Reconnaissance des formes pour l'analyse et l'interprétation d'images

Homework 1-cd: Convolutional Neural Networks

Students: VU Anh Thu
LE Thi Minh Nguyet

In this lab, we will first implement a simple convolutional neural network and train it on the CIFAR-10 dataset. Then, we will test various techniques to enhance the network's learning process, including normalization, data augmentation, variants of SGD, dropout and batch normalization.

Part 1: Introduction to convolutional networks

(Q1)
- The output size is $\left(\frac{x+2p-k}{s}+1\right) \times \left(\frac{y+2p-k}{s}+1\right) \times 1$.
- The number of weights to learn is $k^2 z$.
- If we use a fully-connected layer instead of a convolutional layer to produce an output of the same size, the number of weights required will be $(xyz)^2 + xyz$.

Therefore, it can be seen that by using convolution layers, we significantly decrease the number of weights that need to be learned.

(Q2)
- **Advantages:**
  - As we have seen in (Q1), the first advantage of using convolution layers instead of fully-connected ones is the significant reduction in the number of weights to learn.
  - When processing images, it is important to preserve the spacial topology of the initial inputs throughout the network. Convolution filters can provide outputs with a great robustness to spacial shifting, scaling, and other forms of distortions. In contrast, fully-connected layers cannot ensure this because they completely ignore the topology of the input data.
- **Limits:** Although the number of parameters shrinks, the number of neurons remains considerably large, as the input image size is big and the number of convolution filters increases through the layers, potentially leading to memory issues.

(Q3) Although convolutions make the representations robust to spatial translations, they do not guarantee translational invariance. In fact, they are equivariant, which means that if an image is shifted, the feature maps change in a predictable manner, but they still capture the same features, just at different spatial locations. Spatial pooling helps to overcome this problem (at least for local shift) by combining nearby features into a single value, effectively summarizing local regions and ensuring that small shifts in the inputs will not result in drastically different outputs in the feature maps.

(Q4)
- In the case of the VGG16 network, we can use a portion of the network (from the first convolutional layer up to and including the last max-pooling layer). However, we cannot

use the remaining layers because they are fully-connected, meaning the input of the first fully-connected layer has a fixed size ($7 \times 7 \times 512 = 25088$ in this case). If we have a larger input image, the output of the last max pooling layer will have a size of $y \times y \times 512$, where $y > 7$, making it unable to be fed into the next layer.

- However, because VGG16 was trained to perform best with images of size $224 \times 224 \times 3$, when dealing with larger images, its prediction capability will not be as good as it is with the images of planned size due to differences in image resolutions.

(Q5) A fully-connected layer can be seen as a convolution with filters having the same size as the input, no padding and strike of 1. The number of filters corresponds to the number of output neurons in the layer.

(Q6) We have seen in (Q4) that if we keep VGG16 as it is, we can use all of its convolutional and max pooling layers, and the output of the last max pooling layer will have a size of $x \times x \times 512$, with $x > 7$. Now, if we replace the last three fully-connected layers of the network by their equivalent convolutions, then we can use the entire network, and their output sizes will be as follows:

- **First layer:** $(x - 7 + 1) \times (x - 7 + 1) \times 4096 = (x - 6) \times (x - 6) \times 4096$
- **Second layer:** $(x - 6) \times (x - 6) \times 4096$
- **Third layer:** $(x - 6) \times (x - 6) \times 1000$

Finally, the `SoftMax` function will be applied to the third dimension, and therefore the final output of the network will have a size of $(x - 6) \times (x - 6) \times 1000$.

Although this approach can help retain the spatial information of the input, which can be valuable for tasks requiring an understanding of local patterns, it shows no interest in achieving the intended task which is classifying the input image into 1000 classes.

(Q7) The *receptive field* size of the first two convolutional layers is equal to the kernel size of the convolution filters used in these layers. As the layers go deeper, the receptive fields increase in size after each pooling layer, allowing neurons to capture more global features from the input image.

Part 2: Training from scratch of the model

(Q8) Since the kernel size is $k = 5$, the necessary padding and strike values are $p = \frac{5-1}{2} = 2$ and $s = 1$, respectively.

(Q9) In the network that we are going to implement, no padding are needed and strike value is 2.

