What is Machine Learning?

Task# 1: Add two number

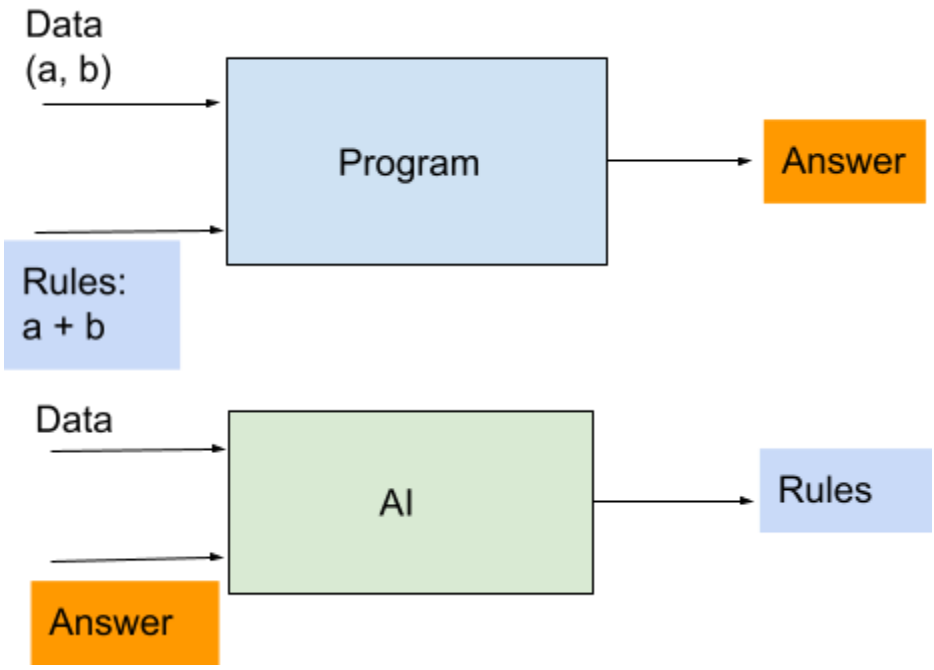
```
def add(a, b):
    return (a+b)
```

Task# 2: Given a photograph, find the human face in the photograph.

```
def find_face(image):
```

- Certain rules to detect faces.
- Some of these rules are brittle.
- Not able to come up with a universally applicable algorithm for face detection.

Addition of two numbers	**Face detection**
We know the exact mathematical form of the function.	We do not know the exact mathematical form of the function.
	As a human we are good at detecting faces from the images.
	Data (Image 1, Image 2, …. , Image n) Answers (1, 0, …, 1)



Linear regression

Predict pay package for students based on the number of hours they study and marks in the final examination.

- Data: {(h_1, s_1), (h_2, s_2), …., (h_n, s_n)}
- Answers: {y_1, y_2, …., y_n}
- Model (Rule):

  y = b + w_1 * h + w_2 * s

  b, w_1 and w_2 are called as parameters of the model.

  Features: h, s

  Label: y

AI/ML:
Given the data (a.k.a training data), find the optimal values of parameters of the model.

Components of ML or AI model:

1. Training/validation/test data e.g. {(h_1, s_1, y_1), (h_2, s_2, y_2), ... , (h_n, s_n, y_n)} where n can be 100s or in 1000s or in millions.
2. Model: e.g. Linear regression: y = b + w_1 * h + w_2 * s
3. Loss function or objective function e.g. MSE
4. Optimization procedures e.g. Gradient Descent
5. Evaluation criteria

Three parts:

1. **Training**: Given a model and training data, we try to obtain parameters of the model such that the error or loss function is minimized.
2. **Inference**: Given the model and its parameters obtained during the training pass, we try to predict a label for a new example with the model and its parameters. For example, assuming model: y = b + w_1 * h + w_2 * s and b = 1, w_1 = 1 and w_2 = 2, what package is expected for a student with h=10, s=90?
   y = 1 + 1 * 10 + 2 * 90 = 191
3. **Evaluation**
   a. We take unseen examples (that were not supplied to the model during training) and calculate the output.
   b. Compare the predicted output with the actual output and calculate certain metrics.

