

# 24 – Convolutional Neural Network

IF4073 Interpretasi dan Pengolahan Citra

Oleh: Rinaldi Munir



Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung  
2022

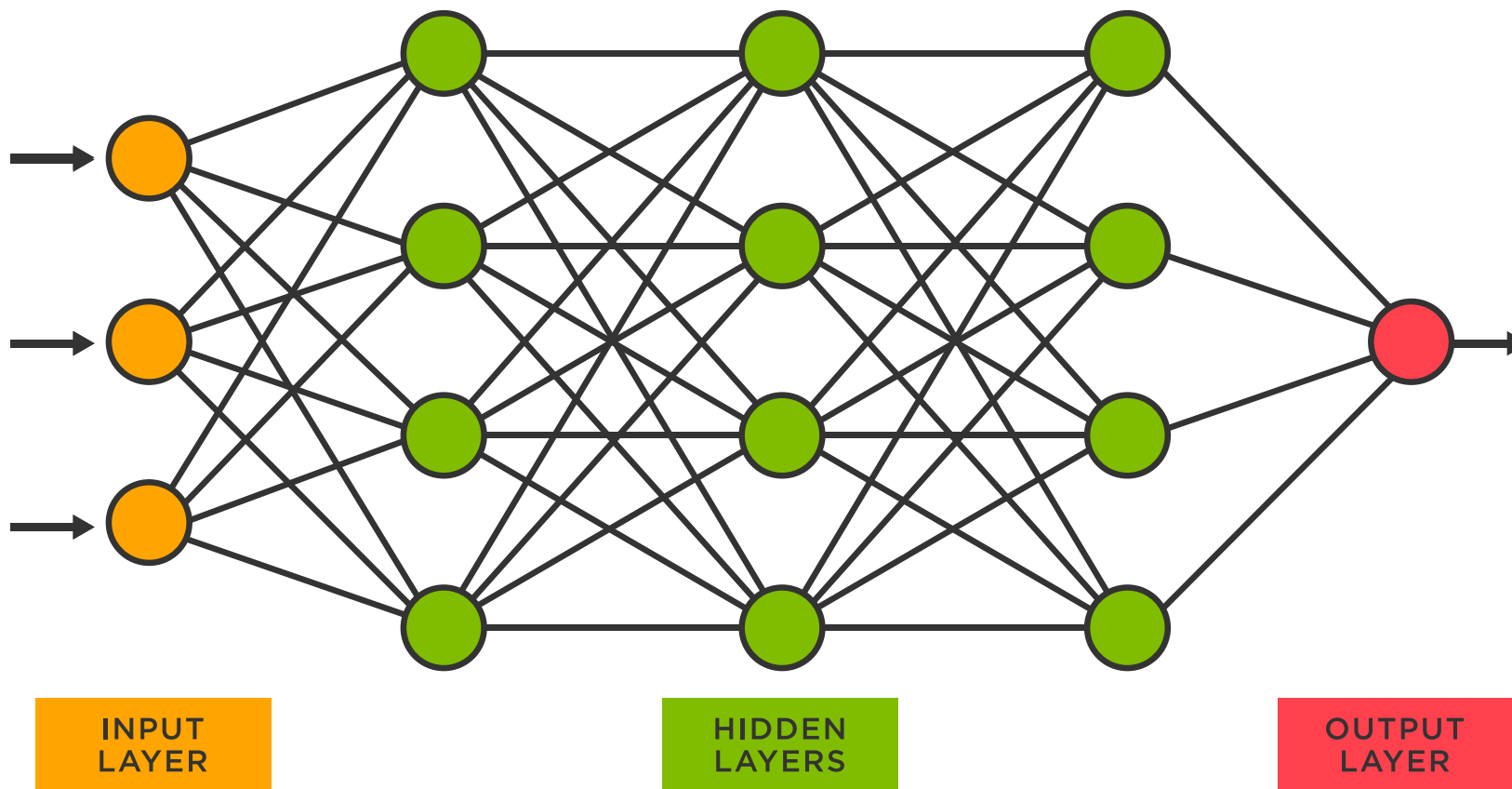
# Referensi

- Sebagian besar materi di dalam PPT ini diambil dari *mathworks*, antara lain:
  - a) <https://www.mathworks.com/help/deeplearning/ug/deep-learning-in-matlab.html>
  - b) <https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html>
- Beberapa *slide* juga diambil dari materi kuliah deep learning tentang CNN

# Pembelajaran Mendalam

- Pembelajaran mendalam (*deep learning*) adalah jenis pembelajaran mesin (*machine learning*) yang dalam hal ini model belajar untuk melakukan tugas klasifikasi langsung dari gambar, teks, atau suara.
- Pembelajaran mendalam biasanya diimplementasikan menggunakan arsitektur jaringan saraf.
- Istilah “deep” (dalam) mengacu pada jumlah lapisan di dalam jaringan—semakin banyak lapisan, semakin dalam jaringan. Jaringan saraf tradisional hanya berisi 2 atau 3 lapisan, sedangkan jaringan dalam dapat memiliki ratusan lapisan.

(Sumber: Mathworks)



Jaringan Syaraf Tiruan (*Artificial Neural Netwok*)

# Computer Vision

- Salah satu kegunaan pembelajaran mendalam adalah di dalam *computer vision*

## Image Classification

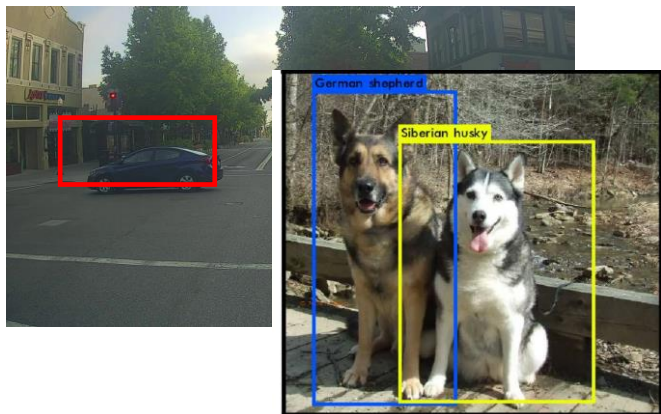


64x64



Cat? (0/1)

## Object detection



## Neural Style Transfer

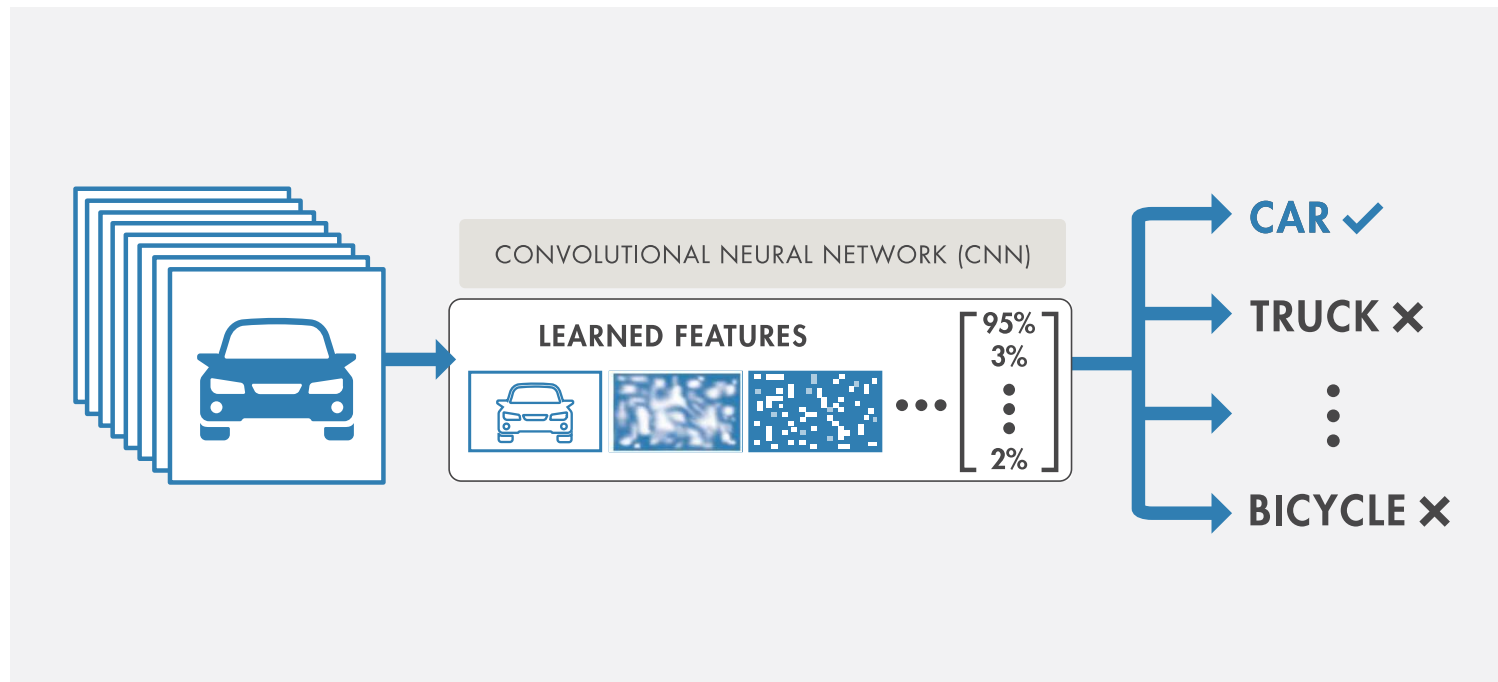


Sumber: Bahan kuliah Deep Learning

# Apa yang Membuat Pembelajaran Mendalam Cocok untuk Computer Vision?

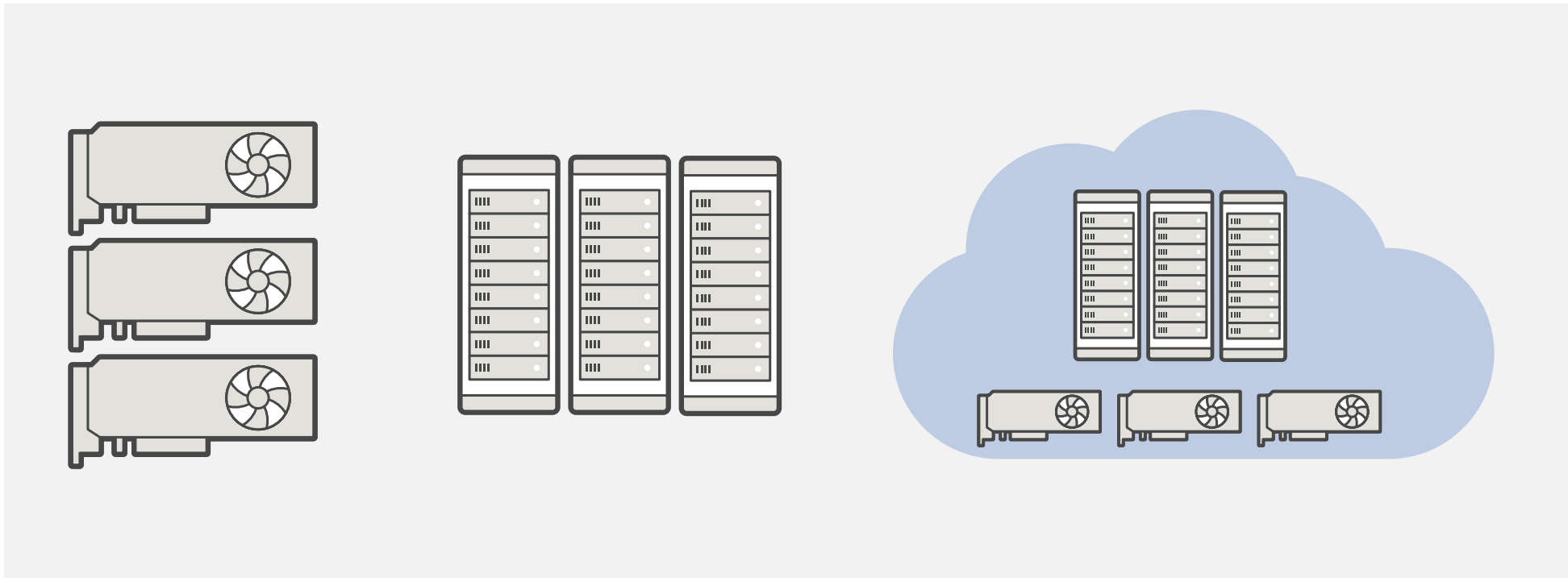
## 1. Kemudahan akses ke kumpulan besar data yang berlabel

Kumpulan data seperti ImageNet, COCO, PASCAL VoC tersedia secara *free*, dan berguna untuk melatih berbagai jenis objek.