(Q10) Number of parameters and output size in each layer is:

Layer	Output Size	Weights
`conv1`	$32 \times 32 \times 32$	$5 \times 5 \times 3 \times 32$
`pool1`	$16 \times 16 \times 32$	0
`conv2`	$16 \times 16 \times 64$	$5 \times 5 \times 32 \times 64$
`pool2`	$8 \times 8 \times 64$	0
`conv3`	$8 \times 8 \times 64$	$5 \times 5 \times 64 \times 64$
`pool3`	$4 \times 4 \times 64$	0
`fc4`	1000	$1024 \times 1000 + 1000$
`fc5`	10	$1000 \times 10 + 10$

The majority of parameters are concentrated in the fully connected layers, while the convolutional layers have significantly fewer weights and the pooling layers introduce no new parameters, making them computationally efficient while still being able to learn the spatial hierarchies of the image.

(Q11) Total number of parameters is $1,191,010$.
The CIFAR-10 training dataset contains 50,000 images. Therefore, the model has significantly more parameters, meaning we are using a relatively small dataset to train a model with a considerably larger number of parameters. As a result, certain techniques may need to be employed to improve performance.

(Q12) The number of parameters of BoW is 0 and that of SVM is $p + 1$, where p corresponds to the number of patterns in the Bag of Words. As we can see, the number of parameters to learn in this CNN is significantly larger than that of BoW and SVM. However, BoW is shallow and handcrafted, limiting its ability to capture complex patterns in data, thus its performance is often inferior to that of a deep, automatic and end-to-end supervised CNN.

(Q14) During training, after calculating the loss, backpropagation is performed using `loss.backward()`, to compute the gradients with respect to the model's parameters. The optimizer then updates the model's weights by performing a step of gradient descent. During the evaluation phase, the loss is computed the same way as in the training phase, but no backpropagation or weight updates are performed.

(Q16)
- **Learning rate**: A high learning rate allows a faster convergence because weights are updated more significantly with each step. However, with a high learning rate, the model might be at risk of overshooting, causing the loss function to oscillate or even diverge instead of converging. Meanwhile, a low learning rate provides more stable convergence but may get stuck in a local minima.
- **Batch size**: A small batch size provides more frequent updates, leading to faster learning but it also introduces more variance in gradient updates, which may cause fluctuations and make training less stable. In contrast, a large batch size produces more accurate gradient estimates, leading to more stable updates, but learning is slower and requires more memory.

The following results are obtained from a single run and may vary across different runs. Moreover, they are obtained when we run the training for 25 epochs with a batch size of 128 and a learning rate of 0.1.

(Q17) At the beginning of the first epoch, the loss is 2.3004 in train and 1.9904 in test, while the top-1 accuracy is 16.4% in train and 21.1% in test. (The test results are not produced by the initial model but by the model after being trained for one epoch.) These non optimal results can be explained by the fact that the weights of the network are randomly initialized at the beginning, leading to an output far from the expected values and thus, high loss and low accuracy.

(Q18) As can be seen from the following figure, the model performs well on the training data, showing a consistent increase in accuracy and decrease in loss throughout the epochs. By the end of training, the model achieves near-perfect performance, reaching an accuracy of 99.98% and a loss of 0.0012. However, on the test data, although the model initially shows promising performance, achieving a loss of 0.901 and a top-1 accuracy of 70.21% by epoch 10, beyond this point, the accuracy begins to fluctuate around this level, and the test loss increases. This suggests the phenomenon of overfitting, as the model effectively learns the training data, capture noise and details specific to the training set, which make it struggles to generalize to new, unseen test data.

