DLE Homework 3
Fine-Tuning and Transfer Learning

Vojtěch Michal, michavo3

Fig. 1: Samoyed image used to feed the pre-trained SqueezeNet model.

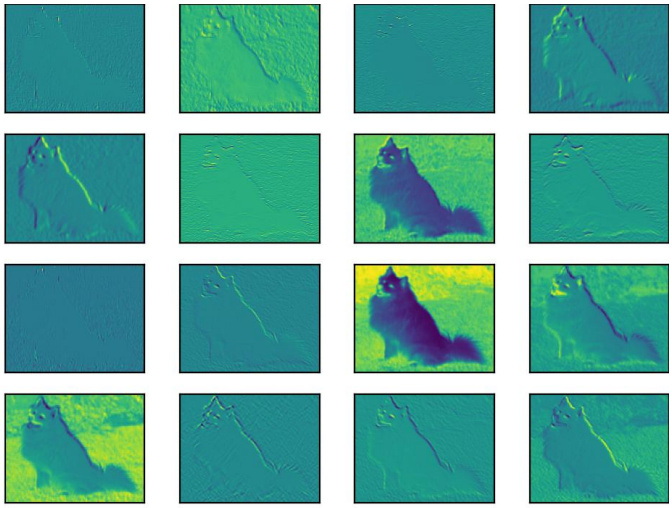


Fig. 2: First 16 output channels of the linear convolution layer

***Abstract*—The assigned[1] homework focuses on transfer learning, i.e. reusing a model previously trained for a different classification task. A pre-trained model for classifying images from the ImageNet dataset is adjusted to handle a different task of classifying cartoons. Several means of model adjustment are investigated – re-training the full model from scratch, using fixed pre-trained coefficients everywhere except the last layer and performing further training on top of pre-trained model coefficients. Results from each method are analyzed and compared.**

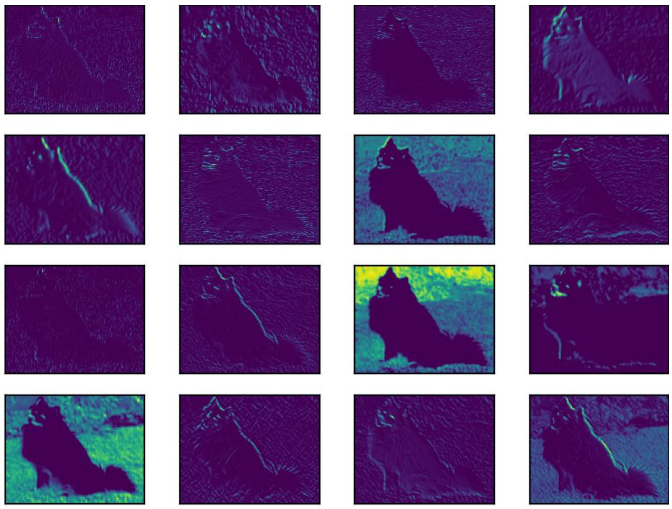


Fig. 3: First 16 output channels of the linear convolution layer after application of nonlinearity

I. Visualization of first layer filters and features

A pre-trained model *SqueezeNet1.0* was fed with the Samoyed image shown in Fig. 1. The five most probable classes according to the model are listed in Tab. I. Since the dog breed Samoyed is among the classes present in the ImageNet dataset, the image is classified with very high certainty. The first convolution layer of SqueezeNet consists of 96 channels, kernels of which are visualized in Fig. 4. Outputs of some of the channels before the subsequent nonlinear activation are shown in Fig 2, while Fig. 3 shows values after the activation function.

Class	Probability
Samoyed	95.04 %
Malamute	0.84 %
Pomeranian	0.74 %
Collie	0.56 %
West Highland white terrier	0.46 %

TABLE I: Five most probable classifications of the image shown in Fig. 1

[1]The homework assignment is available on https://cw.fel.cvut.cz/wiki/courses/bev033dle/labs/lab2_finetune/start

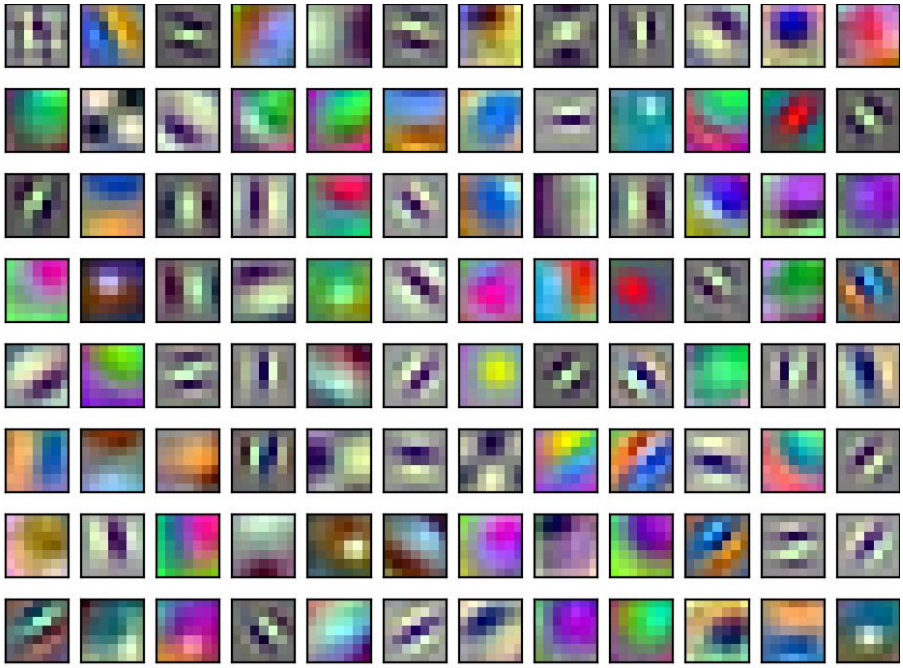


Fig. 4: Weights of the first convolutional layer as images

II. Experimental procedure overview

Before an image can be given to the neural network for processing, it may pass through a preprocessing transformation. Transformations include rescaling, rotating, transformations of color channels etc. One of color channel transformations is normalization using the mean and standard deviation of pixel values corresponding to individual color channels. These calculated values are used to properly normalise the training, validation and testing datasets. Both mean μ and standard deviation σ were calculated using a batch approach[2] in a single iteration over the training dataset. Assuming n samples have already been processed yielding μ_n and σ_n and then we process m new samples yielding μ_m and σ_m, combined statistics μ and σ for the set of $n+m$ samples are

$$\begin{aligned}\mu &= \frac{m}{m+n}\mu_m + \frac{n}{m+n}\mu_n \\ \sigma^2 &= \frac{m}{m+n}\sigma_m^2 + \frac{n}{m+n}\sigma_n^2 + \frac{mn}{(m+n)^2}(\mu_m - \mu_n -)^2.\end{aligned} \tag{1}$$

