Must Know Tips/Tricks in Deep Neural Netw Wei)

Deep Neural Networks, especially ***Convolutional Neu*** computational models that are composed of multiple p of data with multiple levels of abstraction. These meth state-of-the-arts in visual object recognition, object det domains such as drug discovery and genomics.

In addition, many solid papers have been published in source CNN software packages have been made availa tutorials or CNN software manuals. However, it might summary about the details of how to implement an exc networks from scratch. Thus, we collected and conclud DCNNs. **Here we will introduce these extensive im *tips*, for building and training your own deep netv**

（注：因版权所限，未经本人同意，请勿擅自翻译本文，联系方式参

Introduction

We assume you already know the basic knowledge of deep learning, and here we (tricks or tips) in Deep Neural Networks, especially CNN for image-related tasks *augmentation*; **2)** *pre-processing on images*; **3)** *initializations of Networks*; **4)** *so activation functions*; **6)** *diverse regularizations*; **7)** *some insights found from figu multiple deep networks*.

Additionally, the **corresponding slides** are available at `[slide]`. If there are any and slides, or there are something important/interesting you consider that shoul

Sec. 1: Data Augmentation

Since deep networks need to be trained on a huge number of training images to original image data set contains limited training images, it is better to do data au Also, data augmentation becomes the thing must to do when training a deep netv

- There are many ways to do data augmentation, such as the popular **horizon jittering**. Moreover, you could try combinations of multiple different proces scaling at the same time. In addition, you can try to raise saturation and val space) of all pixels to a power between 0.25 and 4 (same for all pixels within factor between 0.7 and 1.4, and add to them a value between -0.1 and 0.1. A

0.1] to the hue (H component of HSV) of all pixels in the image/patch.

- Krizhevsky *et al*. [1] proposed ***fancy PCA*** when training the famous *Alex-Ne* intensities of the RGB channels in training images. In practice, you can first values throughout your training images. And then, for each training image, RGB image pixel (i.e., $I_{xy} = [I_{xy}^R, I_{xy}^G, I_{xy}^B]^T$): $[\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^\mathbf{T}$ v and eigenvalue of the 3×3 covariance matrix of RGB pixel values, respectiv from a Gaussian with mean zero and standard deviation 0.1. Please note tha pixels of a particular training image until that image is used for training aga the same training image again, it will randomly produce another α_i for data "*fancy PCA could approximately capture an important property of natural ir invariant to changes in the intensity and color of the illumination*". To the cl reduced the top-1 error rate by over 1% in the competition of ImageNet 201

Sec. 2: Pre-Processing

Now we have obtained a large number of training samples (images/crops), but pl necessary to do pre-processing on these images/crops. In this section, we will int processing.

The first and simple pre-processing approach is **zero-center** the data, and then two lines Python codes as follows:

```
>>> X -= np.mean(X, axis = 0) # zero-center
>>> X /= np.std(X, axis = 0) # normalize
```

where, X is the input data (NumIns×NumDim). Another form of this pre-processi min and max along the dimension is -1 and 1 respectively. It only makes sense to reason to believe that different input features have different scales (or units), bu importance to the learning algorithm. In case of images, the relative scales of pix in range from 0 to 255), so it is not strictly necessary to perform this additional p

Another pre-processing approach similar to the first one is **PCA Whitening**. In t described above. Then, you can compute the covariance matrix that tells us abou

```
>>> X -= np.mean(X, axis = 0) # zero-center
>>> cov = np.dot(X.T, X) / X.shape[0] # compute the covariance matrix
```

After that, you decorrelate the data by projecting the original (but zero-centered

```
>>> U,S,V = np.linalg.svd(cov) # compute the SVD factorization of the dat
>>> Xrot = np.dot(X, U) # decorrelate the data
```

The last transformation is whitening, which takes the data in the eigenbasis and eigenvalue to normalize the scale:

```
>>> Xwhite = Xrot / np.sqrt(S + 1e-5) # divide by the eigenvalues (which a
```

Note that here it adds 1e-5 (or a small constant) to prevent division by zero. One can greatly exaggerate the noise in the data, since it stretches all dimensions (in variance that are mostly noise) to be of equal size in the input. This can in practi (i.e., increasing 1e-5 to be a larger number).

Please note that, we describe these pre-processing here just for completeness. In used with Convolutional Neural Networks. However, it is also very important to z see **normalization** of every pixel as well.