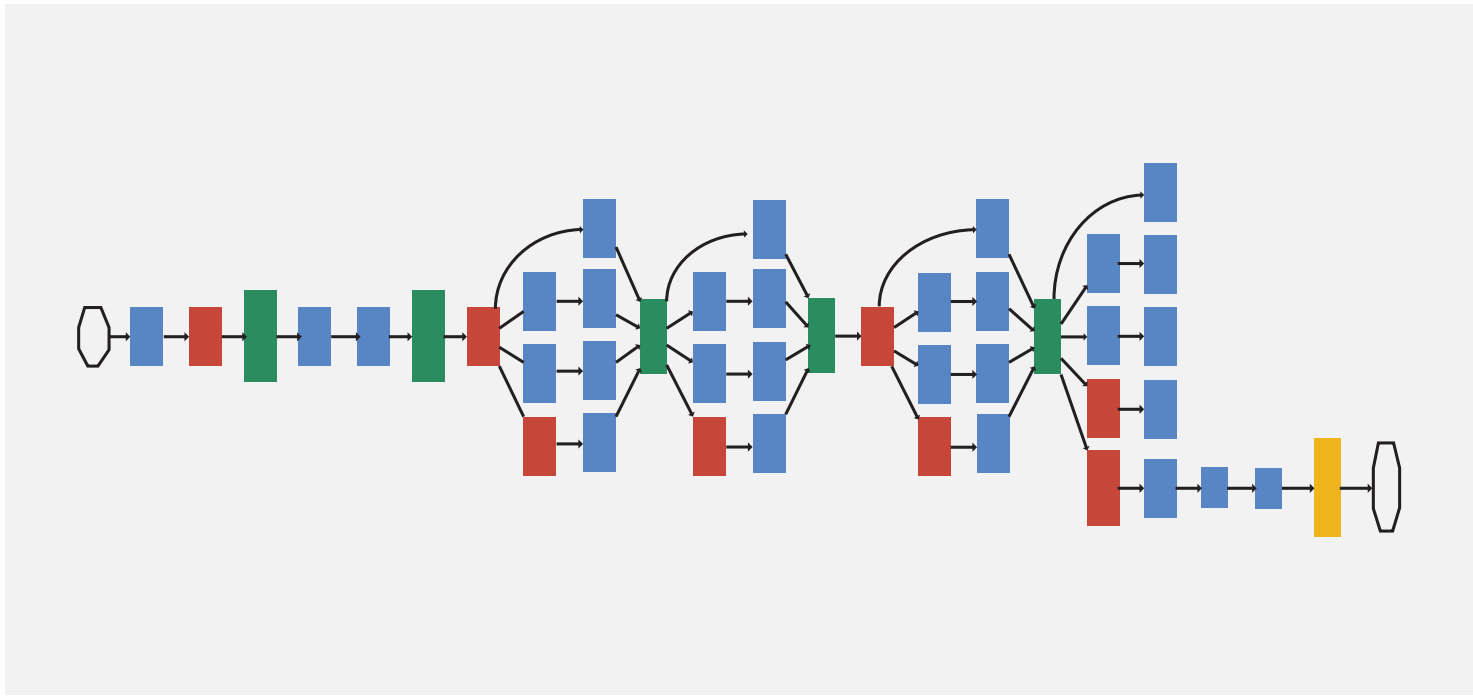
## 2. Peningkatan daya komputasi

GPU berkinerja tinggi (*high performa computing*) mempercepat pelatihan sejumlah besar data yang diperlukan untuk pembelajaran mendalam, mengurangi waktu pelatihan dari berminggu-minggu menjadi berjam-jam.



### 3. Model terlatih (*pretrained*) yang dibangun oleh para ahli

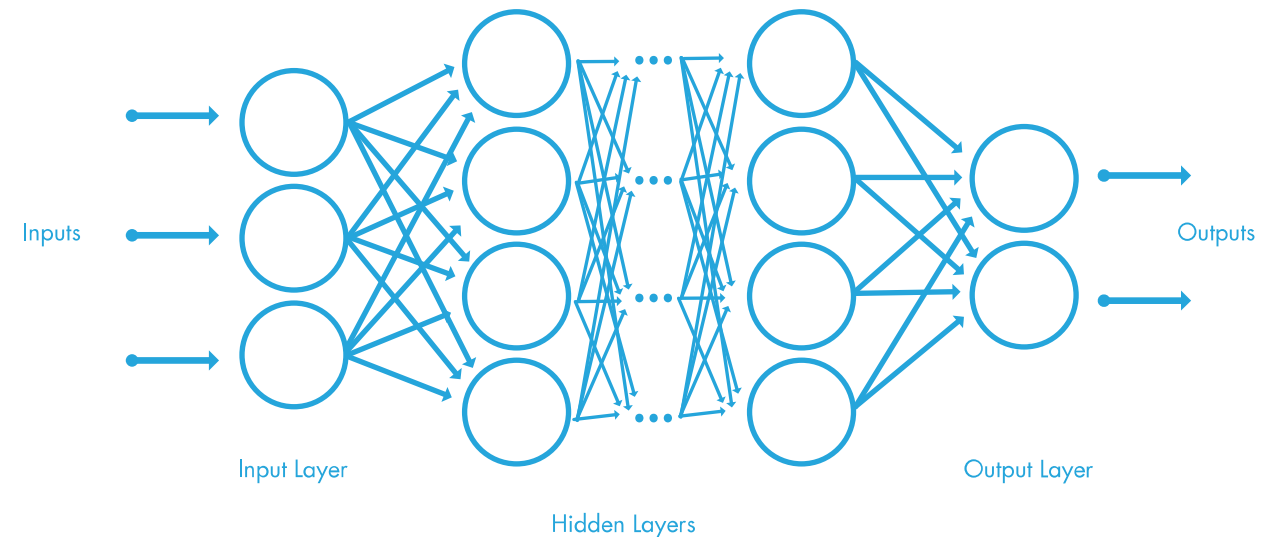
Model seperti AlexNet dapat dilatih ulang untuk melakukan tugas pengenalan baru menggunakan teknik yang disebut pembelajaran transfer (*transfer learning*). Sementara AlexNet dilatih pada 1,3 juta gambar beresolusi tinggi untuk mengenali 1000 objek berbeda, pembelajaran transfer yang akurat dapat dicapai dengan jauh lebih cepat.



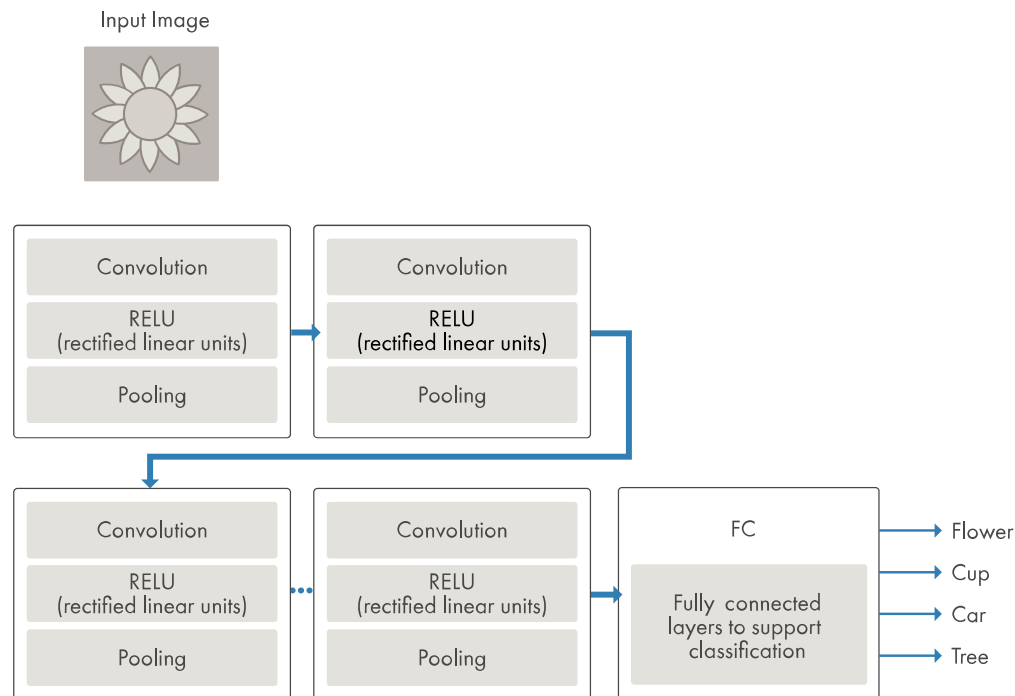
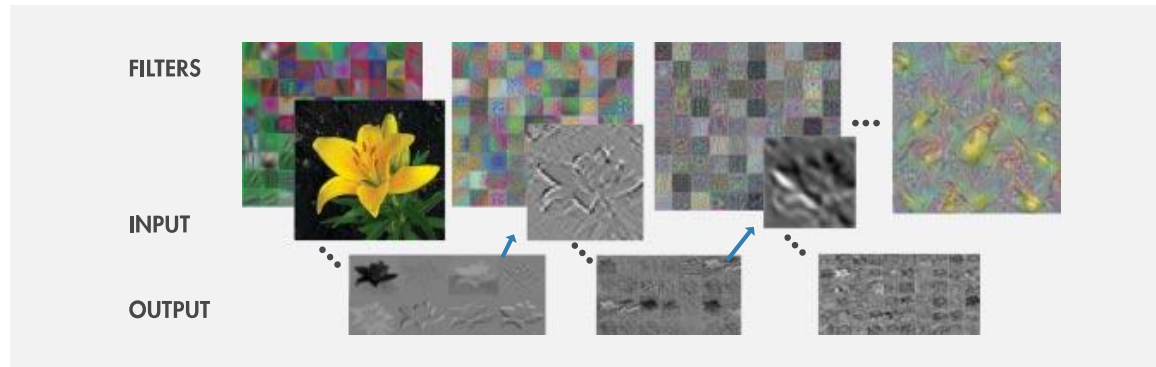


# Deep Neural Network

- *Deep neural network* (DNN) adalah jaringan syaraf tiruan (ANN) dengan banyak lapisan (*layer*) di antara lapisan masukan dan lapisan luaran.
- Lapisan-lapisan tersebut saling berhubungan melalui node, atau neuron, dengan setiap lapisan tersembunyi menggunakan output dari lapisan sebelumnya sebagai inputnya.
- Lapisan-lapisan itu merupakan pemrosesan non-linier, bekerja secara parallel dan terinspirasi oleh sistem saraf biologis.



# Bagaimana DNN Belajar?



Misalkan kita memiliki satu set gambar di mana setiap gambar berisi satu dari empat kategori objek yang berbeda, dan kita ingin jaringan pembelajaran mendalam mengenali secara otomatis objek mana yang ada di setiap gambar.

Kita memberi label pada gambar agar memiliki data pelatihan untuk jaringan.