MLP or multi-layer perceptron or neural network or deep models or feed-forward NNs
These are very popular and state of the art ML models.

b + w1 x1 + w2 x2 + w3 x1^2 + w4 x2^2 + w5 x1x2

Practical walkthrough: playground.tensorflow.org

Neural networks

Different names:

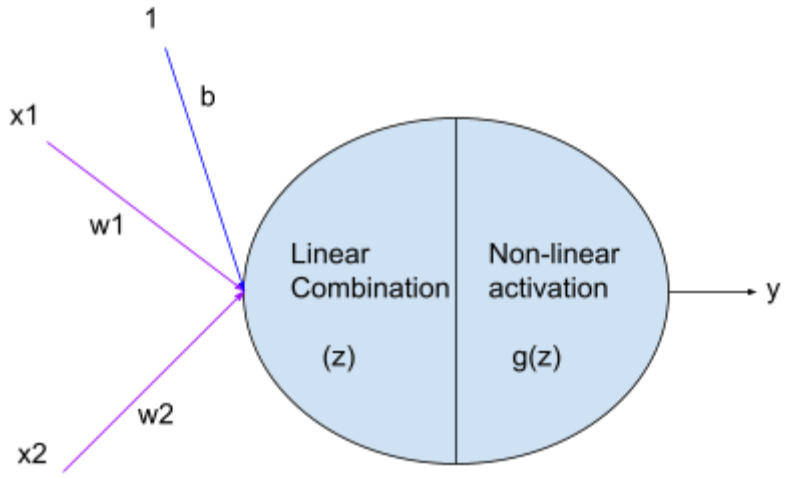
+ MLP (Multi layered perceptron)
+ Feed forward neural networks (FFNN)
+ Deep model

Basic component of neural network

- Input layer, where we specify feature inputs
- Hidden layers
  - Neurons or units
- Output layer

Any NN can have one or more hidden layers.

Neuron or unit



z = b + w1 * x1 + w2 * x2 (Linear combination)
y = g(z) (Non linear activation)

Activation	Formula	When to use?
Sigmoid	y = 1/(1 + exp(-z)) [Visualization]	• In output layer for binary classification. For multiclass, we use softmax as an activation function. • Neurons in hidden layers
RELU (Rectified linear units)	g(z) = 0 if z < 0 else z max(0, z) Any z less than 0, will get 0 from relu function.	• Neurons of hidden layers.
Tanh	y = (exp(z) - exp(-z))/(exp(z) + exp(-z))	• Neurons of hidden layers
Linear	g(z) = z	• In case of linear regression problem, we use linear activation in the output layer. • Neurons of the hidden layer. This is rarely done.

Practical example:

Let's say b=0.5, w1=1, w2=-1, we will study how the forward computation or inference works in a single neuron.

