Day 7: Convolutional Neural Networks

Summer STEM: Machine Learning

Department of Electrical and Computer Engineering
NYU Tandon School of Engineering
Brooklyn, New York

August 11, 2020

NYU TANDON SCHOOL OF ENGINEERING

Outline

Better performance with images

- Encoding locality
- How does an MLP see an image?
- Is this how we see images?

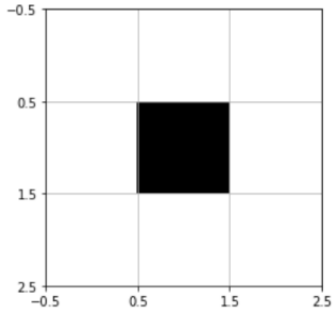
Examples: Lena & Mandrill

Outline

Images in Computer

- Images are stored as arrays of quantized numbers in computers
- Gray scale image: 2D matrices with each entry specifying the intensity (brightness) of a pixel
  - Pixel values range from 0 to 255, 0 being the darkest, 255 being the brightest

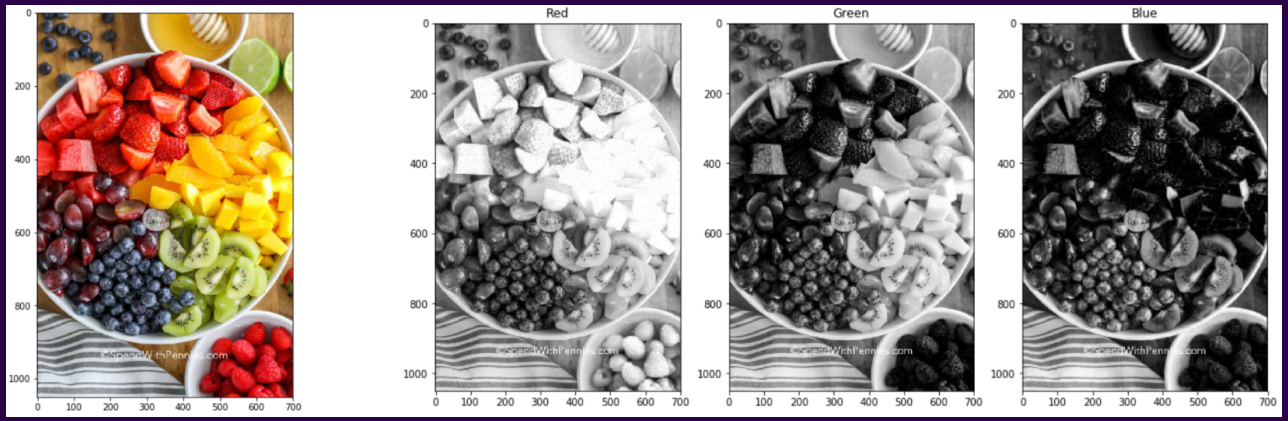
```
[[255 255 255]
 [255   0 255]
 [255 255 255]]
```



NYU TANDON SCHOOL OF ENGINEERING

Color Images

- Color image: 3D array, 2 dimensions for space, 1 dimension for color
  - Can be thought of as three 2D matrices stacked together into a cube, each 2D matrix specify the amount of each color: Red ,Green ,Blue value at each pixel



- Shape of this image: (1050,700,3)
- There are 1050x700 pixels, 3 channels: R,G,B

Outline

Limitations of Fully Connected Network

- Open `lab_mnist.ipynb`
- In MNIST, we used a fully connected network, in which each neuron in the hidden layer is connected to all 28x28 = 784 pixels.
- Higher definition images often contain millions of pixels → It is not practical to use fully connected networks.
- Fully connected networks treat each individual pixel as a feature, it does not utilize the positional relationship between pixels

