High-Level Computer Vision

Exercise 3 Report: Convolutional Neural Networks

Akshay Dodwadmath
akdo00001@stud.uni-saarland.de

Manuela Ceron Viveros
mace00001@stud.uni-saarland.de

Jobin Idiculla Wattasseril
jowa00004@stud.uni-saarland.de

May 28, 2021

Question 1: Implement Convolutional Network

a)

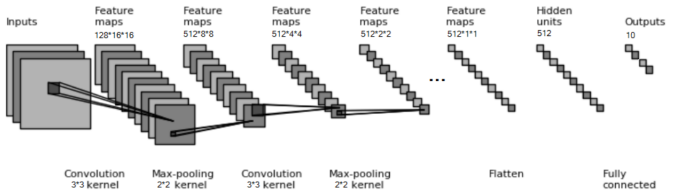

Fig. 1. 5-layered CNN architecture.

We implement the given CNN architecture and perform training on the CIFAR-10 dataset. We obtain a training accuracy of 95.48% and a validation accuracy of 78.7%. Fig. 2 shows the plot of training and validation loss, over the default value of 20 epochs.

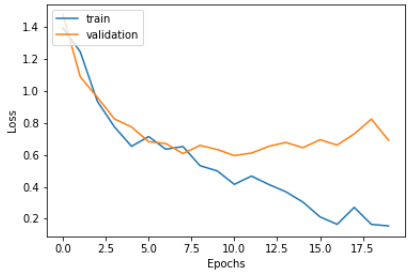

Fig. 2. Training and validation loss during network training.

b)

We implement the `PrintModelSize` function, and use it to calculate the number of parameters of the network. We obtain a value of 7678474 as the total number of trainable parameters of the model.

c)

We implement the `VisualizeFilter` function and use it to visualize the 128 3×3 filters of the first convolutional layer of the network, stacking them into an 8×16 grid of filters. Fig. 3 shows the filter visualisations before training, whereas Fig. 4 shows the filter visualisations after training.

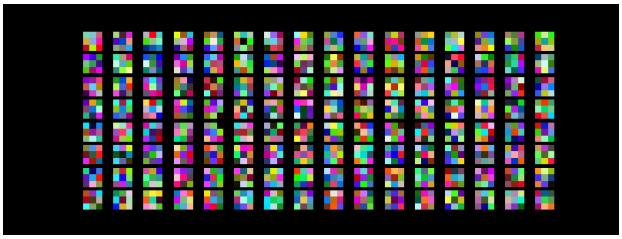

Fig. 3. Filters of the convolutional layer before training.

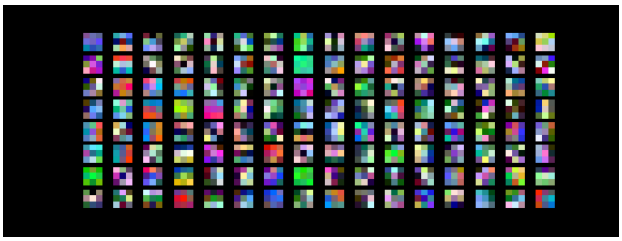

Fig. 4. Filters of the convolutional layer after training.

We observe that before training, the filter weights are randomly initialized, and this is reflected in the visualisation in Fig. 3, as each 1×1 entry in each filter seems to have a random color. However, after training, the learned filters are no longer random and this is observed in the visualisation in Fig. 4, where (i) darker colours seem to dominate across most filters, and (ii) clumping of similar colours within a filter is observed for example in filters at positions $(2, 8), (3, 8), (7, 1), (7, 8)$.

Question 2: Improve training of Convolutional Networks

a)

We add Batch normalization layers to the network, while keeping other hyper-parameters the same. Fig. 5 shows the plot of training and validation loss, over the default value of 20 epochs.

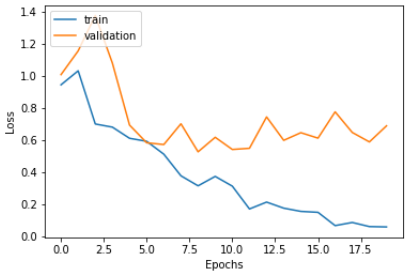

Fig. 5. Training and validation loss with batch normalization.

