

Week 5 – Convolutional Neural Network and Image Classification

ELEC0144 Machine Learning for Robotics

Dr. Chow Yin Lai

Email: uceecyl@ucl.ac.uk

Schedule

Week	Lecture	Workshop	Assignment Deadlines
1	Introduction; Image Processing	Image Processing	
2	Camera and Robot Calibration	Camera and Robot Calibration	
3	Introduction to Neural Networks	Camera and Robot Calibration	Friday: Camera and Robot Calibration
4	MLP and Backpropagation	MLP and Backpropagation	
5	CNN and Image Classification	MLP and Backpropagation	
6	Object Detection	MLP and Backpropagation	Friday: MLP and Backpropagation
7	Path Planning	Path Planning	
8	Kalman Filter SLAM	Path Planning	
9	Extended Kalman Filter SLAM	Path Planning	
10	Particle Filter SLAM	Path Planning	Friday: Path Planning

Content

- Introduction to Image Classification
- Commonly-Used Datasets
- MLP for Image Classification
- History of Convolutional Neural Network (CNN)
- CNN in Details
- MATLAB & Python Examples
- Examples of CNN Architectures
- Transfer Learning

Content

- Introduction to Image Classification
- Commonly-Used Datasets
- MLP for Image Classification
- History of Convolutional Neural Network (CNN)
- CNN in Details
- MATLAB & Python Examples
- Examples of CNN Architectures
- Transfer Learning

Image Classification (1)

- Image classification is one of the most important tasks in computer vision.
- Given an image, **classify the (foreground) object** into the correct category:



[This Photo](#) by Unknown Author is licensed under [CC BY](#)



Cat
Dog
Car
Truck

Image Classification (2)

- Many useful applications! Examples:
 - Classifying between benign and malignant [tumor](#).
 - Classifying between different types of animals, and even different [breeds / sub-species](#) with a species.



[This Photo](#) by Unknown Author is licensed under [CC BY](#)



[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)



[This Photo](#) by Unknown Author is licensed under [CC BY](#)

Challenges in Image Classification (1)

- **Variation** in position and/or orientation of the object in image:



Challenges in Image Classification (2)

- Background clutter



Challenges in Image Classification (3)

- Illumination



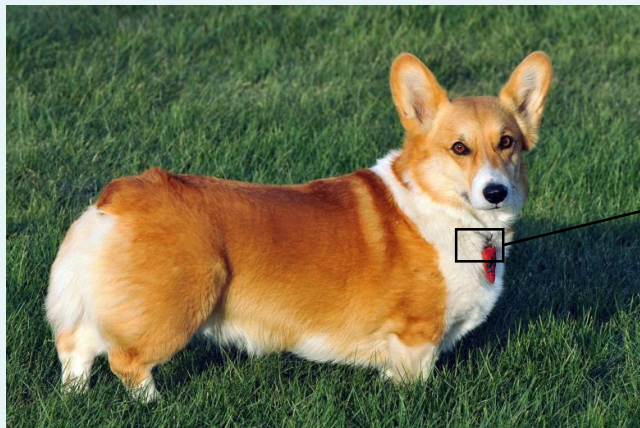
Challenges in Image Classification (4)

- Occlusion



Challenges in Image Classification (5)

- Unlike humans, computers only see a **bunch of numbers!**
 - Numbers from 0 to 255, possibly in RGB layers.



14	95	94	78	92	95	99	77	105	101
0	73	83	103	74	90	149	72	87	83
0	78	94	86	86	95	108	90	83	117
255	78	62	73	81	89	90	102	83	80
176	22	63	78	65	68	57	82	113	59
54	255	0	40	38	58	61	173	136	55
0	202	255	100	16	45	54	67	36	72
0	79	211	103	12	80	67	79	59	58
0	0	80	76	46	62	143	36	43	60
0	0	0	42	34	102	241	115	95	175
196	22	0	63	43	120	164	209	140	75
255	0	7	54	48	48	75	213	78	28
196	170	12	42	53	46	48	100	38	80
13	255	139	43	35	51	56	140	83	65
0	187	255	84	22	66	52	131	94	64
0	30	202	81	32	77	49	99	68	67
33	0	7	45	50	63	60	71	63	118
0	0	0	84	62	63	77	72	42	67
255	53	0	63	60	55	67	65	66	55
209	33	21	57	52	69	67	77	82	80
255	255	4	19	46	59	68	74	64	69
223	255	255	84	41	65	80	68	74	72
		216	54	39	73	74	72	71	65

Some Methods (1)

- Researchers have proposed many methods for image classification. Examples:
 - By **edge detection**:
 - E.g. after edge detection, determine the angles or roundness of ears.
 - Not robust against variations!