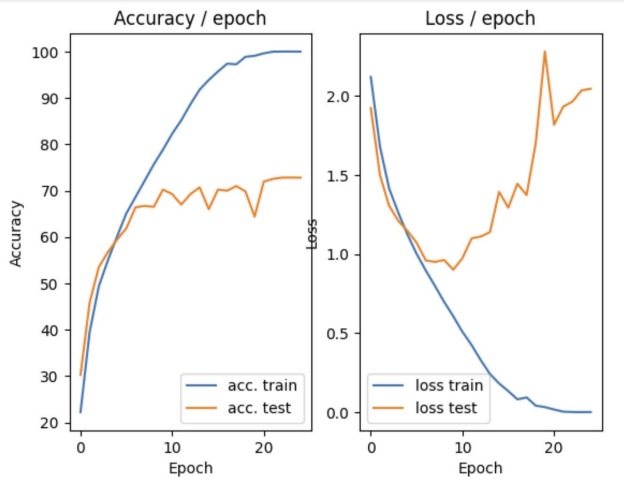
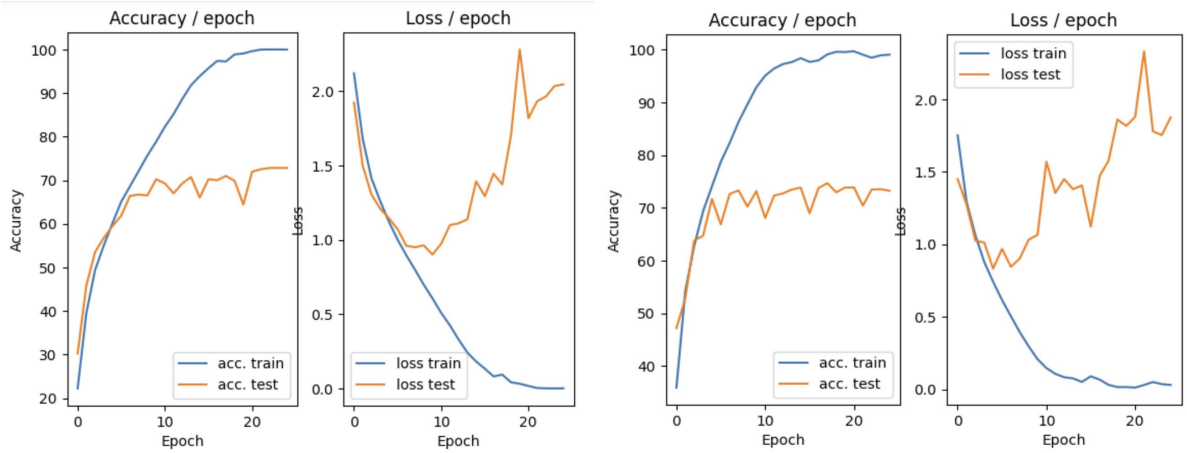


Figure 1: Learning curve for training with 25 epochs, a batch size of 128 and a learning rate of 0.1

Part 3: Results improvements

Before presenting the techniques that we attempted to implement and discussing their results, we would like to note that the following results are obtained from a single run and may vary across different runs. Therefore, conclusions regarding whether a technique can improve the model's performance or not are drawn from these single-run results and may not hold universally.

(Q19) After normalizing the data with the means and standard deviations of each RGB channel over the whole train, we obtained the following results:



(a) Original data **(b)** Normalised data

Figure 2: Learning curves for training with 25 epochs, a batch size of 128 and a learning rate of 0.1

Best result for normalized data is at epoch 5:

	Loss	Top 1 accuracy	Top 5 accuracy
Train set	0.7411	74.05%	98.44%
Test set	0.8319	71.67%	97.38%

Best result for unnormalized data is at epoch 10:

	Loss	Top 1 accuracy	Top 5 accuracy
Train set	0.6065	78.83%	98.99%
Test set	0.9010	70.21%	97.26%

As shown, data normalization can accelerate training convergence, as an optimal performance is reached by epoch 5 with normalized data compared to epoch 10 with unnormalized data. However, the improvement in generalization ability appears to be minimal.

(Q20) We only calculate the means and standard deviations on the training set and use them to standardize both the training and validation sets because we want to guarantee the consistency in data transformation. In fact, if we use one set of mean and standard deviation for the train set and another set for the test set, then after normalizing, the train and test set will have different distributions. This discrepancy can lead to incorrect predictions, as the model is trained on one distribution but tested on another. Moreover, we avoid using the means and standard deviations of the test set in this standardization process to prevent information leakage from the test data into the training process, thus allowing for an accurate evaluation of the model's true generalization ability.

(Q21) We attempted to implement ZCA normalisation and achieved the best result at epoch 7 as following:

	Loss	Top 1 accuracy	Top 5 accuracy
Train set	0.4720	83.63%	99.38%
Test set	0.7221	76.31%	98.51%

