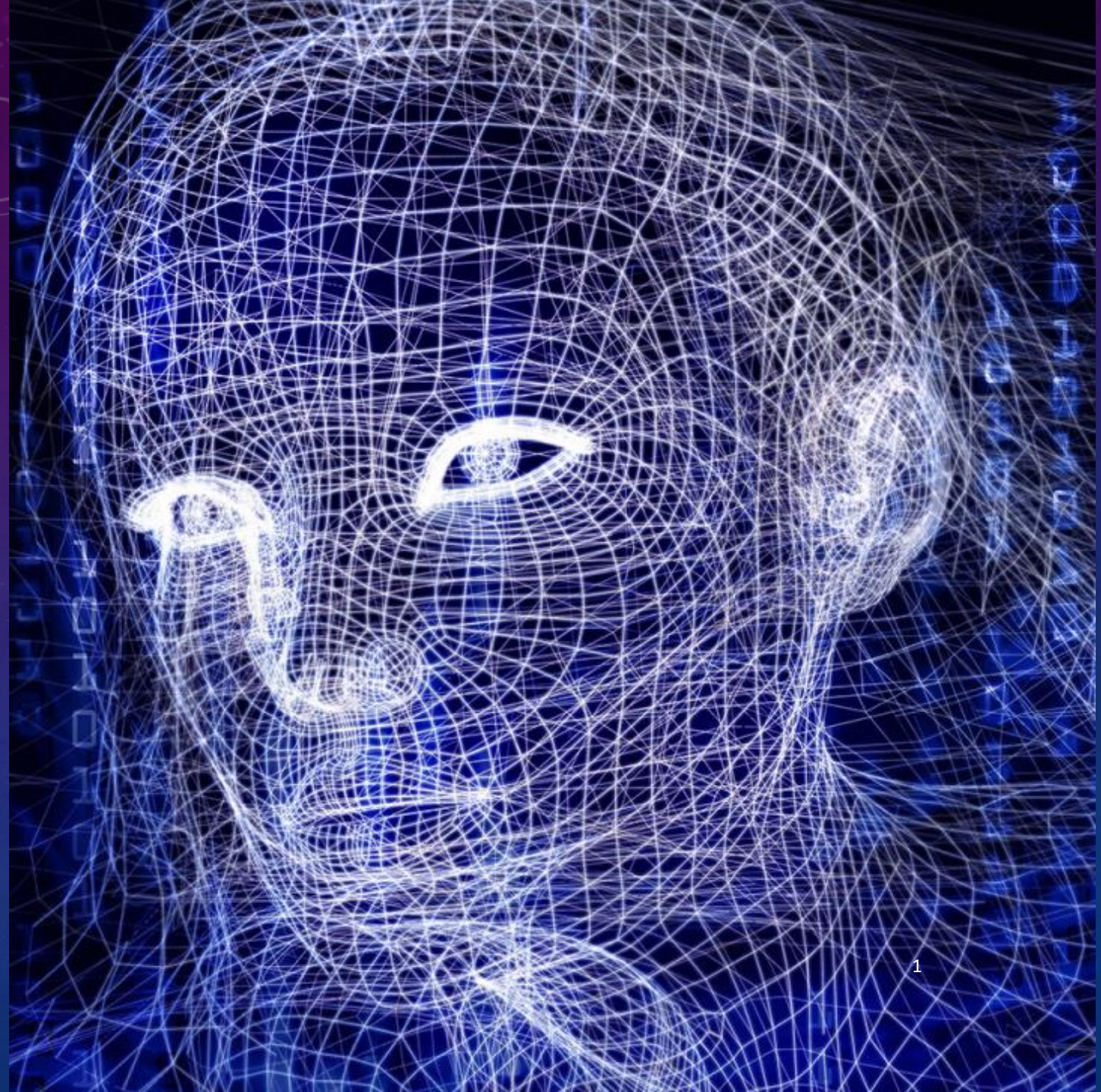
LECTURE SERIES FOR DIGITAL SURVEILLANCE SYSTEMS AND APPLICATION

Architecture of Deep Neural Networks

徐繼聖

Gee-Sern Jison Hsu

National Taiwan University of Science and Technology



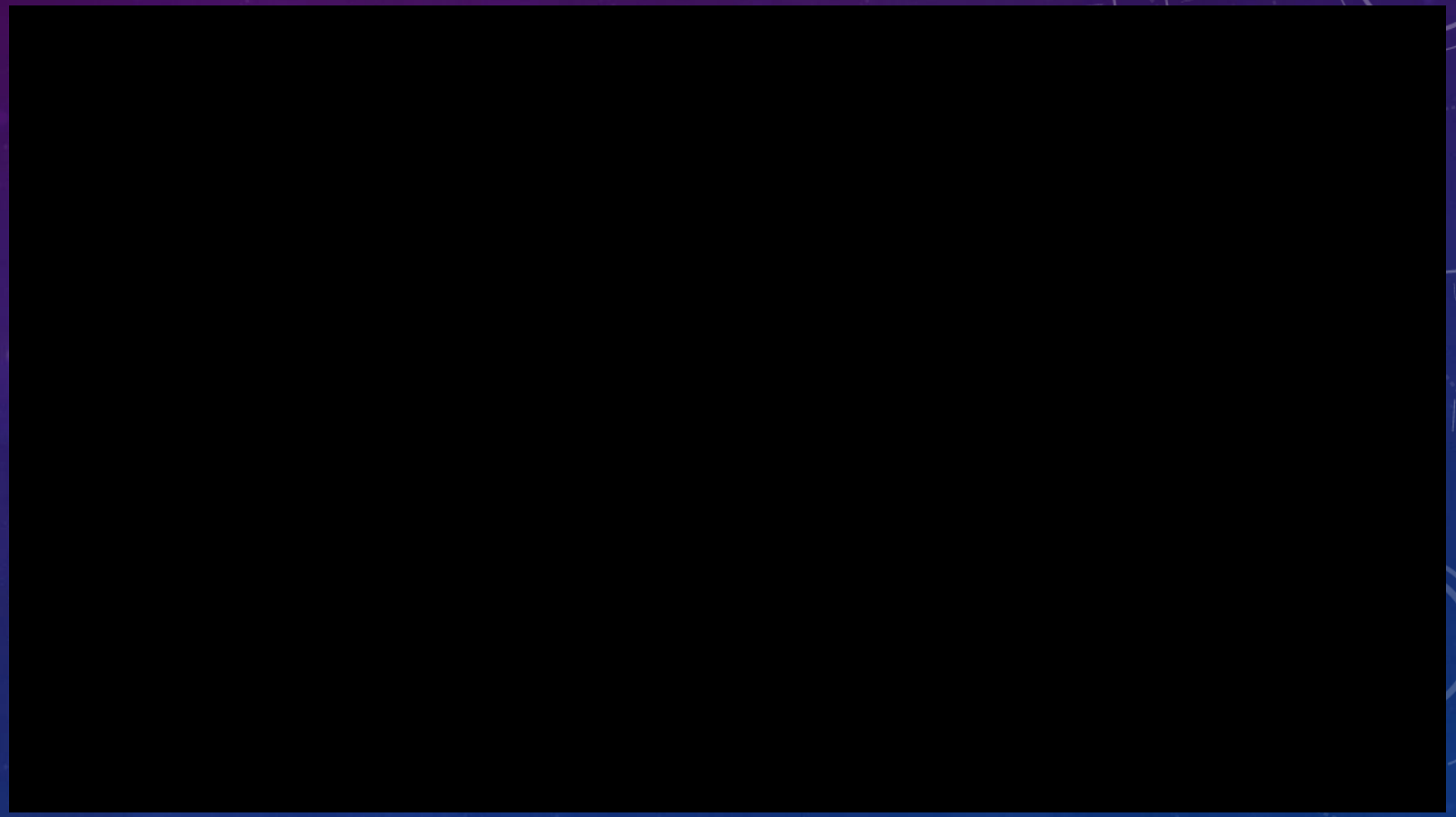
Convolutional Neural Networks

1) Activation Functions
2) Backpropagation
3) Dropout
4) Preprocessing Data
5) Weight Initialization
6) Batch Normalization
7) Babysitting the Learning Process
8) Hyperparameter Optimization
9) Architecture Introduction

