Practical Deep Learning with pytorch

Jakub Cieslik - Senior Data Scientist @ NewYorker (we are hiring DS's and DE's on all levels)

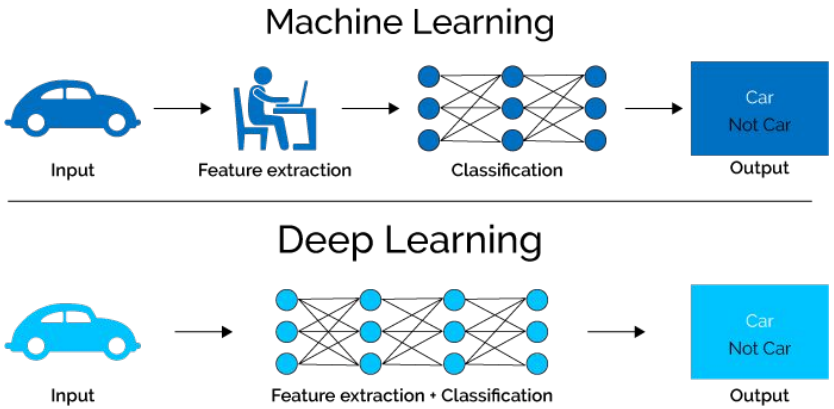
i008@github

Repo for this course:
https://github.com/i008/dsr-pytorch

What is Deep Learning

1. Deep Learning means using a neural network with several layers of nodes between input and output

2. The series of layers between input & output do feature identification and processing in a series of stages just as our brains seem to.



Learning Deep Learning

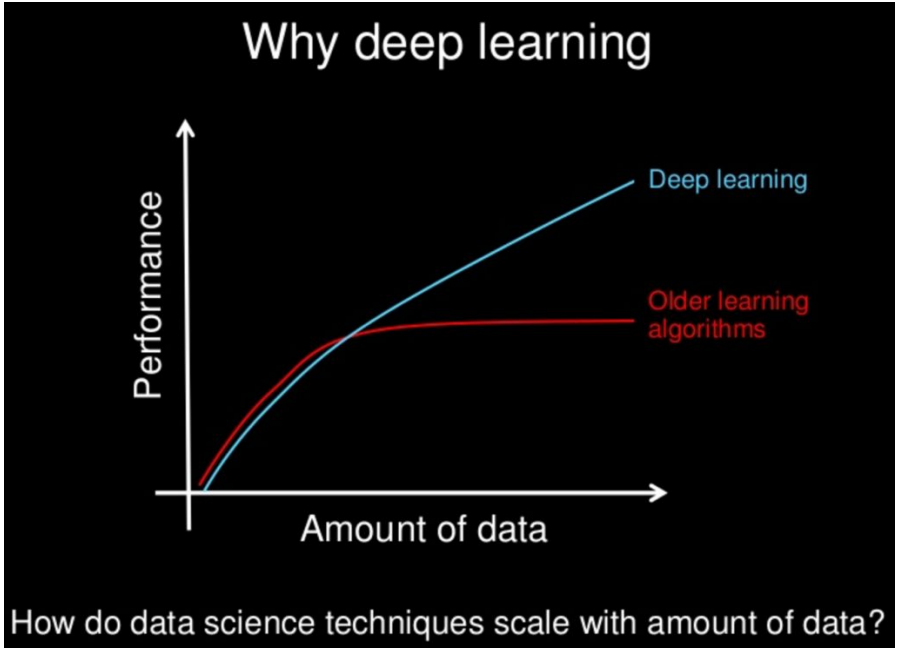
- https://pytorch.org/tutorials/beginner/nn_tutorial.html
- https://www.udacity.com/course/deep-learning-pytorch--ud188
- https://www.d2l.ai/
- http://deeplearning.ai (Andrew NG, The Batch)
- ML Yearning https://d2wvfoqc9gyqzf.cloudfront.net/content/uploads/2018/09/Ng-MLY01-13.pdf
- https://www.youtube.com/playlist?list=PLkt2uSq6rBVctENoVBg1TpCC7OQi31AlC (CS231N by Karpathy)
- AI-Notes https://www.deeplearning.ai/ai-notes/
- https://www.fast.ai/ v2
- https://machinelearningmastery.com/better-deep-learning/
- https://medium.com/@jonathan_hui/index-page-for-my-articles-in-deep-learning-19821810a14
- http://www.kaggle.com
- http://www.arxiv-sanity.com/

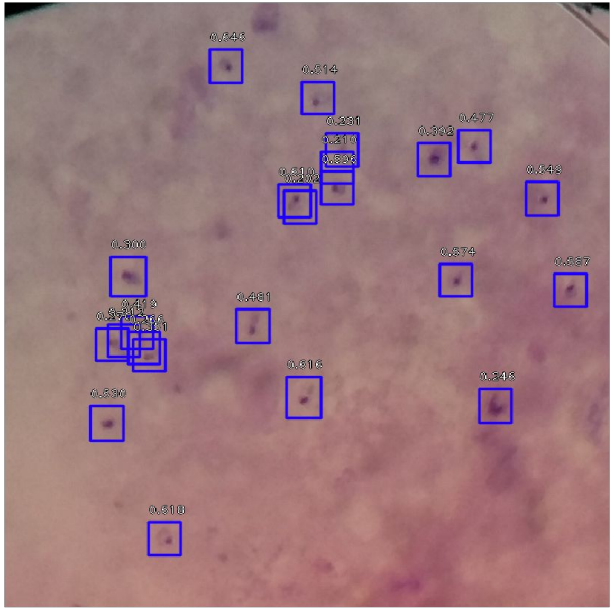
Do i need to have access to a GPU?

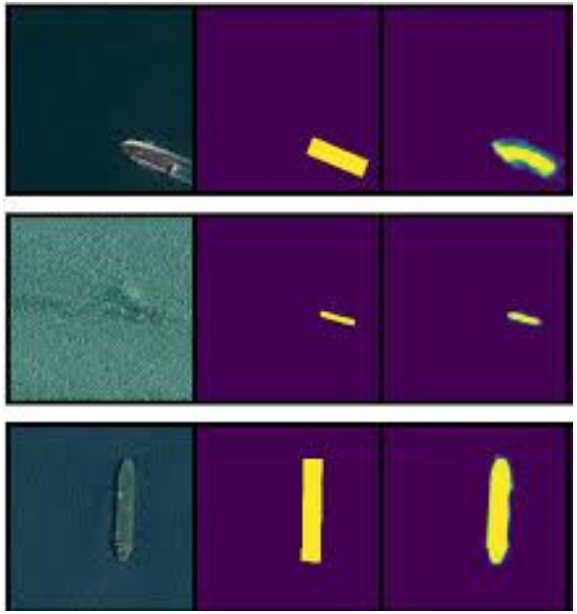
- https://course.fast.ai/start_salamander.html
- https://www.floydhub.com/
- https://colab.research.google.com
- Google Cloud / AWS

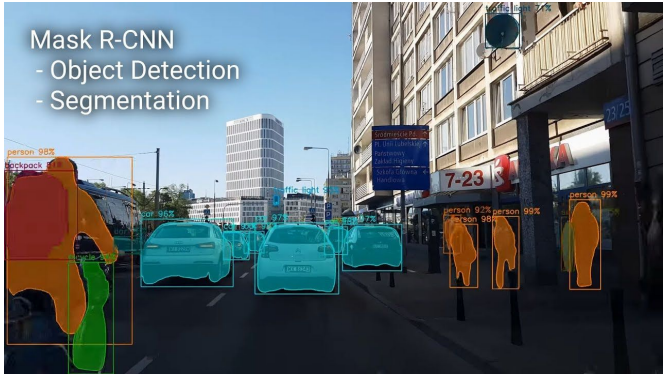
Why Deep Learning

- Computer Vision
  - self driving cars
  - medical diagnosis
  - surveillance
  - identification
  - digitizing documents
  - image search
  - Image recognition
- NLP
  - translation
  - speech2text, text2speech
  - sentiment analysis
  - chatbots








Mask R-CNN
- Object Detection
- Segmentation

Mind Blowing Applications

Image captioning

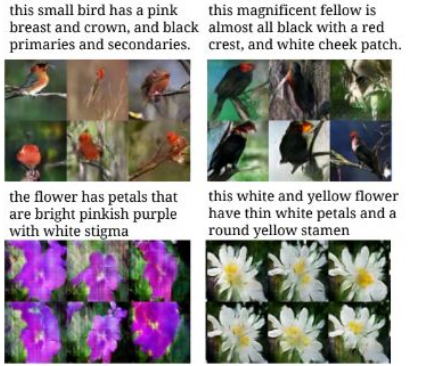
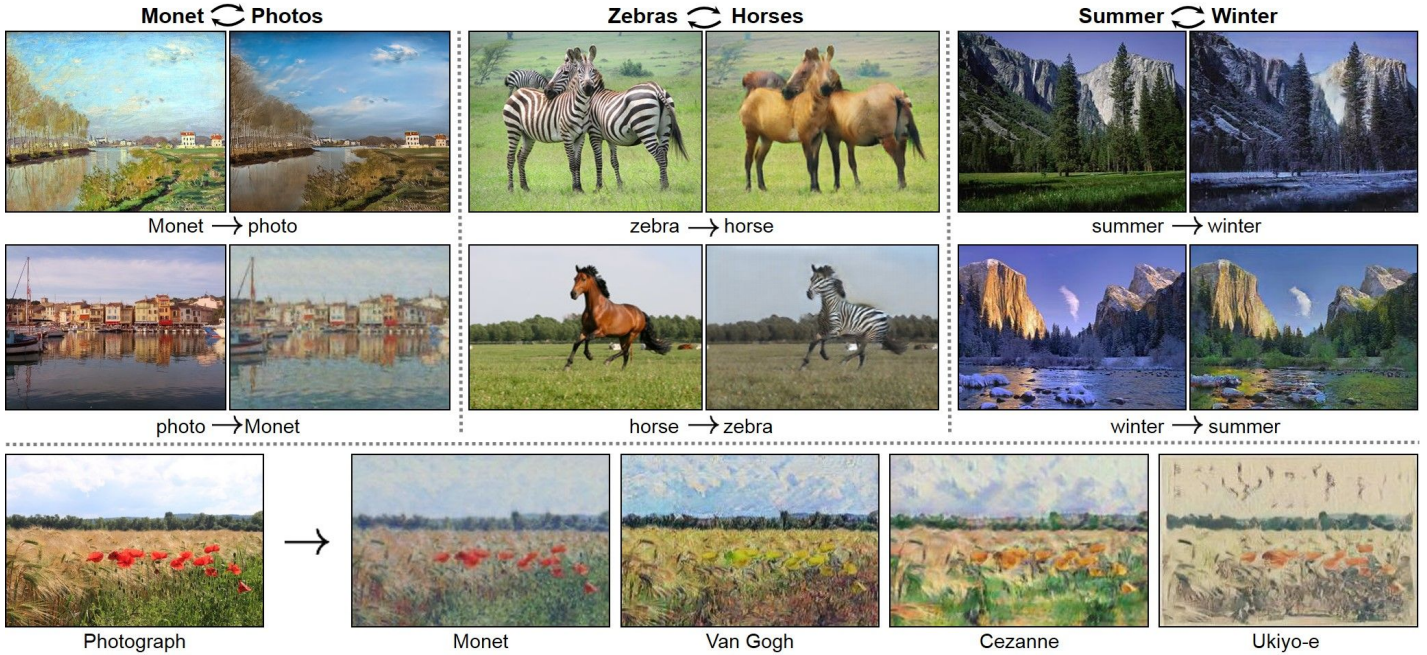


Figure 1. Examples of generated images from text descriptions. Left: captions are from zero-shot (held out) categories, unseen text. Right: captions are from the training set.

text to image

pixel to pixel

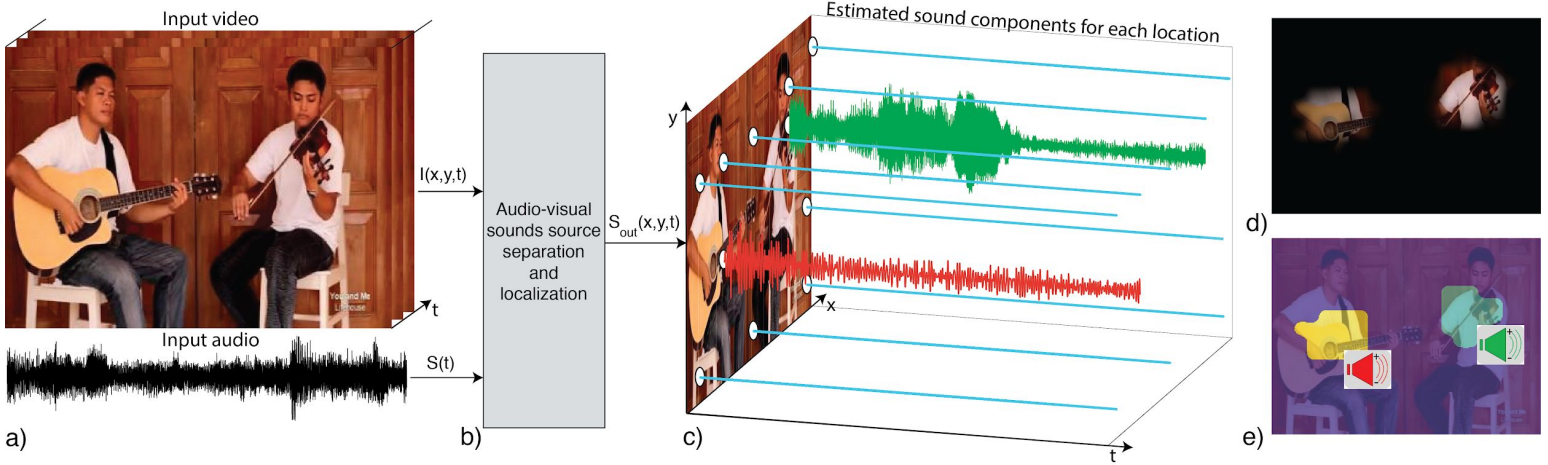
Cycle GAN


Monet ⇄ Photos
Zebras ⇄ Horses
Summer ⇄ Winter
Monet → photo
zebra → horse
summer → winter
photo → Monet
horse → zebra
winter → summer
Photograph
Monet
Van Gogh
Cezanne
Ukiyo-e

GauGan

The sound of pixels

- http://sound-of-pixels.csail.mit.edu/



Voice Style Transfer

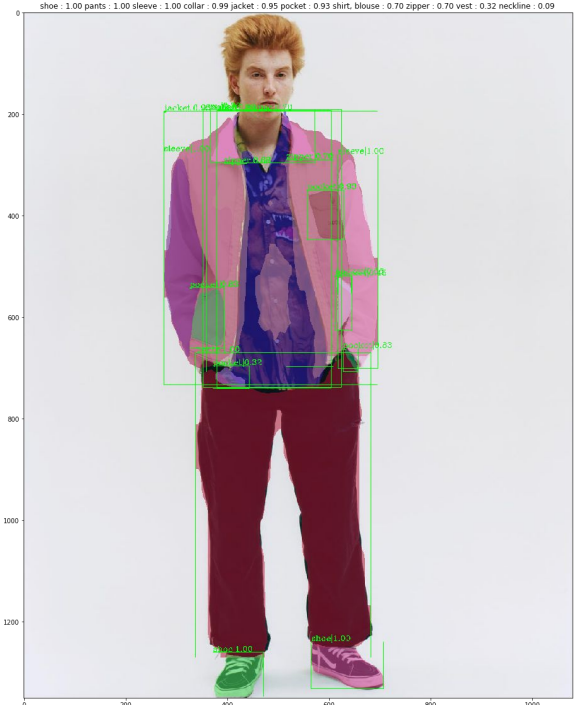
Text to Image / Object Detection

long sleeve shirt black. spread collar. zip closure vertical zippered welt pockets

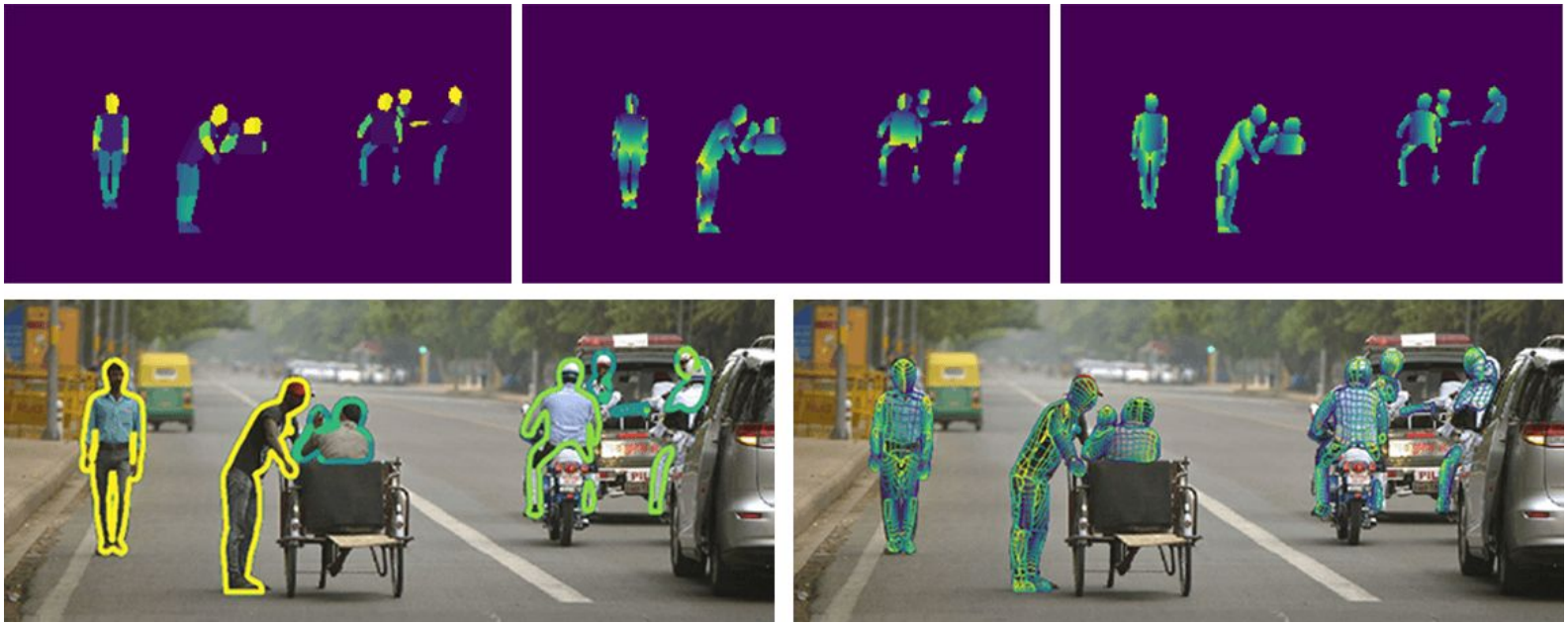
slim-fit jog jeans blue. fade effect distressing throughout. five-pocket styling. belt loop

singlet tank top white . ribbed collar cuff. logo print gray chest . black

short sleeve cotton piqueacute polo purple. ribbed spread collar sleeve cuffs .