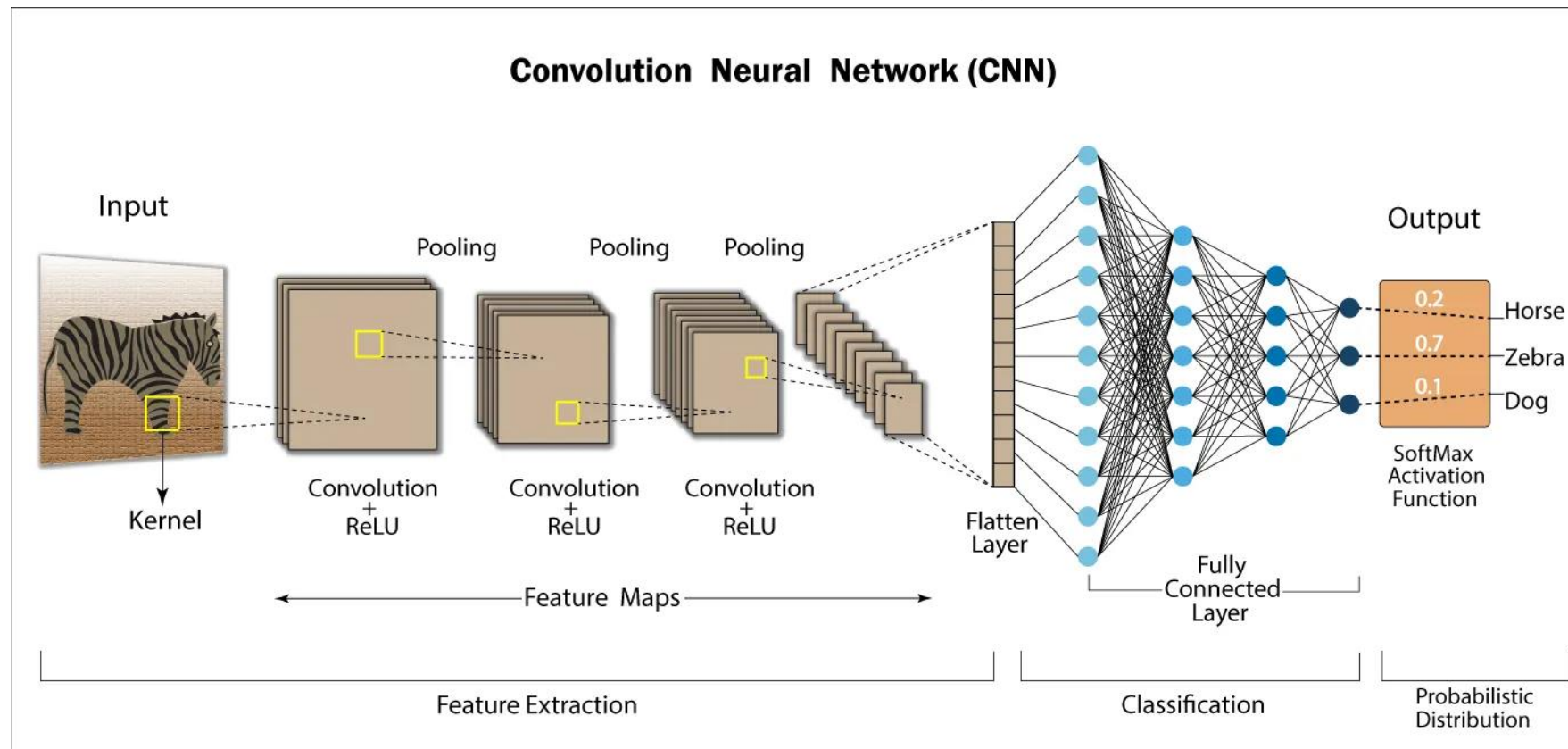
Dengan menggunakan data pelatihan ini, jaringan kemudian dapat mulai memahami fitur spesifik objek dan mengaitkannya dengan kategori yang sesuai.

Setiap lapisan dalam jaringan mengambil data dari lapisan sebelumnya, mengubahnya, dan meneruskannya. Jaringan meningkatkan kompleksitas dan detail dari apa yang dipelajari dari lapisan ke lapisan.

Perhatikan bahwa jaringan belajar langsung dari data—kita tidak terlibat memilih fitur apa yang sedang dipelajari.

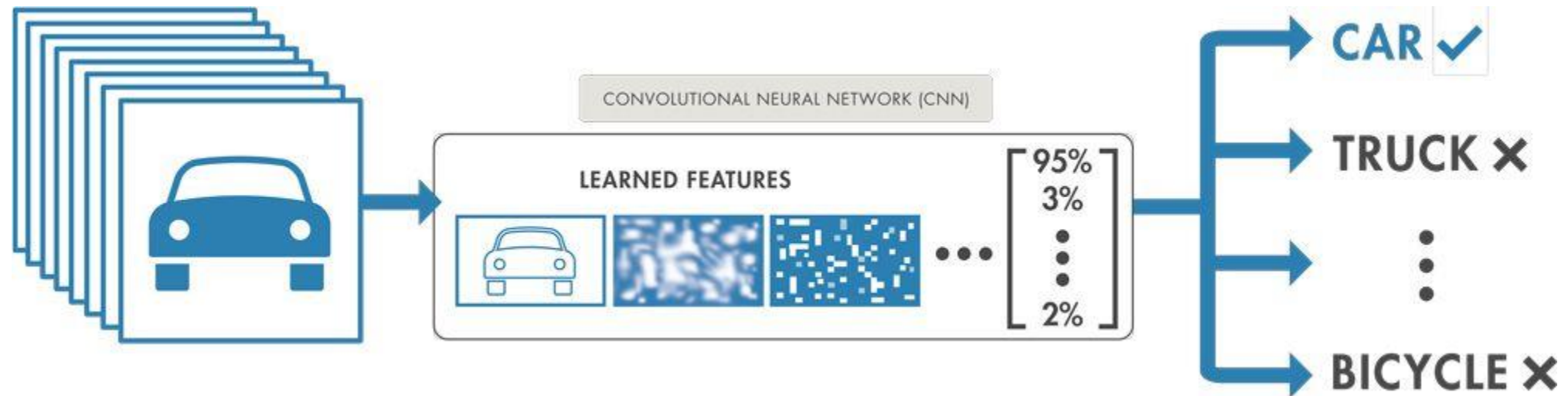
# Convolutional Neural Network

- *Convolutional Neural Network* (CNN atau ConvNet) adalah algoritma pembelajaran mendalam yang populer, umumnya digunakan untuk menganalisis citra seperti pengenalan objek, klasifikasi objek, dll.

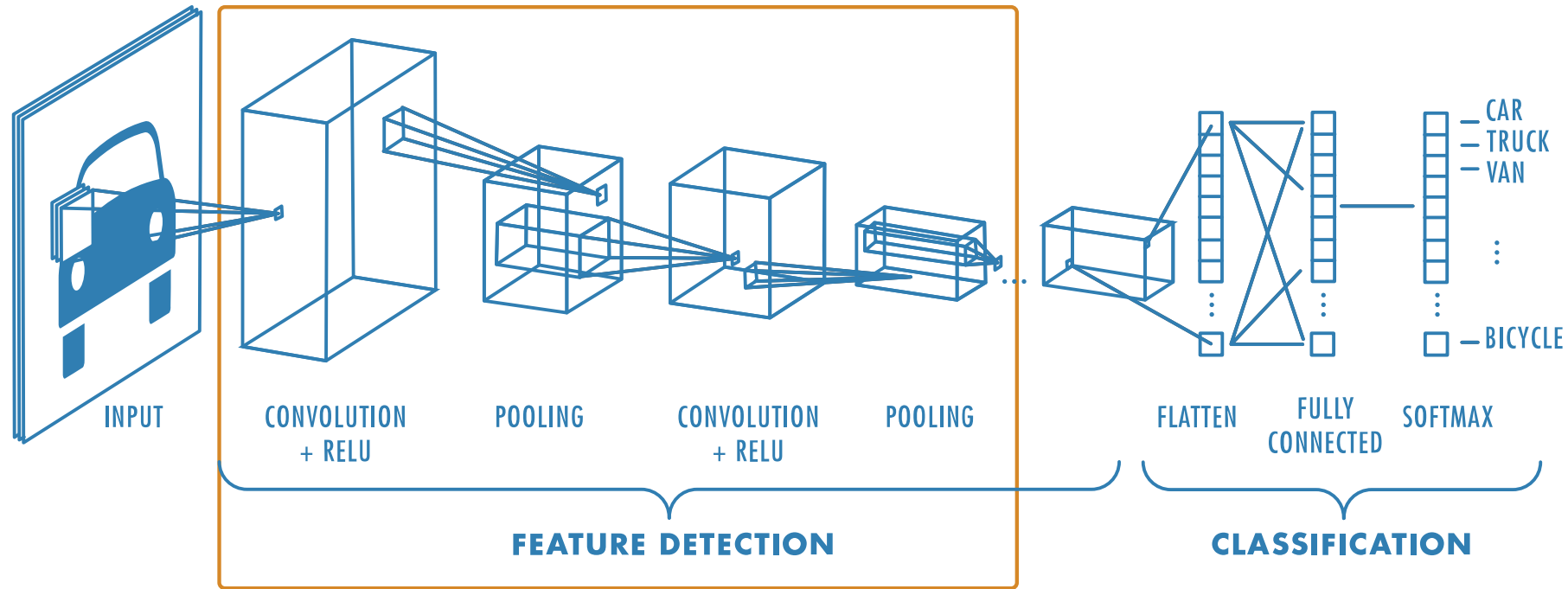


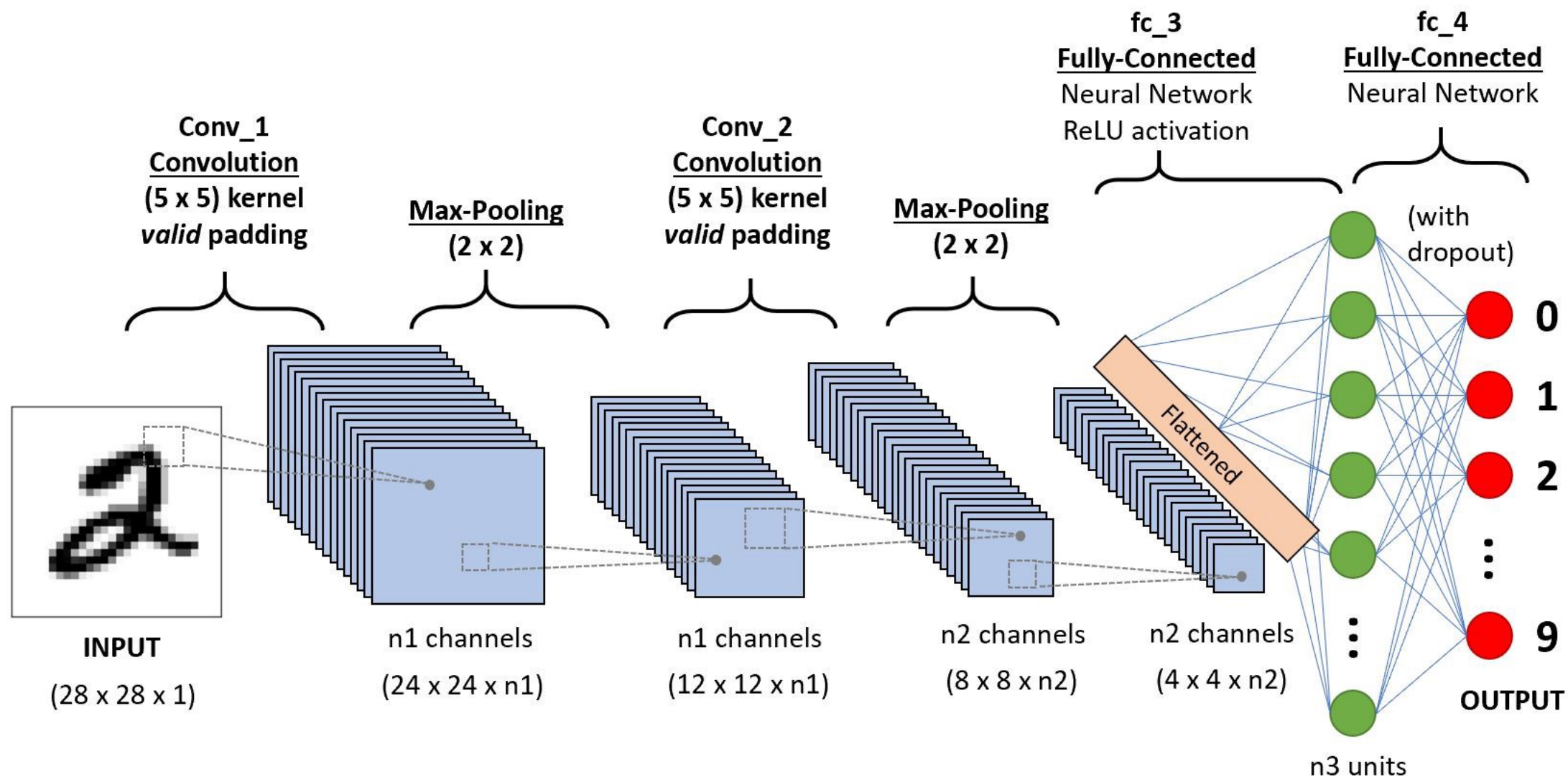
Sumber gambar: <https://developersbreach.com/convolution-neural-network-deep-learning/>

