Linköping University | Department of Computer and Information Science
Project, 1.5 ECTS | TSBB17 – Visual Object Recognition and Detection
Autumn Term 2020

Project 1

– Visual Object Recognition Using Convolutional Neural Networks

Lukas Borggren
Viktor Gustafsson
Gustav Wahlquist

Supervisor: Gustav Häger
Examiners: Per-Erik Forssén and Michael Felsberg

LINKÖPINGS UNIVERSITET

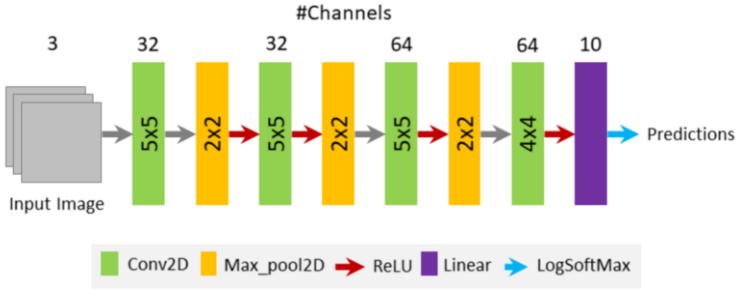
Linköping University
SE-581- 83 Linköping
013-28 10 00, ww.liu.se

Introduction

This project aims to evaluate varying architectures of convolutional neural networks (CNNs) and parameter configurations for visual object recognition on the CIFAR10 dataset. The source code can be found at https://gitlab.liu.se/lukbo262/tsbb17-project1.

Method

A baseline CNN architecture, CVLNet, was extended to create an initial model named GoalNet for the evaluation, as can be seen below.



Initially, GoalNet was evaluated and compared with a multi-stream network architecture consisting of two GoalNet streams – one for RGB and one for HSV. The top-performing architecture was used for further evaluation. Thereafter, the network was modified varying a single property at a time, i.e. two properties – e.g. pooling and dropout – were never *both* changed between two evaluation rounds. After each property evaluation the top-performing model was used for the subsequent evaluation step.

This method is by no means representative of an exhaustive evaluation. Firstly, the influence of the different properties is not evaluated in isolation as the network is altered incrementally. To make an isolated evaluation, each the property should be varied on the baseline GoalNet architecture. Secondly, far from all feasible combinations of properties and parameters are evaluated. This means that the final model configuration is not guaranteed to be the best one possible. All of these shortcomings are attributed to the limited timespan of the project.

To further evaluate how well the models would generalize, techniques such as cross validation could have been employed to ensure that the data splitting does not affect the model performance significantly. Again, this was not implemented due to the limited project scope.

Dataset

The dataset used was CIFAR10, which contains 60,000 color images of size 32x32 pixels in 10 different classes. Each image has a single class label and the labels are uniformly distributed over all classes, meaning that there are 6,000 images of every class. The dataset is divided so that there are 50,000 training images and 10,000 test images.

Training

The models were trained with early stopping with a minimum of 10 and a maximum of 50 training epochs. The early stopping consisted of a linear interpolation of the test losses from the 10 most recent epochs. If the slope of the resulting curve was greater than -0.0001, the training was terminated. During training, the Adam optimizer was used, initially with the learning rate 0.001. Furthermore, a linear learning rate scheduler was used to reduce the learning rate periodically. The learning rate was decreased with 20% in a stepwise fashion every tenth epoch. The training batch size was fixed to 256 images, while the testing batch size is set to 128 images.

Evaluation

Each of the network configurations were evaluated on the training and test data after each epoch. The cross-entropy loss and accuracy were computed for the training and test data separately. After each epoch, AUROC was also calculated on the test images for each class in a one-vs.-rest fashion to ensure that the classification performance was similar for all classes. Additionally, a micro-average AUROC over all classes was calculated to reflect the average performance. Favoring the ROC curve over the PR curve as a metric basis is motivated by that the images are balanced between each class. The training time for each model configures was also measured.