Deeppose (3D)

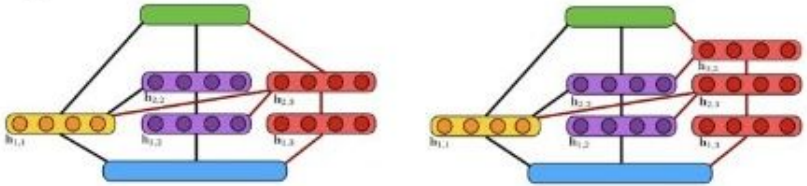


Depth from Videos in the Wild

AdaNET

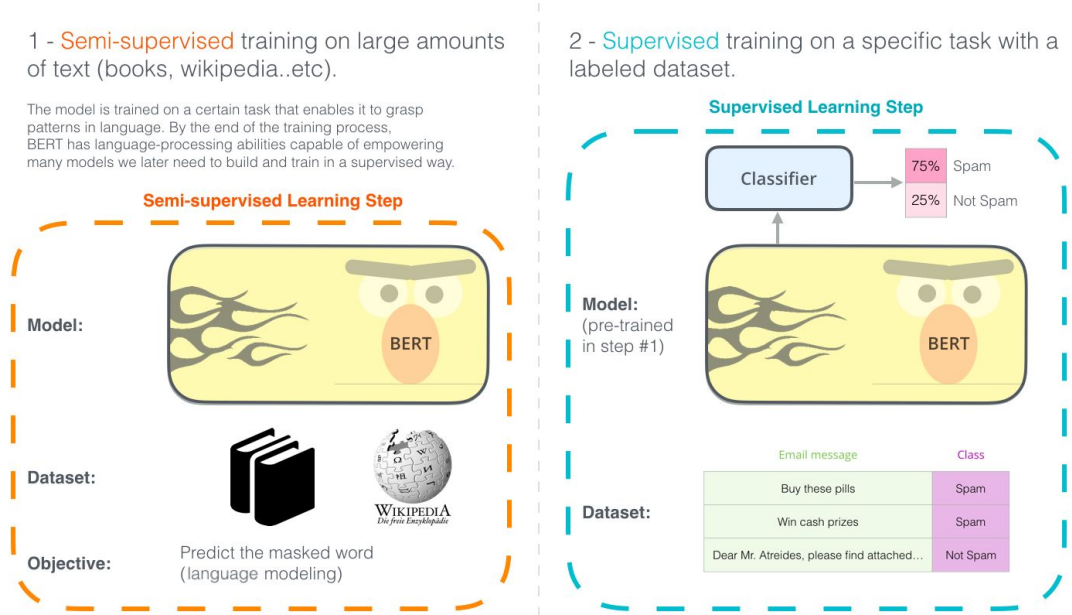
AdaNet

- Incremental construction: At each round, the algorithm adds a subnetwork to the existing neural network;



- Algorithm leverages embeddings previous learned;
- *Adaptively* grows network, balancing trade-off between empirical error and model complexity;
- Learning bound: $R(f) \leq \widehat{R}_{S,\rho}(f) + \frac{4}{\rho} \sum_{k=1}^{l} \|\mathbf{w}_k\|_1 \mathfrak{R}_m(\mathcal{H}_k) + \tilde{\mathcal{O}}\left(\frac{1}{\rho}\sqrt{\frac{\log(l)}{m}}\right)$

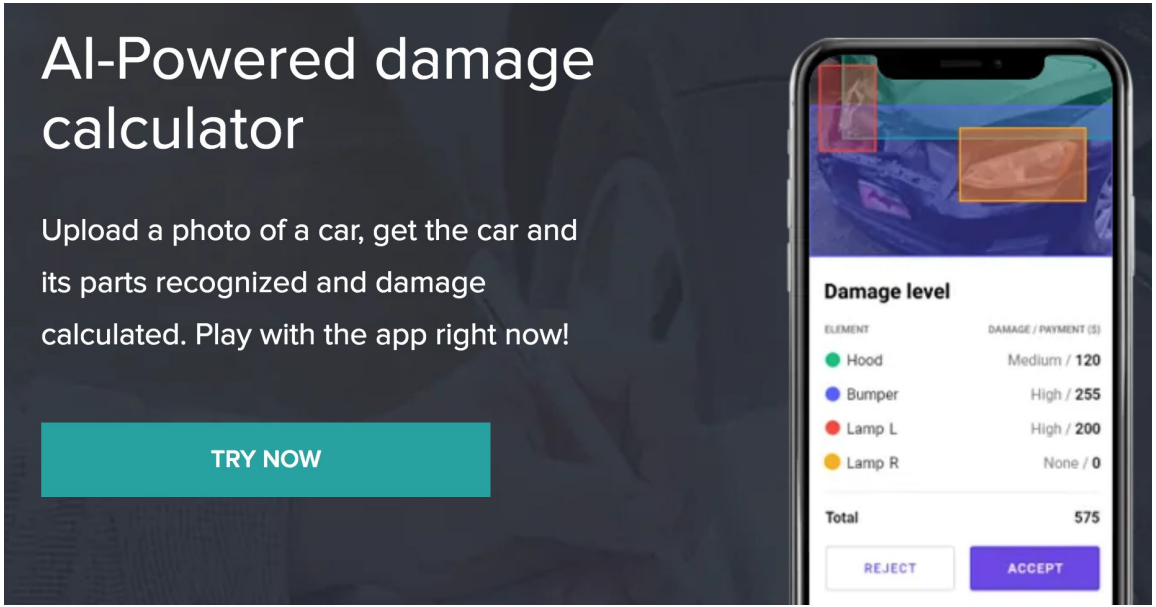
Cutting Edge NLP - Bert



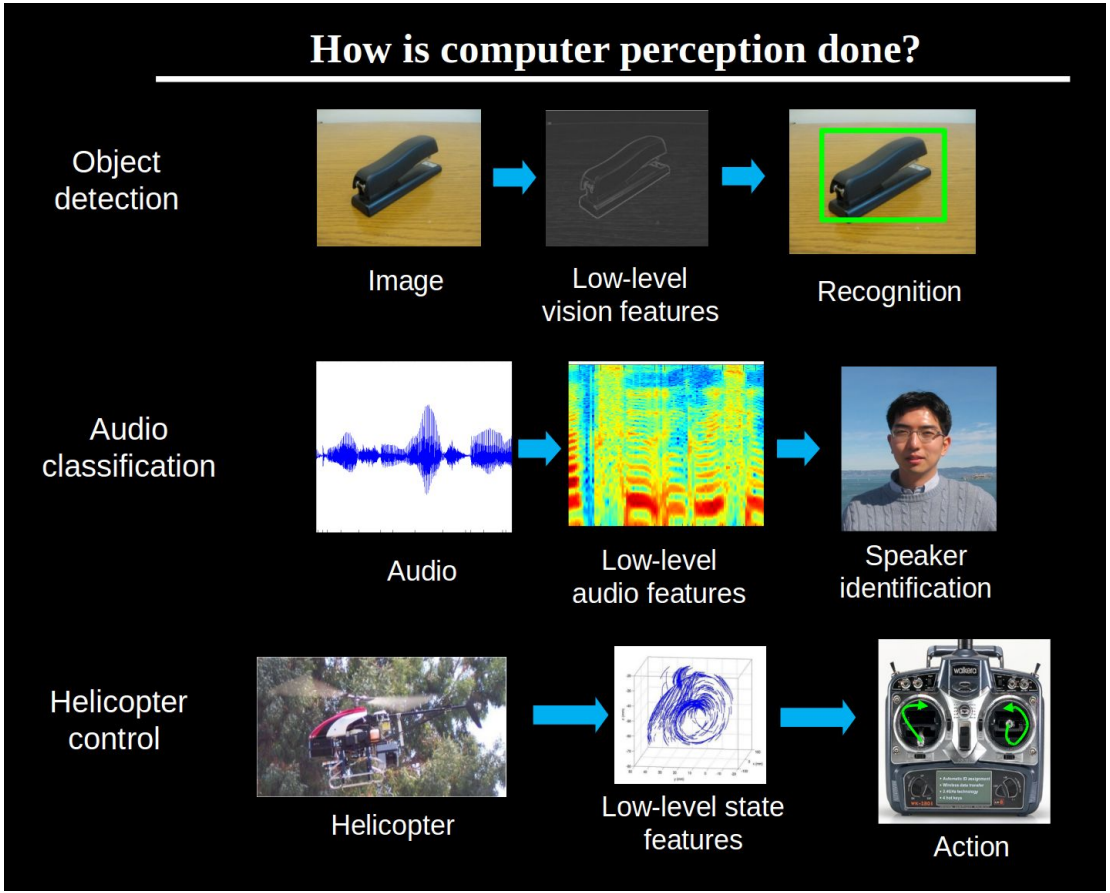
As opposed to directional models, which read the text input sequentially (left-to-right or right-to-left), the Transformer encoder reads the entire sequence of words at once. Therefore it is considered bidirectional, though it would be more accurate to say that it's non-directional. This characteristic allows the model to learn the context of a word based on all of its surroundings (left and right of the word).

Playing Games SC/DoTA

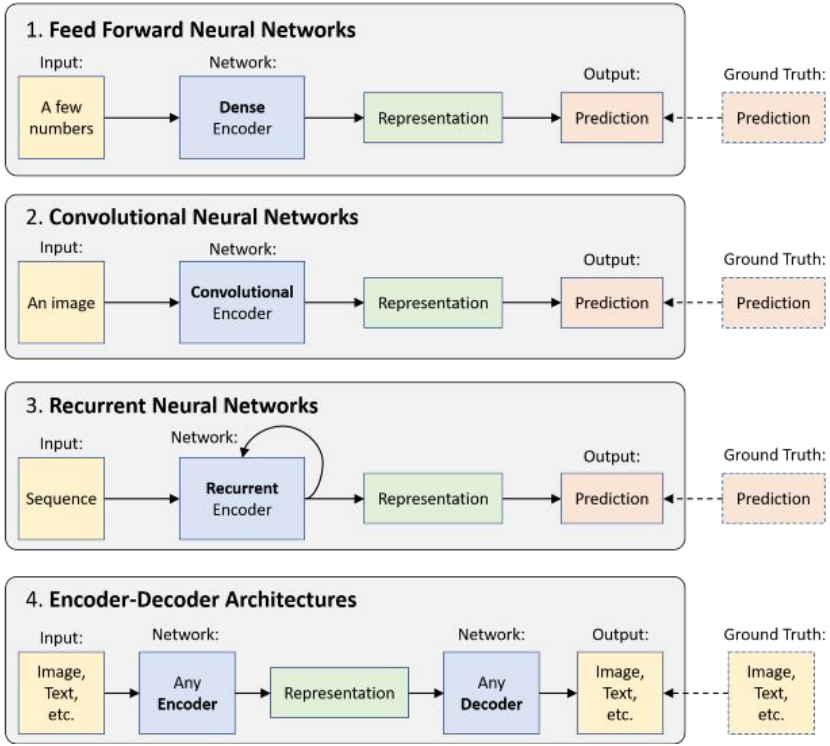
Car Damage Estimation



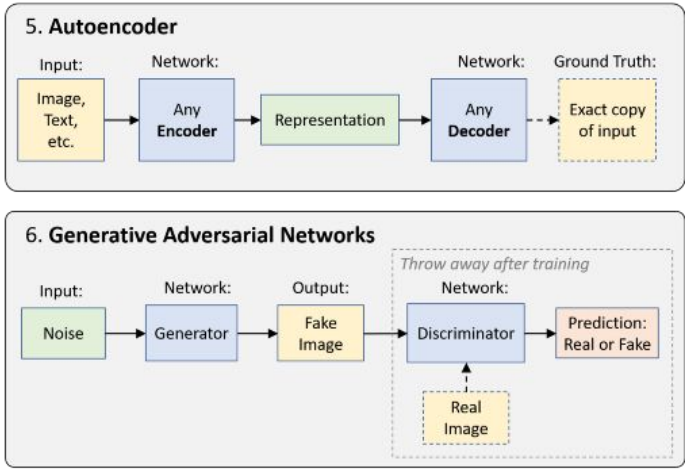
How is computer perception done?



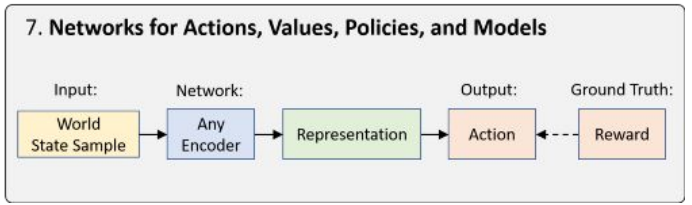
Supervised Learning



Unsupervised Learning



Reinforcement Learning

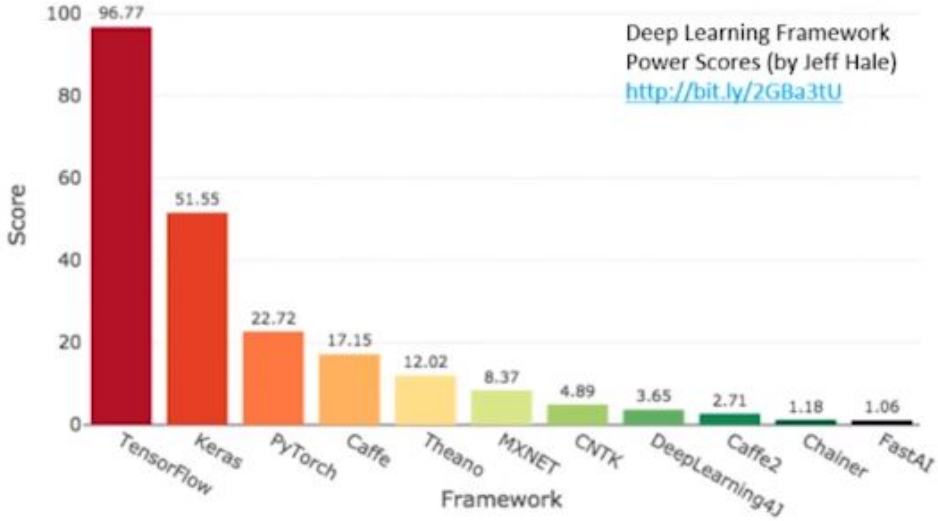


The Secret


kaggle
days
KAGGLE DAYS PARIS, JAN 25-26TH 2019
PAVEL OSTYAKOV

Deep Learning Frameworks

- Pytorch (facebook)
- Tensorflow (google)
- MXNet (amazon)
- Chainer
- CNTK (microsoft)



Why do we teach pytorch

- Growing very fast and it's becoming one of the most popular frameworks
- Very pythonic, writing pytorch code feels very natural to python programmers
- Eager mode (simple, debuggable, hackable, numpy-like)
- Script Mode (v1.0+) (production, no python dependency, optimizable)
- It requires basic knowledge about neural nets and optimization the process is less magical compared to keras for instance.

There are easier ways to use Deep Learning

- Keras
- Fast.ai (build on top of fastai)
- Skorch (medium-level wrapper on top of pytorch)
- Ignite (medium-low level wrapper on top of pytorch

Overall Plan for this workshop

We will Learn about:

- Deep Learning Basics
- Mind Blowing applications of deep learning
- What is Pytorch and how it compares to other DL frameworks
- How do we optimize neural networks (how we teach them stuff)
    - Loss functions
    - optimizers
- Pytorch basics and idioms
- Coding a Perceptron using pytorch
- Learn convolutions and feature representations work
- Learn how data loading is done in pytorch
- Learn how to find the optimal LR with a LR-finder
- Train an image classifier on the Fashion MNIST dataset
- Learn how to use Cosine Annealing with Restarts Scheduling
- Use transfer learning to create a sneaker-not-sneaker classifier
- Train a UNET type network to segment medical images

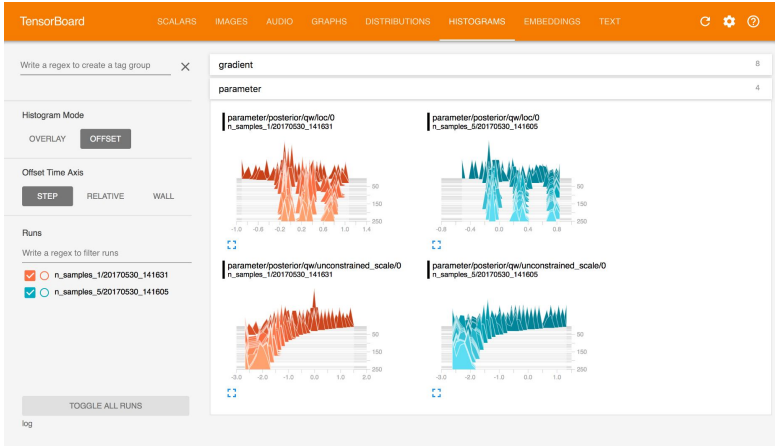
Local Setup

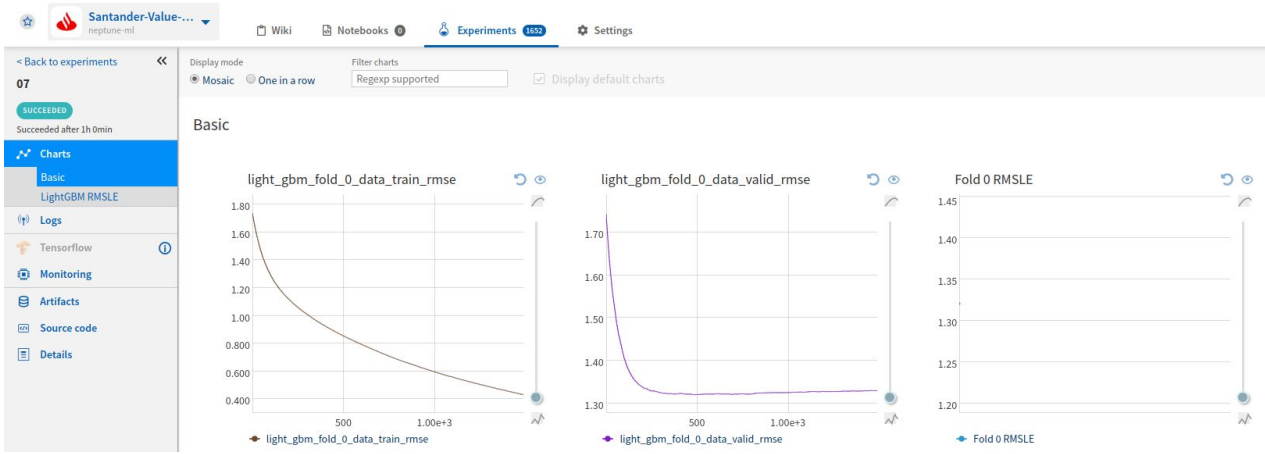
Environment:

- Python 3.6+ (https://www.anaconda.com/download)
- Pytorch 1.0+ (http://pytorch.org)
- Jupyter notebook (Should come with anaconda installation)
- Other packages we will use throughout the course (please install them):
    - easyimages
    - pretrainedmodels
    - seaborn