- CNN merupakan arsitektur jaringan untuk pembelajaran mendalam yang belajar langsung dari data, dengan menghilangkan kebutuhan untuk melakukan ekstraksi fitur secara manual



- CNN dapat disebut juga jaringan syaraf tiruan yang melibatkan konvolusi (CNN = ANN + convolution)

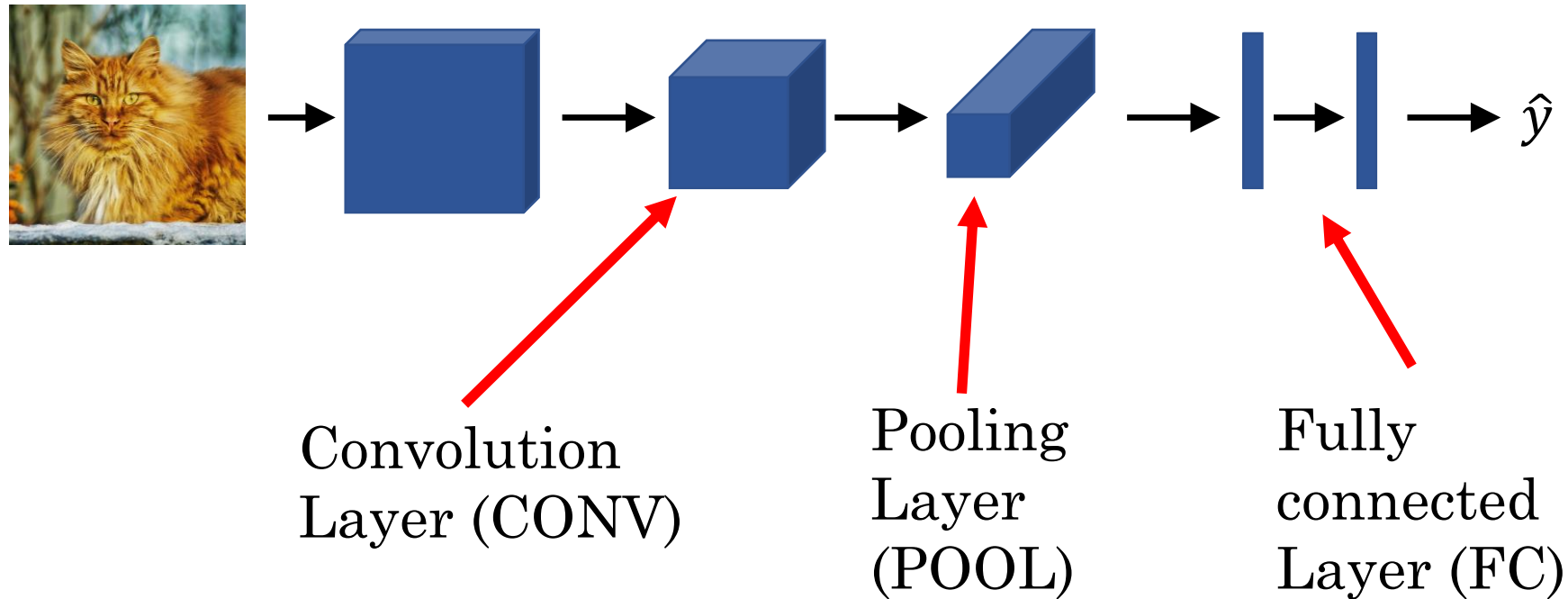




Sumber gambar: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

- *CNN* terdiri dari 3 lapisan utama:
  1. *Convolutional Layer (+ReLU)*,
  2. *Pooling Layer*
  3. *Fully-Connected Layer*.

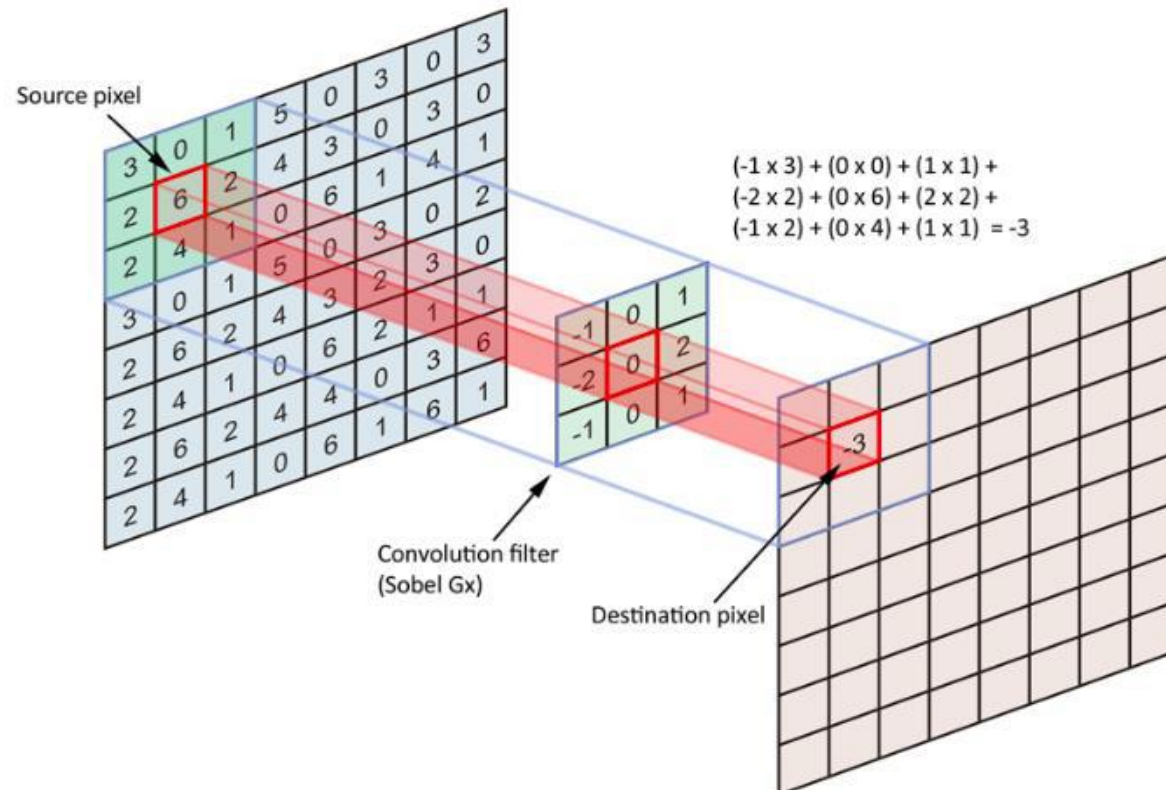
Training set





# Convolutional Layer

- Melakukan operasi konvolusi pada citra masukan dengan sejumlah penapis. Tiap penapis menghasilkan luaran yang disebut *feature map*





Konvolusi pada setiap kanal warna di dalam citra RGB:

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



308

+

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-498

+

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164




+ 1 = -25



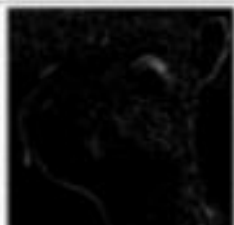
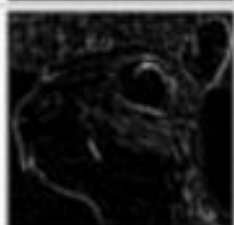
Bias = 1

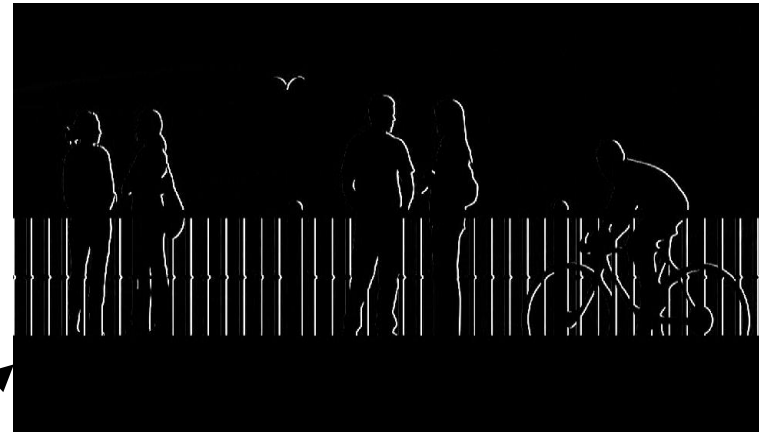
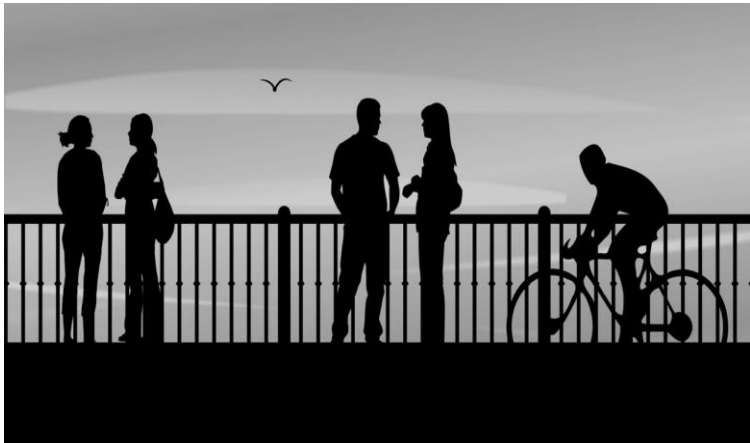
-25				...
				...
				...
				...
...	...	...	...	...

## Convolution Operator

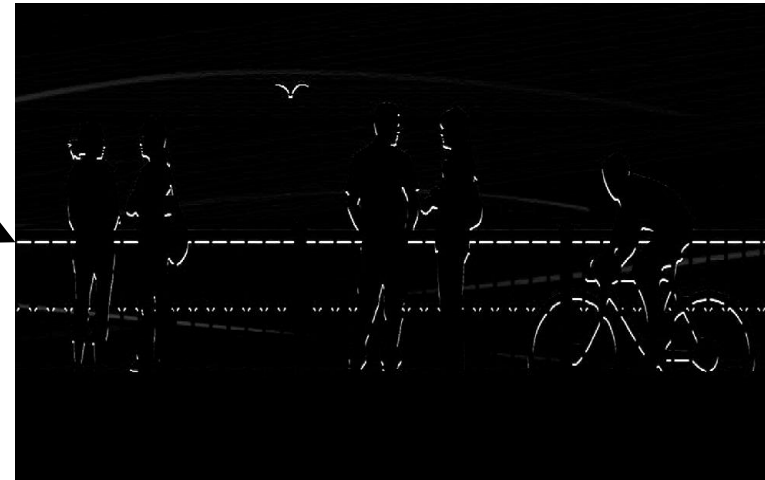
• Different filters will produce different **Feature Maps** for the same input image. For example:

Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	



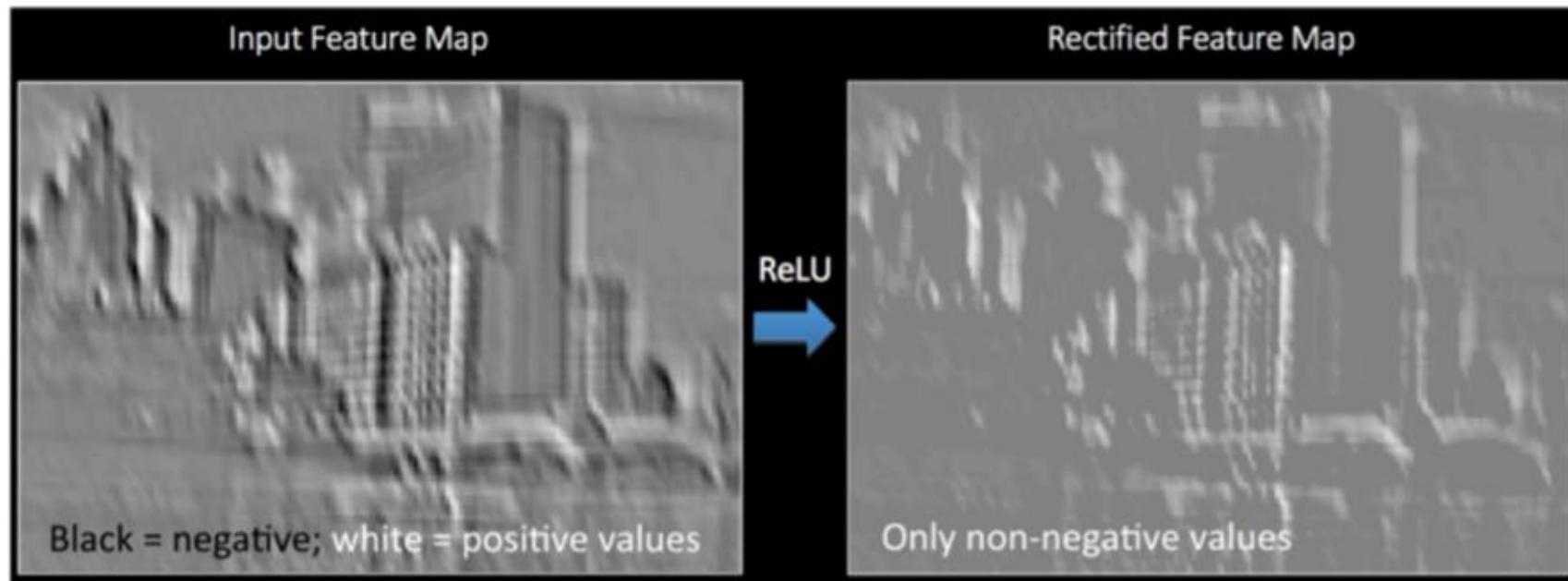
vertical edges



horizontal edges

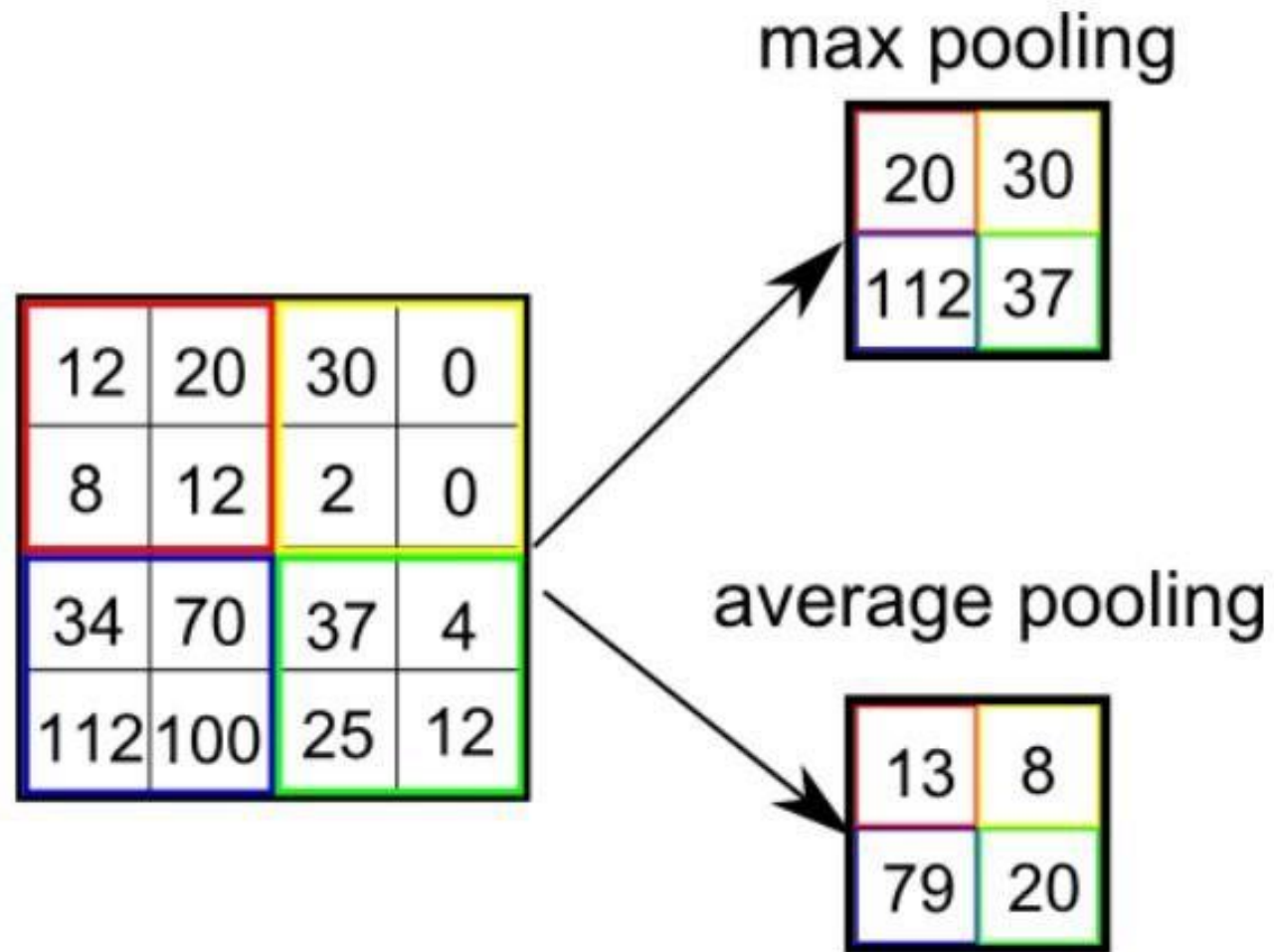
# Rectified Linear Unit (ReLU)

- ReLU adalah layer tambahan yang memungkinkan pelatihan yang lebih cepat dan efektif dengan memetakan nilai negatif ke nol dan mempertahankan nilai positif.
- Pada dasarnya ReLU adalah operasi per-pixel dengan cara mengganti nilai negatif pixel di dalam *feature map* menjadi nol.



# Pooling Layer

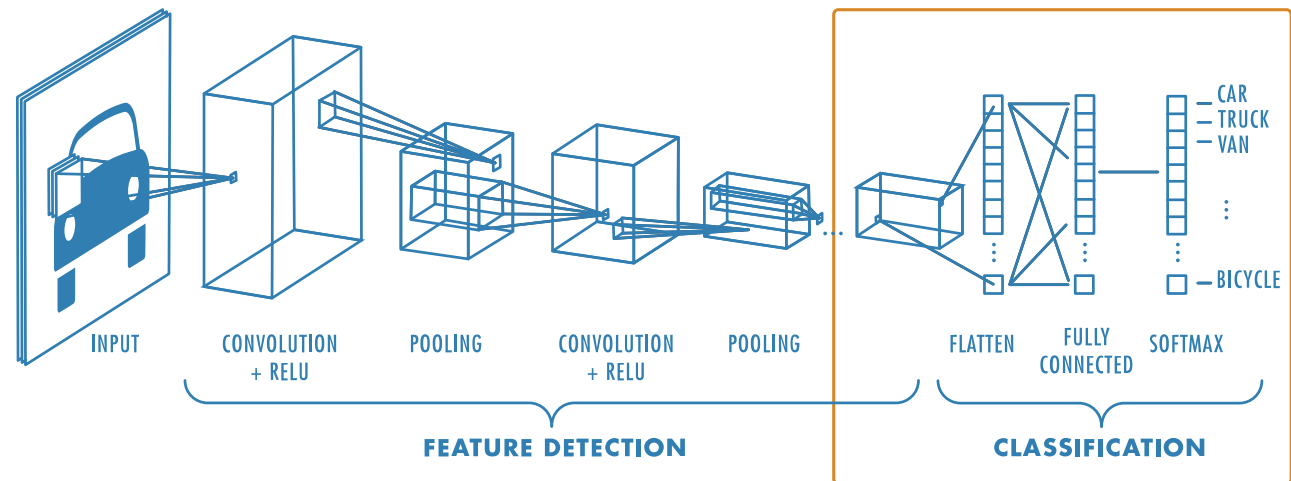
- Mirip dengan *Convolutional Layer*, *Pooling layer* bertanggung jawab untuk mengurangi ukuran spasial dari matriks fitur hasil konvolusi.
- Hal ini bertujuan untuk mengurangi daya komputasi yang diperlukan untuk memproses data melalui pengurangan dimensi
- Ada dua jenis *pooling*: *Max Pooling* dan *Average Pooling*.
  - 1) *Max Pooling* mengembalikan nilai maksimum dari bagian gambar yang dicakup oleh kernel.
  - 2) *Average Pooling* mengembalikan rata-rata semua nilai dari bagian gambar yang dicakup oleh Kernel.



Sumber gambar: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

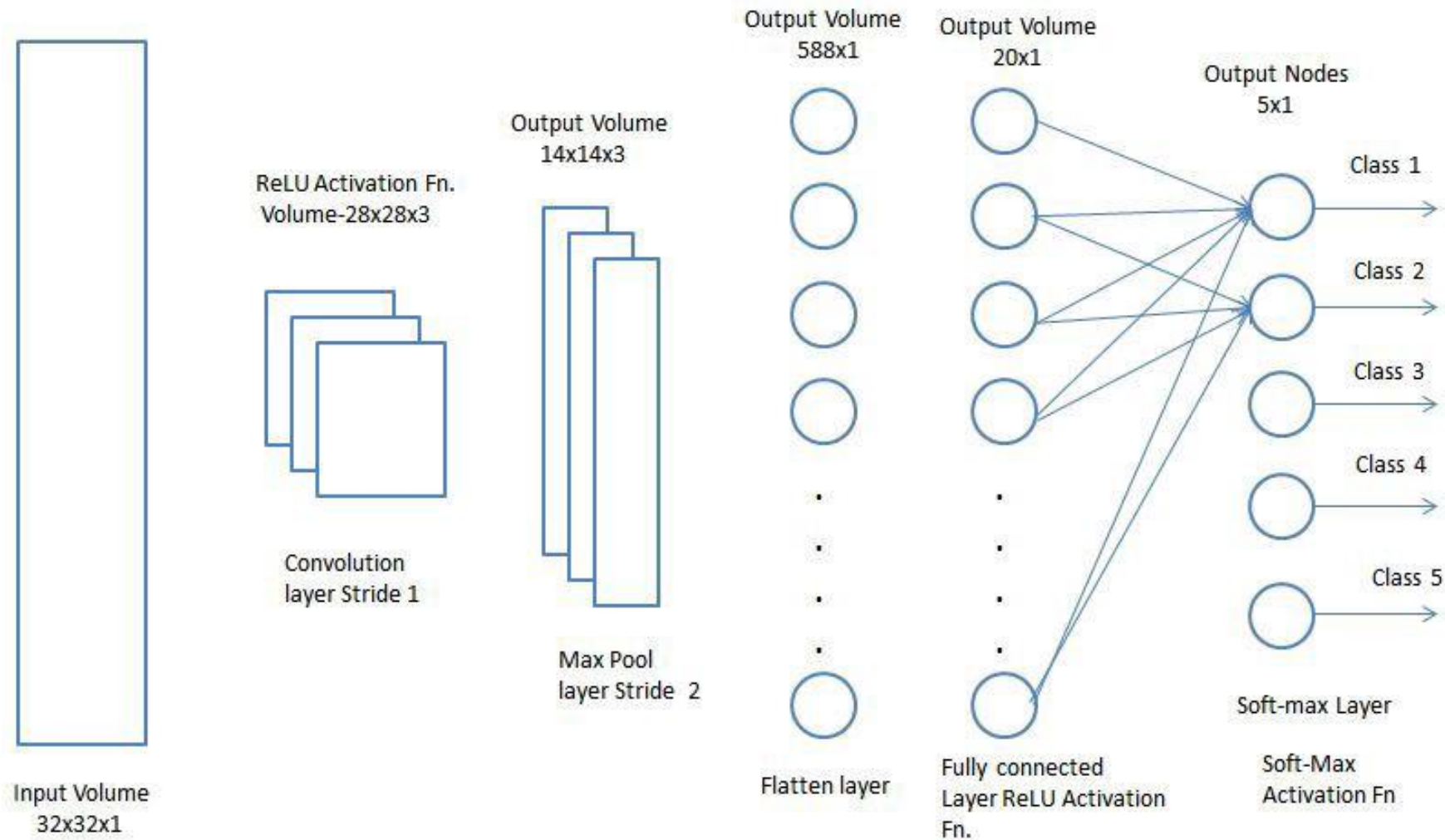
# Fully Connected Layer

- Setelah deteksi fitur, arsitektur CNN beralih ke klasifikasi.
- Lapisan terakhir di dalam CNN adalah *fully connected layer* (FC) yang menghasilkan vektor dimensi K, dalam hal ini K adalah jumlah kelas yang dapat diprediksi oleh jaringan. Vektor ini berisi probabilitas untuk setiap kelas dari setiap gambar yang diklasifikasikan.
- Lapisan terakhir dari arsitektur CNN menggunakan fungsi *softmax* untuk menyediakan luaran klasifikasi.