Convolution

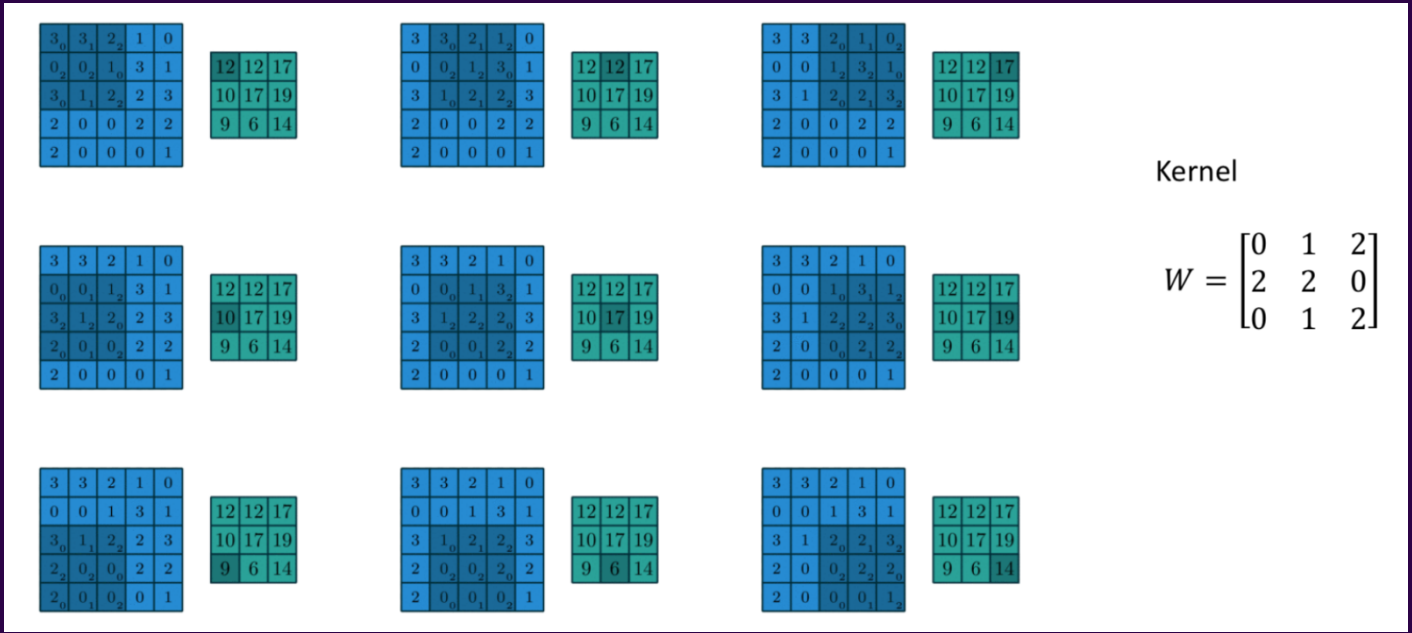
- Introducing a new operation: Convolution

- An operation on an image(matrix) X with a kernel W
- $Z = X \circledast W$

Some Animations, Source: https://towardsdatascience.com

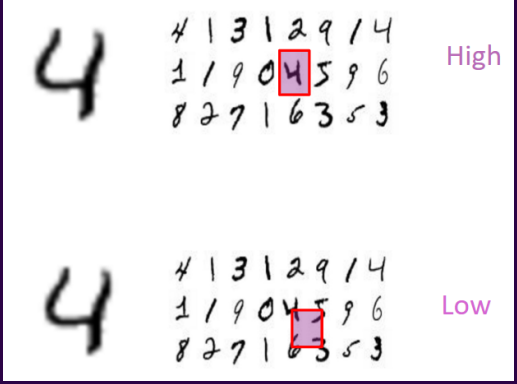
Some Animations, Source: https://cs231n.github.io/convolutional-networks/

Example of a Convolultion



Why Convolution?

- With convolution, each output pixel depends on only the neighboring pixels in the input
- This allows us to learn the positional relationship between pixels
- Use of different kernels allows us to detect features



Understand Kernels

Open `demo_kernels.ipynb`

Review: Convolution and Kernels

How do we represent images in computers?

As matrices (greyscale)

3D arrays / Tensor

- Muti-dimensional arrays / Tensors

1D: vectors — to access an element i

2D: matrices — (i, j)

3D: Tensors — (i, j, k)

Image X | Kernel W

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix} \circledast \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$$

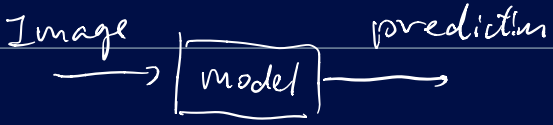
$$X = \begin{bmatrix} x_{11} \\ x_{12} \\ x_{21} \\ x_{22} \end{bmatrix} \quad W = \begin{bmatrix} w_{11} \\ w_{12} \\ w_{21} \\ w_{22} \end{bmatrix}$$

$$X \cdot W = w_{11} x_{11} + w_{12} x_{12} + w_{21} \cdot x_{21} + w_{22} x_{22}$$

weighted sum

Convolution summarizes the information for each region (same size as the kernel) of the image. The pixels that are multiplied by a higer weight have more importance.

→ extract (useful) features



Ex.