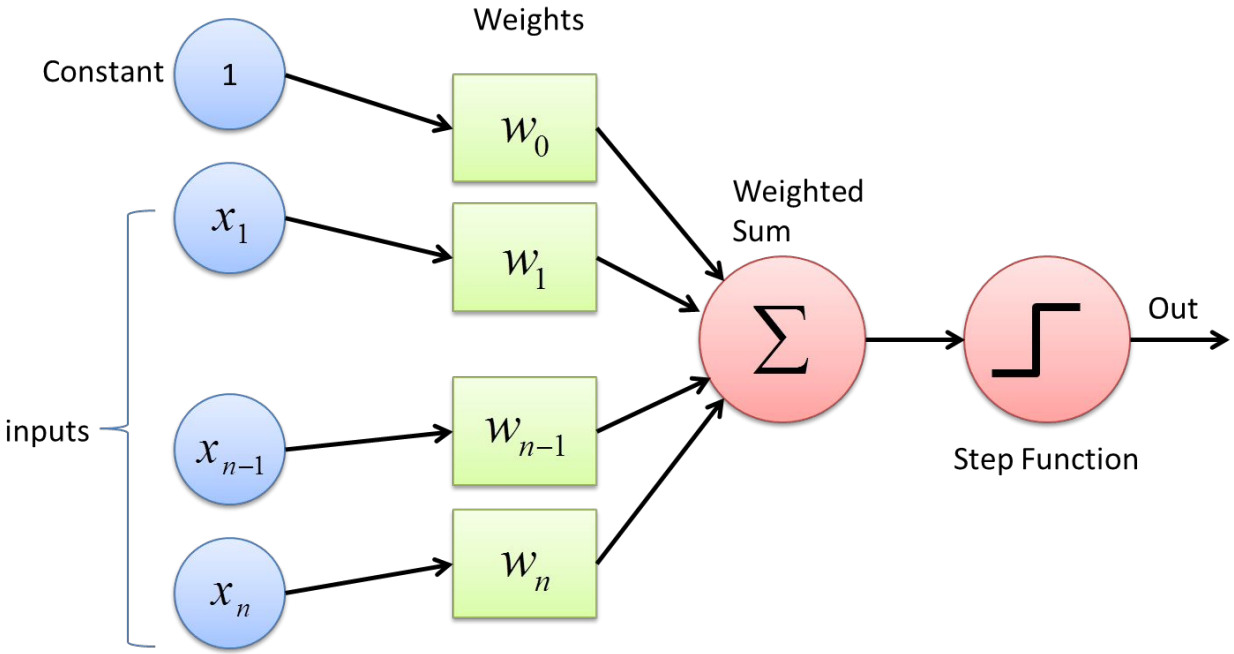
Experiment tracking

- Tensorboard
- neptune.ml
- Many others

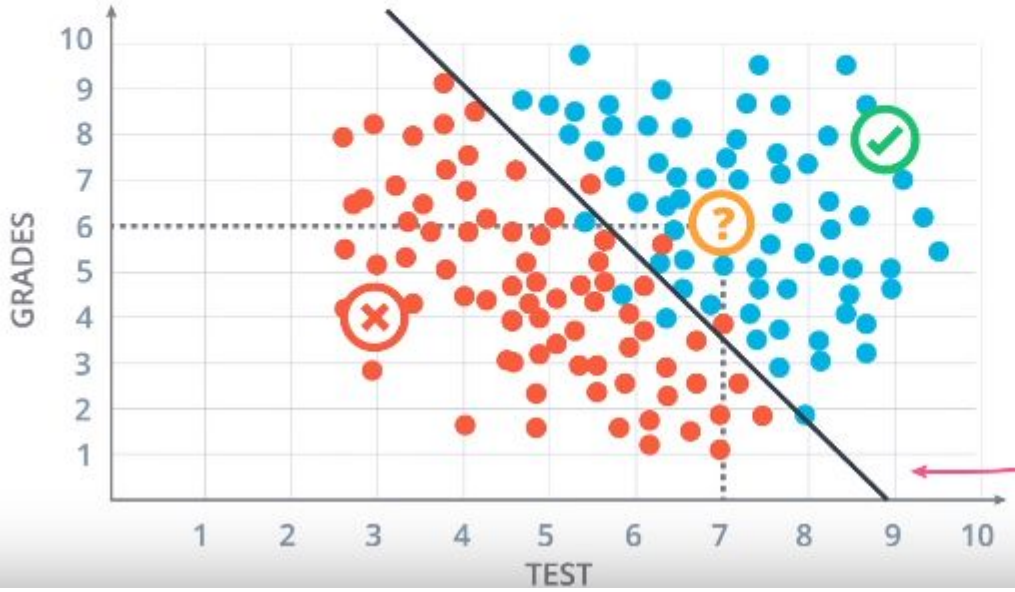




Basic Neural Net: Perceptron



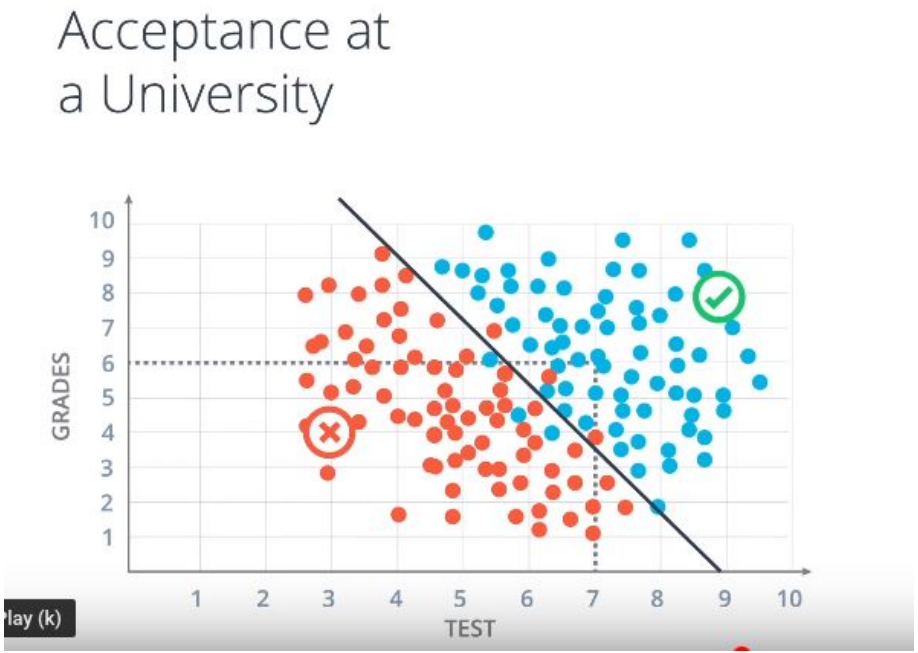
Question



QUESTION

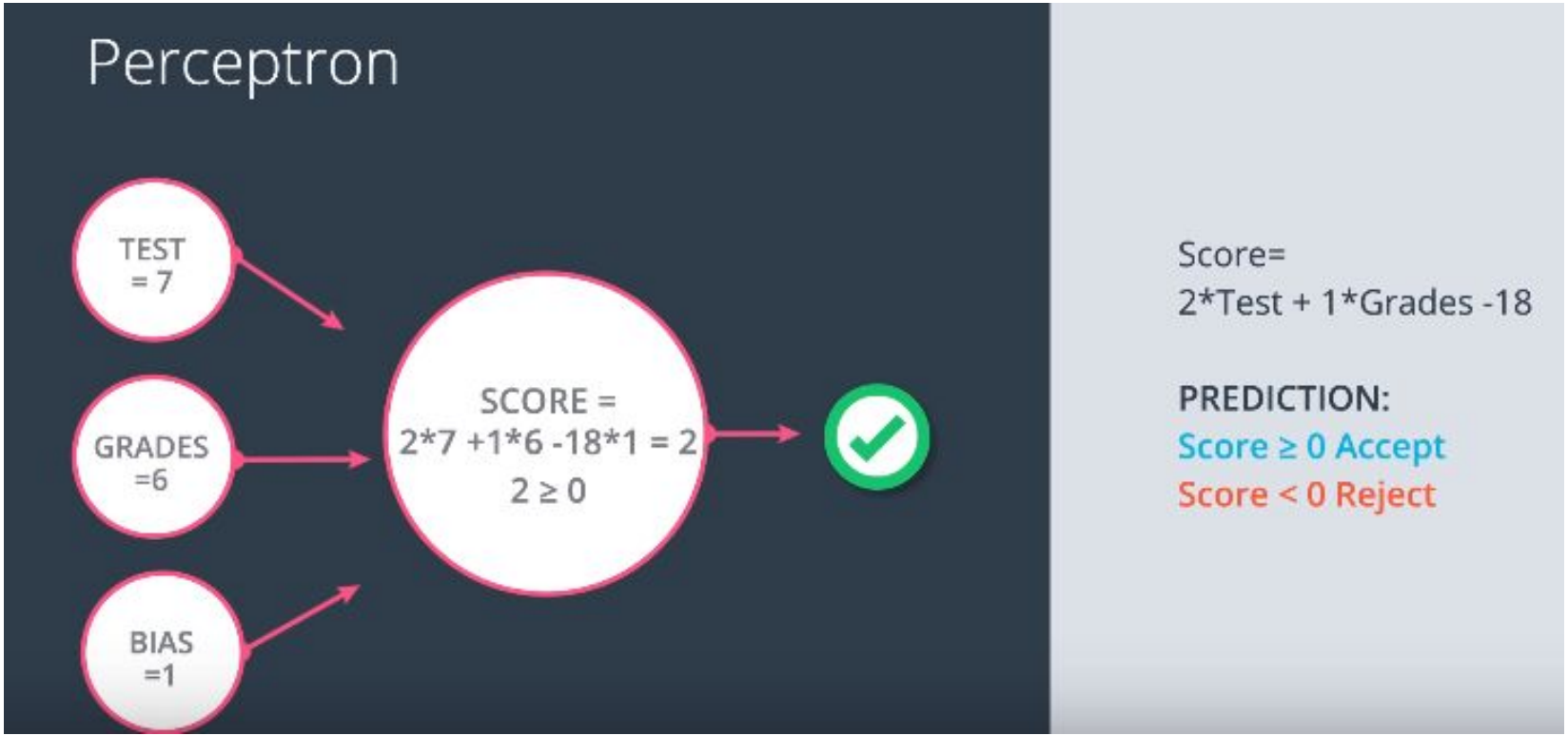
How do we find this line?

Example Data to Model



Perceptron

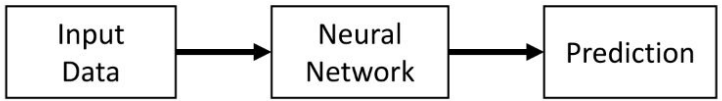

Perceptron
TEST
= 7
GRADES
=6
(7,6)
Score=
2*Test + 1*Grades -18
PREDICTION:
Score ≥ 0 Accept
Score < 0 Reject


Perceptron
TEST
= 7
GRADES
=6
BIAS
=1
SCORE =
2*7 +1*6 -18*1 = 2
2 ≥ 0
Score=
2*Test + 1*Grades -18
PREDICTION:
Score ≥ 0 Accept
Score < 0 Reject

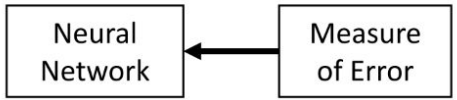
So how Networks Learn?

How Neural Networks Learn: Backpropagation

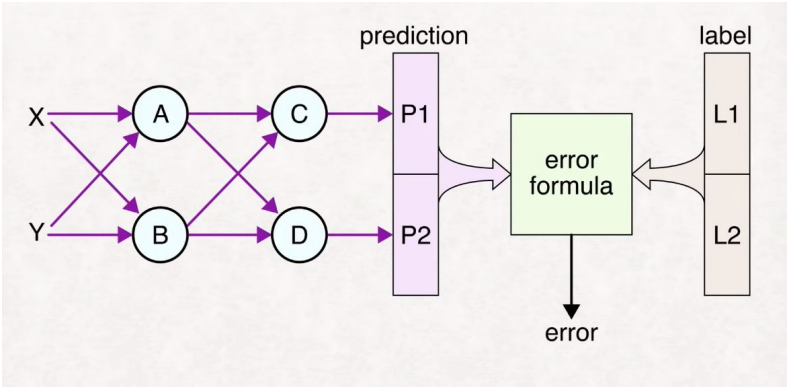
Forward Pass:



Backward Pass (aka Backpropagation):



Adjust to Reduce Error



Error function aka Loss function aka Objective function

Intuition:

- How far are we from the perfect solution
- We want our model (neural net) to bring us as close to the solution as possible

https://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications

Error function properties

- Should be differentiable
- Should be continuous

Intuition:
- Not all mistakes are the same. You should be penalized more given the magnitude of your actions(mistakes, errors).
If we predict with a high confidence class A even though it should be B we should be penalized more then if we predict A with a low confidence.

https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

Error functions

Most widely used losses:

- Mean Squared Error (regression)
- Mean Absolute Error (regression)
- Categorical Cross Entropy (usually classification)
- Binary Cross Entropy (usually classification)

But it's also very common to adapt the loss function for a specific task.
Focal Loss is a good example of that.

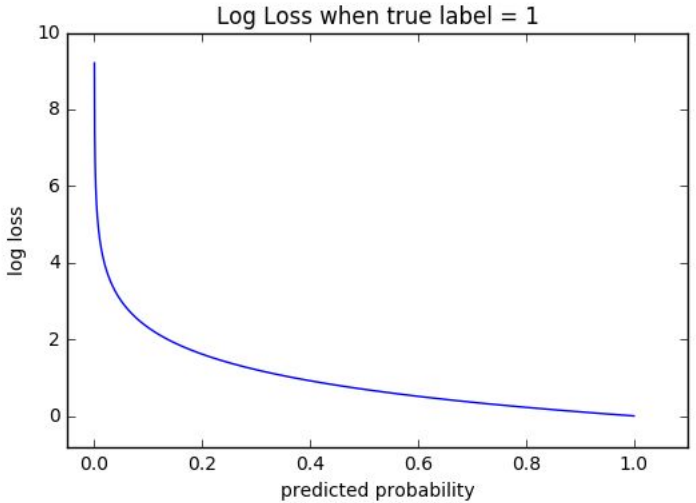
One-Hot Encoding

color	color_red	color_blue	color_green
red	1	0	0
green	0	0	1
blue	0	1	0
red	1	0	0

Cross Entropy

$$\hat{\mathbf{y}} \qquad\qquad \mathbf{y}$$

$$\begin{bmatrix} 0.1 \\ 0.5 \\ 0.4 \end{bmatrix} \qquad D(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_j y_j \ln \hat{y}_j \qquad \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$



Optimization: Gradient Descent (Updating the Weights)

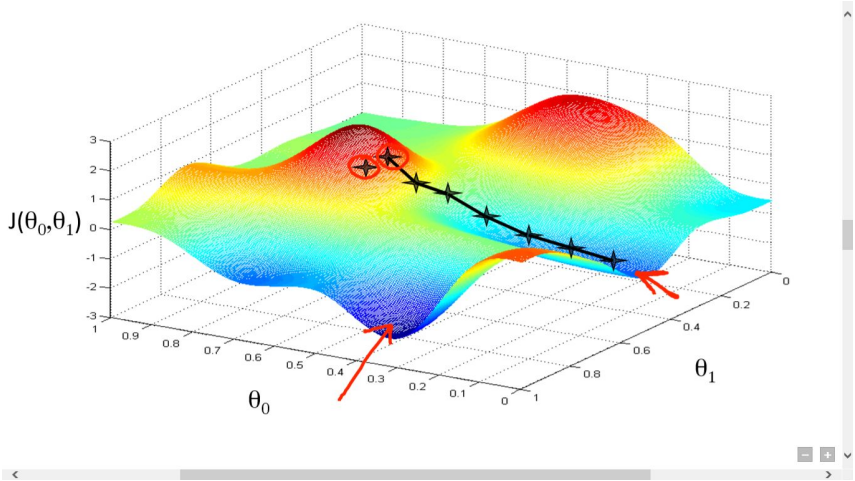
- Finding parameter values that achieve a functions minimum (usually loss)
- Iterative method (as opposed to trying to find a closed form solution)
- Gradient descent in itself has a lot of hyperparameters (how the descent is performed)
    - Learning rate
    - Parameter initialization
    - And others

https://www.deeplearning.ai/ai-notes/optimization/

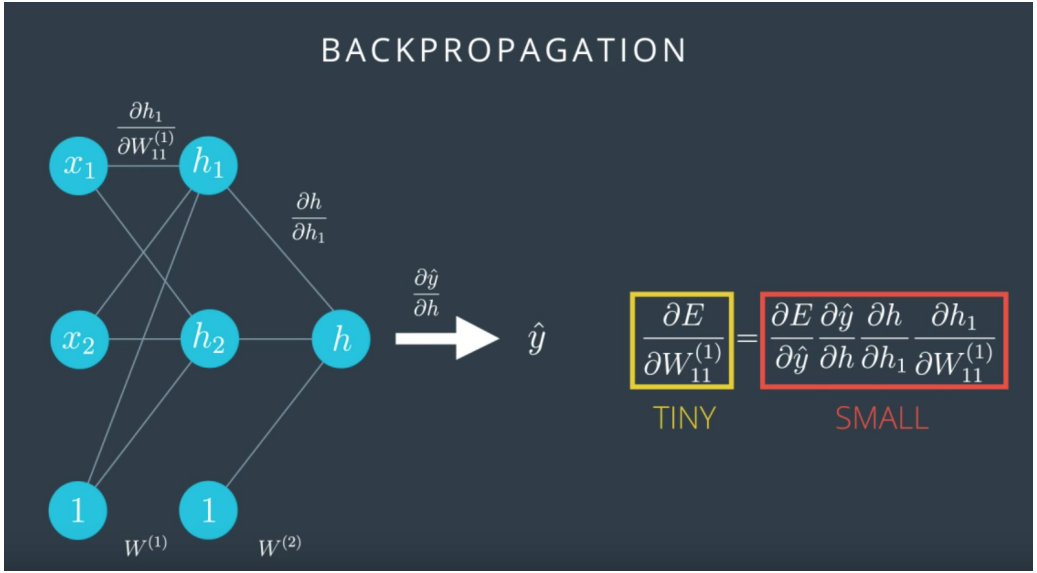
Gradient Descent (hackernoon)

Gradient Descent (Updating the Weights)

- The point of GD (gradient descent) is to minimize the cost/loss function this means finding such combination of weights and biases that will give us the lowest possible loss
- To find the lowest error(deepest valley) in the cost function(with respect to one weight), we need to tweak the parameters of the model. How much do we tweak them though? Enter Calculus. Using calculus, we know that the slope of a function is the derivative of the function with respect to a value. This slope always points to the nearest valley



(Vanishing) Gradients



Types of Gradient Descent

- Batch Gradient Descent (updates weights every epoch - all training samples)
- Stochastic Gradient Descent (updates weights every sample)
- Mini-Batch Gradient Descent (updates weights using minibacthes of N-samples usually in the range of (8-256)

In almost all of the cases when we talk about gradient descent we have mini batch gradient descent in mind.

