Computer Vision - Final Part Two

Ioannis Gatopoulos 12141666, Philipp Ollendorff 11734078,
Konstantin Todorov 12402559, Vincent Roest 10904816

August 31, 2019

Introduction

The second approach to classifying images, which has recently gained a lot of popularity, is using Convolutional Neural Networks (CNN). This report will firstly highlight the inner workings of the CNN used for the present classification task. It will subsequently present a quest to the parameter settings that best perform on the current task. Furthermore, training details are visualized through the development of the CNN's cost function and errors. The result of the learned features are visualized through t-SNE, after which a comparison is offered between the BoW model and the CNN approach.

1 Understanding the Network Architecture

a) The Figure 1 illustrates an excerpt of the MATLAB visualization of the pre-trained neural network (pNN). As we can see, the triplet of Convolution Layer, Pooling Layers and Activation functions (ReLU) is used repeatedly, and specifically, three times in total. The order of this process pattern is heavily dependant on the type of pooling that takes place. When average pooling is used, the process follows the above order (Conv - Pool - ReLU) and is spotted taking place in the middle of this pNN twice. However, in the beginning max pooling is preferred and the order between the pooling layer and the activation function is switched. At the end, in order to apply the softmax function for classification purposes, a fully connected layer is used.

From a parameter dimension perspective, it is also clear that a pattern is used. Although as the depth of the data initially grows ten fold and then double in size with the activation of convolution layers, every pooling layer halves the data size. This pattern is preferable. As we use more and more kernels to discover more complex patterns within existing patterns, the number of parameters grows quadratic. To deal with this, we reduce the dimensions of the data, keeping only the most important features.

b) We are going to look at it layer by layer. First of all, the `input` layer as well as the `mpool`, `apool` and the `relu` have nothing to learn. Therefore there are no learnable parameters in these layers. Thus, the number of parameters is equal to 0. This is also visible on the last row of Figure 1, as we see that the parameter memory (`param mem`) on these layers is `0B`.

The only layers that carry learnable parameters are the convolution ones (`conv`). The number of parameters that they have, can be calculated with the formula

$$P_c = (K^2 \times C + 1)N$$

where K is the size (width) of symmetric kernels used in the Conv Layer, N is the number of kernels and C the number of channels of the input image. For example, the first `conv layer1`, has `(5*5*3+1)*32 = 2432` parameters (including biases). Like before, from the Figure 1, it is clear that the (`conv layer10`) is the layer with the most parameters. This is justified by the fact that until that point, we increase the depth (thus the parameters) and then, at `conv layer12`, we decrease the output channels to 10, in order to apply the softmax function for classification.

Finally, we can spot that `conv layer1` is the one with the biggest size, as it has the greatest `data mem` value. This is expected, as the features of the first convolution layer have the biggest spatial dimensions. After this point, they are reduced by the pooling layers.

```
     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 1: The architecture of the pre-trained neural network

2 Hyperparameter optimization

Choosing appropriate hyperparameters plays a crucial role in the success of a neural network architecture, as it makes a huge impact on the learned model. In this part, we experiment with different parameter settings and use the grid search method to pick up the best setting.

Our training hyperparameters are

