Venice Boat Classification

Andrea Fioraldi 1692419

December 23, 2018

The Problem

We want to construct a model to classify images that contain boats of Venice or does not contain boats.
We used the *ARGOS* training set to train several neural networks and then a preprocessed version of the test set for the validation.
The classes are 24:

`Alilaguna`, `Ambulanza`, `Barchino`, `Cacciapesca`, `Caorlina`, `Gondola`, `Lanciafino10m`, `Lanciafino10mBianca`, `Lanciafino10mMarrone`, `Lanciamaggioredi10mBianca`, `Lanciamaggioredi10mMarrone`, `Motobarca`, `Motopontonerettangolare`, `MotoscafoACTV`, `Mototopo`, `Patanella`, `Polizia`, `Raccoltarifiuti`, `Sandoloaremi`, `Sanpierota`, `Topa`, `VaporettoACTV`, `VigilidelFuoco`, `Water`

The size of each image is 800x240.
We chose *keras* as the library for such networks.

Preprocessing

From the training set we removed all the images with a class that is not present in the training set:

- `Mototopo corto`;

- `Snapshot Barca Multipla`;
- `Snapshot Barca Parziale`;

We renamed the `Snapshot Acqua` class of the test set to `Water` to match the train set. The training dataset directory structure is compatible with keras but this is not true for the test set, so we created a directory compatible with the automated images preprocessing functions of keras. We added also empty folders in order to indicate that a class has no images in the test set.
The directory structure is the following:

- `sc5_test`
  - `class0`
    - `image0_class0.jpeg`
    - ...
    - `imageN_class0.jpeg`
  - ...
  - `class24`
    - `image0_class24.jpeg`
    - ...
    - `imageN_class24.jpeg`

We exploited this directory structure in order to use the *ImageDataGenerator* class of keras for automated image loading and preprocessing. All the images are loaded in color mode RGB with batch size 32. The target size of ImageDataGenerator is one of the parameters that we changed in the experiments. When this parameter is present, the images are stretched to the target size.
After all, in the training set we have 4774 images and in the test set 1672 images.

Classification

The metrics that we used for the classifications are the accuracy and the loss function. The chosen loss function is the Categorical Crossentropy, a logarithmic function that increases when the prediction is far from the label, so it must be kept as lower as possible. We trained the networks using a GeForce GTX 1050 with 1.5 GHz clock and 4 Gb of memory.

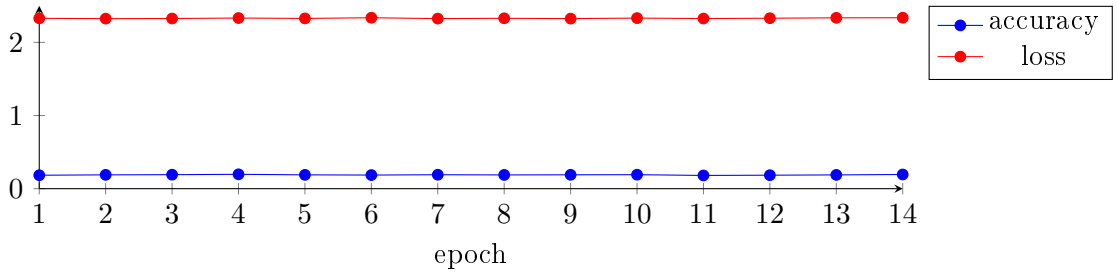
Well-Known Networks

We used different neural networks for this classification problem.
In the beginning, we chose two well-known networks, LeNet and AlexNet.

LeNet

Input shape: 320x320.
Parameters:

- Total params: 55,251,196;
- Trainable params: 55,251,196;
- Non-trainable params: 0;



Training time: 17.750000 minutes.
Final epoch performance:

- Loss: 2.3370;
- Accuracy: 0.1940;

Test set performance:

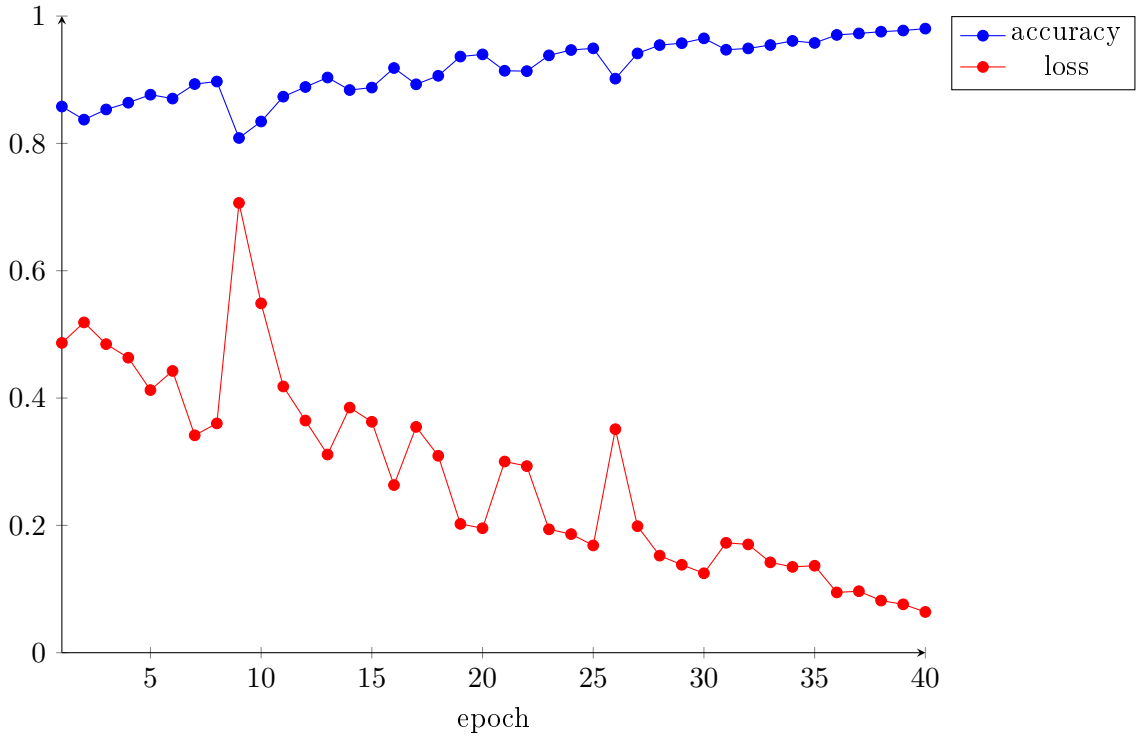
- Loss: 2.202084;
- Accuracy: 0.192909;

In each epoch the gain of this net is very poor we stopped the training after 14 epochs. The performance is very bad so we do not test other configurations with LeNet and switched to AlexNet.

AlexNet

Input shape: 320x320.
Parameters:

- Total params: 43,832,416;
- Trainable params: 43,811,280;
- Non-trainable params: 21,136;



Training time: 31.416667 minutes.
Final epoch performance:

- Loss: 0.0640;
- Accuracy: 0.9803;

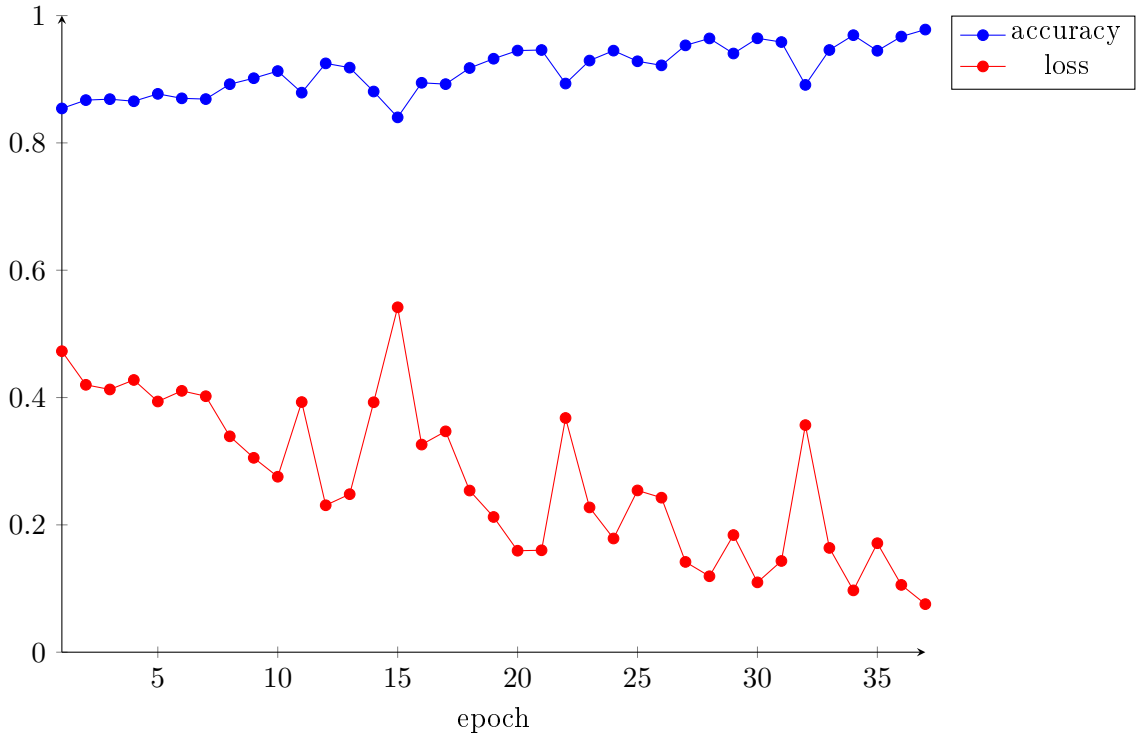
Test set performance:

- Loss: 1.190347;
- Accuracy: 0.810096;

AlexNet seems a good model. The performance on the test set are not as good as the performance on the train set but this is obvious.
81% of accuracy is great for a network trained on such small dataset. Loss is quite high but tolerable.
We also tried to enlarge the input shape.

Input shape: 420x420. Parameters:

- Total params: 78,435,424;
- Trainable params: 78,414,288;
- Non-trainable params: 21,136;



Training time: 72.516667 minutes.
Final epoch performance:

- Loss: 0.0755;
- Accuracy: 0.9780;

Test set performance:

- Loss: 0.866917;
- Accuracy: 0.839543;

As you can see in the graph, using such input size the accuracy and the loss is more unstable in time. Probably we stopped too early the training and the network needs a greater number of epoch to converge. Despite this, the result is better than the first run.

Why not other known networks?

We tried also the Inception networks from Google but they are too complex for our hardware. The GPU memory goes full almost immediately.

AndreaNet

After using the known networks we decided to create a custom network called AndreaNet. As seen before, LeNet is a bad model on this dataset so we started from it with the aim to create a net with performance comparable with AlexNet.
We created 3 versions of AndreaNet with different parameters tuning and in the end, it is a completely new neural network.

Version 1

This is the diff of the LeNet layers and AndreaNet v1 layers:

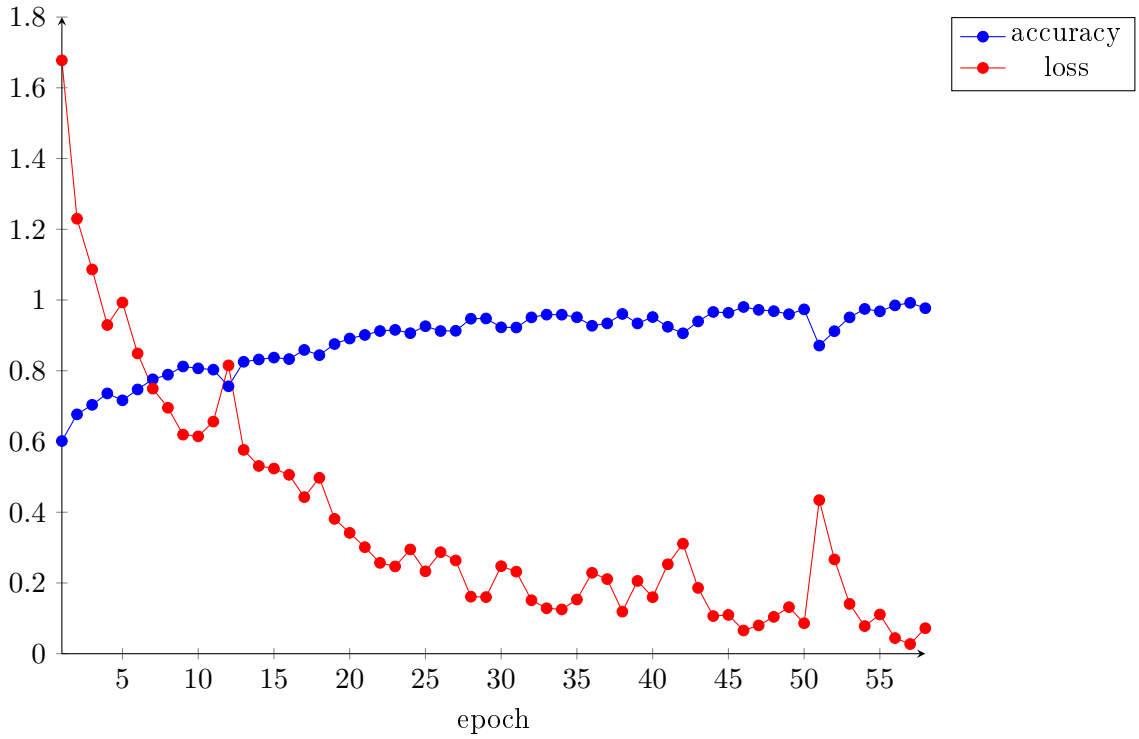
LeNet	**AndreaNet v1**
Convolutional 6 kernels 5x5	Convolutional 6 kernels 11x11
Average Pooling 2x2 stride 2x2	Average Pooling 2x2 stride 2x2
Convolutional 16 kernels 5x5	Convolutional 16 kernels 8x8
Average Pooling 2x2 stride 2x2	Average Pooling 2x2 stride 2x2
Convolutional 120 kernels 5x5	Convolutional 120 kernels 8x8
	Average Pooling 2x2 stride 2x2
	Convolutional 240 kernels 2x2
Fully connected, 84 units	Fully connected, 2048 units (dropout 0.5)
	Fully connected, 512 units
Fully connected, 24 units	Fully connected, 24 units

All of them uses the tanh activation function except for the last that uses softmax, as in LeNet.
As you can see in the diff we enlarged the size of kernels to test if this can be a tuning that allows the net to perform better. After all layers, we also added BatchNormalization, one of the advantages of AlexNet, and this helped us to create a deeper network that is,

generally, better. We also added a convolutional layer and a larger dense layer for the creation of a deeper network.
But why deeper is better? As explained in a clear way here (`https://www.quora.com/Why-are-deeper-neural-networks-better-than-wider-networks`) in theory 2 hidden layers are enough but a deeper networks needs less neorons to reach results like the 2 hidden layers network, so it is a way to increase the results quality without increasing too much the training an the convergence time.

Input shape: 320x320.
Parameters:

- Total params: 3,288,080;
- Trainable params: 3,282,196;
- Non-trainable params: 5,884;



Training time: 15.716667 minutes.
Final epoch performance:

- Loss: 0.0721;

- Accuracy: 0.9769;

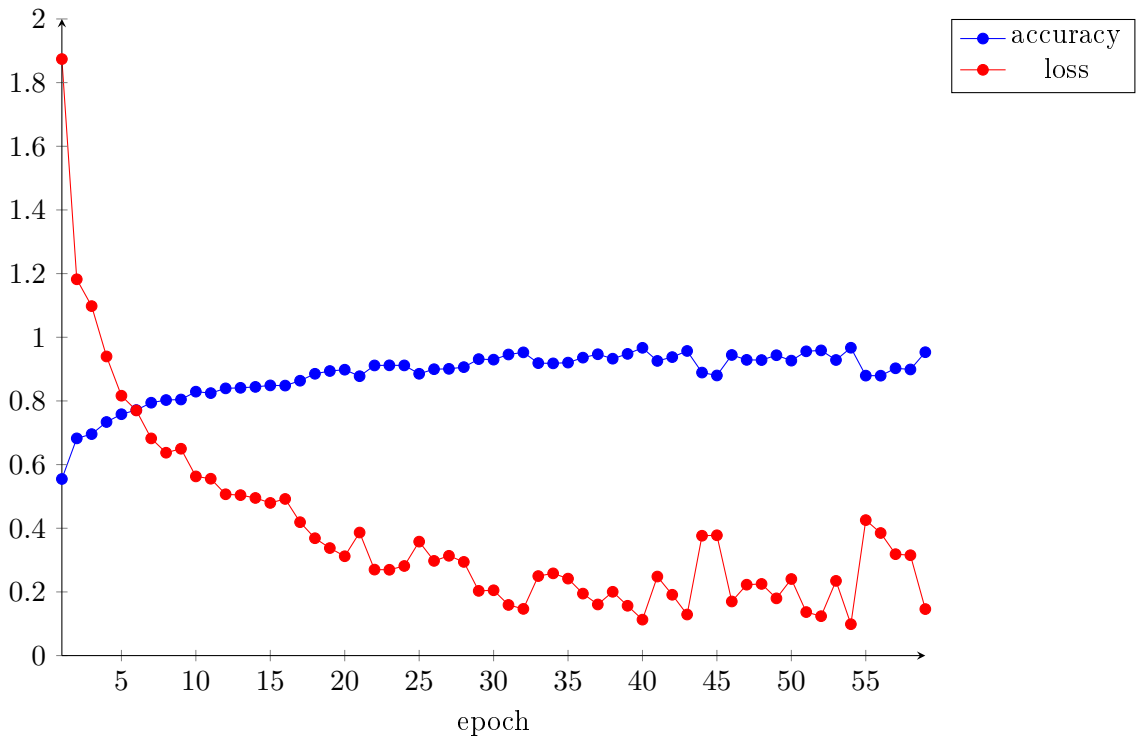
Test set performance:

- Loss: 1.466683;
- Accuracy: 0.767428;

The results are very good respect LeNet. We also tried to get better results enlarging the input shape. Input shape: 420x420.

Parameters:

- Total params: 5,745,680;
- Trainable params: 5,739,796;
- Non-trainable params: 5,884;



Training time: 66.366667 minutes.
Final epoch performance:

- Loss: 0.1461;

- Accuracy: 0.9531;

Test set performance:

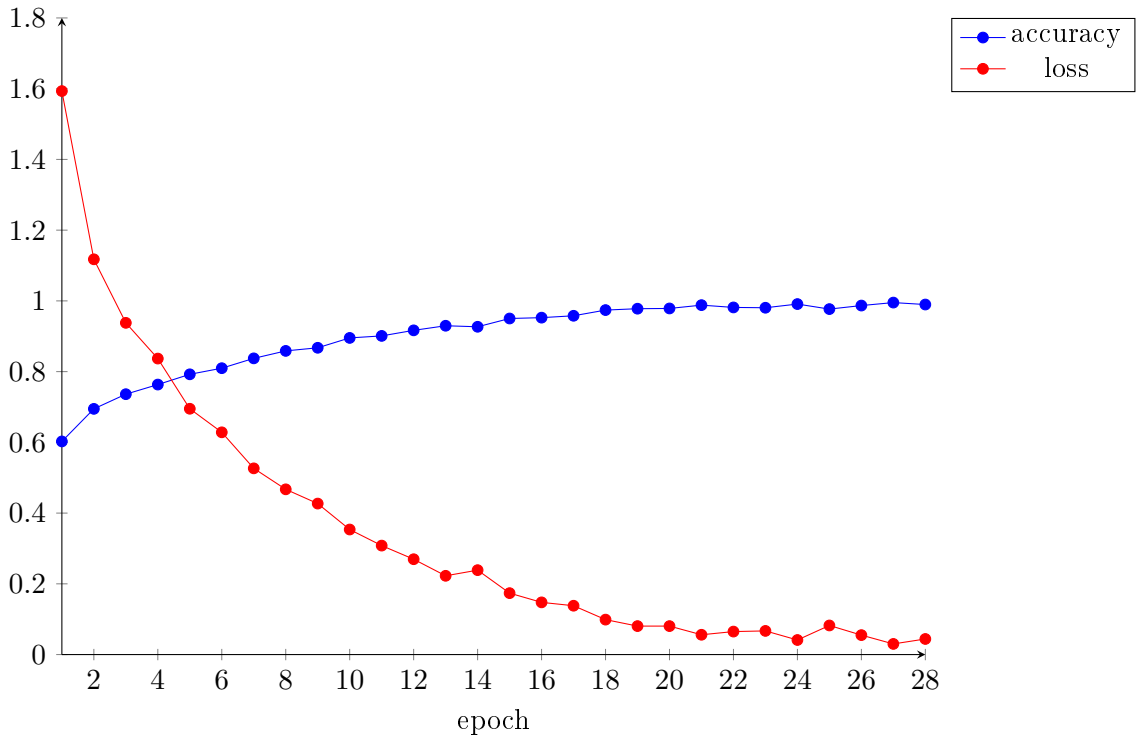
- Loss: 0.999524;
- Accuracy: 0.802284;

These results are very near to the results of AlexNet.