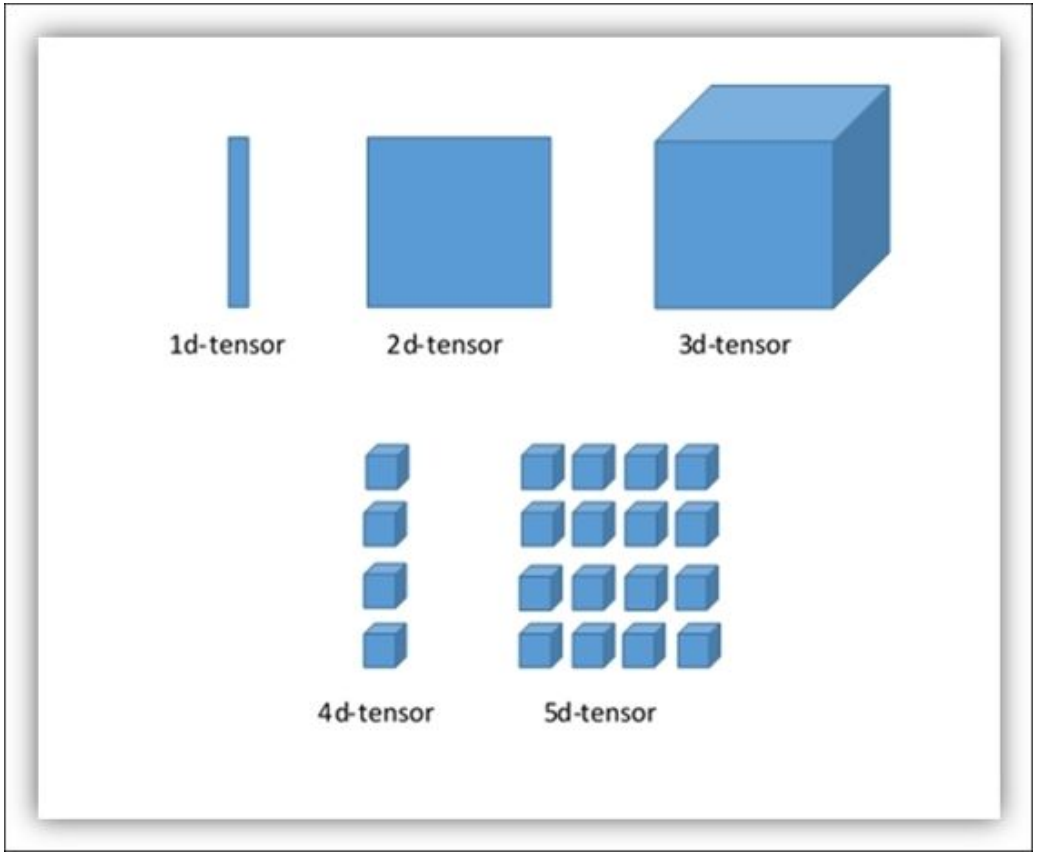
Mini-Batch Size

Mini-Batch size: Number of training instances the network evaluates per weight update step.

- Larger batch size = more computational speed
- Smaller batch size = (empirically) better generalization

"Training with large minibatches is bad for your health. More importantly, it's bad for your test error. Friends don't let friends use minibatches larger than 32."
- Yann LeCun
Revisiting Small Batch Training for Deep Neural Networks (2018)

What are (n-rank) Tensors?



Optimizers

Usually different variations of the Gradient Descent algorithm are in use. To combat common problems while optimizing neural networks (for instance powering over flat regions of the loss landscape)

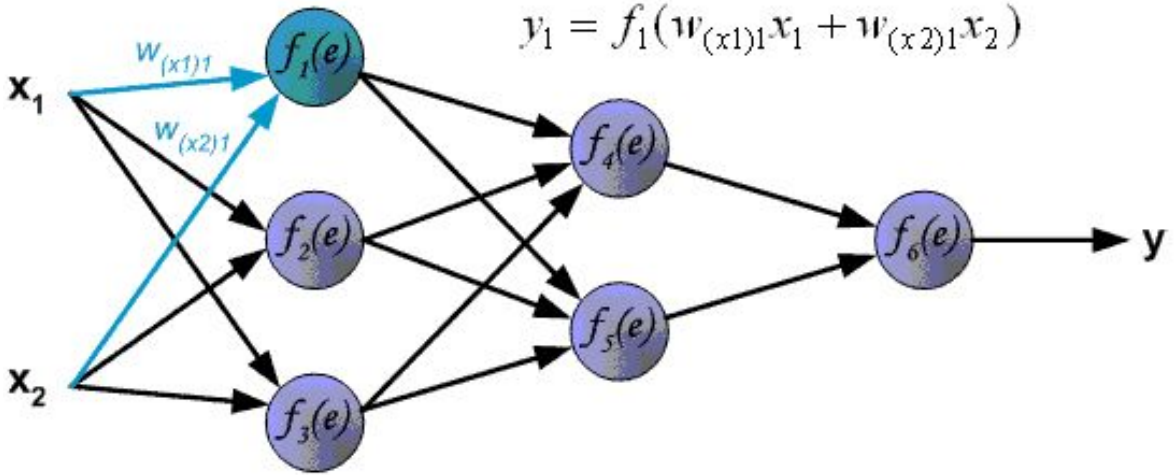
Momentum - Accumulate past gradient to continue in their direction

- RMSProp
- Adagrad
- AdaDelta
- Adam

Optimizer	Update rule	Attribute
(Stochastic) Gradient Descent	$W = W - \alpha dW$	• Gradient descent can use parallelization efficiently, but is very slow when the data set is larger the GPU's memory can handle. The parallelization wouldn't be optimal. • Stochastic gradient descent usually converges faster than gradient descent on large datasets, because updates are more frequent. Plus, the stochastic approximation of the gradient is usually precise without using the whole dataset because the data is often redundant. • Of the optimizers profiled here, stochastic gradient descent uses the least memory for a given batch size.
Momentum	$V_{dW} = \beta V_{dW} + (1 - \beta) dW$ $W = W - \alpha V_{dW}$	• Momentum usually speeds up the learning with a very minor implementation change. • Momentum uses more memory for a given batch size than stochastic gradient descent but less than RMSprop and Adam.
RMSprop	$S_{dW} = \beta S_{dW} + (1 - \beta) dW^2$ $W = W - \alpha \frac{dW}{\sqrt{S_{dW}} + \varepsilon}$	• RMSprop's adaptive learning rate usually prevents the learning rate decay from diminishing too slowly or too fast. • RMSprop maintains per-parameter learning rates. • RMSprop uses more memory for a given batch size than stochastic gradient descent and Momentum, but less than Adam.
Adam	$V_{dW} = \beta_1 V_{dW} + (1 - \beta_1) dW$ $S_{dW} = \beta_2 S_{dW} + (1 - \beta_2) dW$ $Vcorr_{dW} = \frac{V_{dW}}{(1 - \beta_1)^t}$ $Scorr_{dW} = \frac{S_{dW}}{(1 - \beta_2)^t}$ $W = W - \alpha \frac{dW}{\sqrt{S_{dW}} + \varepsilon}$	• The hyperparameters of Adam (learning rate, exponential decay rates for the moment estimates, etc.) are usually set to predefined values (given in the paper), and do not need to be tuned. • Adam performs a form of learning rate annealing with adaptive step-sizes. • Of the optimizers profiled here, Adam uses the most memory for a given batch size. • Adam is often the default optimizer in machine learning.

FP

g?



Evaluation metrics

Why are metrics important?

Training objective (cost function) is only a proxy for real world objective.

- Metrics help capture a business goal into a quantitative target (not all errors are equal).
- Helps organize ML team effort towards that target.

Example:

We build a spam detector our business metric could be accuracy or some version of F-score. Both of which are really not suitable as loss functions. We use cross-entropy as the proxy, to improve the business metric.

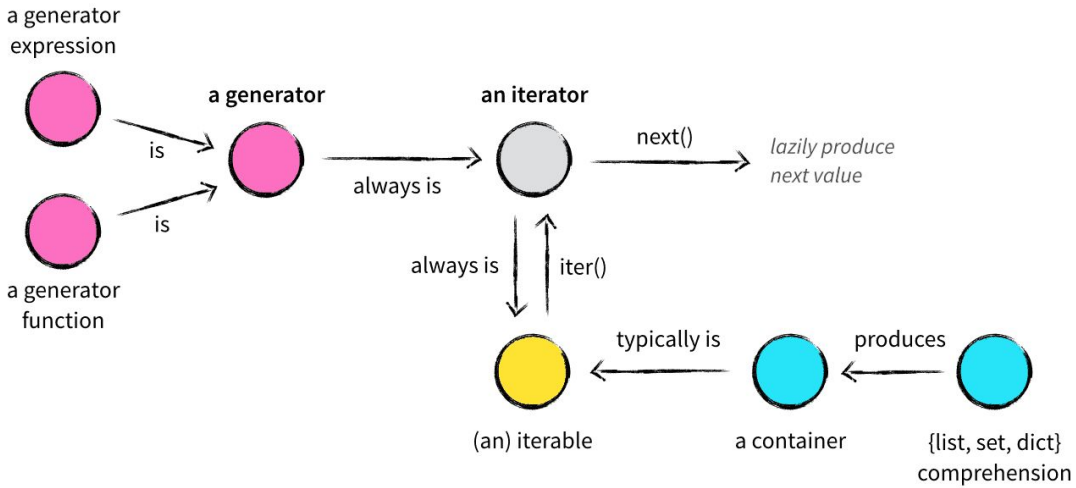
Python refresher - classes

```python
# Class / super

class Person:
    # initializing the variables
    name = ""
    age = 0
    def __init__(self, person_name, person_age):
        self.name = person_name
        self.age = person_age
    def show_name(self):
        print(self.name)
    def show_age(self):
        print(self.age)

class Student(Person):
    studentId = ""
    def __init__(self, student_name, student_age, student_id):
        Person.__init__(self, student_name, student_age)
        super().__init__(student_name, student_age)
        self.studentId = student_id
    def get_id(self):
        return self.studentId  # returns the value of student id

# Super allows us to initialize the super class without needing to provide the name of if explicitly
# super().__init__(student_name, student_age) == Person.__init__(self, student_name, student_age)

```

Python refresher - dunder methods

```python
# Some important Magic/Dunder methods

class A:
    def __init__(self, data):
        self.data = data

    def __len__(self):
        return len(self.data)

    def __getitem__(self, i):
        return self.data[i]

    def __str__(self):
        return "This is alternative representation off data {}".format(self.data)

    def __repr__(self):
        return self.__str__()

    def __call__(self):
        print("I can be called like a function without specifying a method")

data = A([1,2,3,4,5])
len(data) # prints 5
data[0] # prints 1
data() # prints  "I can be called like a function without specifying a method"
# etc
```

Python refresher - iterators and generators



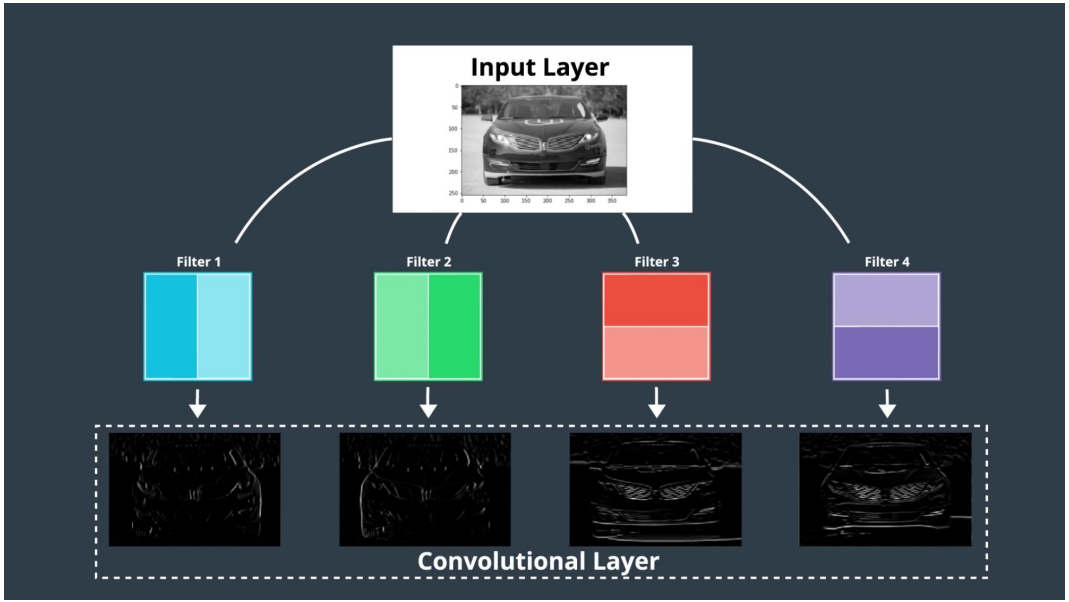
Pytorch

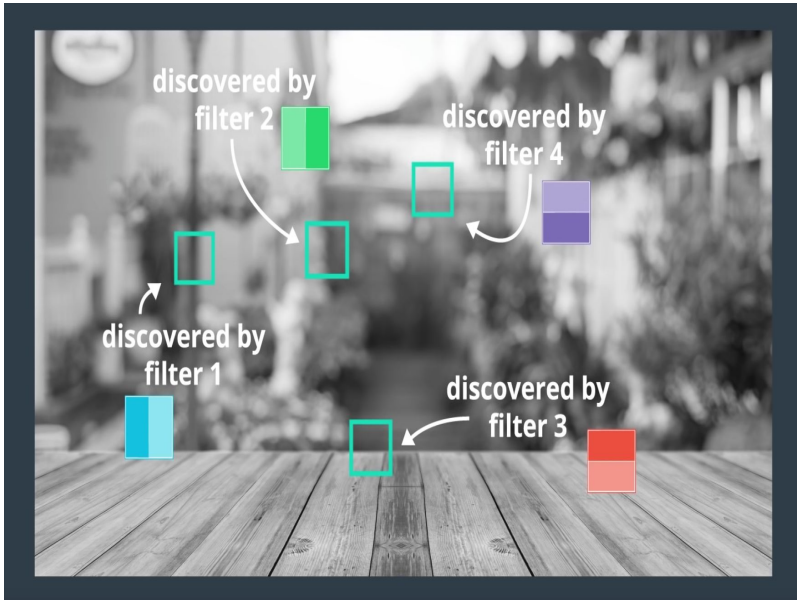
Package	Description
torch	Numpy like library with GPU support
torch.autograd	Give differentiation support for all torch ops
torch.nn	Neural network library integrated with autograd
torch.optim	Optimization for torch.nn (ADAM, SGD, RMSPROP, etc...)
torch.multiprocessing	Memory sharing between tensors
torch.utils	DataLoader, Training and other utility functions
torch.legacy	Old code ported from Torch

Types of Layers

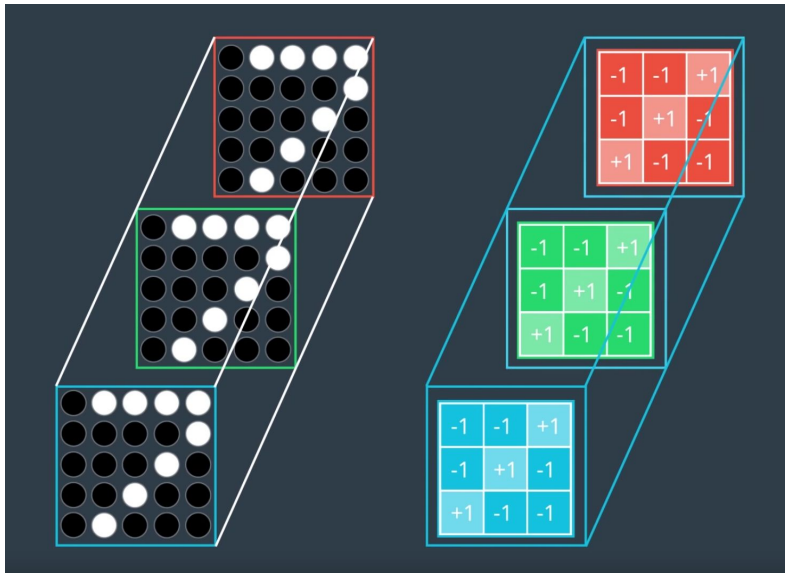
- (adaptive) Pooling (max/avg) Layer
- BatchNorm
- Dropout
- GRU/GRUCell/LSTM
- UnPool
- Activation
- Up-sampling
- Linear
- Convolutional

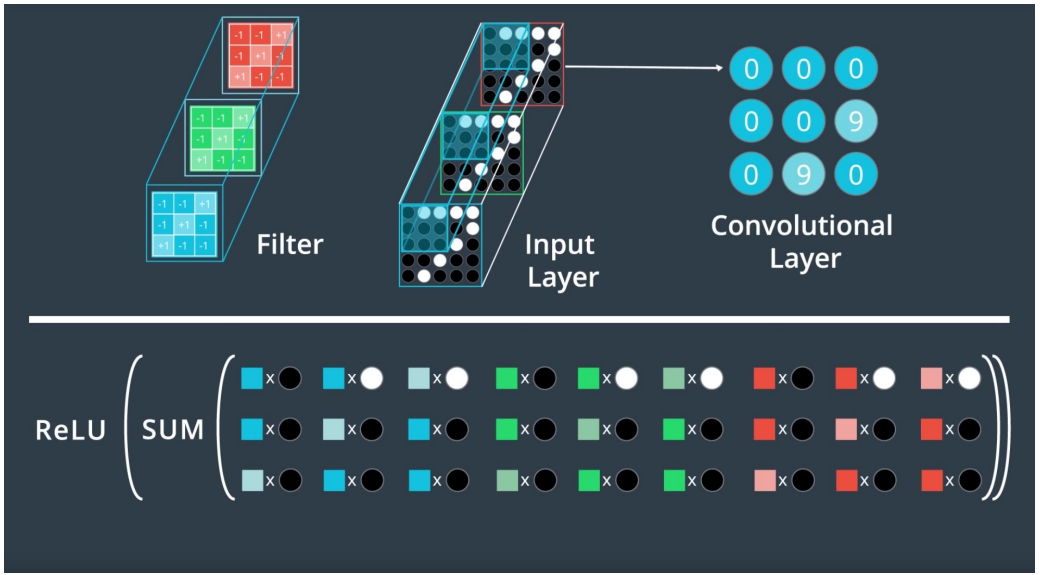
Convolutions on grayscale images



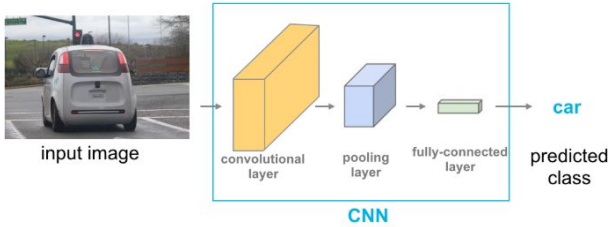


Convolutions on RGB(multi channel images)

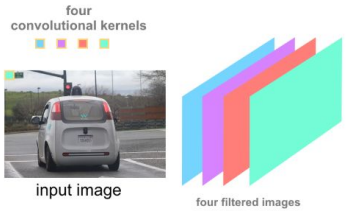




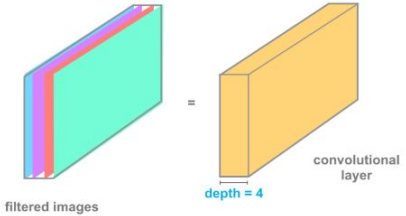
Convolutions



Layers in a CNN.