Iteratively evaluting (1) over the whole training set yields values

$$\begin{aligned}\mu &= \begin{bmatrix}0.8116 & 0.7858 & 0.7380\end{bmatrix}, \\ \sigma &= \begin{bmatrix}0.2730 & 0.2863 & 0.3374\end{bmatrix}.\end{aligned} \tag{2}$$

As expected, these values differ significantly from the statistics of the ImageNet dataset with μ of each color channel in the $[0.4, 0.5]$ range. The difference in means can be explained quite intuitively – cartoons tend to use white background and vibrant bright colors with higher numeric values. On the other hand, the difference in standard deviations is not as simple to explain as it would be very bold to claim anything about the "richness" or "variability" of photos versus cartoons. My intuition relies on the properties of uniform distribution; if we assumed that all pixel values are uniformly distributed in the range $[0, 1]$, then the standard deviation would be $\sigma \approx 0.288$. This value approximately matches the calculated standard deviation σ from the image dataset.

The following sections of this report analyze the performance of *ResNet-18*-based models during various experiments. Tested models use the structure of ResNet-18 with the last layer (the "classifier") replaced with a seven-output linear layer appropriate for classifying seven categories of images. In each experiment, a different part of the model is trained, or a different dataset is utilized. In each experiment, the model is trained for 50 epochs and the training loss, validation loss and validation accuracy are visualized. The training and validation datasets are split using the Pareto principle in the 80 % - 20 % ratio. In each experiment, learning and evaluation are repeated for several values of the learning rate hyperparameter $l_r \in \{0.03, 0.01, 0.003, 0.001, 0.0001\}$. The hyperparameter combination (learning rate and stopping epoch) that maximizes the validation accuracy is recorded during each experiment and the resulting model is tested using a previously unseen test set. Finally, some examples of misclassified images are shown in Fig. 28 together with the correct label and model predictions (classes and the corresponding confidences).

A. Selection of the optimizer algorithm

All experiments were run using both the Stochastic Gradient Descent (SGD) and Adam optimization algorithms. Some issues with SGD were observed during the early stages of implementation. Due to a programming error, the model structure was inappropriate for the classification task, as the final

[2]This approach not hard to derive using the definition of sample variance and sample mean. My implementation was based on derivation in https://notmatthancock.github.io/2017/03/23/simple-batch-stat-updates.html

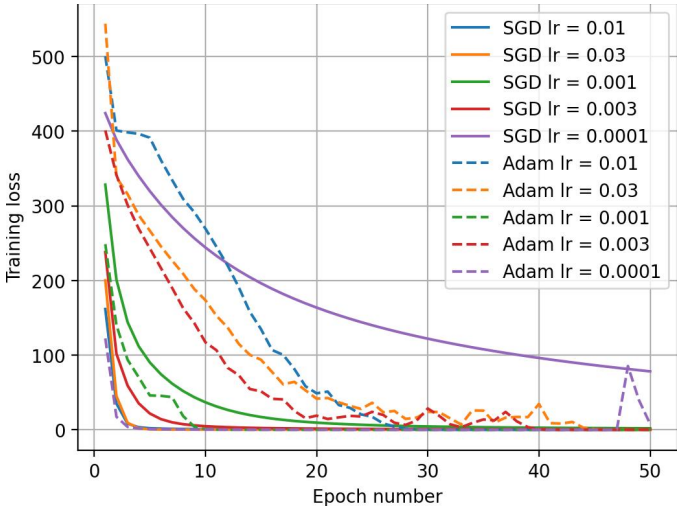


Fig. 5: Training losses for Adam vs. SGD optimizers.

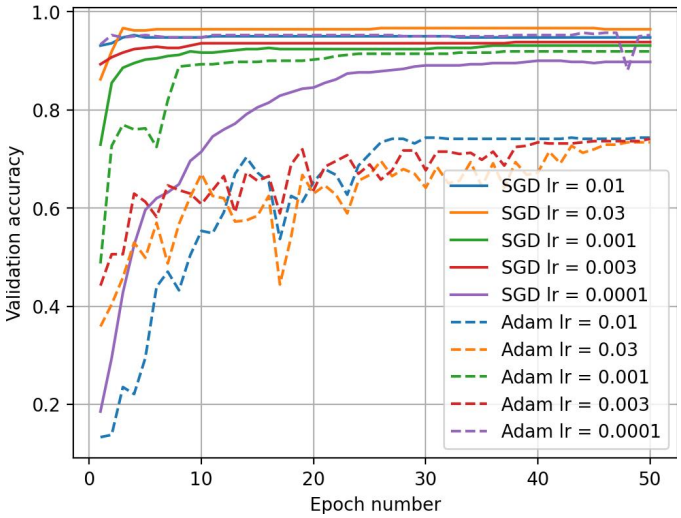


Fig. 6: Validation accuracies for Adam vs. SGD optimizers.

classifier layer was constructed with more outputs than classes in the training set. In that case, the choice of SGD led to an occasional explosion of gradients and a collapse of the learning process through the emergence of NaNs. Higher learning rates ($l_r = 0.03$ and $l_r = 0.01$) were especially susceptible. When such a degenerate case occurred, the model could not improve over time, leading to low validation accuracy constant over the rest of the learning process. In these cases, the Adam optimization algorithm presented a safer and more stable alternative. Although the model structure was incorrect (with a thousand output classes while there were only 7 classes in the training data), learning with Adam did not fail and converged to not-entirely-bad results (test accuracies in the range $[20, 30]$ %).

When the implementation error was corrected, and SGD started to converge every time, it yielded results comparable to those of Adam (as compared in Fig. 5 and 6 for the full model fine-tuning experiment with larger training set described in Section V). Since all experiments were performed for both optimization methods, all subsequent sections present comparison of SGD and Adam in Tables II through VII. No statistically significant pattern was observed regarding whether SGD or Adam yields better results for a particular experiment.

There was, however, a noticeable difference in the time taken by training: the SGD took on average $5.57\,\mathrm{s}$ per epoch while fine-tuning the model (see Section V), whereas Adam required $7.49\,\mathrm{s}$ on average – a 34 % slow-down.