Stanford lecture

Content:

2:00 CNN 4:50Activation Functions
25:19 Maxout
27:22 Preprocessing data
33:35 Weight Initialization
48:56 Batch Normalization
1:04:42 Babysitting the learning Process
1:10:22 Hyperparameter Optimization



https://www.youtube.com/watch?v=wEoyxE0GP2M&t=4s

Backpropagation – What is it really doing? – by BlueBrown

https://www.youtube.com/watch?v=Ilg3gGewQ5U&t=1s [13:53]

Backpropagation – Chain Rule – by BlueBrown

https://www.youtube.com/watch?v=tIeHLnjs5U8 [10:00]

Backpropagation

DEFINITION

1) **Calculate the forward phase** for each input-output pair $(\vec{x_d}, y_d)$ and store the results $\hat{y}_d$, a_j^k, and o_j^k for each node j in layer k by proceeding from layer 0, the input layer, to layer m, the output layer.

2) **Calculate the backward phase** for each input-output pair $(\vec{x_d}, y_d)$ and store the results $\frac{\partial E_d}{\partial w_{ij}^k}$ for each weight w_{ij}^k connecting node i in layer $k-1$ to node j in layer k by proceeding from layer m, the output layer, to layer 1, the input layer.

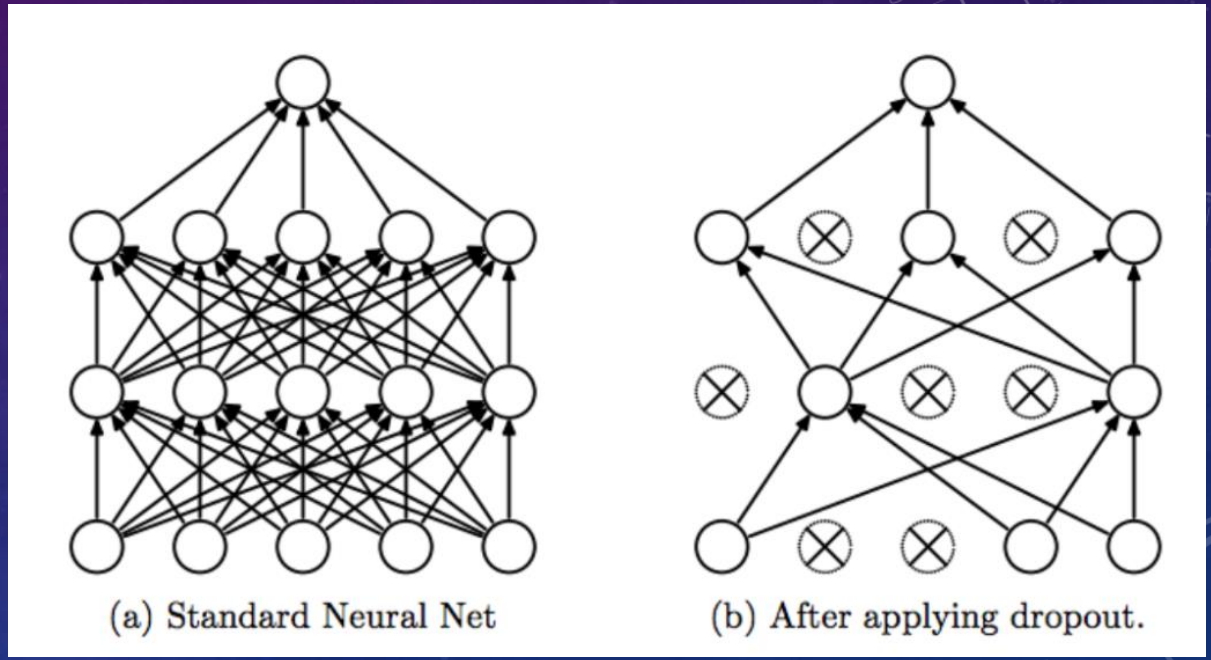
a) Evaluate the error term for the final layer δ_1^m by using the second equation.
b) Backpropagate the error terms for the hidden layers δ_j^k, working backwards from the final hidden layer $k = m - 1$, by repeatedly using the third equation.
c) Evaluate the partial derivatives of the individual error E_d with respect to w_{ij}^k by using the first equation.

3) **Combine the individual gradients** for each input-output pair $\frac{\partial E_d}{\partial w_{ij}^k}$ to get the total gradient $\frac{\partial E(X,\theta)}{\partial w_{ij}^k}$ for the entire set of input-output pairs $X = \{(\vec{x_1}, y_1), \ldots, (\vec{x_N}, y_N)\}$ by using the fourth equation (a simple average of the individual gradients).

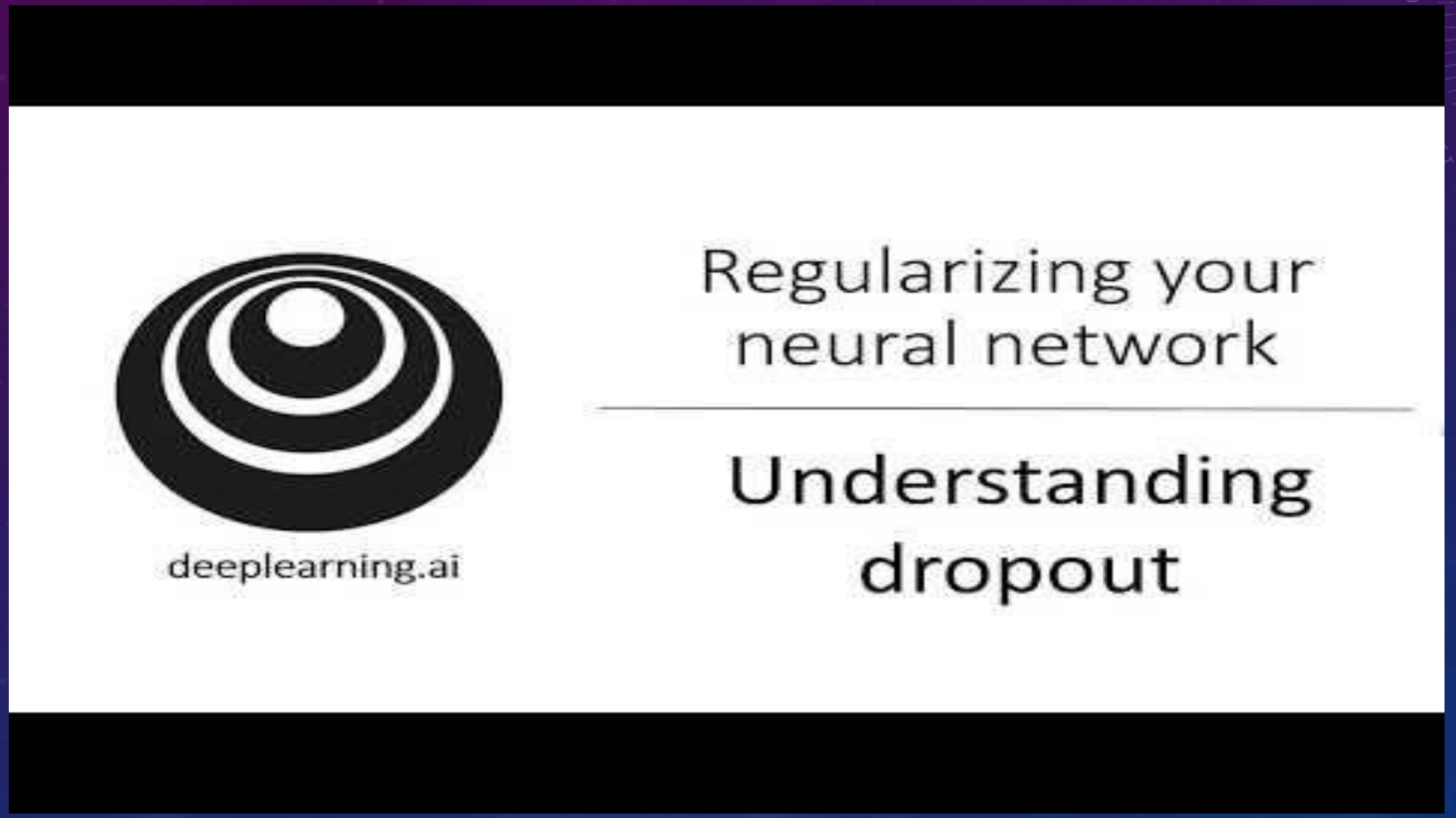
4) **Update the weights** according to the learning rate α and total gradient $\frac{\partial E(X,\theta)}{\partial w_{ij}^k}$ by using the fifth equation (moving in the direction of the negative gradient).