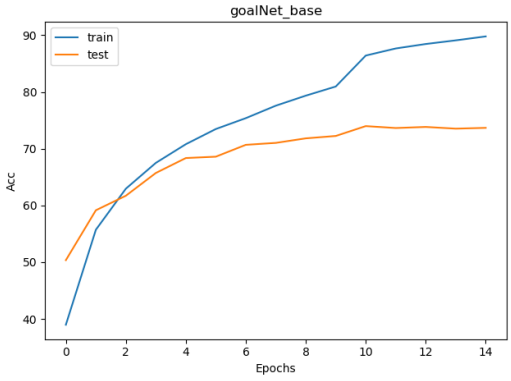
Results

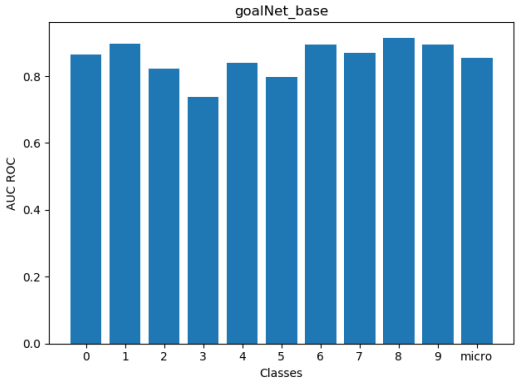
In this section, a short description for each alteration of the model is included together with the metrics that iteration of the model produced. We choose to present the most relevant metrics.

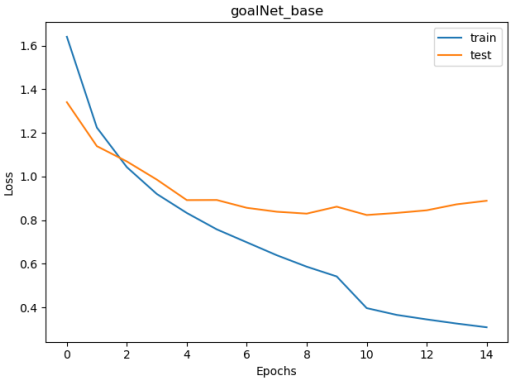
GoalNet

This is the baseline model for this project. The structure of the model is displayed in the method section.

Max train accuracy: 89.778 Max test accuracy: 73.98 Training time: 63.7 s
Early stop at epoch: 15 Average AUROC: 0.85

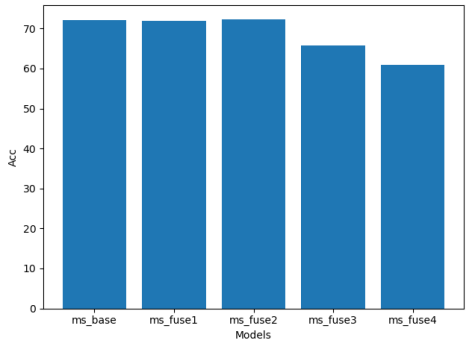






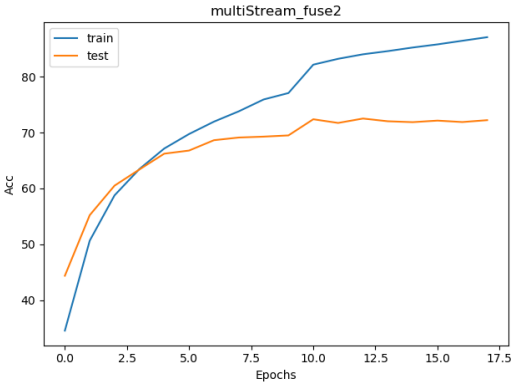
Multi-Stream GoalNet

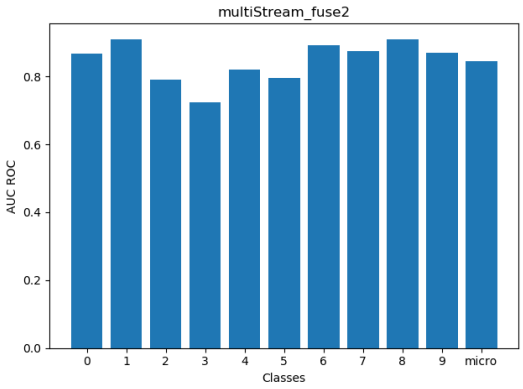
Fusing before the fully connected layer was done by first flattening the two streams and then concatenating the resulting one-dimensional tensors. Fusing streams before convolutional layers was done by concatenating the two separate output tensors over the channel dimension. An alternative to this could have been to concatenate over the image size dimensions, but we didn't see this as intuitively logical as this would equate to constructing an image where the first half is in RGB and the second in HSV.