In general, for a fixed SGD optimization algorithm, training from scratch (Sections III and VI) and fine tuning the whole model (Sections V and VI-C) took roughly the same time (typically around $3\,\mathrm{ms}$ per training example). When only the last layer was trained, the required time normalized by the training set size decreased by approximately 30 %. Should the training time be an important factor for the target application, using a pretrained model and fine tuning only the last classifier layer is the best was to deploy a reasonably performing model quickly. Nevertheless whenever enough time is available for training, the roughly 10 % increase in training accuracy is very well worth the investment.

III. Training from scratch

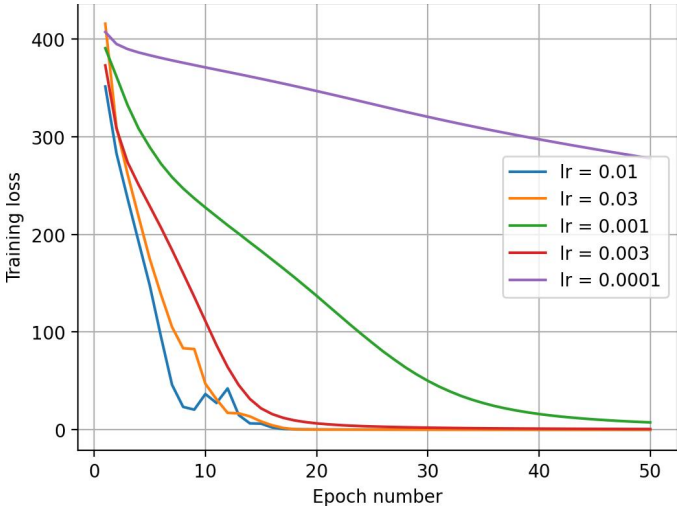


Fig. 7: Part 3 - Training loss for various learning rates

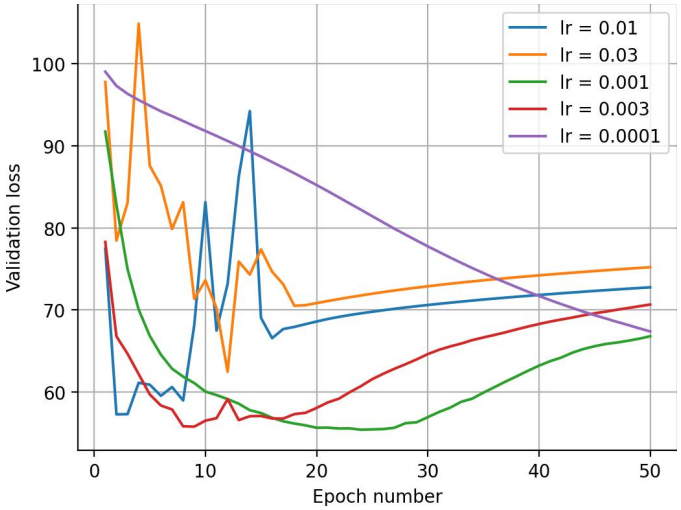


Fig. 8: Part 3 - Validation loss for various learning rates

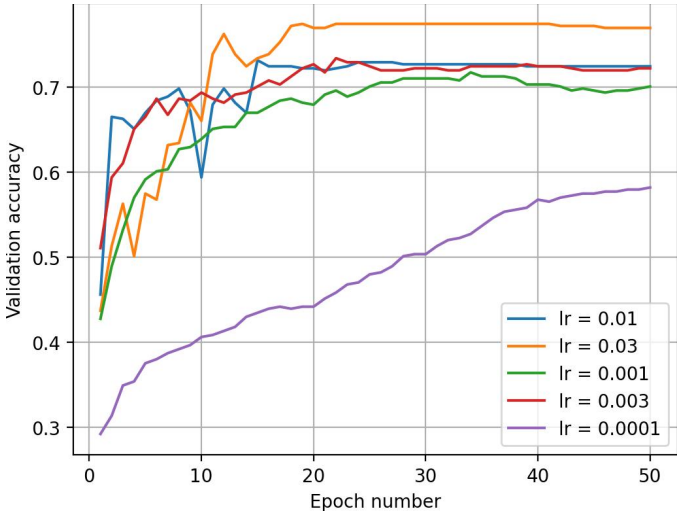


Fig. 9: Part 3 - Validation accuracy for various learning rates

Optimizer	Learning rate	Epoch	Val. accuracy	Test accuracy
SGD	0.03	18	77.43 %	60.60 %
Adam	0.001	47	81.24 %	64.41 %

TABLE II: Part 3 - Best achieved performance

In this part, the ResNet-18-based model was trained from scratch by initializing the model of the given structure with random values of all parameters. Since the training dataset contains significantly less images than the ImageNet dataset (containing more than a million images), imperfect prediction performance is expected.

Since the training set is far too small to train the ResNet-18 properly, a lower model accuracy is expected. Figures 7 through 9 show the evolution of training loss, validation loss and validation accuracy during the experiment. Table II contains information about the most successful model and hyperparameter set. As the training set is small, it is easy for the large complex model to overfit the training set (note the close-to-zero training loss in Fig. 7) without achieving very high test accuracy – i.e. generalizing well to general features of individual cartoon classes.

When all parameters are initialized randomly, the model initially lacks any complex behaviour (such as the ability to detect specific visual features). One could argue that roughly 1600 training samples (80 / 20 % training/validation split) are not enough for the model to build relationships required for complex generalizations to emerge. Instead of directly assigning visual features to individual classes, the model first has to learn to detect such features and ”wastes” computational resources doing that. An improvement is expected to occur in the following section, when the model is allowed to use pre-trained relationships.