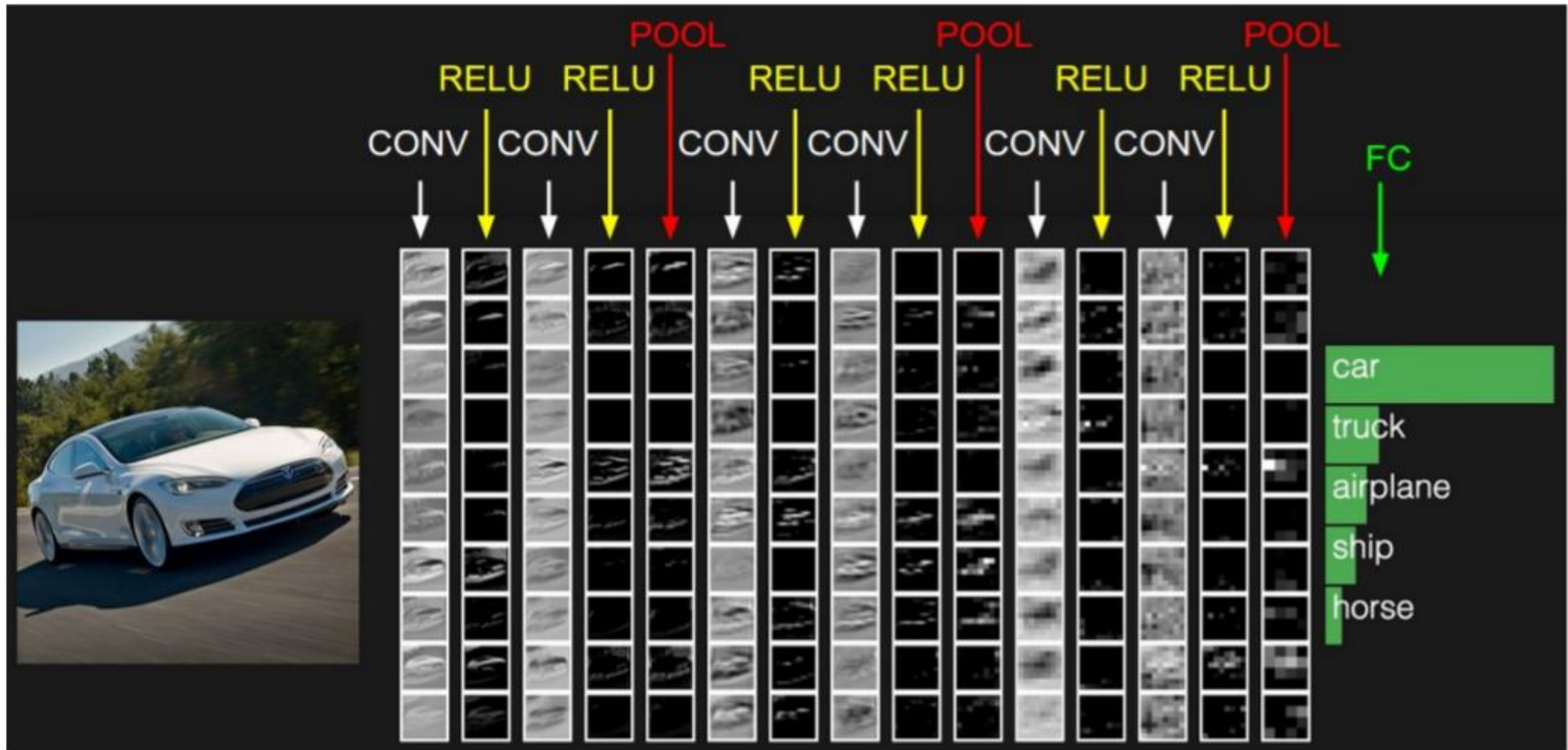


# Classification - Fully Connected Layer





**Example:** Input >> [ [ Conv >> ReLU ] \* 2 >> Pool ] \* 3 >> FC

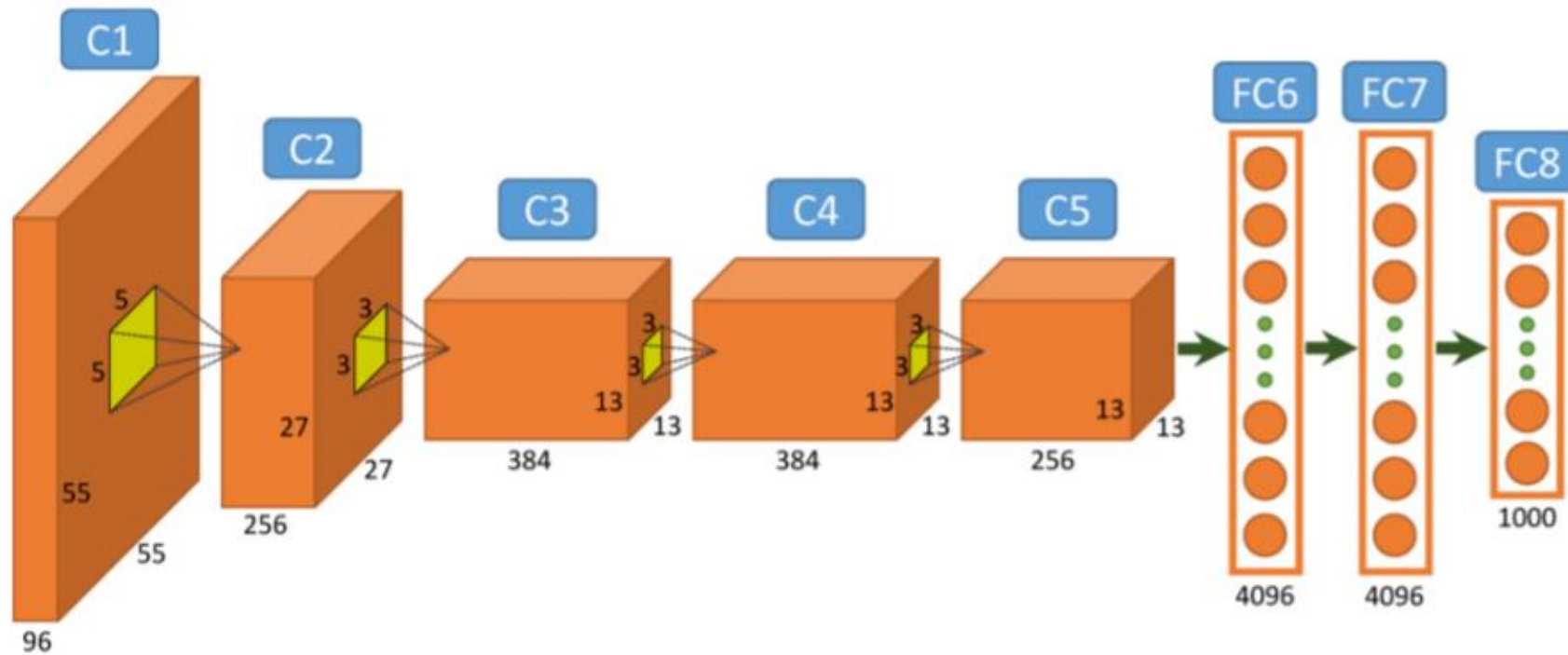


Beberapa arsitektur CNN:

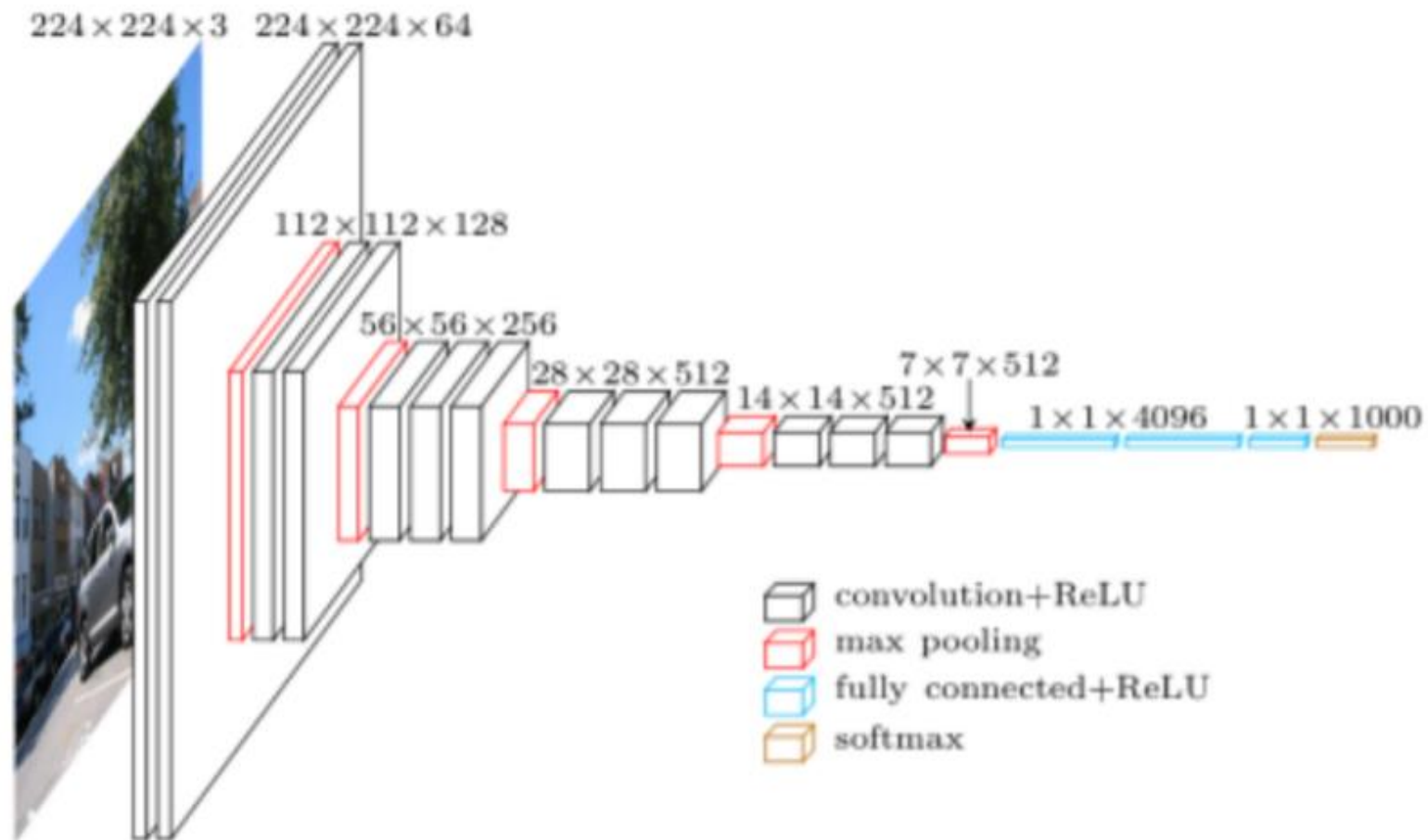
1. LeNet
2. AlexNet
3. VGGNet
4. GoogLeNet
5. ResNet
6. ZFNet
7. dll

# AlexNet

- Famous for winning the **ImageNet** Large Scale Visual Recognition Challenge (**ILSVRC**) in 2012

