Evaluation of image classification techniques

Manuele Macchia

Machine Learning and Deep Learning
Politecnico di Torino
A.Y. 2019/2020

Contents

1 Introduction

The homework consists in training a convolutional neural network (CNN) for image classification, specifically AlexNet [1]. This neural network contains eight learned layers; the first five are convolutional layers, and the remaining three are fully connected layers. The dataset that we use for training is Caltech-101 [2], which consists of hundreds of pictures of objects belonging to 101 categories.

Firstly, we train AlexNet from scratch on Caltech-101. Then, we perform transfer learning, exploiting the pretrained version of AlexNet on the ImageNet dataset [3]. We evaluate data augmentation techniques by preprocessing training images. Lastly, we experiment with another convolutional neural network, ResNet [4], and compare the results.

2 Data

The Caltech-101 dataset consists of pictures of objects belonging to 101 categories, with hundreds of images per category. The resolution of the images is not standard, however most images are about 300 by 200 pixels. Figure 1 shows a batch of 64 images taken from the dataset.

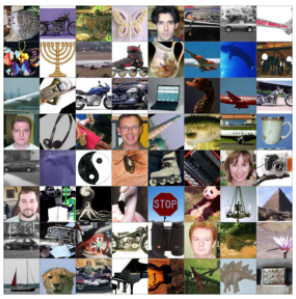


Figure 1: Samples taken from Caltech-101

The dataset is split into a training set and a test set. Performing k-fold cross-validation is too computationally expensive, therefore we hold out half of the training set for validation. The dataset is split in a stratified fashion, so that every set has approximately the same class distribution.

We use the training set to train the neural network, the validation set to perform hyperparameter tuning and model selection, and the test set to evaluate the resulting model. There are about nine thousand images in total. We exclude from the dataset the additional *background* category provided. The train and validation sets contain 2892 samples each, while the test set contains 2893 images.

3 Implementation

We use the *PyTorch* library to implement the neural networks and perform the experiments. More specifically, we use PyTorch's implementation of AlexNet, ResNet-18 and ResNet-34, and the data utilities of the library.

All the models we evaluate use the cross entropy loss function, useful for training a classification problem with multiple classes such as ours. We use stochastic gradient descent (SGD) with momentum as the optimization algorithm for minimizing the loss function. The mini-batch size for SGD is set to 256, allowing for larger learning rates.

The training, validation and test sets are transformed in order to comply with the resolution required by AlexNet. More specifically, we resize all images to 256 pixels on the short side, preserving their original ratio. Then, we crop a 224 by 224 pixels patch at the center of the image, we transform it into a PyTorch tensor and we normalize it with mean 0.5 and standard deviation 0.5 for all three channels.

3.1 Implementation details

`notebook.ipynb` contains the main logic of the program and the results of the experiments that we carried out. The Caltech-101 dataset is available along with train and test indices (`train.txt` and `test.txt`) and two other python files. `caltech.py` contains the Caltech-101 handler class, with methods to initialize and handle the dataset and to perform a stratified split for training and validation sets. `manager.py` contains the network manager class, which handles training, validation and testing of a neural network, as well as logging results such as training and validation loss and accuracy over epochs. All project files are available at https://github.com/manuelemacchia/image-classification-techniques.

3.2 Training from scratch

Firstly, we train AlexNet from scratch using the training set. After each training epoch, we evaluate the model on the validation set and save the model with the lowest loss. Finally, the best model is evaluated on the test set.

The standard implementation trains using stochastic gradient descent with momentum 0.9 for 30 epochs with an initial learning rate of 0.001. The weight decay parameter is set to 0.00005. This regularization parameter is useful to penalize the biggest weights, which are often related to learning errors, improving generalization. The standard training procedure adopts a strategy for decreasing the learning rate by a factor 0.1 every 20 epochs, which we will refer to as decaying policy.

We begin to search the hyper-parameter space by tuning the learning rate first. This is the most significant hyper-parameter to tune, as it has a significant impact on the model performance [5]. We search for the best learning rate between 0.1 and 0.00001, as often suggested in deep learning literature [6]. Then, we adjust the decaying policy hyper-parameters, as they are strictly related to the learning rate. We refer to the number of epochs after which the learning rate decays as step size, and the decrease factor as γ. We perform a grid search with values $[10, 15, 20]$ for the step size and $[0.05, 0.1, 0.2]$ for γ.