IV. Fine-Tuning the Classifier

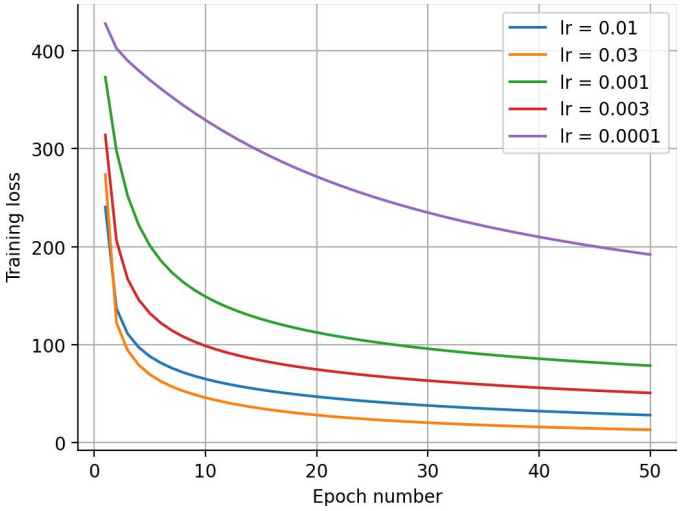


Fig. 10: Part 4 - Training loss for various learning rates

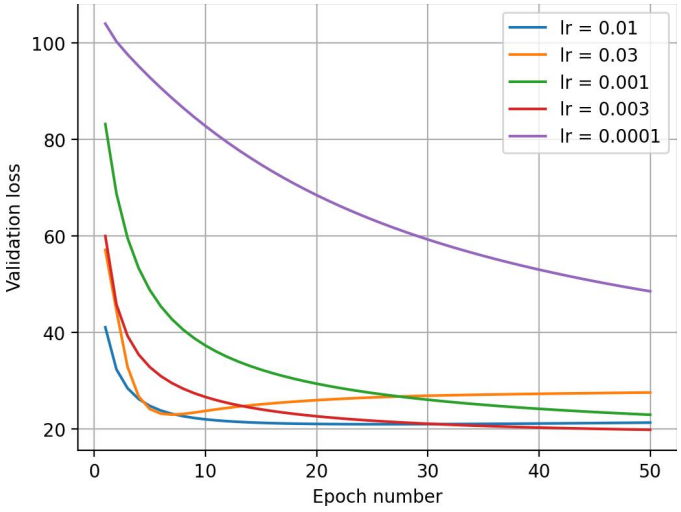


Fig. 11: Part 4 - Validation loss for various learning rates

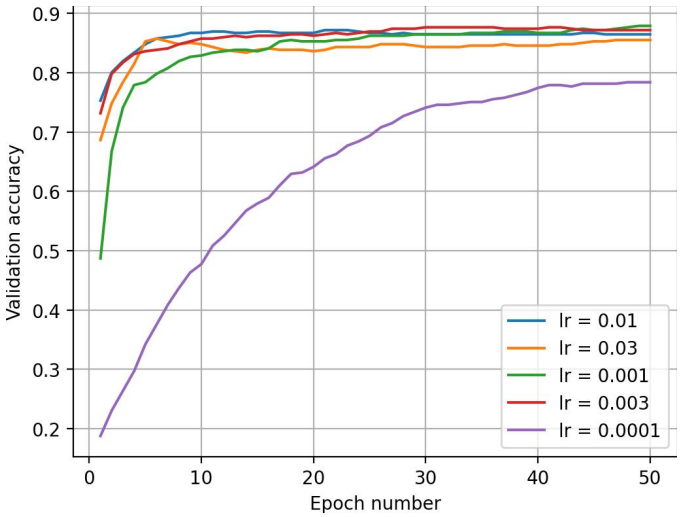


Fig. 12: Part 4 - Validation accuracy for various learning rates

Optimizer	Learning rate	Epoch	Val. accuracy	Test accuracy
SGD	0.001	48	87.89 %	80.50 %
Adam	0.0001	38	88.36 %	81.78 %

TABLE III: Part 4 - Best achieved performance

In this part, the classifier module of the ResNet-18-based model was fine-tuned by freezing all other internal parameters except for the last linear layer. The model was trained for 50 epochs with various learning rates.

Figures 10 through 12 show the evolution of training loss, validation loss and validation accuracy during the experiment. Table III contains information about the most successful model. They confirm the expectation that this model should outperform the model from Section III since there are fewer parameters to learn and much of the model is already well-trained for detection of visual features. The training dataset available in all cases studied in this report is significantly smaller than the ImageNet dataset ResNet-18 was trained on. Using as many pre-trained weights as possible and only investing the available training set into the last layer will improve the model's performance when compared to training from scratch. The test accuracy increased significantly from 60 % to 80 %.

It should be noted that all curves in Fig. 10 through 12 are smoother (lacking peaks) than their counterparts in Section III. My explanation builds on the vastly different optimization landscape. Training of a single linear layer is – in some mathematical sense – more well-behaved, as the optimization landscape has less dimensions and the gradient of loss is simpler. On the other hand, training many hidden layers with nonlinear activation functions at the same time intuitively leads to wild gradients and optimization landscapes. Stepping through such complicated landscape leads to non-monotonic decrease of the training loss and growth of validation accuracy.

Also note that the model is no longer overfitting the training set (compare the non-zero loss in Fig. 10 to its Section III counterpart in Fig. 7). Since all parameters except for the last linear layer are frozen, the model is forced to use the unchanged outputs of hidden layers, i.e. unchanged visual features detected by hidden layers of the network. Sensitivity to these features emerged during training for a different classification task and therefore may not be optimal for the cartoon classification task at hand. By prohibiting the training of hidden layers, we asked the model to build relationships between some classes and visual features, but these visual features may be meaningless for the new classification task. To observe further improvement, the next step in Section V will be to allow the model to also adjust the features it can detect by unfreezing the parameters of hidden layers and training them as well.

