深度学习与框架
Deep Learning and Frameworks

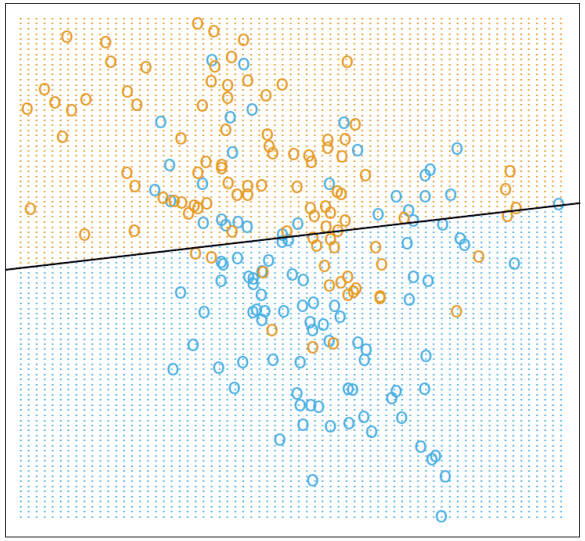
李洋 副教授

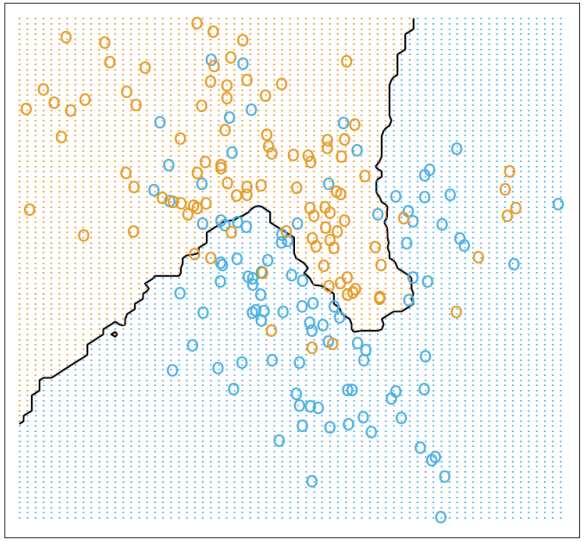
计算机科学与技术学院

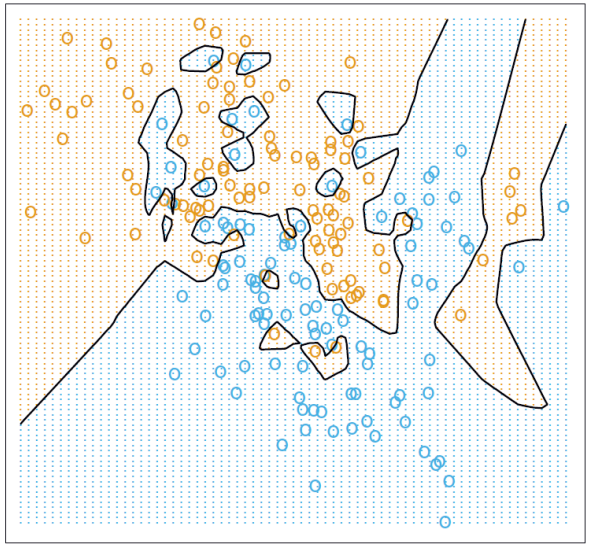
华东师范大学

YLI@CS.ECNU.EDU.CN

RECALL: Linear Separator vs. k nearest neighbor







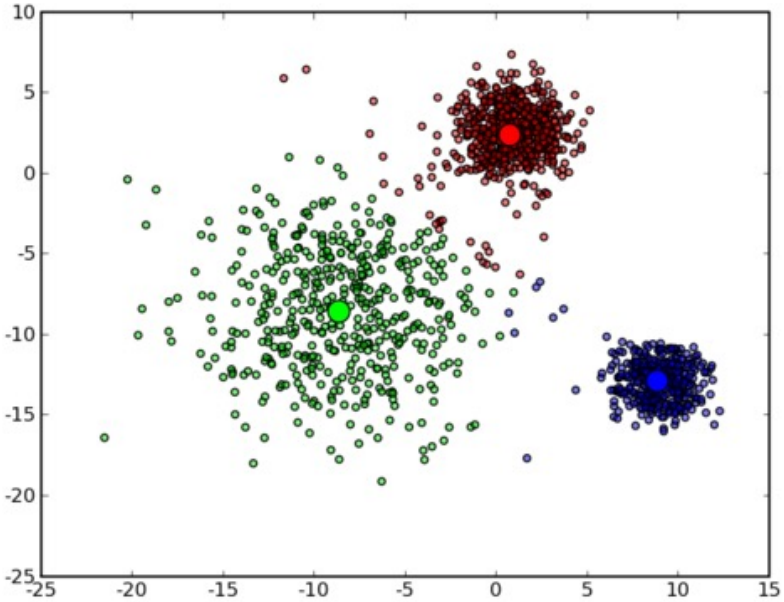
- Which one is better?
- How many parameters?
  - Effective number of parameters

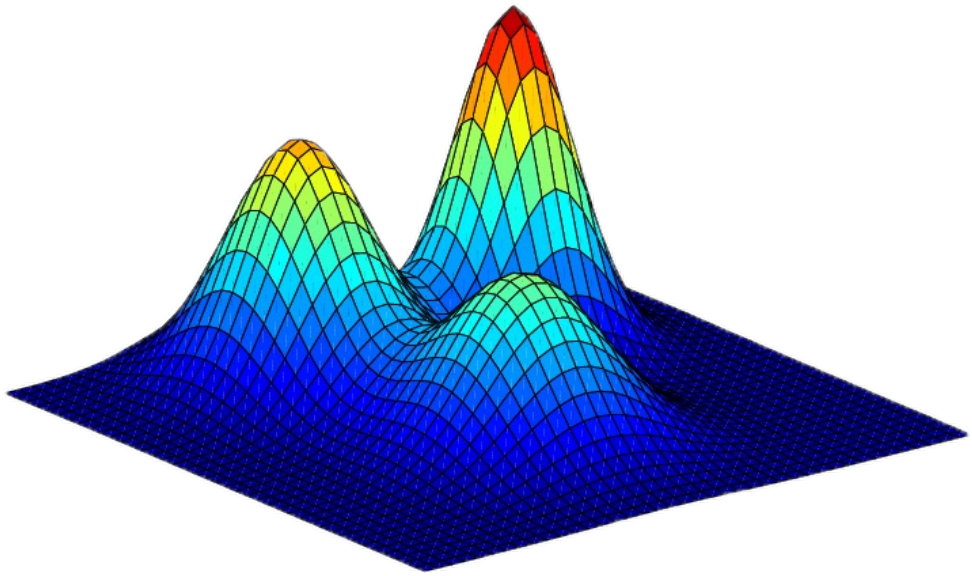
RECALL: The K-Means Clustering Method

- Given k, the k-means algorithm is implemented in four steps:
  1. Partition objects into k nonempty subsets
  2. Compute seed points as the centroids of the clusters of the current partitioning (the centroid is the center, i.e., mean point, of the cluster)
  3. Assign each object to the cluster with the nearest seed point
  4. Go back to Step 2, stop when the assignment does not change

RECALL: Gaussian Mixture Model

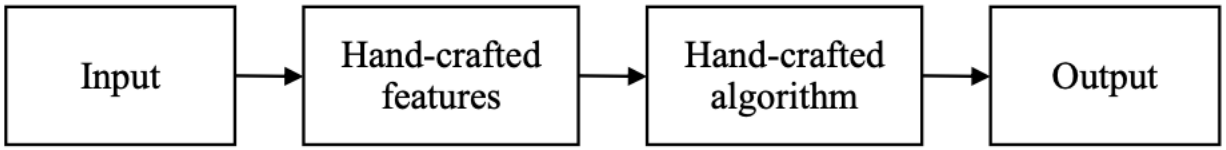
- Gaussian Mixture Model (GMM) is one of the most popular clustering methods which can be viewed as a linear combination of different Gaussian components.



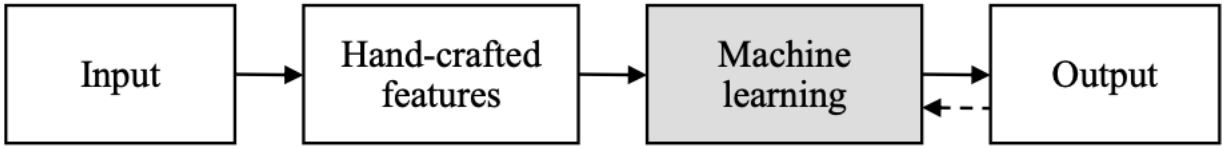


Big map

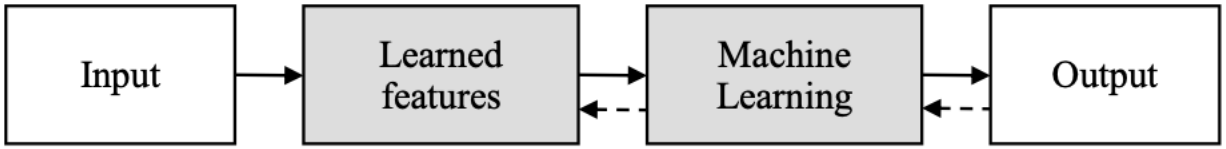
- All we talked is about very traditional methods to deal with perception problem
- Do we have alternative way to solve the problem?



(a) Traditional vision pipeline

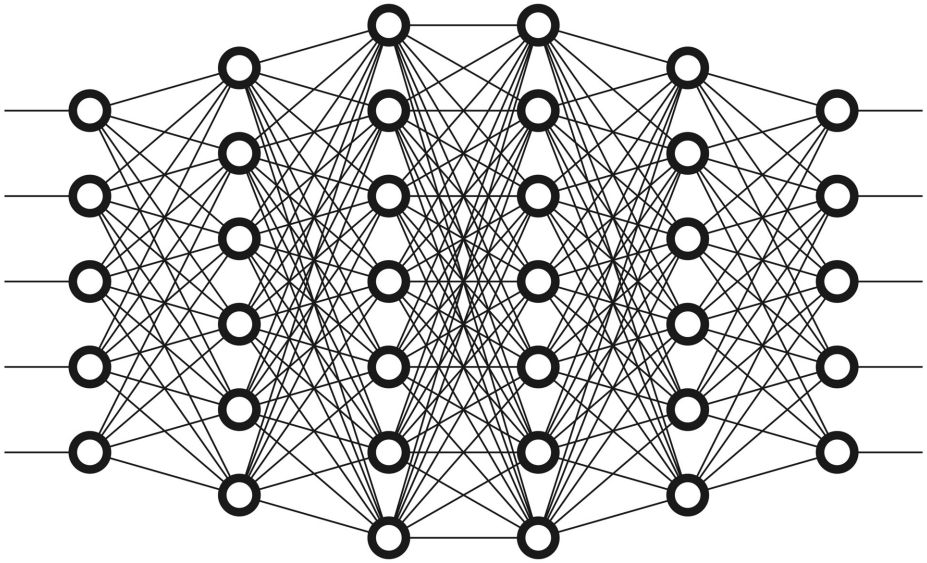


(b) Classic machine learning pipeline



Deep learning

- **Deep learning** (also known as **deep structured learning**) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised. (wiki)



History

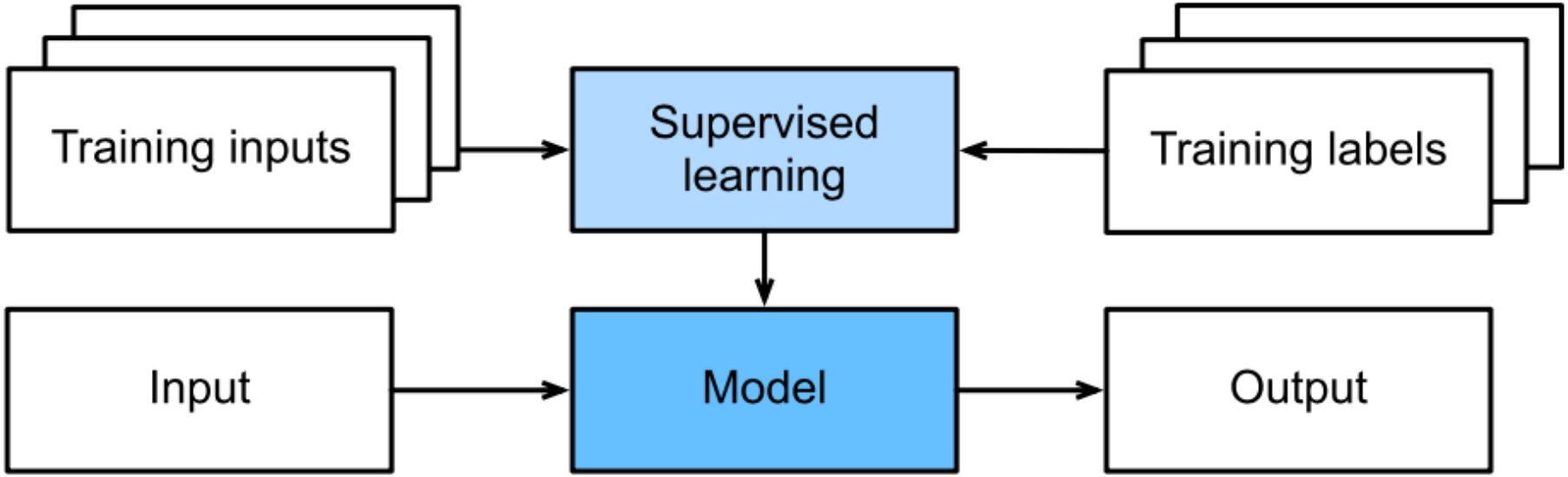
- 1958: Frank Rosenblatt creates the perceptron, an algorithm for pattern recognition based on a two-layer computer neural network using simple addition and subtraction
- multilayer perceptrons was published by Alexey Ivakhnenko and Lapa in 1967
- In 1989, Yann LeCun et al. applied the standard backpropagation algorithm
- 2006s: The term “deep learning” begins to gain popularity after a paper by Geoffrey Hinton and Ruslan Salakhutdinov showed how a many-layered neural network could be pre-trained one layer at a time.
- In 2012, a similar system by Krizhevsky et al.[6] won the large-scale ImageNet competition by a significant margin over shallow machine learning methods. Near human performance in MNIST by Cireşan, Dan
- 2014, Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In ACM International Conference on Multimedia. ACM, 675–678.
- In March 2019, Yoshua Bengio, Geoffrey Hinton and Yann LeCun were awarded the Turing Award for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing.