Due to the high computational cost and limitations of the development environment, we are not able to perform a grid search with learning rate, step size and γ jointly. We therefore decide to begin tuning the learning rate separately, using step size and γ of the standard implementation. Once the best learning rate is fixed, we tune the decaying policy parameters jointly.

We monitor both loss and accuracy for both training and validation sets, with respect to the training epochs, and tune the hyper-parameters accordingly. We compute the loss at each epoch by averaging it over all mini-batches.

3.3 Transfer learning

The Caltech-101 dataset is a relatively small dataset, and convolutional neural network such as AlexNet need a large amount of data to reach high accuracy scores. To reduce the amount of annotated data needed and to avoid learning the network parameters from scratch, we may use transfer learning.

The implementation of AlexNet provided by PyTorch offers a pre-trained version of the neural network. The dataset used to train this network is ImageNet [3], an image database of millions of samples. We may leverage this pre-trained model to solve a task similar to the original one, such as image classification on Caltech-101.

We carry out the same data transformations as before, except for normalization. We use statistics computed on the ImageNet dataset, i.e., mean $(0.485, 0.456, 0.406)$ and standard deviation $(0.229, 0.224, 0.225)$. Each element of the tuple corresponds to a channel.

Firstly, we tune the learning rate, searching the hyper-parameter space between 0.001 and 0.00001. We choose smaller learning rates for fine-tuning in order not to drift too much from the original learned weights, as we expect the pre-trained model to be already relatively good. The last fully connected layer of the pre-trained model is re-initialized to a new fully connected layer with 101 output neurons, as the original model trained on ImageNet outputs 1000 values instead. This new fully connected layer will be trained from scratch.

Then, we set the best learning rate for our model and tune the decaying policy hyper-parameters, performing the same grid search as before. The models are trained for 50 epochs, so that we may evaluate the difference from the standard number of epochs.

Finally, with the best hyper-parameters found, we perform network freeze experiments by training only fully connected layers, only convolutional layers and finally both convolutional layers and the last fully connected layer.

3.4 Data augmentation

Caltech-101 is a relatively small dataset, and data augmentation techniques may help in improving the results and avoiding overfitting. We apply transformations to the original images, e.g., random crop and horizontal flip, to artificially increase the cardinality of the dataset.

We apply three different sets of preprocessing for training images, shown in Figure 2, and compare the results. The first set consists in a center crop, a color jitter transformation that consists in randomly changing the brightness, contrast and saturation of the image, and a random horizontal flip which flips the image with probability 0.5. The second set consists in a random crop, stronger color jittering and a random horizontal flip. The

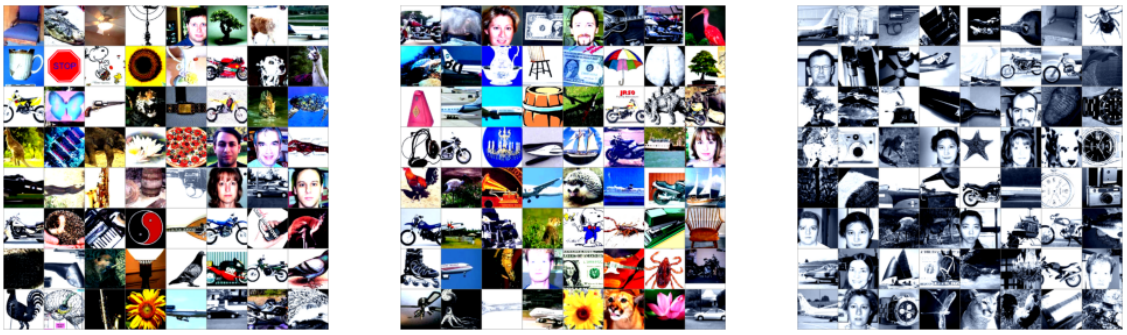


Figure 2: Data augmentation, visualizations of the three transformation sets on samples taken from Caltech-101

third set performs a random crop, a grayscale transformation and a random horizontal flip with probability 0.5.

We train three neural networks, one for each data transformation set, and compare them to a baseline network without any data augmentation technique. We use the best hyper-parameters previously found and apply transfer learning from a model originally trained on ImageNet. No network freezing is applied, therefore we train all parameters of the network.

3.5 ResNet-18 and ResNet-34

We compare the best results achieved with AlexNet to ResNet-18 and ResNet-34. We tune the learning rate of the ResNet networks, searching for the best learning rate between 0.1 and 0.0001. We use the standard decaying policy hyper-parameters: step size 20 and γ equal to 0.1. All networks are trained for 30 epochs using transfer learning. The models are pre-trained on ImageNet.