Sec. 3: Initializations

Now the data is ready. However, before you are beginning to train the network, y

All Zero Initialization

In the ideal situation, with proper data normalization it is reasonable to assume t be positive and half of them will be negative. A reasonable-sounding idea then m *zero*, which you expect to be the "best guess" in expectation. But, this turns out t in the network computes the same output, then they will also all compute the sar and undergo the exact same parameter updates. In other words, there is no sour weights are initialized to be the same.

Initialization with Small Random Numbers

Thus, you still want the weights to be very close to zero, but not identically zero. neurons to small numbers which are very close to zero, and it is treated as *symm* neurons are all random and unique in the beginning, so they will compute distinc diverse parts of the full network. The implementation for weights might simply lo $N(0,1)$ is a zero mean, unit standard deviation gaussian. It is also possible to use distribution, but this seems to have relatively little impact on the final performan

Calibrating the Variances

One problem with the above suggestion is that the distribution of the outputs fro variance that grows with the number of inputs. It turns out that you can normaliz 1 by scaling its weight vector by the square root of its *fan-in* (i.e., its number of i

```
>>> w = np.random.randn(n) / sqrt(n) # calibrating the variances with 1/s
```

where "randn" is the aforementioned Gaussian and "n" is the number of its input network initially have approximately the same output distribution and empiricall detailed derivations can be found from Page. 18 to 23 of the slides. Please note t

consider the influence of ReLU neurons.

Current Recommendation

As aforementioned, the previous initialization by calibrating the variances of neu
more recent paper on this topic by He *et al*. [4] derives an initialization specifica
that the variance of neurons in the network should be $2.0/n$ as:

```
>>> w = np.random.randn(n) * sqrt(2.0/n) # current recommendation
```

which is the current recommendation for use in practice, as discussed in [4].

Sec. 4: During Training

Now, everything is ready. Let's start to train deep networks!

- **Filters and pooling size**. During training, the size of input images prefers
  CIFAR-10), 64, 224 (e.g., common used *ImageNet*), 384 or 512, etc. Moreov
  (e.g., 3×3) and small strides (e.g., 1) with zeros-padding, which not only re
  improves the accuracy rates of the whole deep network. Meanwhile, a speci
  with stride 1, could preserve the spatial size of images/feature maps. For th
  pooling size is of 2×2.

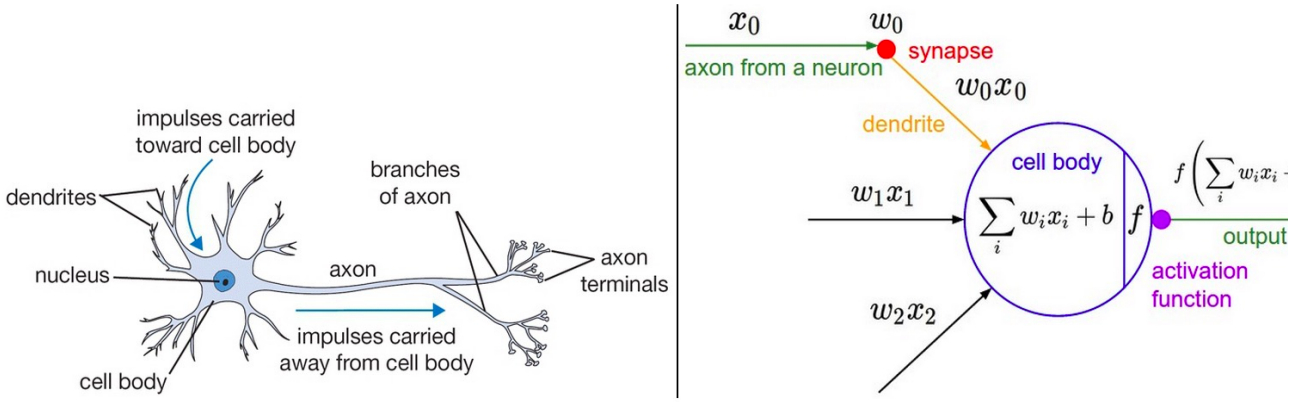
- **Learning rate**. In addition, as described in a blog by Ilya Sutskever [2], he
  mini batch size. Thus, you should not always change the learning rates (LR)
  obtaining an appropriate LR, utilizing the validation set is an effective way.
  beginning of your training is 0.1. In practice, if you see that you stopped ma
  divide the LR by 2 (or by 5), and keep going, which might give you a surpris