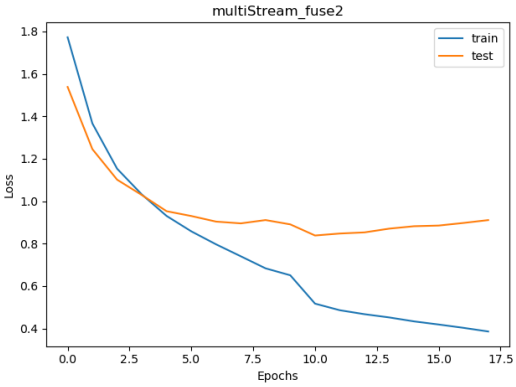


The variation *"ms_fuse2"* performed best and the metrics for that iteration of the model is displayed below.

Max train accuracy: 87.078 Max test accuracy: 72.52 Training time: 204.8 s
Early stop at epoch: 18 Average AUROC: 0.85





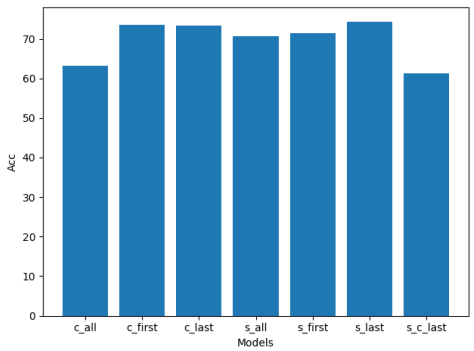


The multi-stream architecture has lower accuracy compared to Goalnet. Our hypothesis is this is that HSV and RGB are two different representations of essentially the same dimensions. The HSV representation probably has some advantages over RGB, for instance by isolating color to one dimension. However, this classification task might not benefit from the HSV representation, e.g. because there is no need to linearly separate colors i.e. color does not correlate well with different classes. Furthermore, if additional linear layers would be added those could probably learn the HSV transformation from RGB if needed, as the transformation is mostly linear except from a max operation.

Our reasoning for why performance suffers from fusing on earlier layers is similar to the reasoning above. The model has potential to benefit from HSV because of the difference in image representation from RGB, and thusly the architecture should intrinsically be able to process the color spaces differently carrying out e.g. convolutions. However, the earlier the data streams are fused, the less the network is able to handle them separately, as some layers will receive inputs of both data representations, e.g convolutional layers with less output channels than input channels.

