Foundations of Deep Learning

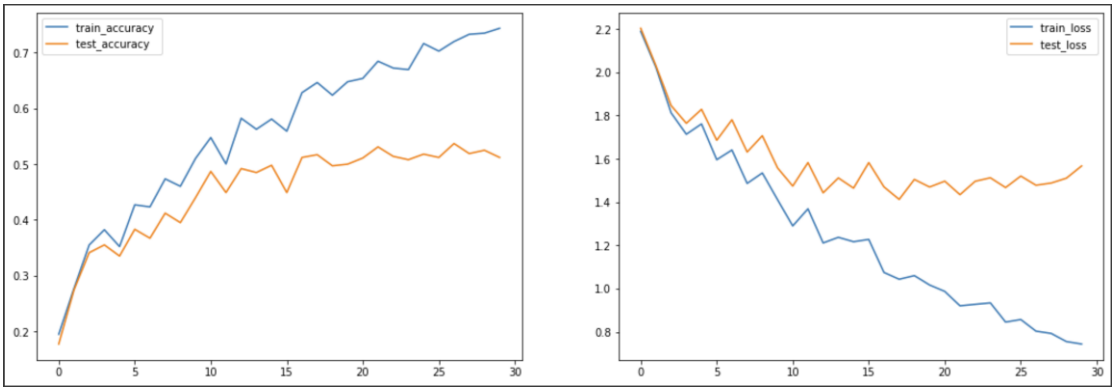
HW1 – Part 3 – Convolutional Neural Network

1. **Baseline:**

   We started our grid search with the same values we used for the feed forward network:
   Learning rate: [1e-1, 1e-2, 1e-3]
   Initialization standard deviation: [1e-5, 1e-3, 1e-1]
   Momentum coefficient: [0.9, 0.95, 0.99]
   We found the same best values: **0.01, 0.1 and 0.9,** respectively.
   To validate our result, we made another search with the following values:
   Learning: [0.01, 0.005]
   Initialization standard deviation: [0.1, 1]
   Momentum coefficient: [0.9, 0.95]
   However, the previous values remained the best configuration.

Results	Train	Test
Loss	0.6	1.59
Accuracy	75%	54% (best), 51% (final)

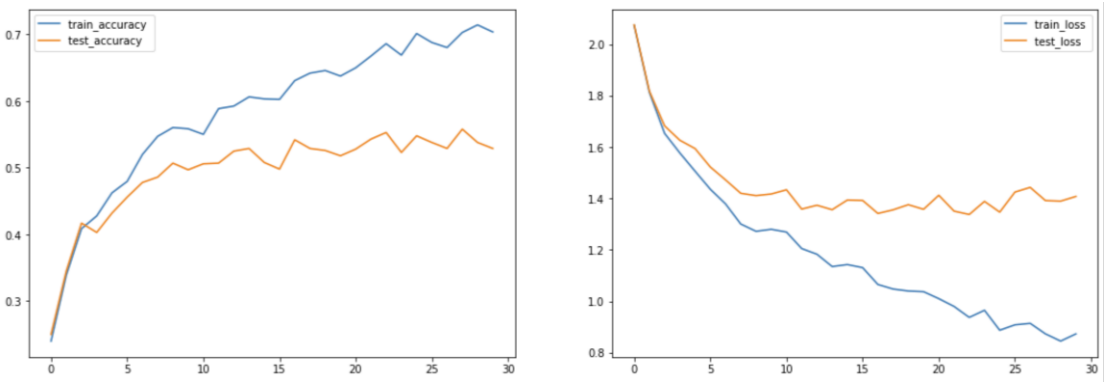
   We ran for 30 epochs – it was enough for the model to converge:



2. **Optimization:**

   Here we used the Adam optimizer. This led to faster convergence at first, and then a slower but steady decline in the training loss. We did not reach our previous scores on the training set, however we achieved better accuracy on the test set, indicating less overfitting and better generalization.