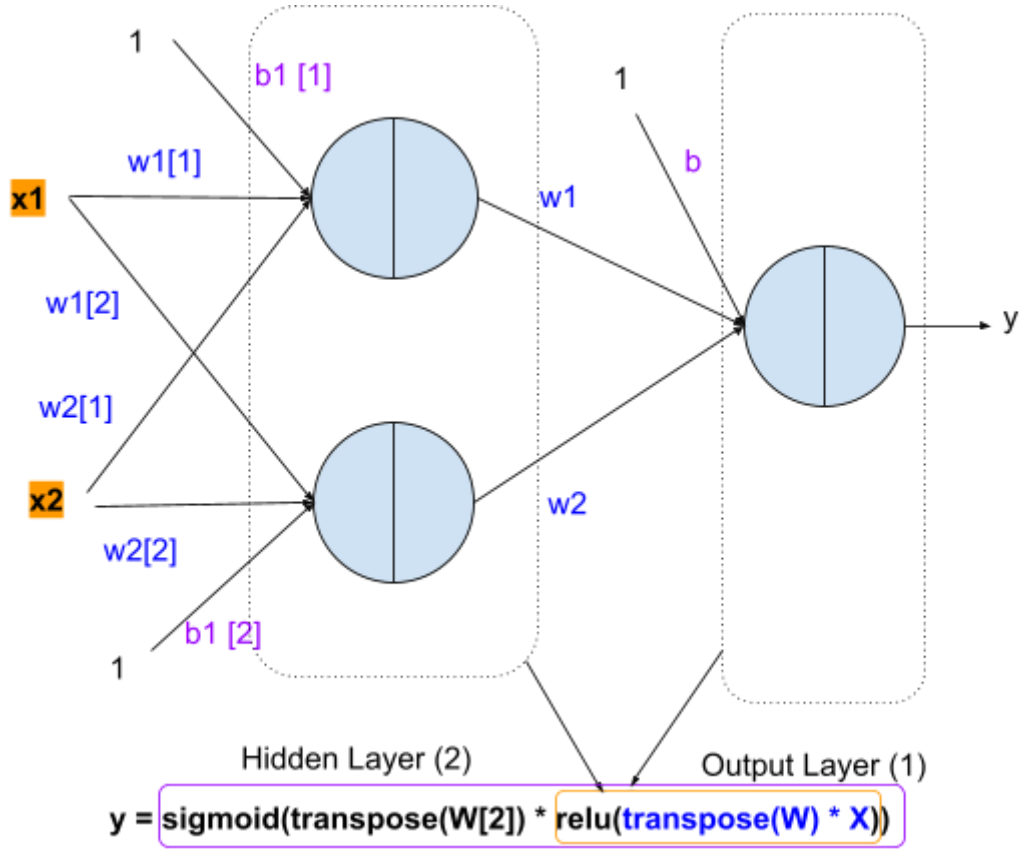
x1	**x2**	z = b + w1x1 + w2x2 z =0.5 + 1 * x1 -1 * x2	Sigmoid	RELU max(0, z)	Linear	Tanh
1	**1**	0.5	0.62	0.5	0.5	
1	**2**	-0.5	0.38	0	-0.5	
0.5	**1**	0	0.5	0	0	

Number of parameters: 3
In case, you have m features, how many parameters for a single neuron? m+1

Let's see what is the effect of linear activation on the neural network through playground exercise.

Combination of neuron or architecture of neural network

g1(z) = g1(b1[1] + w1[1] * x1 + w2[1] * x2) = g1(transpose**(W[1]) * X))**
g2(z) = g2(b1[2] + w1[2] * x1 + w2[2] * x2) = g2(transpose**(W[2]) * X))**



X = Feature vector -> column vector

Feed-forward neural network or MLP. Assuming relu as an activation for the hidden layer, let's write down equations for forward computation:

First neuron in the first layer:

- z[1] = b1[1] + w1[1] * x1 + w2[1] * x2
- g(z)[1] = max(0, z[1])

Second neuron in the first layer:

- z[2] = b1[2] + w1[2] * x1 + w2[2] * x2
- g(z)[2] = max(0, z[2])

Output layer: Activation:

- Sigmoid

    - z = b + w1 * g(z)[1] + w2 * g(z)[2]
    - y = sigmoid(z)
- Linear activation:
    - y = z

Example: [Forward computation]

Let's first fix the parameter values. This is usually done through a learning or training process.