At the end of 20 epochs, we report a training accuracy of 97.94% and a validation accuracy of 81.6%. On comparing Fig. 2 and Fig. 5, we observe that

the validation loss fluctuates more with batch normalization implying slower convergence, but the validation accuracy is higher with batch normalization, implying less overfitting.

b)

We implement early stopping to stop training and save the current best model, when the validation accuracy stops improving. We include an `early_stopping_tolerance` value to control the stagnancy of the validation accuracy; `early_stopping_tolerance` = 3 means that if the validation accuracy does not improve over 3 epochs, the training is terminated.

To verify the performance gains with early stopping, we evaluate a model with and without early stopping; we conduct 3 training runs with: (i) a model with an `early_stopping_tolerance` = 3 that saves the best model weights (named **BestModel** in Tab. 1) and (ii) a model that runs for 50 epochs, saving only the last epoch's weights (named **LastModel** in Tab. 1). We observe that the average validation accuracy is comparable for both models, but the model that ran for 50 epochs exhibits more overfitting, as evidenced by its higher training accuracy. Also, we see that the number of epochs for which the model with early stopping ran, is significantly shorter than 50 epochs, thereby being faster. Tab. 1 summarizes the results of our experiment:

	Average over 3 runs		
	Training accuracy	**Validation accuracy**	**Epochs per run**
BestModel	94.00	82.33	14, 13, 11
LastModel	99.99	83.83	50, 50, 50

Tab. 1. Accuracies with and without early stopping.

Question 3: Improve generalization of Convolutional Networks

a)

We added data augmentation to effectively improve the size of our dataset. Using the `torchvision.transforms` package of PyTorch, we tried different geometric and color space data augmentations for the CIFAR-10 dataset. We tuned the hyperparameters of each of the techniques to get the best validation performance. Here we show the results of some of our transformations by plot-

ting the different validation accuracies of each of the transformation with their respective hyperparameters :