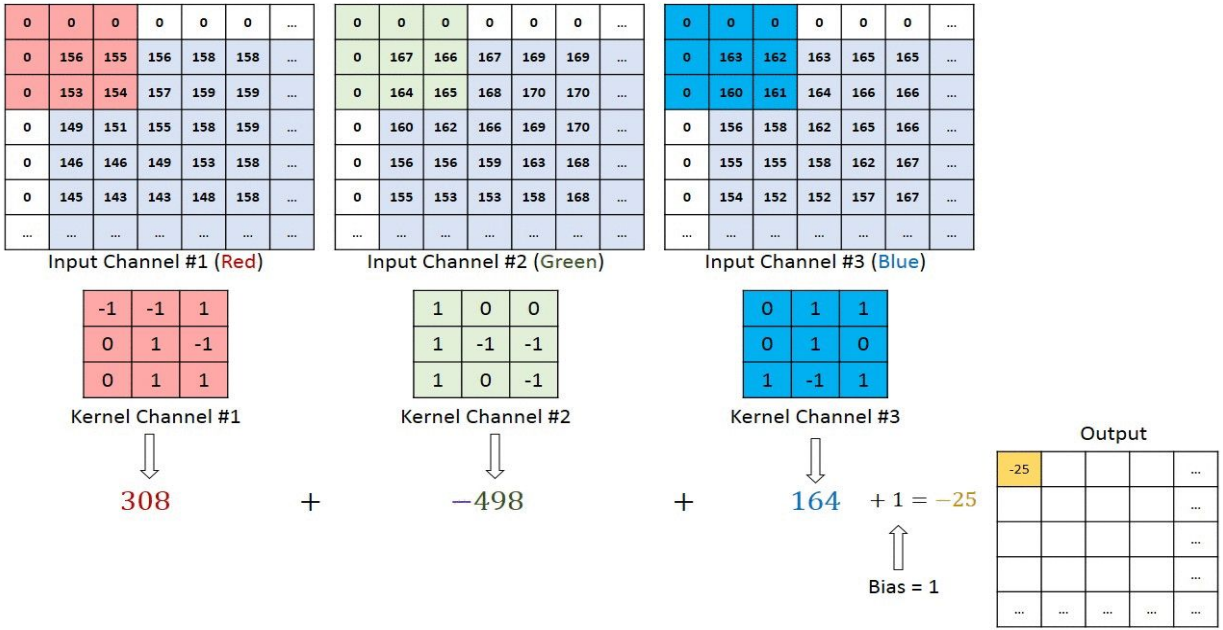
4 kernels = 4 filtered images.



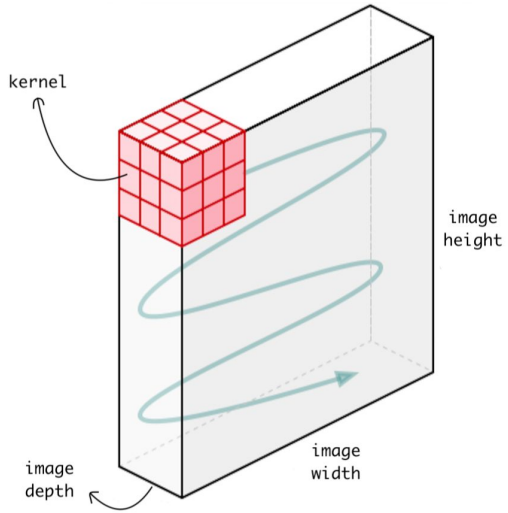
A convolutional layer.

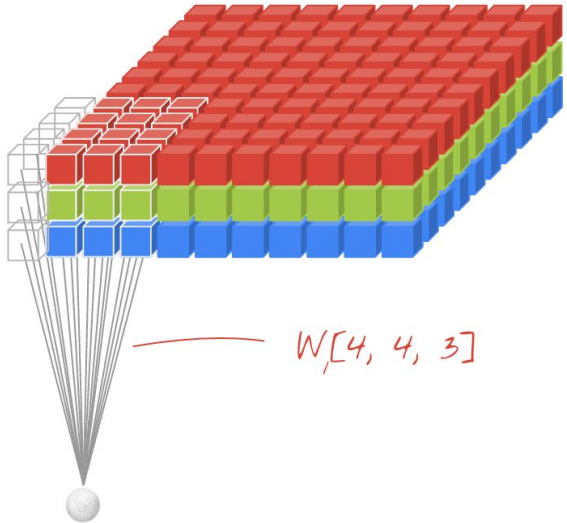
https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html

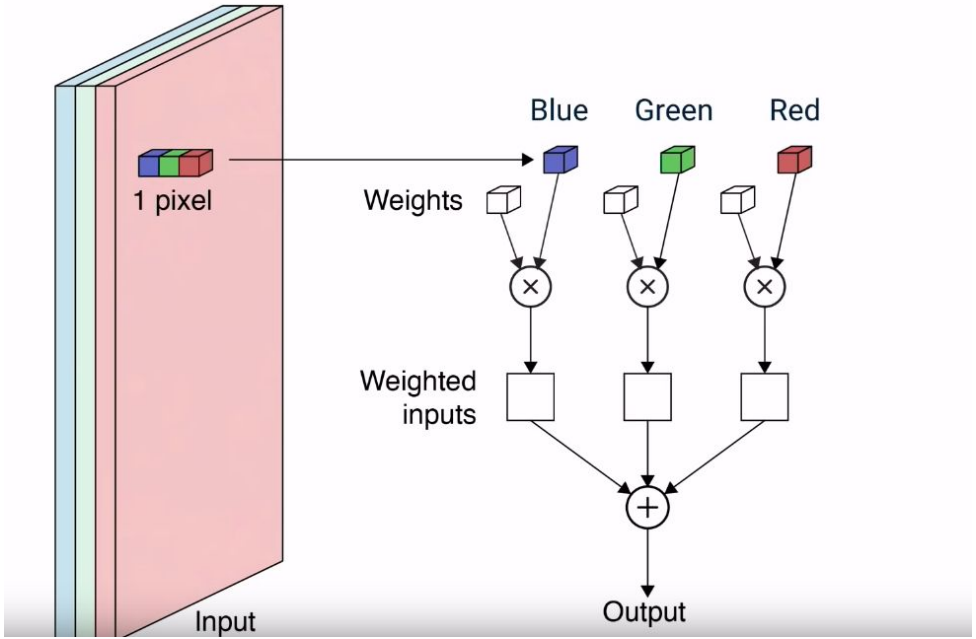
Convolutions

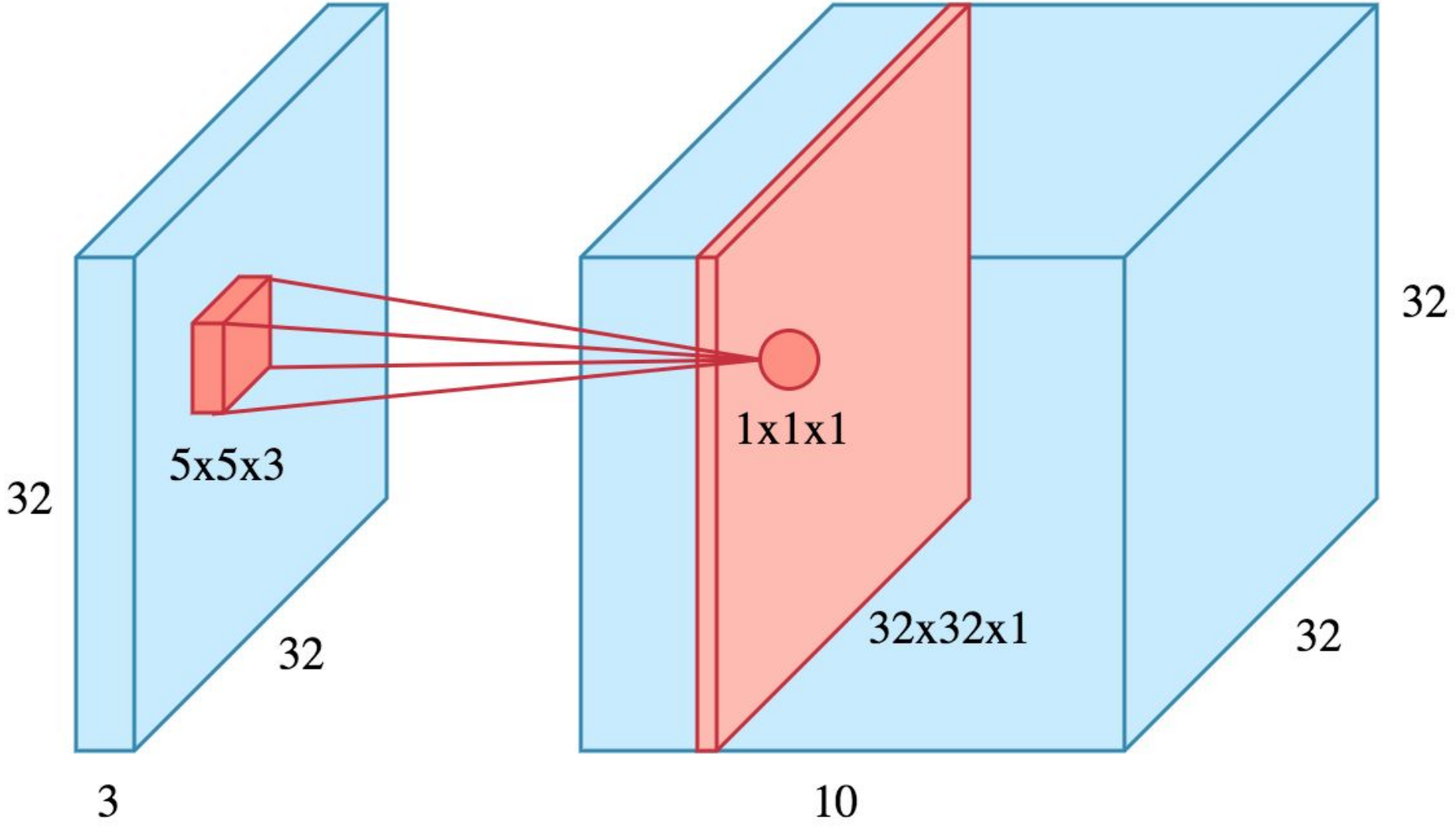


Convolutions Convolutions ...

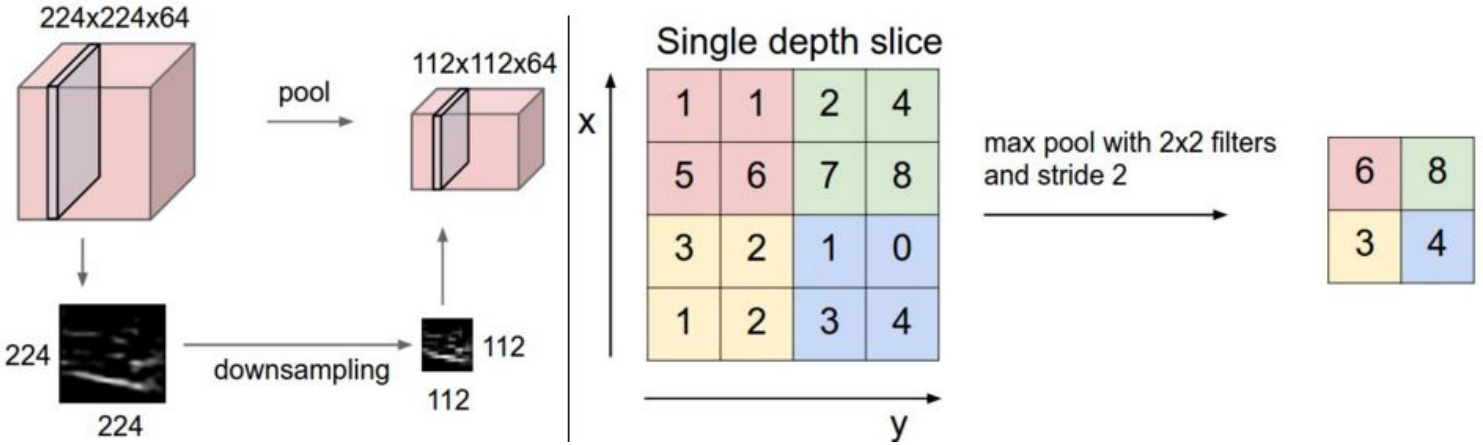





Blue
Green
Red
1 pixel
Weights
Weighted
inputs
Input
Output

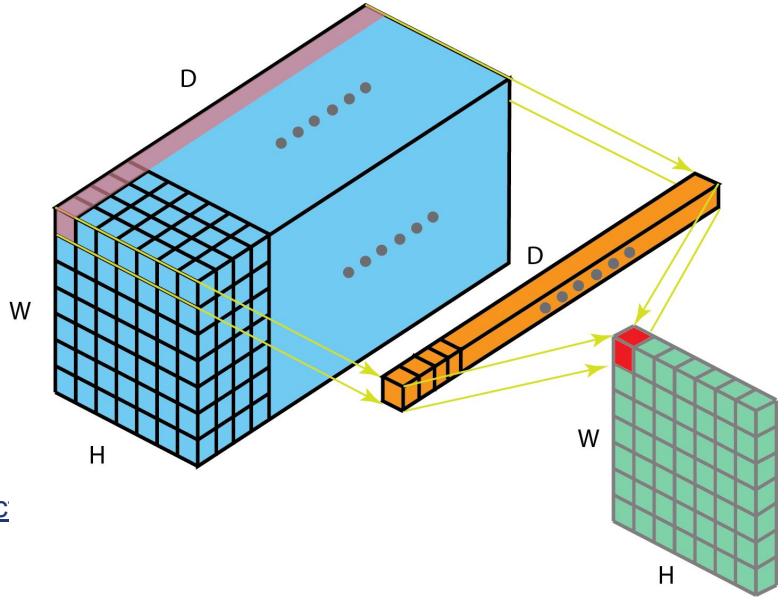

32
5x5x3
32
3
1x1x1
32x32x1
32
32
10

Pooling


224x224x64
pool
112x112x64
224
224
downsampling
112
112
Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

1x1 Convolutions

- Dimensionality reduction for efficient computations
- Efficient low dimensional embedding, or feature pooling
- Applying nonlinearity again after convolution

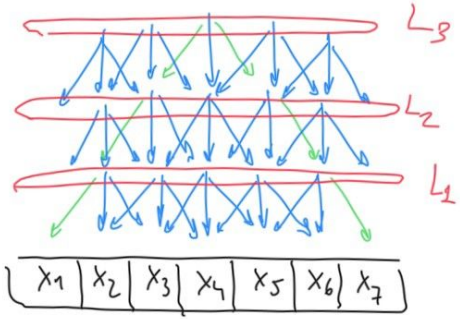


https://towardsdatascience.com/a-comprehensive-introduc
669281e58215

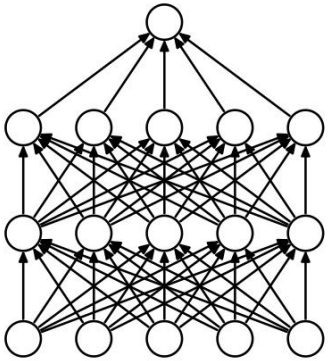
Receptive field

https://medium.com/@santi.pdp/receptive-fields-in-convolutional-neural-networks-6368a699d838

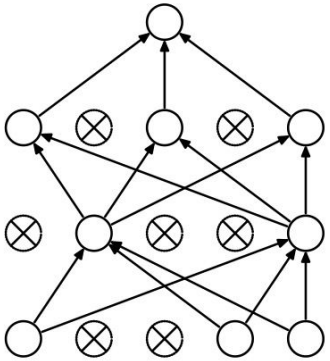
The amount of context that a neuron sees in the input to predict its out



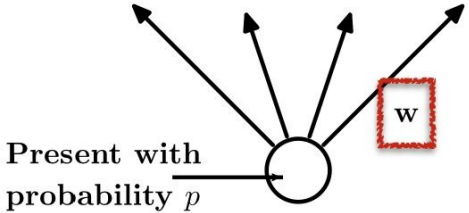
Dropout



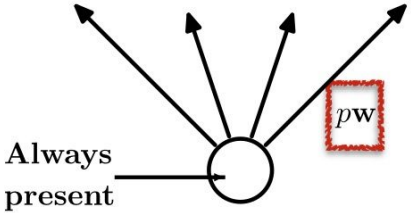
(a) Standard Neural Net



(b) After applying dropout.

