Classifying Scene Images Using Convolutional Neural Networks

Zhongjie He
he.zho@husky.neu.edu

ABSTRACT

In my final project, I applied several advanced deep convolutional neural works including ResNet, GoogLeNet-Inception, SqueezeNet and DenseNet to the dataset from a competition held by challenger.ai[1].

I reshape the dataset into 3 different shapes: 48x48, 96x96, 176x176 and then applied different networks to them. Among all the networks, I believe the best one is DenseNet because it performs well even on smallest sized images. The best top-3 accuracy I got is 80.69% by GoogLeNet-Inception v4 but the computation cost is also the highest.

In this report, an introduction into the problem as well as the dataset is first made and followed by several advanced neural network case studies in section 2. Then in section 3, I talked about how I applied the CNNs and results are listed. Data augmentation is talked in section 4 and conclusions are made in the section 5.

1. INTRODUCTION

In the year of 1966, Marvin Minsky at MIT spend the summer with his students linking a camera to the computer and getting the computer describe what it saw[2]. This is the very first beginning of the computer vision. After decades' development, Convolutional Neural Networks (CNNs) have became the most popular approach to solve computer vision problems, including scene classification. In the recent years, this area has taken off because of two main reasons. One is that we are now able to train deep enough Neural Networks. Another is that with the help of internet, we are now being able to access huge enough amount of labeled images as training data. The rise of both the computer software and specialized hardware such as GPUs helps us to train a large neural network much faster than ever before. The larger the network is, the better performance we will got got under huge training-set.

In the competition, challenger.ai gathered images in different sizes and different ratios all from the internet. They collected 80,000 of images in 80 classes and each class contains 600 to 1,100 images. Images are divided into 4 parts. 1. Training-set (70%). 2. Validation-set (10%). 3. Test-set A (10%). 4. Test-set A B (10%). Test-sets are for result evaluation only and the labels of each image are not provided. The score of the competition is measured by the top-3 accuracy on the test-set.

Figure 1. Labeled images with different shapes in the dataset. Left: tower. Right top: farm field. Bottom middle: bridge. Bottom right: desert.

2. RELATED WORKS

Recent years, there has been many breakthroughs in computer vision especially in image classification. Deeper neural networks, complicate network connection strategy as well as new layers has been put forward.

2.1. ALEXNET[3]

In the ILSVRC[4] 2012, Krizhevsky et al. from University of Toronto designed a network later called AlexNet by which CNN is first used in image classification on large dataset with modern computers. It got 16% percent top-5 error rate on the ImageNet dataset which is really a huge improvement as the value on the same task in ILSVRC 2010 and ILSVRC 2011 were 27% and 28% respectively.

AlexNet is quite strait forward compared to today's CNNs. It contains 5 convolutional layers followed by 3 fully connected layers. In their paper, Max-pooling is used to reduce the parameter number and Dropout is used in CNNs to prevent overfitting.

2.2. VGG-16[5]

In the year of 2014, also on the ILSVRC challenge, Karen et al. from Oxford University brought the VGG to the world along with the idea that deeper CNNs get better performance in image classification.

Instead of using so many hyper parameters, all convolution layers in VGG-16 are 3 x 3 filter and 1 strides with same padding. The key that made VGG-16 successful is the depth of the network and the huge numbers of trainable parameters which is about 138 million.

VGG-16 also follows the pattern that when goes deeper into the network, the number of channel rises while the width and height of the volume reduces.

2.3. GOOGLENET - INCEPTION[6]

Inception network is a quite complex one. Instead of choosing from the size of the convolution layer, it uses them all. Different sizes of convolution layer is applied to the input and the results are concatenated together. In order to fit the dimension when performing the concatenation, same padding is used.

To reduce the number of parameters, 1x1 convolution bottleneck layer is applied before each 3x3 and 5x5 convolution layers. This saved a lot of computation cost without hurting the performance too much.

