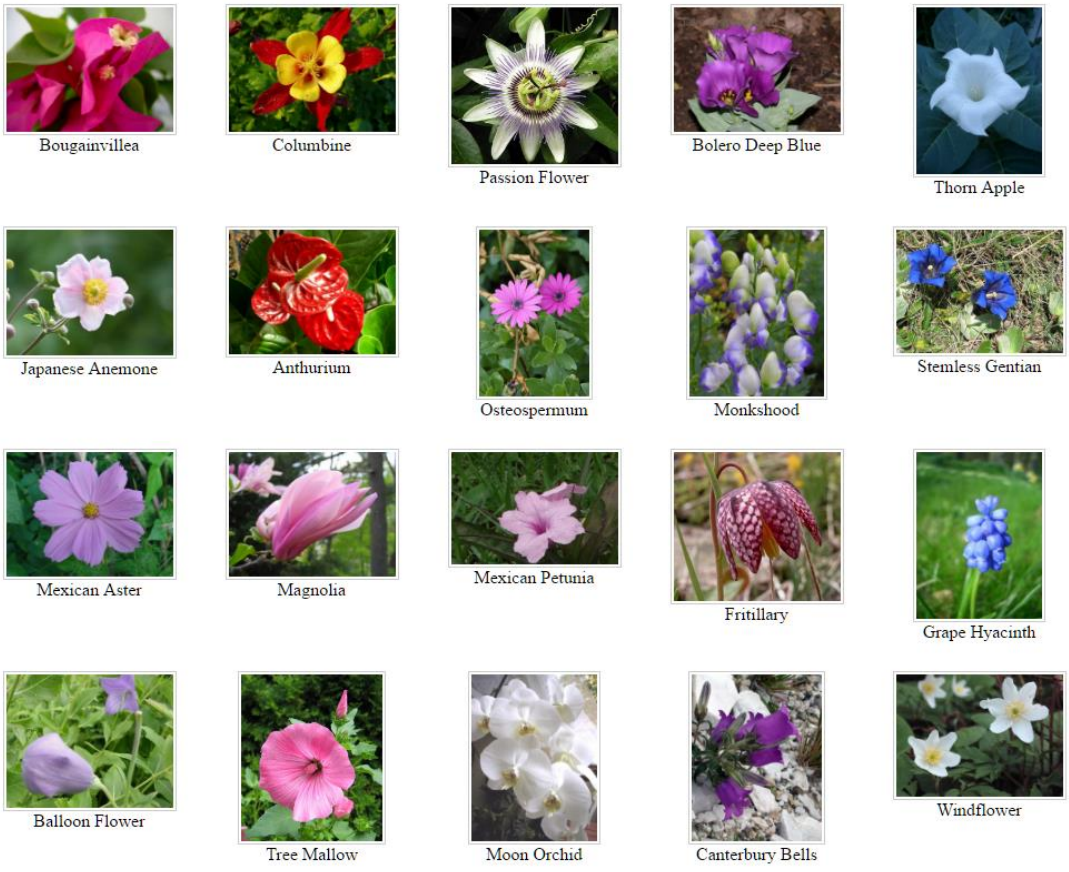
CNN for Classifying 102 flowers categories using transfer learning:

Dor Cohen 301216321 | Stav Sofer 301783619

Goal – Build a convolutional neural network that will classify flowers images to 102 different flowers categories with the highest accuracy possible.

Data – In one of the tutorials in Deep Learning course we were working with the Oxford flowers dataset 17 categories. We chose for this project to work with the extended dataset of 102 categories. Each category contains 40 to 258 images with average size of 500*600 pixels. The dataset contains 8189 images all in all.

Project sources and logs - https://github.com/dorcoh/oxford102



Project Challenges

- Dataset size – the Oxford flowers dataset contains 8189 images at total, a very small dataset when referring to neural network algorithm. Coping with a small dataset is one of the biggest challenges every programmer who works with neural network on a new problem must handle. Collecting a large number of image and classify them for training is a tedious and expensive issue, thus one will prefer the smallest dataset possible for reaching the wanted accuracy.
- New framework – During our course we usually worked with the Torch framework and programmed using Lua. This project we worked with another famous deep learning framework – Caffe. We chose Python to be our programming language.
- Transfer Learning – In this project we used neural networks that already been trained on huge datasets (not flowers datasets). We made use of the pre-trained weights to increase our accuracy significantly. In particular, we used 2 famous deep learning trained neural networks – AlexNet and VGG-S. Working through integrating these networks and their weights helped us understanding the power of transfer learning and was an important challenge to cope with.
- Object complexity – The task of classifying an image to one of 102 categories of different flowers is relatively complex because of the small differences between the different categories (compare with the task of classify a car versus a truck), the impact of the angle of the image taken and the light in it.