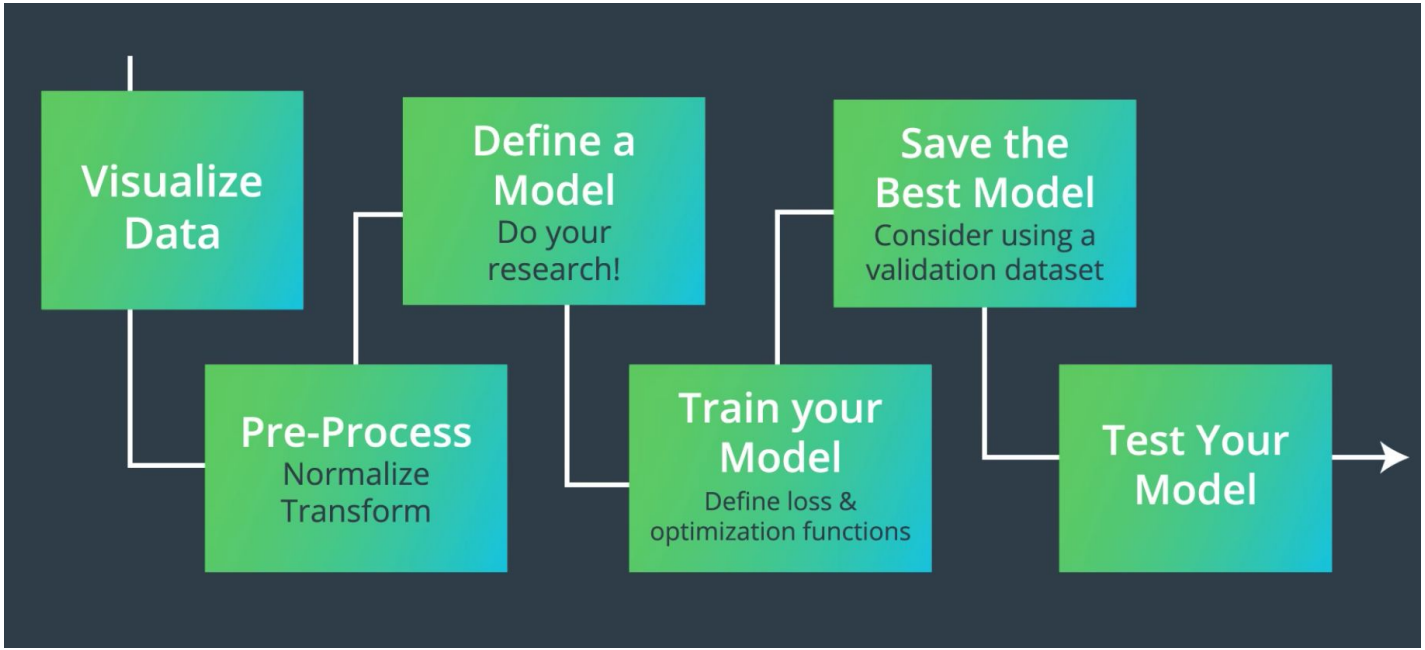


(c) At training time



(d) At test time

Image classification steps



Define a pytorch model

```
class Perceptron(nn.Module):
    def __init__(self, n_in, n_out):

        super(Perceptron, self).__init__()
        self.fc = nn.Linear(n_in, n_out, bias=True)

    def forward(self, x):
        return self.fc(x)
```

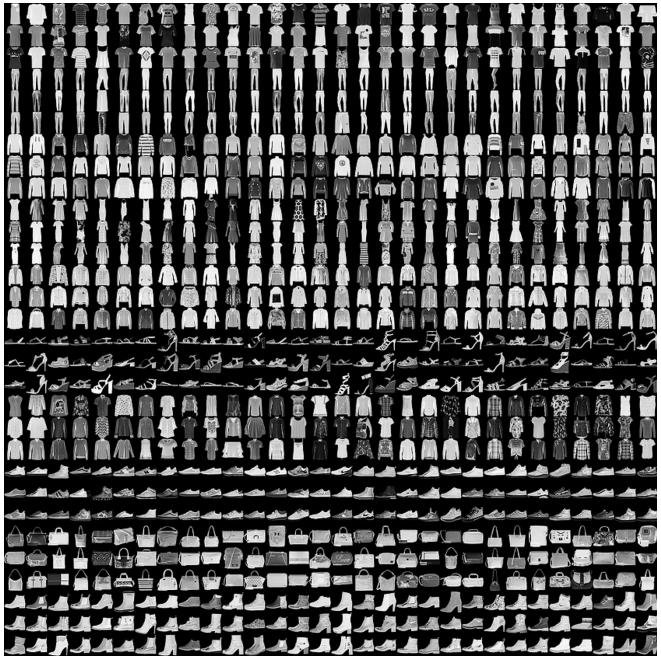
Loading Data with Pytorch

Loading data with pytorch is very well designed and comprises of 3 main modules, that nicely play together.

- torch.utils.data.dataloader (responsible for efficiently loading, and sampling data)
- torch.utils.data.dataset (a base for our dataset we have to override __len__ and __getitem__ methods)
- torch.utils.data.sampler (provides indices to be sampled (sampling strategy))
    - Sequential Sampler
    - Random Sampler
    - Subset Sampler

Fashion MNIST

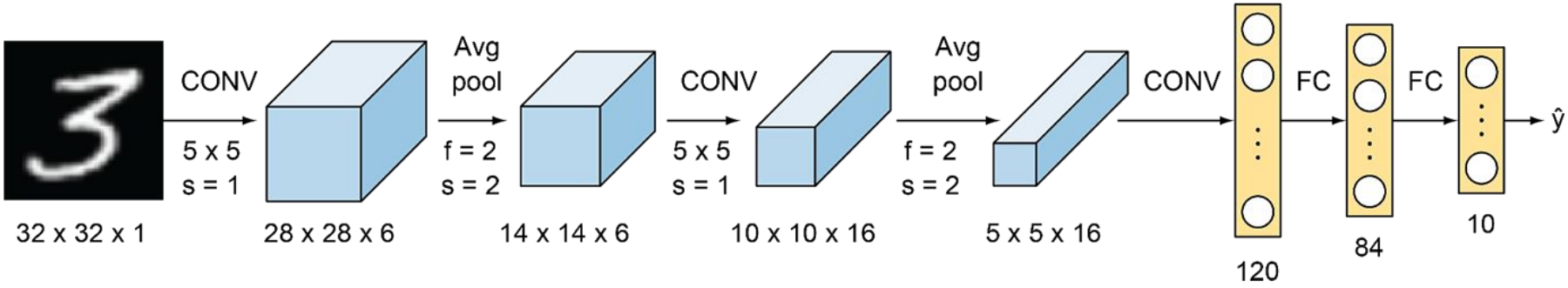
- 28x28x1 images
- 10 classes
- 10,000 samples



```
pd.read_csv('fashion-mnist_test.csv')
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781
0	0	0	0	0	0	0	0	0	9	8	...	103	87	56	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	...	34	0	0	0	0	0	0
2	2	0	0	0	0	0	0	14	53	99	...	0	0	0	0	63	53	31
3	2	0	0	0	0	0	0	0	0	0	...	137	126	140	0	133	224	222
4	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
5	2	0	0	0	0	0	44	105	44	10	...	105	64	30	0	0	0	0
6	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
7	6	0	0	0	0	0	0	0	1	0	...	174	136	155	31	0	1	0
8	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	...	57	70	28	0	2	0	0
10	3	0	0	0	0	0	0	0	1	0	...	12	0	0	0	0	0	0
11	4	0	0	0	0	0	0	1	1	0	...	2	0	21	138	151	71	0
12	4	0	0	0	0	0	0	0	0	0	...	118	73	0	164	225	123	0
13	6	0	0	0	0	0	0	1	1	0	...	0	0	0	0	17	0	0
14	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
15	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
16	6	0	0	0	0	0	0	1	5	0	...	90	60	0	129	146	78	0
17	3	0	0	0	0	0	0	0	0	15	...	3	0	3	0	0	0	0
18	6	0	0	0	0	0	0	0	2	2	...	0	0	0	5	3	1	0
19	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	50	107
20	4	0	0	0	0	0	0	0	0	0	...	187	210	52	0	2	0	0
21	4	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	0
22	2	0	0	0	0	0	0	1	0	0	...	0	0	0	118	158	92	0
23	1	0	0	0	0	0	0	0	0	0	...	227	96	0	0	0	0	0
24	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
25	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
26	8	0	0	0	0	0	0	0	0	0	...	39	31	19	19	0	0	0
27	4	0	0	0	0	0	0	0	0	0	...	50	17	0	0	0	1	4
28	4	0	0	0	0	0	0	0	0	0	...	0	0	215	219	172	0	0
29	1	0	0	0	0	0	0	0	0	0	...	197	177	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

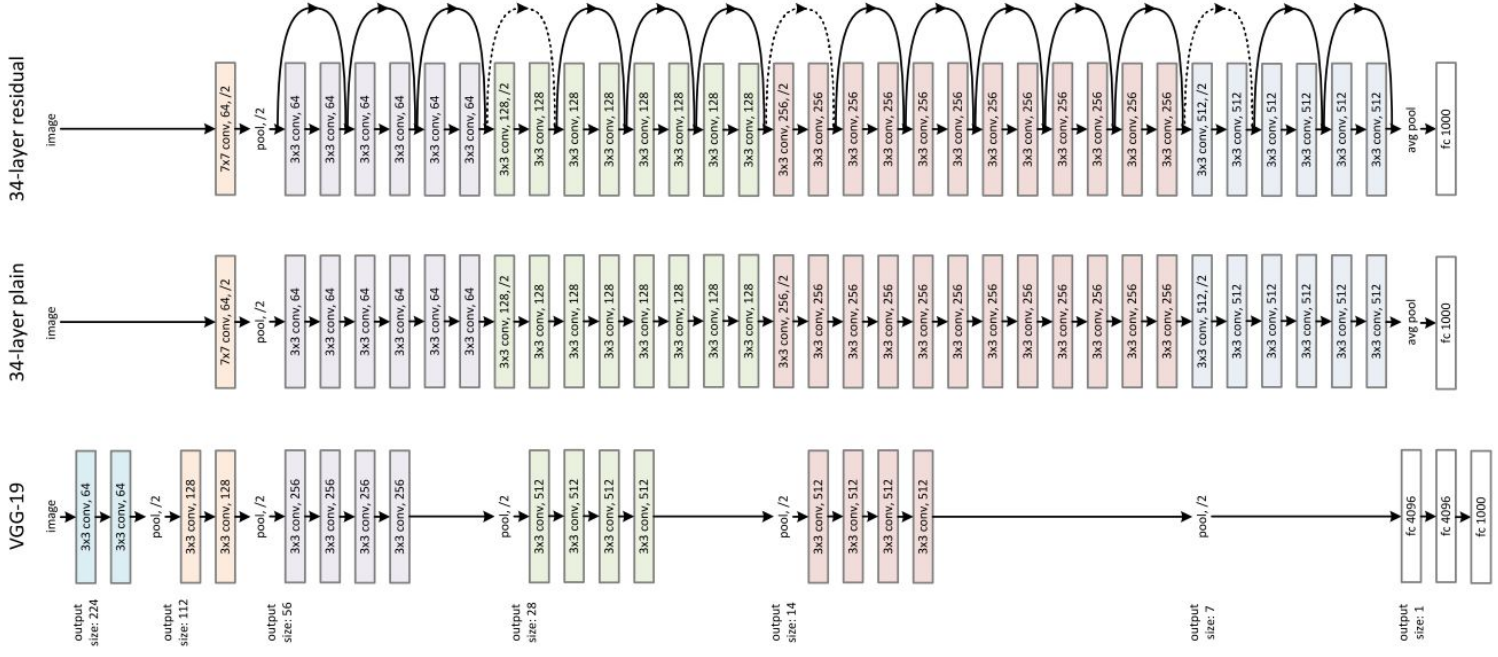
Lenet 5


CONV
5 x 5
s = 1
Avg
pool
f = 2
s = 2
CONV
5 x 5
s = 1
Avg
pool
f = 2
s = 2
CONV
FC
FC
ŷ
32 x 32 x 1
28 x 28 x 6
14 x 14 x 6
10 x 10 x 16
5 x 5 x 16
120
84
10

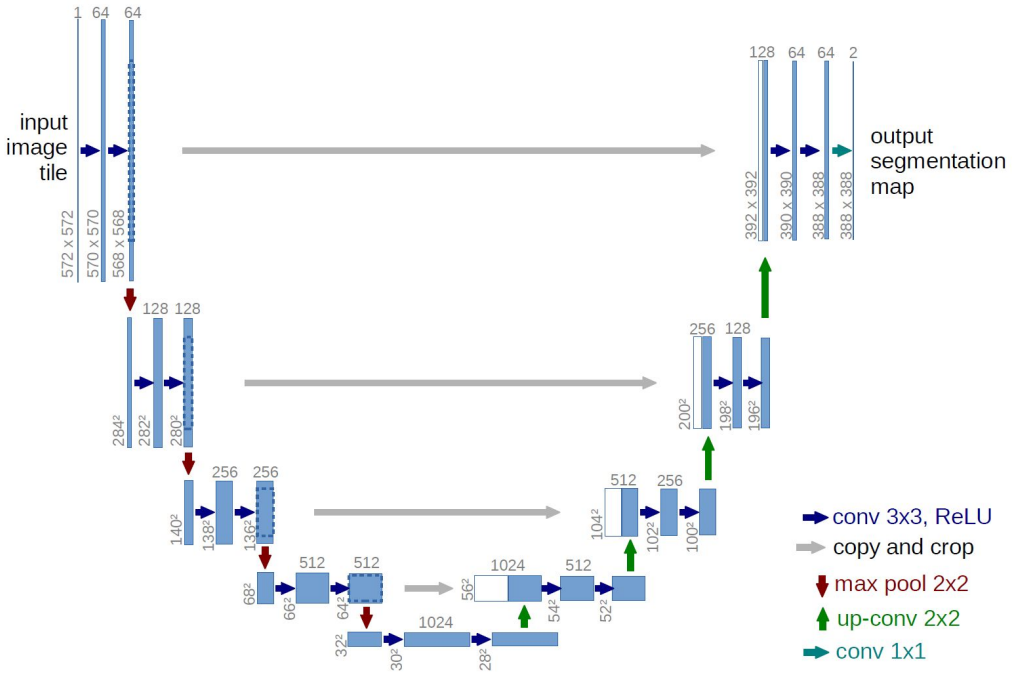
Loading the Fashion-Mnist dataset

```
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
import torchvision.transforms as transforms

class FashionMnist(Dataset):

    def __len__(self):
        return len(self.metadata_df)

    def __init__(self, metadata_df,
                 transform=None,
                 augmentator=None):

        self.metadata_df = metadata_df.copy()
        self.augmentator = augmentator
        self.transform = transform

    def load_image_and_target(self,index):
        oneimage = self.metadata_df.iloc[index]
        image, y = PIL.Image.fromarray(
            np.array(oneimage[1:]).reshape(28, 28).astype('uint8'), 'L'), oneimage[0]
        return image, y

    def __getitem__(self, index):
        X, y = self.load_image_and_target(index)
        X = self.transform(X)

        return X, y

    def collate_func(self, batch):
        pass

transform = transforms.Compose([transforms.ToTensor()])
fmnist = FashionMnist(df, transform=transform)

image, label = fmnist[0]