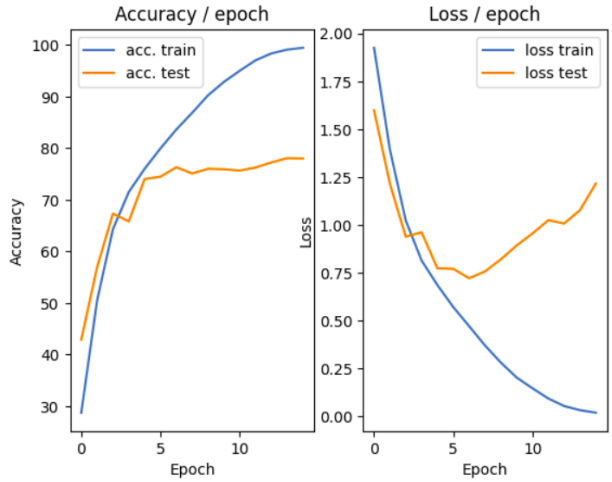
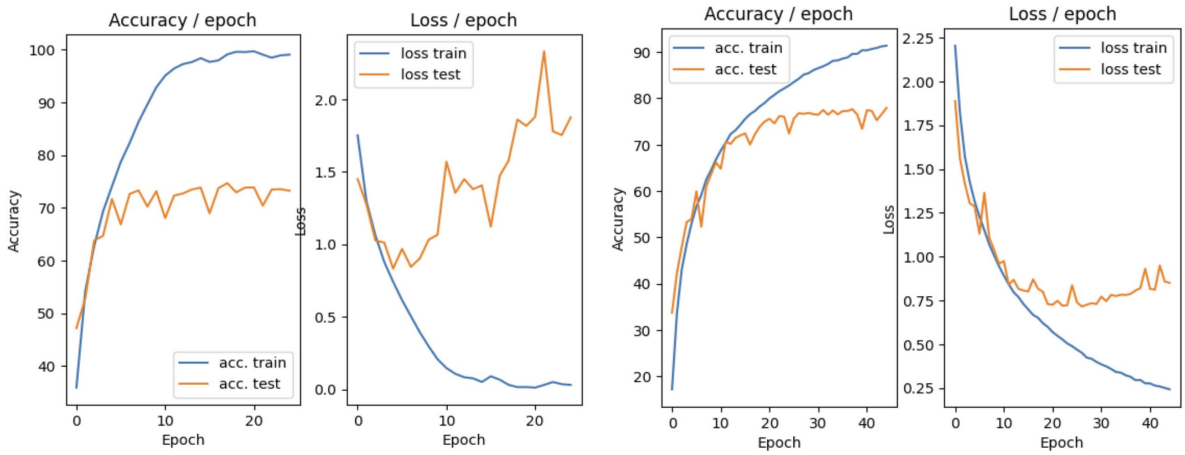


Figure 3: ZCA Normalisation (15 epochs, a batch size of 128 and a learning rate of 0.1)

Comparing these results with those obtained from training with both original and standard-normalized data, it is clear that applying ZCA normalization could lead to substantial improvements

in the network's performance. The model achieves the highest Top-1 accuracy on the test set at 76.31% and reduces the test loss to 0.7221. Moreover, ZCA normalization can also help speed up training convergence, as it converges after 20 epochs rather than 25 epochs.

(Q22) We artificially increased the training set size by randomly cropping the images to a size of 28×28 and randomly flipping them with a probability of 0.5. We also introduced to the test set a center crop of size 28×28 and obtained the following results:



(a) Normalised data (25 epochs)

(b) Augmented data (45 epochs)

Figure 4: Learning curves for training with a batch size of 128 and a learning rate of 0.1

Best result for augmented data is at epoch 21:

	Loss	Top 1 accuracy	Top 5 accuracy
Train set	0.5692	80.00%	99.04%
Test set	0.7411	74.05%	98.44%

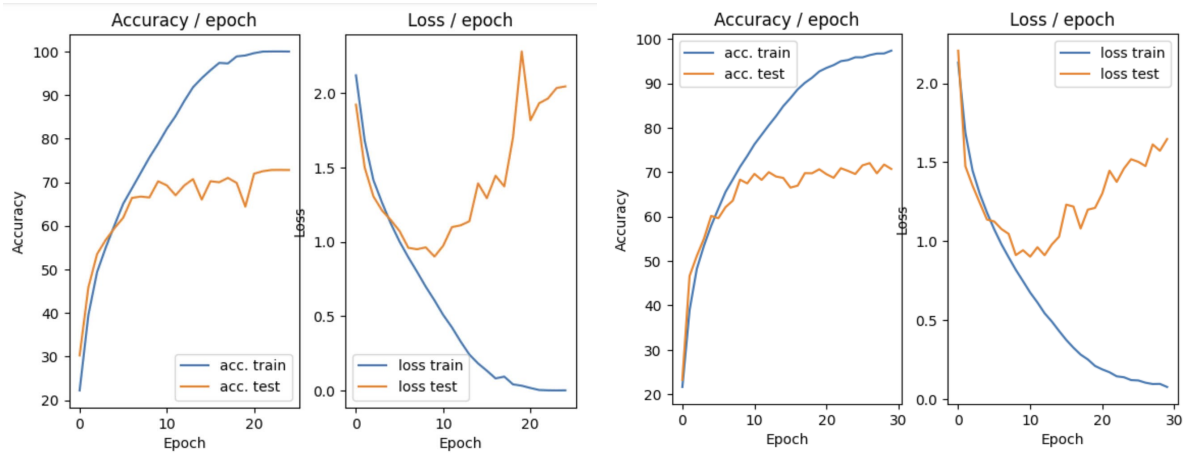
The results indicate that data augmentation can enhance the learning process, as the model outperforms both the original model and the model trained with standard-normalized data, achieving a test loss of 0.7411 and a top-1 accuracy of 74.05%. However, it can be seen that the ZCA normalization still yields the best performance to date. Moreover, while the augmented data improves performance, the training speed is significantly slower due to the increased dataset size resulting from the augmentation process.