Dropout

- Dropout is a term which is used deep learning and is a common technique used for training and enhancing the performance of Networks. We simply drop out units during training and avoid becoming too dependent on the feedback of single neurons.
- Regularization method



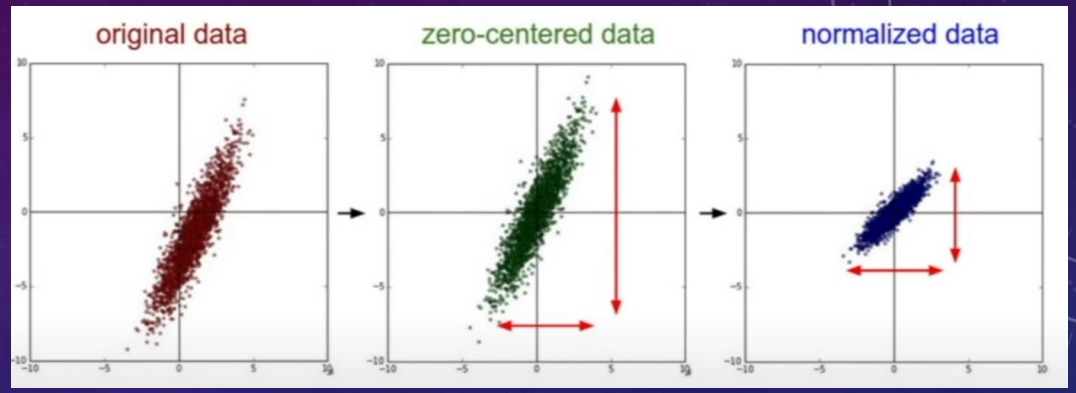
Dropout explained by Ng



https://www.youtube.com/watch?v=ARq74QuavAo&t=142s

Other Regularization methods are Preprocessing

- Consider the original data in red on the right as our input to any layer of our neural network. To make our network more robust to any kind of input data, we want to generalize this input data by first zero-centering and then normalizing the data.
- **Zero-centering:** subtract the mean
- **Normalizing:** divide by standard deviation