V. Full Fine-Tuning

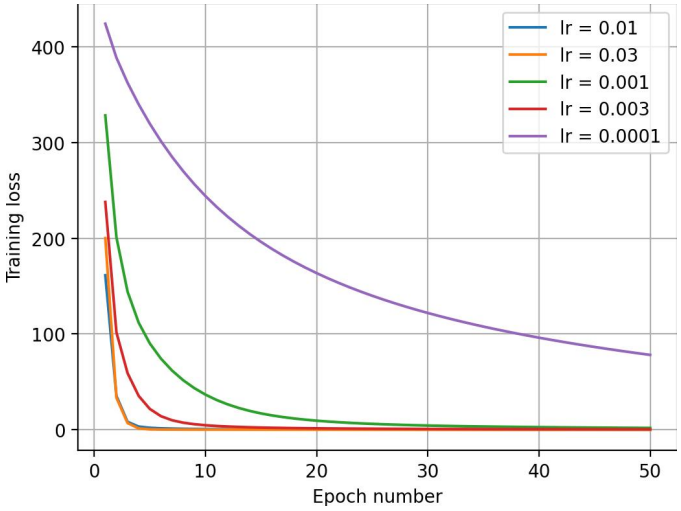


Fig. 13: Part 5 - Training loss for various learning rates

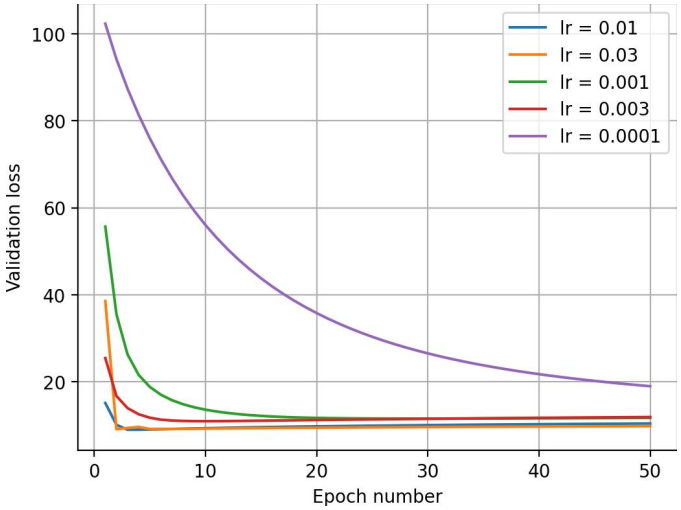


Fig. 14: Part 5 - Validation loss for various learning rates

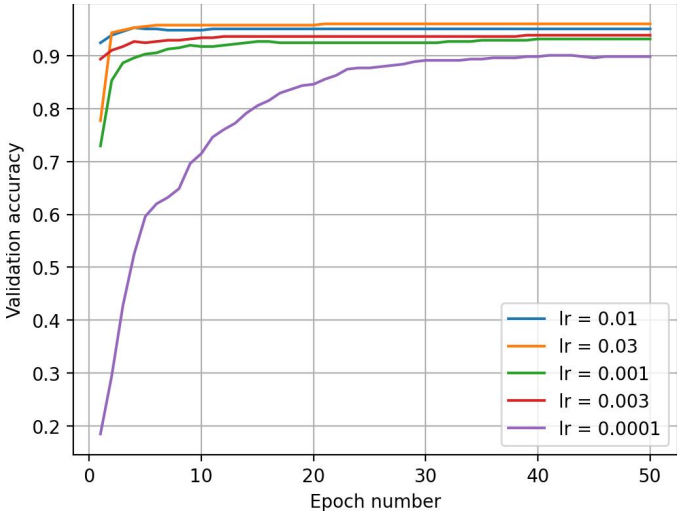


Fig. 15: Part 5 - Validation accuracy for various learning rates

Optimizer	Learning rate	Epoch	Val. accuracy	Test accuracy
SGD	0.03	2	96.67 %	92.80 %
Adam	0.0001	43	95.72 %	93.22 %

TABLE IV: Part 5 - Best achieved performance

In this part, the whole ResNet-18-based model was fine-tuned by using pre-trained values of internal parameters and improving them further using the available training data set. The model can now combine both advantages analyzed in Sections III and IV – using pre-trained weights gives the model a head-start for the training process whilst an adjustment of hidden-layer weights is still possible if needed for better performance.

Figures 13 through 15 show the evolution of training loss, validation loss and validation accuracy during the experiment. Table IV contains information about the most successful model. The model can (eventually) reduce the training loss to zero, as shown in Fig. 13 but overfitting to the training set is not as severe as in Section III, resulting in great validation and test accuracy listed in Table IV.

To continue the line of thought started in Section IV, this model achieves the best test accuracy (and hence best generalization). The pre-trained network is already very capable of detecting visual features useful for a different classification task on the ImageNet dataset. The training process therefore does not need to build all relationships from scratch (like in Section III) and can focus on their adjustment only. In layman's term, the model can identify and keep useful visual features whilst changing visual features not relevant for the new classification task.

VI. Few-Shot

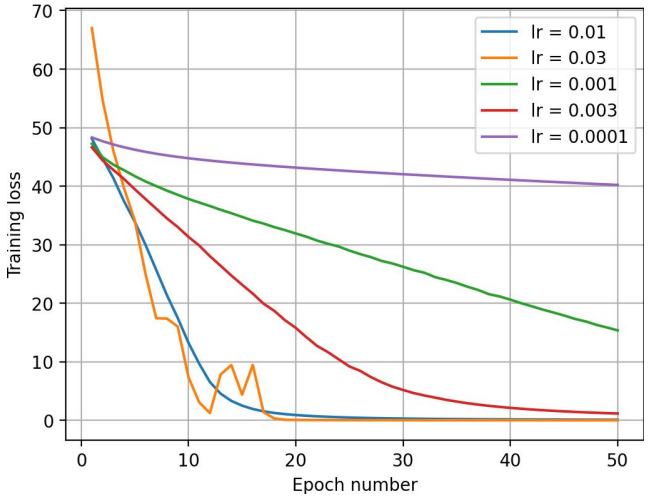


Fig. 16: Part 6.3 - Training loss for various learning rates

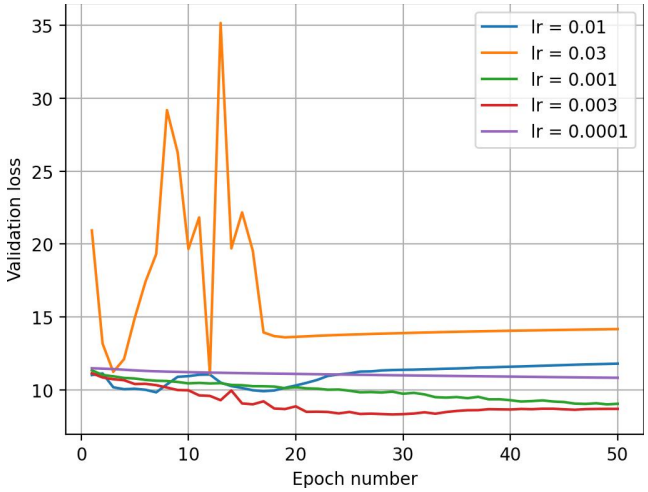


Fig. 17: Part 6.3 - Validation loss for various learning rates

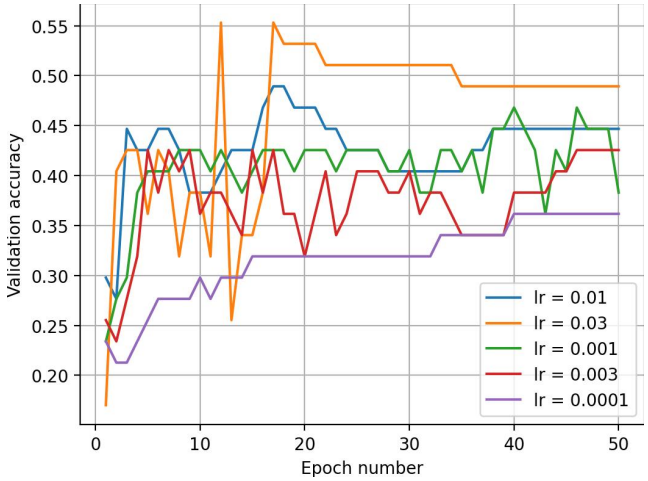


Fig. 18: Part 6.3 - Validation accuracy for various learning rates

Optimizer	Learning rate	Epoch	Val. accuracy	Test accuracy
SGD	0.03	11	55.32 %	48.62 %
Adam	0.003	23	61.70 %	44.92 %

TABLE V: Part 6.3 - Best achieved performance

This Section repeats experiments from Sections III through V but uses a significantly smaller training dataset "Few-Shot" with only roughly 190 samples instead of roughly 1600. This leads to higher overfitting (worse generalization) by the model and, by extension, to worse test accuracy.