Laplacian Omnidirectional Edge Detection

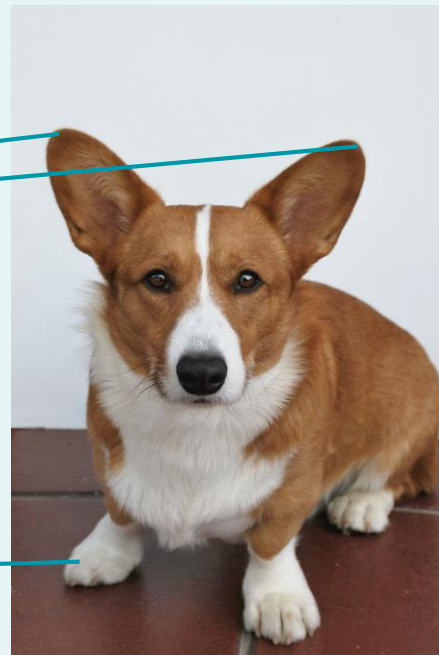


Some Methods (2)

- By **matching**:
 - Match certain features with known objects.



Known:



[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)

Some Methods (3)

- The methods mentioned rely on humans to **hand-engineer** some features.
 - If you recall the lecture from previous week, we classified different iris using **multilayer perceptron**.
 - But there, we also hand-engineered the features i.e. sepal length, sepal width, petal length and petal width.
- Can we just let our machine learning algorithm to **learn from images directly**, without pre-specifying the features?
 - That's the topic for today!

Content

- Introduction to Image Classification
- **Commonly-Used Datasets**
- MLP for Image Classification
- History of Convolutional Neural Network (CNN)
- CNN in Details
- MATLAB & Python Examples
- Examples of CNN Architectures
- Transfer Learning

Datasets (1)

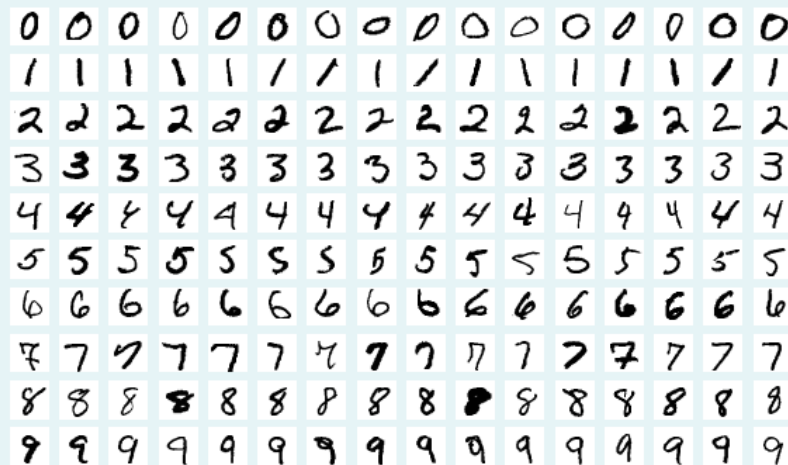
- As you have learnt previously, to train our machine-learning model (MLP, or CNN which you will learn today), you will need **training data** and **validation data**.
- For image classification, the data is obviously in the form of **images**.
 - You could perhaps download many images from Google and label them one-by-one, but that's a lot of work!
 - Fortunately, there are some **well-known datasets** which we can use directly.
 - This will also allow fair comparisons among researchers.

Datasets (2)

- **MNIST** (Modified National Institute of Standards and Technology) dataset:

- **Handwritten digits**

- 10 classes: 0 to 9
- 28 x 28 grayscale images
- 50K training images
- 10K test images
- Error rate is now around 0.17%.
- Not often used because deemed too simple by today's standard.