2.4. RESNET[7]

The full name of ResNet is residual network and its main idea is residual connections between layers which can help to train very deep neural networks. In a typical neural network, l^{th} layer's input goes throw a linear layer z and then a non-linearity $g(z)$ such as ReLU repeatedly. This block can be noted as follows:

$$
\begin{aligned}
z^{[l+1]} &= W^{[l+1]}a^{[l]} + b^{[l+1]} \\
a^{[l+1]} &= g^{[l+1]}(z^{[l+1]}) \\
z^{[l+2]} &= W^{[l+2]}a^{[l+1]} + b^{[l+1]} \\
a^{[l+2]} &= g^{[l+2]}(z^{[l+2]}) .
\end{aligned}
$$

Equation 1. Plain block

In the ResNet, however, when applying the second non-linear function g, the input $z^{[l+2]}$ will be added by $a^{[l]}$ and this makes $a^{[l]}$ directly collected to $a^{[l+2]}$ so that this is also called *shortcut* or *skip connection* and blocks with this connection is called residual block:

$$
\begin{aligned}
z^{[l+1]} &= W^{[l+1]}a^{[l]} + b^{[l+1]} \\
a^{[l+1]} &= g^{[l+1]}(z^{[l+1]}) \\
z^{[l+2]} &= W^{[l+2]}a^{[l+1]} + b^{[l+1]} \\
a^{[l+2]} &= g^{[l+2]}(z^{[l+2]} + a^{[l]}) .
\end{aligned}
$$

Equation 2. Residual block

In the ResNet paper, Kaiming et al. indicated that maybe because of the overfitting issue, simply stacking large number of plain layers will actually result in a lower training and test accuracy which can be avoid by ResNet. They trained several residual nets and the ensemble top-5 error rate on ImageNet dataset is 3.57% which surpassed the human-level performance 5.1%[8].

2.5. DENSENET[9]

DenseNet can be regarded as an advanced version of ResNet. Convolution layers are designed to be densely connected with each

other. A typical residual block in equation 2 can be simplified as: $a^{[l+1]} = f^{[l+1]}(a^{[l]}) + a^{[l]}$.

In the dense connectivity, information is much more further influenced between layers: $a^{[l+1]} = f^{[l+1]}(a^{[0]}, a^{[1]}, \ldots, a^{[l]})$.

2.6. FRACTALNET[10]

In very recent years, so many teams are working on the residual connectivity between years and they have made extraordinary achievement. This year Gustav et al. in their FractalNet paper proves that simply apply a expansion rule to generate a deep networks without residuals can also performs well. In a fractal block with C columns, the fractal expansion rule is:

$$f_{C+1}(a^{[l]}) = f_C(f_C(a^{[l]})) \oplus f_1(a^{[l]})$$

Equation 3. Fractal expansion rule

And the first function in the paper can be as simple as: $f_1(a^{[l]}) = Conv(a^{[l]})$.

This architecture is scalable, larger C value will leads to a deeper fractal block which takes more computational cost and also performs better.

There will be $P_C = \begin{cases} 1 & \text{if } C = 1 \\ P_{C-1}^2 + 1 & \text{if } C > 1 \end{cases}$ different paths from the input to the concatenation layer in one fractal block. Very similar to the weight-level dropout, path-level dropout can be applied in a fractal block as regularization.

2.7. SQUEEZENET[11]

SqueezeNet is a quite simple network with relative little number of parameters. It is not super-powerful but very practical. Its' simple architecture and small number number of parameters makes it easy to implement and to training on.

In the paper, input is squeezed by a 1x1 convolution layer and then expanded by both a 1x1 and 3x3 convolution layer and results of the expanded layers are concatenated together. Compared to AlexNet, the accuracy is similar but it uses 50 times fewer parameters.

3. RESULTS

I'm using a Google Compute Engine VM with a NVIDIA K80 GPU to do all the work. All my NN related work are done by using Keras[12] with TensorFlow[13] as backend.