Convolution for Multiple Channels

- A kernel for each channel. Could be same kernel, or different
- Perform a convolution for each of the channel, with the respective kernel
- Sum the results

Feature Maps

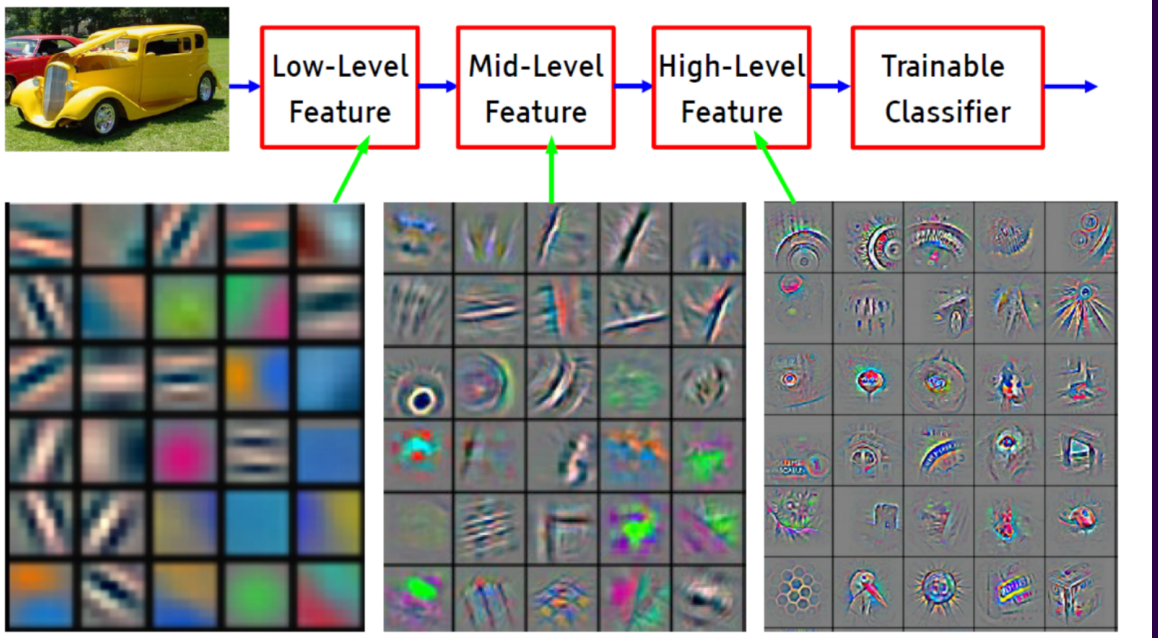


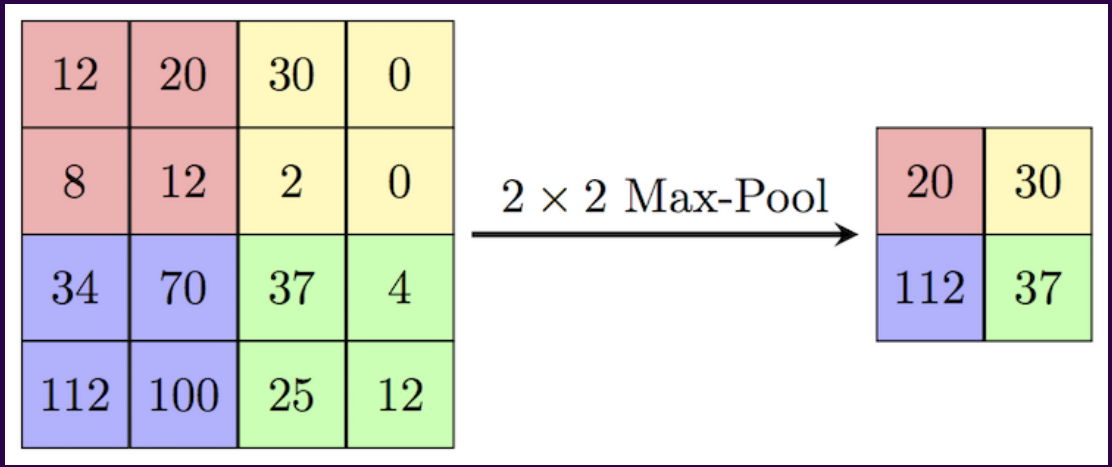
Figure: Figure from Y. LeCun Presentation, "What's wrong with deep learning?". Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013].

Demo: Building a CNN

Open `demo_cnn.ipynb`

Max-Pooling

- Down-samples the inputs
- Provides translation invariance. Why?
- Apply after activation!