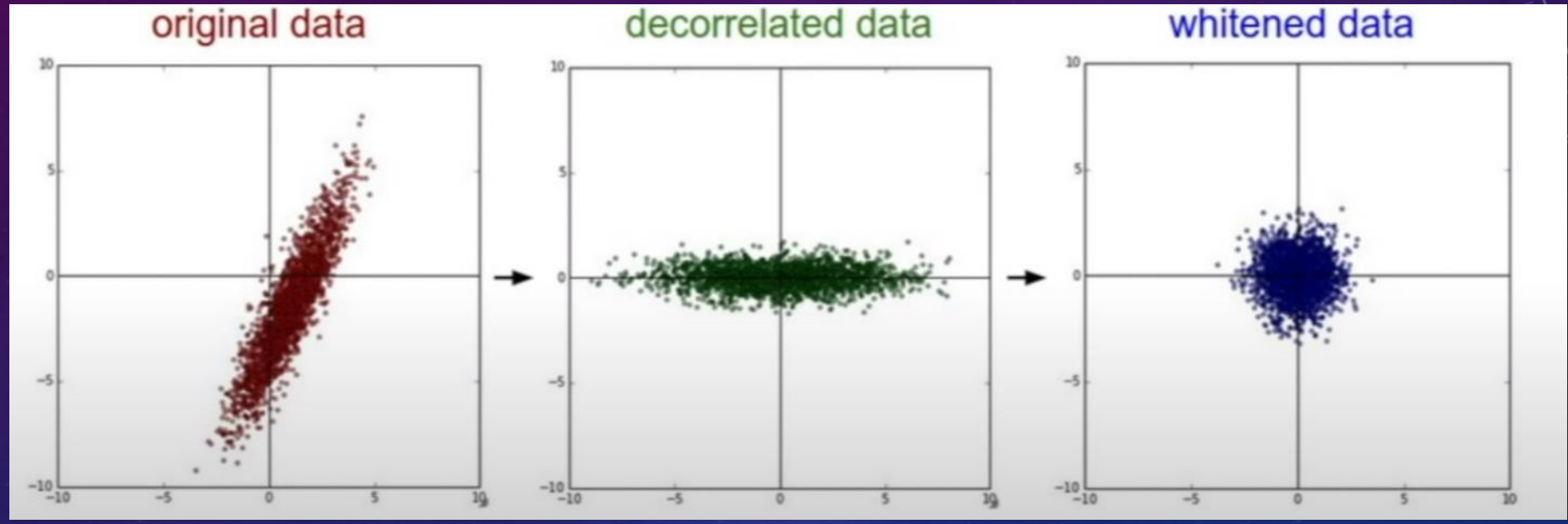

```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

Data Preprocessing for ML

https://www.youtube.com/watch?v=NBm4etNMT5k [3:30]

Other forms of regularization might be PCA and Whitening



Data has diagonal covariance matrix

Covariance matrix is identity matrix

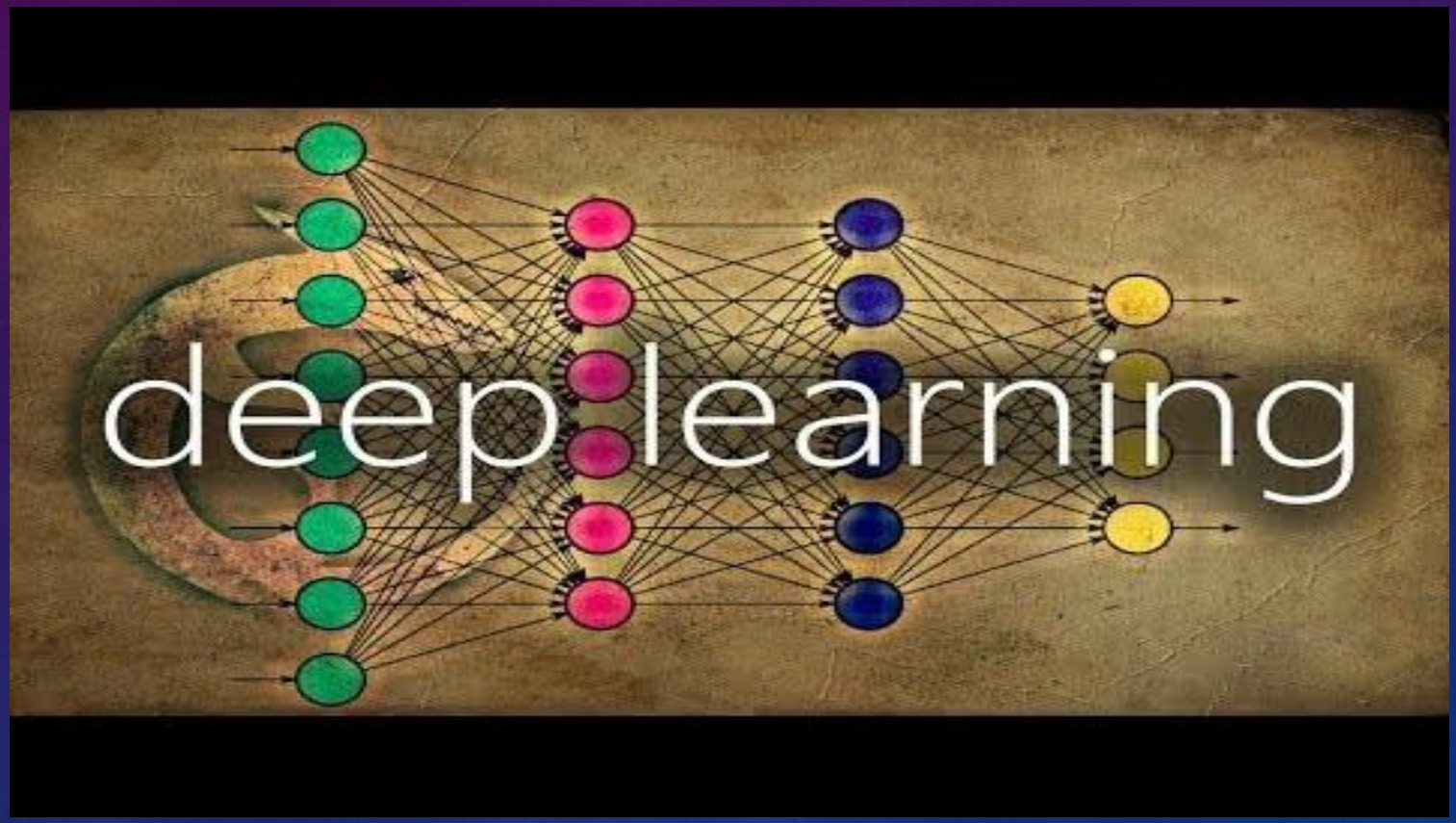
What is a covariance Matrix?

- *In probability theory and statistics, a **covariance matrix** (also known as **auto-covariance matrix**, **dispersion matrix**, **variance matrix**, or **variance–covariance matrix**) is a square matrix giving the covariance between each pair of elements of a given random vector. In the matrix diagonal there are variances, i.e., the covariance of each element with itself.*

$$\mathrm{K_{XX}} = \begin{bmatrix} \mathrm{E}[(X_1 - \mathrm{E}[X_1])(X_1 - \mathrm{E}[X_1])] & \mathrm{E}[(X_1 - \mathrm{E}[X_1])(X_2 - \mathrm{E}[X_2])] & \cdots & \mathrm{E}[(X_1 - \mathrm{E}[X_1])(X_n - \mathrm{E}[X_n])] \\ \mathrm{E}[(X_2 - \mathrm{E}[X_2])(X_1 - \mathrm{E}[X_1])] & \mathrm{E}[(X_2 - \mathrm{E}[X_2])(X_2 - \mathrm{E}[X_2])] & \cdots & \mathrm{E}[(X_2 - \mathrm{E}[X_2])(X_n - \mathrm{E}[X_n])] \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{E}[(X_n - \mathrm{E}[X_n])(X_1 - \mathrm{E}[X_1])] & \mathrm{E}[(X_n - \mathrm{E}[X_n])(X_2 - \mathrm{E}[X_2])] & \cdots & \mathrm{E}[(X_n - \mathrm{E}[X_n])(X_n - \mathrm{E}[X_n])] \end{bmatrix}$$

$$\mathrm{K_{XX}} = \mathrm{cov}[\mathbf{X}, \mathbf{X}] = \mathrm{E}[(\mathbf{X} - \mu_{\mathbf{X}})(\mathbf{X} - \mu_{\mathbf{X}})^{\mathrm{T}}] = \mathrm{E}[\mathbf{X}\mathbf{X}^T] - \mu_{\mathbf{X}}\mu_{\mathbf{X}}{}^T \quad \text{(Eq.1)}$$

Weight Initialization

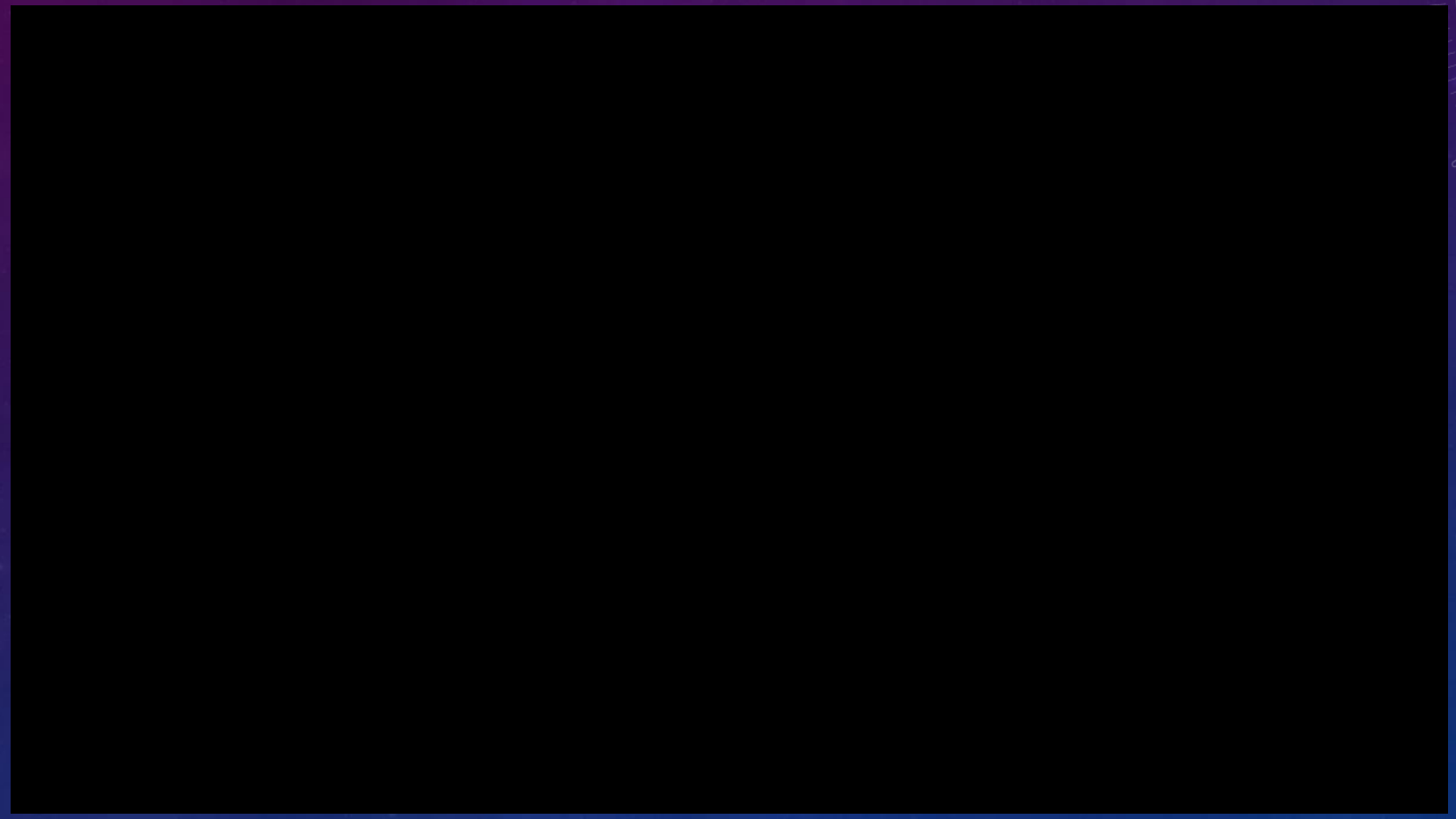


https://www.youtube.com/watch?v=8krd5qKVw-Q [10:00]

Weight Initialization – But Why?

- Prevent layer activation outputs from vanishing (0) or exploding (∞) during forward pass