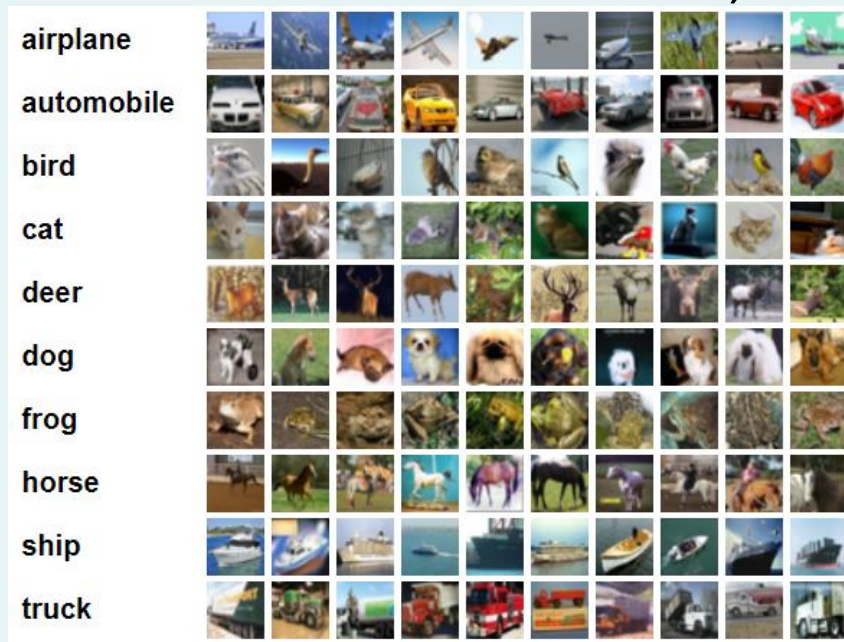


<https://commons.wikimedia.org/wiki/File:MnistExamples.png>

Datasets (3)

- **CIFAR-10** (Canadian Institute for Advanced Research)

- 10 classes
- 32x32 RGB images
- 50K training images
- 10K test images
- Collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.
- The Krizhevsky and Hinton are 2/3 authors of the famous AlexNet!

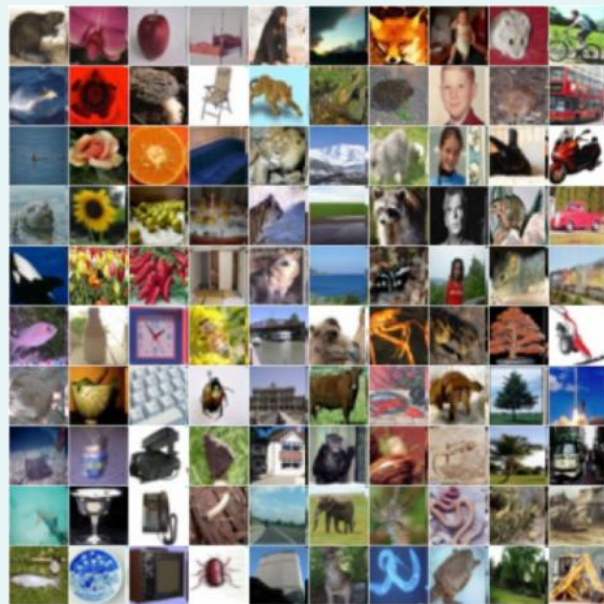


<https://www.cs.toronto.edu/~kriz/cifar.html>

Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

Datasets (4)

- **CIFAR-100** (Canadian Institute for Advanced Research)
 - 100 classes
 - 32x32 RGB images
 - 50K training images
 - 10K test images



https://web.stanford.edu/~hastie/CASI_files/DATA/cifar-100.html

Datasets (5)

- ImageNet

- 1000 classes
- 1.3 million training images
- 50K validation images
- 100K test images
- Images have different sizes but often resized to 256x256.
- A model is considered correct if the **top 5 labels** contain THE correct image.
- **ImageNet challenge** from 2010 to 2017 – propelled the research in image classification, and many famous networks have been created during this time.