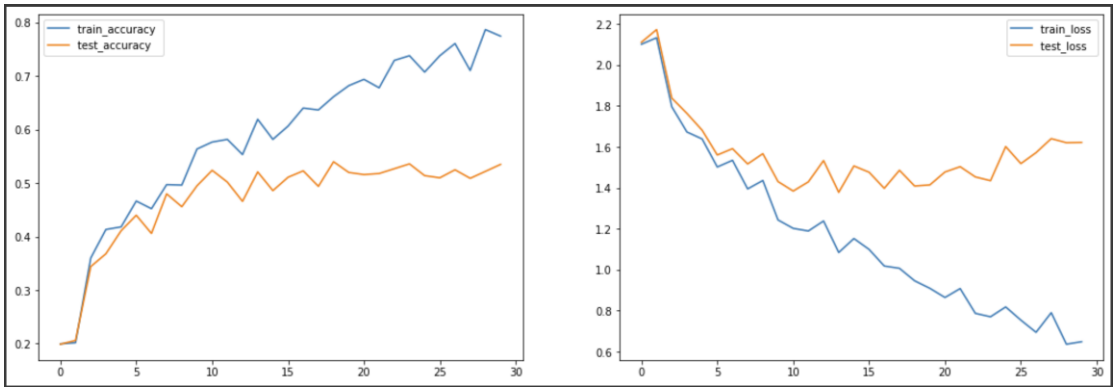
Results	Train	Test
Loss	0.85	1.4
Accuracy	70%	56.5% (best), 53% (final)



3. **Initialization:**

Using Xavier initialization, we observed slightly faster convergence time at the beginning compared to the baseline (albeit not as fast as Adam), and then a similar pattern. Although we achieved our best training accuracy so far, there was no improvement in the test accuracy.

Results	Train	Test
Loss	0.63	1.62
Accuracy	79%	53.7% (best), 53.5% (final)

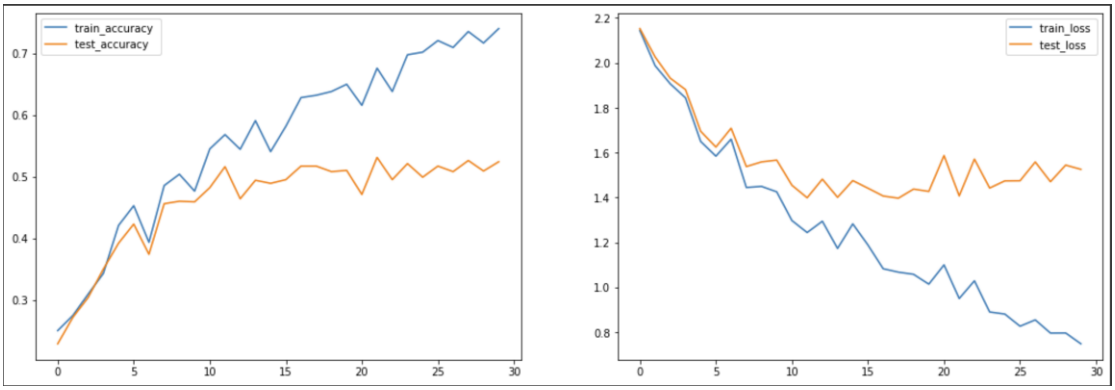


4. **Regularization:**

   a. **Weight Decay:**

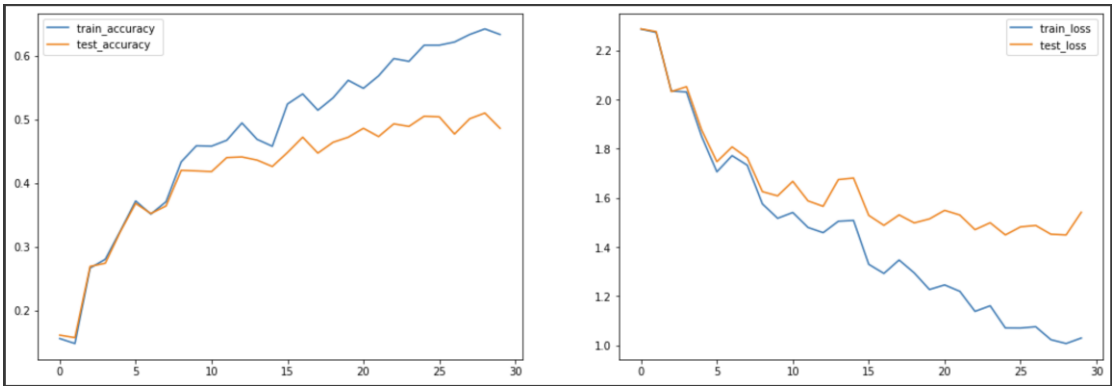
   We tried the following values: (1e-2, 1e-3, 1e-4, 1e-5). 1e-2 took too long to fit, and 1e-5 performed best. However, it has virtually no effect on the results, as can be seen in the graphs which are very similar to the baseline case.