3.1. PREPROCESS

As mentioned before, all images in my dataset are all collected from the internet so that they are in different shape. In order to make use of neural networks in image classification, the first step is to reshape all the images into the same size typically in the shape of square. Useful information may lost during the reshape. One extreme example is that a round soccer ball will easily be reshape to a elliptical football and may cause the model confuse between the football field scene and the soccer field scene. To evaluate how this effects the whole dataset, I calculated the *relative aspect ratio* for an image with width w and height h as $R = \frac{\min(w,h)}{\max(w,h)}$.

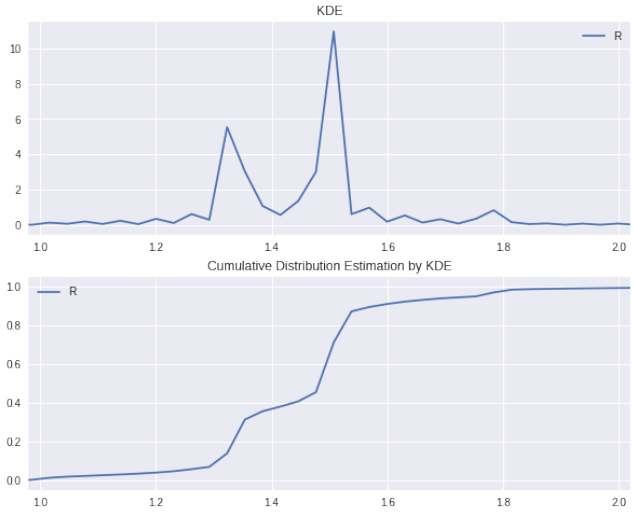


Figure 2. Training set relative aspect ratio distribution. Top: KDE, Bottom: Cumulative distribution estimation.

When performing a reshaping, images with larger R will be more strongly squeezed. The cumulative distribution plot of R indicates that approximately over 50% of the images have the $R > 1.5$. When reshaping these images to squared shape, they are relatively strong squeezed. So simply reshape the whole dataset to square will leads to the information loss.

3.2. BASELINE MODEL

In the very first beginning, I implemented a baseline model from the scratch, it contains 4 different convolutional layers in total. Mimicking the AlexNet, the deeper layers have smaller width and height bug a larger number of channels. After about 10 epochs, this model tend to plateau at the top-3 accuracy is around 40%. This act as a benchmark for my future work.

3.3. RESNET

Keras officially implemented the ResNet50. Though much smaller than the 152 layers architecture in the original paper, the computational cost of ResNet50 is still unaffordable for me. So I implemented the ResNet18 and ResNet34 by myself. Im my implementation, I find out that in order to perform a concatenate, $z^{[l+2]}$ and $a^{[l]}$ should have the same dimension. So same padding is widely used in the ResNet.

I trained 18, 34, 50 layers ResNet with all images reshape to 48x48, 96x96 and 176x176 and the batch size for different sized images are 512, 256 and 128 respectively. My computer resource do not allow me to use higher resolution images. All the ResNets are initialized by He normalizer[14] and have the L2 regularizer weight-decay $l_2 = 10^{-4}$ and the optimizer is Adam. Results can be found in Table1. and plots of loss values during epochs can be found in appendix.

It turns out that I do not gain any accuracy prompt by using deeper ResNets. ResNet is designed to classify millions images from a thousand classes and my dataset may be too small for it.