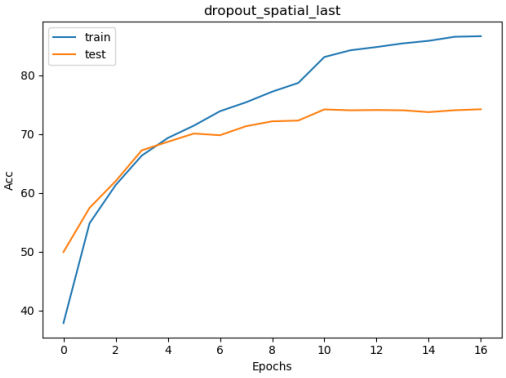
Dropout

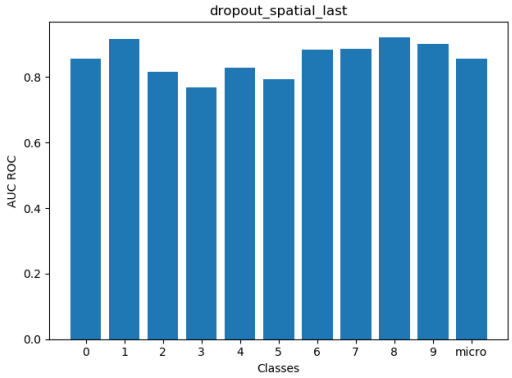
7 different configurations of dropout layers were tested. The three first consisted of applying channel-wise dropout layers in various parts of the architecture. In the first variation, labeled as *c_all* in the following histogram, channel-wise dropout layers were placed after each pooling layer. In the second variation *c_first*, channel-wise dropout layers were placed after the first pooling layer and in the third variation *c_last*, channel-wise dropout layers were placed after the last pooling layer. The next three variations *s_all*, *s_first* and *s_last* were distributed like the first three, but with spatial dropout layers instead. However, in the variation *s_last* the dropout is made at the last layer before the fully connected layers, unlike the *c_last* were it is made after the last pooling layer. The last variation *s_c_last* that was tested consisted of all the dropout layers from *c_all* together with the dropout layers from *s_last*.

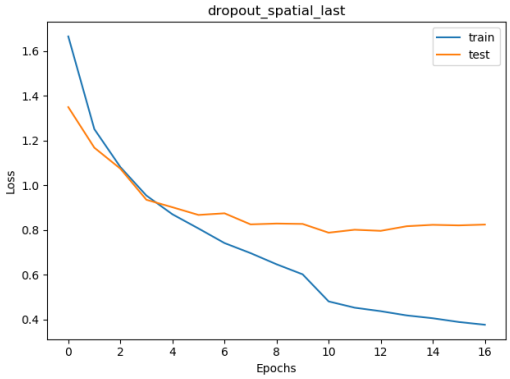


The variation *s_last* performed best and improved the model and the metrics for that iteration of the model is displayed below.

Max train accuracy: 86.688 Max test accuracy: 74.25 Training time: 74,3s
Early stop at epoch: 17 Average AUROC: 0,857







Dropout works as a form of regularization, preventing the model from overfitting on the training data by producing a model that is more robust to variations in data by randomly dropping activations. As random neurons are dropped, the ability for the model to fit the data reduces. For this reason, dropout can reduce performance if the model is shallow relative to the classification task and dataset. This is probably the reason for why dropout on all layers reduces the performance.
The regularization on the last layer seems to be enough to increase the generalized performance by preventing overfitting, whilst still not decreasing the model's capacity to fit the data too much to decrease performance.

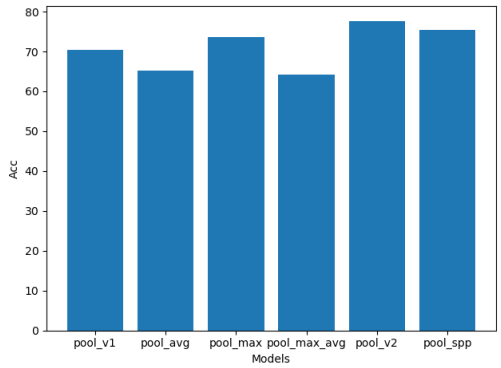
Pooling

Network Adjustments

First, max, average and spatial pooling was compared. The spatial pooling was achieved by using three adaptive pooling layers of sizes 1, 2, and 4 followed by concatenating the output tensor. The initial GoalNet structure results in a 3x3 pixels final image size before the fully connected layer, which a spatial pyramid can't effectively divide in size. Therefore, we choose to slightly increase the final image size by removing the last pooling layer whilst testing the spatial pyramid pooling. Also, a third variation was evaluated denoted *pool_max_avg* which uses two average poolings first and then a max pooling layer as the third and final pooling.

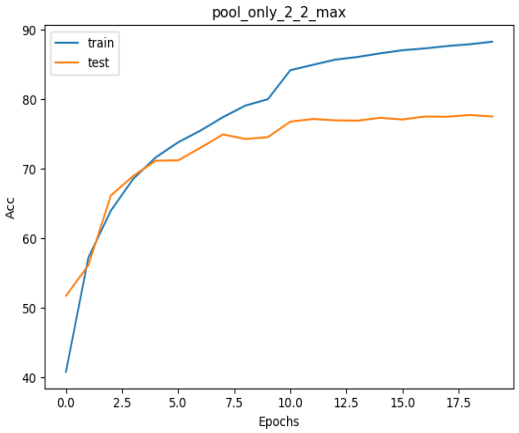
We used two variants to vary the sizes of the pooling. We denote the structure as (s1, x1) (s2, x2) (s3, x3) where *s* is the pooling window size and *x* is the stride for the window. For these variants we used max pooling, as it performed better than average pooling in the previous experiment. As these two variations changes the final size of the image, the fully connected layer had to be adapted in size. The number of neurons in the fully connected layers was changed to 576 for v1 and 1600 for v2.