<https://medium.com/syncedreview/sensetime-trains-imagenet-alexnet-in-record-1-5-minutes-e944ab049b2c>

Content

- Introduction to Image Classification
- Commonly-Used Datasets
- **MLP for Image Classification**
- History of Convolutional Neural Network (CNN)
- CNN in Details
- MATLAB & Python Examples
- Examples of CNN Architectures
- Transfer Learning

MLP for Image Classification (1)

- Let's start by using the MLP model which you already learnt.
- We will use **CIFAR-10** data as example.
 - First of all, visit: <https://www.cs.toronto.edu/~kriz/cifar.html>
 - Download the Matlab version:

Version

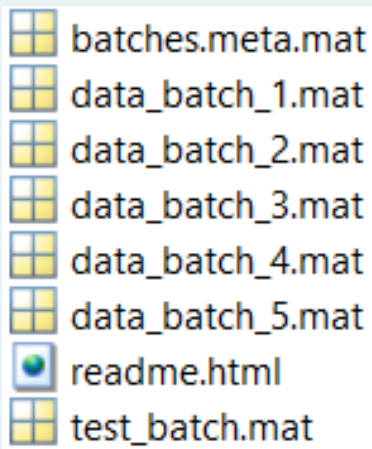
CIFAR-10 python version

CIFAR-10 Matlab version ←

CIFAR-10 binary version (suitable for C programs)
 - The file will be a “cifar-10-matlab.tar.gz” file.

MLP for Image Classification (2)

- To extract the file, start Matlab.
- Key in `untar('cifar-10-matlab')` in the command window.
- It will create a folder called “cifar-10-batches-mat”, with the following contents:



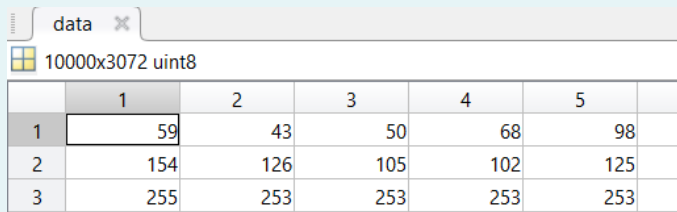
MLP for Image Classification (3)

- Double click “`batches_meta.mat`”, and you will see the label names:

1	airplane
2	automobile
3	bird
4	cat
5	deer
6	dog
7	frog
8	horse
9	ship
10	truck

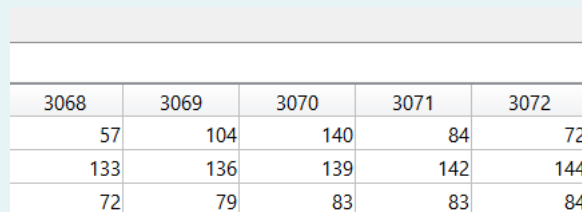
MLP for Image Classification (4)

- Next, double click “`data_batch_1.mat`”, and you will see several items:
- **Data**: 10,000 training data: 10,000 rows, each having an array of size 1x3072.



	1	2	3	4	5
1	59	43	50	68	98
2	154	126	105	102	125
3	255	253	253	253	253

...



3068	3069	3070	3071	3072
57	104	140	84	72
133	136	139	142	144
72	79	83	83	84

⋮

⋮

- **Labels**: 10,000 labels. (Note: The labels are **0-indexed**, i.e. 0 is airplane, 1 is automobile etc. We can **add 1** to avoid confusion)
- Same structure for “`data_batch_2/3/4/5`” and “`test_batch`”.

MLP for Image Classification (5)

- Question: Why does each row have a size of 1x3072?
- It's because we “**flatten**” our RGB 2D image (3x32x32) into a single array!

14	95	94	78	92	95	99	77	105	101										
0	73	78	94	86	86	95	108	90	83	117	108								
0	36	78	73	62	73	81	89	90	102	83	80	93	107						
255	159	22	34	63	78	65	68	57	82	113	59	71	72						
176	125	255	131	0	40	38	58	61	173	136	55	58	68						
54	85	202	123	255	100	16	45	54	67	36	72	87	55						
0	63	79	85	211	103	12	80	67	79	59	58	64	79						
0	69	0	52	80	76	46	62	143	36	43	60	69	48						
0	72	0	59	0	42	34	102	241	115	95	175	0	73						
0	67	22	62	0	63	43	120	164	209	140	75	40	72						
196	91	0	59	7	54	48	48	75	213	78	28	55	69						
255	106	170	70	12	42	53	46	48	100	38	80	72	74						
196	92	255	98	139	43	35	51	56	140	83	65	72	86						
13	49	187	104	255	84	22	66	52	131	94	64	90	90						
0	100	30	46	202	81	32	77	49	99	68	67	77	80						
0	78	0	86	7	45	50	63	60	71	63	118	113	80						
		0	78	0	84	62	63	77	72	42	67	107	165						
				0	63	60	55	67	65	66	55	61	46						