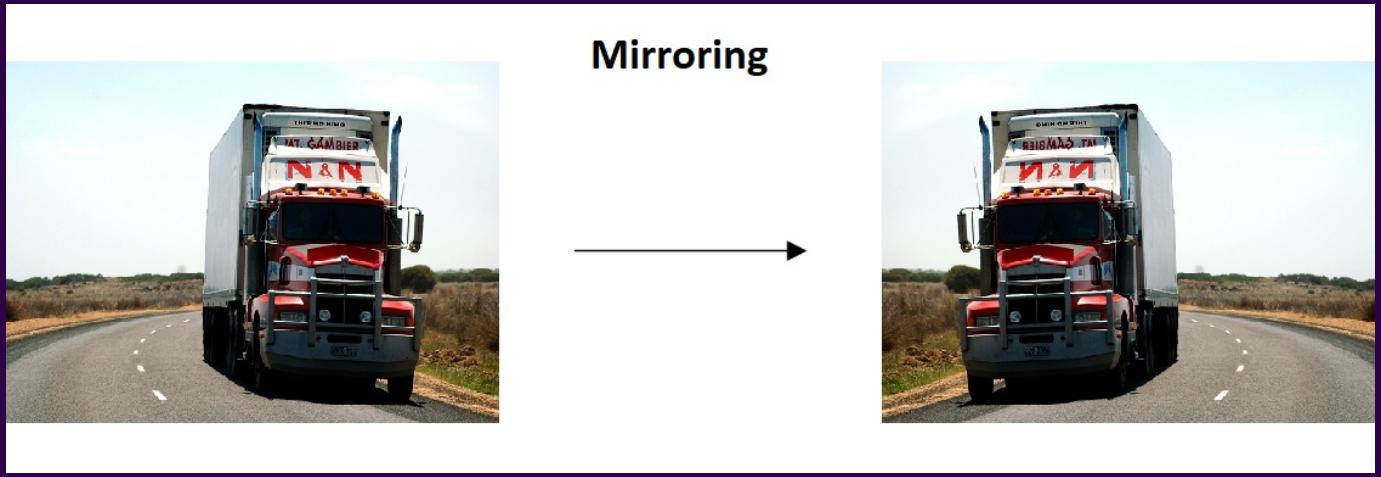


Outline

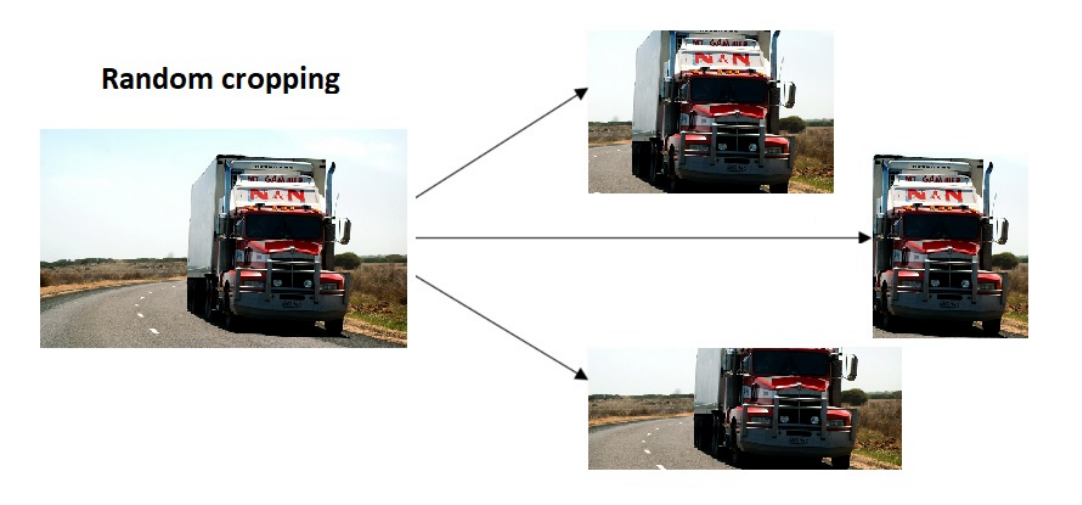
Data Augmentation

- Image classification is a difficult task
- We need more data !
- Labeling is expensive and time-consuming.
- How can we create new images ?