Fathers of the Deep Learning

- Revolution Receive ACM A.M. Turing Award 2018
- Bengio, Hinton and LeCun Ushered in Major Breakthroughs in Artificial Intelligence

Overall purpose

- Deep learning still follows all rules in ML area
- Usually, it is in a supervised learning setting



Neuron

- Deep neural networks (DNNs) are *feedforward* computation graphs composed of thousand of simple interconnected "neurons" (*units*)
- The unit perform weighted sum

as

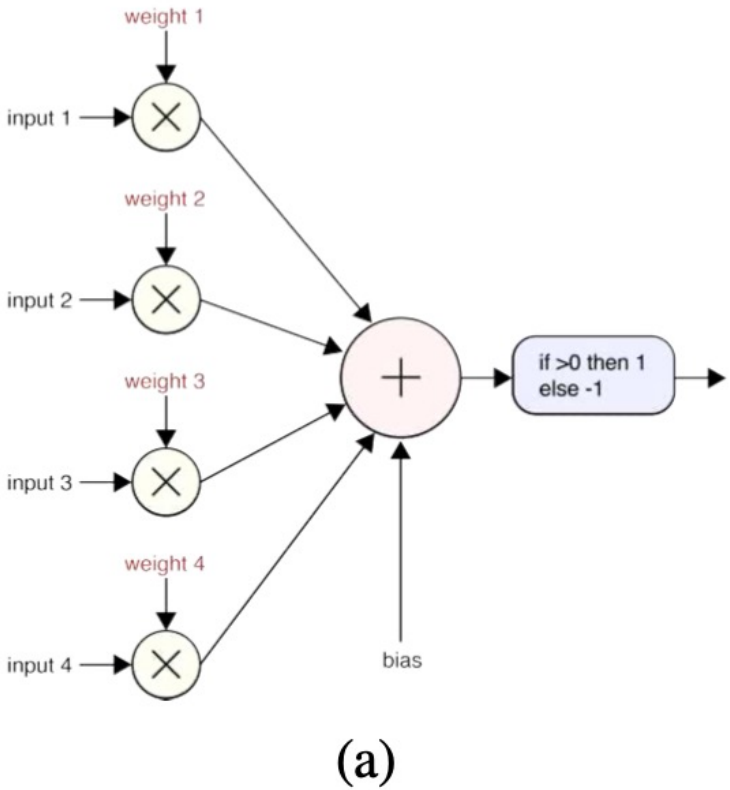
$$s_i = \mathbf{w}_i^T \mathbf{x}_i + b_i$$

- followed by a activation function

$$y_i = h(s_i),$$

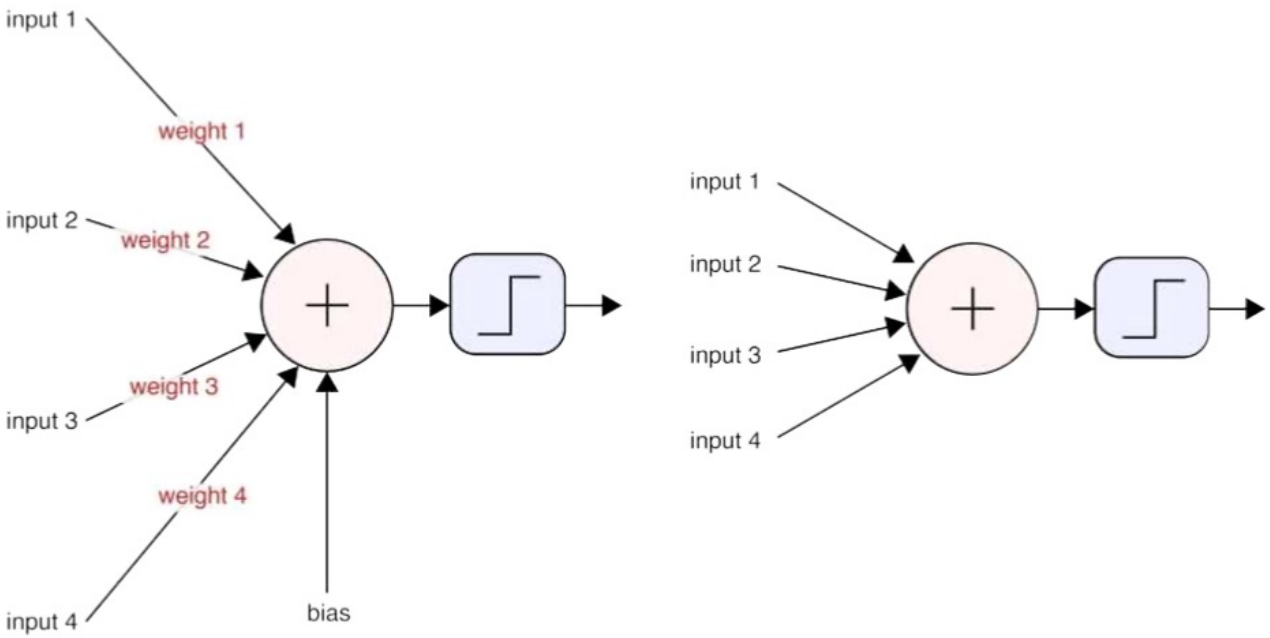
- It used to call *perceptrons*

(Rosenblatt 1958)



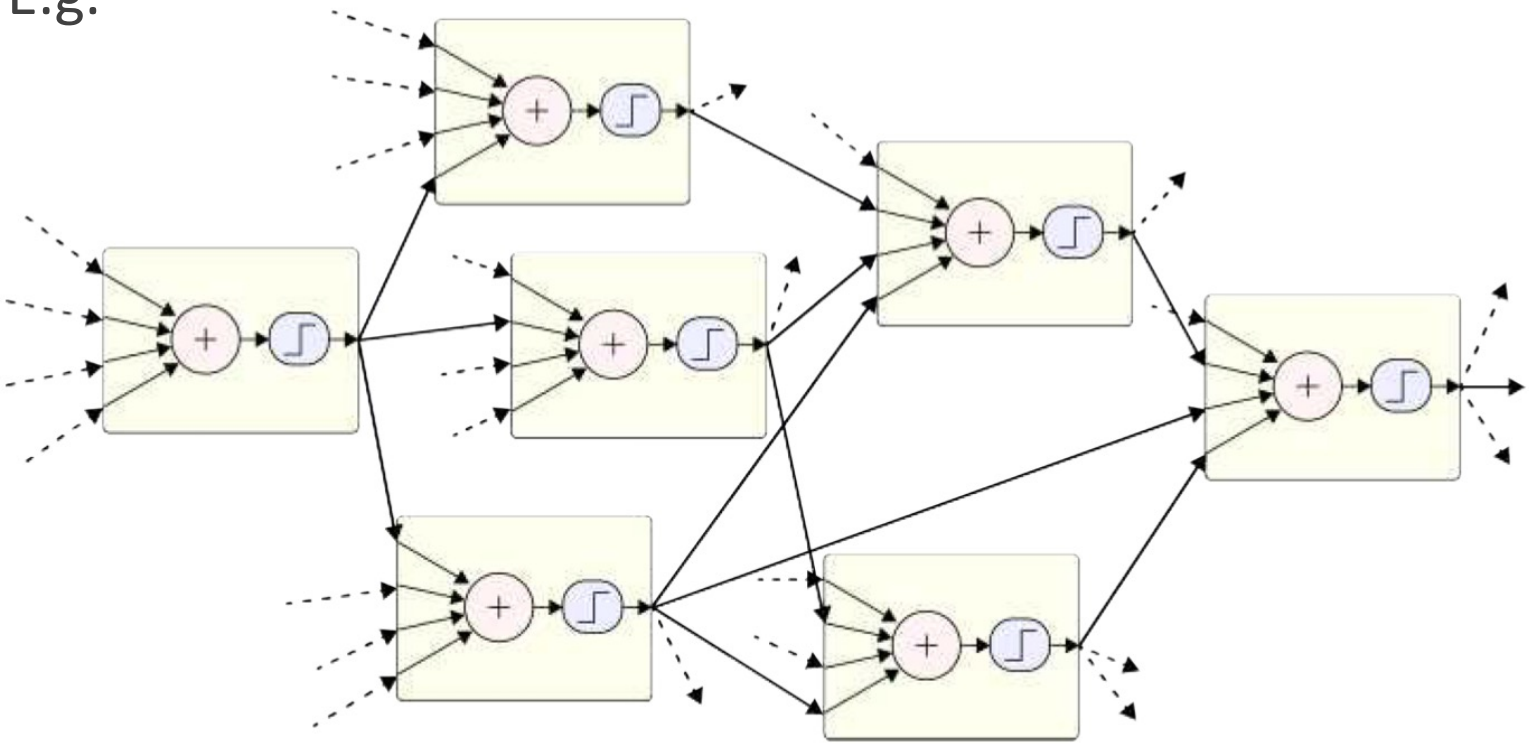
Weights and Layers

- The networks are represented naturally in a graph fashion
- For convenience, people hide the weights and activation function as well
- E.g.



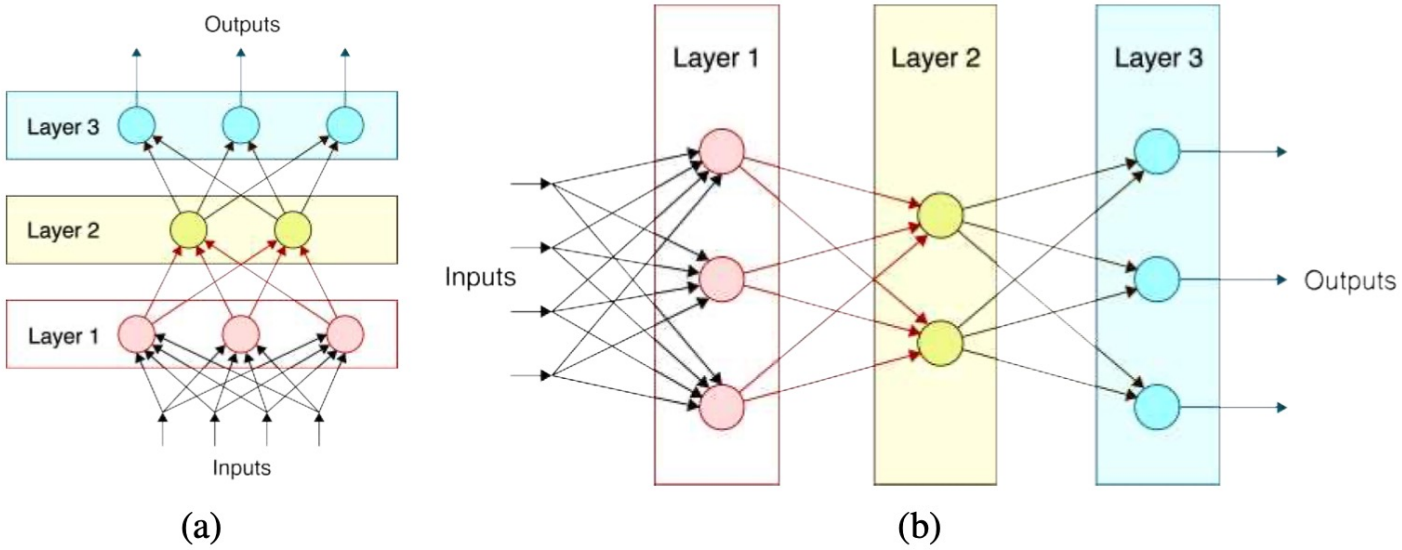
Weights and Layers

- In fact, we can use neurons to construct any neural networks. It can be very messed up
- E.g.