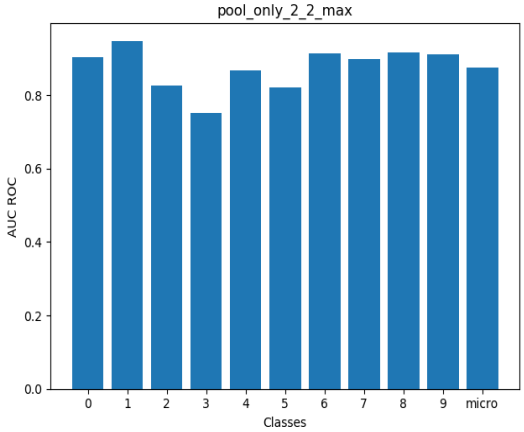
v1: (8,8) (2,1) (2,1)
v2: (2,2) (2,2) (2,2)

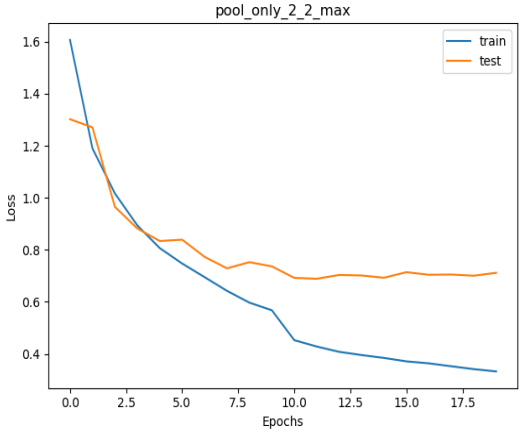


The variation *pool_v2* performed best and the metrics for that iteration of the model is displayed below.

Max train accuracy: 88.306 Max test accuracy: 77.76 Training time: 86.8 s
Early stop at epoch: 20 Average AUROC: 0.857







Evidently, reducing the pooling window size from (4,4) to (2,2) on the first pooling layer improved the performance of the model. This could be due to the fact that it keeps more of the original data when forwarding in the network. Potentially, this is beneficial here as the original image inputs are of relatively low resolution. If the resolution would be higher, reducing data with pooling could be more beneficial and result in a model with higher robustness. This since fine-grain variations in the image will not affect the result as much and the model gets faster with a reduced amount of data in the rest of the network.

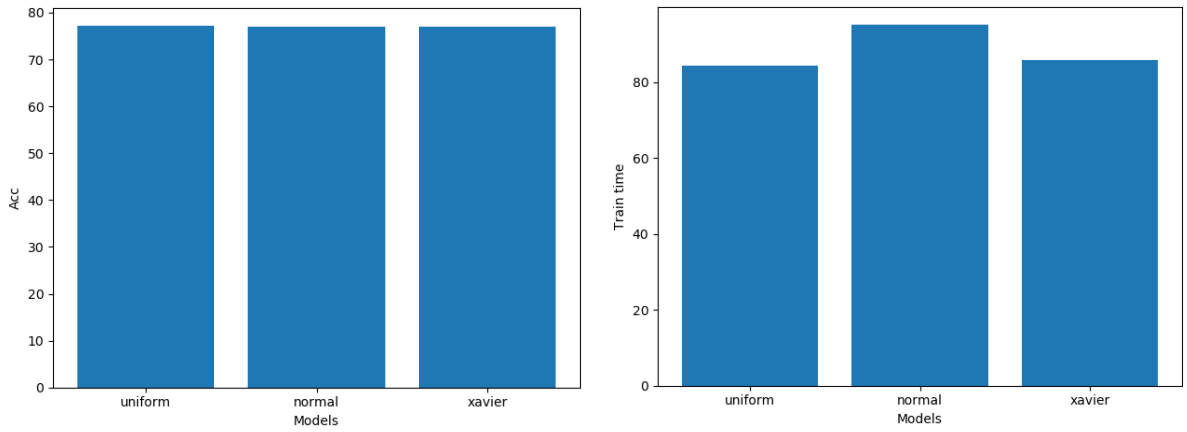
The effect of introducing a spatial pyramid pooling layer seems to be marginal. The performance is slightly better than the baseline but since we had to do other changes in the network in this iteration, it is hard to know exactly what lead to the improvement. However, the biggest advantage of using spatial pyramid pooling layers is that it can handle inputs of varying size and still give a set output, which can be very beneficial when working with dataset consisting of data with varying size.