- **Fine-tune on pre-trained models**. Nowadays, many state-of-the-arts deep
  research groups, i.e., *Caffe Model Zoo* and *VGG Group*. Thanks to the wond
  trained deep models, you could employ these pre-trained models for your ov
  improving the classification performance on your data set, a very simple yet
  pre-trained models on your own data. As shown in following table, the two r
  new data set (small or big), and its similarity to the original data set. Differe
  utilized in different situations. For instance, a good case is that your new da
  training pre-trained models. In that case, if you have very little data, you ca
  features extracted from the top layers of pre-trained models. If your have qu
  a few top layers of pre-trained models with a small learning rate. However,
  from the data used in pre-trained models but with enough training images,
  tuned on your data also with a small learning rate for improving performanc
  contains little data, but is very different from the data used in pre-trained m
  data is limited, it seems better to only train a linear classifier. Since the data
  to train the classifier from the top of the network, which contains more data
  work better to train the SVM classifier on activations/features from somewh

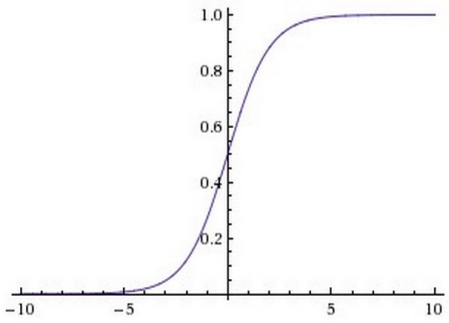
	very similar dataset	very different dataset
very little data	Use linear classifier on top layer	You're in trouble... Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a large number of layers

Sec. 5: Activation Functions

One of the crucial factors in deep networks is *activation function*, which brings t
will introduce the details and characters of some popular activation functions an



Sigmoid



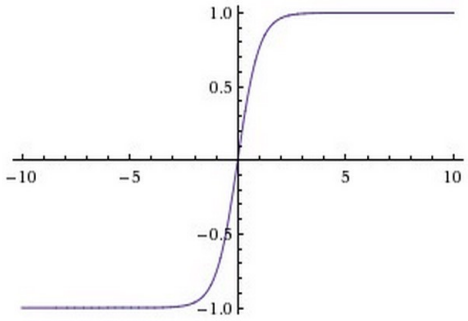
The sigmoid non-linearity has the mathematical fo
valued number and "squashes" it into range betwe
numbers become 0 and large positive numbers be
frequent use historically since it has a nice interpr
from not firing at all (0) to fully-saturated firing at

In practice, the sigmoid non-linearity has recently fallen out of favor and it is rar

drawbacks:

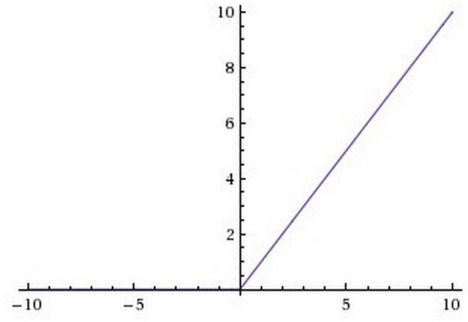
1. *Sigmoids saturate and kill gradients*. A very undesirable property of the sig activation saturates at either tail of 0 or 1, the gradient at these regions is a propagation, this (local) gradient will be multiplied to the gradient of this ga Therefore, if the local gradient is very small, it will effectively "kill" the grad through the neuron to its weights and recursively to its data. Additionally, o initializing the weights of sigmoid neurons to prevent saturation. For examp most neurons would become saturated and the network will barely learn.
2. *Sigmoid outputs are not zero-centered*. This is undesirable since neurons in Network (more on this soon) would be receiving data that is not zero-center during gradient descent, because if the data coming into a neuron is always $f = w^T x + b$), then the gradient on the weights w will during back-propagat negative (depending on the gradient of the whole expression f). This could dynamics in the gradient updates for the weights. However, notice that onc batch of data the final update for the weights can have variable signs, some is an inconvenience but it has less severe consequences compared to the sa

tanh(x)



The tanh non-linearity squashes a real-valued nu sigmoid neuron, its activations saturate, but unli centered. Therefore, in practice the tanh non-lin nonlinearity.