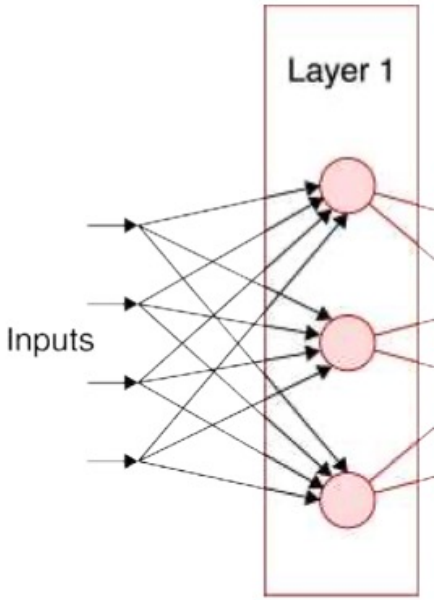


Weights and Layers

- But, people usually layer them up in an organized format
- When connecting all units between layers, we call it a *fully connected* (FC) layer
- A network that consist only of fully connected layers is now often called a *multi layer perceptron* (MLP).



Activation functions



- Without activation function, the whole DNN is just a linear transformation as

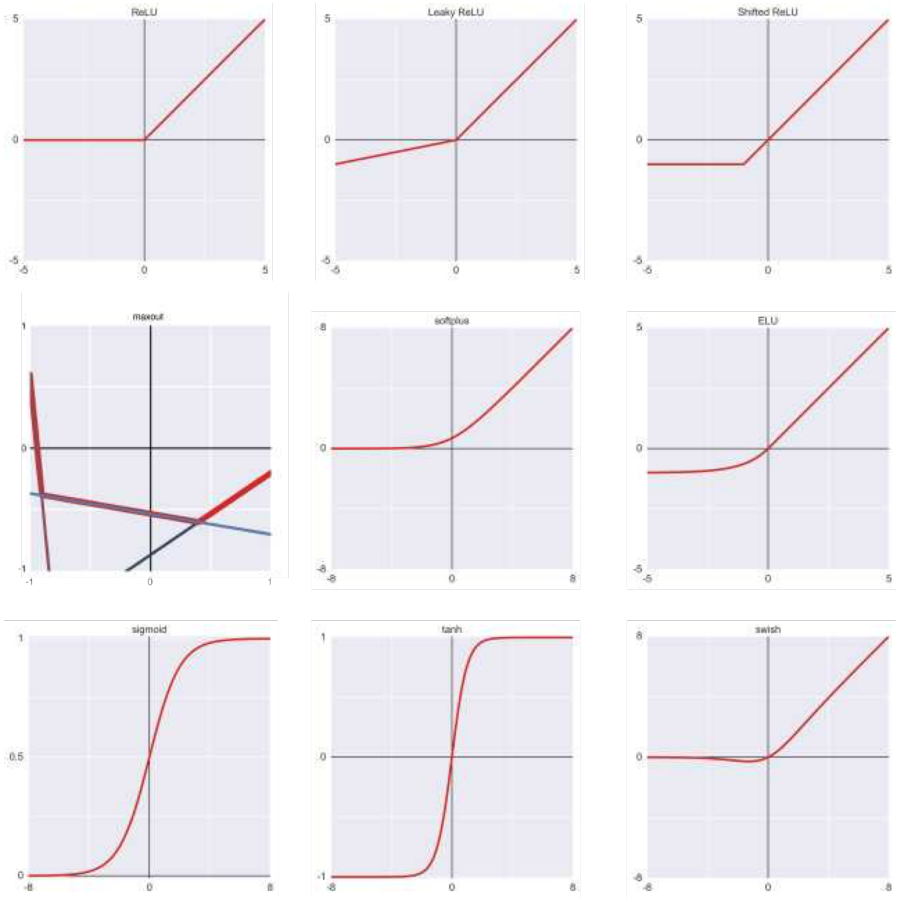
$$\mathbf{s}_l = \mathbf{W}_l \mathbf{x}_l,$$

- The activation functions give DNN non-linear power and solve the real world problem in an AI way.
- The current most popular activation function is Rectified Linear Unit (ReLU)

$$h(y) = \max(0, y)$$

Activation functions

- Here are some common activation functions
  - *ReLU*
  - *leaky ReLU*
  - *shifted ReLU*
  - *maxout*
  - *softplus*
  - *ELU*
  - *sigmoid*
  - *tanh*
  - *swish*
- Please note their range of output is different



Loss function

- In order to get the weights in the neural network, we need to define a loss function to tell the NN what kind of behaviors we want.
- We are already familiar with some types of loss/cost/measurement function

$$E(\mathbf{w}) = \sum_n E_n(\mathbf{w}) = -\sum_n \|\mathbf{y}_n - \mathbf{t}_n\|^2,$$

output of NN, $y=f(x_n;w)$

ground truth labeling

- This loss usually performs regression task, and we call it L_2 loss function.

Loss function

- Another commonly used loss is cross-entropy loss/logistic loss which is to tell NN to behave like a classifier,

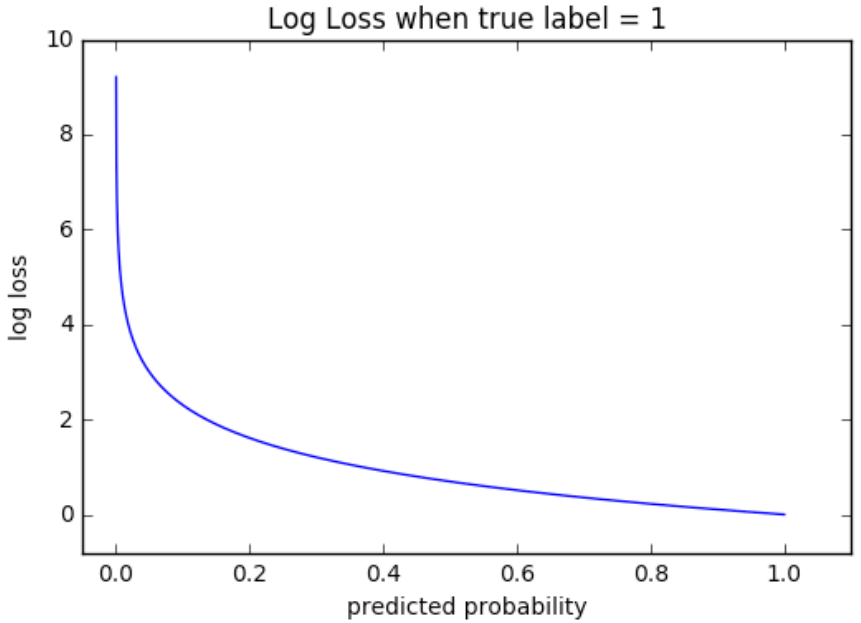
for binary

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) = -\left[t \log(p) + (1-t) \log(1-p)\right]$$

ground truth labeling

output of NN, $p=f(x_n;w)$

- the loss looks like this:
- The essential math is to match two distribution with KL measurement.



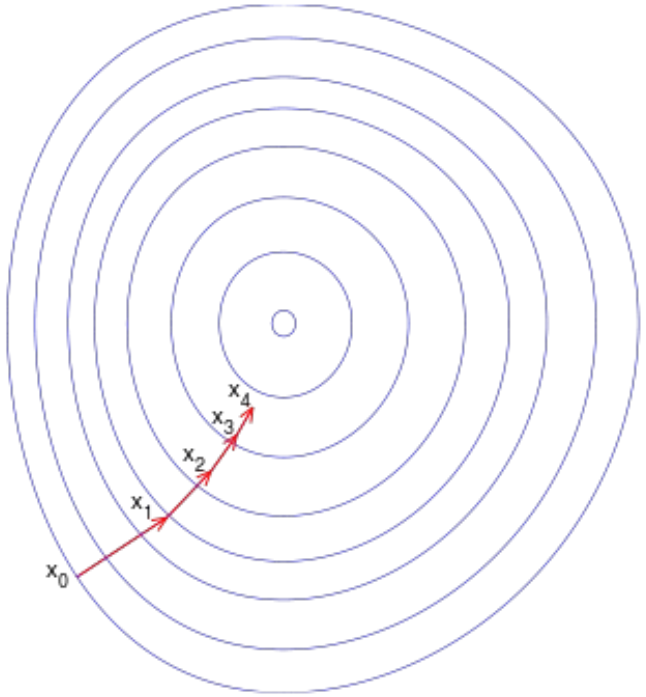
Gradient descent

- Once we have defined loss function and the structure of our neural network, we need to calculate the weights and parameters of the neural networks with training data
- Gradient descent is used to optimized the loss function and to get the weights of the NN

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

- Please recall the gradient is a direction of "change". We just need to ensure every time ,we have a better guess

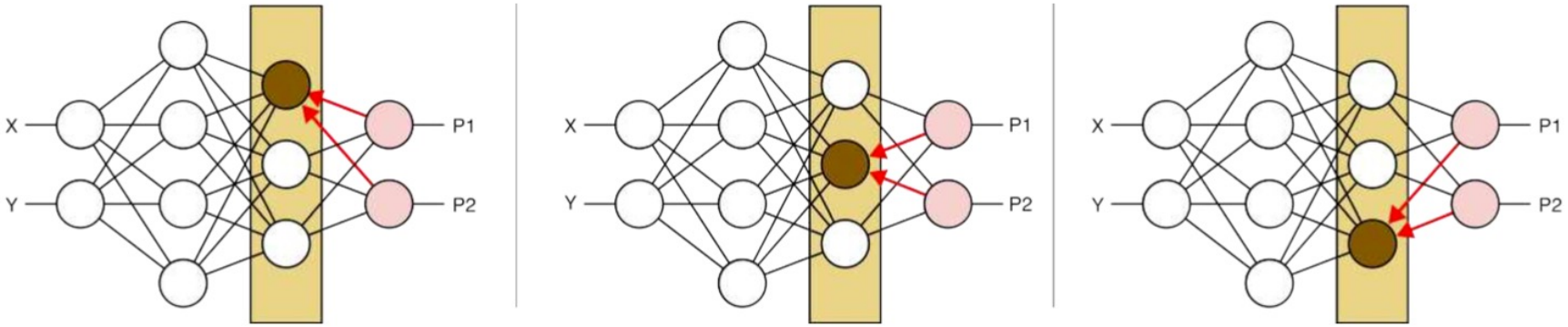
$$F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \cdots,$$



Backpropagation

- At the heart of deep learning, backpropagation is used to the process of calculating the gradient of the whole NN
- Recall the chain rule when you learn calculus

$$\frac{dC}{da^L} \cdot \frac{da^L}{dz^L} \cdot \frac{dz^L}{da^{L-1}} \cdot \frac{da^{L-1}}{dz^{L-1}} \cdot \frac{dz^{L-1}}{da^{L-2}} \cdots \frac{da^1}{dz^1} \cdot \frac{\partial z^1}{\partial x}.$$



Initialization

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

- Please note, gradient descent is a local optimization algorithm
- It cannot give a global optimal, which means initialization matters.
- Also, we need to make sure all weights are not zeros, otherwise the gradient descent fails
- A commonly used initialization is *He initialization*

```
import numpy as np
W = np.random.randn(node_in, node_out) / np.sqrt(node_in / 2)
```

Universal approximation theorem

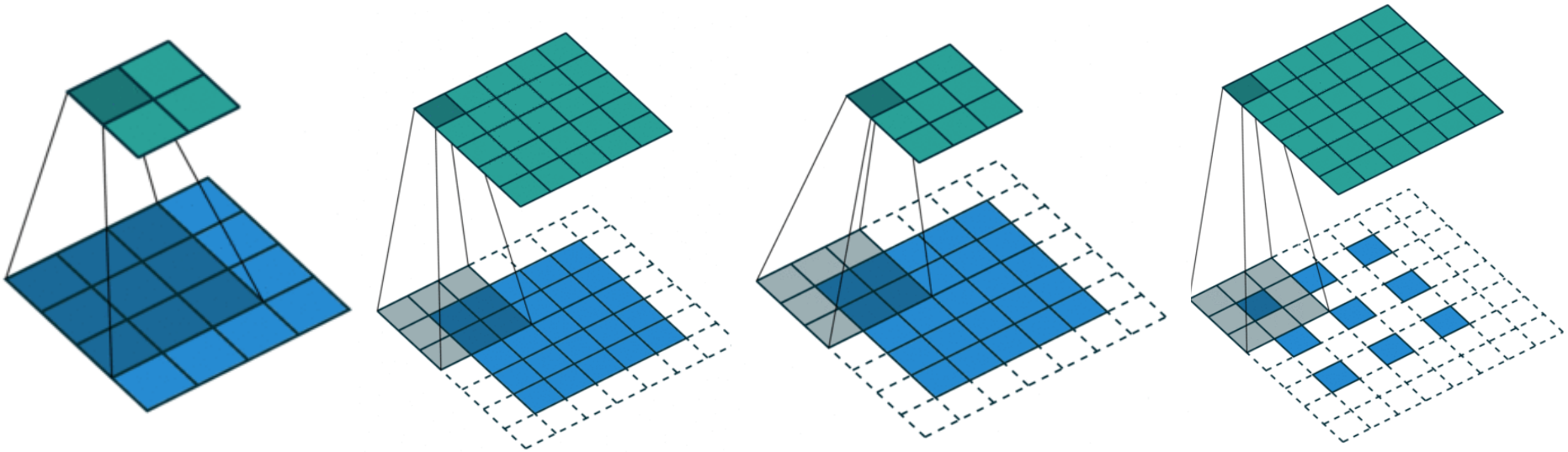
- In theory, you got everything with deep learning now!
- Universal approximation theorem states that, given an arbitrary-width NN, it can approximate any continuous function.
- Cheers!
- You got everything you need in Deep Learning
- in theory