(Q23) This horizontal symmetry approach is not usable on all types of images.
Flipping images horizontally can be useful for images which orientation does not significantly alter the class of the objects, such as: natural images (images of animals, landscapes, etc...), or objects that are symmetrical (like balls, cars, etc...).
However, this approach may not be suitable for images where direction matters (like arrows or traffic signs) as horizontal symmetry may confuse the model by introducing incorrect classes. Text or numbers might not perform well with horizontal flip sometimes as it can distort text and numerical information, making them unrecognizable. Also, facial recognition is not a suitable type as faces are typically not symmetrical in a way that allows flipping, thus applying horizontal flipping on images containing faces could lead to misclassification or loss of important features.

(Q24) Limitations of data augmentation through transformation are:

- Overfitting to augmented data: As the augmented data are generated from the dataset itself, these simple transformations might not introduce enough variety in the data and thus, the model might essentially memorize these variations without learning to generalize beyond them.
- Loss of Information: Some transformations may remove important contextual information, making it harder for the model to learn effectively.
- For certain cases, the augmenting dataset can lead to longer training times and computational costs without necessarily improving model performance.

(Q25) Here we tried out `transforms.RandomRotation(10)` (which randomly rotates the image by up to 10 degrees in either direction) for the train set and kept the test set the same and obtained the following results:



(a) Original model (25 epochs)

(b) `RandomRotation` (30 epochs)

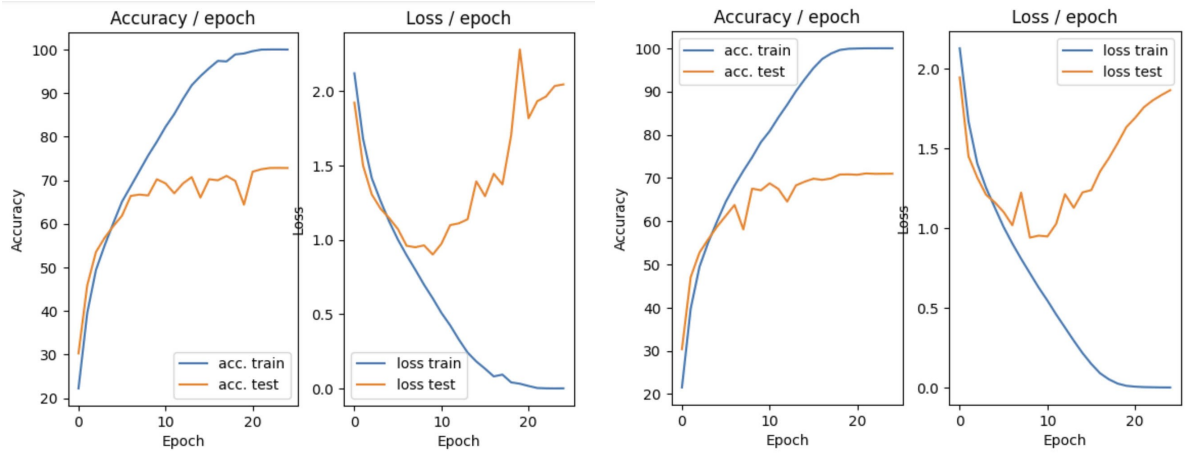
Figure 5: Learning curves for training a batch size of 128 and a learning rate of 0.1

The best result for this is achieved at epoch 11 as following:

	Loss	Top 1 accuracy	Top 5 accuracy
Train set	0.6752	76.29%	98.69%
Test set	0.9030	69.63%	97.44%

As we can see, the application of random rotation negatively impacts the model's performance. At epoch 11, the model achieves a training loss of 0.6752 and a top-1 accuracy of 76.29%, compared to 0.6065 and 78.83% for the original model. This decline suggests that the added variability from rotation may have introduced noise, making it harder for the model to learn effectively. Moreover, these results also reinforces the idea that sometimes augmenting dataset can lead to longer training times without necessarily improving model performance, as it took 30 epochs for the model trained on randomly rotated data to converge.