- This is to avoid the gradients become 0 or too large in order for the backpropagation algorithm to work

Batch Normalization explained



https://www.youtube.com/watch?v=DtEq44FTPM4 [8:48]

Batch Normalization Mathematically

- Batch Normalization manipulates the layer inputs by calculating a batch's mean and variance. The data is then scaled and shifted
- Therefore, Batch Normalization is a special kind of preprocessing. The mathematical procedure can be seen on the right.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

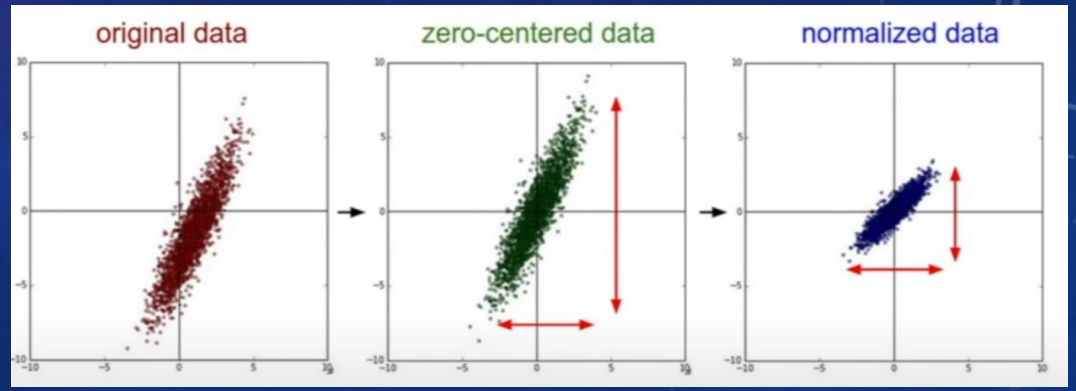
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

The simple equation behind batch normalization



Advantages of Batch Normalization

- Enables higher learning rates
- Accelerates the learning process
- Training Neural Nets with Sigmoid activations
- Bridged the other normalization methods such as Layer Normalization and weight normalization

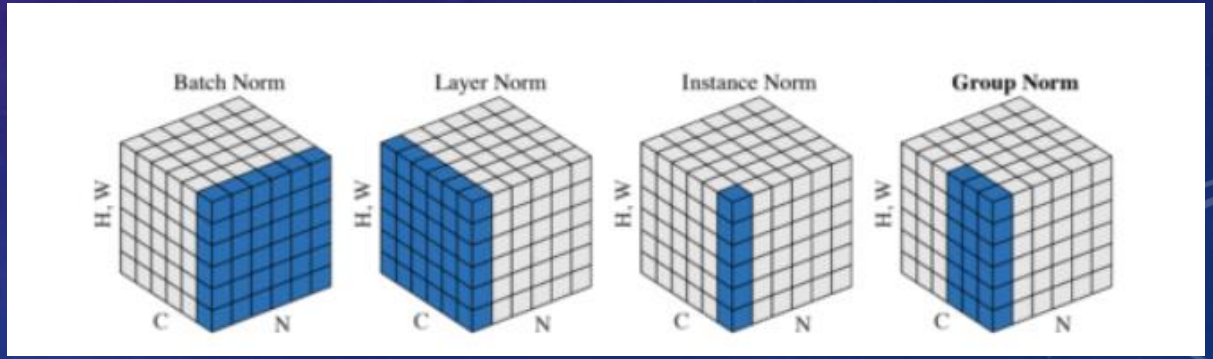
Layer Normalization

Mean of a layer

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$

variance

H=# of hidden units in one layer. Details in the paper https://arxiv.org/pdf/1607.06450.pdf



N is the batch axis, C is the Channel axis, (H,W) represent the spatial axis. The blue color indicates a normalization by same mean and variance, computed by the aggregation of the colored blocks

Babysitting the learning process

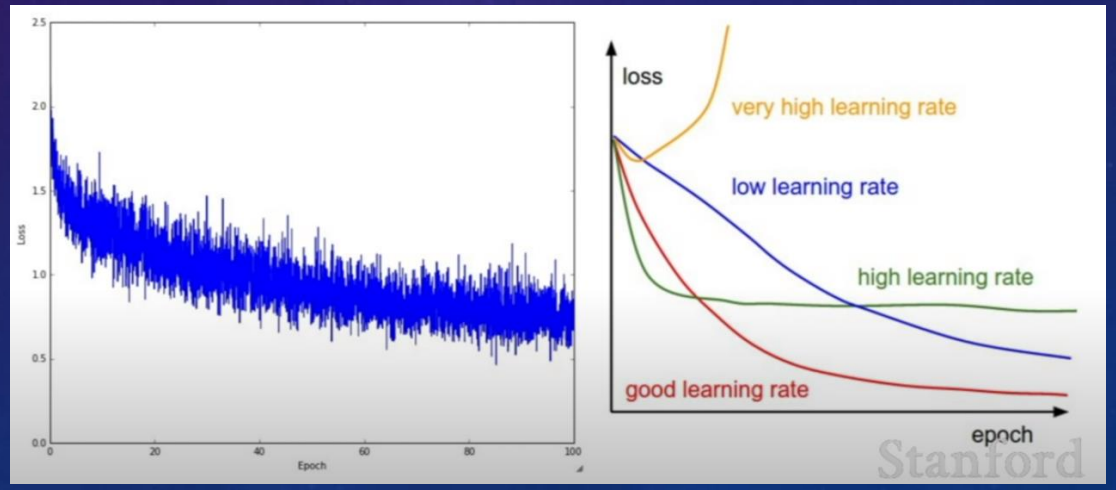
https://www.youtube.com/watch?v=qDbpYUbf3e0&feature=emb_logo [8:26]

Hyperparameter optimization

https://www.youtube.com/watch?v=ttE0F7fghfk [9:50]

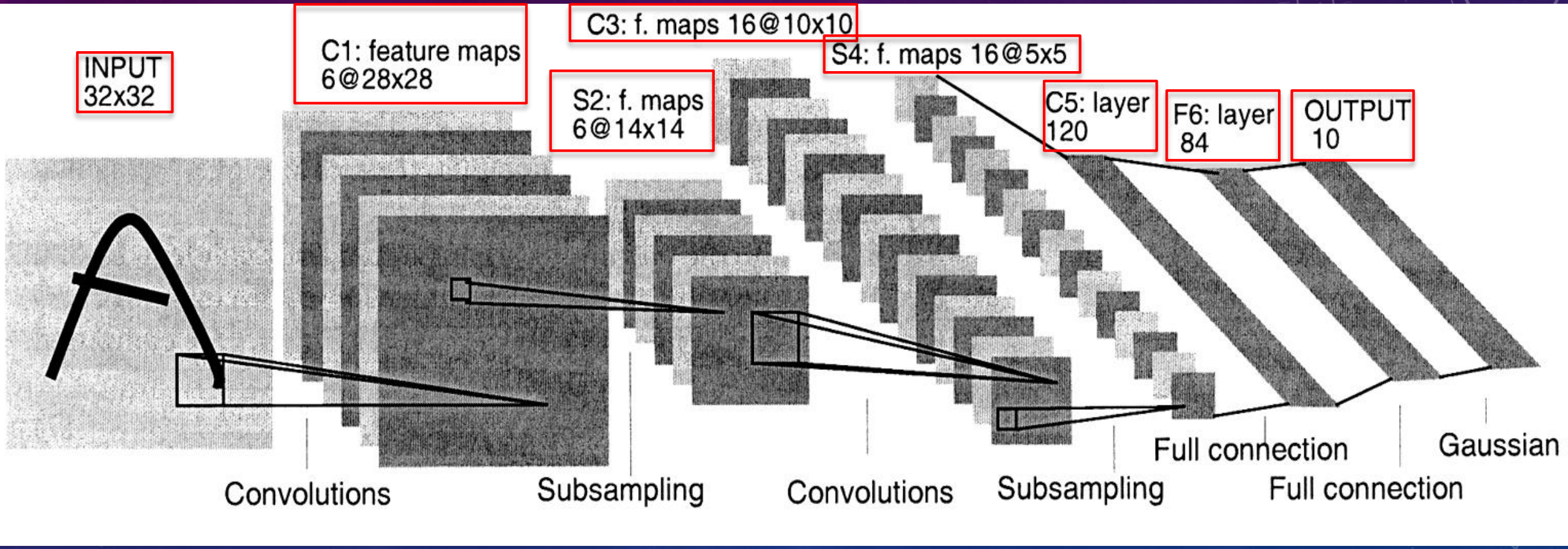
Hyperparameter optimization

- Hyperparameters we can adjust
  - Network architecture
  - Learning rate, its decay schedule, update type
  - Regularization methods
- Observe the loss curve and your image outputs to make improvements