In practice

- Sadly, in practice, there are still many problems in Deep learning.
- For example, deep learning is basically a machine learning approach. It means DL follows all restrictions in ML
  - overfitting
  - training problem
- Also, we cannot construct an arbitrary width NN

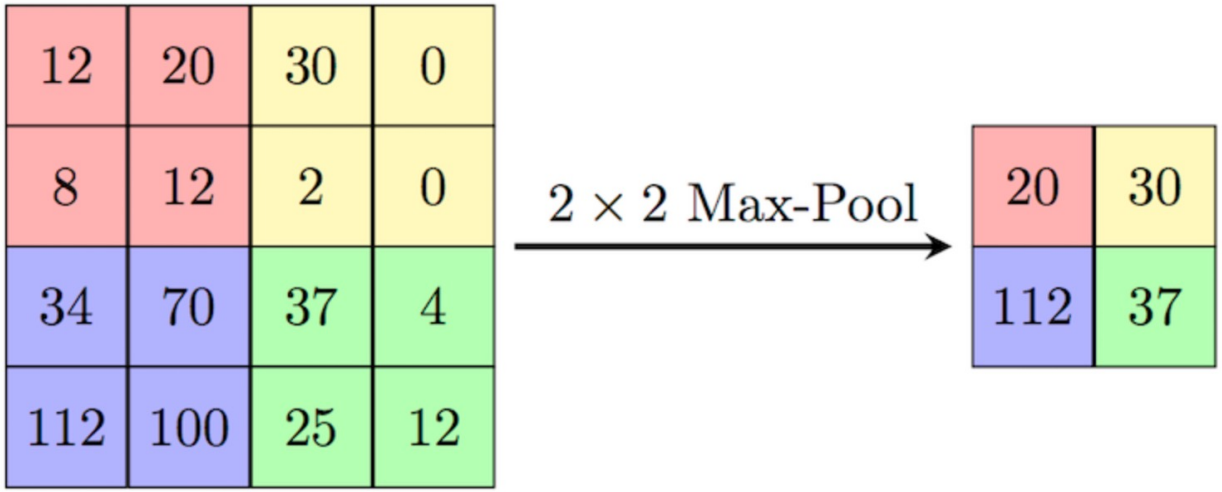
Convolutional layers

- Recall the convolutional operations, it is also a very important concept in DL
- reduces the weights
- translational invariant



Pooling

- To further improve the non-linear attribute of the neural network, pooling layer is added.
- The most popular one is max pooling
- It also has strides, and kernel size



Regularization

- Recall the overfitting problem in ML, it basically means we have more parameters in the ML model.
- The idea is to set parameters to zeros/small values

$$E_{\mathrm{W}} = \sum_{k} \|\mathbf{w}_k\|^p,$$

$$E(\{w_k\}) = E_{\mathrm{D}} + \lambda E_{\mathrm{W}}$$

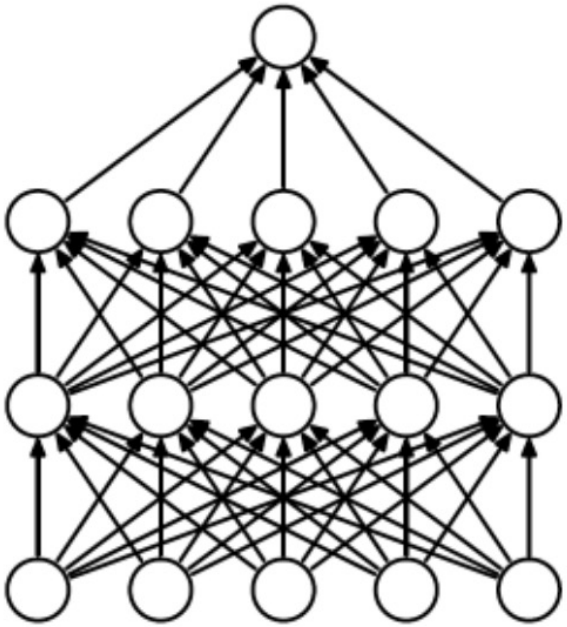
- When p = 2, we have least squares problem, and we call it weight decay in DL
- When p=1, we call it L_0 norm, and it can be solved by LASSO.
- It controls the weights towards zeros

Dataset augmentation

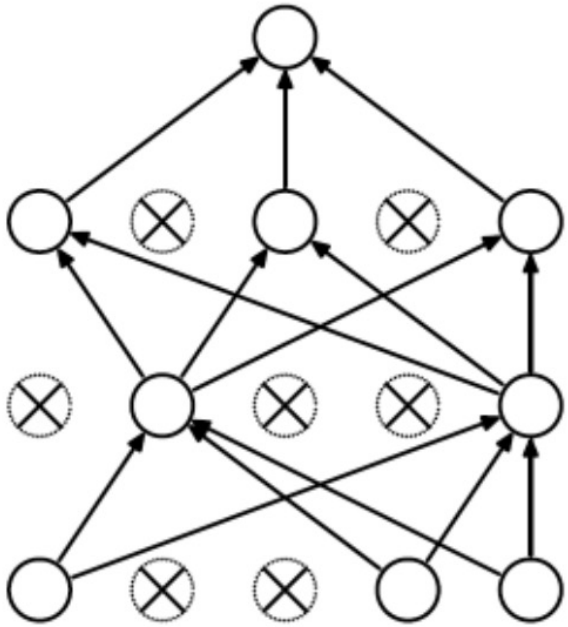
- A more straight forward way is to increase the dataset size
- We can modify the dataset and have additional examples
- This process is called dataset augmentation

Dropout

- Another way is to randomly drop some weights in the learning process.
- It also improves the generalization and reduce overfitting



(a) Standard Neural Net



(b) After applying dropout.

Batch normalization

- One of the classic problems with iterative optimization techniques is poor *conditioning*
- The gradient vary greatly in magnitude for each sample
- A technique to alleviate the problem is Batch Normalization
- It tries to normalize the mean and variance in each channel

$$\mu_i = \frac{1}{|\mathcal{B}|}\sum_{n\in\mathcal{B}} s_i^{(n)}$$

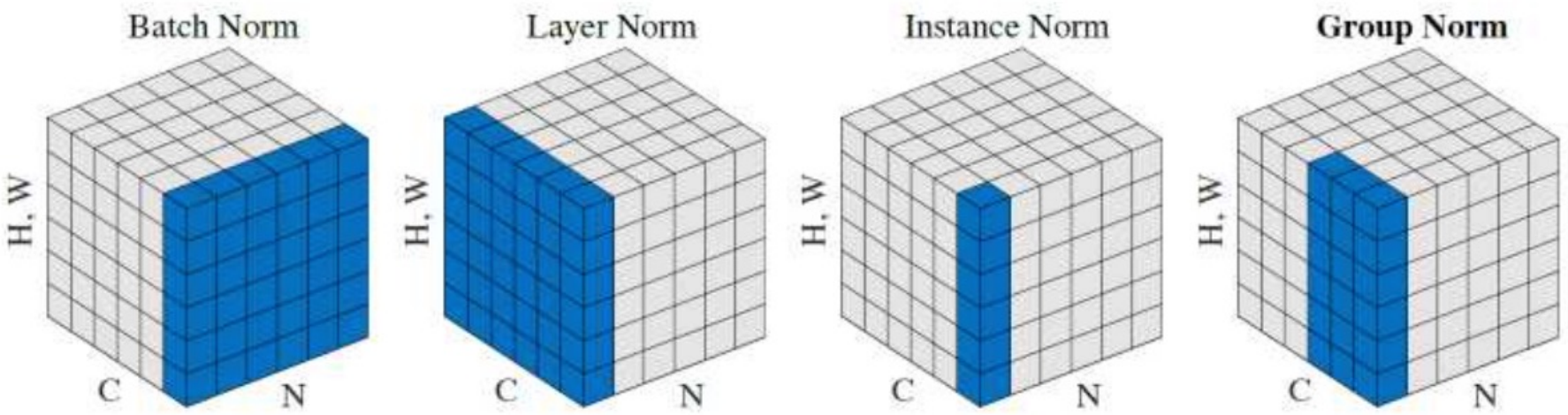
$$\sigma_i^2 = \frac{1}{|\mathcal{B}|}\sum_{n\in\mathcal{B}} (s_i^{(n)} - \mu_i)^2$$

$$\hat{s}_i^{(n)} = \frac{s_i^{(n)} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}},$$

$$y_i = \gamma_i \hat{s}_i + \beta_i.$$

Another normalization

- There are many other normalization methods
- The essential insight is to reduce the bias
- The pixels in blue are normalized by the same mean and variance.

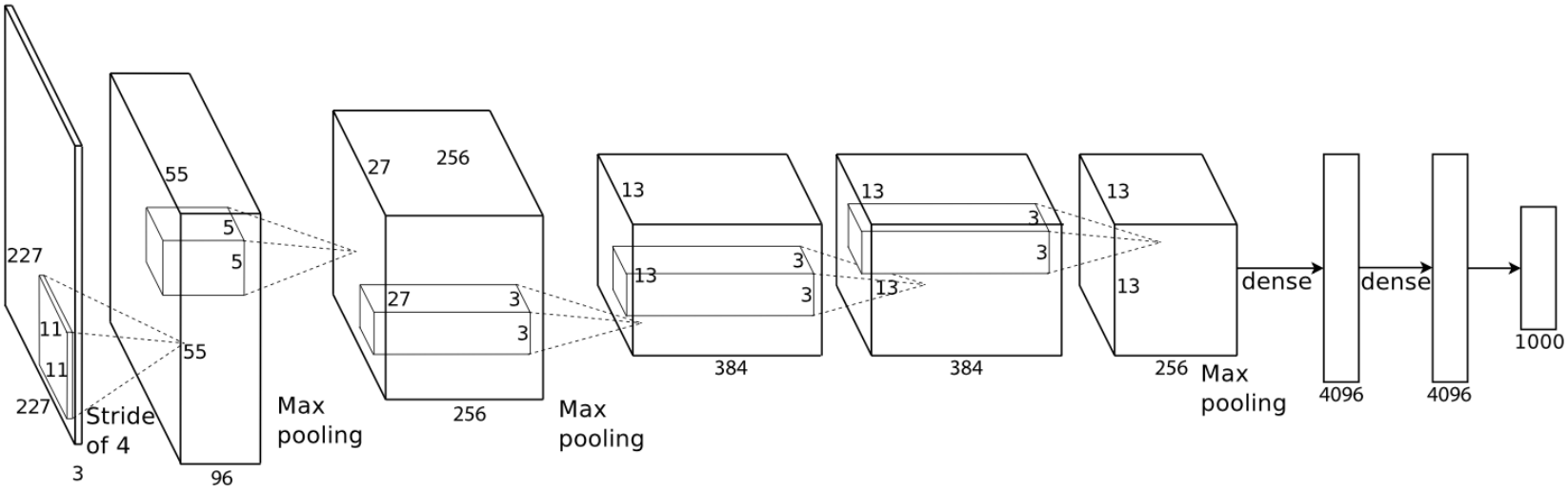


Training and optimization

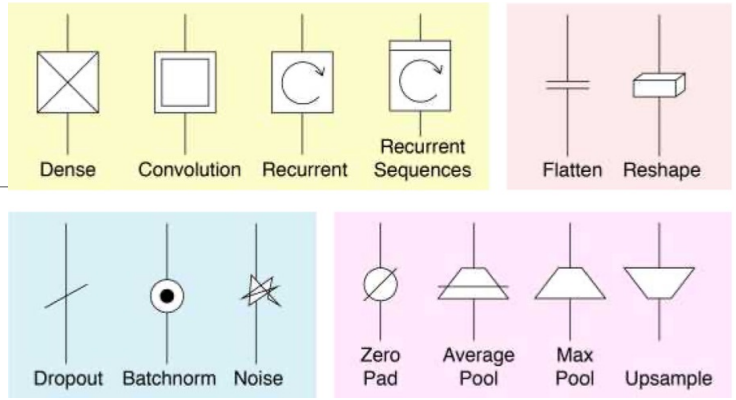
- A number of more sophisticated optimization techniques have been applied to deep network training
  - *Nesterov momentum*
  - *AdaGrad*
  - *RMSProp*
  - *Adam*
- Everything we discussed here has been implemented in modern ML framework, e.g. PyTorch, Jittor and so on.