A. Training From Scratch

Figures 16 through 18 show the evolution of interesting quantities during the experiment and Table V contains information about the most successful model. Comparing with Table II (the same task but with larger training dataset) shows roughly 12 % decrease in the test accuracy.

A possibly obvious but still noteworthy trend can be observed in Fig. 16 (or later in Fig. 22) when compared with its counterpart from Section III in Fig. 7 (or Section V in Fig. 13). Since the model's expressiveness is very rich considering the size of available training sets, it can achieve zero training loss on either of the training datasets. Nevertheless due to the smaller size of the "Few-Shot" dataset, each training epoch achieves a smaller improvement of the model. Keeping the maximal number of training epochs at 50 for both larger and smaller datasets may lead to some performance loss an unfair comparisons, since the model training process converges slower when each epoch consists of less training samples.

B. Fine-Tuning

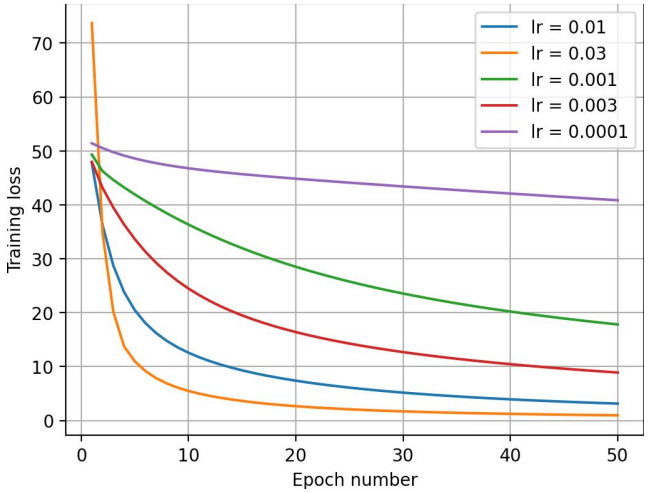


Fig. 19: Part 6.4 - Training loss for various learning rates

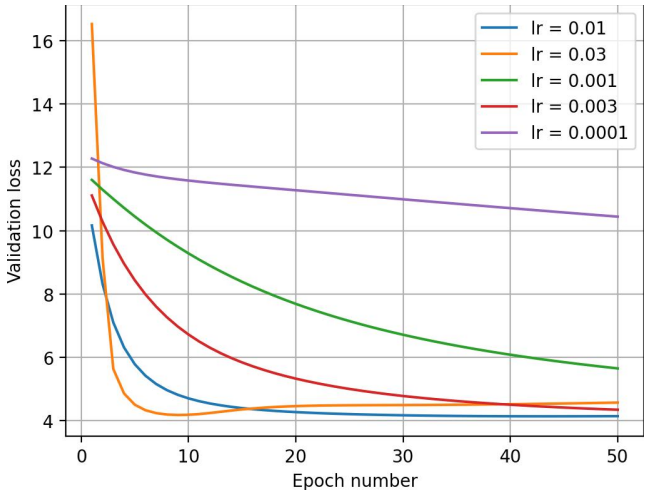


Fig. 20: Part 6.4 - Validation loss for various learning rates

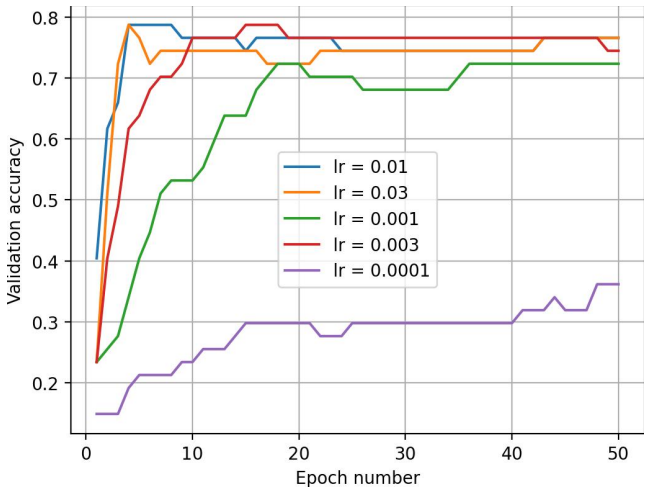


Fig. 21: Part 6.4 - Validation accuracy for various learning rates

Optimizer	Learning rate	Epoch	Val. accuracy	Test accuracy
SGD	0.01	3	78.72 %	66.08 %
Adam	0.03	21	85.11 %	69.64 %

TABLE VI: Part 6.4 - Best achieved performance

Figures 19 through 21 show the evolution of interesting quantities during the experiment and Table VI contains information about the most successful model. Comparing with Table III (the same task but with larger training dataset) shows roughly 14 % decrease in the test accuracy.