Hyperparameters

Weight Initialization

The default weight initialization used in previous evaluations is based on a uniform distribution with a range based on the number of parameters n in each network layer: $[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$. Testing to draw initial

weights from a normal distribution with mean 0 and standard deviation $\frac{1}{6\sqrt{n}}$ and using Xavier initialization respectively, the results can be seen below.

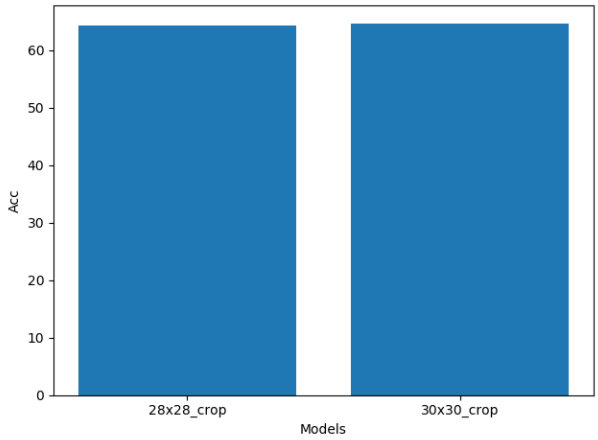


Evidently, the model performs very similar for all weight initailizations methods and the training time does not change substantially. This means that the model converges to reasonable local minima regardless of which of the three initialization methods is used.

Input Image Size

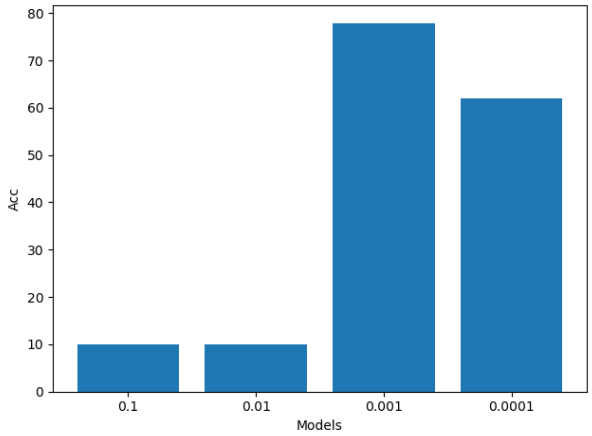
Network Adjustments

We used the randomized crop function. Only a slight crop was used as larger one would have necessitated removal of pooling layers to retain a reasonable amount of data. We adjusted the fully connected layer by removing nodes to accommodate the smaller final image size.



Both crops performed worse than the baseline model. We assume this is due to the fact that the datasets images are already quite small and tightly cropped, meaning a crop removes important information in the image. A crop always removes information in the image, and as long as the crop is not ‘smart’ or trained, it might remove parts of the item to classify by chance, i.e. important information.

Learning Rate



The higher learning rates results in a significantly worse performance. As the gradient direction is only valid for an area around the current weights, a too large step might not result in proper convergence. This is especially true if the loss function is highly irregular. This is since a step size too large means the weight update might not actually always decrease the loss, as the gradient direction changes continuously for all value updates of the weights. This might result in oscillation around local optimums or missing local optimums with large hessians around the optimum.

Conclusions

The final model which was incrementally built was the original GoalNet + a spatial dropout layer before the fully connected layers + changing the first pooling layer's window size from (4,4) to (2,2).
The rest of the tested alterations either performed worse or were more time intensive.