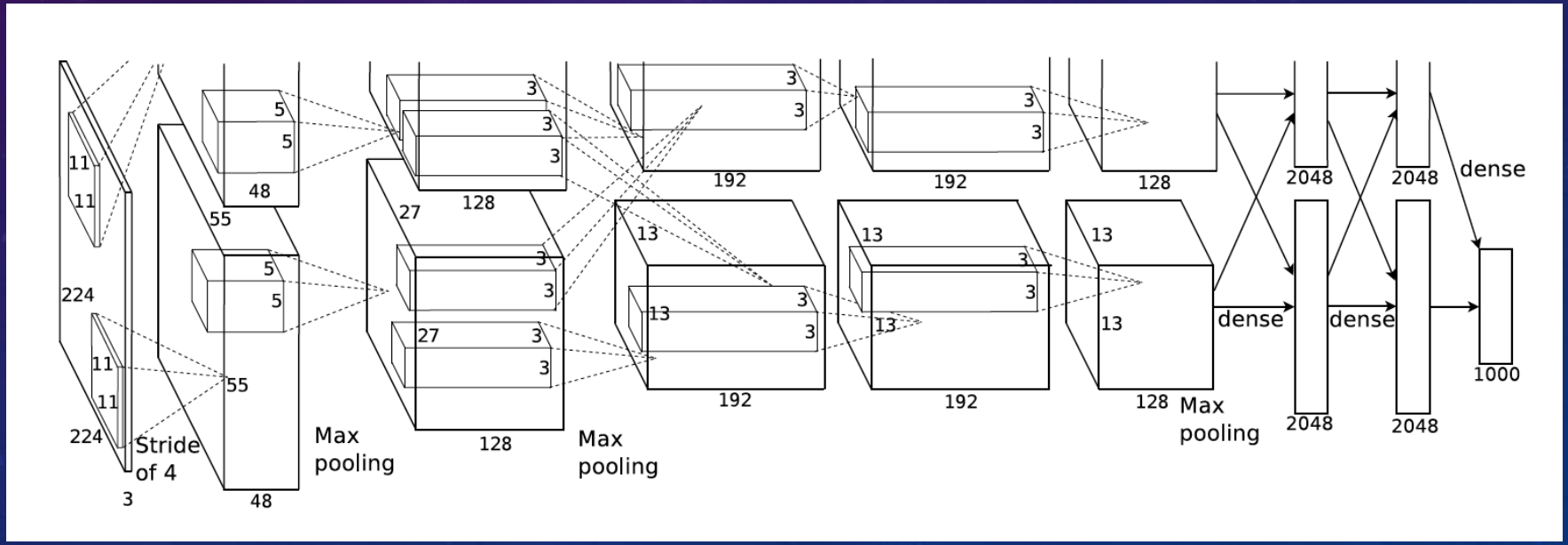
Architecture Introduction

The architecture of LeNet5

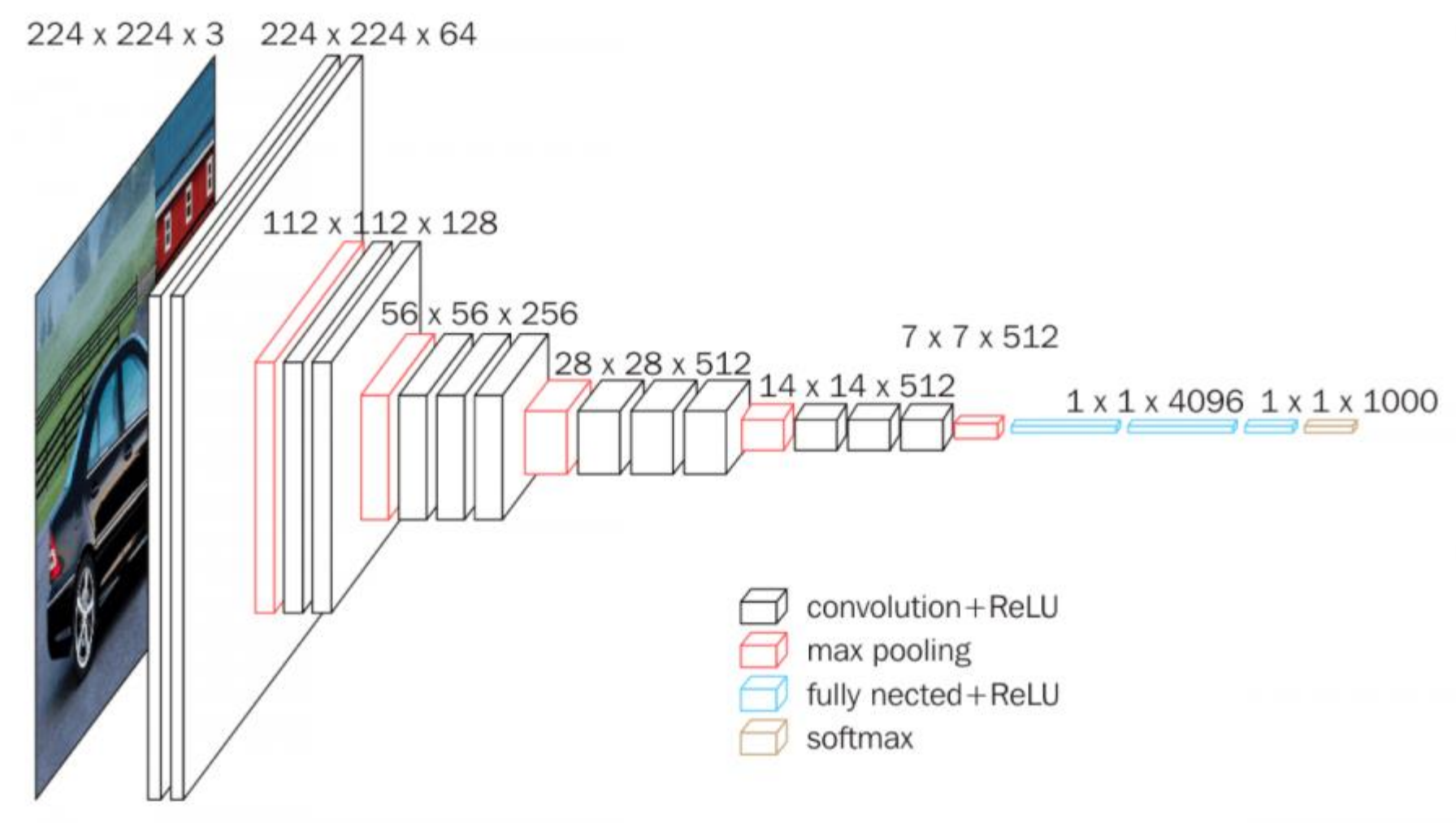


AlexNet

- Use Relu instead of Tanh to add non-linearity. It accelerates the speed by 6 times at the same accuracy.
- Use dropout instead of regularisation to deal with overfitting. However the training time is doubled with the dropout rate of 0.5.
- Overlap pooling to reduce the size of network. It reduces the top-1 and top-5 error rates by 0.4% and 0.3%, repectively.

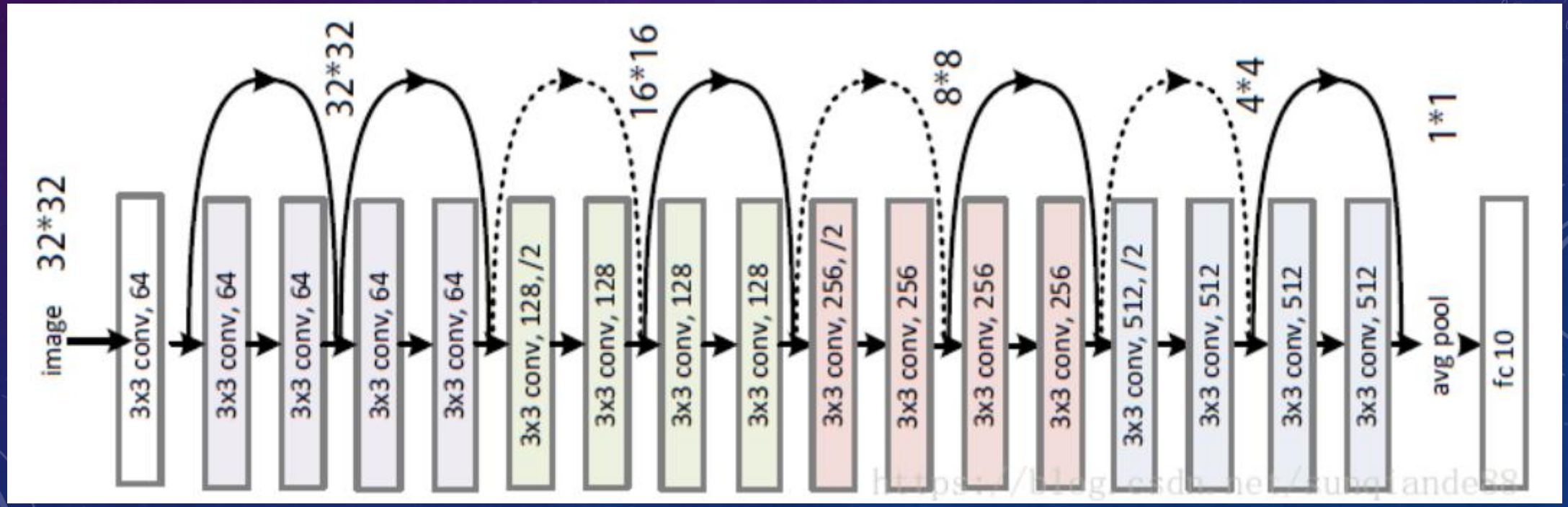


ILSVRC-2014 VGG Model



ResNet18

- Since AlexNet, the state-of-the-art CNN architecture is going deeper and deeper.
- However, increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the notorious vanishing gradient problem
- The core idea of ResNet is introducing a so-called “identity shortcut connection” that skips one or more layers, as shown in the following figure:



CNN Architectures – Stanford University School of Engineering

LeNet (02:47 - 03:15)

AlexNet (03:16 – 05:26)

(05:48 – 06:39)

(06:56 – 07:33)

(08:14 – 11:54)

(12:08 – 12:53)

(12:56 – 13:48)

(14:02 – 15:29)

VGGNet (15:30 – 17:10)

(18:15 – 20:58)

(25:05 – 27:36)

(28:28 – 28:37)

GoogLeNet (28:37 – 33:17)

(36:50 – 40:11)

(41:20 – 43:55)

(47:07 – 47:23)

ResNet (47:24 – 53:08)

(53:56 – 55:01)

(58:16 – 1:02:33)

CNN Architectures – Stanford University School of Engineering

https://www.youtube.com/watch?v=DAOcjicFr1Y&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&index=9 [1:17:39]

See Coding Manual, Chapter 3, Page 74.

LeNet Parameters

Layer Name	Input W×H×D	Kernel W×H×D/S	Output W×H×D	Params
C1: conv2d	32×32×1	5×5×6	28×28×6	1×5×5×6+6 =156 weights biases
S2: pool/2	28×28×6	2×2/2	14×14×6	0
C3: conv2d	14×14×6	5×5×16	10×10×16	6×5×5×16+16 =2,416
S4: pool/2	10×10×16	2×2/2	5×5×16	0
C5: conv2d	5×5×16	5×5×120	1×1×120	16×5×5×120+120 =48,120