Running the networks

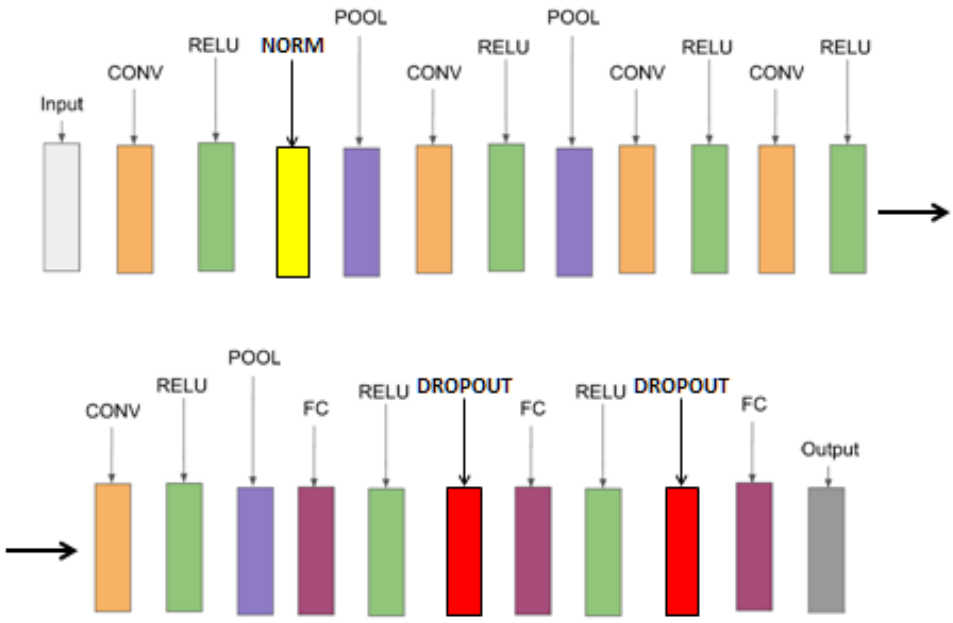
While working through our project, we had many attempts for understanding better which model will fit best for our goal. In this part, we will describe in details 4 of our most important and successful models. Later we will mention some of our insights with respect to these attempts.

For our project, we used the famous VGG-S and AlexNet models in different configurations. The AlexNet was suggested by Alex Krizhevsky back in 2012 and had been an important part in the big leap forward of the deep learning field. This paper has been cited for more than 6,000 times since then. The VGG-S model, suggested later by the Visual Geometry Group of Oxford University considered as another milestone in the short history of deep neural networks. As we will see, the later approach suggested some changes in the network architecture that bring better results in some of the cases, such as in our case.

VGG-S without transfer learning –

In this attempt we took the original VGG-S network architecture and modified it to our needs (as we done in all our trials). The changes we applied in the network are – modify the first layer to get the flowers dataset and the last layer to produce classification to our 102 different classes. No pre-trained weights were taken.

