Edge Computing for Computer Vision

By Ahsan Ijaz

GPUs for Deep Learning … WHY ???

- Optimized for training
- Process multiple computations simultaneously.
- Large cores, for better computation of multiple parallel processes
- Large amount of data ? GPU's memory bandwidth most suitable.

Google Colab

- Hosted Jupyter notebook
- No setup needed
- Excellent free version
- Free access to Google computing resources like GPUs and TPUs

Kaggle

AirBnB for Data Scientists

- **no-setup, customizable, Jupyter Notebooks environment. Access free GPUs and a huge repository of community published data & code**
- **Crowd-sourced platform to attract, nurture, train and challenge data scientists**
- **Data scientists are theorist … some hand on practice with Kaggle**
- **Code and Datasets**

Epoch

ENTIRE dataset is passed through the neural network only ONCE

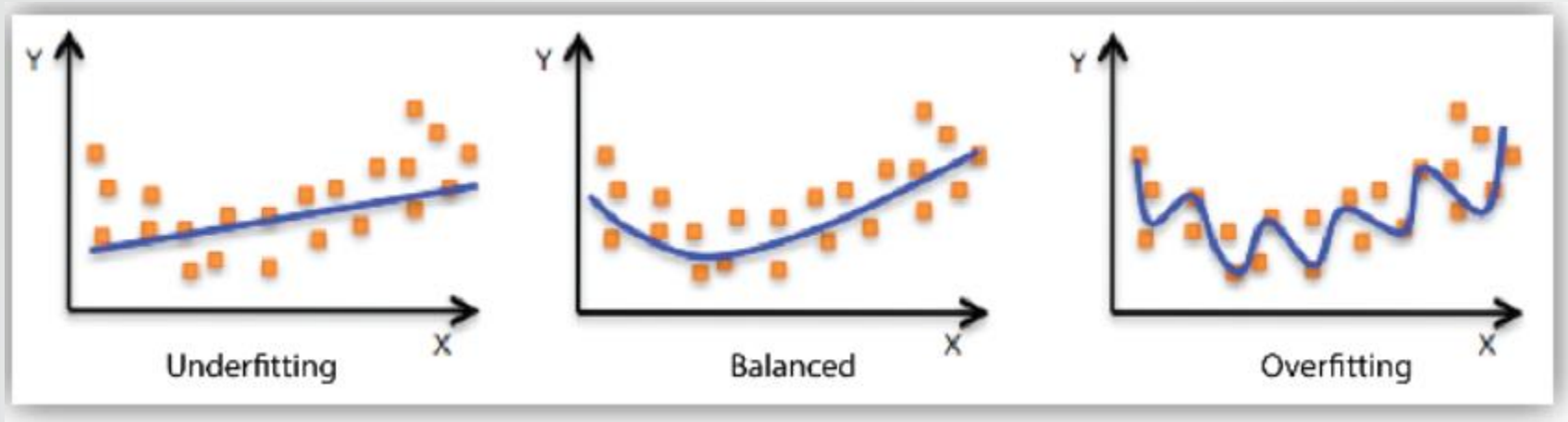
Iteration

Total number of training examples present in a single batch.

Batch

Number of batches needed to complete one epoch.