ResNet-18 and ResNet-34 are larger networks compared to AlexNet, with more layers and more parameters to learn. As they consume more GPU memory, we decrease the batch size to 64. No network freezing is applied, therefore we train all parameters of the networks.

4 Results and discussion

4.1 Training from scratch

The standard implementation trains with an initial learning rate of 0.001, step size 20 and γ equal to 0.1. After 30 epochs, the model reaches an accuracy on the test set of 9.2%, which is a very bad result. The training and validation losses are very high, showing a very slow rate of convergence.

Firstly, we tune the initial learning rate only, setting the step size to 20 and γ to 0.1. The results are shown in Figure 3a. The plots demonstrate that if the learning rate is too large, the loss oscillates, therefore the model has trouble converging and may even diverge.

LR	Loss	Accuracy
0.001	0.80417	0.80290
0.0001	1.40903	0.66667
0.00001	3.64157	0.24205

Table 1: Transfer learning, learning rate tuning results on validation set with decaying policy set to step size 20 and $\gamma = 0.1$

Step	γ	Loss	Accuracy
10	0.05	0.80016	0.80774
10	0.1	0.71971	0.82227
10	0.2	0.75871	0.80498
15	0.05	0.78159	0.81501
15	0.1	0.78332	0.80498
15	0.2	0.78108	0.81086
20	0.05	0.79103	0.79529
20	0.1	0.75765	0.79703
20	0.2	0.77609	0.80705

Table 2: Transfer learning, decaying policy tuning results on validation set with learning rate set to 0.001

Moreover, if the learning rate is too small, learning proceeds too slowly and may become stuck at a high loss value.

A good compromise for our network and dataset is 0.01, which produces the smallest validation loss and the highest accuracy at the last epoch. We can observe a downward trend of the validation loss, which suggests that the model has not yet reached a plateau and may produce better results given a higher amount of epochs to train.

We fix the learning rate to 0.01 and tune the decaying policy hyper-parameters. The loss and accuracy over epochs for the validation set is shown in Figure 3b. The models perform very similarly until the 10th epoch. After that, the most aggressive decaying policies, such as the ones with step size 10, perform worse than the ones with higher step size. The best decaying policy, reaching the lowest loss value on the validation set and the highest accuracy, is step size 20 with γ equal to 0.1.

Finally, we test the optimized hyper-parameters on the test set. We train the neural network from scratch, using initial learning rate 0.01, step size 20 and γ equal to 0.1. After 30 epochs, the model reaches an accuracy of 30.9% on the test set, which is higher than the score obtained with the standard parameters, but still quite a bad result.

4.2 Transfer learning

We carry out hyper-parameter tuning in the transfer learning setting. The learning rate that minimizes the loss on the training and validation sets is, by far, 0.001. Models with lower learning rates converge at higher losses and obtain lower accuracy scores on the validation set, as shown in Table 1, where the scores of the best models on the validation set are reported.

We set the initial learning rate to 0.001 and tune the decaying policy hyper-parameters. The scores of the best performing models are reported in Table 2. The results show that the best combination of step size and γ is 10 and 0.1. This model reaches an accuracy score on the test set of 81.6%.

All models evaluated reach their best results well ahead of the 50th epoch mark, therefore we will set the number of training epochs to 30 from now on.

We compare the model trained with the best hyper-parameters to partially frozen networks. The validation loss and accuracy plot over epochs in Figure 4 shows how badly the network performs when only convolutional layers are fine-tuned and the rest of the network is frozen. On the other hand, training only fully connected layers or training both convolutional layers and the last fully connected layer yields results comparable to the model obtained by training the whole network. This highlights the importance of training the last fully connected layer for the desired task, as it is essential to correctly classify test samples.

Training only fully connected layers and training both convolutional layers and the last fully connected layer result in similar accuracy scores on the test set to the fully trained network, albeit slightly lower, with 80.8% and 81.2% respectively. The only exception is the network where only convolutional layers are trained, which performs worse than the network trained from scratch, scoring an accuracy of 22.8% on the test set.

4.3 Data augmentation

The results of data augmentation techniques are reported in Figure 5. The validation loss and accuracy plots show how only the first set of transformations reached a result comparable to the baseline within 30 training epochs. All results are worse than the baseline, which uses a square crop at the center of the original image without any further transformation.