Layers architecture -

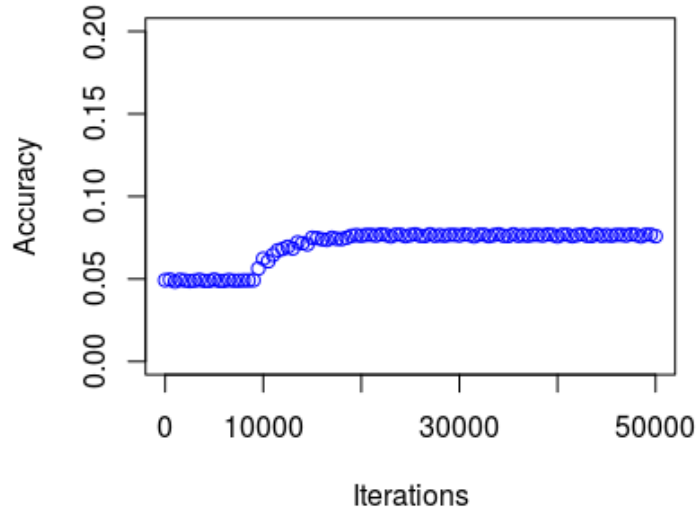


Convolution parameters –

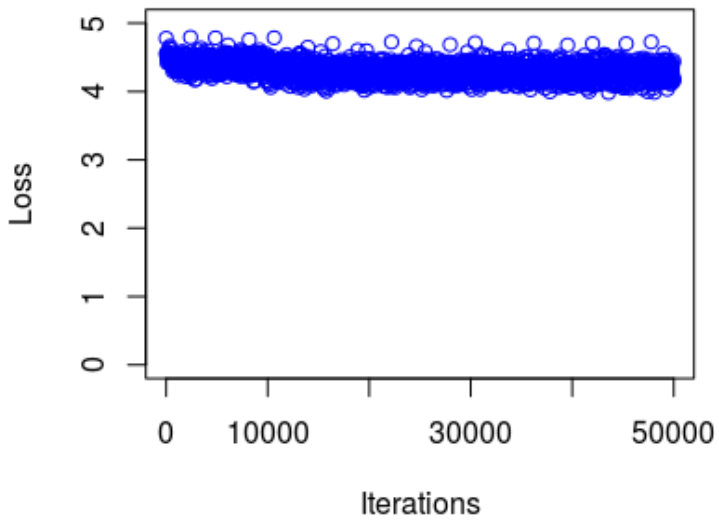
	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5
Output size	96	256	512	512	512
Kernel size	7	5	3	3	3
Stride	2	1	1	1	1

We set the learning rate to be 0.01, SGD optimization, and trained the network using GPU. Total of 50,000 iterations took 7:17 hours.

Accuracy as function of iterations in training –



Loss as function of iterations in training –



Best accuracy achieved at iteration 25,500 with 7.7%

As we can see, the original network, without pre-trained weights worked poorly in this case. We will now see how transfer learning comes to our aid.

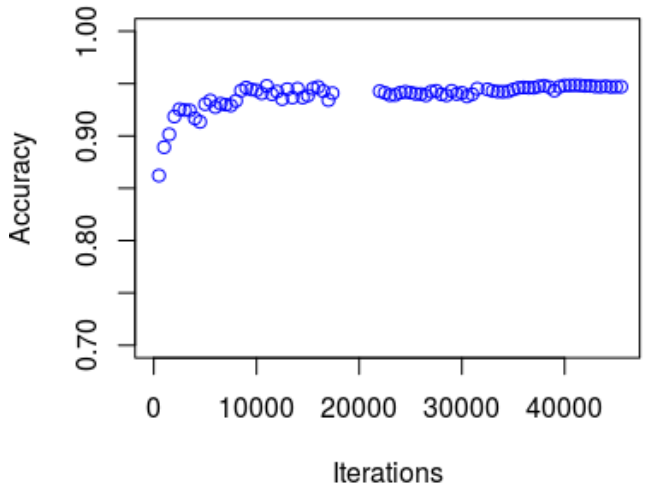
VGG-S with transfer learning –

We took the same VGG-S (except minor changes in the strides), but this time we used a pre-trained network, that has been trained over ImageNet dataset – more than 1.2 million(!) images on its training set that were originally classified 1000 categories.

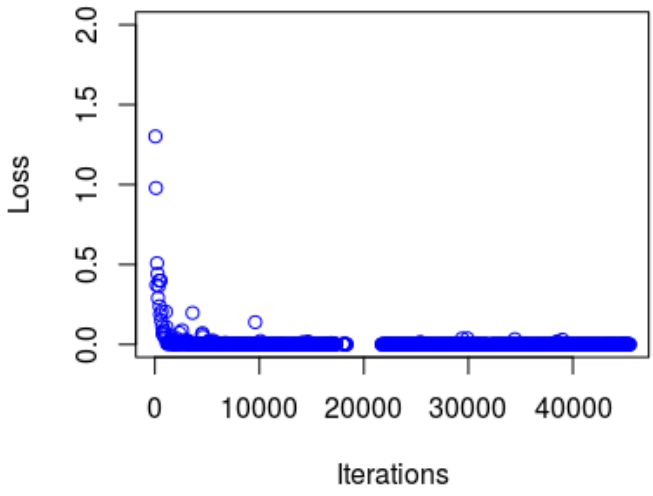
We set a low learning rate for each of the layers weights we took to enable adjustments for the layer's learnable parameters when training over our dataset. In the last fully connected layer we set a much larger learning rate because this is a new layer that had not been trained before.