Results	Train	Test
Loss	0.78	1.55
Accuracy	73.5%	53% (best), 52.9% (final)

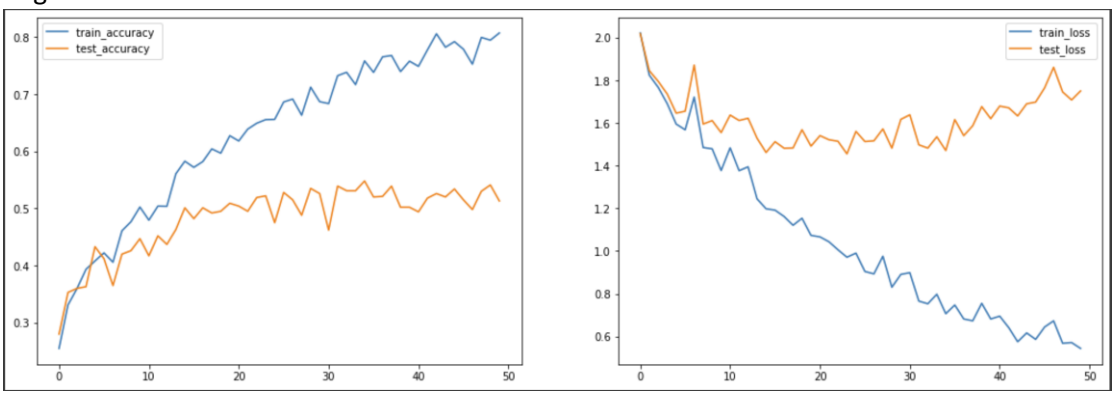


b. Dropout:

We added two dropout layers, one after each convolution layer and before the ReLU activation. As per the best practices we found online, we experimented with relatively low dropout rate at the first layer and a higher rate at the second layer. For rates of 0.1 and 0.2, we got the following graphs:



There is notably less overfit, however the test accuracy also decreased.
Hence, we made another attempt, this time with higher dropout rates of 0.15 and 0.3, and 50 epochs, to allow the model to better fit the scarce data. Indeed, we reached peak performance of 54.4% test accuracy at around 35 epochs, and then we began to overfit:



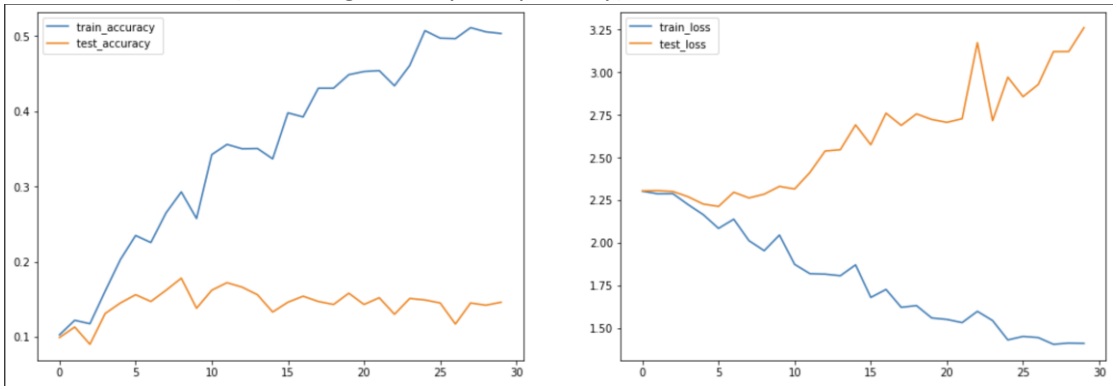
Results	Train	Test
Loss	0.58	1.79
Accuracy	80.5%	54.4% (best), 50.8% (final)