The accuracy score on the test set of the first and second transformation sets are very similar, 81% and 80.8%, while the grayscale transformation set obtains 75.2%. This suggests that this transformation set excessively modifies the underlying data distribution of the training set by eliminating color from the image, degrading the performance on the test set.

4.4 ResNet-18 and ResNet-34

The best learning rate found for both ResNet models is 0.01. With this learning rate and the standard decaying policy, the models converge very fast and perform better than AlexNet. The results of ResNet-18 and ResNet-34 against the AlexNet baseline are reported in Figure 6. The plots show that both ResNet models converge faster than AlexNet and are about 10% more accurate on the validation set.

The accuracy score of ResNet-18 on the test set is 93.0%, while ResNet-34 scores 93.8%, a better result as we may expect from a larger neural network. We may expect to get even better results with a more thorough hyper-parameter tuning procedure. In comparison, AlexNet reaches an accuracy score on the test set of 81.9%.

5 Conclusion

We train AlexNet, a convolutional neural network for image classification, on the Caltech-101 dataset, which consists of pictures of objects belonging to 101 categories.

Firstly, we train AlexNet from scratch, obtaining suboptimal results both before and after hyper-parameter tuning. Then, we perform transfer learning from a network pre-trained on ImageNet. The resulting model scores well on the test set, reaching an accuracy of 81.6%. We compare this result to partially frozen network training, obtaining comparable results except for only training convolutional layers. We apply data augmentation techniques, but we do not achieve better results on the evaluation sets.

Finally, we compare ResNet-18 and ResNet-34 to AlexNet. As expected, we get better results. The best model, ResNet-34, achieves an accuracy score of 93.8% on the test set.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, May 2017.

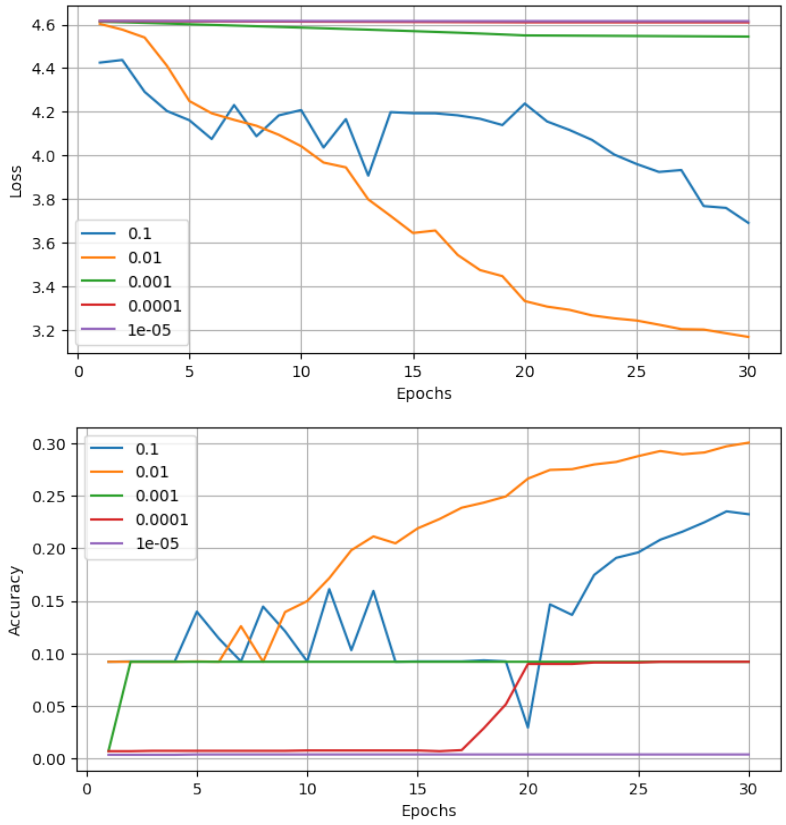
[2] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, page 9, 2004.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, Miami, FL, June 2009. IEEE.