Rectified Linear Unit



The Rectified Linear Unit (ReLU) has become ve computes the function $f(x) = max(0, x)$, which is

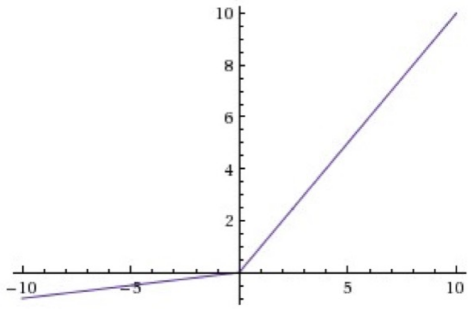
There are several pros and cons to using the ReLUs:

1. (*Pros*) Compared to sigmoid/tanh neurons that involve expensive operations

implemented by simply thresholding a matrix of activations at zero. Meanwl saturating.

2. (*Pros*) It was found to greatly accelerate (e.g., a factor of 6 in [1]) the conve compared to the sigmoid/tanh functions. It is argued that this is due to its li
3. (*Cons*) Unfortunately, ReLU units can be fragile during training and can "die through a ReLU neuron could cause the weights to update in such a way tha datapoint again. If this happens, then the gradient flowing through the unit That is, the ReLU units can irreversibly die during training since they can g example, you may find that as much as 40% of your network can be "dead" the entire training dataset) if the learning rate is set too high. With a proper frequently an issue.

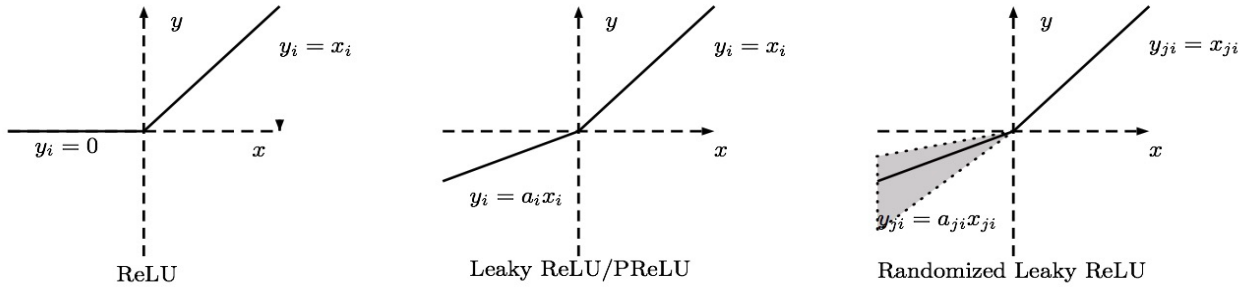
Leaky ReLU



Leaky ReLUs are one attempt to fix the "dying R being zero when $x < 0$, a leaky ReLU will instea so). That is, the function computes $f(x) = \alpha x$ if a small constant. Some people report success w the results are not always consistent.

Parametric ReLU

Nowadays, a broader class of activation functions, namely the ***rectified unit fa*** will talk about the variants of ReLU.



The first variant is called *parametric rectified linear unit* (*PReLU*) [4]. In PReLU, from data rather than pre-defined. He *et al*. [4] claimed that PReLU is the key fac performance on ImageNet classification task. The back-propagation and updatin straightforward and similar to traditional ReLU, which is shown in Page. 43 of th

Randomized ReLU

The second variant is called *randomized rectified linear unit* (*RReLU*). In RReLU, randomized in a given range in the training, and then fixed in the testing. As mer Data Science Bowl (NDSB) competition, it is reported that RReLU could reduce c Moreover, suggested by the NDSB competition winner, the random α_i in training time it is fixed as its expectation, i.e., $2/(l+u) = 2/11$.

In [5], the authors evaluated classification performance of two state-of-the-art CN functions on the *CIFAR-10*, *CIFAR-100* and *NDSB* data sets, which are shown in t *these two networks, activation function is followed by each convolutional layer. A indicates* $1/\alpha$*, where* α *is the aforementioned slopes*.

Activation	Training Error	Test Error
ReLU	0.00318	0.1245
Leaky ReLU, $a = 100$	0.0031	0.1266
Leaky ReLU, $a = 5.5$	0.00362	**0.1120**
PReLU	0.00178	0.1179
RReLU	0.00550	**0.1119**

Table 3. Error rate of CIFAR-10 Network in Network with different activation function

Activation	Training Error	Test Error
ReLU	0.1356	0.429
Leaky ReLU, $a = 100$	0.11552	0.4205
Leaky ReLU, $a = 5.5$	0.08536	**0.4042**
PReLU	0.0633	0.4163
RReLU	0.1141	**0.4025**

Table 4. Error rate of CIFAR-100 Network in Network wit different activation function

From these tables, we can find the performance of ReLU is not the best for all th larger slope α will achieve better accuracy rates. PReLU is easy to overfit on sm smallest, while testing error is not satisfactory), but still outperforms ReLU. In ac other activation functions on NDSB, which shows RReLU can overcome overfittir data than that of CIFAR-10/CIFAR-100. ***In conclusion, three types of ReLU va original ReLU in these three data sets. And PReLU and RReLU seem bette reported similar conclusions in [4]***.