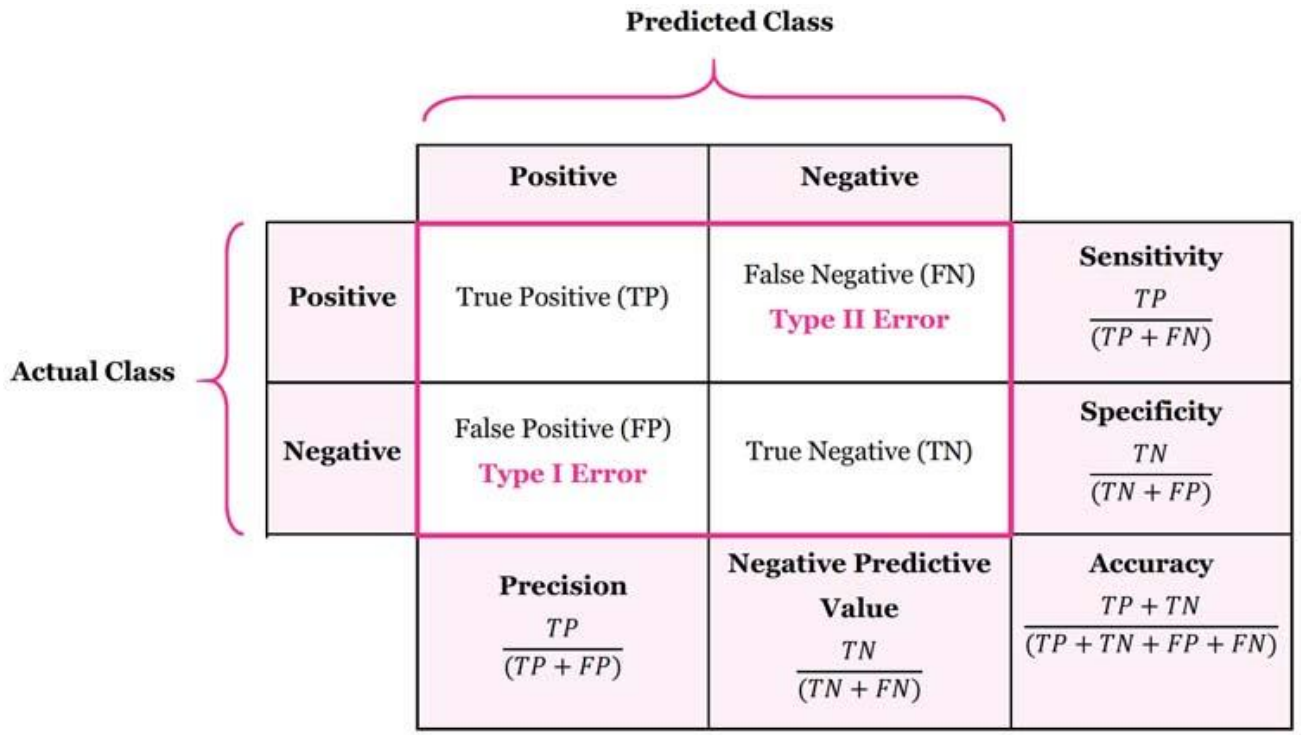
Underfitting & Overfitting



- Model performs poorly on the training data

- Model performs well on the training data
  - Does not perform well on the evaluation data
  - Unable to generalize

Confusion Matrix



Actual Class \ Predicted Class	Positive	Negative	
Positive	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP+FN)}$
Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$
	Precision $\frac{TP}{(TP+FP)}$	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

Metrices used for Accurate predictions

Performance Measures for Classification

$$recision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

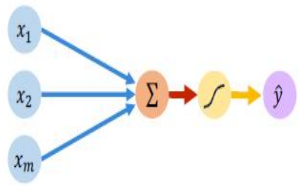
$$F1 = \frac{2 \times precision \times rec}{precision + recali}$$

$$ccuracy = \frac{TP + TN}{TP + FN + TN + F}$$

$$ecificity = \frac{TN}{TN + FP}$$

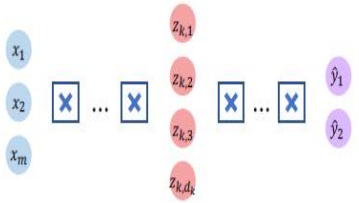
The Perceptron

- Structural building blocks
- Nonlinear activation functions



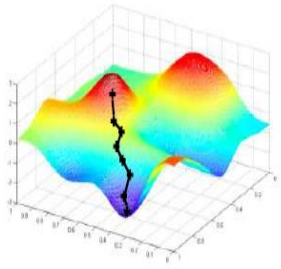
Neural Networks

- Stacking Perceptrons to form neural networks
- Optimization through backpropagation