First hidden layer

- First neuron
    - b1[1] = 2
    - w1[1] = 1
    - W2[1] = 1
- Second neuron
    - b1[2] = -3
    - w1[2] = 1
    - w2[2] = -1

Second hidden layer

- b = 3
- w1 = 0.5
- w2 = 0.5

The training example: x1=10, x2=20

First neuron in the first layer:

- z[1] = b1[1] + w1[1] * x1 + w2[1] * x2 = 2 + 1 * 10 + 1 * 20 = 32
- g(z)[1] = max(0, z[1]) = max(0, 32) = 32

Second neuron in the first layer:

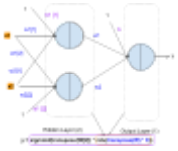
- z[2] = b1[2] + w1[2] * x1 + w2[2] * x2 = -3 + 1 * 10 - 1 *20 = -13
- g(z)[2] = max(0, z[2]) = max(0, -13) = 0

Output layer: Activation:

- Sigmoid
    - z = b + w1 * g(z)[1] + w2 * g(z)[2] = 3 + 0.5 * 32 + 0.5 * 0 = 19
    - y = sigmoid(z) = sigmoid(19) = 0.9999

- Linear activation: y = z = 19

Note on implementation: The forward computation is implemented through linear algebra operations - vector matrix multiplication.

- Parameter vector
- Feature matrix
- 

The computation in the first hidden layer is demonstrated in matrix form in the following figure:

	u1	u2
b	2	-3
w1	1	1
w2	1	-1

Parameter Matrix (W)

	b	w1	w2
u1	2	1	1
u2	-3	1	-1

transpose(W)
2 x 3

*

	e1
	1
x1	10
x2	20

X (3 x 1)

=

e1
2 * 1 + 1 * 10 + 1 * 20 = 32
-3*1 + 1 * 10 -1 * 20 = -13

relu

e1
32
-13

=

e1
32
0

For the second layer:

	b	w1	w2
n1	3	0.5	0.5

*

e1
1
32
0

= 3 * 1 + 0.5 * 32 + 0.5 * 0 = 19

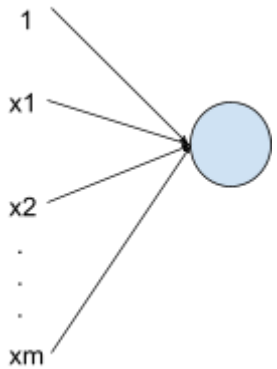
transpose(W[2]) * g(Z[1])

sigmoid (19) = 0.99999

Now let's work it out with two examples:

First example x1:10, x2: 20
Second example x1:5, x2: 10

Parameter vector (W): (m+1) x k



	n1	n2
b	2	-3
w1	1	1
w2	1	-1

Transpose(W): k units and (m+1) parameters: k x (m+1)

	b	w1	w2
u1	2	1	1
u2	-3	1	-1

Feature matrix (X) = m * n where **m**: # features and **n**: # examples

	e1	e2
	1	1
x1	10	5
x2	20	10

Z = Transpose(W) * X: k x n

e1	e2
2 * 1 + 1 * 10 + 1 * 20 = 32	2 * 1 + 1 * 5 + 1 * 10 = 17
-3*1 + 1 * 10 -1 * 20 = -13	-3 * 1 + 1 * 5 -1 * 10 = -8

,

G(Z) = relu(Z)

e1	e2
max(0, 32) = 32	max(0, 17) = 17

max(0, -13) = 0	max(0, -8) = 0

Output layer (activation: Sigmoid): sigmoid(transpose(W[2])*G(Z))
Transpose(W[2])

	b	w1	w2
n1	3	0.5	0.5

G(Z)

e1	e2
1	1
max(0, 32) = 32	max(0, 17) = 17
max(0, -13) = 0	max(0, -8) = 0

Z[2] = transpose(W[2]) * G(Z) = [1, 3] * [3, 2] = [1,2]

3 * 1 + 0.5 * 32 + 0.5 * 0 = 19
3 * 1 + 0.5 * 17 + 0.5 * 0 = 11.5

y = sigmoid(Z[2])

sigmoid(19) = 1	sigmoid(11.5) = 1

In matrix notation:

- **y = sigmoid(transpose(W[2]) * relu(transpose(W) * X))**

	Output activations
Binary classification problem (2 classes)	Sigmoid activation
Regression	Linear activation

Training of neural networks

1. Loss functions
2. Forward computation (Inference)
3. Optimization algorithm (G.D.)

Forward Computation: y = sigmoid(transpose(W[2]) * relu(transpose(W[1]) * X))

Training needs to find parameters W[2] and W[1] given the training data such that the loss is minimized.