	Image Size	Top-3 Accuracy	Val Top-3 Accuracy	Average Epoch Time
ResNet18	48x48	99.99%	50.88%	49.06s
	96x96	99.99%	66.66%	119.48s
	176x176	99.99%	74.83%	351.25s
ResNet34	48x48	99.97%	47.85%	80.92s
	96x96	99.98%	67.63%	194.56s
	176x176	100.00%	76.03%	552.93s
ResNet50	48x48	100.00%	49.49%	121.60s
	96x96	100.00%	64.73%	328.43s
	176x176	100.00%	74.94%	997.24s

Table 1. ResNet training results

Another notable thing here is the overfitting issue. ResNet does not include any dropout layer which is typically used to avoid overfitting in a neural network. I tried to add dropout layers with drop rate $r = 0.5$ after each convolution layer but it does not reduce the overfitting at all. Actually Batch Normalization[15] is heavily used in ResNet, which can make layers more independent to each other and can also act as regularization because similar to dropout, it adds some noise to each hidden layer's activations.

3.4. GOOGLENET - INCEPTION

GoogLeNet inception is also provided officially by keras and in their implementation, minimum input image size is 139x139. Because of its complexity, I don't have time to implemented it by my self to fit my smaller sized images so I only tried my 176x176 dataset on it. According to the same guess discussed before in the ResNet section, I quite doubt the performance when such large and complex networks is performed on small sized images.

	Top-3 Accuracy	Val Top-3 Accuracy	Average Epoch Time
Inception V3	99.98%	63.05%	750.25s
Inception V4	99.99%	80.69%	1489.22s

Table 2. GoogLeNet training results

The Inception V4[16] is their latest version this year in which residual connections between inception blocks are applied so that this version is also called Inception-ResNet.

3.5. SQUEEZENET

I implemented the SqueezeNet by my self and applied it to my dataset. When training the SqueezeNet, I found out that it is super sensitive to hyper parameters especially learning rate. A larger or smaller learning rate will leads the SqueezeNet not to convergence at all. I'm using the Adam optimizer and the learning rate $lr = 0.3 \times 10^{-3}$ seems to be a good point to start with. And because of the small learning rate, I have to train the network for a very long time until it convergences, though the average epoch time is only 2 minutes.

	Top-3 Accuracy	Val Top-3 Accuracy	Average Epoch Time
SqueezeNet	71.14%	64.55%	124.23s

Table 3. SqueezeNet training results

Note that the accuracy on the training set is not as high as it is in ResNet and Inception. This is because of its simplicity, SqueezeNet is quite hard to overfitting on training data.

3.6. DENSENET

I also implemented DenseNet by Keras. But when training a DenseNet on a TensorFlow backend, I can only feed a tiny amount of input data to the GPU or else the memory will blow up, if I want to use 176x176 image, the batch size would have to be too small to make sense. This is because concatenate layers are heavily used in DenseNet to densely connect different convolution layers and currently in TensorFlow's implementation, concatenation will allocate new memory for the output[17]. Due to this, unfortunately, I can only train DenseNet on my 48x48 dataset.

I build a 22 layer DenseNet and a 40 layer one. The latter is exact the same as that is used in the original DenseNet paper. They both performs nearly the same as those trained by 96x96 networks. So I think this is the best network for my dataset and the the dense connectivity do help. Similar to ResNet, deeper DenseNet does not help much.

	Top-3 Accuracy	Val Top-3 Accuracy	Average Epoch Time
DenseNet 22	66.92%	66.97%	231.43s
DenseNet 40	65.65%	65.86%	694.76s

Table 4. DenseNet training results

4. DATA AUGMENTATION

Data augmentation is widely used in training a neural network classifier. I performed the popular augmentation strategy 10-crop on my training data to get 10 times of images as before. Specifically, first pre-reshape the image to 64x64, which is slightly larger than the target size 48x48, and then crop the 4 corner and center of the image by target size. These 5 cropped images along with their horizontally flipped copies compose the augmented data.

In the preprocess section, potential information loss when changing the aspect ratio is discussed. In order to prevent this, I also tried to pre-reshape the images respect their aspect ratios and make the shorter one of edge of the image to be 64.

I trained the ResNet 18 on two new augmented dataset on It turns out that both augmented datasets do not help much on the top-3 accuracy.

