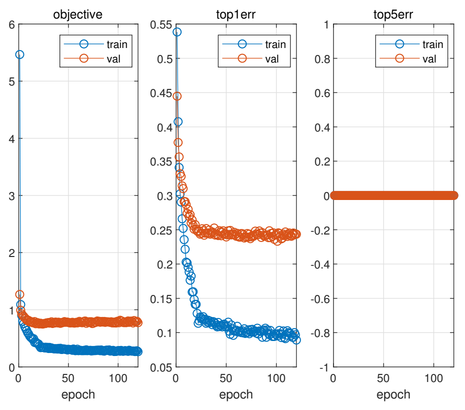
(b) ***batchSize*** $= 100$ & ***numEpochs*** $= 40$.



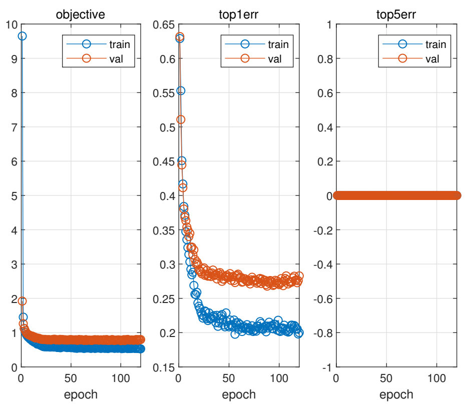
(c) ***batchSize*** $= 50$ & ***numEpochs*** $= 80$.



(d) ***batchSize*** $= 100$ & ***numEpochs*** $= 80$.



(e) ***batchSize*** $= 50$ & ***numEpochs*** $= 120$.



(f) ***batchSize*** $= 100$ & ***numEpochs*** $= 120$.

Figure 1: *Training results of applying different values for* ***batchSize*** *and* ***numEpochs****.*

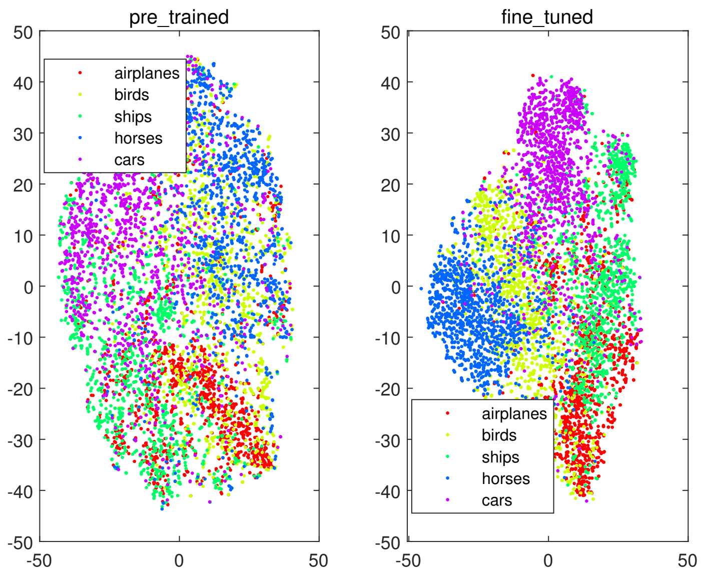


Figure 2: *Visualization of the feature space of the pre-trained network and the fine-tuned network.*

We can see that the accuracy of SVM using fine-tuned features and the accuracy of fine-tuned CNN are similar and are significantly higher compared to using SVM with pre-trained features. In section 6 we know that the feature space of pre-trained network is more blended than the feature space of fine-tuned network, which explains the reason of getting better performances for fine-tuned CNN and SVM with fine-tuned features. On the other hand, as the performance does not differ much between fine-tuned CNN and SVM with fine-tuned features, the indication is that for the image classification task, using discriminative features are more important than choosing among different classification methods.

Comparing to the results we get from using the BOW method in Part 1:

1. The mean classification accuracy of SVM using **key-point SIFT extractor** is: 53.9%.
2. The mean classification accuracy of SVM using using **dense-sampling SIFT extractor** is: 63.2%.

We can infer that using features extracted from a CNN, especially when they are pre-trained on a large dataset and fine-tuned on the down-stream task, is significantly better than the traditional SIFT+BOW method for image classification, which again reflects the importance of features for our task.

8 Conclusion

In the previous sections, we implement and discuss the five-class image classification task using transfer learning in CNN. We first pre-process our data to match the input of the pre-trained network. Then, we adjust some network architecture in order to meet the requirement of our task, and fine-tune some hyperparameters to boost the performance of the network. In the experimental stage, we first visualize the feature space of the pre-trained and fine-tuned models, and then provide a qualitative comparison on several classification strategies including SVM and CNN with different features (i.e. pre-trained on CNN, fine-tuned on CNN and trained on BOW with SIFT descriptors). We conclude that the classification accuracies of the fine-tuned CNN and SVM using fine-tuned features are similar and better than other settings.