[14,95,94,...,101,0,73,...,78,94,86,...,108,78,73,...,62,73,81,...,107,63,78,...,55,61,46]

MLP for Image Classification (6)

- Matlab code so far: load data

```
%% Settings

numberBatches = 5;
imageRow = 32;
imageColumn = 32;
exampleImage = 5; % car

%% Load Data

addpath('cifar-10-batches-mat')
TrainingData = [];
TrainingLabels = [];

for i = 1:numberBatches
    load(['data_batch_',num2str(i),'.mat'])
    TrainingData = [TrainingData;data];
    TrainingLabels = [TrainingLabels;labels+1]; %+1 because label was zero-indexed
end

load('test_batch.mat')
TestData = data;
TestLabels = labels+1;
```

MLP for Image Classification (7)

- Matlab code so far: Unflatten one image for visualisation

```
%% "Unflatten" one image for visualisation
```

```
Ired = TrainingData(exampleImage,1:imageRow*imageColumn);  
IredUnflatten = reshape(Ired,[imageRow,imageColumn])';  
Igreen = TrainingData(exampleImage,imageRow*imageColumn+1:2*imageRow*imageColumn);  
IgreenUnflatten = reshape(Igreen,[imageRow,imageColumn])';  
Iblue = TrainingData(exampleImage,2*imageRow*imageColumn+1:3*imageRow*imageColumn);  
IblueUnflatten = reshape(Iblue,[imageRow,imageColumn])';  
Iexample(:,:,1) = IredUnflatten;  
Iexample(:,:,2) = IgreenUnflatten;  
Iexample(:,:,3) = IblueUnflatten;  
figure,imshow(Iexample)
```

Image:



Label: 2 (automobile)

MLP for Image Classification (8)

- A few more things to do before we put the data into MLP:
- Scale the data from 0-255 to 0-1.
 - This is because we will have a lot of multiplications and do not want the numbers to explode!
 - Steps: Change from uint8 to double, then divide by 255.
- Encode our labels to one-hot encoding.
 - E.g. airplane 1 \rightarrow [1 0 0 0 0 0 0 0 0 0]
 - E.g. automobile 2 \rightarrow [0 1 0 0 0 0 0 0 0 0]

MLP for Image Classification (9)

- Matlab code so far: Scale data and one-hot encoding:

```
%% Scale 0-255 to 0-1
```

```
TrainingDataScaled = double(TrainingData)/255;  
TestDataScaled = double(TestData)/255;
```

```
%% One-Hot Encoding
```

```
TrainingLabelsEncoded = zeros(length(TrainingLabels),numberClasses);
```

```
for i=1:length(TrainingLabels)  
    TrainingLabelsEncoded(i,TrainingLabels(i)) = 1;  
end
```

```
TestLabelsEncoded = zeros(length(TestLabels),numberClasses);
```

```
for i=1:length(TestLabels)  
    TestLabelsEncoded(i,TestLabels(i)) = 1;  
end
```

TrainingLabels	
50000x1 uint8	
	1
1	7
2	10
3	10
4	5



TrainingLabels

TrainingLabelsEncoded

50000x10 double

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	1	0	0	0
2	0	0	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0	0	1
4	0	0	0	0	1	0	0	0	0	0

MLP for Image Classification (10)

- One more thing to do before we can feed the data to the MLP.
- At the moment, the **data format** is:

→ Pixels in each image

Image 1, 2, 3... ↓

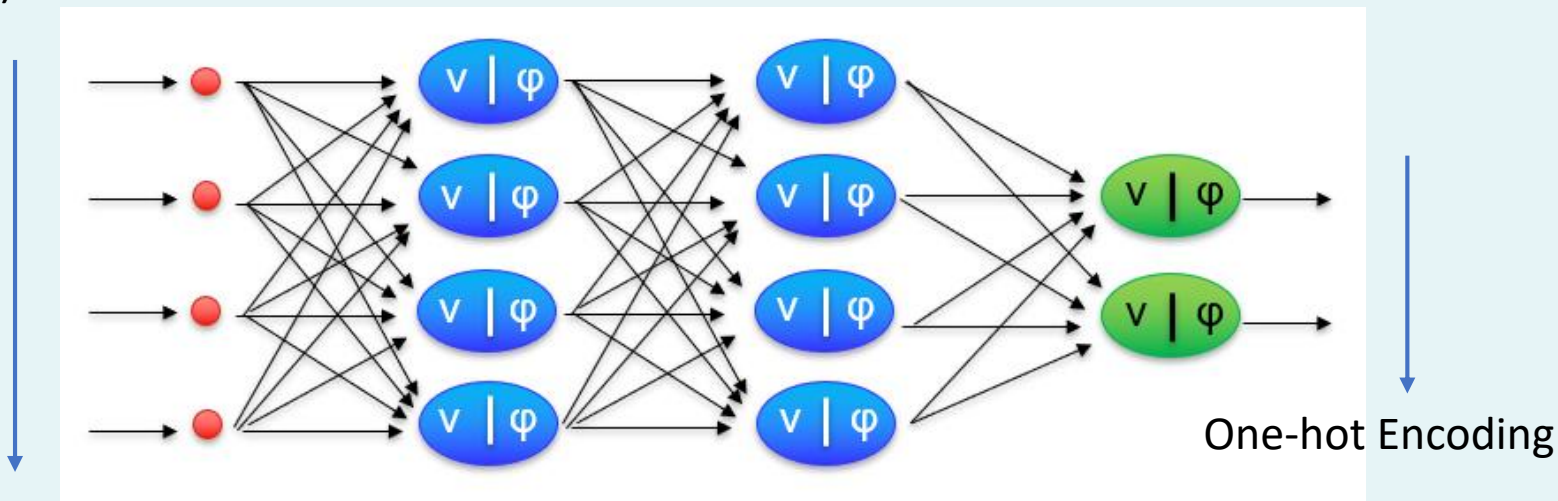
TrainingDataScaled						
50000x3072 double						
	1	2	3	4	5	6
1	0.2314	0.1686	0.1961	0.2667	0.3843	0.4667
2	0.6039	0.4941	0.4118	0.4000	0.4902	0.6078
3	1	0.9922	0.9922	0.9922	0.9922	0.9922
4	0.1098	0.1451	0.1490	0.1647	0.1725	0.1569
5	0.6667	0.6588	0.6941	0.7176	0.7098	0.6941

MLP for Image Classification (11)

- For MLP, what we need is:

Image 1, 2, 3... one at a time
(sequential) →

Labels 1, 2, 3... one at a time
(sequential) →



Pixels in each image

MLP for Image Classification (12)

- So we will need to **transpose** the data and labels:

```
%% Transpose Data and Label
```

```
TrainingDataTransposed = TrainingDataScaled';  
TestDataTransposed = TestDataScaled';  
TrainingLabelsTransposed = TrainingLabelsEncoded';  
TestLabelsTransposed = TestLabelsEncoded';
```

MLP for Image Classification (13)

- We are now ready to specify the MLP and train it.
 - The code was already explained last week.

```
%% Specify the structure and learning algorithm for MLP
```

```
net=newff(minmax(TrainingDataTransposed),[1000,512,10],{'tansig','tansig','logsig'},'traingdx');  
net.trainparam.show=2000; % epochs between display  
net.trainparam.lr=0.01; % learning rate  
net.trainparam.epochs=200; % maximum epochs to train  
net.trainparam.goal=1e-4; % performance goal, training will stop if this is reached
```

```
%% Train the MLP
```

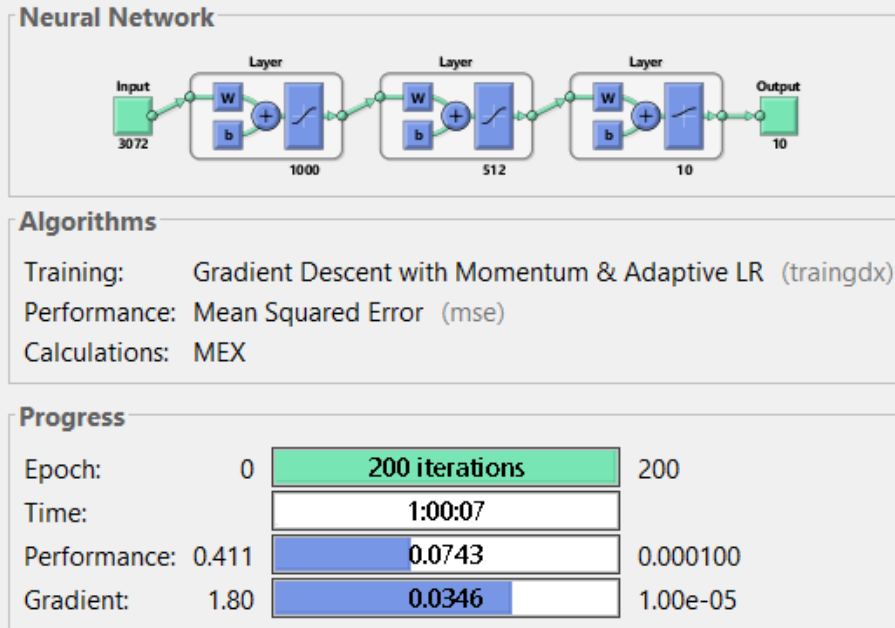
```
[net,tr]=train(net,TrainingDataTransposed,TrainingLabelsTransposed);
```

```
%% Test the MLP
```

```
net_output = sim(net,TestDataTransposed);
```