Architecture

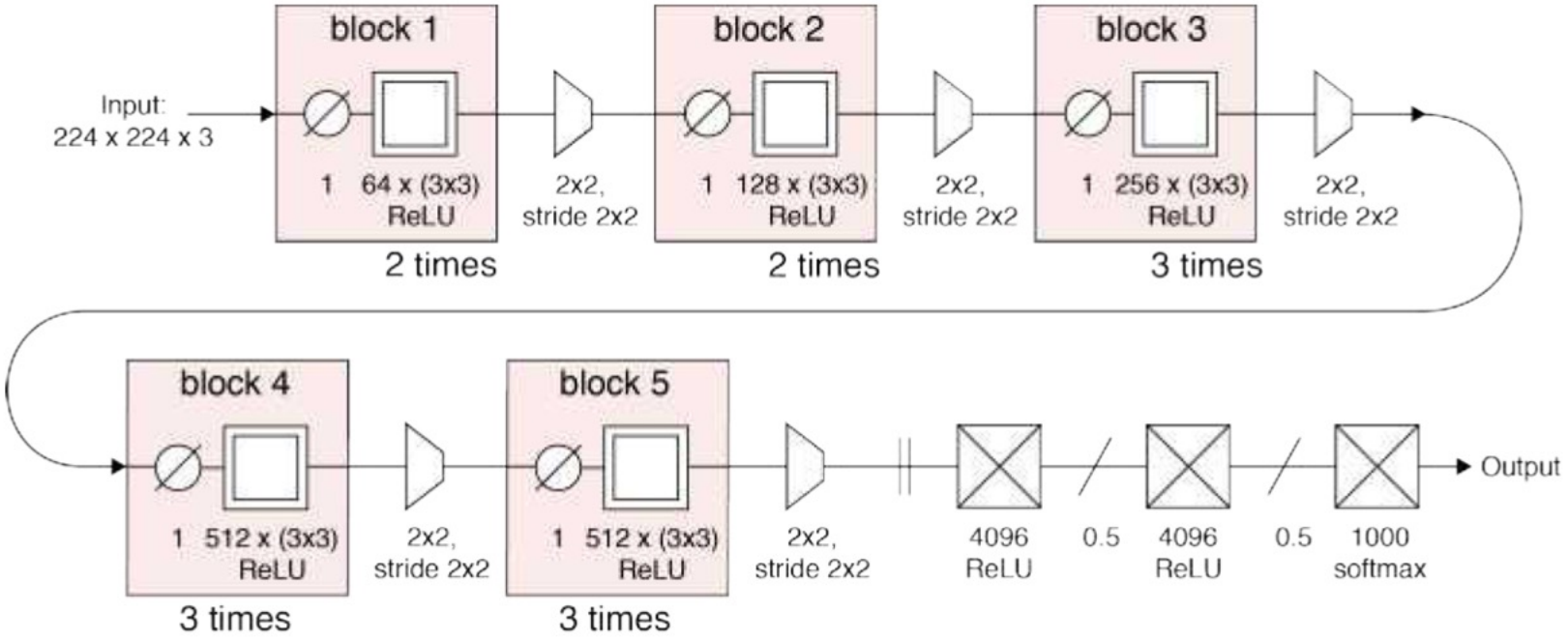
- The architecture of NN is very important for performance
- The first milestone architecture is *AlexNet*
- Basically, it uses conv. pooling and fully connected layers and a softmax layer



VGG16

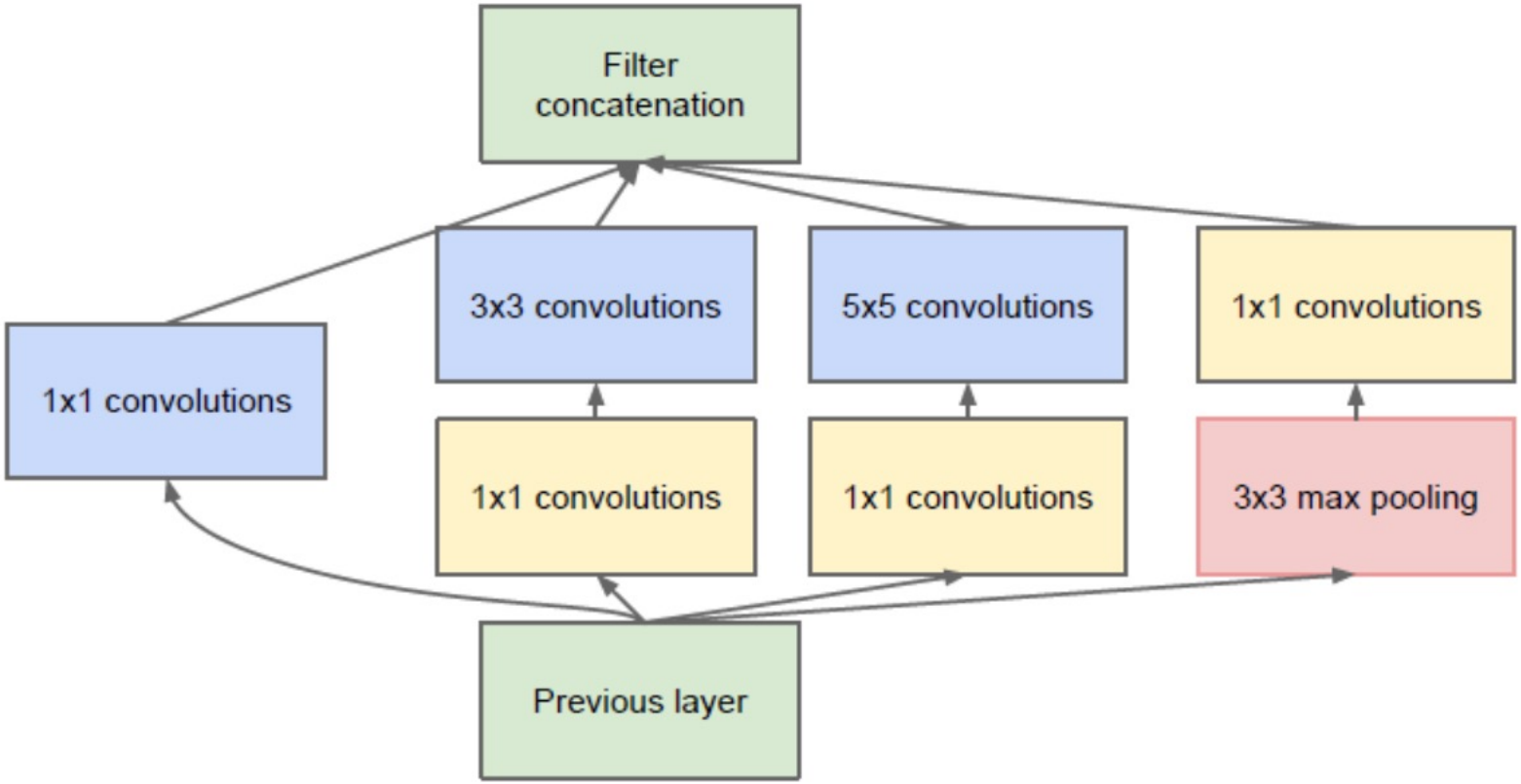


- People found that the performance increased as the NN goes deeper.



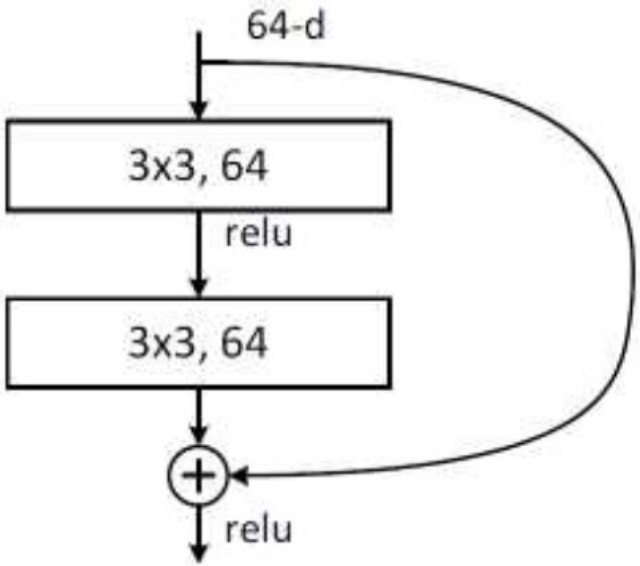
Inception modules

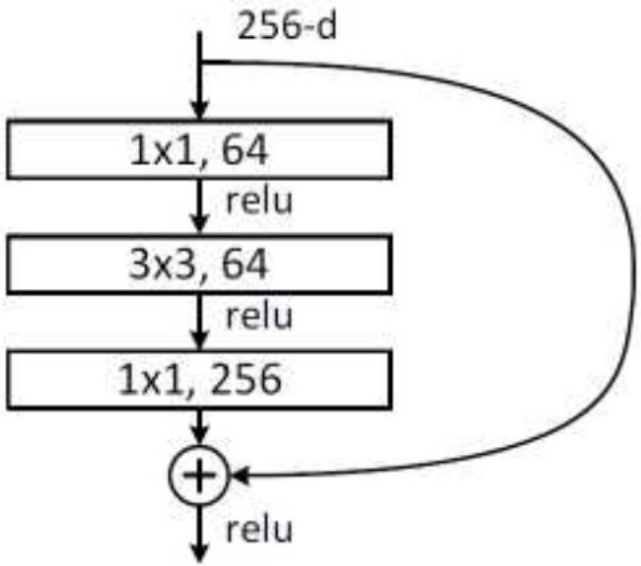
- GoogleNet is made up with inception modules



ResNet residual networks

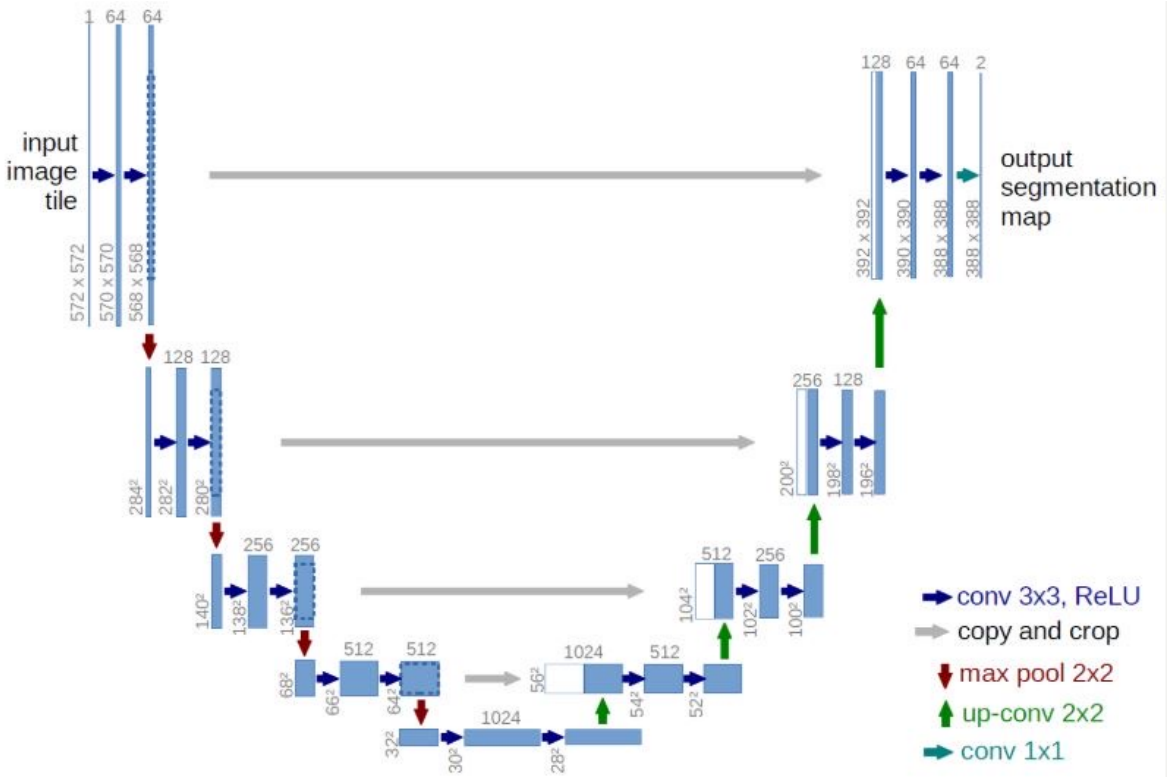
- The current most commonly used feature structure is ResnetXXX
- The idea is to pass gradient through the layers