Version 2

In order to perform even better than AndreaNet v1 we created a second version. We maintained the layer structure of the v1 but we changed all the activation functions of the convolutional layers to ReLU. We also have chosen to try the SGD optimizer so we replaced adam with it.
Parameters:

- Total params: 3,288,080;
- Trainable params: 3,282,196;
- Non-trainable params: 5,884;



Training time: 7.983333 minutes.
Final epoch performance:

- Loss: 0.0441;
- Accuracy: 0.9897;

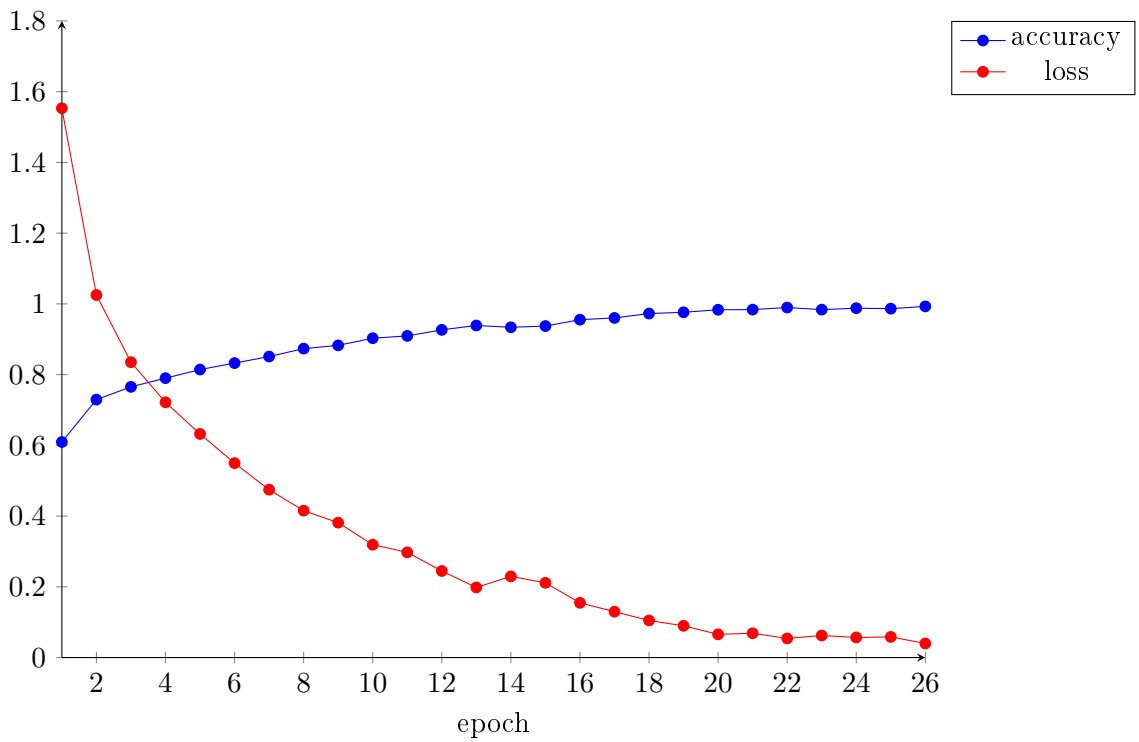
Test set performance:

- Loss: 0.988192;
- Accuracy: 0.822716;

Bingo! The results with input shape 320x320 are even better than the results of AlexNet!
Input shape: 420x420.

Parameters:

- Total params: 5,745,680;
- Trainable params: 5,739,796;
- Non-trainable params: 5,884;



Training time: 33.600000 minutes.
Final epoch performance:

- Loss: 0.0399;
- Accuracy: 0.9929;

Test set performance:

- Loss: 0.868631;
- Accuracy: 0.829327;

Enlarging the input shape for version 2 to 420x420 seems to not give a big gain. AlexNet won this round.