(Q26) We decreased the learning rate by 5% after every epoch and obtain the following results:



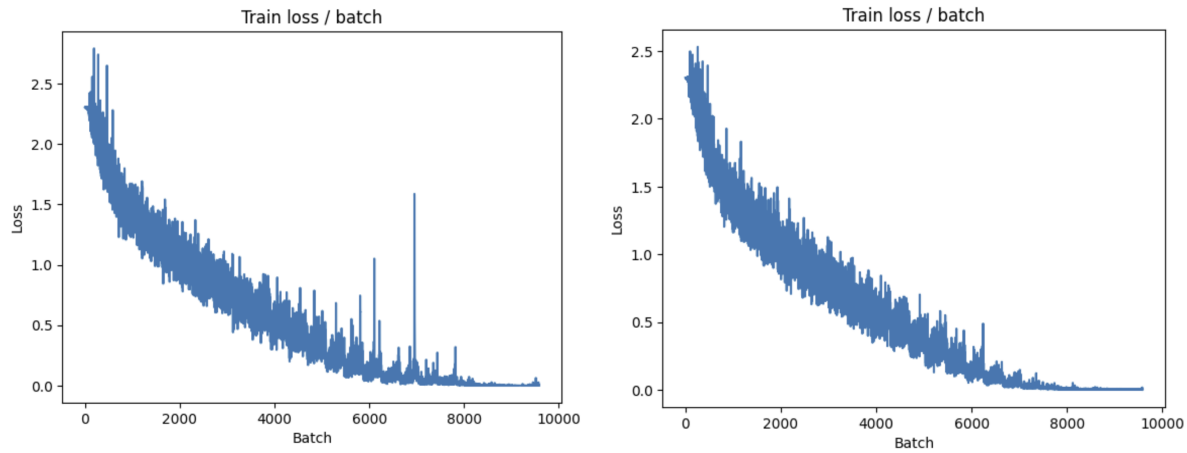
(a) Original model **(b)** Exponential decay of factor 0.95 of learning rate

Figure 6: Learning curves for training with 25 epochs, a batch size of 128 and a (initial) learning rate of 0.1

Best result for decreasing learning rate after every epoch is at epoch 9:

	Loss	Top 1 accuracy	Top 5 accuracy
Train set	0.7196	74.76%	98.54%
Test set	0.9427	67.53%	97.05%

It can be seen that, even when being compare to the original model, applying the learning rate schedule `ExponentialLR` with `gamma`$=0.95$ results in a decline in model performance, as both training and test accuracy are lower than those of the original model, indicating reduced effectiveness in learning. However, a positive outcome of the learning rate schedule is an improvement in learning stability as can be seen in the following figure.



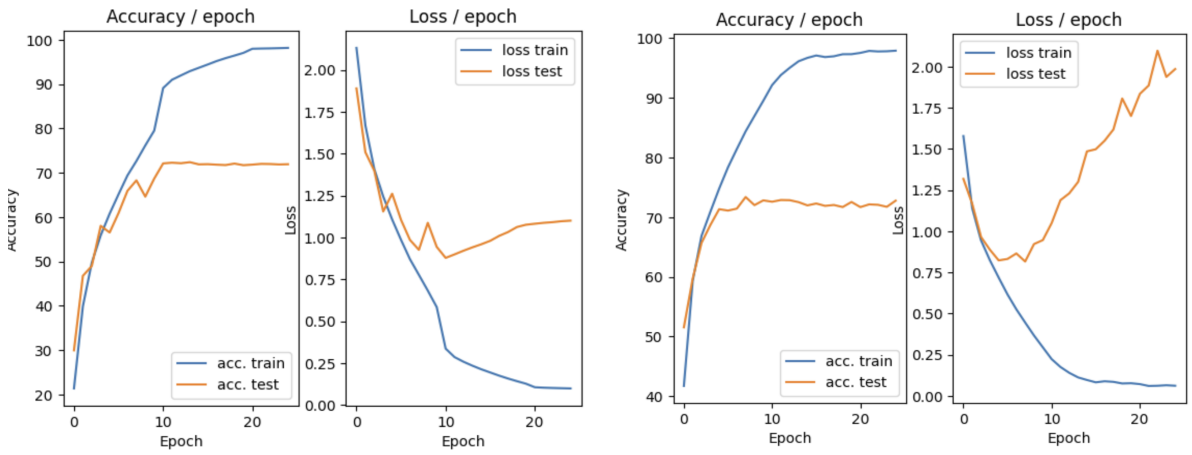
(a) Original model **(b)** Exponential decay of factor 0.95 of learning rate