fmnist_dl = DataLoader(fmnist, batch_size=32)
X, y = next(iter(fmnist_dl))
```

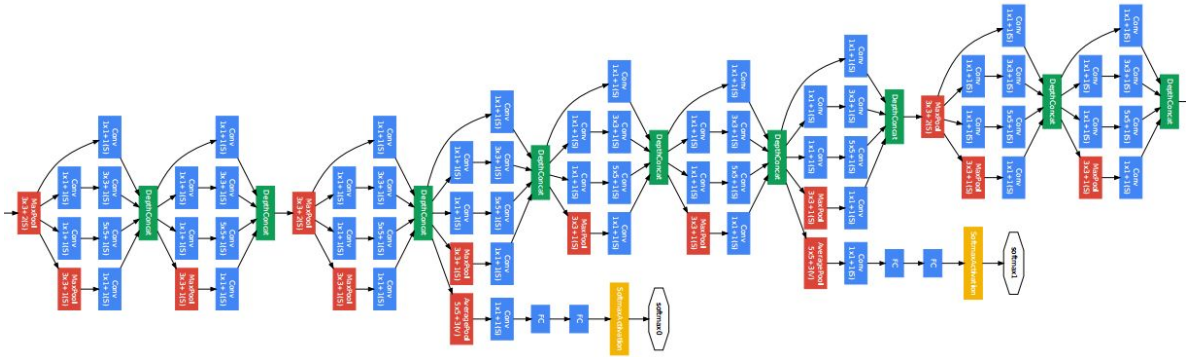
Famous Architectures - Resnet

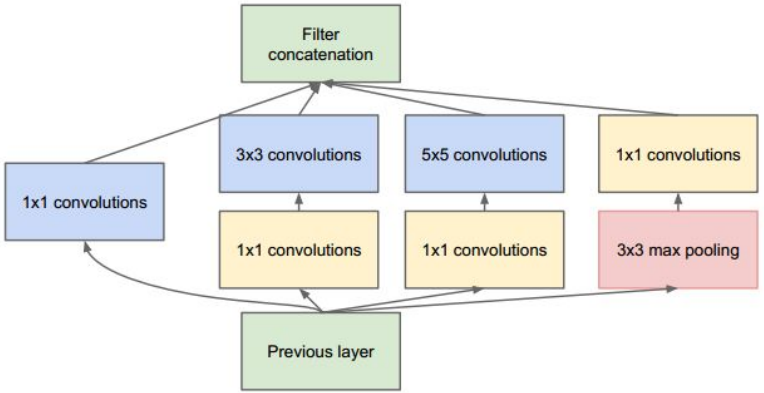


Famous Architectures - UNet



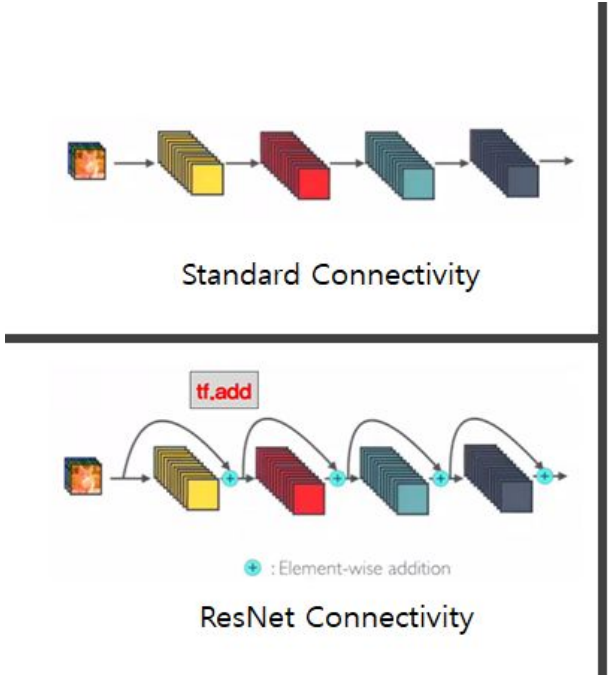
Inception

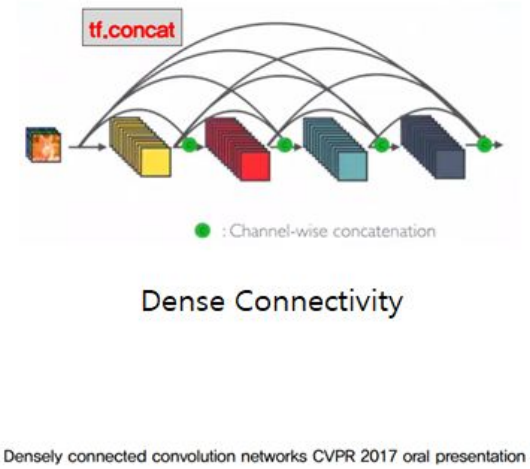



Filter concatenation
1x1 convolutions
3x3 convolutions
5x5 convolutions
1x1 convolutions
1x1 convolutions
1x1 convolutions
1x1 convolutions
3x3 max pooling
Previous layer

DenseNet

- Strong gradient flow
- Computational efficient
- More diversified features
- Maintains low complexity features
- Creates uncorrelated features





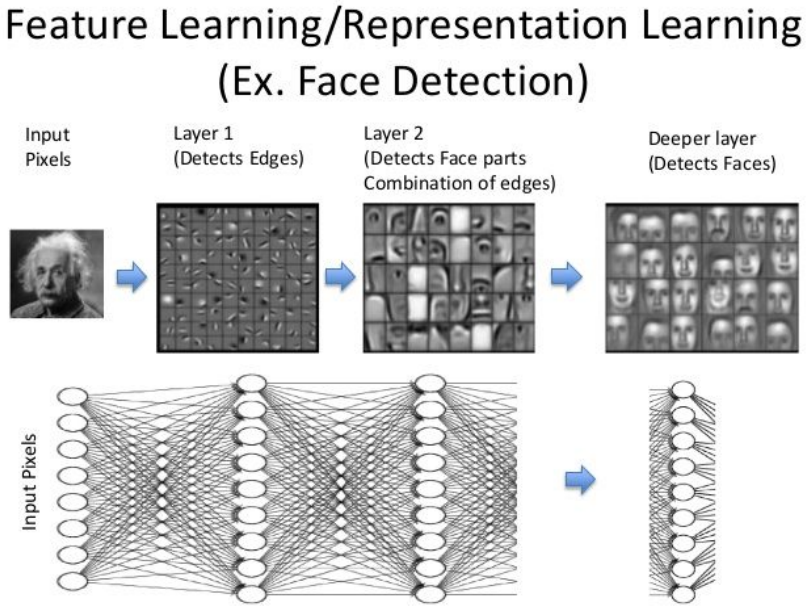
Densely connected convolution networks CVPR 2017 oral presentation slide

Training a classifier on Fashion Mnist

Check notebok pytorch-fmnist.ipynb

Feature Representation

- Deep Networks Learn (re)usable features
- Usually deeper layers produce more dataset/task specific features



Model Interpretability - GradCAM

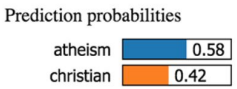
Gradient-weighted Class Activation Mapping

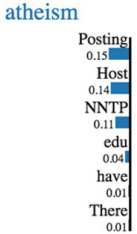
Grad-CAM for "Cat"

Grad-CAM for "Dog"

Model Interpretability - Lime

https://github.com/marcotcr/lime

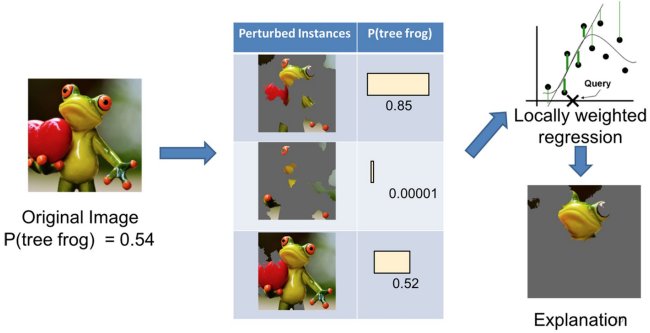




Text with highlighted words
From: johnchad@triton.unm.edu (jchadwic)
Subject: Another request for Darwin Fish
Organization: University of New Mexico, Albuquerque
Lines: 11
NNTP-Posting-Host: triton.unm.edu

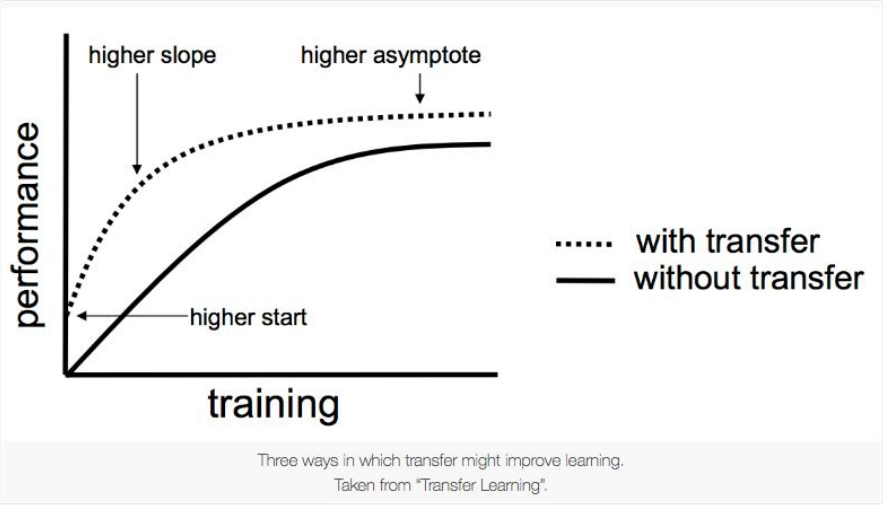
Hello Gang,

There have been some notes recently asking where to obtain the DARWIN fish.
This is the same question I have and I have not seen an answer on the
net. If anyone has a contact please post on the net or email me.



Transfer Learning

Transfer learning and domain adaptation refer to the situation where what has been learned in one setting … is exploited to improve generalization in another setting



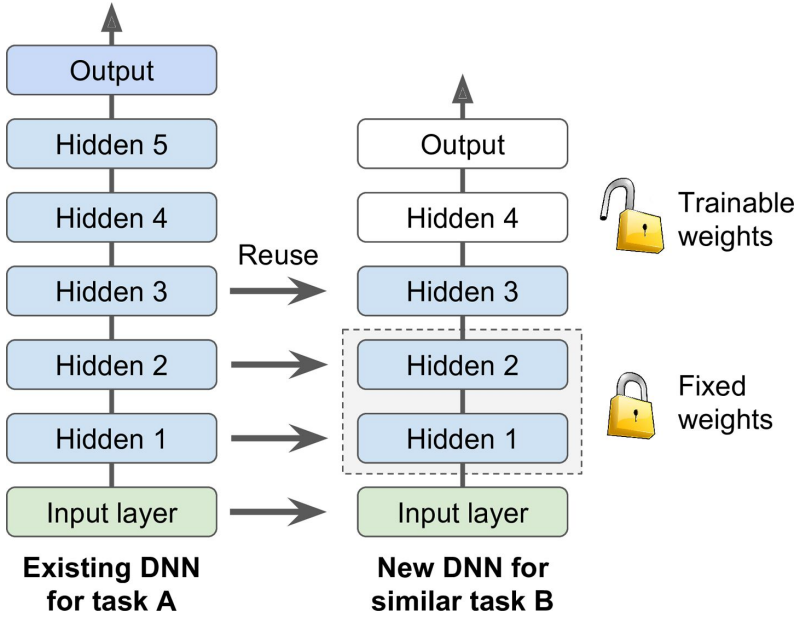
Three ways in which transfer might improve learning.
Taken from "Transfer Learning".

Using pre-trained models (imagenet)

Before making predictions we need to make sure that we preprocess the input data properly. Few things that we need know about the model-training process:

- Input normalization (mean, std)
- Input-size
- Input range (0-1, 0-255)
- Input space (BGR, RGB, other)

Transfer Learning