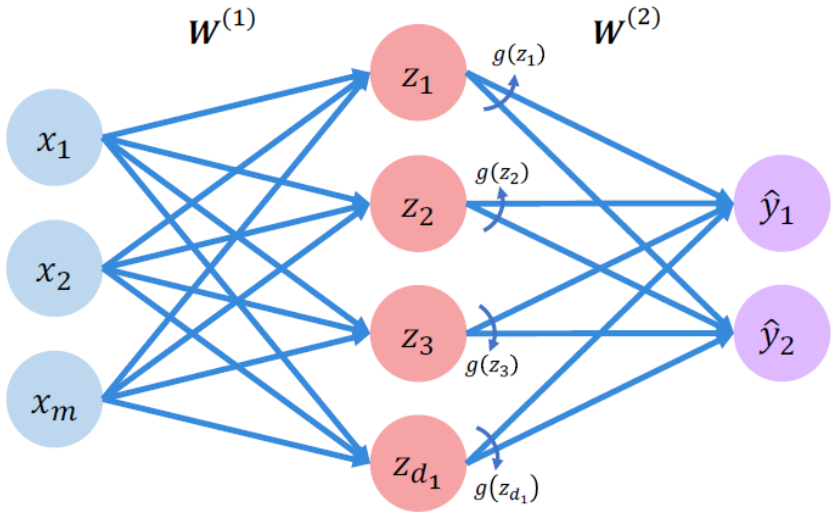


Training in Practice

- Adaptive learning
- Batching
- Regularization



Single Layer Neural Network



Inputs Hidden Final Output

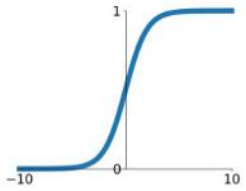
$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j \, w_{j,i}^{(2)}\right)$$

multilayer perceptron

activation functions

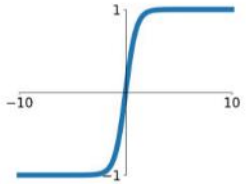
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



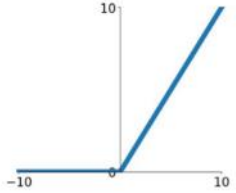
tanh

$$\tanh(x)$$



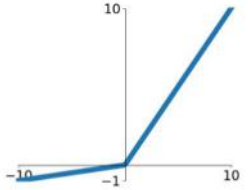
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

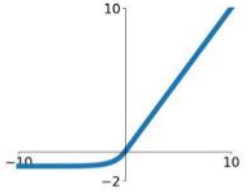


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Vanilla update

NUST SEECS

- The simplest form of update is to change the parameters along the negative gradient direction (since the gradient indicates the direction of increase, but we usually wish to minimize a loss function)

- Assuming a vector of parameters **x** and the gradient **dx**, the simplest update has the form:

```
# Vanilla update
x += - learning_rate * dx
```

where **learning_rate** is a hyperparameter - a fixed constant

- When evaluated on the full dataset, and when the learning rate is low enough, this is guaranteed to make non-negative progress on the loss function

Learning Optimizers

Momentum update

NUST SEECS

- ***Momentum*** or ***SGD with momentum*** is method which helps accelerate gradients vectors in the right directions, thus leading to faster converging
- It modifies gradient descent in two ways that make it more similar to the physical picture
  - First, it introduces a notion of "velocity" for the parameters we're trying to optimize
    - The gradient acts to change the velocity, not (directly) the "position", in much the same way as physical forces change the velocity, and only indirectly affect position
  - Second, the momentum method introduces a kind of friction term, which tends to gradually reduce the velocity