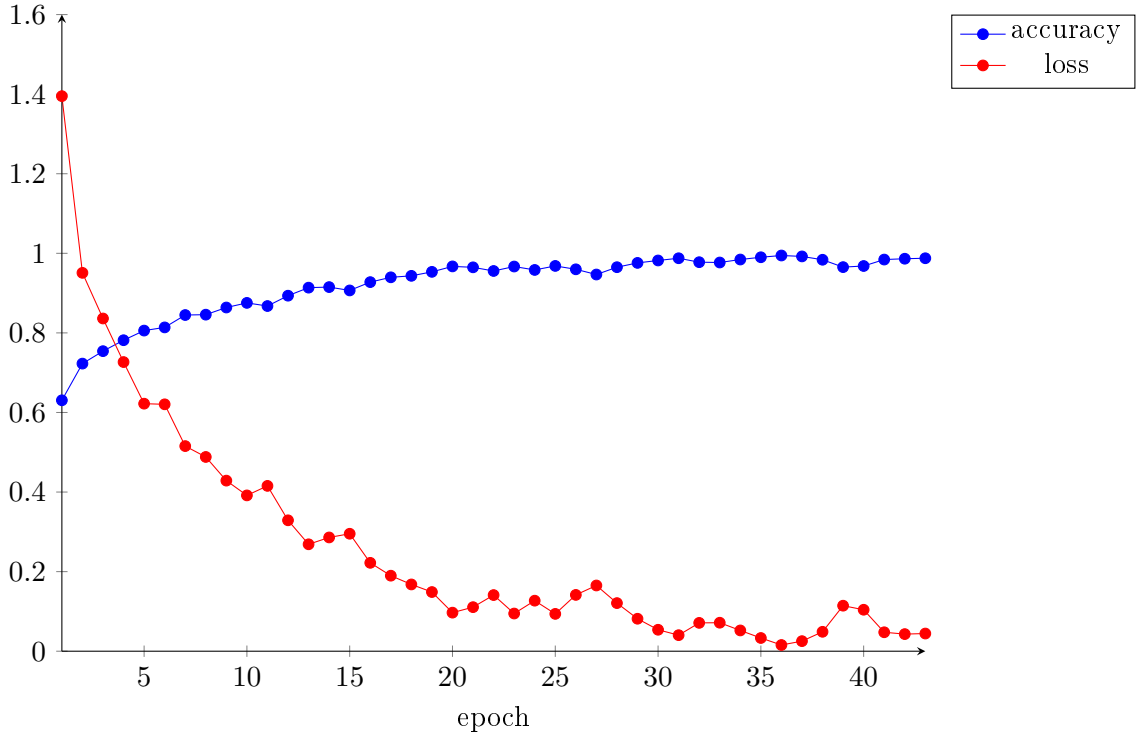
Version 3

In this version, we tried to enlarge the size of the convolutional layers. We also replaced the activation function of the dense layer to ReLU and changed back the optimizer to adam.
Now the differences between the layers structure from LeNet are very marked:

LeNet	**AndreaNet v3**
Convolutional 6 kernels 5x5	Convolutional 12 kernels 11x11
Average Pooling 2x2 stride 2x2	Average Pooling 2x2 stride 2x2
Convolutional 16 kernels 5x5	Convolutional 32 kernels 8x8
Average Pooling 2x2 stride 2x2	Average Pooling 2x2 stride 2x2
Convolutional 120 kernels 5x5	Convolutional 240 kernels 8x8
	Average Pooling 2x2 stride 2x2
	Convolutional 360 kernels 2x2
Fully connected, 84 units	Fully connected, 2048 units (dropout 0.5)
	Fully connected, 512 units
Fully connected, 24 units	Fully connected, 24 units

Input shape: 320x320.
Parameters:

- Total params: 4,892,080;
- Trainable params: 4,885,672;
- Non-trainable params: 6,408;



Training time: 14.550000 minutes.
Final epoch performance:

- Loss: 0.0444;
- Accuracy: 0.9876;

Test set performance:

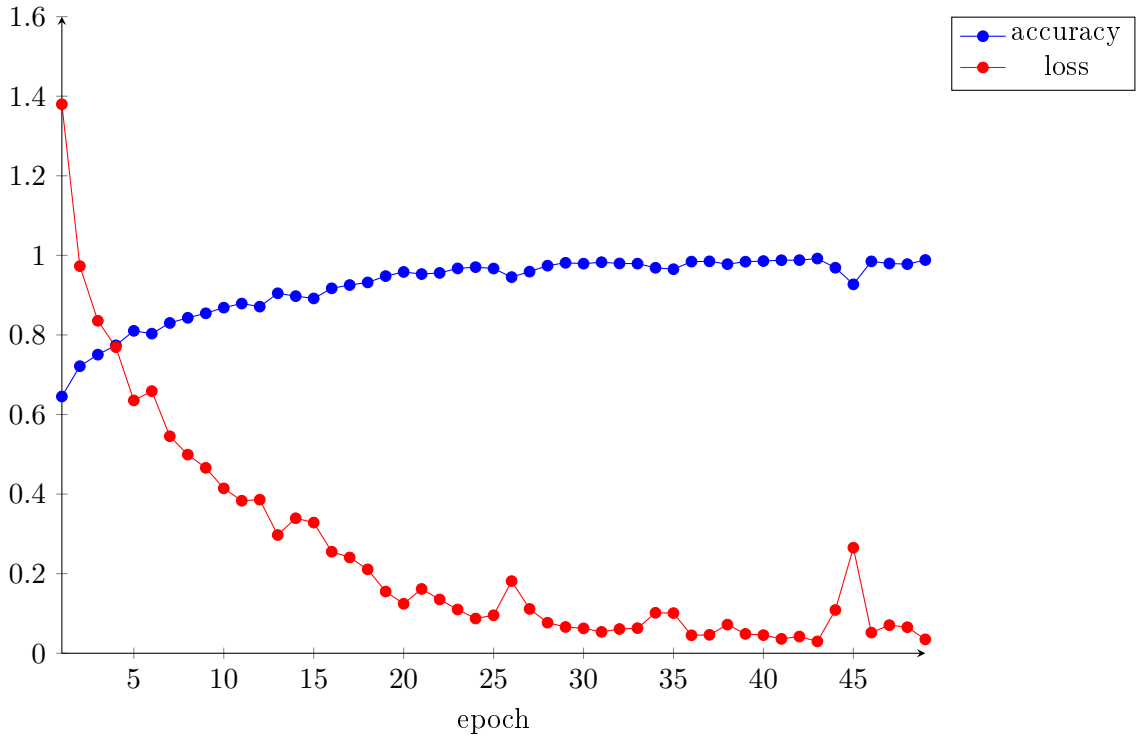
- Loss: 1.029947;
- Accuracy: 0.844351;

We performed even better than AlexNet with input shape 420x420. Adding a layer instead of enlarging the existing layers is generally preferable, but in this case was a good choice. The ReLU function is also very good and the gain respect tanh (or sigmoid) is tangible.
For curiosity, we also tried the other input shape.

Input shape: 420x420.
Parameters:

- Total params: 8,578,480;

- Trainable params: 8,572,072;
- Non-trainable params: 6,408;



Training time: 30.083333 minutes.
Final epoch performance:

- Loss: 0.0349;
- Accuracy: 0.9883;

Test set performance:

- Loss: 0.896843;
- Accuracy: 0.855769;

With much fewer parameters and much less training time than AlexNet we reached a result that is better than the expectations, comparable with AlexNet and with an accuracy even better.

Conclusions

We report in a compact way the results:

Model	Input Shape	Accuracy	Loss	Training Time
LeNet	320x320	0.192909	2.202084	17.7m
AlexNet	320x320	0.810096	1.190347	31.4m
	420x420	0.839543	0.866917	72.5m
AndreaNet v1	320x320	0.767428	1.466683	15.7m
	420x420	0.802284	0.999524	66.3m
AndreaNet v2	320x320	0.822716	0.988192	7.9m
	420x420	0.829327	0.868631	33.6m
AndreaNet v3	320x320	0.844351	1.029947	14.5m
	420x420	0.855769	0.896843	30.0m

As you can easily see AndreaNet v3 is the better choice for performance and for training time the choice is between it and the version 2. This net has only 8 million parameters (with input 420x420). Lenet has 55 million parameters and AlexNet 78 million. This allows us to train the network using less time and memory. In fact, memory is one of the biggest issues in training neural networks on a laptop.
This is a great goal for us, we created a speedy network with high performances starting from LeNet and, in the end, better even than AlexNet.