F6: conv2d	1×1×120	1×1×84	1×1×84	120×1×1×84+84 =10,164
F7: conv2d	1×1×84	1×1×10	1×1×10	84×1×1×10+10 =850
			Total	61,706

AlexNet Parameters

AlexNet Network - Structural Details												
Input			Output			Layer	Stride	Pad	Kernel size		in	out	# of Param
227	227	3	55	55	96	conv1	4	0	11	11	3	96	34944
55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	0
27	27	96	27	27	256	conv2	1	2	5	5	96	256	614656
27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	0
13	13	256	13	13	384	conv3	1	1	3	3	256	384	885120
13	13	384	13	13	384	conv4	1	1	3	3	384	384	1327488
13	13	384	13	13	256	conv5	1	1	3	3	384	256	884992
13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	0
						fc6			1	1	9216	4096	37752832
						fc7			1	1	4096	4096	16781312
						fc8			1	1	4096	1000	4097000
Total													**62,378,344**

VGG16 Parameters

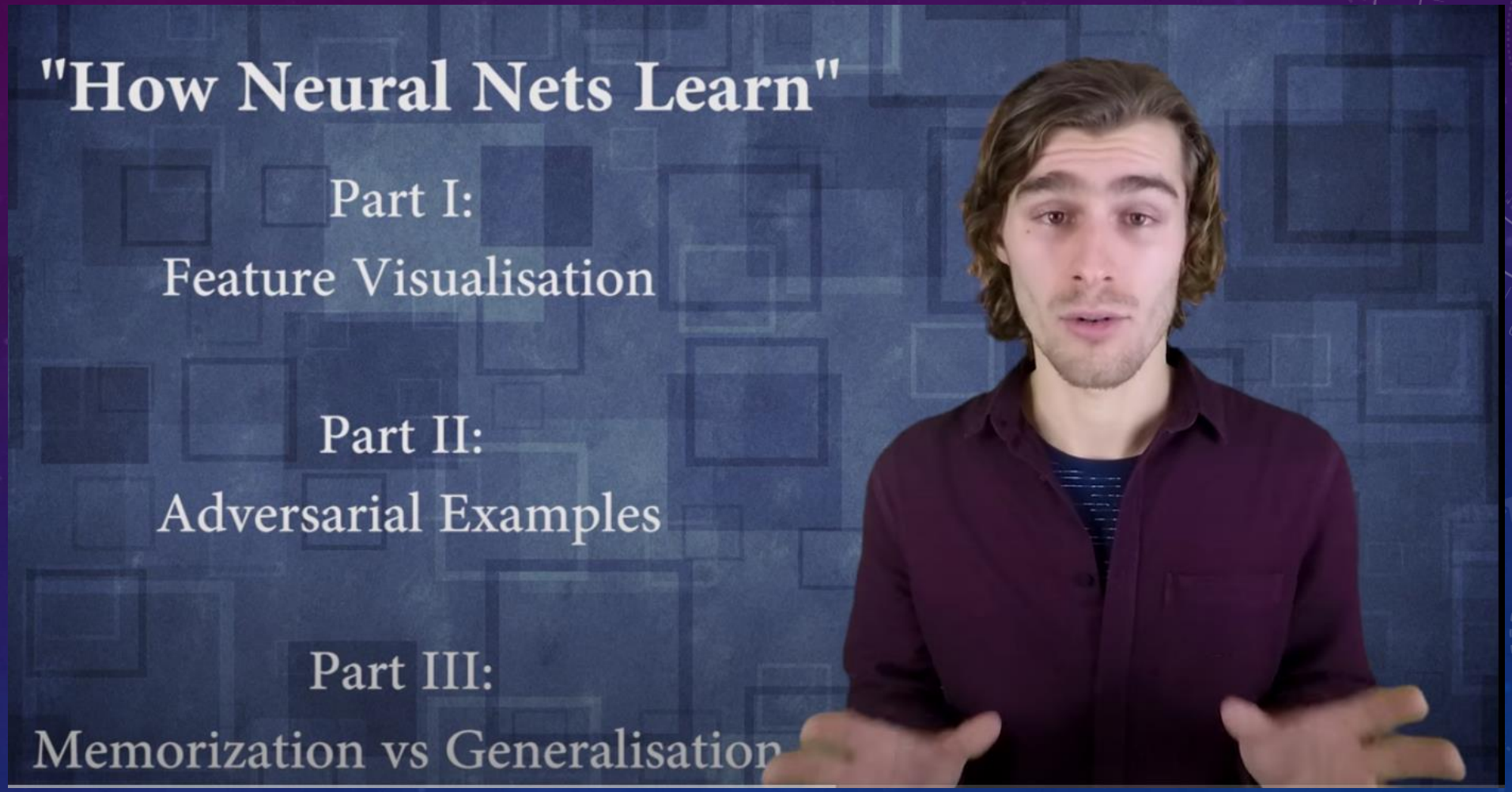
Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 224, 224, 3)	0	
block1_conv1 (Convolution2D)	(None, 224, 224, 64)	1792	input_1[0][0]
block1_conv2 (Convolution2D)	(None, 224, 224, 64)	36928	block1_conv1[0][0]
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0	block1_conv2[0][0]
block2_conv1 (Convolution2D)	(None, 112, 112, 128)	73856	block1_pool[0][0]
block2_conv2 (Convolution2D)	(None, 112, 112, 128)	147584	block2_conv1[0][0]
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0	block2_conv2[0][0]
block3_conv1 (Convolution2D)	(None, 56, 56, 256)	295168	block2_pool[0][0]
block3_conv2 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv1[0][0]
block3_conv3 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv2[0][0]
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0	block3_conv3[0][0]
block4_conv1 (Convolution2D)	(None, 28, 28, 512)	1180160	block3_pool[0][0]
block4_conv2 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv1[0][0]
block4_conv3 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv2[0][0]