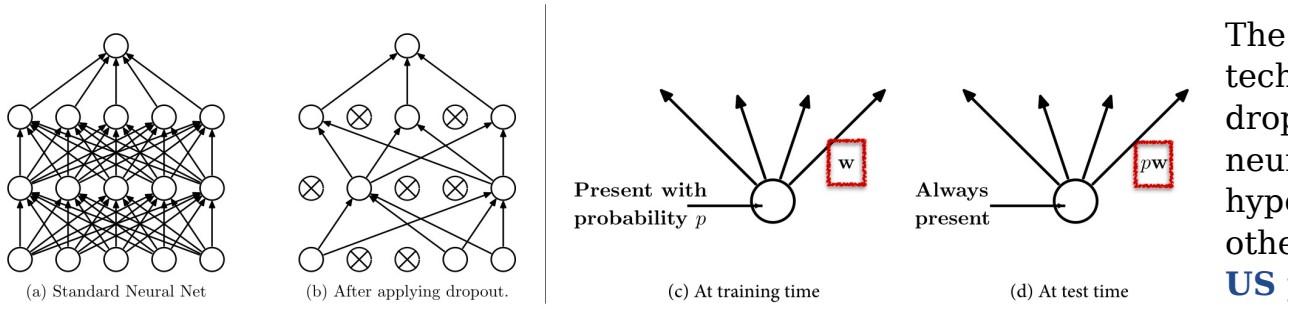
Sec. 6: Regularizations

There are several ways of controlling the capacity of Neural Networks to prevent

- **L2 regularization** is perhaps the most common form of regularization. It c squared magnitude of all parameters directly in the objective. That is, for ev term $\frac{1}{2}\lambda w^2$ to the objective, where λ is the regularization strength. It is com because then the gradient of this term with respect to the parameter w is s regularization has the intuitive interpretation of heavily penalizing peaky w weight vectors.
- **L1 regularization** is another relatively common form of regularization, wh $\lambda|w|$ to the objective. It is possible to combine the L1 regularization with the called Elastic net regularization). The L1 regularization has the intriguing p become sparse during optimization (i.e. very close to exactly zero). In other

end up using only a sparse subset of their most important inputs and becom
In comparison, final weight vectors from L2 regularization are usually diffus
not concerned with explicit feature selection, L2 regularization can be expe
L1.

- **Max norm constraints**. Another form of regularization is to enforce an abs
the weight vector for every neuron and use projected gradient descent to er
corresponds to performing the parameter update as normal, and then enfor
weight vector $\vec{w}$ of every neuron to satisfy $\| \vec{w} \|_2 < c$. Typical values of c are
improvements when using this form of regularization. One of its appealing p
"explode" even when the learning rates are set too high because the update
- **Dropout** is an extremely effective, simple and recently introduced regulariz
that complements the other methods (L1, L2, maxnorm). During training, dr
Neural Network within the full Neural Network, and only updating the para
the input data. (However, the exponential number of possible sampled netw
share the parameters.) During testing there is no dropout applied, with the
prediction across the exponentially-sized ensemble of all sub-networks (mor
practice, the value of dropout ratio $p = 0.5$ is a reasonable default, but this



(a) Standard Neural Net (b) After applying dropout. (c) At training time (d) At test time

The
tech
drop
neu
hyp
othe
US

Sec. 7: Insights from Figures

Finally, from the tips above, you can get the satisfactory settings (e.g., data proc
etc.) for your own deep networks. During training time, you can draw some figur
effectiveness.

- As we have known, the learning rate is very sensitive. From Fig. 1 in the fol
cause a quite strange loss curve. A low learning rate will make your training
large number of epochs. In contrast, a high learning rate will make training
will also drop into a local minimum. Thus, your networks might not achieve
good learning rate, as the red line shown in Fig. 1, its loss curve performs s
performance.
- Now let's zoom in the loss curve. The epochs present the number of times fo
there are multiple mini batches in each epoch. If we draw the classification

performs like Fig. 2. Similar to Fig. 1, if the trend of the loss curve looks too is low; if it does not decrease much, it tells you that the learning rate might curve is related to the batch size. If the “width” looks too wide, that is to say large, which points out you should increase the batch size.