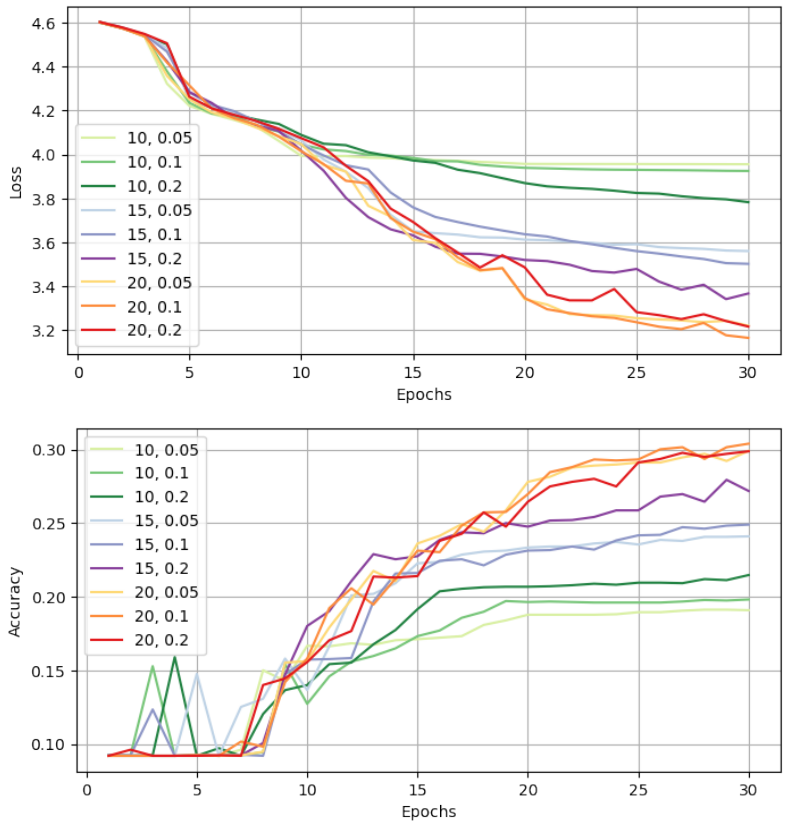
[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. arXiv: 1512.03385.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts, 2016.

[6] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *arXiv:1206.5533 [cs]*, September 2012. arXiv: 1206.5533.



(a) Learning rate tuning



(b) Decaying policy tuning

Figure 3: Training from scratch, loss and accuracy over epochs on validation set

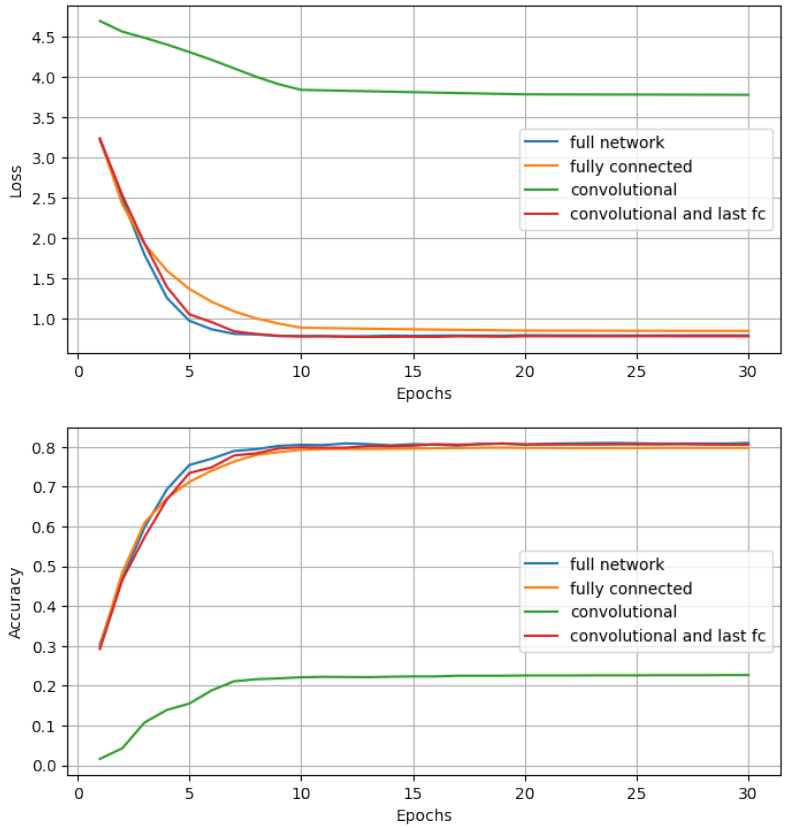


Figure 4: Transfer learning, comparison between training all weights, training only the fully connected layers, training only the convolutional layers and training convolutional layers and the last fully connected layer. Showing loss and accuracy on the validation set.