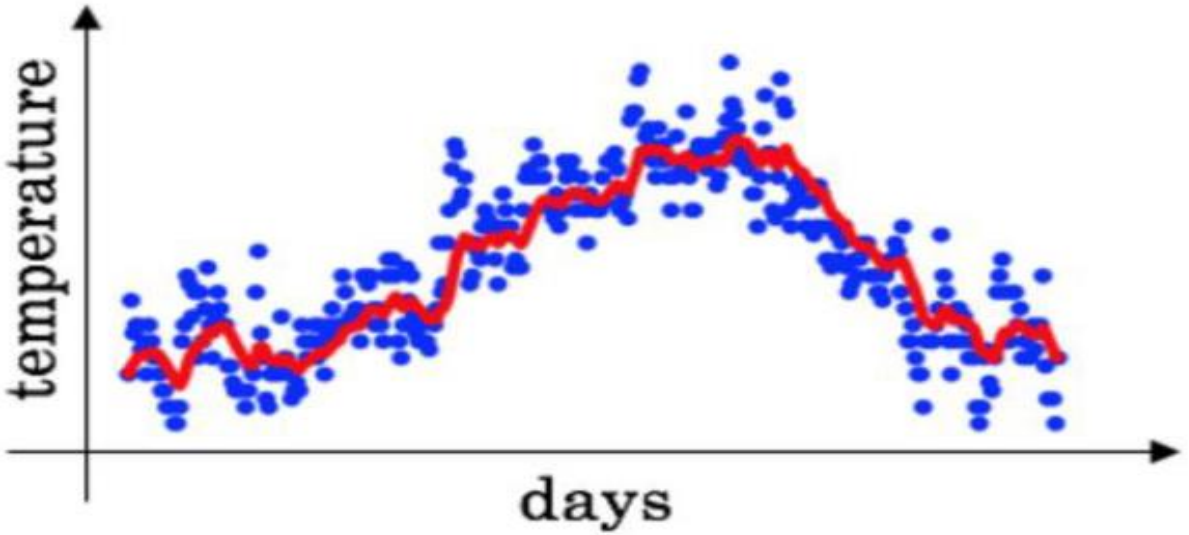
```
# Momentum update
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

Exponential Weighted Average

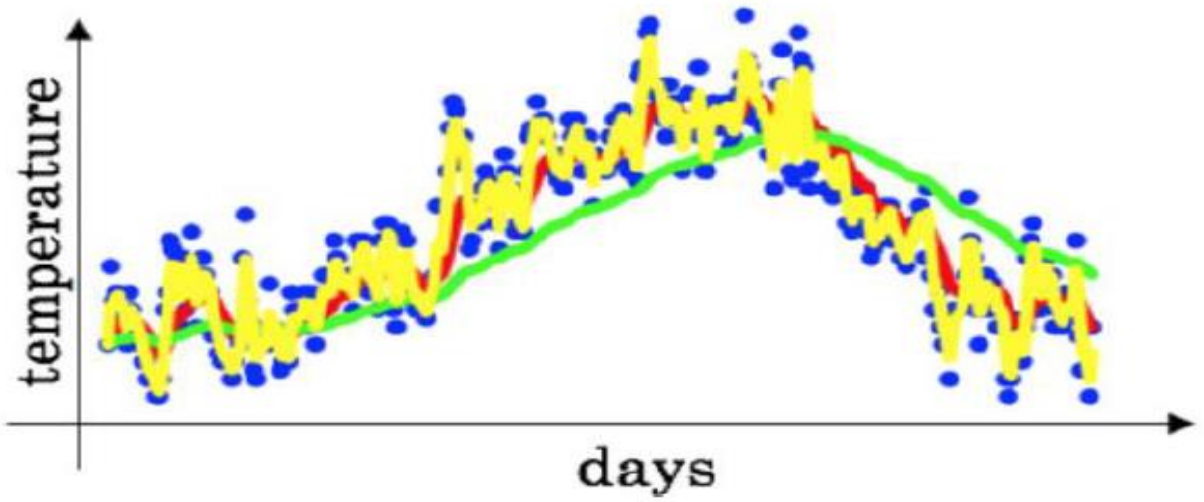
NUST SEECS

May think averaging over $\approx \frac{1}{1-\mu}$

i.e., for $\mu =$ 0.5, 0.9 and 0.98, we are getting estimates by averaging over 2, 10, and 50 last values respectively



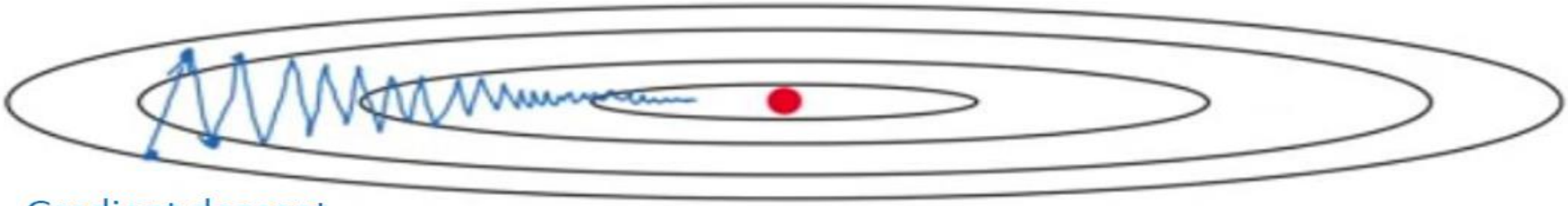
$\mu = 0.9$



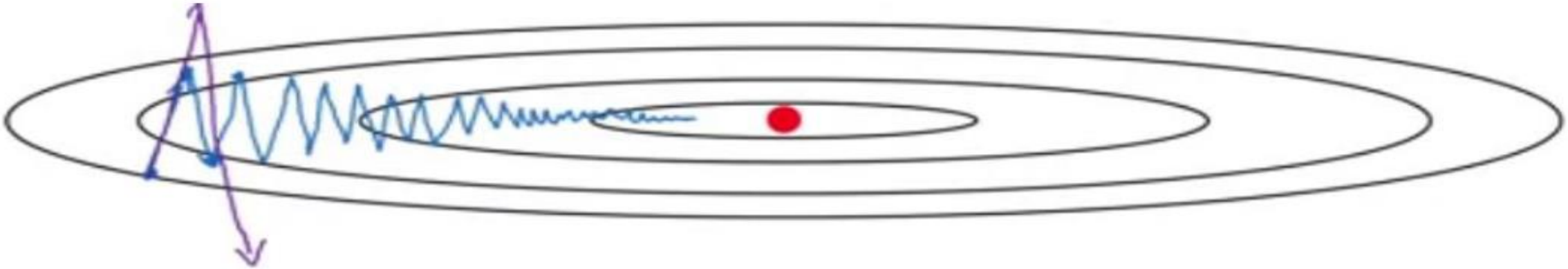
$\mu = 0.5$
$\mu = 0.9$
$\mu = 0.98$