Loss function: function(actual output, predicted output)

- Regression: Mean squared error
- Classification
    - Binary classification: Cross entropy loss
    - Multiclass classification:
        - Categorical cross entropy loss or
        - Sparse categorical cross entropy loss

Optimization:

- Gradient descent optimization

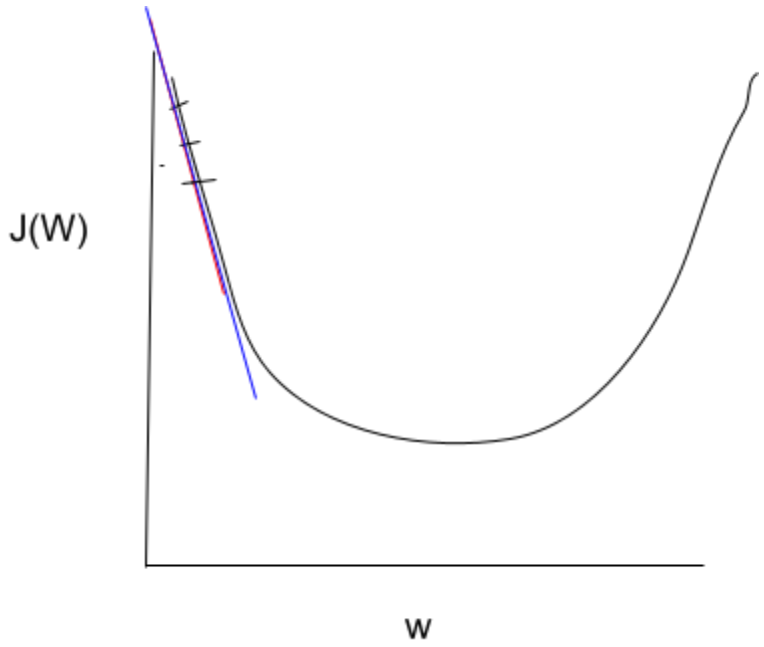
Training procedure:

1. Randomly initialize **W**[1] and **W**[2]
2. Repeat until convergence
    a. Compute y using the forward computation step and let's call it y_hat for every training example.
    b. Loss: J(W) = Loss(y (actual), y_hat (predicted))
    c. Update **W**[1] and **W**[2] with the GD update rule. The update is calculated for every single parameter in W.

i. w[1][i](new) := w[1][i](old) - lr * partial derivative of loss w.r.t w1[i].

d. Update simultaneously all W.

Partial derivative of loss w.r.t. the parameter is calculated with a special procedure called backpropagation.