My intuition of this is that just as shown in the Figure 8, none of the 10 crops of a 'river' image looks like a 'river' image to me. Generally, sometimes only looking at the whole image can

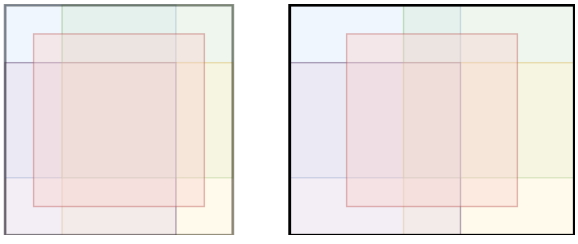


Figure 3. 10 crop. Left: 10 crop on a squared image. Right: 10 crop on a rectangular image.

we have enough information to classify it. Simply cropping these images just add more noise to the training set rather than useful information.

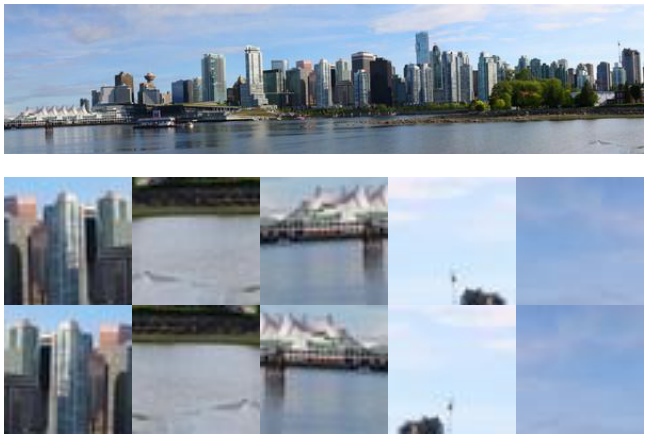


Figure 4. River labeled image and its 10 crops. Top: original image. Bottom: 10 crops.

5. CONCLUSION AND FUTURE WORK

My overall experience is that in this entry level competition, people have stronger computation power will get better score. If I have multiple machines I can train different networks with different hyper parameters at the same time.

About the lacking of memory that talked earlier, which prevented me from training on the larger sized images. This problem can be solved by making use of the keras iterator api. The iterator allows loading batch data from the local disk during the training time. I tried several times but the cost of loading data from the disk made the training process much slower.

The second thought is that I could actually do better in data augmentation. Instead of cropping the image, other augmentation such as tuning the brightness, greyscale or alpha can also be performed. The image iterator is also capable of performing the data augmentation in real time.

One last thing is that there are many pre-trained models that I can make use of to perform a transfer learning. For instance, my 80 classes is very likely covered by the ImageNet 1000 classes, or at least low level features are shared between images. I can take a pre-trained ImageNet model and replace the output layer with my 80 labels SoftMax. Hopefully this will help me save a lot of time in training the model.

Image Size	Network	Top-3 Accuracy	Val Top-3 Accuracy	Average Epoch Time
48x48	**ResNet18**	99.99%	50.88%	**49.06s**
	ResNet34	99.97%	47.85%	80.92s
	ResNet50	100.00%	49.49%	121.60s
	DenseNet22	66.92%	**66.97%**	231.43s
	DenseNet40	65.65%	65.86%	694.76s
96x96	**ResNet18**	99.99%	66.66%	**119.48s**
	ResNet34	99.98%	**67.63%**	194.56s
	ResNet50	100.00%	64.73%	328.43s
176x176	**ResNet18**	99.99%	74.83%	351.25s
	ResNet34	100.00%	76.03%	552.93s
	ResNet50	100.00%	74.94%	997.24s
	Inception V3	99.98%	63.05%	750.25s
	Inception V4	99.99%	**80.69%**	1489.22s
	SqueezeNet	71.14%	64.55%	**124.23s**

Table 5. All training results

All my implementation of networks can be found on my GitHub repo[18].