Gradient Descent with Momentum

NUST SEECS



Gradient descent



Gradient descent with
higher learning rate

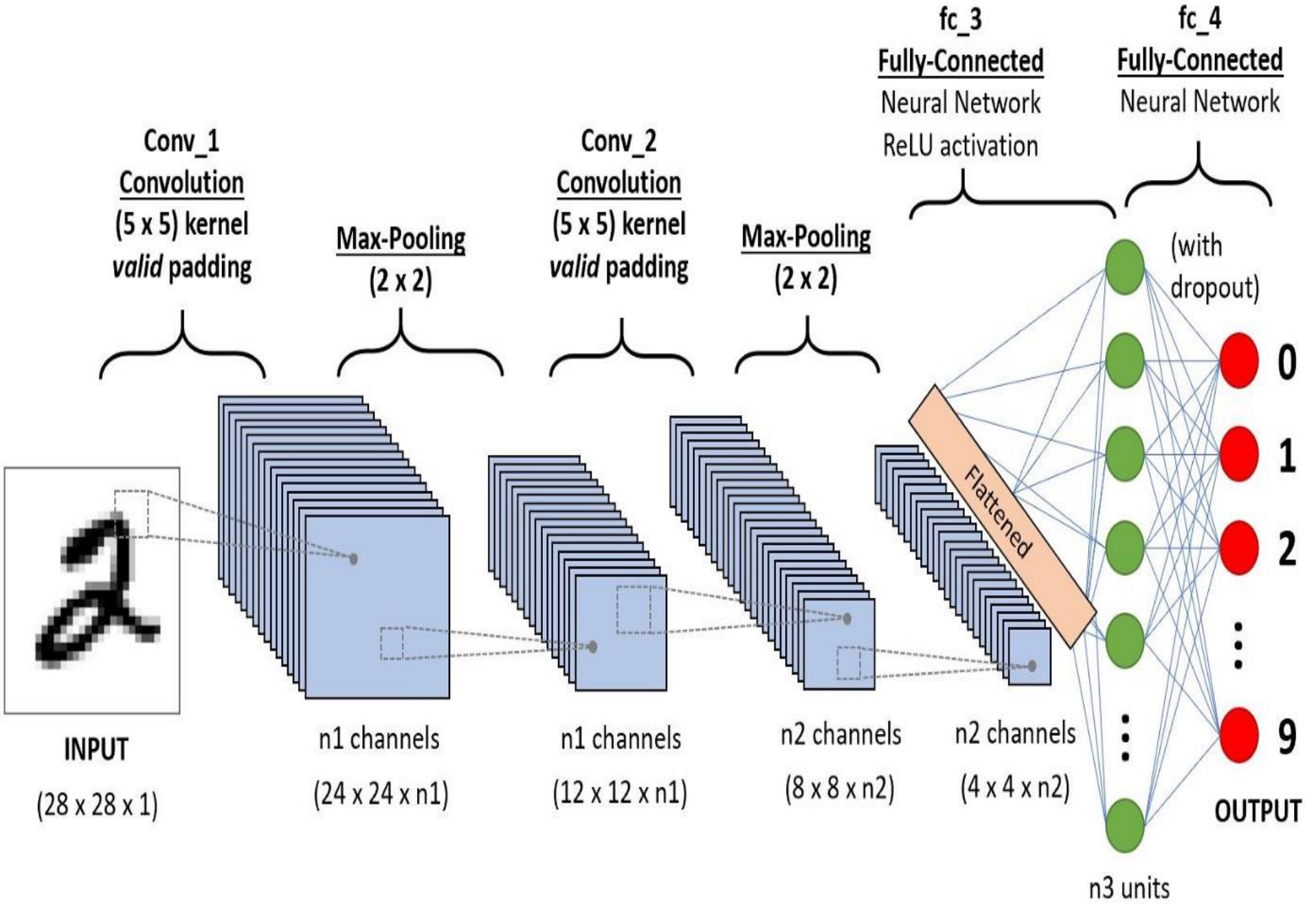
Slower learning

Faster learning

Learning Optimizers (conti...)

Some others are:

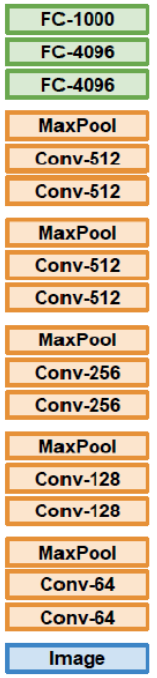
- **NesterovMomentum**
- **RMSProp**
- **AdaGrad**
- **Adam**



convolutional neural network

Transfer learning

focuses on storing knowledge gained while solving one problem and applying it to a different but related problem



More specific

More generic