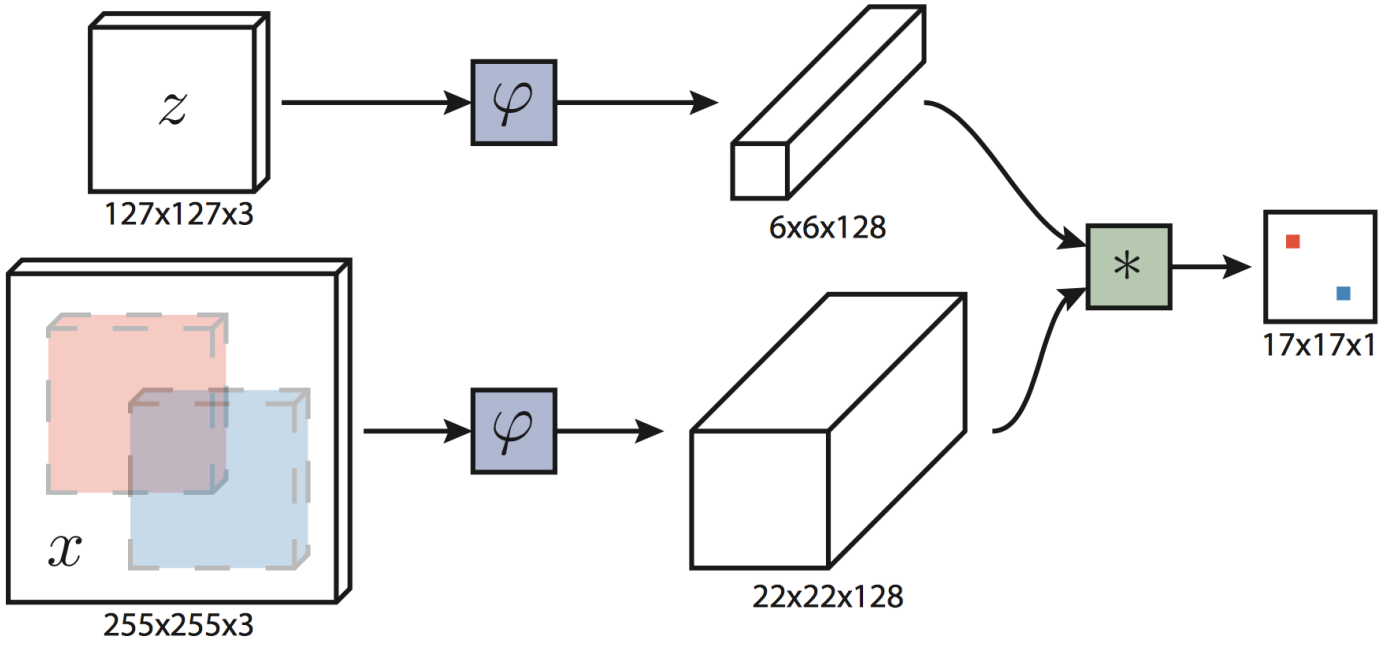
U-Net

- Another popular structure is U-Net
- This is very commonly used in segmentation task



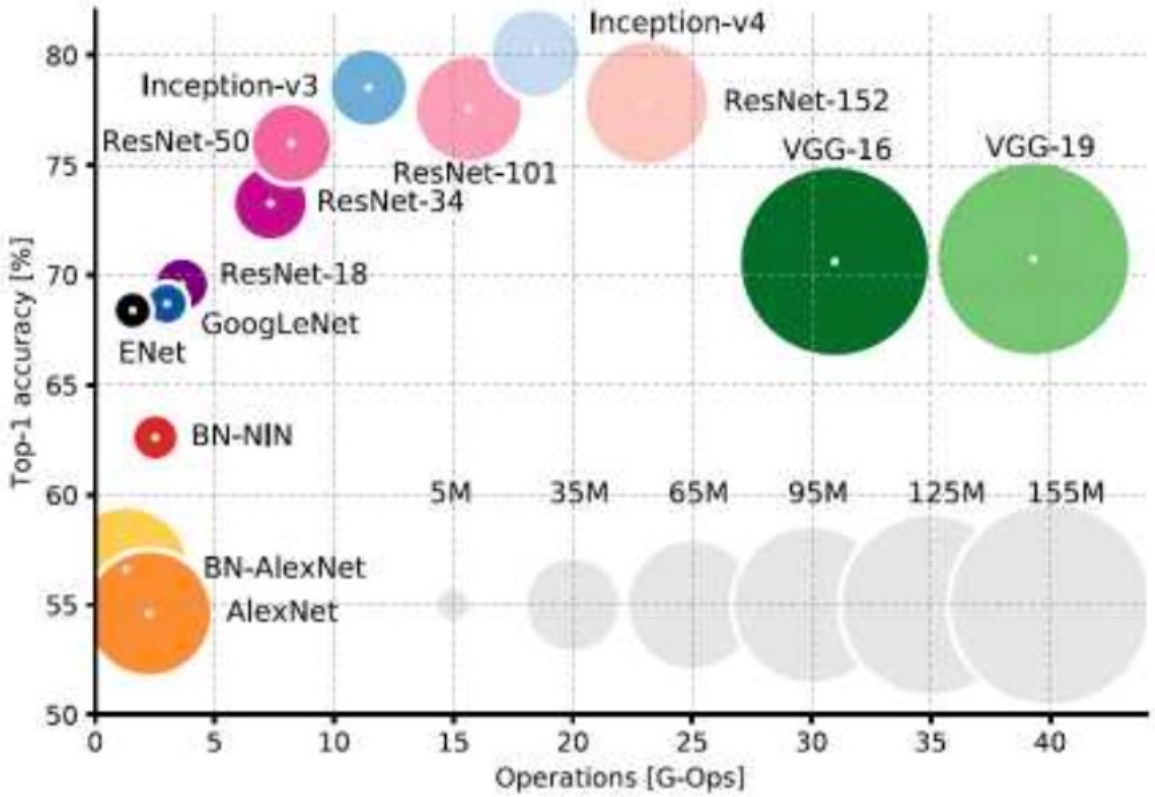
Siamese Networks

- We already met the siamese network in tracking lesson
- It is used for pair images



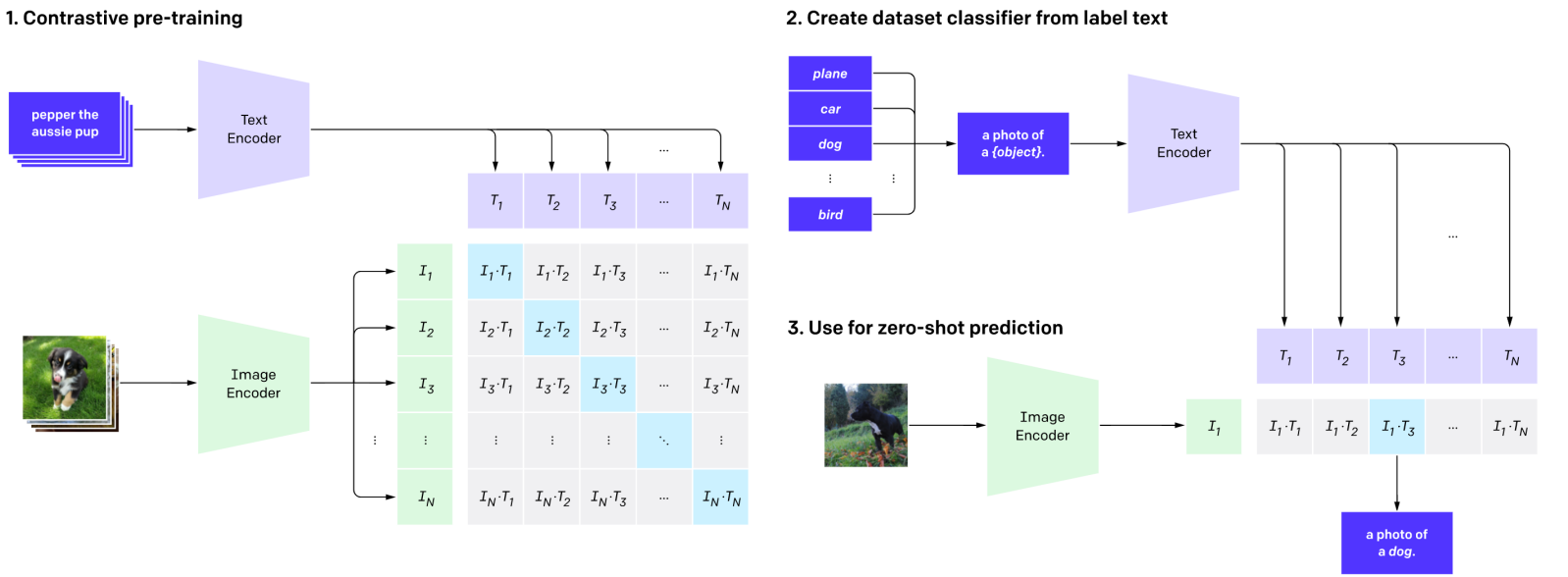
Famous Backbone

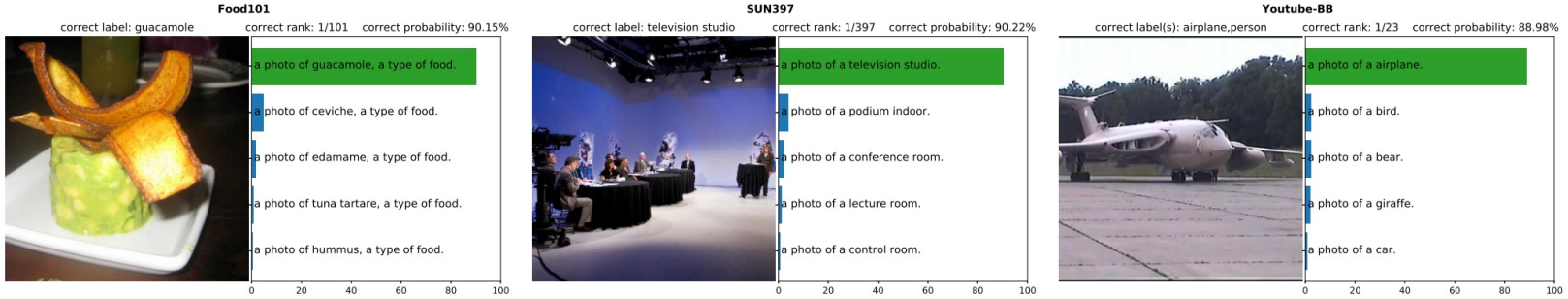
- Usually, people don’t train their network from beginning
- They will use backbone network with pre-trained weights



Visual Transformer

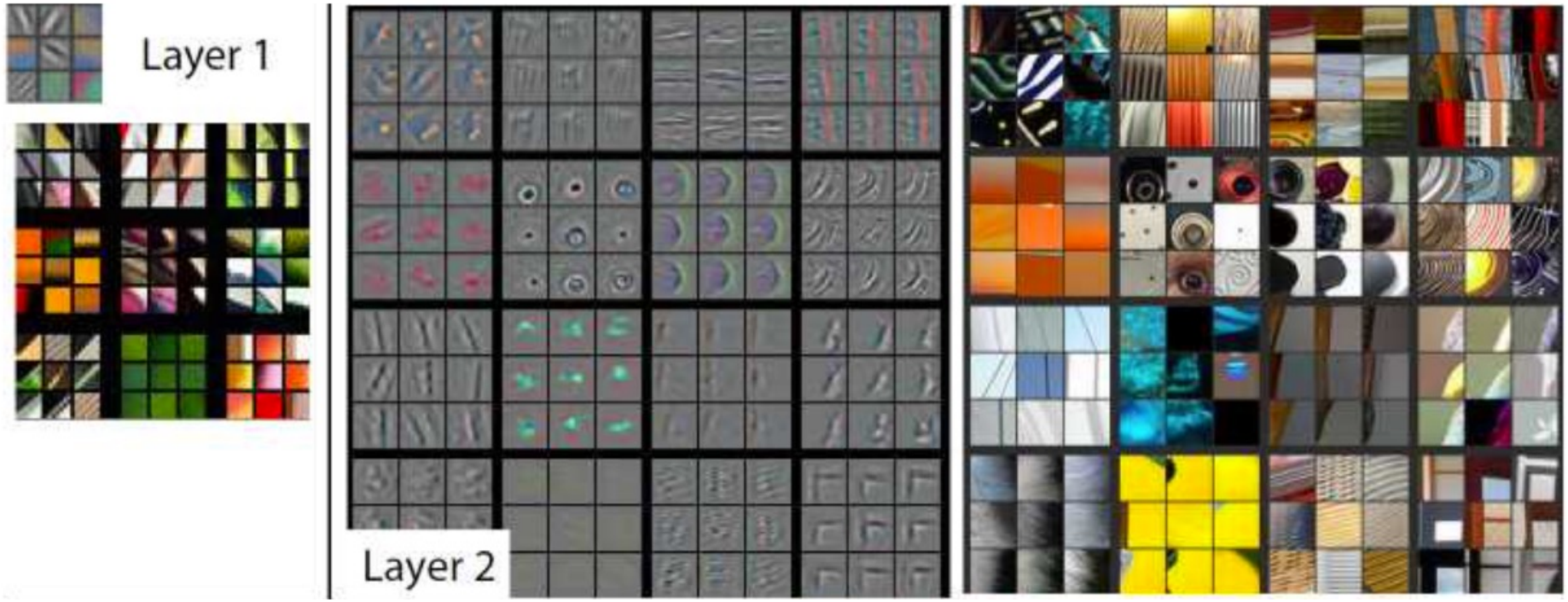
Contrastive Language-Image Pre-training (CLIP)