[1] http://challenger.ai/

[2] https://dspace.mit.edu/handle/1721.1/6125

[3] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

[4] http://www.image-net.org/challenges/LSVRC/

[5] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[6] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[7] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[8] Russakovsky, Olga, et al. "Imagenet large scale visual recognition challenge." *International Journal of Computer Vision* 115.3 (2015): 211-252.

[9] Huang, Gao, et al. "Densely connected convolutional networks." *arXiv preprint arXiv:1608.06993* (2016).

[10] Larsson, Gustav, Michael Maire, and Gregory Shakhnarovich. "Fractalnet: Ultra-deep neural networks without residuals." *arXiv preprint arXiv:1605.07648* (2016).

[11] Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size." *arXiv preprint arXiv:1602.07360* (2016).

[12] https://keras.io/

[13] https://www.tensorflow.org/

[14] He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." *Proceedings of the IEEE international conference on computer vision*. 2015.

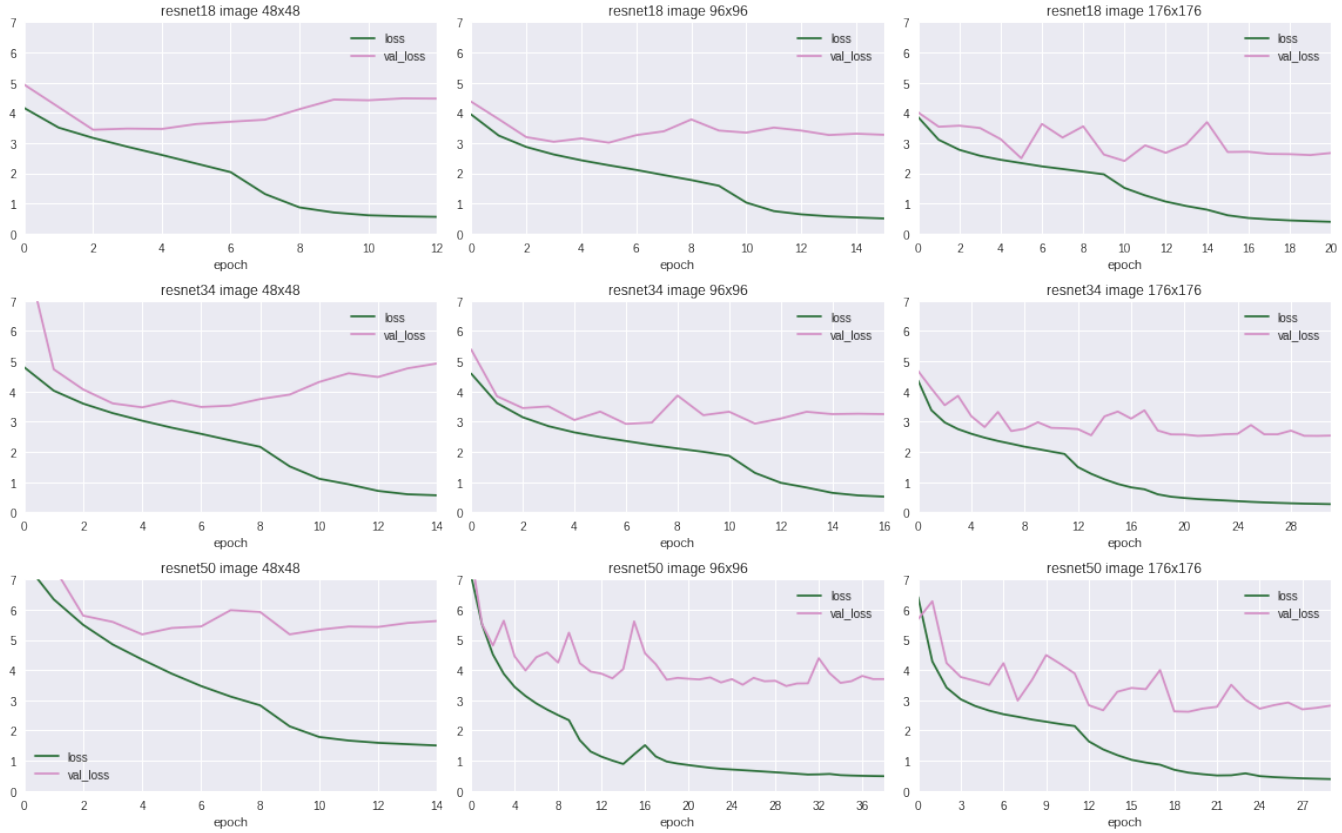
[15] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." International Conference on Machine Learning. 2015.