```
learningRate = [ 0.0001*ones(1,20) ...  0.0002*ones(1,20)...  0.0005*ones(1,10)...].
```

Especially, we tuned the parameters of batch size as well as the number of epochs in the space {`50, 100`} and {`40, 80, 120`} respectively. This results in a total of six runs. The results can be found in Table 1, where the validation error is reported. As we can see, the best performance is achieved with the values of `50` for batch size and `120` for epochs. The performance of this particular set-up on the training and validation set is visible on Figure 2. We can quite clearly see from the cost function that as the number of epochs increases, there is a growing discrepancy between the validation and training cost. This indicates a tendency of this network to overfit on the training data. Therefore, it might be wise to cap the number of iterations to prevent the validation error from increasing after a certain number of epochs (at least not increase the number of epochs beyond 120). This is in line with the expectations that one has with the size and complexity of the current dataset.

Batch Size	Epochs	FT-CNN
50	40	0.68
50	80	0.75
50	120	**0.76**
100	40	0.67
100	80	0.73
100	120	0.74

Table 1: Convolution Neural Network error on validation set using various hyperparameters

3 Structures

3.1 Input Data

The Input Data was already given in the imdb files and only needed to be transformed for our purposes. We extract individual images from the .mat files, resize them to 32 x 32 x 3 and adjust the classes to our needed

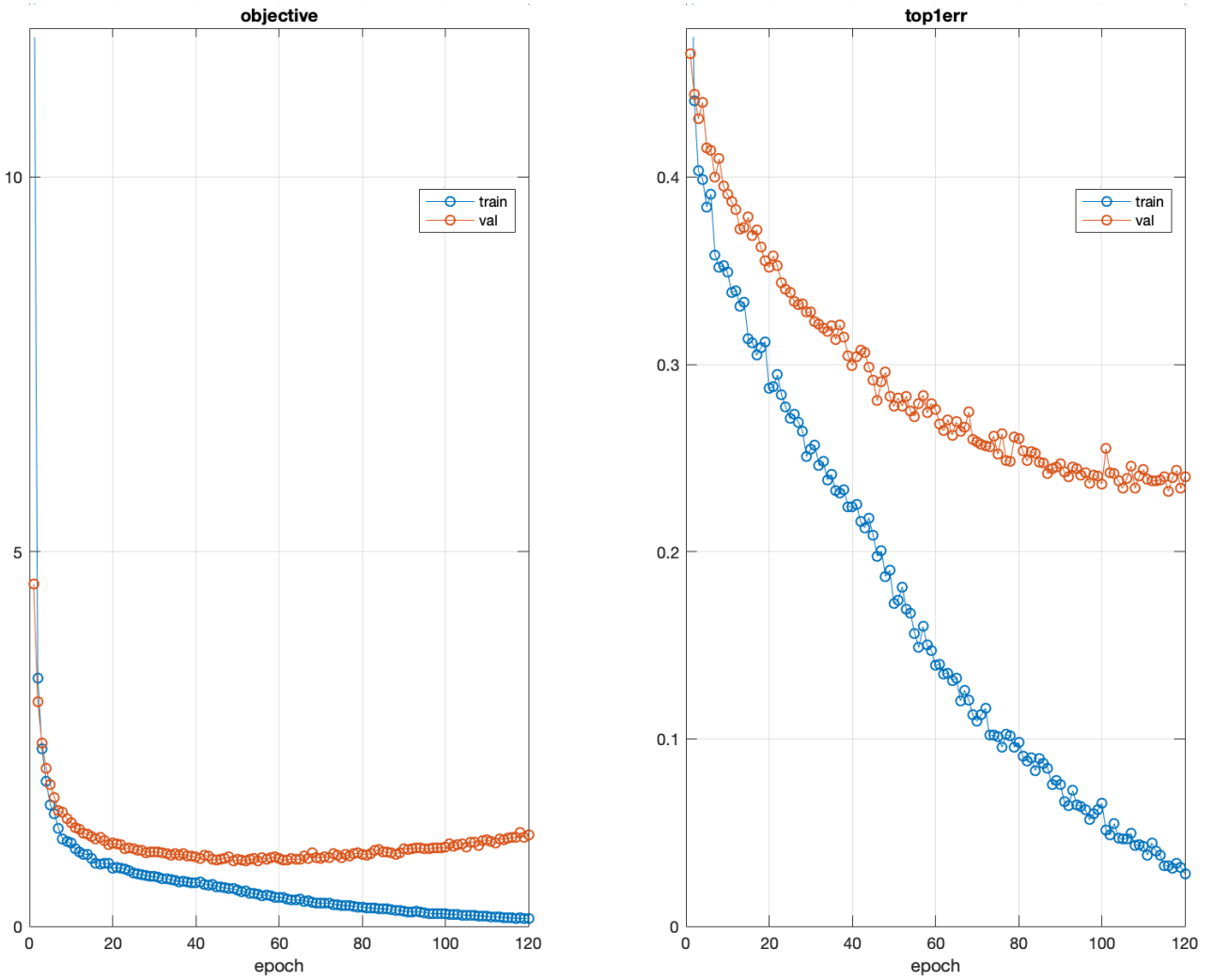


Figure 2: Plots of the objection function and the error on the train and validation set using the hyperparameters Batch Size = 50 and epochs = 120.

order: airplanes(1), birds(2), ships(3), horses(4), cars(5). We further split the data into training and test sets.

3.2 Network

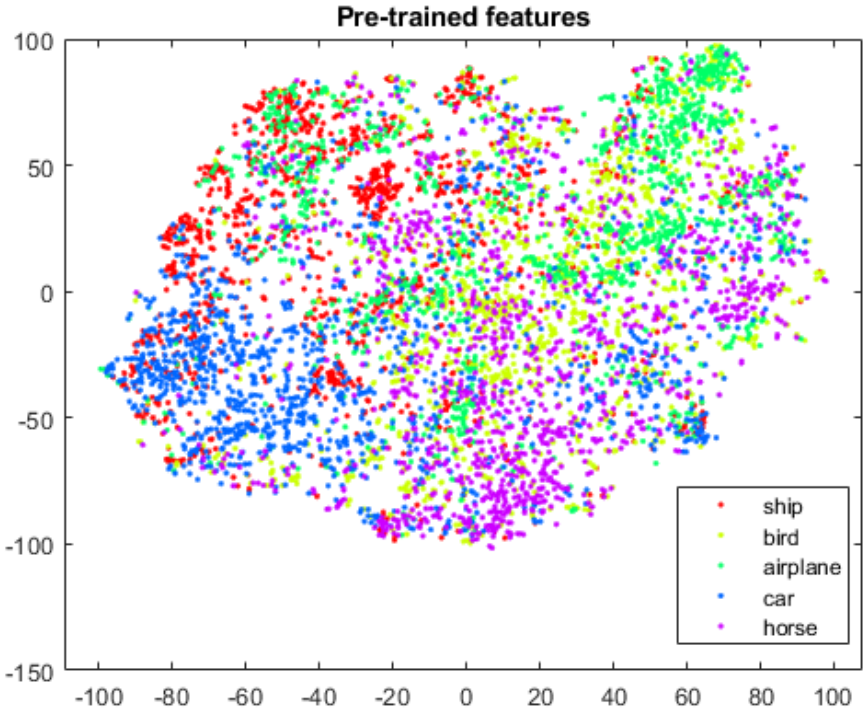
The given pre-trained network classifies 10 different classes and therefore needed to be adjusted to our use-case. To do this, we adjusted only the final output layer to size **64 x 5**. The input dimension depends on the output dimension of the previous layer, while the final output dimension is equal to the number of predicted classes.

4 Visualization

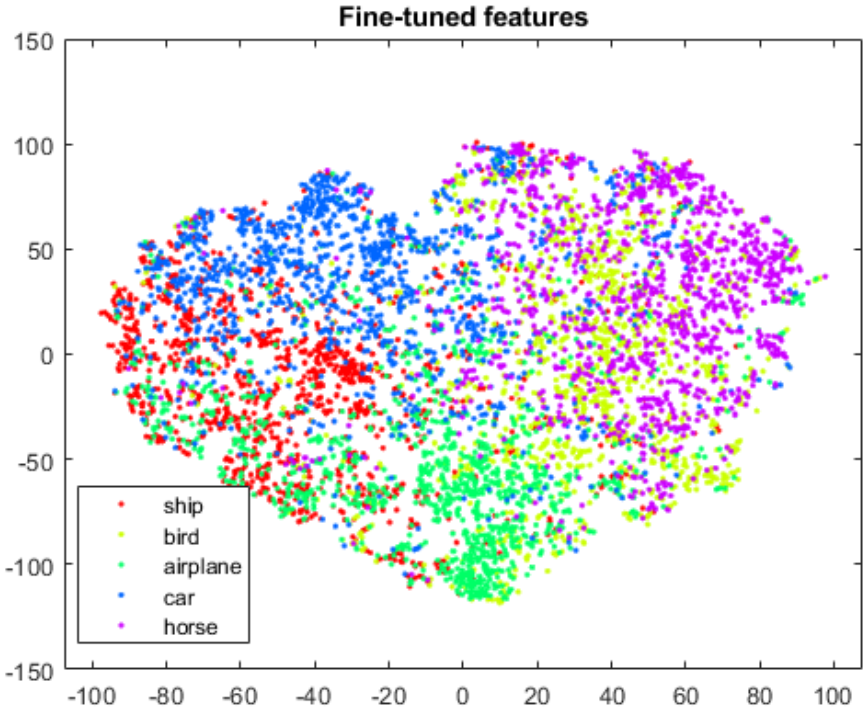