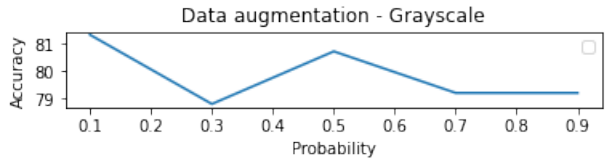

Fig. 6. Data Augmentation with Grayscale transform

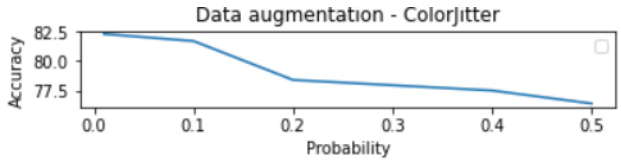

Fig. 7. Data Augmentation with Color Jitter transform

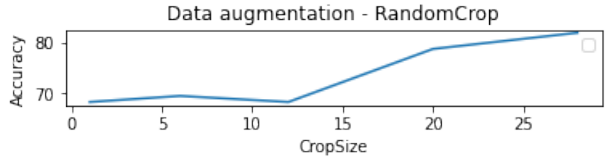

Fig. 8. Data Augmentation with Random Cropping

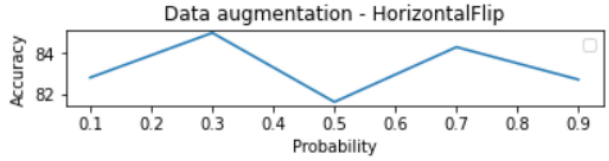

Fig. 9. Data Augmentation with Horizontal Flip

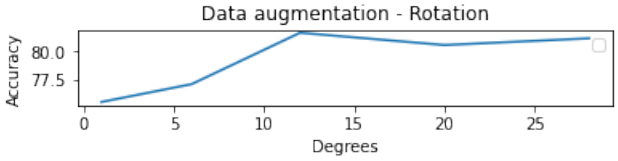

Fig. 10. Data Augmentation with Rotation

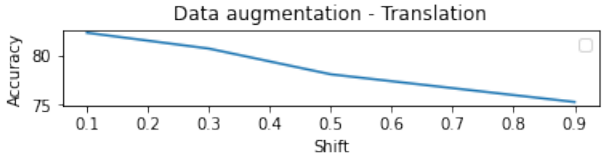

Fig. 11. Data Augmentation with Translation

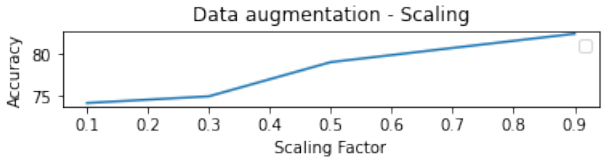

Fig. 12. Data Augmentation with Scaling

From these figures, we can see that most of our transformations are not improving the validation accuracies when they are enabled. In fact, only the horizontal flip (with probability = 0.3) and rotation (degrees = 12) transformations are giving a reasonable improvement in the validation accuracies. Enabling these two transformations, we ran the training and were able to get a validation accuracy of 84.1% and test accuracy of 83.9% with 20 epochs and `early_stopping_tolerance` = 3. Data augmentation seems to improve the generalization of our network.

b)

We add Dropout layers to the network as a regularization technique and tune the dropout probability hyperparameter p. We experiment with values of p in the interval $[0.1, 0.9]$ with increments of 0.1. We also disable data augmentations from the previous step to observe the effect of using Dropout layers. Fig. 13 shows the plot of the training and validation accuracies corresponding to each value of p.

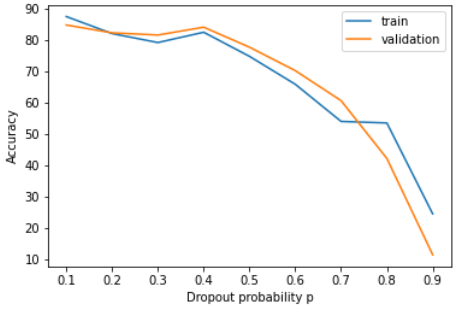

Fig. 13. Dropout probability vs training/validation accuracy.

In our experiments, we observe the highest validation accuracy of 84.9% with $p = 0.1$ (training accuracy = 87.61%). However, we also observe that for $p = 0.4$, we get a close validation accuracy of 84.2% with a slightly lesser training accuracy of 82.59% implying lesser overfitting.

We also trained a model with the data augmentations enabled along with dropout; we set the number of epochs to 40, the dropout probability to 0.1 (best value from above), and the `early_stopping_tolerance` to 5. The model training got terminated at epoch 26 and we observed a higher validation accuracy of 87.2%. Though there was an increase in the training accuracy as well (90.87%), we also report a very good test accuracy of 87.2%. This means our model with both data augmentation and dropout enabled is able to generalize well. Fig. 14 and Fig. 15 show the plots of training/validation loss and accuracy respectively:

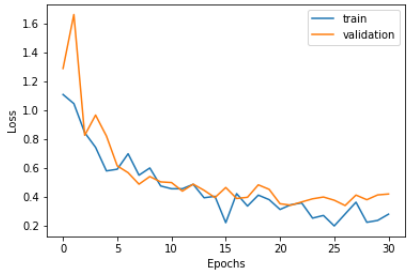

Fig. 14. Training and validation loss with Data Augmentations + Dropout.

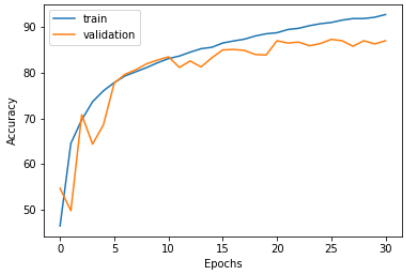

Fig. 15. Training and validation accuracy with Data Augmentations + Dropout.

Question 4: Use pretrained networks

a)

We instantiate the ImageNet-pretrained version of the `VGG_11_bn` model as the backbone of our network. We then remove its last average pooling and fully connected layers, and replace them with two fully connected layers of size 256 and 10 (number of classes) respectively. We also add a BatchNorm and ReLU layer between these two fully connected layers.