C. Full Fine-Tuning

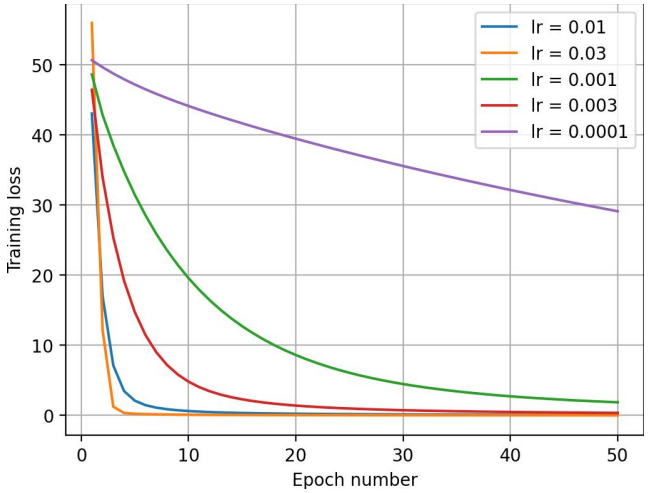


Fig. 22: Part 6.5 - Training loss for various learning rates

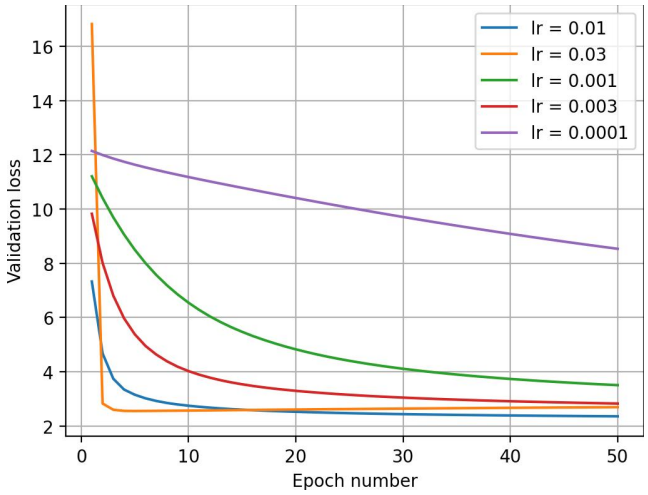


Fig. 23: Part 6.5 - Validation loss for various learning rates

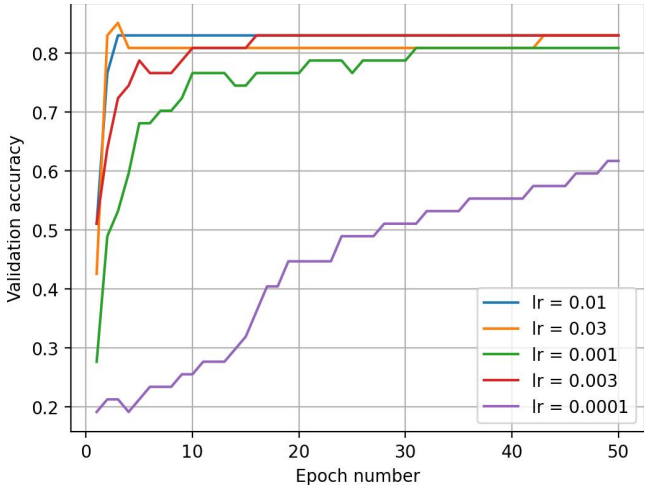


Fig. 24: Part 6.5 - Validation accuracy for various learning rates

Optimizer	Learning rate	Epoch	Val. accuracy	Test accuracy
SGD	0.03	2	85.10 %	77.94 %
Adam	0.0001	3	85.11 %	79.70 %

TABLE VII: Part 6.5 - Best achieved performance

Figures 22 through 24 show the evolution of interesting quantities during the experiment and Table VII contains information about the most successful model. Comparing with Table IV (the same task but with larger training dataset) shows roughly 15 % decrease in the test accuracy.

VII. Training Data Augmentation

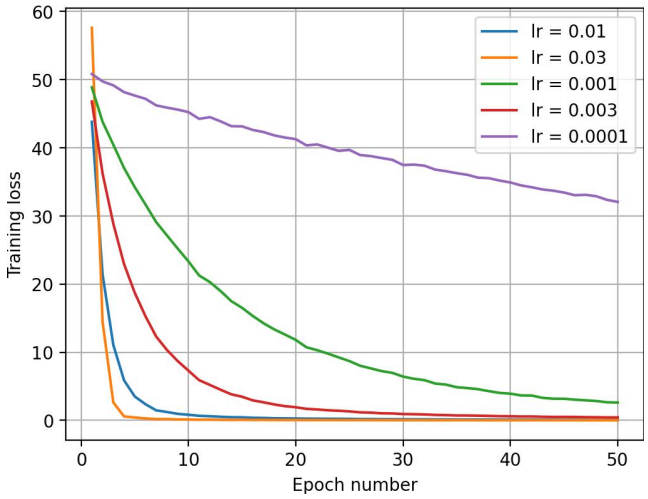


Fig. 25: Part 7 - Training loss for various learning rates

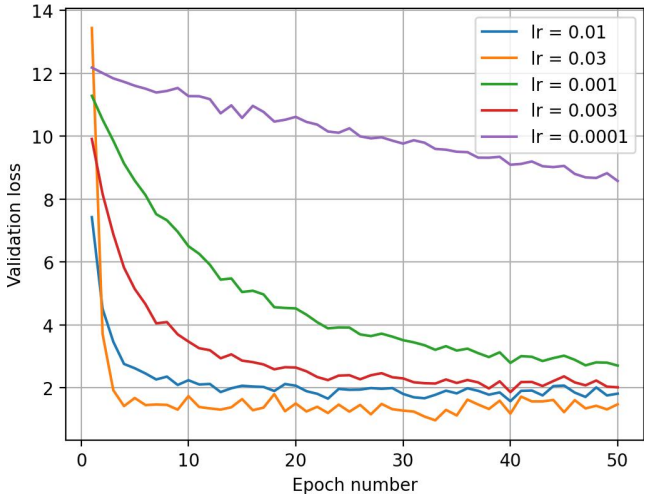


Fig. 26: Part 7 - Validation loss for various learning rates

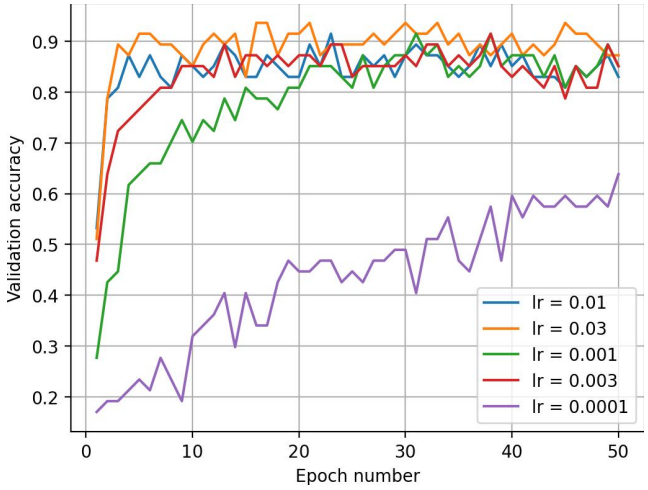


Fig. 27: Part 7 - Validation accuracy for various learning rates

To improve the performance when given a limited amount of training data, the training set was augmented by simple random transformations. This section presents results achieved by fine-tuning the full model (same as Section V and VI-C) using an augmented training dataset.