We set the network learning rate to be 0.001, SGD optimization, and trained the network using GPU again. Total of 45,450 iterations took 19:44 hours.

Accuracy as function of iterations in training –



Loss as function of iterations in training –



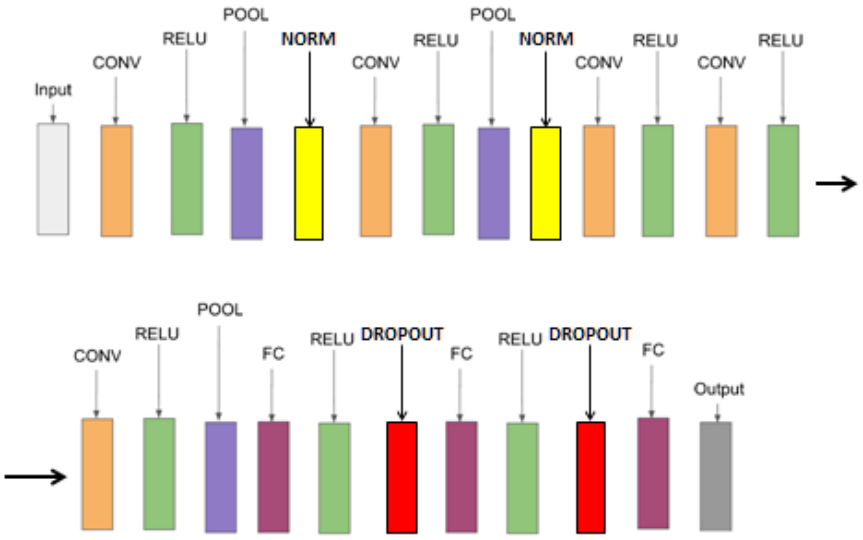
Best accuracy achieved at iteration 41,100 with 94.8%

As we can see, the pre-trained network did a good job, especially if we compare it with our first attempt. We will refer to these results soon.

AlexNet with transfer learning –

After being impressed with the power of a pre-trained network, we decided to see how AlexNet will work on our problem. After applying the needed changes, we got a pre-trained network. Again, the training has been done on more than 1.2 million images from ImageNet data with 100 categories originally.