Figure 7: Training loss through out the training

(Q27) Decreasing the learning rate during training can improve learning for some reasons:

- Refinement in later stages: As training progresses, the model's parameters get closer to the optimal values. Therefore, reducing the learning rate at these stages enables the model to make finer updates to the weights, leading to a stable convergence without overshooting.
- Reduce overfitting: By slowing down updates in later stages, the model resists fitting every detail in the training data. This reduction in fitting allows it to maintain a broader, more generalizable understanding of patterns in the data, which helps prevent overfitting and improves performance on new, unseen data.

(Q28) We tested the standard SGD algorithm with an initial learning rate of 0.1 and a learning rate schedule `StepLR` which decreases the learning rate by 90% every 10 epochs and another variants of SGD algorithm, `Adam` with a learning rate of 0.001. We obtained the following results:



(a) SGD with `StepLR` **(b)** `Adam`

Figure 8: Learning curves for training with 25 epochs and a batch size of 128

Best result of the standard SGD algorithm with a learning rate schedule `StepLR` is at epoch 11:

	Loss	Top 1 accuracy	Top 5 accuracy
Train set	0.3373	89.11%	99.52%
Test set	0.8165	73.40%	97.86%

Best result of the Adam algorithm is at epoch 8:

	Loss	Top 1 accuracy	Top 5 accuracy
Train set	0.4462	84.41%	99.73%
Test set	0.8788	72.11%	97.71%

As can be seen from the results, both techniques above can improve the network's learning process. Although the model's performance on the validation set is not as good as when trained with augmented data by a random crop and a random horizontal symmetry or with normalized data by ZCA, the results on training set are better. This can be explained by the fact that both SGD with step decay and Adam focus on efficiently adjusting learning rates to enhance convergence.

(Q29) We added a dropout layer with a probability of 0.65 between `fc4` and `fc5` and obtained the following results:

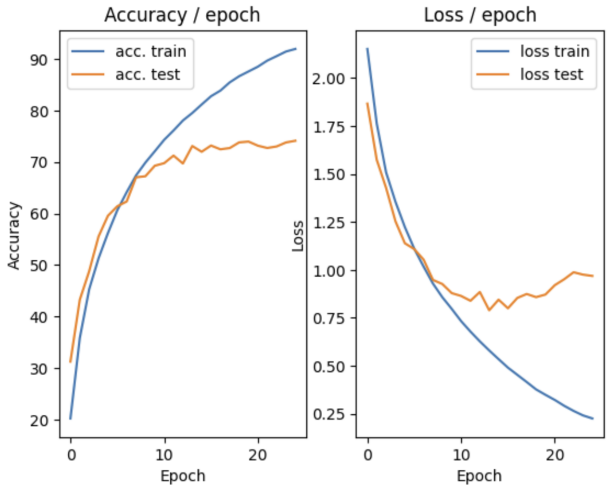


Figure 9: Dropout (25 epochs, a batch size of 128 and a learning rate of 0.1)

Best result for this method is at epoch 14:

	Loss	Top 1 accuracy	Top 5 accuracy
Train set	0.5802	79.49%	99.09%
Test set	0.7898	73.09%	97.66%

It can be seen that this technique can also lead to improvements in the network's performance. However, it hasn't caught up with ZCA whitening and data augmentation.

(Q30) Regularization is a technique that adds some constraints or penalties on the model's complexity. This technique can prevent the model from becoming too complex and fitting the training data too closely, thereby reducing overfitting.

(Q31)
- By randomly dropping neurons, dropout prevents the neurons of the network from specializing in identifying certain patterns, relying on others to detect different ones.
- Dropout introduces various variations in the input data, forcing the network to learn more general features that are useful across a range of samples rather than specific features unique to the training set, thus increasing its robustness.
- By turning off some neurons, dropout effectively reduces the model's capacity during training, which helps in preventing it from becoming overly complex and memorizing the data. This results in a simpler model that generalizes better to unseen data.