	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 larger number of layers

TensorFlow

- free and open-source software library
- particular focus on training and inference of deep neural networks
- a symbolic math library based on dataflow and differentiable programming.

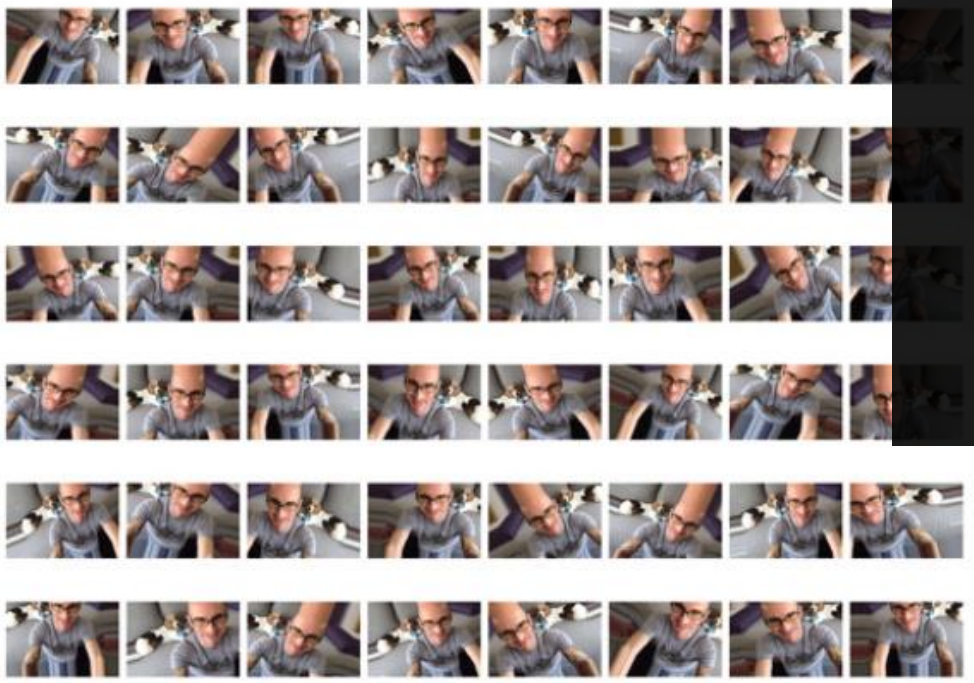
classification in artificial intelligence

- a predictive modeling problem where a class label is predicted for a given example of input data

Data Augmentation