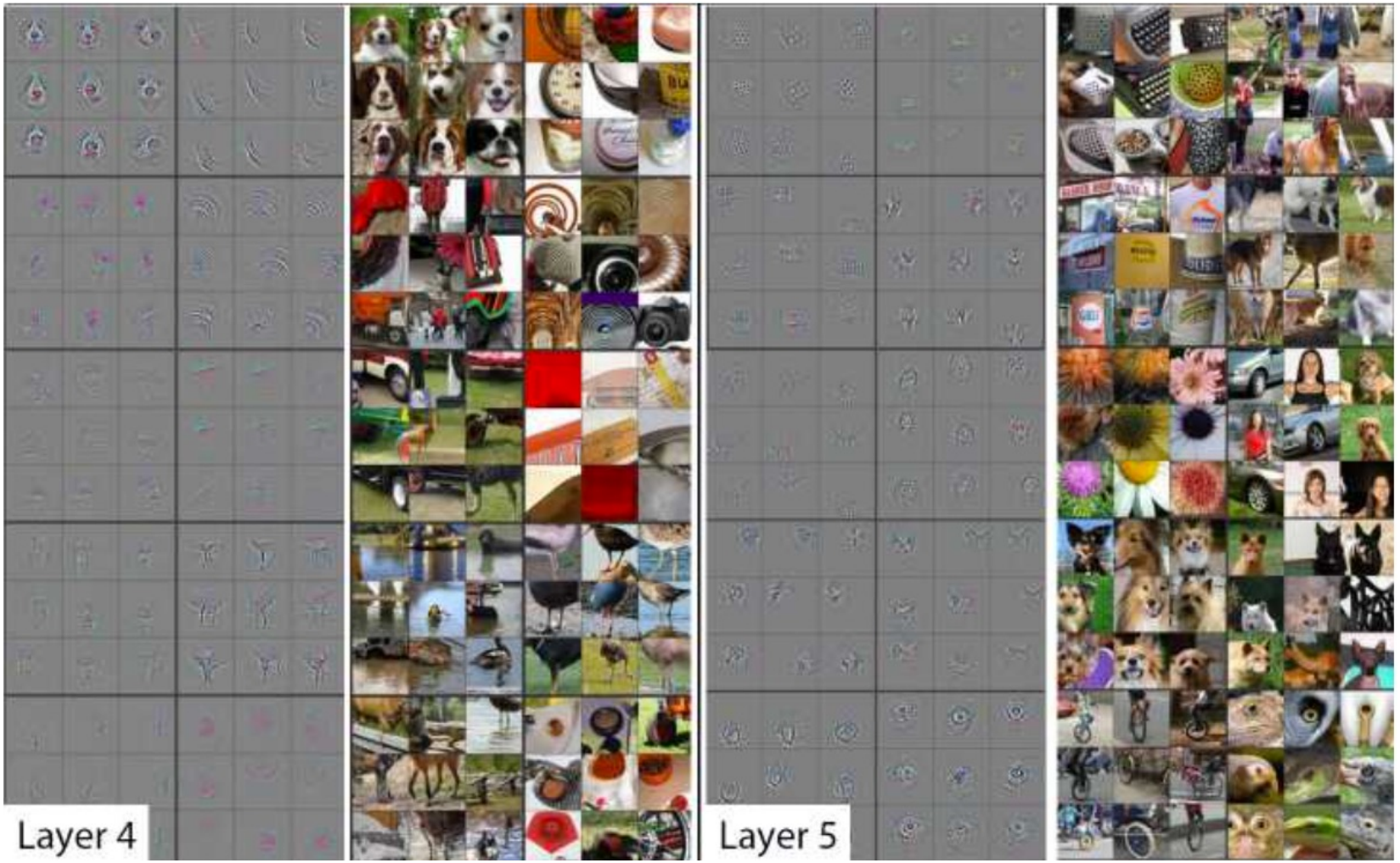


visualization

- The visualization of the weights give some hints about what on the earth is NN doing.



visualization



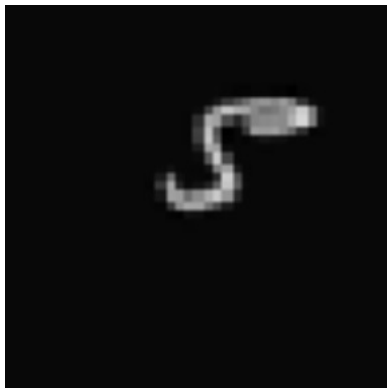
Thinking

- everything has been implemented in deep learning framework tools
- But, deep learning is still an open problem in academic
- why?

- Once we get familiar with basic concepts, the deep learning topic can be anything you can imagine
  - the input and output
  - the structure
  - how to train
  - any insight loss
  - how to use it in practice

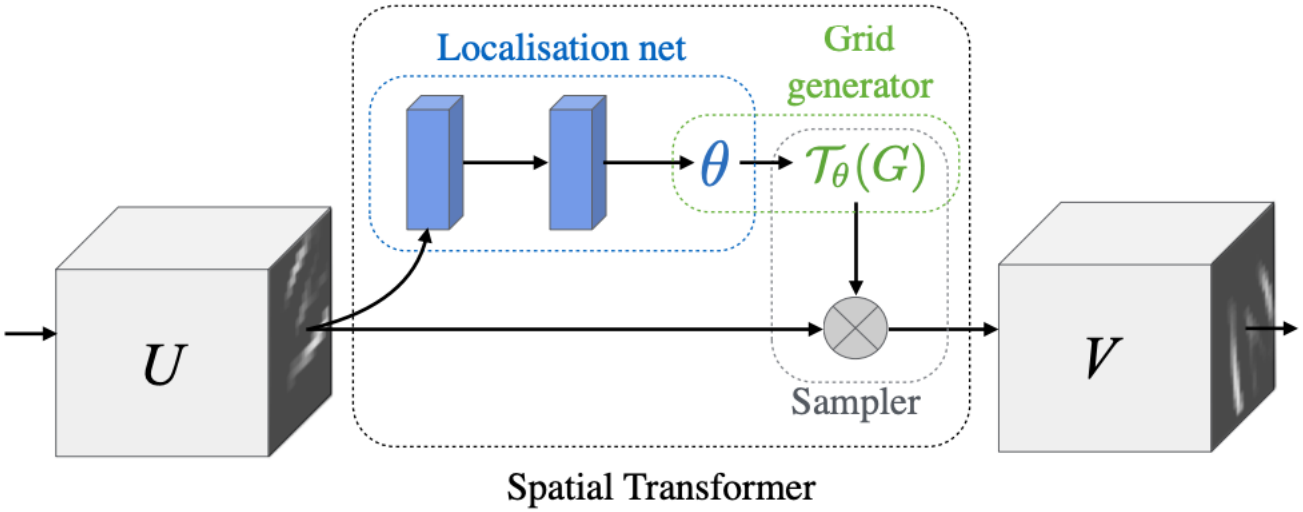
Geometrics with deep learning

- Alright, we come to the topic of geometrics with deep learning.
- But, how?
- Suppose we have a hand-write digit, we know it has some geometric deformation.



Spatial Transformation Networks

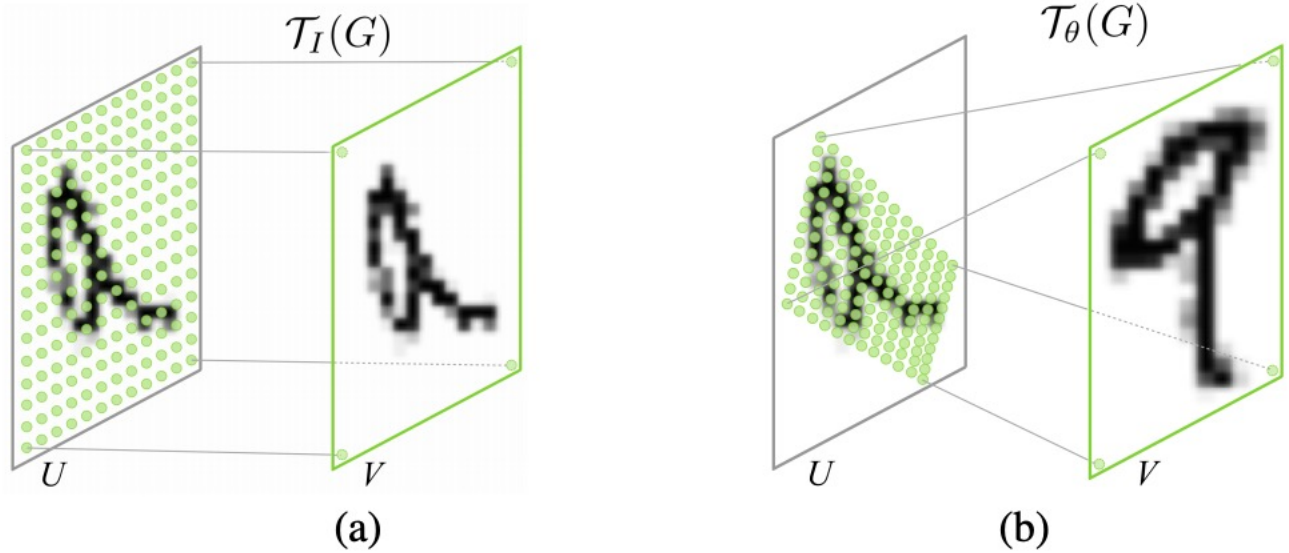
- A milestone is *spatial transformation networks*
- It links traditional geometrics to deep learning tech.
- The idea is very simple: *let the networks learn it by itself*



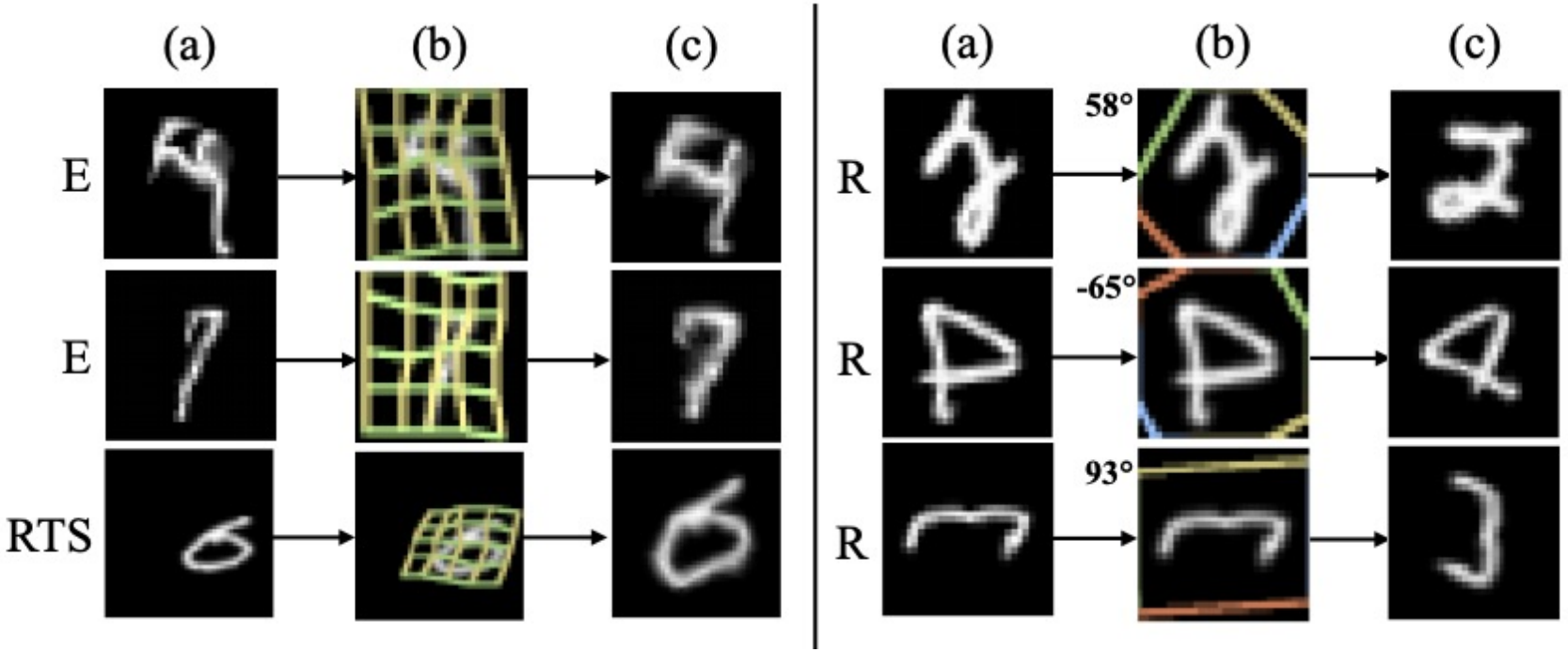
Spatial Transformation Networks

- The core of STN is how to sample/warp the image

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta(G_i) = \mathtt{A}_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