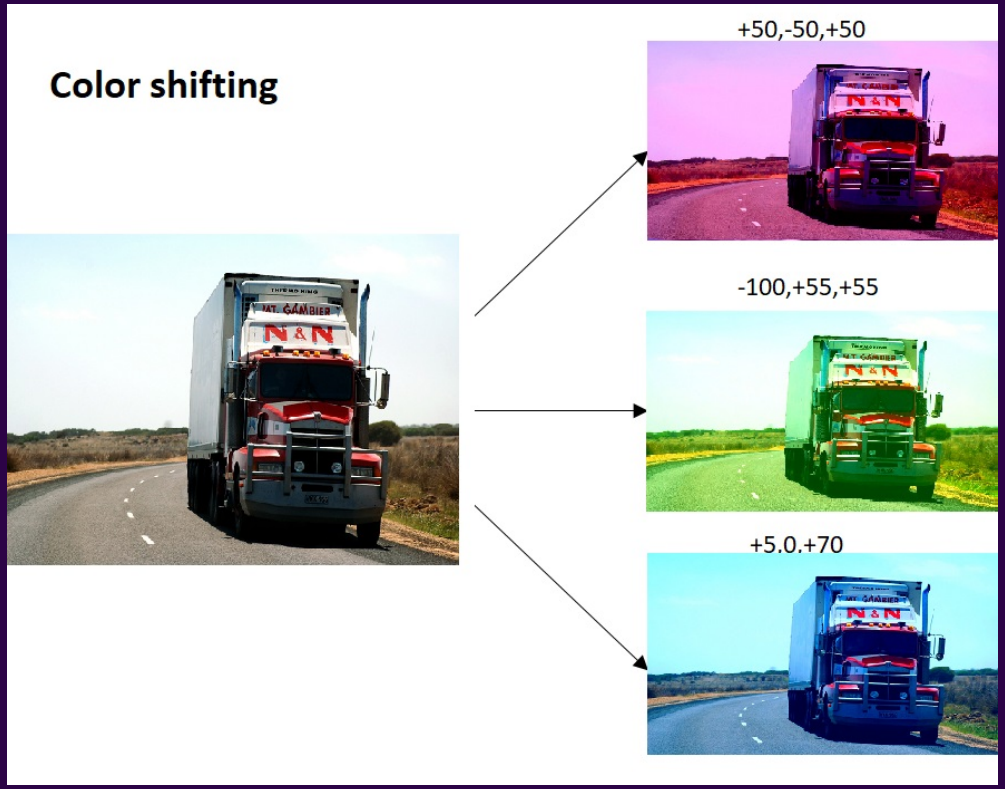
Data Augmentation



Data Augmentation

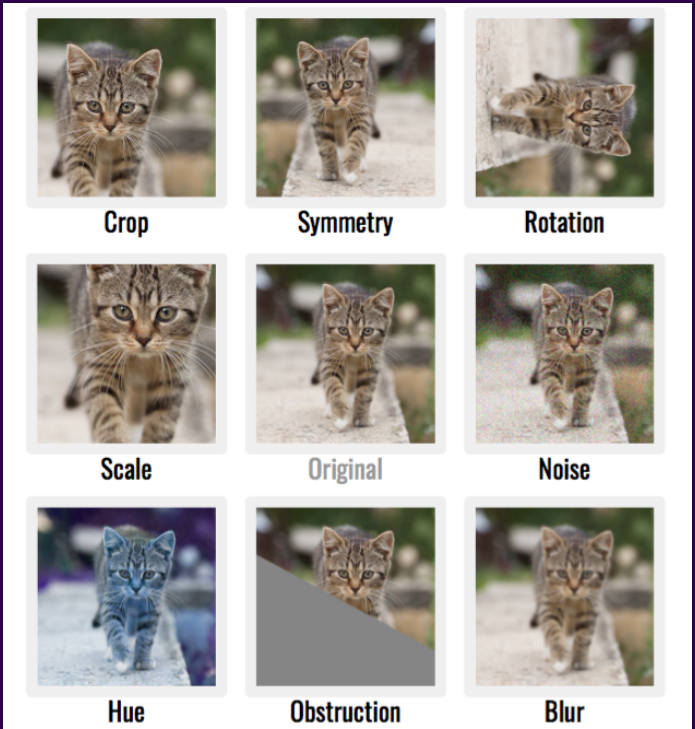


Data Augmentation



http://datahacker.rs/deep-learning-data-augmentation

Data Augmentation



https://medium.com/@wolframalphav1.0/easy-way-to-improve-image-classifier-performance-part-1-mixup-augmentation-with-codes-33288db9