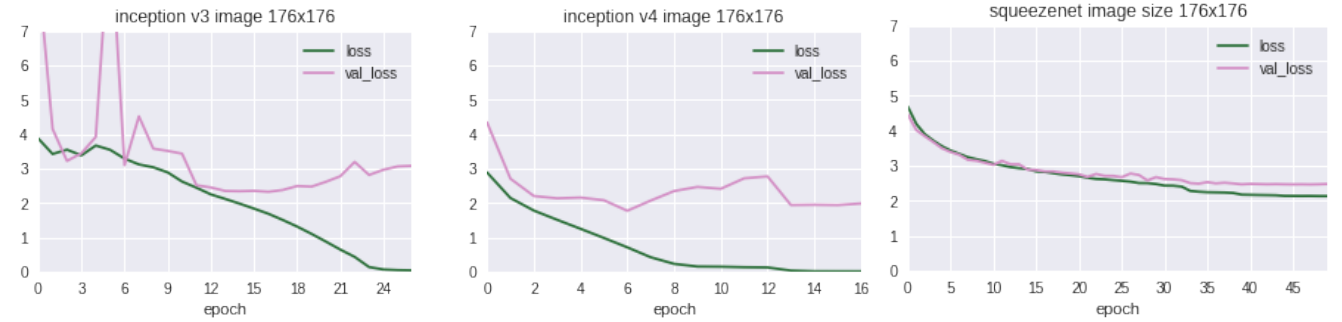
[16] Szegedy, Christian, et al. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning." *AAAI*. 2017.

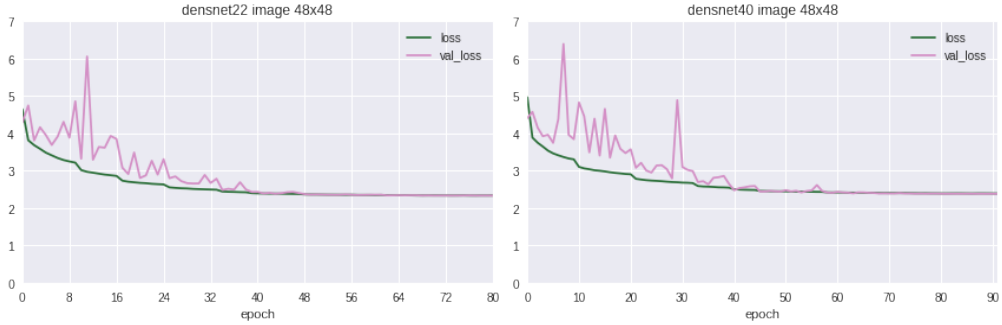
[17] https://github.com/tensorflow/tensorflow/blob/v1.4.1/tensorflow/core/kernels/concat_op.cc#L130

[18] https://github.com/eric6356/keras-cnn

APPENDIX


resnet18 image 48x48
resnet18 image 96x96
resnet18 image 176x176
resnet34 image 48x48
resnet34 image 96x96
resnet34 image 176x176
resnet50 image 48x48
resnet50 image 96x96
resnet50 image 176x176
loss
val_loss
epoch


inception v3 image 176x176
inception v4 image 176x176
squeezenet image size 176x176
loss
val_loss
epoch


densnet22 image 48x48
densnet40 image 48x48
loss
val_loss
epoch