5. **PCA**

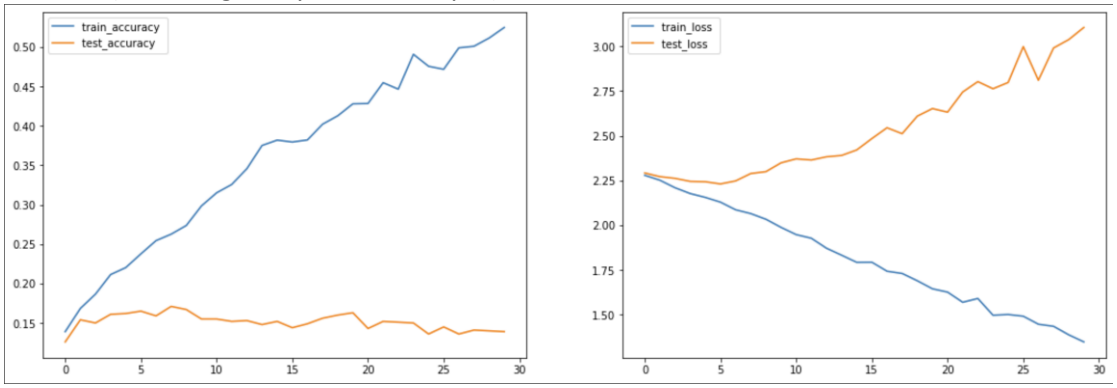
We chose 225 components based on the analysis performed on CIFAR10 here: Neural network on PCA extracted features | Towards Data Science. It suggests that 217 components explain over 95% of the variation and give good performance, and 225 is a close square.

Using PCA significantly shortens training time, as there are less parameters and the data is of much smaller dimensions. However, to do PCA we need to flatten the data, thus losing locality and other features of the image. Therefore, accuracy drops significantly:

Multi-channel PCA (flattening each layer separately):



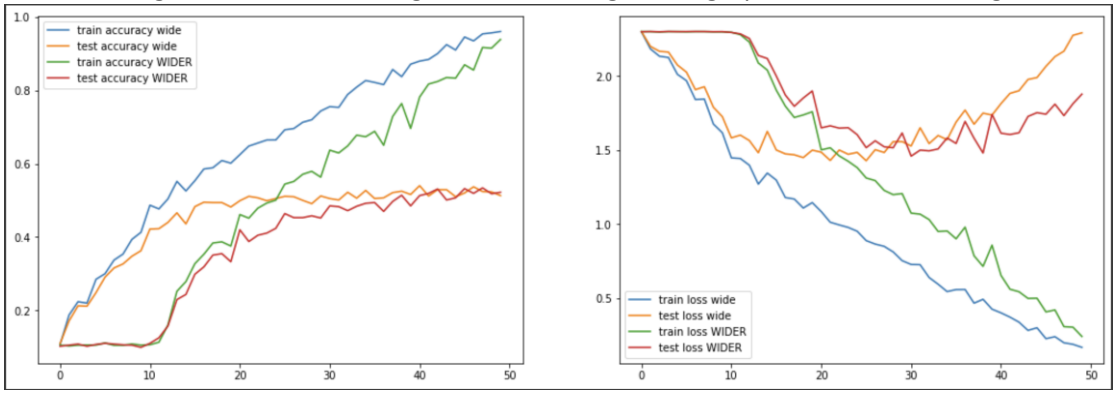
Flat PCA (flattening all layers of the input):



In both cases we peaked at around 18% accuracy on the test set. Results were similar when using 100 components (corresponding to the 99 components in the link above, which explain over 90% of the variation) and 400 components.

6. **Network width:**

In this section we experimented with different filter sizes. We ran for 50 epochs to allow for the enlarged models to converge, and according to the graphs it is indeed enough.



As could be expected, the wide model converges faster, but it also performs better. The WIDER model apparently requires more epochs or some tweaking to the configuration to work best, specifically the learning rate, as evidenced by the flat line during the first epochs. It would allow it to fit the training set better, but not necessarily generalize better.

Wide model:

Results	Train	Test
Loss	0.12	2.4
Accuracy	95.6%	54%

Wider model:

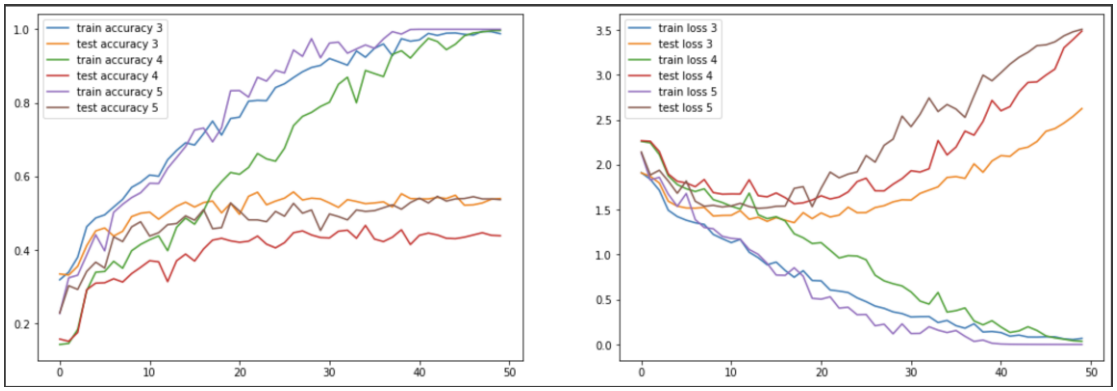
Results	Train	Test
Loss	0.2	1.88
Accuracy	94%	53.9%