Layers architecture -

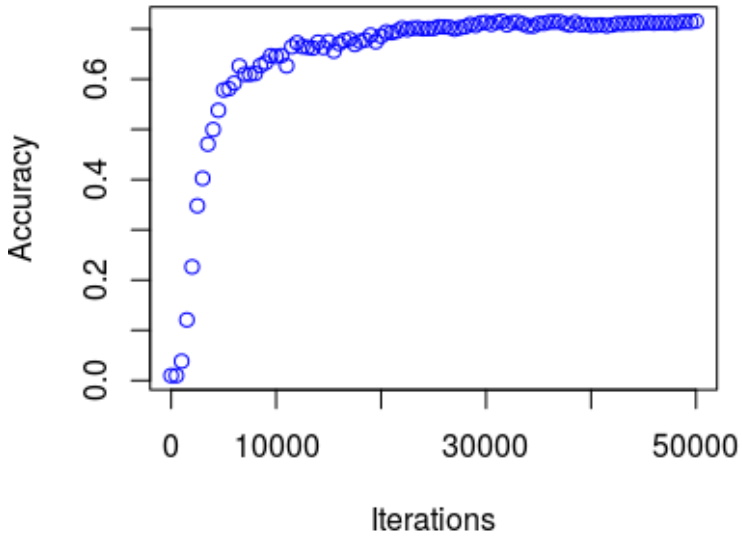


Convolution parameters –

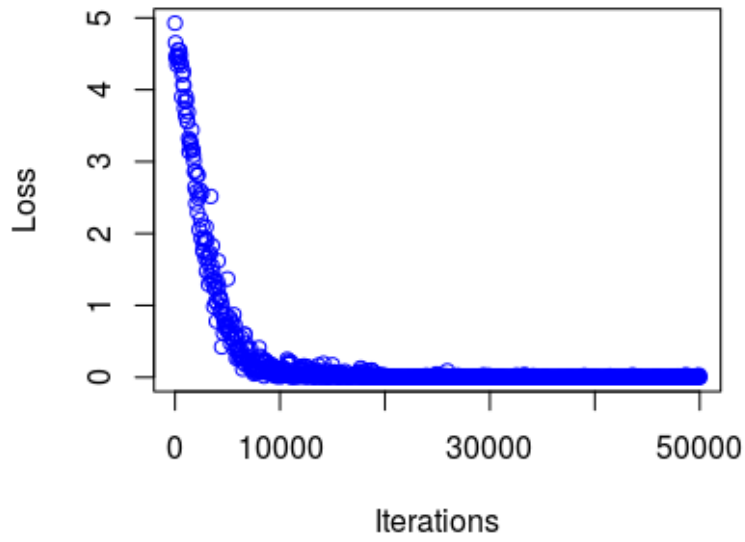
	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5
Output size	96	256	384	384	256
Kernel size	11	5	3	3	3
Stride	4	1	1	1	1

We set the learning rate to be 0.001, SGD optimization, and trained the network using GPU. Total of 50,000 iterations took 5:57 hours.

Accuracy as function of iterations in training –



Loss as function of iterations in training –



Best accuracy achieved at iteration 31,500 with 71.5%

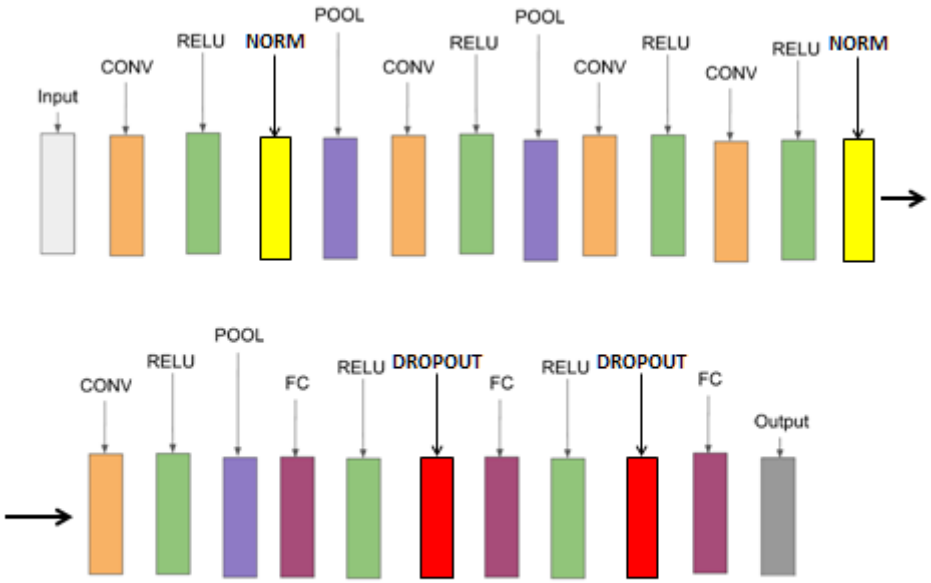
AlexNet results were inferior when comparing with the VGG-S. We will try to address and reason these results as well.

Last attempt – VGG-S with minor improvement –

As we saw, the VGG-S pre-trained did very well. After examining the network architecture, we thought that only one normalization layer for such a big network might be not enough and it worth trying adding another normalization layer to the winning network and hope to earn some improvement. No other changes has been made, compare with our second trial.

We added to new layer right after the fourth RelU layer.

Layers architecture -



Total of 50,000 iterations took 23:21 hours.

Best accuracy achieved at iteration 16,000 with 95.8%

- ➔ Adding another normalization layer gave us 1% accuracy improvement and reached it top accuracy 25,000 iterations earlier than the original VGG-S.

Results summary

	VGG-S	VGG-S pre-trained	AlexNet pre-trained	VGG-S pre-trained 2 norm
Top accuracy	7.7%	94.8%	71.5%	95.8%
Top accuracy iteration	25500	41100	31500	16000
Training time (hours)	7:17	19:44	5:57	23:21

Make predictions

Now when we got a satisfied result, we will test our best model.

We sent to our VGG-S model a new image of our favorite flower – the Corn Poppy. According to the categories mapping (class_labels), we expect to get category 26 with high probability. We will print the 5 highest probabilities the model returned.

Input -

Output –

```
/home/stavsofe@st.technion.ac.il/caffe-oxford102/input/parag_agasani3.jpg
(26, 0.98226929)
(83, 0.0065956311)
(58, 0.002759503)
(96, 0.0017469984)
(31, 0.0009043329)
26
```

As expected, the Corn Poppy image was classified correctly to it category (26) with a satisfied probability of 0.982.