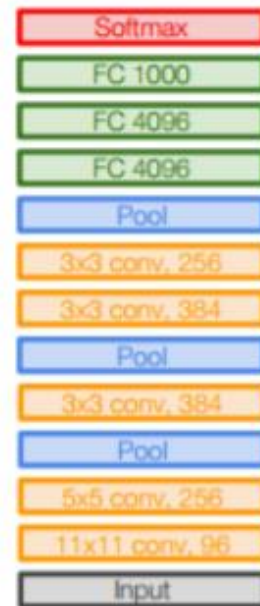


# VGGNet

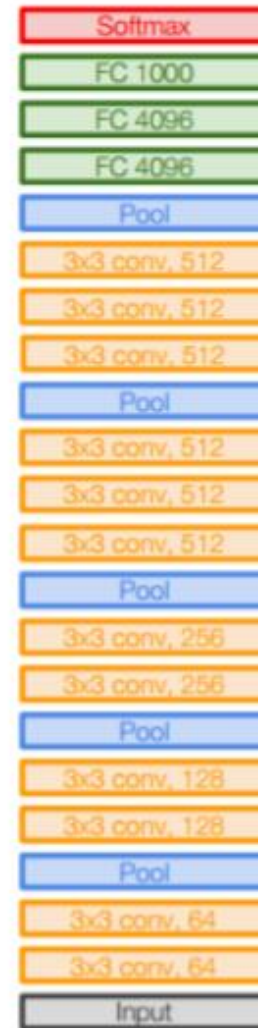


Sumber gambar: Bahan kuliah Deep Learning

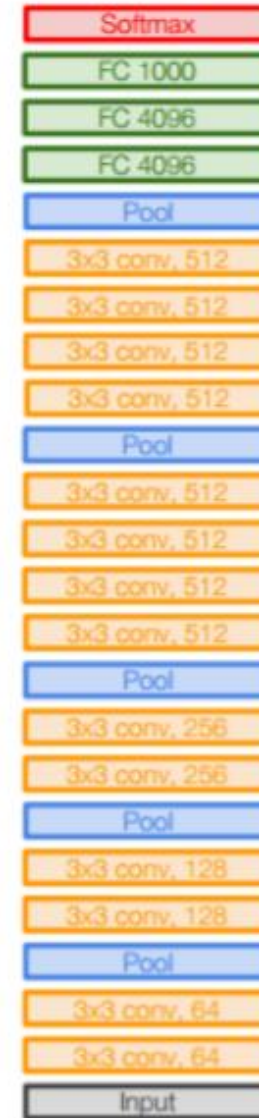
# AlexNet vs. VGGNet (16 and 19)



AlexNet

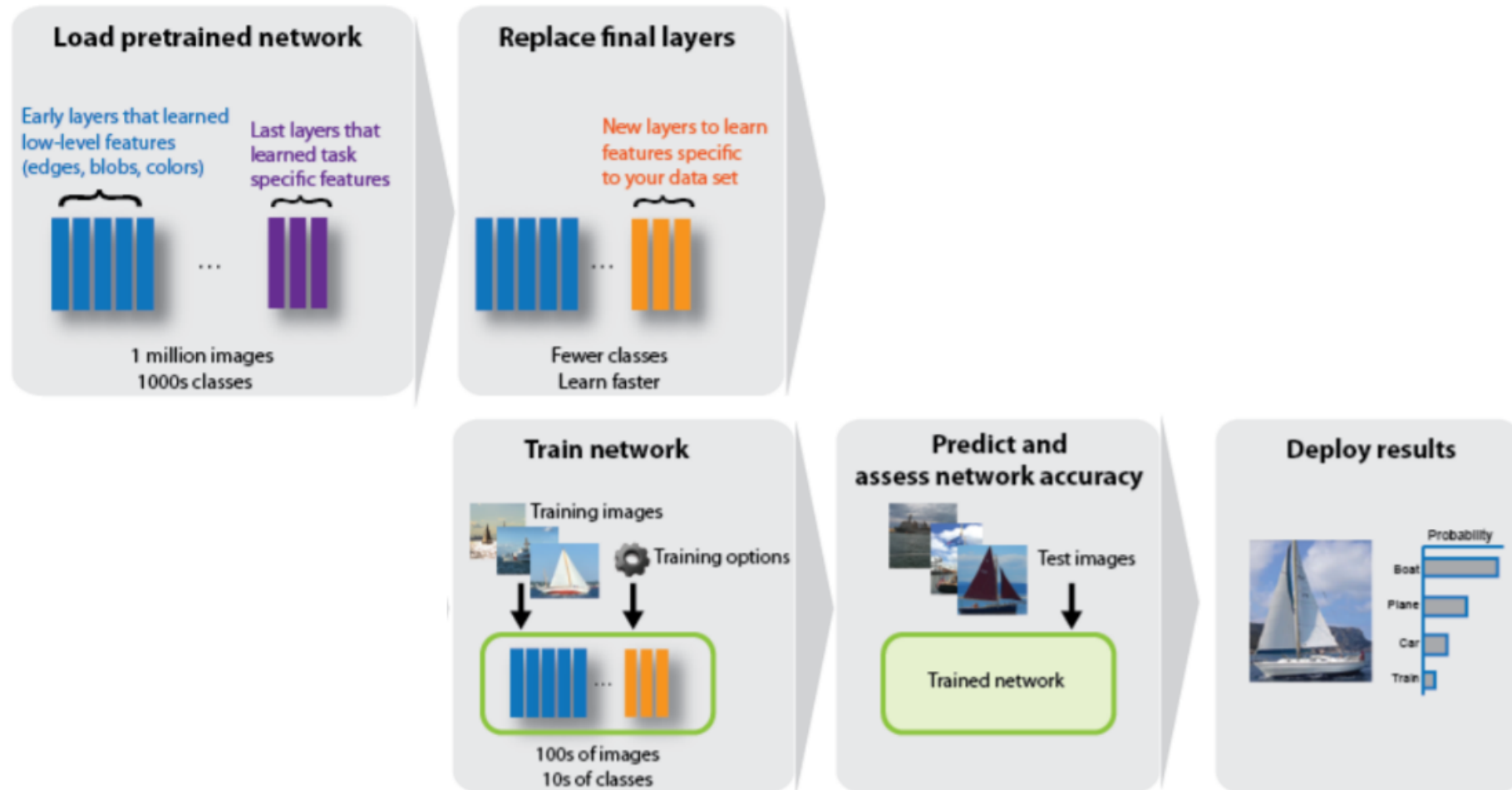


VGG16



VGG19

- Most pre-trained models used the ImageNet Dataset (1 Million of images and 1,000 Classes)





# Contoh program CNN menggunakan Matlab

A. Goal: Mengklasifikasi angka-angka (0-9) berupa citra tulisan tangan

## 1. Load and Explore Image Data

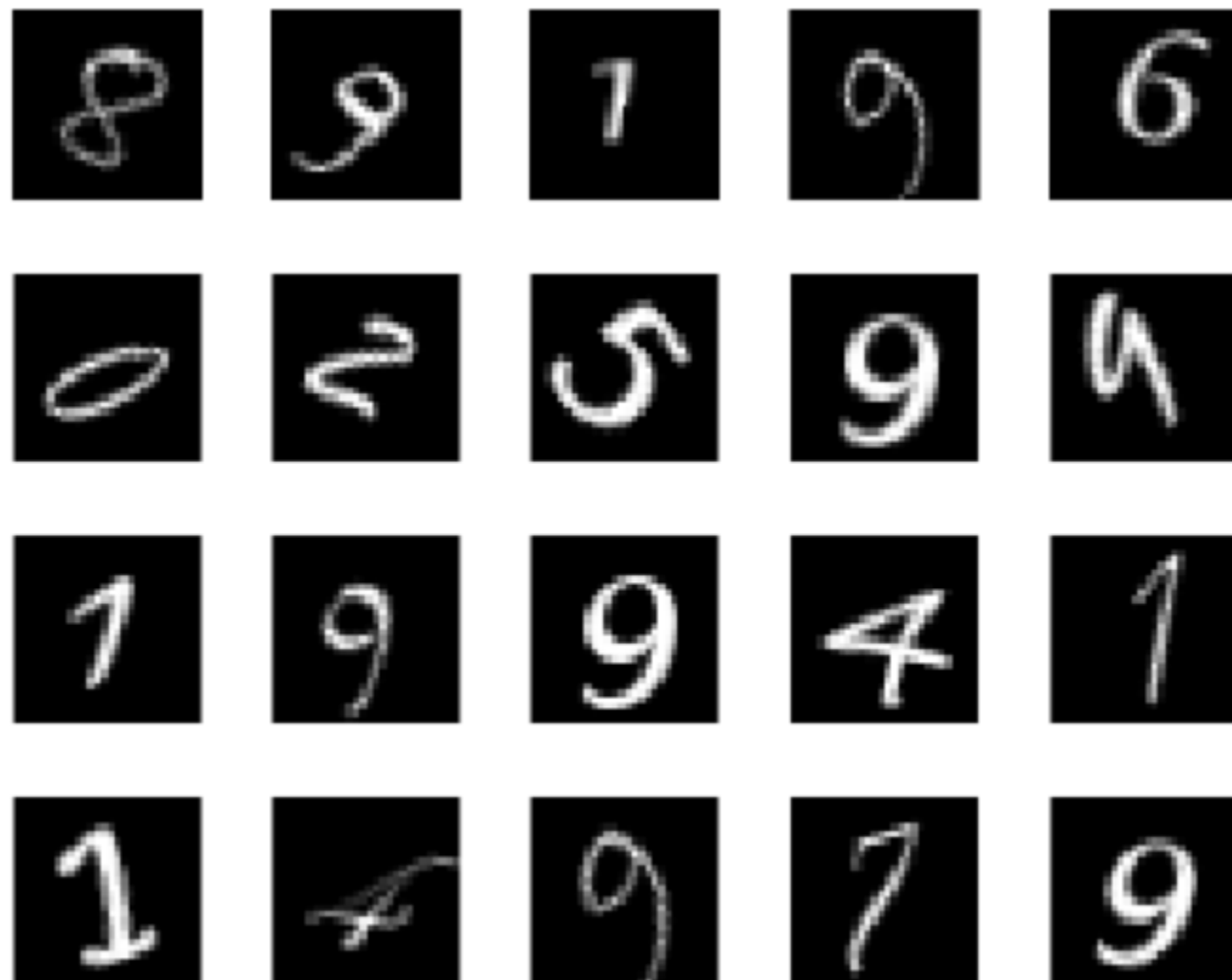
### Load and Explore Image Data

Load the digit sample data as an image datastore. `imageDatastore` automatically labels the images based on folder names and stores the data as an `ImageDatastore` object. An image datastore enables you to store large image data, including data that does not fit in memory, and efficiently read batches of images during training of a convolutional neural network.

```
digitDatasetPath = fullfile(matlabroot, 'toolbox', 'nnet', 'nndemos', ...  
    'nndatasets', 'DigitDataset');  
imds = imageDatastore(digitDatasetPath, ...  
    'IncludeSubfolders', true, 'LabelSource', 'foldernames');
```

Display some of the images in the datastore.

```
figure;  
perm = randperm(10000,20);  
for i = 1:20  
    subplot(4,5,i);  
    imshow(imds.Files{perm(i)});  
end
```





Calculate the number of images in each category. `labelCount` is a table that contains the labels and the number of images having each label. The datastore contains 1000 images for each of the digits 0-9, for a total of 10000 images. You can specify the number of classes in the last fully connected layer of your network as the `OutputSize` argument.

```
labelCount = countEachLabel(imds)
```

`labelCount=10x2 table`

Label	Count
0	1000
1	1000
2	1000
3	1000
4	1000
5	1000
6	1000
7	1000
8	1000
9	1000

You must specify the size of the images in the input layer of the network. Check the size of the first image in `digitData`. Each image is 28-by-28-by-1 pixels.

```
img = readimage(imds,1);  
size(img)
```

```
ans = 1×2
```

```
28    28
```

## 2. Specify Training and Validation Sets

Divide the data into training and validation data sets, so that each category in the training set contains 750 images, and the validation set contains the remaining images from each label. `splitEachLabel` splits the datastore `digitData` into two new datastores, `trainDigitData` and `valDigitData`.

```
numTrainFiles = 750;  
[imdsTrain,imdsValidation] = splitEachLabel(imds,numTrainFiles,'randomize');
```

### 3. Define Network Architecture

Define the convolutional neural network architecture.

```
layers = [  
    imageInputLayer([28 28 1])  
  
    convolution2dLayer(3,8,'Padding','same')  
    batchNormalizationLayer  
    reluLayer  
  
    maxPooling2dLayer(2,'Stride',2)  
  
    convolution2dLayer(3,16,'Padding','same')  
    batchNormalizationLayer  
    reluLayer  
  
    maxPooling2dLayer(2,'Stride',2)  
  
    convolution2dLayer(3,32,'Padding','same')  
    batchNormalizationLayer  
    reluLayer  
  
    fullyConnectedLayer(10)  
    softmaxLayer  
    classificationLayer];
```

**Image Input Layer** An `imageInputLayer` is where you specify the image size, which, in this case, is 28-by-28-by-1. These numbers correspond to the height, width, and the channel size. The digit data consists of grayscale images, so the channel size (color channel) is 1. For a color image, the channel size is 3, corresponding to the RGB values. You do not need to shuffle the data because `trainNetwork`, by default, shuffles the data at the beginning of training. `trainNetwork` can also automatically shuffle the data at the beginning of every epoch during training.