(Q32) A high dropout rate introduces strong regularization during training, which can lead to underfitting, as too much information is discarded, making the model fail to capture complex patterns in the data. Moreover, a higher dropout rate may slow down learning, as the model has fewer active neurons per iteration. Conversely, if a dropout rate is too low, the model may still overfit the training data, providing little benefit from the dropout layers.

(Q33) The behavior of the dropout layer differs significantly between the training and test phases, and there are also several different behaviors in the test phase. A typical one is that during evaluation phase, dropout is not applied, but the weights are scaled down by the dropout probability. This ensures consistent output scaling between training and testing, preventing incorrect predictions.

(Q34) We added a 2D batch normalization layer after each convolution layer and obtained the following results:

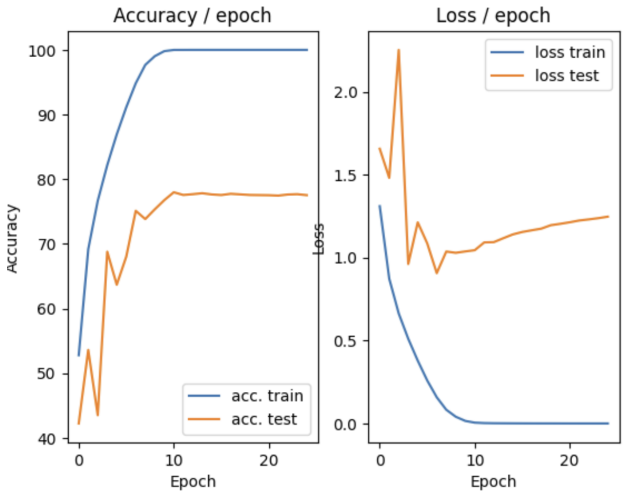


Figure 10: 2D batch normalization (25 epochs, a batch size of 128 and a learning rate of 0.1)

Best result for this method is at epoch 7:

	Loss	Top 1 accuracy	Top 5 accuracy
Train set	0.1582	94.84%	99.95%
Test set	0.9064	75.11%	97.92%

This final technique also facilitates the network's training process as the model achieves a test top 1-accuracy of 75.11% compared to 70.21% of the original model. Moreover, a noticeable point is that this technique helps the model to converge much faster than any other technique, as it reaches a near-perfect performance on the training set by just around epoch 10, whereas others require at least 20 epochs to achieve similar results.

Conclusions

In **part 3**, we have successfully implemented several techniques to improve the model's performance:

- ZCA whitening: This data pre-processing technique decorrelates pixels in the images, allowing the model to focus on learning learning meaningful nonlinear information rather than managing redundant correlations between adjacent pixels, thereby enabling faster and more effective learning.
- Data augmentation with a random crop and a random horizontal flipping: This technique exposes the model to a wider range of possible variations in the input data, enabling it to become more robust to variations and noise present in real-world scenarios, improving its performance when encountering new, unseen data.
- Add a learning rate schedule that decreases the learning rate by 90% every 10 epochs: This technique can provide smaller but more precise adjustments to the model's parameters in later stages of training when the model is close to optimal, allowing it to converge more stably and potentially leading to better performance on the validation set.

- Adam algorithm: This algorithm provides an adaptive learning rate for each parameter based on past gradients and past squared gradients. If past gradient indicate a good direction, the optimizer continues in that direction with a similar or slightly increased learning rate. Conversely, if past gradients suggest instability, the learning rate is reduced to stabilize training. Therefore, it can enhance the learning process in general.

- Dropout: This technique randomly drops neurons during training with a chosen probability. It acts as a type of regularization, penalizing any behaviors that focus on memorizing a specific pattern in the data because it forces the active neurons in the network to learn generalizable information to compensate for the lost information, and thereby reduces overfitting.

- Batch normalization: In our implementation, we added a 2D batch normalization layer after each convolution layer. In the training phase, for each feature map, it computes the mean and standard deviation across all pixels and all examples from the output of the previous layer, then uses these values to normalize the output before it is passed to the next layer. The mean and standard deviation are then used to update the "running" statistics which are used for normalization in the evaluation phase.
  This technique can make the training process more stable and effective because it prevents the information across the network from becoming too small or too large, therefore reducing the risk of exploding or vanishing gradients. Moreover, thanks to normalization, the network becomes less sensitive to differences in scale, which improves its generalization ability.