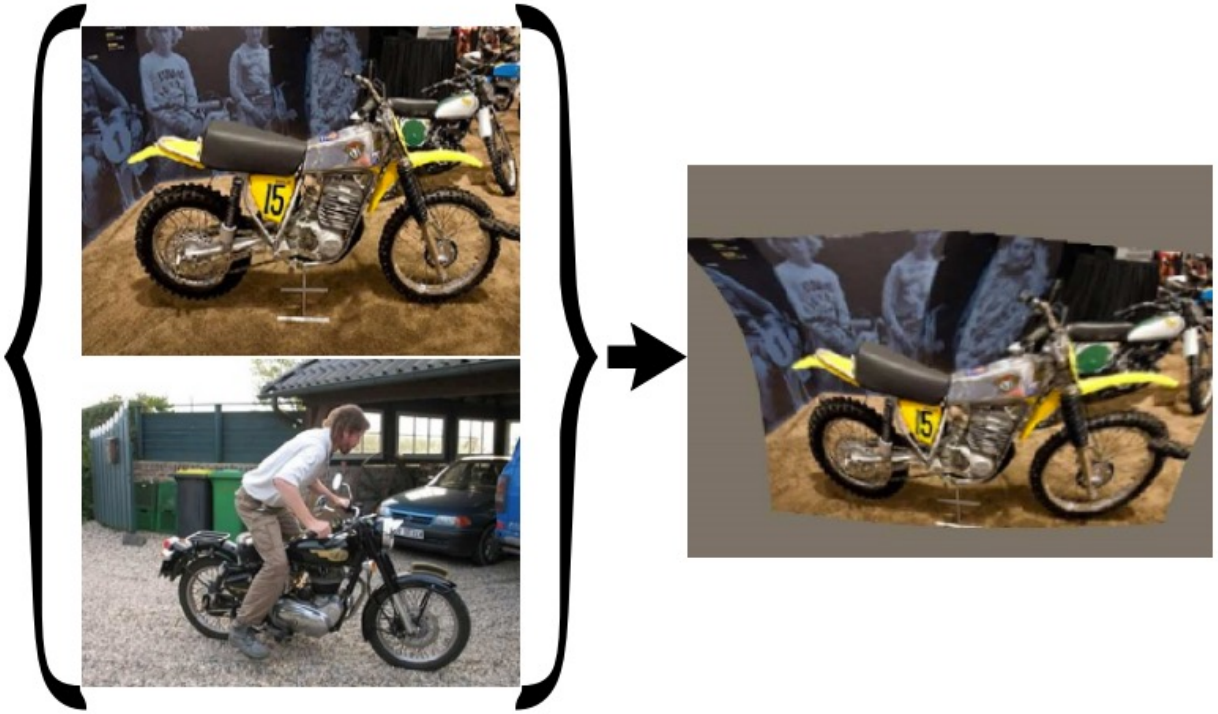


results



Other usage

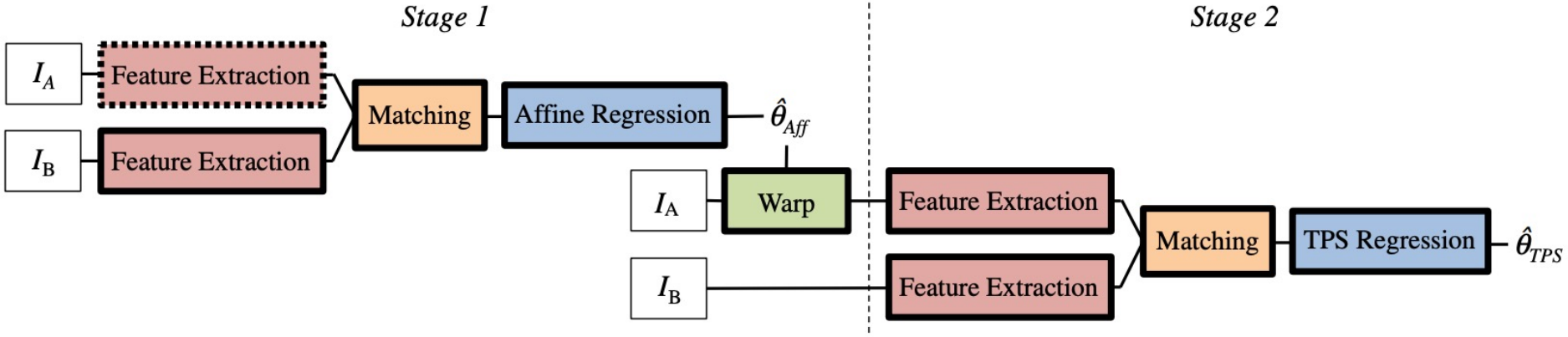
- The reason why STN matters is that the idea can be applied into any geometric related topics



Convolutional neural network architecture for geometric matching. CVPR 2017

geometric matching

- With STN, we can make up our networks with geometric constraints!
- Please note, people add all ingredients together to construct their fancy work!



Results

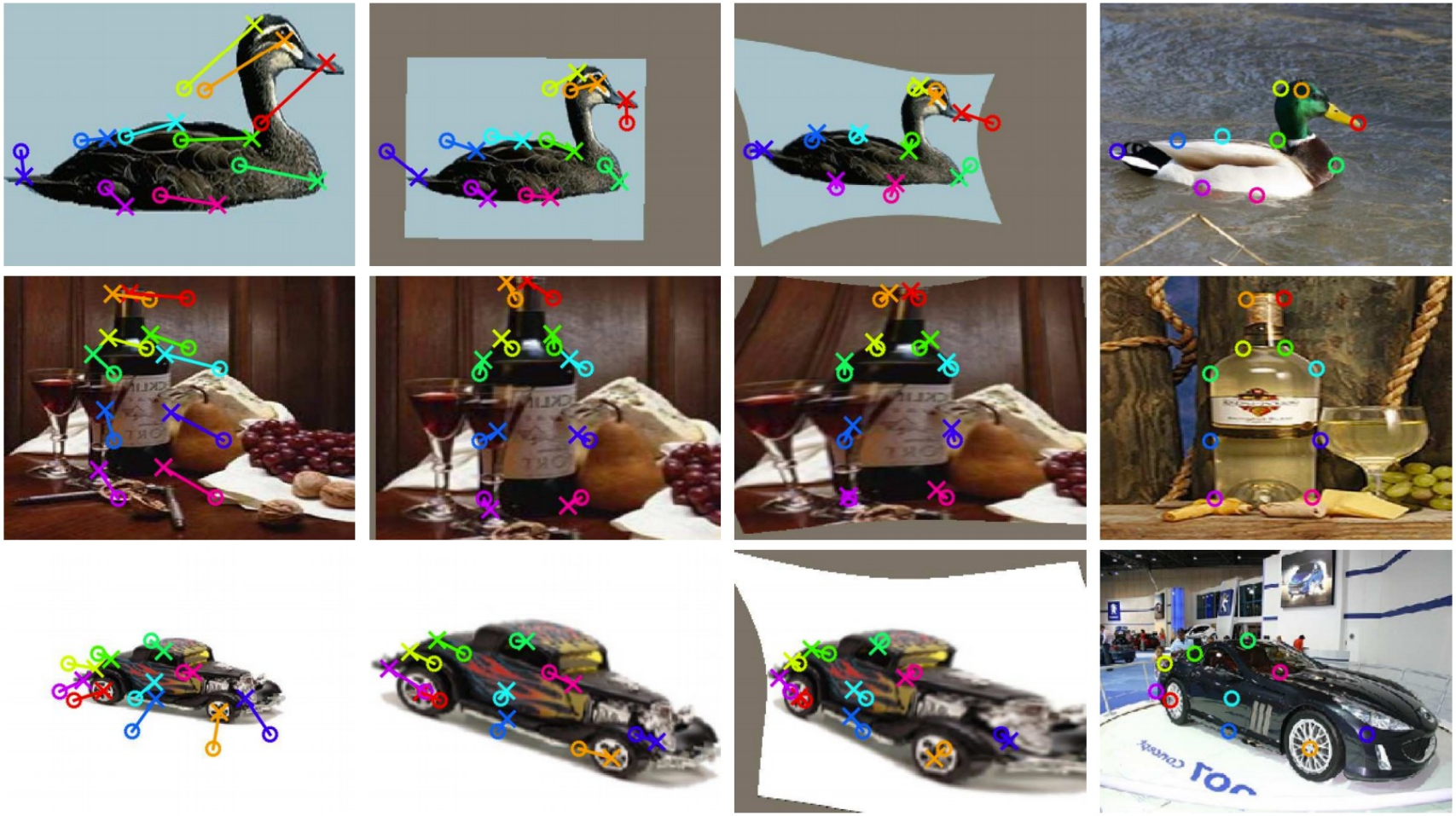


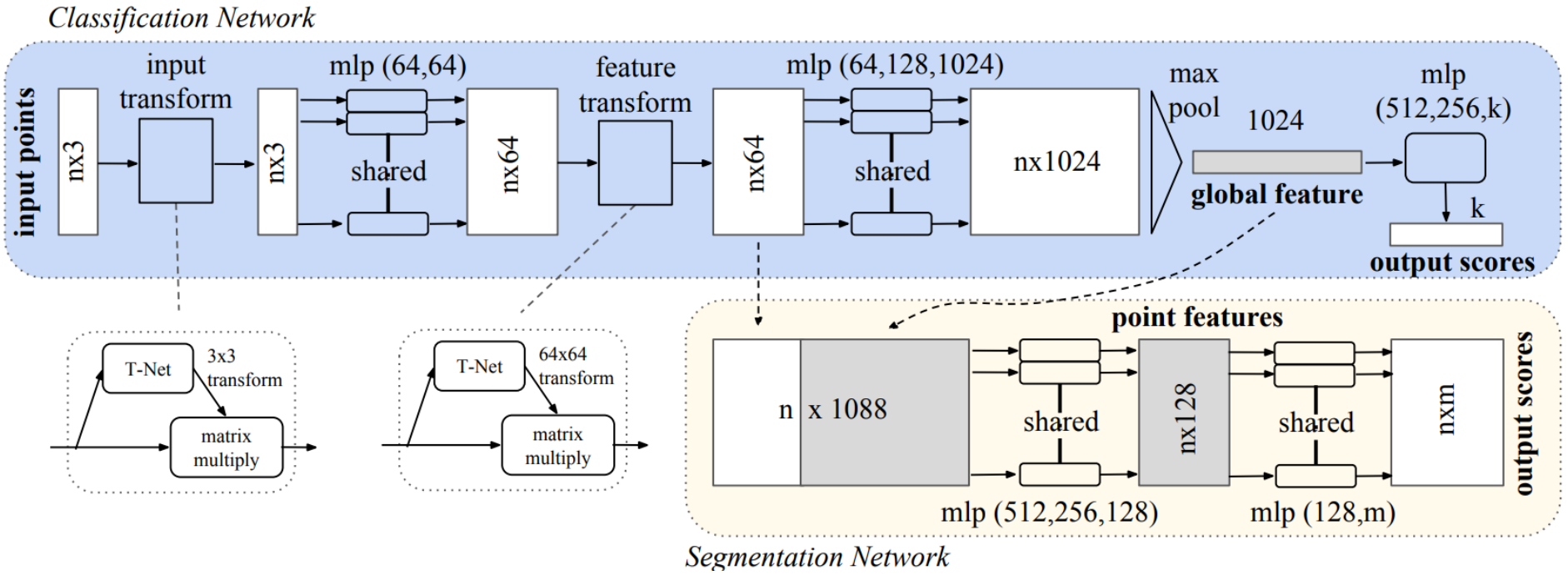
Image A | Aligned A (affine) | Aligned A (affine+TPS) | Image B

How about 3D?

- The idea of geometric constraint can be also applied into 3D space!
- Thinking of how to achieve it?
- Be careful about the input and output

PointNet

- Another milestone in 3D task is called PointNet
- Here is its structure



results

Deep Learning Framework

- There is a lot of frameworks we can use
  - PyTorch
  - TensorFlow
  - Mindspore
  - Jax
  - Taiji
- They all provide building block
  - auto-grad
  - data loader
  - basic block for NN
  - …

Example

```
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor
```

- We use pytorch as an example
- first we need prepare our dataset

```
# Download training data from open datasets.
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
)

# Download test data from open datasets.
test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor(),
)
```

Data loading

```
batch_size = 64

# Create data loaders.
train_dataloader = DataLoader(training_data, batch_size=batch_size)
test_dataloader = DataLoader(test_data, batch_size=batch_size)

for X, y in test_dataloader:
    print(f"Shape of X [N, C, H, W]: {X.shape}")
    print(f"Shape of y: {y.shape} {y.dtype}")
    break
```

Out:

```
Shape of X [N, C, H, W]: torch.Size([64, 1, 28, 28])
Shape of y: torch.Size([64]) torch.int64
```

Building Networks

```python
# Get cpu, gpu or mps device for training.
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
print(f"Using {device} device")
```

```
Using cuda device
```

```python
# Define model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork().to(device)
print(model)
```

```
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

Optimizing the parameters

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
```

```
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if batch % 100 == 0:
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

Testing the results

```
def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
```

```
epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(train_dataloader, model, loss_fn, optimizer)
    test(test_dataloader, model, loss_fn)
print("Done!")
```

Using your model

```
torch.save(model.state_dict(), "model.pth")
print("Saved PyTorch Model State to model.pth")
```

```
model = NeuralNetwork().to(device)
model.load_state_dict(torch.load("model.pth"))
```

```
model.eval()
x, y = test_data[0][0], test_data[0][1]
with torch.no_grad():
    x = x.to(device)
    pred = model(x)
    predicted, actual = classes[pred[0].argmax(0)], classes[y]
    print(f'Predicted: "{predicted}", Actual: "{actual}"')
```

Summary

- Deep Learning basic
- Traditional deep learning
- geometric method
- STN & PointNet

Reference

- https://en.wikipedia.org/wiki/Deep_learning
- https://gombru.github.io/2018/05/23/cross_entropy_loss/
- https://en.wikipedia.org/wiki/Cross_entropy#Cross-entropy_loss_function_and_logistic_regression
- https://en.wikipedia.org/wiki/Gradient_descent
- http://neuralnetworksanddeeplearning.com/chap4.html
- https://cg.cs.tsinghua.edu.cn/jittor/
- https://www.analyticsvidhya.com/blog/2019/07/heroes-of-machine-learning-experts-researchers/
- https://awards.acm.org/about/2018-turing
- https://theaisummer.com/vision-transformer/
- https://openai.com/research/clip