**Convolutional Layer** In the convolutional layer, the first argument is `filterSize`, which is the height and width of the filters the training function uses while scanning along the images. In this example, the number 3 indicates that the filter size is 3-by-3. You can specify different sizes for the height and width of the filter. The second argument is the number of filters, `numFilters`, which is the number of neurons that connect to the same region of the input. This parameter determines the number of feature maps. Use the 'Padding' name-value pair to add padding to the input feature map. For a convolutional layer with a default stride of 1, 'same' padding ensures that the spatial output size is the same as the input size. You can also define the stride and learning rates for this layer using name-value pair arguments of `convolution2dLayer`.

**Batch Normalization Layer** Batch normalization layers normalize the activations and gradients propagating through a network, making network training an easier optimization problem. Use batch normalization layers between convolutional layers and nonlinearities, such as ReLU layers, to speed up network training and reduce the sensitivity to network initialization. Use `batchNormalizationLayer` to create a batch normalization layer.

**ReLU Layer** The batch normalization layer is followed by a nonlinear activation function. The most common activation function is the rectified linear unit (ReLU). Use [reluLayer](#) to create a ReLU layer.

**Max Pooling Layer** Convolutional layers (with activation functions) are sometimes followed by a down-sampling operation that reduces the spatial size of the feature map and removes redundant spatial information. Down-sampling makes it possible to increase the number of filters in deeper convolutional layers without increasing the required amount of computation per layer. One way of down-sampling is using a max pooling, which you create using [maxPooling2dLayer](#). The max pooling layer returns the maximum values of rectangular regions of inputs, specified by the first argument, `poolSize`. In this example, the size of the rectangular region is `[2,2]`. The 'Stride' name-value pair argument specifies the step size that the training function takes as it scans along the input.

**Fully Connected Layer** The convolutional and down-sampling layers are followed by one or more fully connected layers. As its name suggests, a fully connected layer is a layer in which the neurons connect to all the neurons in the preceding layer. This layer combines all the features learned by the previous layers across the image to identify the larger patterns. The last fully connected layer combines the features to classify the images. Therefore, the `OutputSize` parameter in the last fully connected layer is equal to the number of classes in the target data. In this example, the output size is 10, corresponding to the 10 classes. Use [fullyConnectedLayer](#) to create a fully connected layer.

**Softmax Layer** The softmax activation function normalizes the output of the fully connected layer. The output of the softmax layer consists of positive numbers that sum to one, which can then be used as classification probabilities by the classification layer. Create a softmax layer using the [softmaxLayer](#) function after the last fully connected layer.

**Classification Layer** The final layer is the classification layer. This layer uses the probabilities returned by the softmax activation function for each input to assign the input to one of the mutually exclusive classes and compute the loss. To create a classification layer, use [classificationLayer](#).

## 4. Specify Training Options

After defining the network structure, specify the training options. Train the network using stochastic gradient descent with momentum (SGDM) with an initial learning rate of 0.01. Set the maximum number of epochs to 4. An epoch is a full training cycle on the entire training data set. Monitor the network accuracy during training by specifying validation data and validation frequency. Shuffle the data every epoch. The software trains the network on the training data and calculates the accuracy on the validation data at regular intervals during training. The validation data is not used to update the network weights. Turn on the training progress plot, and turn off the command window output.

```
options = trainingOptions('sgdm', ...  
    'InitialLearnRate',0.01, ...  
    'MaxEpochs',4, ...  
    'Shuffle','every-epoch', ...  
    'ValidationData',imdsValidation, ...  
    'ValidationFrequency',30, ...  
    'Verbose',false, ...  
    'Plots','training-progress');
```

## 5. Train Network Using Training Data

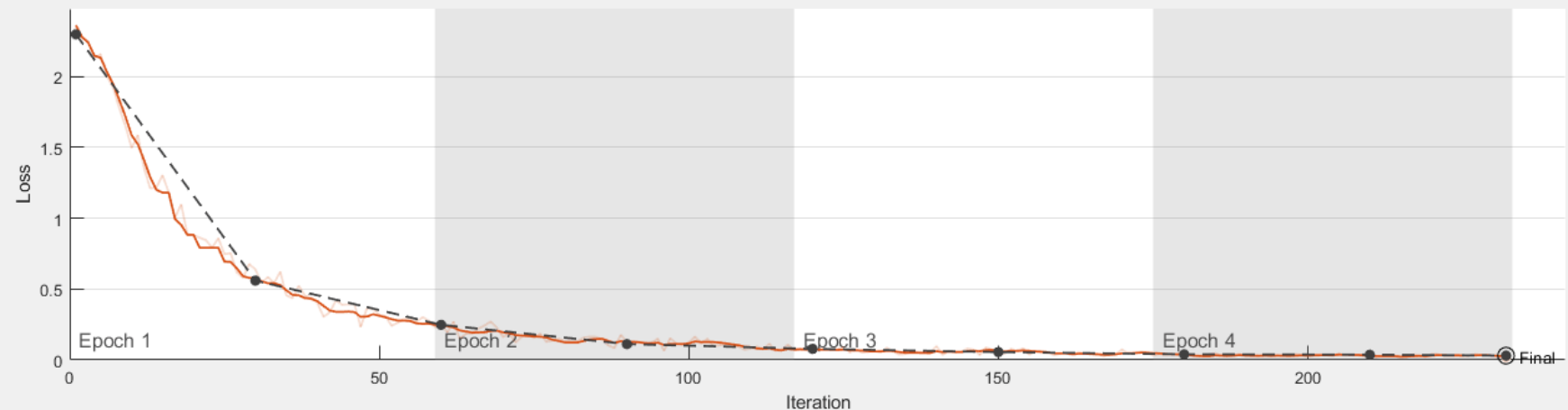
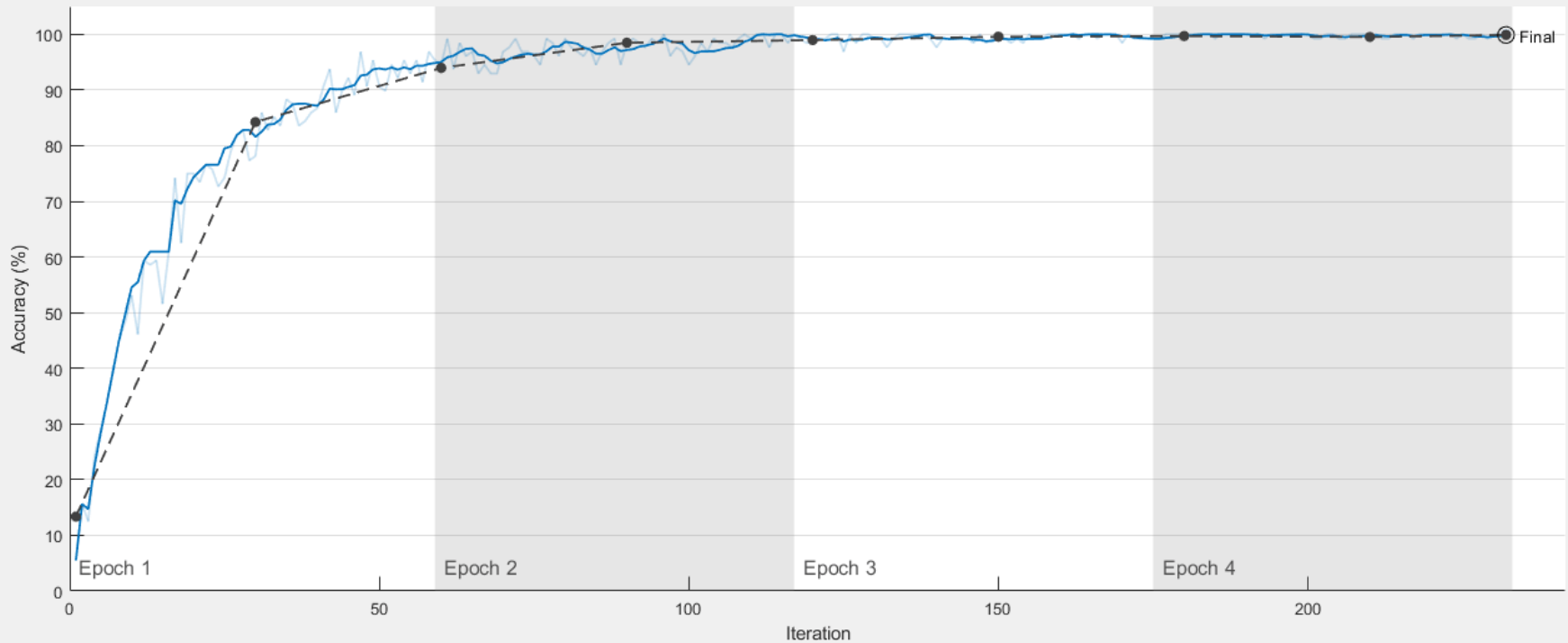
Train the network using the architecture defined by layers, the training data, and the training options. By default, `trainNetwork` uses a GPU if one is available, otherwise, it uses a CPU. Training on a GPU requires Parallel Computing Toolbox™ and a supported GPU device. For information on supported devices, see GPU Computing Requirements (Parallel Computing Toolbox). You can also specify the execution environment by using the 'ExecutionEnvironment' name-value pair argument of `trainingOptions`.

The training progress plot shows the mini-batch loss and accuracy and the validation loss and accuracy. For more information on the training progress plot, see Monitor Deep Learning Training Progress. The loss is the cross-entropy loss. The accuracy is the percentage of images that the network classifies correctly.

```
net = trainNetwork(imdsTrain, layers, options);
```



# Training Progress (13-Jul-2018 13:29:26)



## Results

Validation accuracy: 99.88%  
 Training finished: Reached final iteration

## Training Time

Start time: 13-Jul-2018 13:29:26  
 Elapsed time: 36 sec

## Training Cycle

Epoch: 4 of 4  
 Iteration: 232 of 232  
 Iterations per epoch: 58  
 Maximum iterations: 232

## Validation

Frequency: 30 iterations  
 Patience: Inf

## Other Information

Hardware resource: Single GPU  
 Learning rate schedule: Constant  
 Learning rate: 0.01

[Learn more](#)

## Accuracy

— Training (smoothed)  
 —●— Training  
 - - ● - - Validation

## Loss

— Training (smoothed)  
 —●— Training  
 - - ● - - Validation

## 6. Classify Validation Images and Compute Accuracy

Predict the labels of the validation data using the trained network, and calculate the final validation accuracy. Accuracy is the fraction of labels that the network predicts correctly. In this case, more than 99% of the predicted labels match the true labels of the validation set.

```
YPred = classify(net,imdsValidation);  
YValidation = imdsValidation.Labels;  
  
accuracy = sum(YPred == YValidation)/numel(YValidation)
```

```
accuracy = 0.9988
```

B. Use an existing network, such as GoogLeNet, a CNN trained on more than a million images. GoogLeNet is most commonly used for image classification. It can classify images into 1000 different categories, including keyboards, computer mice, pencils, and other office equipment, as well as various breeds of dogs, cats, horses, and other animals.

## An Example Using GoogLeNet

You can use GoogLeNet to classify objects in any image. In this example, we'll use it to classify objects in an image from a webcam installed on a desktop. In addition to MATLAB®, we'll be using the following:

- Deep Learning Toolbox™
- Support package for [using webcams in MATLAB](#)
- Support package for using [GoogLeNet](#)

After loading GoogLeNet we connect to the webcam and capture a live image.

```
camera = webcam;           % Connect to the camera
nnet = googlenet;          % Load the neural net
picture = camera.snapshot; % Take a picture
```

Next, we resize the image to 224x224 pixels, the size required by GoogLeNet.

```
picture = imresize(picture,[224,224]); % Resize the picture
```

GoogLeNet can now classify our image.

```
label = classify(nnet, picture); % Classify the picture  
image(picture);                 % Show the picture  
title(char(label));             % Show the label
```

coffee mug