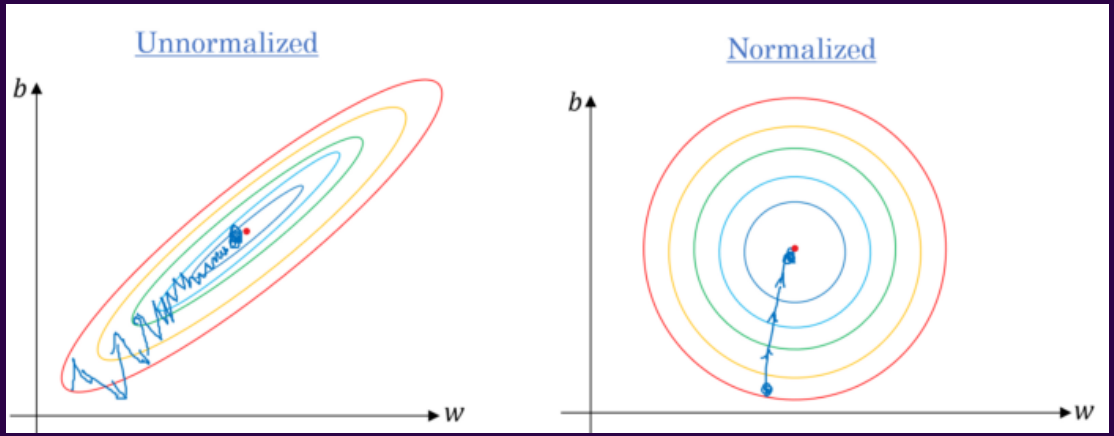
Data Normalization

- Given the dataset (x_i, y_i) for $i = 1, 2, \ldots, N$
- Mean: $\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$
- Variance: $\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2$
- Standard deviation : $\sigma \ (= \sqrt{\sigma^2})$

Data Normalization

- Given the dataset (x_i, y_i) for $i = 1, 2, \ldots, N$
- Mean: $\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$
- Variance: $\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2$
- Standard deviation : σ $(= \sqrt{\sigma^2})$
- **Normalization** : Replace each x_i by $x_i' = \frac{x_i - \bar{x}}{\sigma}$
- The new dataset will have a mean of 0 and a variance of 1.

Data Normalization



https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

Batch Normalization

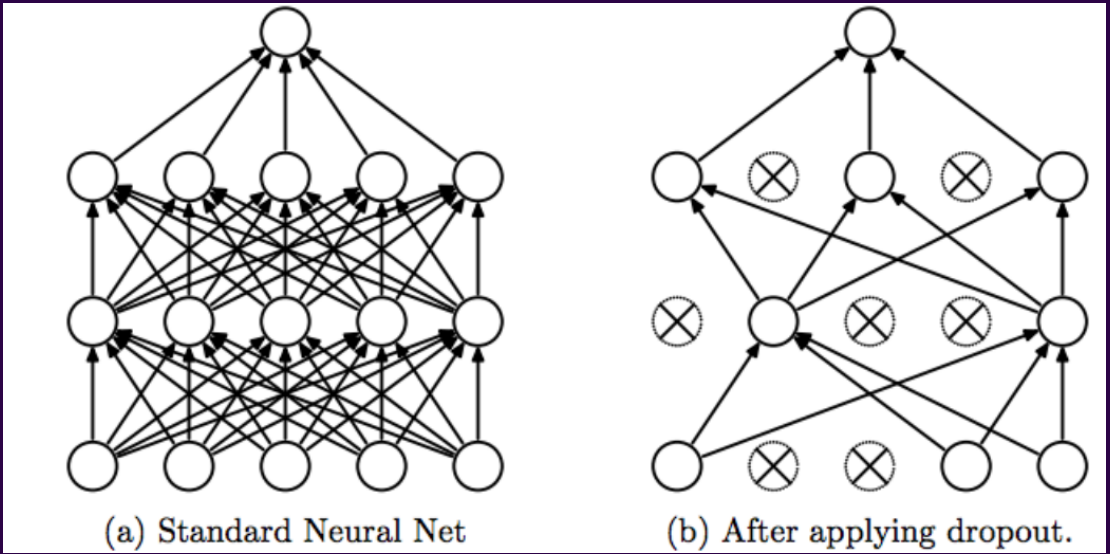
- We normalize the inputs to the network. Why not do that for the inputs to the hidden layers?
- Batch norm: normalize the inputs to a layer for each mini-batch.
- Apply before activation!