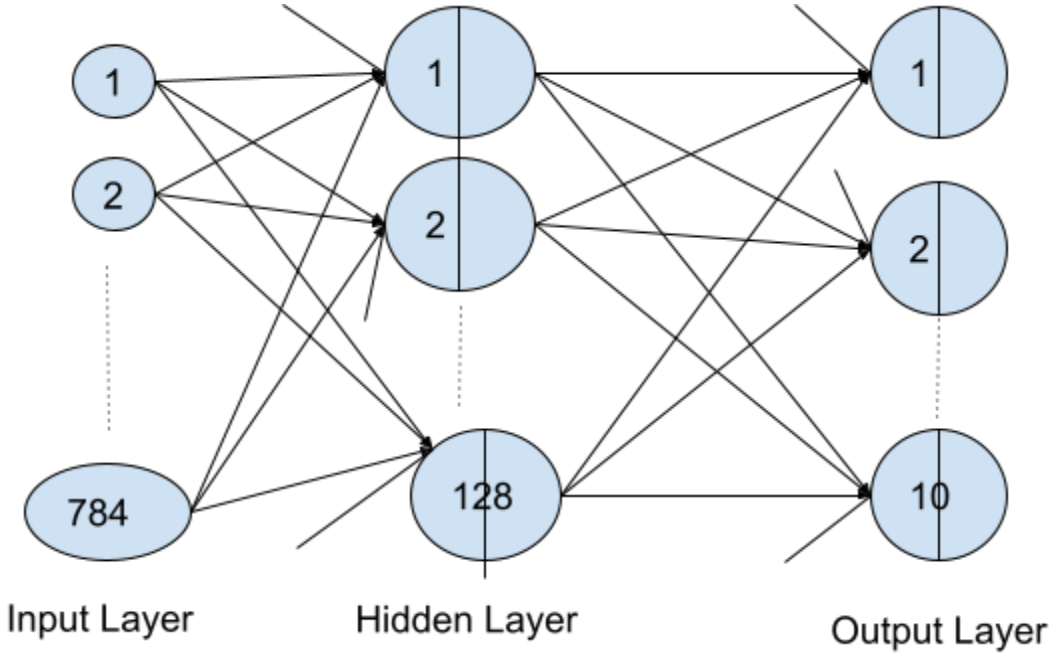
Linear regression - MSE

J(W) = 1/2n * \sum_i=1^n (y[i] - y_hat[i])^2

Lab Session

Regularization:

1. Early stopping
2. Dropout (50%)



Different architectures of neural networks

+ Convolutional neural networks (CNNs): Image understanding, Text processing
+ Recurrent neural networks (RNNs): Language understanding

300*300 image, 128 units, 10 units

+ Flatten: 90000
+ Dense1: 128 [90001*128] = O(11 million) parameters
+ Dense2: 10 [129*10 = 1290]

28*28:

+ Flattern: 784
+ Dense1: 128 [785*128]
+ Dense2: 10 [129*10]

Convolutional neural networks

+ Convolution - Filter
+ Pooling

Convolution
Image which is 5x5 image.

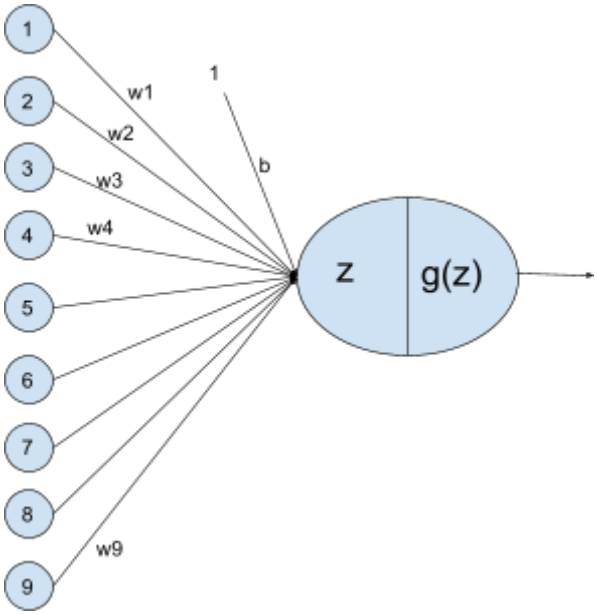
1	0	0	1	1
0	1	1	0	0
1	1	1	1	1
1	1	1	1	1
0	0	0	0	0

Filter

w1	w2	w3
w4	w5	w6
w7	w8	w9

Output (# rows in image - # rows in filter + 1, # col in image - # cols in filter + 1)

f(0,0)	**f(0,1)**	**f(0,2)**
f(1,0)	**f(1,1)**	**f(1,2)**
f(2,0)	**f(2,1)**	**f(2,2)**



f(0,0) = g(z) = g(b + w1x1 + w2x2 + … + w9x9)
= relu(z)

For this example:

Linear Combination:

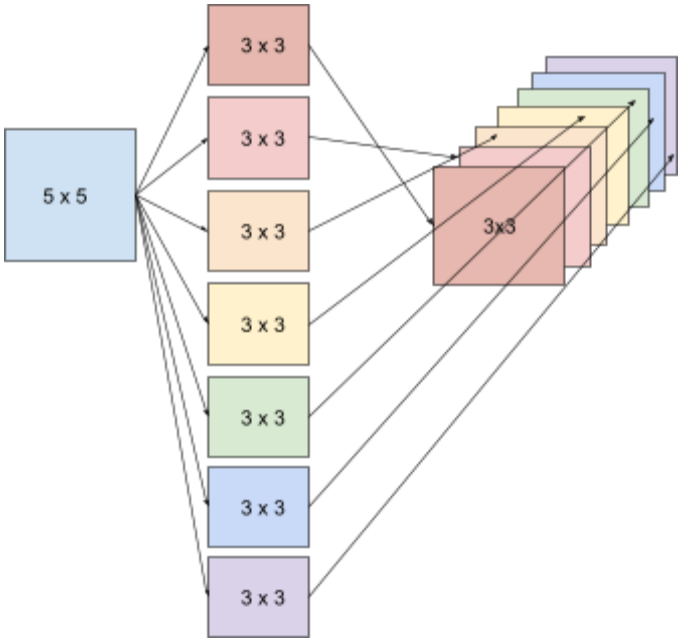
1 * w1	0 * w2	0 * w3
0 * w4	1 * w5	1 * w6
1 * w7	1 * w8	1 * w9

z = b + 1 * w1 + 0 * w2 + 0 * w3 + 0 * w4 + 1 * w5 + 1 * w6 + 1 * w7 + 1 * w8 + 1 * w9
g(z) = relu(z)

Concretely for 7 filters for an image of shape (5, 5) and filter shape (3, 3):

3x3	3x3	3x3	3x3	3x3	3x3	3x3

Output shape = (7, 3, 3)



Pooling operation (2x2 filter)

1	**2**	**3**	**4**
5	**6**	**7**	**8**
9	**10**	**11**	**12**
13	**14**	**15**	**16**

Max pooling

max(v1, v2, v5, v6)	**max(v3, v4, v7, v8)**
max(v9, v10, v13, v14)	**max(v11, v12, v15, v16)**

Average pooling

avg(v1, v2, v5, v6)	**avg(v3, v4, v7, v8)**
avg(v9, v10, v13, v14)	**avg(v11, v12, v15, v16)**

Image → convolution → pooling → convolution → pooling → flatten → FCN → output

Image → features → FCN → output

For example,

- Input image is 28x28, and
- First layer: convolution with 16 filters of size 3x3 with stride = 1
- Second layer: Pooling 2x2 with stride = 2
- Third layer: convolution with 32 filters of size 3x3 with stride = 1

	Input shape	**Output shape**	**Parameters**
Input image is 28x28	(28, 28)		
First layer: convolution with **16 filters** of size **3x3** with stride = 1	(28, 28)	(16, 26, 26)	16 * (3*3 + 1) = 160
Second layer: Pooling 2x2 with stride = 2		(16, 13, 13)	0
Third layer: convolution with **32 filters** of size **3x3** with stride = 1 3x3x16 Since the input has 16 channels, the filter will be of size 3*3*16.	(**16**, 13, 13)	(32, 11, 11)	32 *(3*3*16 + 1) = 32 * 145 = 4640
Pooling 2x2 with stride of 2		(32, 5, 5)	0
Flatten		32 * 5 * 5 = 800	

Transfer learning

[Self study] RNNs:

- Chris Olah's blog on LSTM

CNN Colab

Practical links:

- https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/guide/basic_training_loops.ipynb

- https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/customization/custom_training_walkthrough.ipynb#scrollTo=kesTS5Lzv-M2

Pre-trained model:

Classic one	As a feature	Transfer learning (Fine tuning)
Image	Image	Image
Pretrained model	Pretrained model	Pretrained model - 1-K
	FFNN	Pretrained model - K-M
		FFNN
Output	Output	Output

Transfer learning colab