- create variations of the images that can improve the ability of the fit models to generalize what they have learned to new images.
- This helps to increase the performance of the model by generalizing better and thereby reducing overfitting

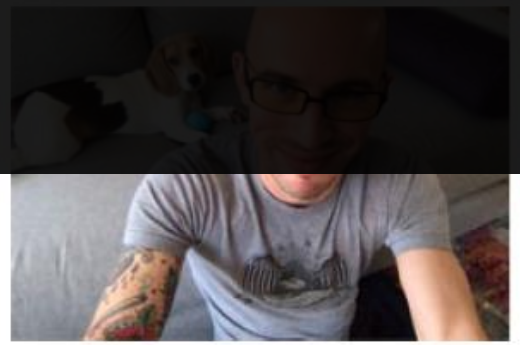
- Offline
  - Make copies of training images with various transformations (Geometric / Photometric / Affine)
  - Rotation
  - Flipping
  - Elastic Deformation
  - Color Transformations
- Online
  - Using ImageDataGenerators

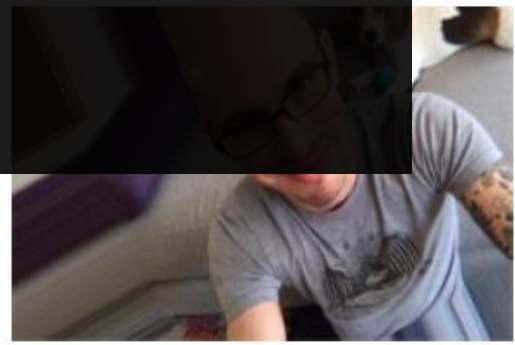


Input Image