Batch Normalization

```
model = models.Sequential()
# More layers
model.add(layers.Conv2D(64, (3, 3)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
# More layers
model.add(layers.Flatten())
model.add(layers.Dense(64))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

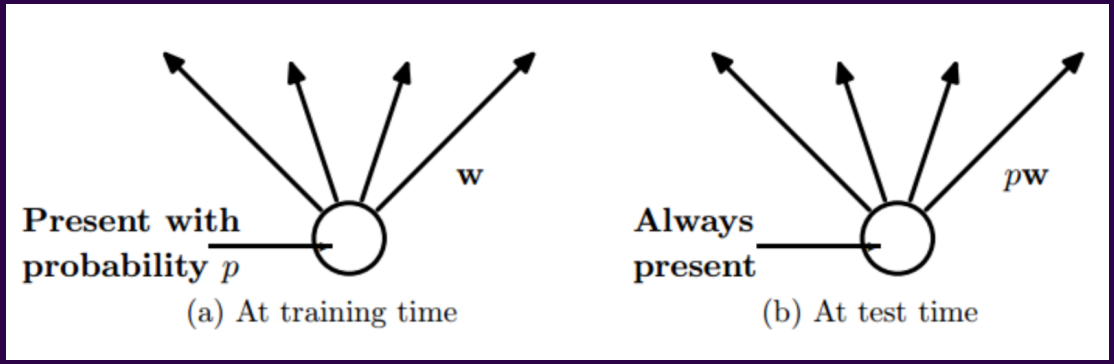
Dropout

- Patented by Google
- Randomly disable neurons and their connections between each other.



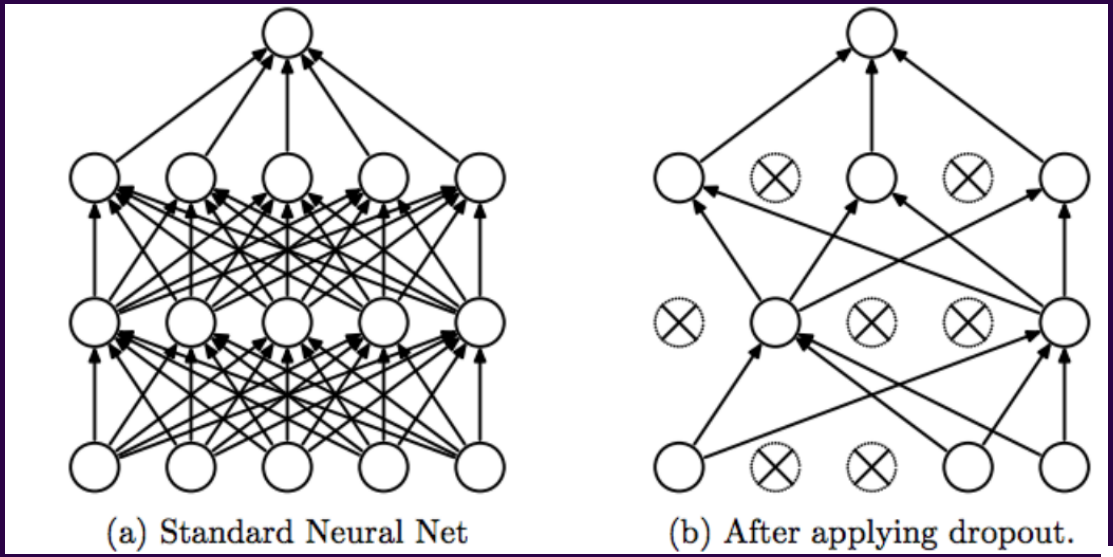
Dropout

- In train mode, each neuron is present with probability p.
- In eval mode, multiply the weights with p.
- Apply after activation!



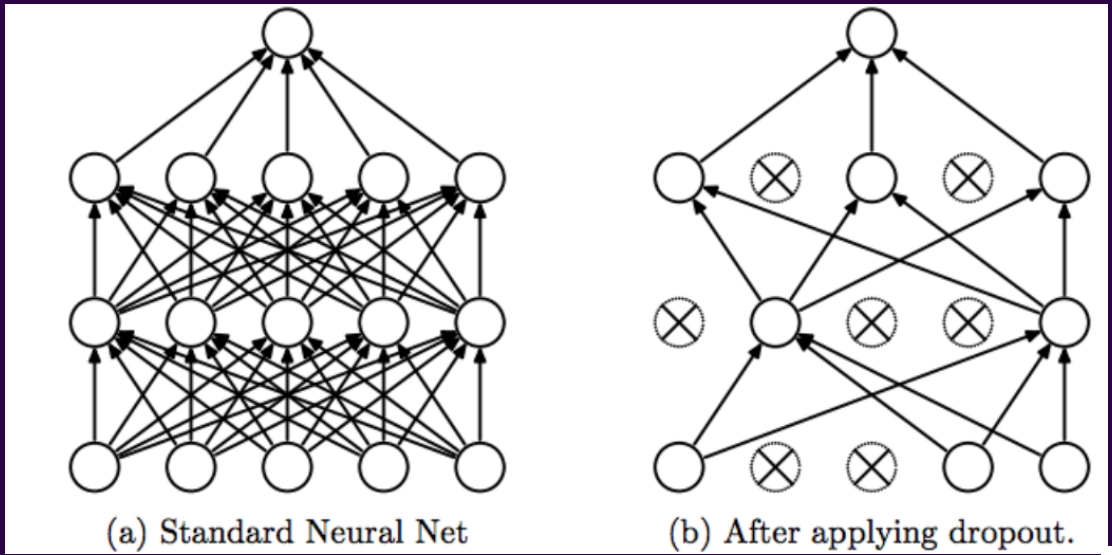
Dropout

- This is the same as using a neural network with the same amount of layers but less neurons per layer.
- The more neurons the more powerful the neural network is, and the more likely it is to overfit.



Dropout

- This also means that the model can not rely on any single feature, therefore would need to spread out the weights.
- When spreading out the weights the size also shrinks, thus giving a similar effect to L2 regularization



Outline

Open Source Implementation

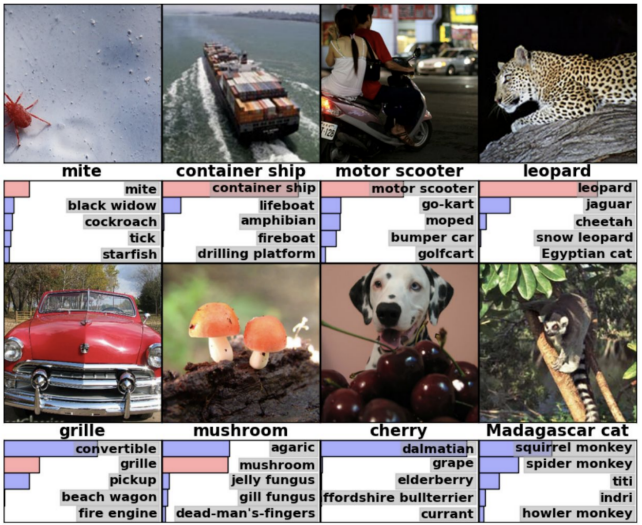