Conclusions, Insights and remarks

We would like to address two issues that seem most significant to try and explain.

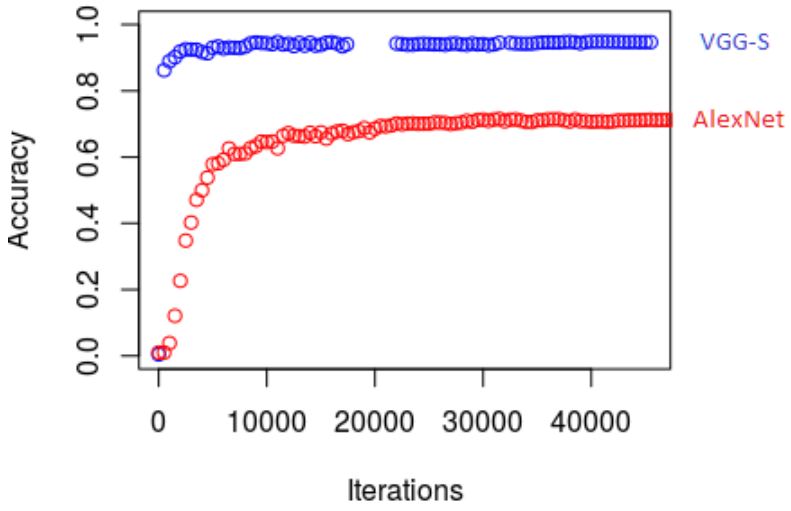
1. The large difference between the VGG-S and the pre-trained VGG-S results

As we worked through the project, we realized that in our problem, the difference between not-trained weights and pre-trained weights will be significant but did not guess such a large one. While the VGG-S did better than we thought, the not-trained network did worse than expected. We think the following arguments can explain some of the seen phenomenon –

- Datasets size differences – First and most important – Our training set is of less than 6500 images. This is by no means too small for such a big network as VGG. Moreover, the pre-trained network trained over more than 1,200,000 images and this difference obviously had a big influence over our results.
- Network size – As already mentioned, while our dataset is especially small, the network is deep and with many parameters. This ratio probably affects as well.
- Multi-categories classification – While the dataset is small, there are 102 different categories. We think a smaller number of categories could work better when training network for scratch – the less categories are, the better chances the network will classify correctly even when it has low probability for a specific input image.

Regardless of the reason to these results, we were highly impressed by the power of transfer learning and the relatively easy way of doing it with the given nowadays frameworks. Although the pre-trained dataset was of totally different categories, and even though we took all the layers (not only the earliest) as is, we reached impressively good accuracy.

2. The difference between VGG-S and AlexNet results



More than 20% accuracy difference between AlexNet and VGG-S top score made us think of possible reasons for it. As one can see, the layers architecture differences between the two networks are not significant (the Pooling layer moves before the normalization layer in the VGG). If we have a better look over the convolution layers in the two networks we will see some major differences –

		Conv 1	Conv 2	Conv 3	Conv 4	Conv 5
VGG	Output size	96	256	512	512	512
	Kernel size	7	5	3	3	3
	Stride	2	1	1	1	1
AlexNet	Output size	96	256	384	384	256
	Kernel size	11	5	3	3	3
	Stride	4	1	1	1	1

- Smaller 7*7 kernel size with stride of 2 in VGG, compare with 11*11 and stride of 4 is a big difference that may have been benefited with the VGG results.
- VGG wider deep layers – as we go deeper, VGG offering a wide layers with 512 outputs over "only" 384 outputs in the AlexNet
- AlexNet comes first – It is always harder to be the first. We can guess that the VGG group in Oxford already seen the AlexNet work and referenced their work to it, thus they enjoyed the AlexNet pioneered network and added their improvements.

- Remark – the input images for AlexNet were cropped to 227*227 while VGG-S to 224*224. This might also affect our results

Further things we have learned –

- Transfer learning is a powerful technique that worth trying whenever dealing with a new problem through neural networks
- When having such a small dataset and one decided training from scratch, large neural network might not be the best approach
- Caffe deep learning framework makes coding life easier when mainly using prototxt files to define your network. Still, its installations and configurations are not trivial at all, and its interface is not rich. Torch and Google Tensorflow may be a better choice for some of the cases (and some of the programmers)