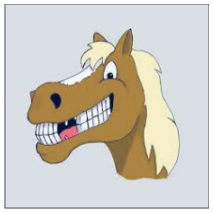
The PyTorch library offers a wide spectrum of parameterized transformations, including transformations of image geometry (flipping along individual axes, rotations etc.) or color channels (channel inversions, Grayscale, blur etc.). To get a feel for the impact data augmentation has on the training set, I have tested the model with many transformations and summarized achieved results in Table VIII. All models were fully fine-tuned and tested on the small ("Few-Shot") dataset. The baseline for comparison is fine-tuned model from Section VI-C trained on the small dataset with no transformations, yielding results matching the performance presented in Tab. VII. The goal of augmentation is to increase the test accuracy further, in this case reach or exceed 80 %.

There are several noteworthy trends to highlight. First of all, none of the utilized color channel transformations with the exception of random inversion of individual color channels lead to any improvement of model's performance and instead caused a noticeable degradation of test accuracy. My interpretation is that true colors are very important for classification of cartoons – jitter could be useful for real images that already contain (or may contain) some noise, but most of the available training, validation and test sets are digitally created "perfect" images. This could be an explanation for why the Gaussian blur transformation did not achieve significant improvement either – it managed to slightly exceed 80 % of test accuracy for some choices of the kernel size, best results were acquired for kernel of size 7.

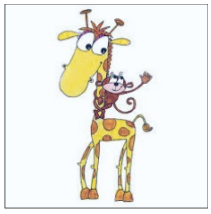
More significant improvements were achieved using geometric transformations. Random horizontal flip by itself managed to achieve almost 83 % of test accuracy. This result improved further when followed by a random rotation (random affine transformation with no translation). The sequence of horizontal flip followed by random rotation was tested many times with various hyperparameters, yielding the best test accuracy approx 84.5 % for angle in the range $[-10, 10]$ °.

Applied augmentation(s)	Learning rate	Epoch	Val. accuracy	Test accuracy
None (Baseline for comparison)	0.03	7	85.11 %	77.99 %
ColorJitter(brightness=0.01)	0.03	3	78.72 %	17.46 %
ColorJitter(brightness=0.05)	0.03	3	80.85 %	21.96 %
ColorJitter(contrast=0.05)	0.03	7	85.11 %	14.56 %
ColorJitter(saturation=0.05)	0.03	20	82.98 %	18.60 %
ColorJitter(hue=0.05)	0.01	20	78.72 %	33.54 %
Greyscale()	0.03	1	80.85 %	68.60 %
RandomInvert(p=0.5)	0.03	5	93.62 %	78.08 %
RandomHorizontalFlip(p=0.5)	0.03	12	97.87 %	82.78 %
RandomAffine(± 1 deg)	0.01	8	87.23 %	78.46 %
RandomHorizontalFlip(p=0.5), RandomAffine(± 1 deg)	0.03	4	93.62 %	80.31 %
RandomHorizontalFlip(p=0.5), RandomAffine(± 5 deg)	0.03	9	95.74 %	83.44 %
RandomHorizontalFlip(p=0.5), RandomAffine(± 7 deg)	0.03	13	97.87 %	82.35 %
RandomHorizontalFlip(p=0.5), RandomAffine(± 10 deg)	0.03	17	97.87 %	84.54 %
RandomHorizontalFlip(p=0.5), RandomAffine(± 22 deg)	0.01	24	97.87 %	81.83 %
RandomHorizontalFlip(p=0.5), RandomAffine(± 30 deg)	0.03	14	95.74 %	81.12 %
RandomHorizontalFlip(p=0.5), RandomAffine(± 45 deg)	0.01	17	97.87 %	79.46 %
RandomHorizontalFlip(p=0.5), RandomAffine(± 60 deg)	0.01	27	97.87 %	77.99 %
RandomHorizontalFlip(p=0.5), RandomAffine(± 10 deg), RandomAffine(± 10 deg)	0.03	12	97.87 %	82.64 %
GaussianBlur(1)	0.03	2	89.36 %	78.56 %
GaussianBlur(3)	0.01	11	91.49 %	77.75 %
GaussianBlur(5)	0.01	38	89.36 %	80.31 %
GaussianBlur(7)	0.03	34	95.74 %	81.97 %
GaussianBlur(9)	0.01	7	91.49 %	80.12 %
GaussianBlur(13)	0.03	33	97.87 %	79.65 %
GaussianBlur(17)	0.01	34	91.49 %	80.27 %
GaussianBlur(23)	0.03	34	95.74 %	78.37 %
GaussianBlur(27)	0.01	36	91.49 %	80.22 %

TABLE VIII: Part 7 - Best achieved performance using various transformations and SGD optimizer.

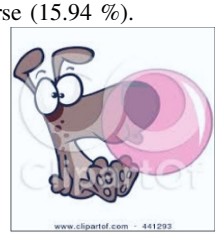


(a) Part 3 - Misclassified horse image predicted as dog (50.41 %), giraffe (20.32 %) or horse (15.94 %).

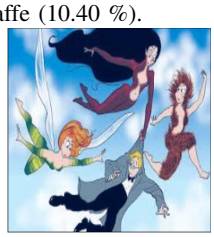


(b) Part 3 - Misclassified giraffe image predicted as person (64.12 %), dog (19.68 %) or giraffe (10.40 %).

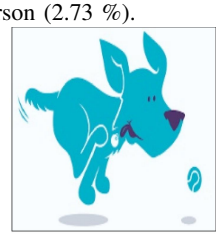
(c) Part 3 - Misclassified dog image predicted as elephant (78.72 %), dog (15.70 %) or person (2.73 %).



(d) Part 4 - Misclassified dog image predicted as elephant (63.41 %), dog (18.09 %) or horse (7.42 %).



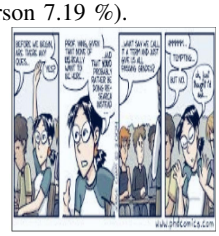
(e) Part 4 - Misclassified person image predicted as horse (90.15 %), person (6.78 %) or elephant (1.49 %).



(f) Part 4 - Misclassified dog image predicted as horse (45.32 %), elephant (42.07 %) or person 7.19 %).

(g) Part 5 - Misclassified giraffe image predicted as dog (89.07 %), elephant (9.68 %) or horse (0.81 %).

(h) Part 5 - Misclassified guitar image predicted as giraffe (70.81 %), person (15.76 %) or horse (9.84 %).



(i) Part 5 - Misclassified person image predicted as elephant (36.22 %), giraffe (27.02 %) or person (18.04 %).

Fig. 28: Examples of misclassified images with correct labels and model predictions.