After using the t-SNE library[Laurens van der Maaten, 2008] for our designed network with the best found hyper-parameters, on Figure 3 we can see the result - the features for the pre-trained and fine-tuned network are displayed in the 2D space. There are clearly visible differences between the two, which correspond to the theory - using a network which was not trained for this task will result in invalid predictions - such that they are all over the feature space and have no grouping of some sort (Figure 3a). After fine-tuning our network this changes and the features are now grouped more distinctively into five different clusters where also we can see some overlaps for some of the classes (Figure 3b). The *bird* and *airplane* classes have some similar representations, where *bird* and *horse* are very similar in the final outcomes and overlap on the features plot.

Still, we have to acknowledge the fact that there are some similarities between the two plots as well. For the classes *airplane*, *horse* and *car* - most feature points are still mostly concentrated in one area of the 2D plot

even before fine-tuning. Though in the case of pre-trained network, the *horse* class for example also contains features from the entire plot. The similarity in the feature representation could be explained as the CIFAR-10 [Krizhevsky, 2009] dataset that was used to train the network has some classes which are the same as those used in the STL-10 [Adam Coates, 2011] dataset that we are using to test (and so fine-tune) our network.



(a) Pre-trained features



(b) Fine-tuned features

Figure 3: Visualized features using t-SNE

SVM pre-trained	SVM fine-trained	fine-tuned CNN	Bag of Words
61.92%	70.47%	65.00%	67.45%

Table 2: Classification accuracy of various models

5 Evaluation Accuracy

Lastly, we measured the accuracy of three different configurations. Our results can be found in Table 2. The SVM fine-trained model was expected to perform at least as good as the fine-tuned CNN model, as it only adds an additional layer on top of it. Naturally the SVM fine-trained should outperform the SVM pre-trained model, as it more closely resembles the needs for this particular classification task.