7. **Network Depth:**

For this experiment we limited the number of max pooling layers to 3, to avoid reaching a 1x1 matrix too soon: one after the first convolution layer, one after the last, and one in the middle (for k=4, it's after the third layer). In addition, to make the class more generic, we decrease the filter size exponentially with each layer, ending in 32 channels instead of 16 like the baseline.

Finally, we used Adam optimizer for this comparison, as our default configuration for SGD did not work at all for k=4,5 and we wanted to compare "apples to apples".

The results, after 50 epochs:



There is a slight "anomaly" with k=4, maybe due to its non-uniform architecture w.r.t the max pooling layers, causing it to underperform.

Unsurprisingly, these larger models perfectly fit the training set, but there are no gains in test accuracy compared to the baseline – a clear case of overfitting, where the "shallowest" model is actually the most accurate. It seems that above all else, our model will benefit from more data.

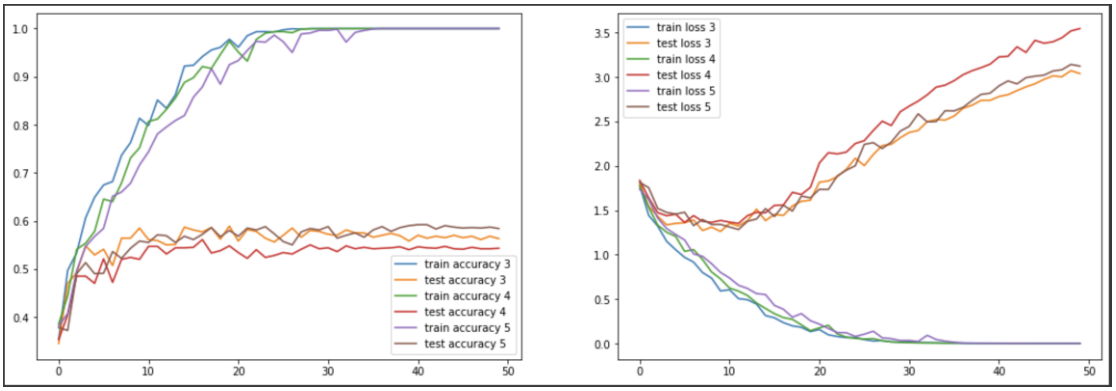
Model	**Metric**	**Train**	**Test**
K=3	Loss	0.07	2.62
	Accuracy	99.4%	55.8%
K=4	Loss	0.04	3.48
	Accuracy	99.7%	47.6%
K=5	Loss	0	3.5
	Accuracy	100%	54.6%

8. **Residual Connections:**

Our implementation is adapted from: [Machine-Learning-Collection/pytorch_resnet.py at master · aladdinpersson/Machine-Learning-Collection (github.com)]

To be able to add the input of some layer to the output of some later layer, we could not use shrinking filters like before, nor kernels of size 3.

We used Adam optimizer again, and got the following results:



Previously, we did not encounter the degradation problem of deep networks (our training accuracy did not drop), since our network was not deep enough for it to take effect (the paper deals with networks dozens of layers deep). However, the best performing network was the shallow one with 3 layers.

Using skip connections, we achieved our best results thus far, approaching 60% accuracy on the test set, and the network that achieved this was the deep one with 5 layers.

It is possible that by adding skip connections and allowing the deeper networks to learn identity mappings, they can build upon the learning done in the first layers and tend to only improve from that point. They also converged faster.

Model	**Metric**	**Train**	**Test**
K=3	Loss	0.002	3.04
	Accuracy	100%	58.9%
K=4	Loss	0.001	3.54
	Accuracy	100%	56.1%
K=5	Loss	0.001	3.12
	Accuracy	100%	59.2%