MLP for Image Classification (14)

- The network structure is illustrated below:



MLP for Image Classification (15)

- At the end, we calculate the **accuracy** as follows:

$$\text{Accuracy} = \frac{\text{number of correctly classified images}}{\text{total number of images}} \times 100\%$$

- In the MATLAB code, it was coded as:

$$\text{Accuracy} = \frac{\left(\begin{array}{c} \text{total number of images} \\ - \text{number of incorrectly classified images} \end{array} \right)}{\text{total number of images}} \times 100\%$$

MLP for Image Classification (16)

- The code is:

```
%% Accuracy
```

```
[modelMaxValue,modelMaxIndex] = max(net_output',[],2); % Transpose again into row!  
modelError = double(TestLabels) - modelMaxIndex;  
nonZeroError = nnz(modelError); % number of non-zero errors  
Accuracy = (length(TestLabels)-nonZeroError)/length(TestLabels)*100; % correct/all*100%
```

- The accuracy for the test set is 40.07%.
 - Better than random (10%).
 - Can perhaps be improved by different network structure, settings etc. – Feel free to try out yourselves!

```
Accuracy =  
  
40.0700
```

Problems with MLP ImClassification (1)

- Enormous number of weights!
 - Because the data was flattened, the input data dimension was 3072.
 - The number of neurons in the first hidden layer would naturally be in similar order of the input data dimension (in this case 1000).
 - We thus have 3,072,000 weights here, plus 1000 biases.
 - Similar calculations for second hidden layer and output layer.
 - In total more than 3 million weights to learn!
 - ... and that's just for 32 x 32 images!

Problems with MLP ImClassification (2)

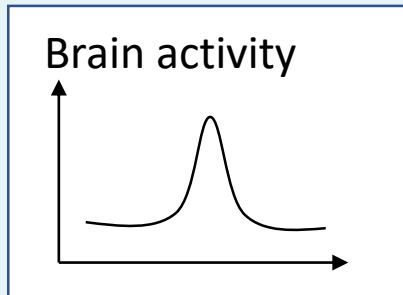
- Spatial properties of image was destroyed by “flattening” process.
 - Images are spatially ordered data.
 - Nearby pixels (not just left/right but also up/down) should carry useful information.
 - By flattening the 2D image into a single row vector (or column vector), we lost this valuable information.

Content

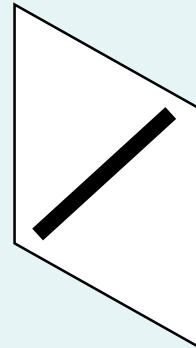
- Introduction to Image Classification
- Commonly-Used Datasets
- MLP for Image Classification
- **History of Convolutional Neural Network (CNN)**
- CNN in Details
- MATLAB & Python Examples
- Examples of CNN Architectures
- Transfer Learning

Hubel and Wiesel 1959 (1)

- David Hubel and Torsten Wiesel investigated the **electrical activity of neurons** in the brains of cats.
- By showing specific patterns on a slide projector, they noticed that **some patterns** stimulated activities in **specific parts of the brain**.
- E.g. some neurons fired when lines at one angle are shown, whereas other neurons responded more to other angles.