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0	block4_conv3[0][0]
block5_conv1 (Convolution2D)	(None, 14, 14, 512)	2359808	block4_pool[0][0]
block5_conv2 (Convolution2D)	(None, 14, 14, 512)	2359808	block5_conv1[0][0]
block5_conv3 (Convolution2D)	(None, 14, 14, 512)	2359808	block5_conv2[0][0]
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0	block5_conv3[0][0]
flatten (Flatten)	(None, 25088)	0	block5_pool[0][0]
fc1 (Dense)	(None, 4096)	102764544	flatten[0][0]
fc2 (Dense)	(None, 4096)	16781312	fc1[0][0]
predictions (Dense)	(None, 1000)	4097000	fc2[0][0]

Total params: 138357544

ResNet18 Parameters

ResNet18 - Structural Details														
#	Input Image			output			Layer	Stride	Pad	Kernel		in	out	Param
1	227	227	3	112	112	64	conv1	2	1	7	7	3	64	9472
	112	112	64	56	56	64	maxpool	2	0.5	3	3	64	64	0
2	56	56	64	56	56	64	conv2-1	1	1	3	3	64	64	36928
3	56	56	64	56	56	64	conv2-2	1	1	3	3	64	64	36928
4	56	56	64	56	56	64	conv2-3	1	1	3	3	64	64	36928
5	56	56	64	56	56	64	conv2-4	1	1	3	3	64	64	36928
6	56	56	64	28	28	128	conv3-1	2	0.5	3	3	64	128	73856
7	28	28	128	28	28	128	conv3-2	1	1	3	3	128	128	147584
8	28	28	128	28	28	128	conv3-3	1	1	3	3	128	128	147584
9	28	28	128	28	28	128	conv3-4	1	1	3	3	128	128	147584
10	28	28	128	14	14	256	conv4-1	2	0.5	3	3	128	256	295168
11	14	14	256	14	14	256	conv4-2	1	1	3	3	256	256	590080
12	14	14	256	14	14	256	conv4-3	1	1	3	3	256	256	590080
13	14	14	256	14	14	256	conv4-4	1	1	3	3	256	256	590080
14	14	14	256	7	7	512	conv5-1	2	0.5	3	3	256	512	1180160
15	7	7	512	7	7	512	conv5-2	1	1	3	3	512	512	2359808
16	7	7	512	7	7	512	conv5-3	1	1	3	3	512	512	2359808
17	7	7	512	7	7	512	conv5-4	1	1	3	3	512	512	2359808
	7	7	512	1	1	512	avg pool	7	0	7	7	512	512	0
18	1	1	512	1	1	1000	fc					512	1000	513000
Total														11,511,784

‘How neural networks learn’ - Part I: Feature Visualization



https://www.youtube.com/watch?v=McgxRxi2Jqo&feature=youtu.be&ab_channel=ArxivInsights [14:59]