- Another tip comes from the accuracy curve. As shown in Fig. 3, the red line line is the validation one. When the validation accuracy converges, the gap will show the effectiveness of your deep networks. If the gap is big, it indica accuracy on the training data, while it only achieve a low accuracy on the va model overfits on the training set. Thus, you should increase the regularizat no gap meanwhile at a low accuracy level is not a good thing, which shows that case, it is better to increase the model capacity for better results.

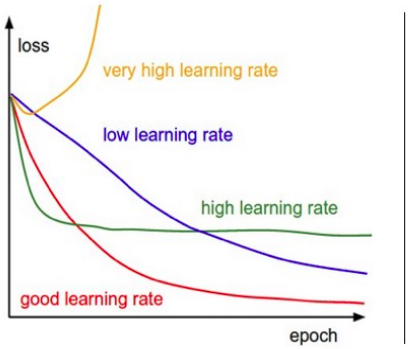


Figure 1

Figure 2

Sec. 8: Ensemble

In machine learning, ensemble methods [8] that train multiple learners and then of-the-art learning approach. It is well known that an ensemble is usually signific and ensemble methods have already achieved great success in many real-world t challenges or competitions, almost all the first-place and second-place winners u

Here we introduce several skills for ensemble in the deep learning scenario.

- **Same model, different initialization**. Use cross-validation to determine t multiple models with the best set of hyperparameters but with different ran approach is that the variety is only due to initialization.
- **Top models discovered during cross-validation**. Use cross-validation to then pick the top few (e.g., 10) models to form the ensemble. This improves danger of including suboptimal models. In practice, this can be easier to pe retraining of models after cross-validation. Actually, you could directly selec from Caffe Model Zoo to perform ensemble.
- **Different checkpoints of a single model**. If training is very expensive, so

taking different checkpoints of a single network over time (for example afte ensemble. Clearly, this suffers from some lack of variety, but can still work r of this approach is that is very cheap.

- **Some practical examples**. If your vision tasks are related to high-level ima still images, a better ensemble method is to employ multiple deep models tr different and complementary deep representations. For example in the Cult associated with ICCV'15, we utilized five different deep models trained on i the cultural images supplied by the competition organizers. After that, we e features and treat them as multi-view data. Combining "early fusion" and "l achieved one of the best performance and ranked the 2nd place in that chal the *Stacked NN* framework to fuse more deep networks at the same time.

Miscellaneous

In real world applications, the data is usually **class-imbalanced**: some classes h instances, while some have very limited number of images. As discussed in a rec CNNs are trained on these imbalanced training sets, the results show that imbal severely negative impact on overall performance in deep networks. For this issue training data by directly up-sampling and down-sampling the imbalanced data, w solution is one kind of special crops processing in our challenge solution [7]. Bec are imbalanced, we merely extract crops from the classes which have a small nu hand can supply diverse data sources, and on the other hand can solve the class- adjust the fine-tuning strategy for overcoming class-imbalance. For example, you parts: one contains the classes which have a large number of training samples (i classes of limited number of samples. In each part, the class-imbalanced problem beginning of fine-tuning on your data set, you firstly fine-tune on the classes whi samples (images/crops), and secondly, continue to fine-tune but on the classes w

References & Source Links

1. A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with I *NIPS*, 2012
2. A Brief Overview of Deep Learning, which is a guest post by Ilya Sutskever.
3. CS231n: Convolutional Neural Networks for Visual Recognition of *Stanford* Andrej Karpathy.
4. K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Classification. In *ICCV*, 2015.
5. B. Xu, N. Wang, T. Chen, and M. Li. Empirical Evaluation of Rectified Activa *Deep Learning Workshop*, 2015.
6. N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdino Neural Networks from Overfitting. *JMLR*, 15(Jun):1929−1958, 2014.

7. X.-S. Wei, B.-B. Gao, and J. Wu. Deep Spatial Pyramid Ensemble for Cultural *Looking at People Workshop*, 2015.
8. Z.-H. Zhou. Ensemble Methods: Foundations and Algorithms. *Boca Raton, F* 978-1-439-830031)
9. M. Mohammadi, and S. Das. S-NN: Stacked Neural Networks. Project in *Sta*
10. P. Hensman, and D. Masko. The Impact of Imbalanced Training Data for Cor *Project in Computer Science*, DD143X, 2015.