```
cnn = resnet50(pretrained=True)
freeze_model(cnn)
cnn.avgpool = nn.AdaptiveAvgPool2d(1) # This will allow to use different input sizes
cnn.last_linear = nn.Sequential(nn.Linear(cnn.last_linear.in_features, 1024),
                                nn.Linear(1024, 10))
```

Types of Datasets

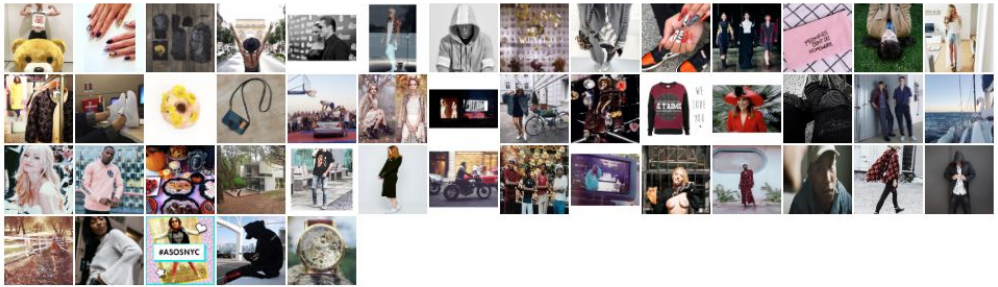
- **Training** set - we train the system on this data
- **Development set** - Used to tune parameters, select features etc
- **Test** set - used to evaluate the performance of the algorithm but not to make decisions about parameters

Choosing Dev and Test data

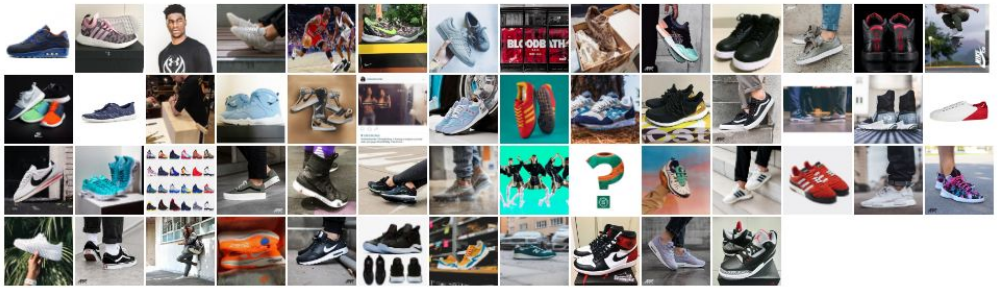
- Both dev and test data should reflect data you expect to get in the future
- Both should generally be from the same distribution

Sneakers not Sneakers Dataset

Drawing negatives



Drawing sneakers



Simple Transfer Learning Project

Creating a sneakers/shoe detector based on instagram and google search.
We collect a dataset of sneakers and not sneakers. Using a pretrained network and transfer learning we are able to a

chieve a good classifier with very little data and time spend

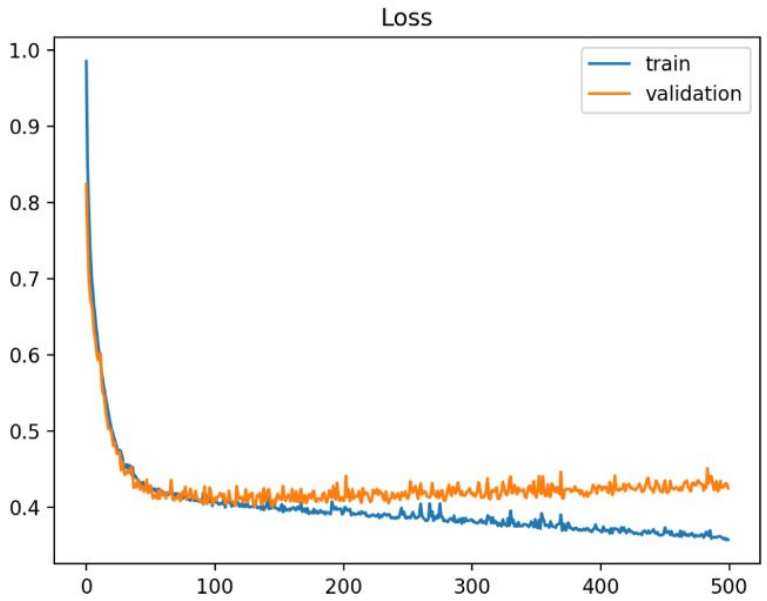
Useful tool to develop other projects (it allows easy image download from google-images)

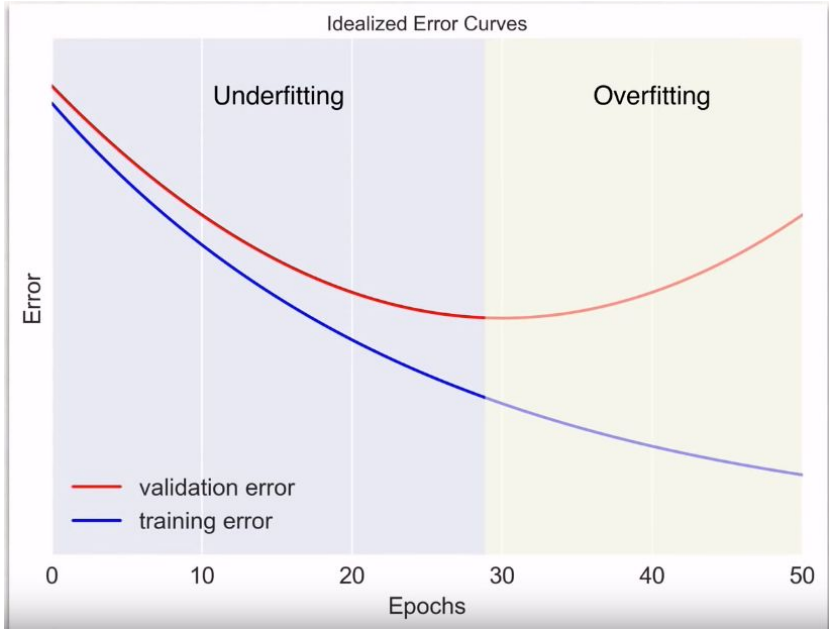
https://github.com/hardikvasa/google-images-download

Making Deep Learning Better

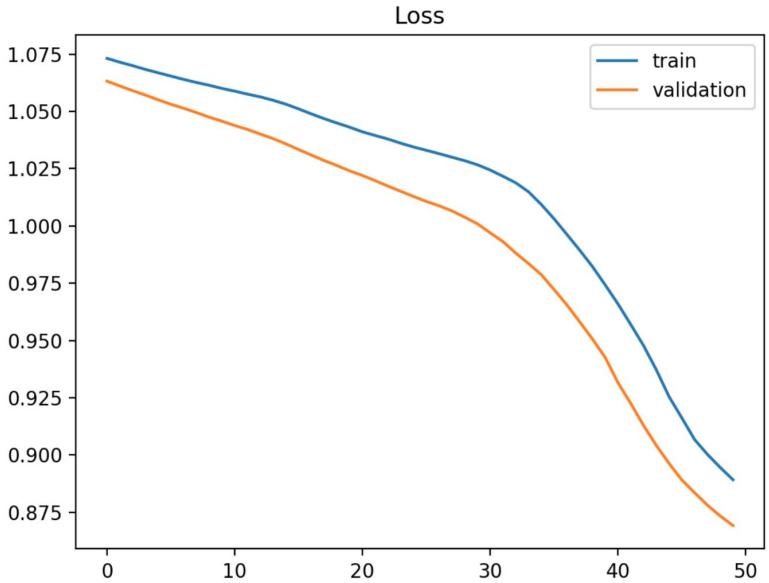
- Better Learning
  - Depth of the model
  - Batch size
  - Optimizer
  - LR
- Better Generalization
  - Regularization (Dropout, Augmentation)
  - Activity regularization, Weight regularization
  - Noise regularization
  - Early Stopping
- Better Predictions
  - Model Averaging Ensambles
  - Weight Averaging Ensambles
  - Snapshot Ensambles

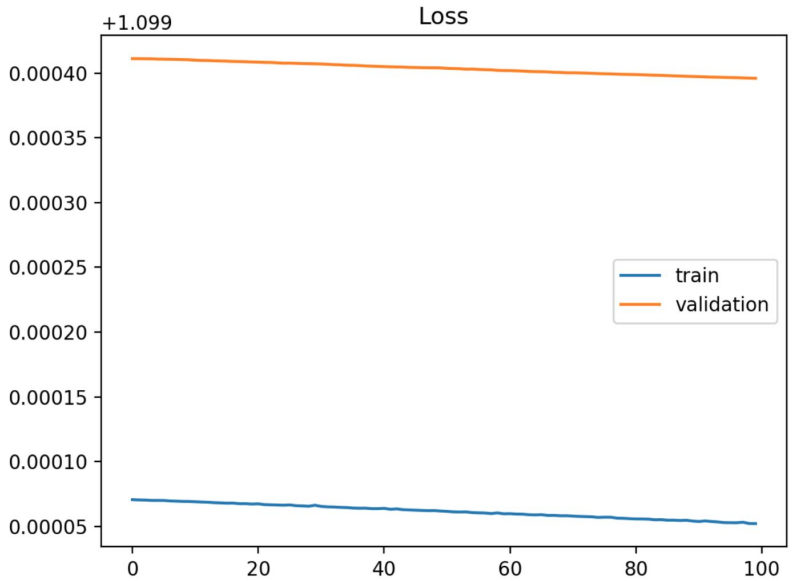
Understanding Learning Curves - Overfitting





Learning Curves - Underfitting





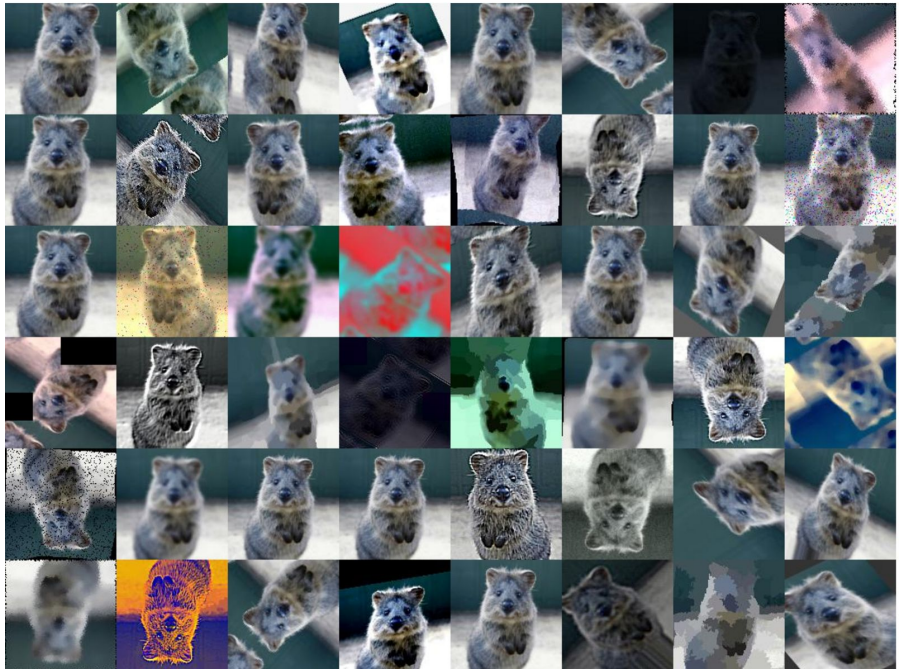
Overfitting - Augmentation

TIPS:

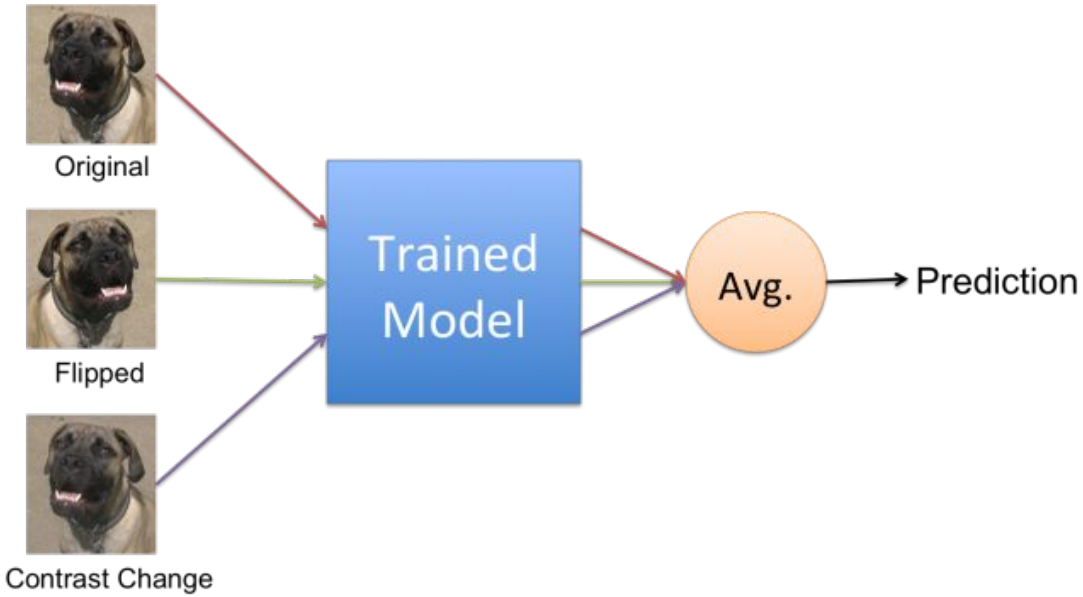
- Augmenting images can be a great regularizer that is pretty flexible and easy to control
- Data loading performance will suffer
- Use common sense while choosing the transformations

Useful libraries:

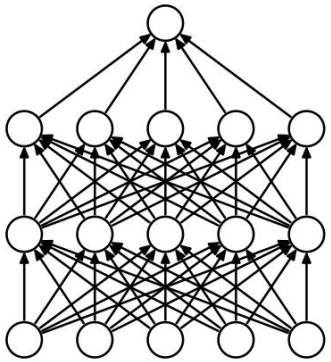
- Albumentations
- Imgaug
- torchvision.transforms



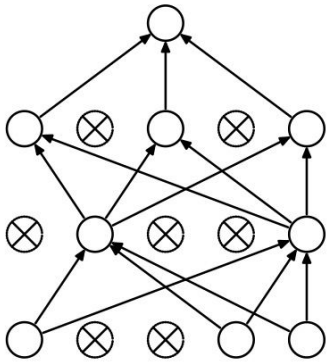
TTA (Test Time Augmentation)



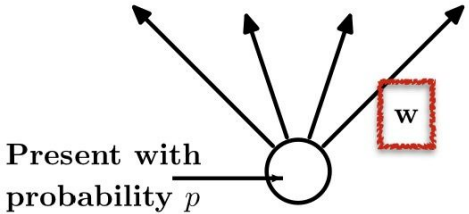
Overfitting - Dropout



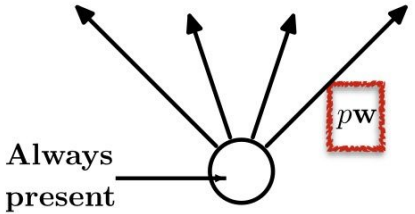
(a) Standard Neural Net



(b) After applying dropout.



(c) At training time

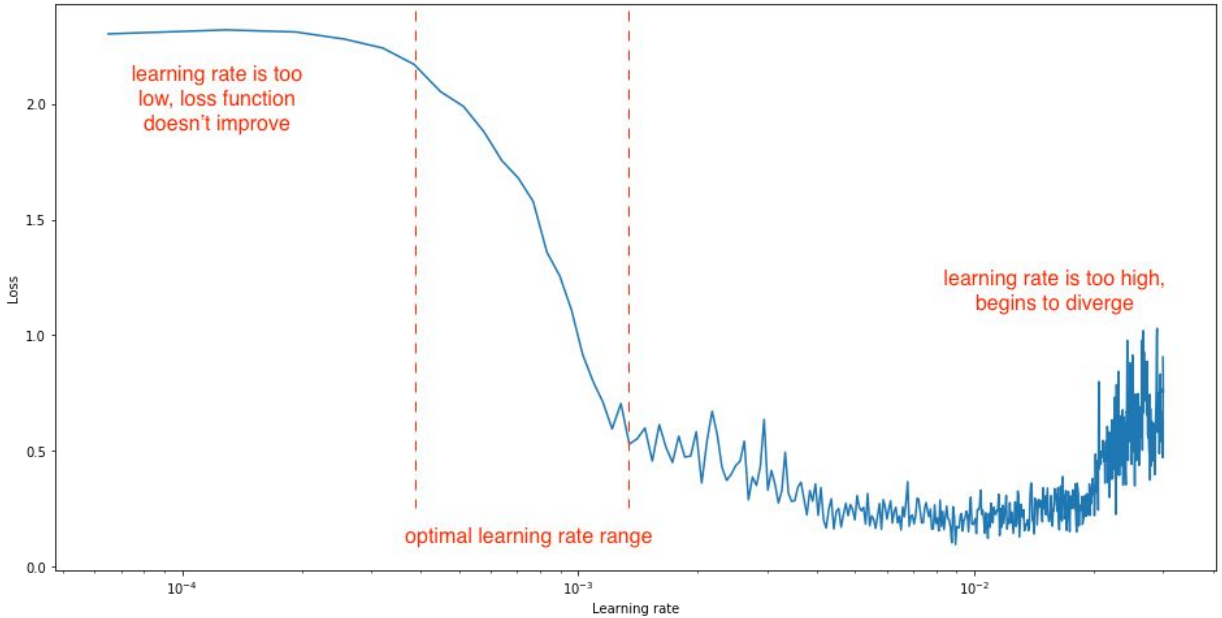


(d) At test time

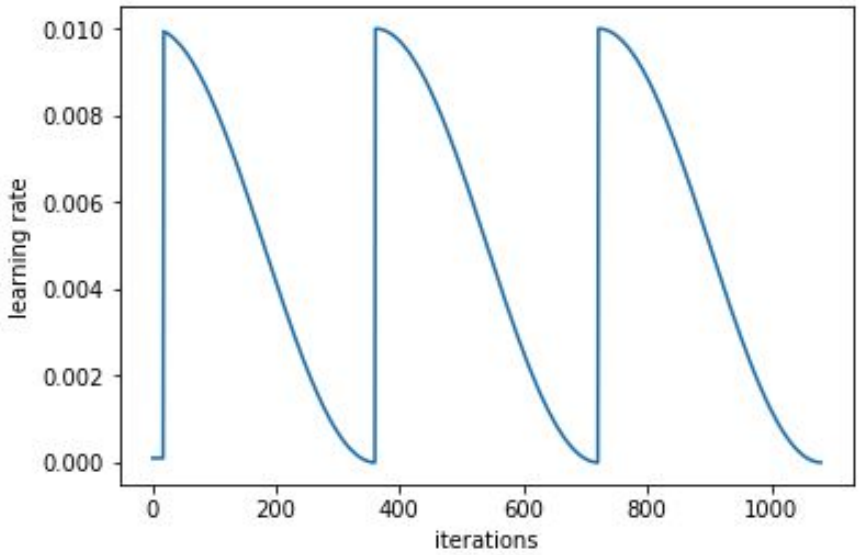
Overfitting - Early Stopping

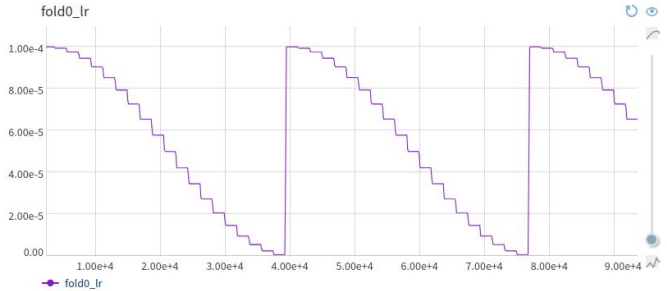
Finding a good Learning Rate

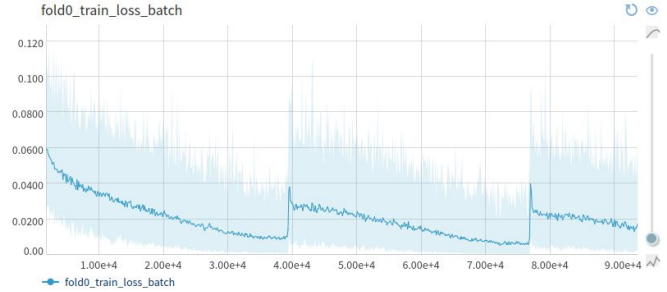
Simple idea:
We are passing batches through the network and updating weights starting with very low learning rates up to very high ones.
Observing this curve can give us hints on what the optimal LR might be.



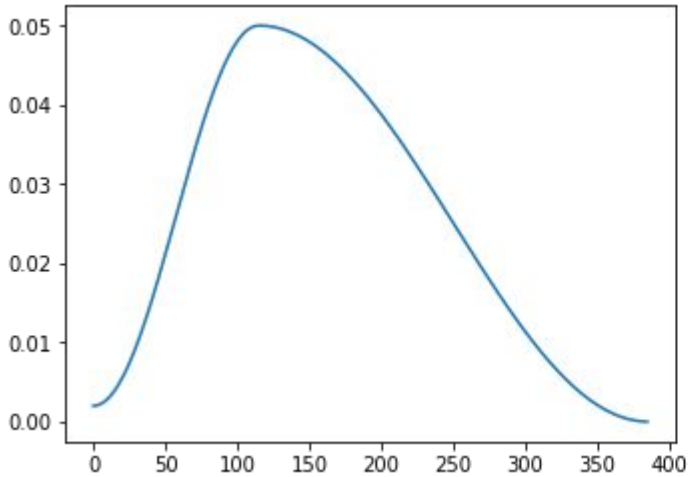
Cyclical Learning Rates (Cosine Annealing \w Restarts)

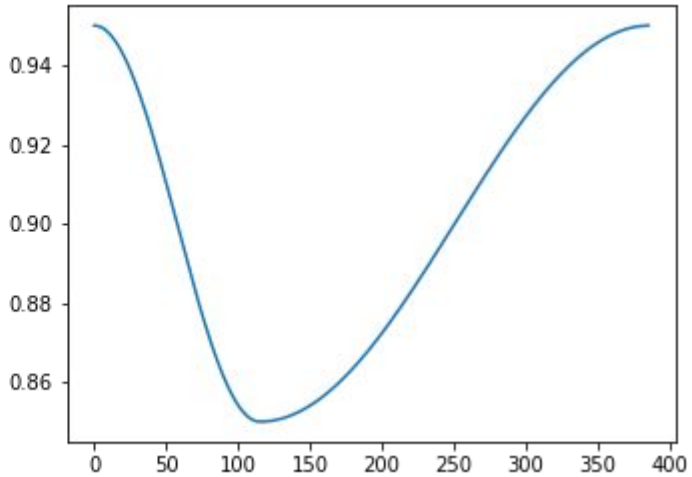






One Cycle Policy





Snapshot Ensambling

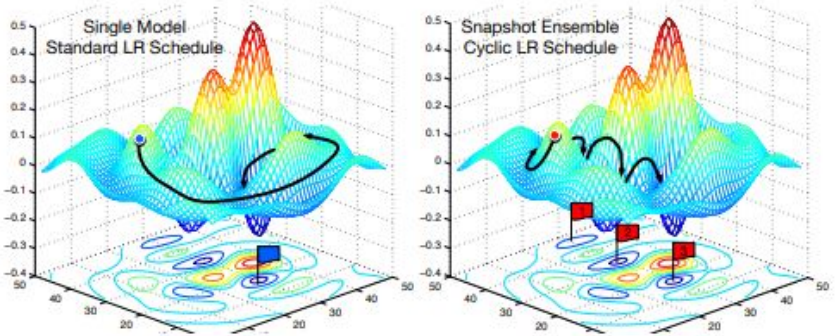


Figure 1: **Left:** Illustration of SGD optimization with a typical learning rate schedule. The model converges to a minimum at the end of training. **Right:** Illustration of Snapshot Ensembling. The model undergoes several learning rate annealing cycles, converging to and escaping from multiple local minima. We take a snapshot at each minimum for test-time ensembling.

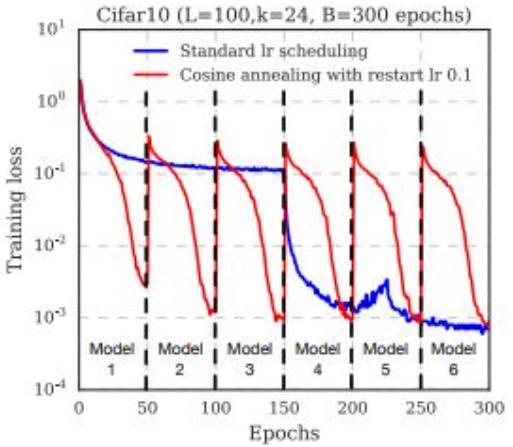
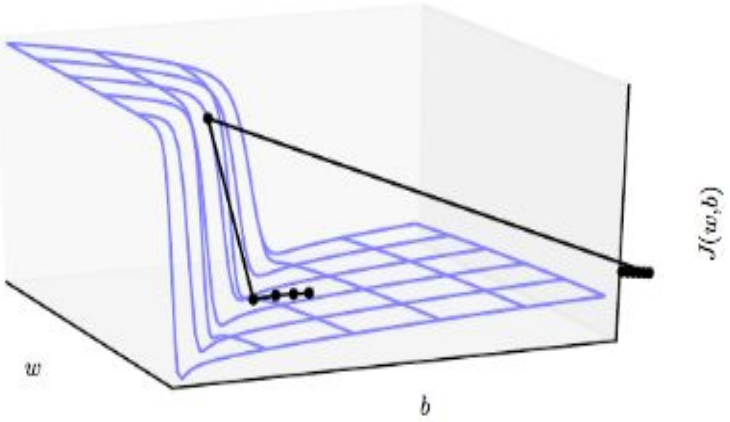


Figure 2: Training loss of 100-layer DenseNet on CIFAR10 using standard learning rate (blue) and $M = 6$ cosine annealing cycles (red). The intermediate models, denoted by the dotted lines, form an ensemble at the end of training.

Gradient Clipping




```
if clip:
    torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
```

Your task for next week.

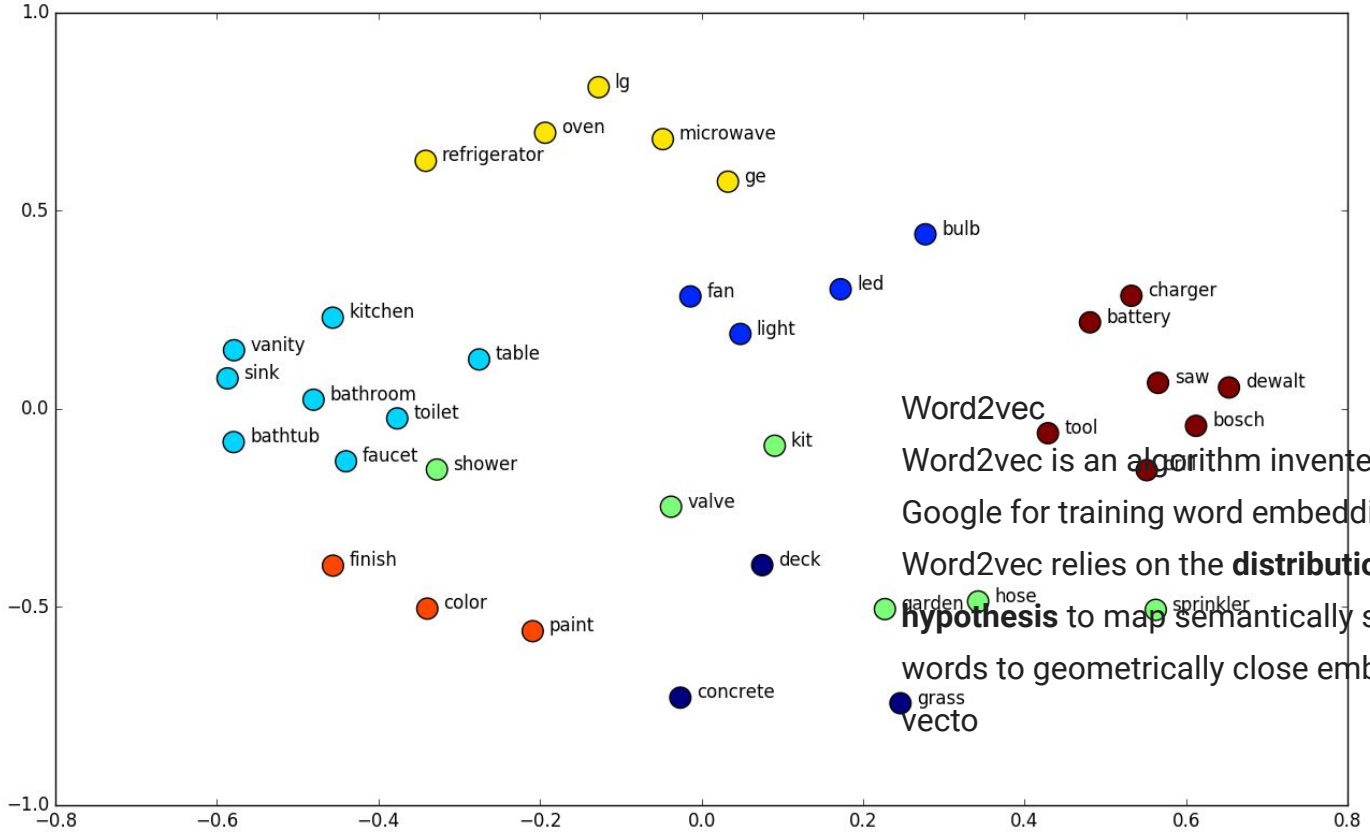
- Set up a system where you have access to a GPU.
- Create and interesting dataset by scraping google-images:
  - https://google-images-download.readthedocs.io/en/latest/examples.html#

- Train a basic classification model
- Validate it

Embeddings

An embedding is a relatively low-dimensional space into which you can translate high-dimensional vectors.

Embeddings make it easier to do machine learning on large inputs like sparse vectors representing words. Ideally, an embedding captures some of the semantics of the input by placing semantically similar inputs close together in the embedding space. An embedding can be learned and reused across models.



Word2vec

Word2vec is an algorithm invented at Google for training word embeddings. Word2vec relies on the **distributional hypothesis** to map semantically similar words to geometrically close embedding vecto

Deep Learning for tabular Data - Categorical Embeddings

Sunday	[.8, .2, .1, .1]
Monday	[.1, .2, .9, .9]
Tuesday	[.2, .1, .9, .8]

Embeddings **capture richer relationships and complexities than the raw categories** (one-hot encoded)
They also are more memory efficient (in most cases)

Embedding layer in pytorch

```
number_of_categories = 7
embedding_vector_size = 4
embedding = nn.Embedding(number_of_categories, embedding_vector_size)
```

```
print(embedding.weight.shape)
embedding.weight
```

```
torch.Size([7, 4])

Parameter containing:
tensor([[ 1.5269, -0.0295,  1.2453, -2.1048],
        [ 0.4434, -0.1562, -1.7598, -1.1095],
        [ 0.2742,  1.1970, -0.4884,  1.7665],
        [-0.2231,  0.7523,  0.4051,  0.3121],
        [-0.0547, -1.0645, -2.1366, -1.2324],
        [-0.4949,  0.6056,  0.1188, -1.4783],
        [-0.0629,  0.9525,  0.3891,  0.4428]], requires_grad=True)
```