The table also includes a metric for the Bag-of-Words model. As we previously only determined the mean average precision values for BoW we adapted the code from the first part to also account for accuracy. When comparing these results we clearly see how close all these models are to each other. SVM fine-trained is still the highest performing, but the difference is almost negligible. When the CNN is tuned further we expect it to outperform the BoW approach in accuracy. Still, it will be interesting to compare different metrics as well to confirm a definite superiority of CNN over BoW. As such, we propose further experiments on tuned CNNs as future work, this time reporting precision, recall, accuracy and potentially other metrics.

6 Conclusion

As concluded before, the classifiers presented in this report proved to be more accurate than the BoW model. This is completely in tandem with recent developments, where CNN's are taking control over Computer Vision tasks, both in the image and video spectrum. Their flexibility to learn from the data on the fly and subsequent resistance to variability (lighting, color, position) plays a central role in the efficacy of this type of model, especially when compared to the rather static vocabulary construction methods that are employed in the BoW model. As with a lot of CNN tasks, the classification accuracy could be boosted by introducing a deeper network [Szegedy et al., 2015]. However, if the dataset in this report would not be augmented - either by rotation, scaling, flipping or any other variant introduction - we hypothesize that the present task will be even more prone to overfitting than it was with our simple CNN. These two methods will have to be explored simultaneously to effectively increase the network size. Other methods to boost the accuracy include varying the cost function optimizer (i.e. Adam, Adagrad or RMSProp) and learning rate decay methods during the learning phase. Both can be introduced to the present architecture in no time, which would allow for easy experiments without having to alter the network structure.

References

[Adam Coates, 2011] Adam Coates, Honglak Lee, A. Y. N. (2011). An analysis of single-layer networks in unsupervised feature learning. AISTATS.

[Krizhevsky, 2009] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

[Laurens van der Maaten, 2008] Laurens van der Maaten, G. H. (2008). Visualizing data using t-sne.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.