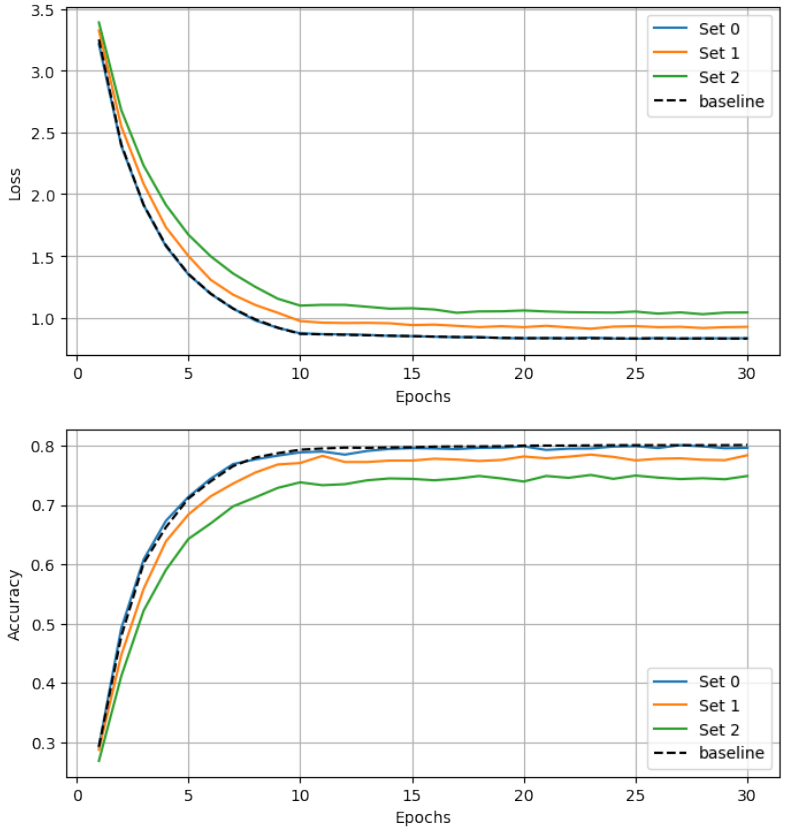


Figure 5: Data augmentation, comparison between data transformation sets presented in Figure 2 and a baseline without any data augmentation technique. Showing loss and accuracy over epochs on the validation set.

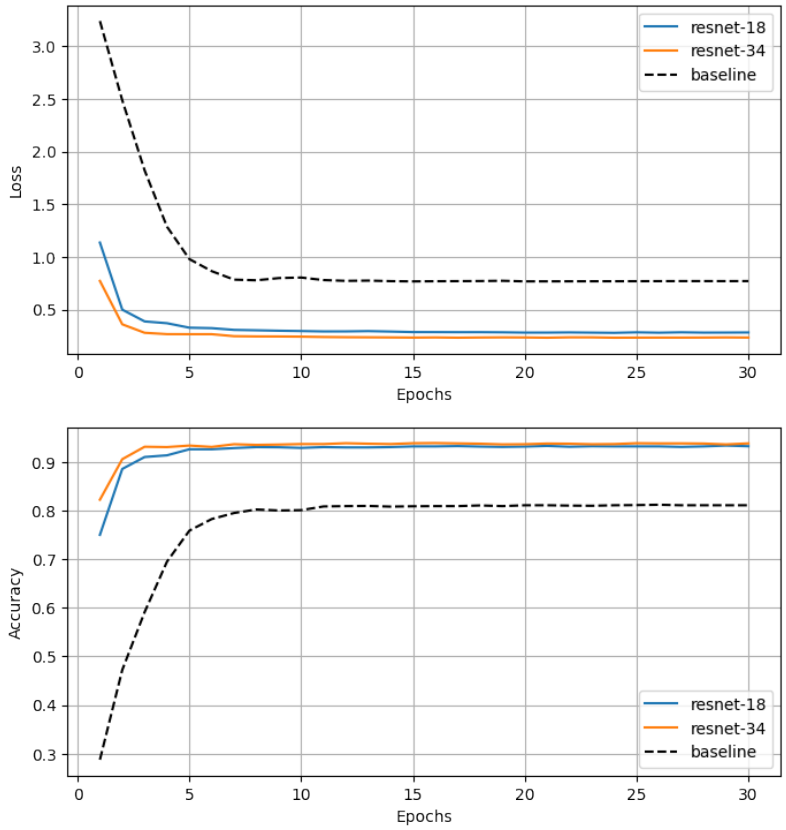


Figure 6: Comparison between ResNet-18, ResNet-34 and AlexNet (baseline). Showing loss and accuracy over epochs on the validation set.