- If you are interested in building on top of a deep learning research paper it is a good idea to first go online and see if there is a nice open source implementation instead of starting from scratch.
- If you are in the field of computer vision, many networks would require an extensive hyperparameter search and multiple GPUs to train, this process might take weeks.
- People now open source their results along with the weights and these can be seen as a nice initialization to your application.

Imagenet Challenge

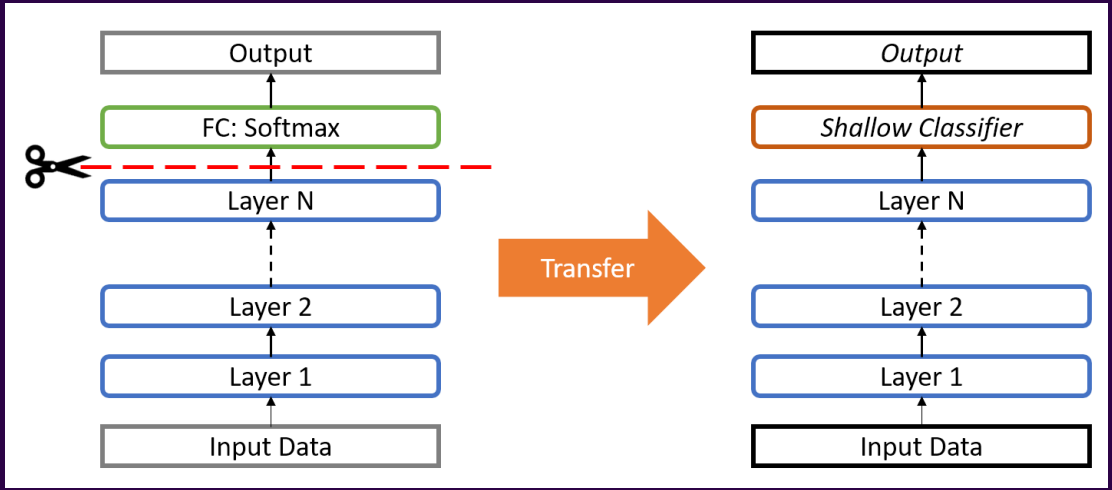
ImageNet Challenge

IMAGENET

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.



Transfer Learning



https://www.oreilly.com/library/view/hands-on-transfer-learning/9781788831307/d94586c6-1c46-4794-aded-22442a4f81d8.xhtml

Transfer Learning

- You can freeze the early layers and replace the last few layers to match your own application needs (e.g. different number of classes, different activation functions).
- Only train the replaced layers and use the weights of the early layers "as-is".
- This is similar to transferring the knowledge from one network to another, thus the name transfer learning.

Lab: Cats vs. Dogs

Open `lab_cats_vs_dogs.ipynb`