For training, we first ensure that the data is normalized with the correct mean and variance (as used in the original training of the `VGG_11_bn` network) in the data loader. Then, we train only the newly added layers, while disabling gradient updates for the other parameters of the model. In particular, we set `fine_tune = True` and `pretrained = True`, append the parameters of the new layers to `params_to_update` and set `requires_grad = False` for all other parameters. We also make use of early stopping as implemented in **Q2(b)** with `early_stopping_tolerance = 5`. Fig. 16 and Fig. 17 show the plots of training/validation loss and accuracy respectively.

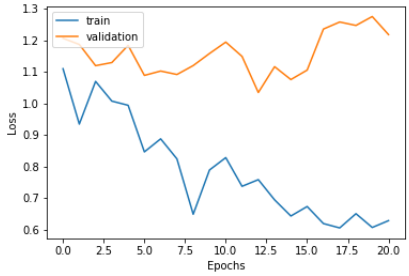

Fig. 16. Training and validation loss (VGG layers frozen).

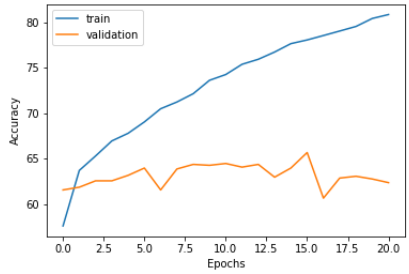

Fig. 17. Training and validation accuracy (VGG layers frozen).

We report a training accuracy of 78.71%, validation accuracy of 65.4% and a test accuracy of 63.8% with this model.

b)

To account for the domain shift between the ImageNet dataset and the CIFAR-10 dataset, we load the model with best weights obtained by training only the new FC layers from part **Q4(a)**, unfreeze the VGG layers and train all parameters of the network. For comparison, we also additionally train a baseline model from scratch without loading the pretrained ImageNet weights. We also make use of early stopping during the training of the models as implemented in **Q2(b)**, with `early_stopping_tolerance = 5`.

For the sake of brevity, the naming convention for our models is as follows:

frozen - model trained on only the new FC layers, freezing the VGG layers (flags `pretrained = True` and `fine_tune = True`).

fulltrain - model trained on all parameters, after first loading the best weights from the `frozen` model, and unfreezing the VGG layers (flags `pretrained = True` and `fine_tune = False`).

baseline - model trained from scratch without loading the pretrained ImageNet weights (flags `pretrained = False` and `fine_tune = False`).

Fig. 18 and Fig. 19 give a comparison of the training/validation loss and accuracy plots respectively, for the three models:

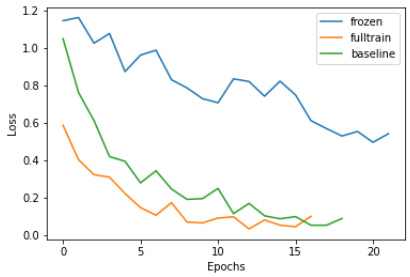

Fig. 18. Training and validation loss for the three models.

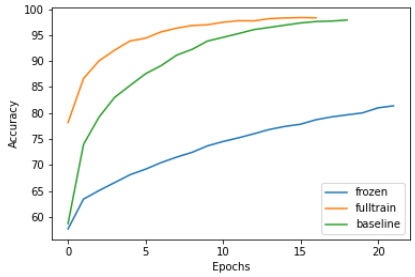

Fig. 19. Training and validation accuracy for the three models.

Tab. 2 summarizes the models and the training/validation/test accuracies from our experiments:

	frozen	**fulltrain**	**baseline**
pretrained	True	True	False
fine_tune	True	False	False
VGG layers	Not updated	Updated	Updated
FC layers	Updated	Updated	Updated
FC layer weights	-	Best weights from **frozen**	-
Training accuracy	78.71	97.77	96.47
Validation accuracy	65.4	89	87.2
Test accuracy	63.8	87.3	85.2

Tab. 2. Description of models with training/validation/test accuracies.

These are our observations:

- From our experiments, the best-performing model (based on validation and test accuracies) is the **fulltrain** model.
- The **fulltrain** model significantly improves over the **frozen** model, as observed from the validation and test accuracies, as it adapts to the domain of the CIFAR-10 dataset.
- The **baseline** model performs surprisingly well, considering that it doesn't make use of the pretrained ImageNet weights. It has comparable validation and test accuracy to the **fulltrain** model.