Augmented Images

Keras





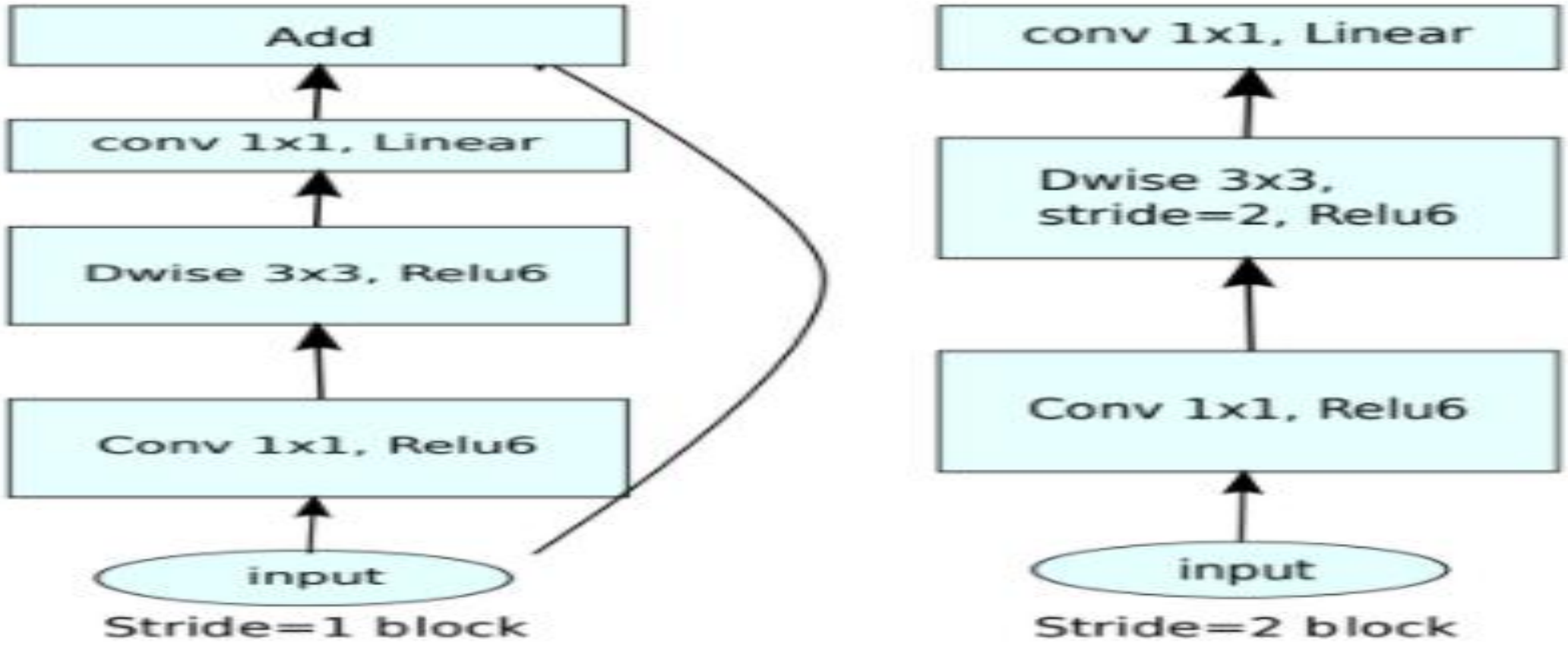
VISION

Scene Segmentation

MobileNetV2 (Object Detection)

https://www.youtube.com/watch?v=vfCvmenkbZA

- Based on an inverted residual structure where the residual connections are between the bottleneck layers.
- Intermediate expansion layer uses lightweight depthwise convolutions to filter features as a source of non-linearity
- Architecture contains fully convolution layer with 32 filters, followed by 19 residual bottleneck layers



(d) Mobilenet V2