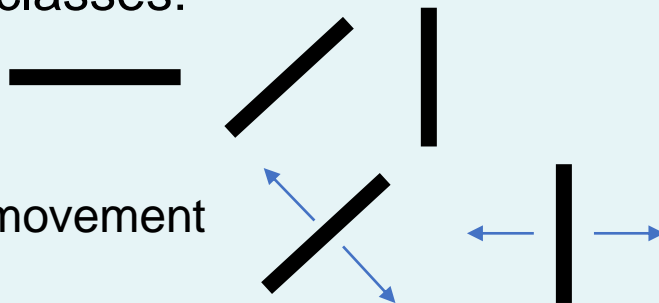


Tungsten
Electrode in
primary visual
cortex

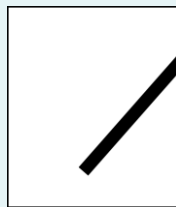
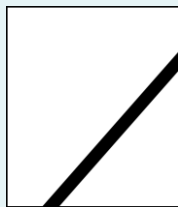


Hubel and Wiesel 1959 (2)

- They also observed other neurons responded to **edges** or **motion** in certain directions.
- They categorized the cells into three classes:
 - **Simple cells**: Respond to orientation
 - **Complex cells**: Respond to edges and movement
 - **Hypercomplex cells**: Respond to movement with an end point



No
response



Response

Hubel and Wiesel 1959 (3)

- Hubel and Wiesel went on to win the Nobel Price in Physiology or Medicine in 1981.

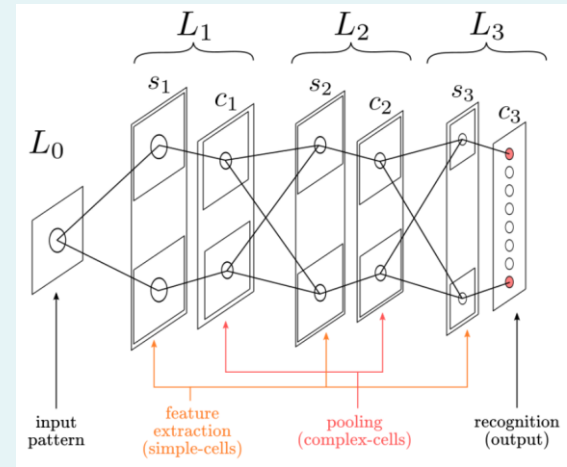


<http://braintour.harvard.edu/archives/portfolio-items/hubel-and-wiesel>

- Their work inspired the computational model of visual system which mimic the behavior of simple cells and complex cells → Next slide.

Kunihiko Fukushima 1980 (1)

- Fukushima proposed “**Neocogniton**”, a computational model of visual system containing **simple and complex cells**, inspired by Hubel and Wiesel’s discovery.
 - Simple cells – Convolution.
 - Complex cells – Pooling.
 - However, no training algorithm at the time.
 - Inspired the Convolutional Neural Network.



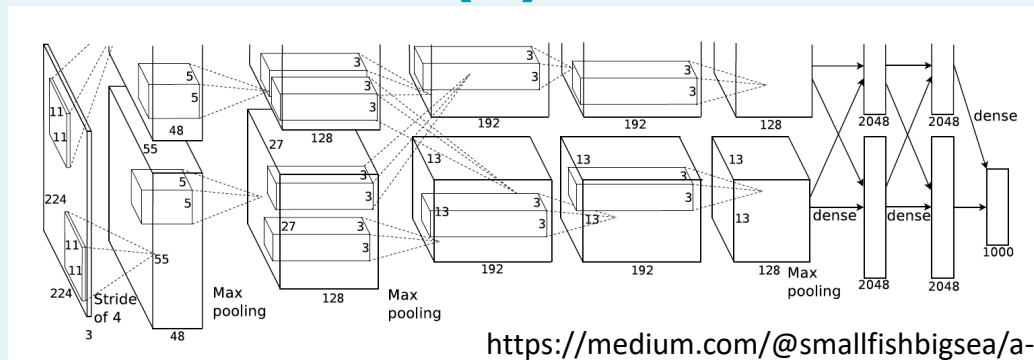
<https://medium.com/@zahraelhamraoui1997/the-convolutional-neural-network-theory-and-implementation-of-lenet-5-and-alexnet-5266e4577e96>

Yann LeCun et al. 1989 (1)

- LeCun et al. at Bells Lab applied the **back-propagation algorithm** to network architecture similar to Neocognitron in 1989.
- Successful **commercial** application in identifying hand-written **zip code** numbers, and **reading checks**.
- The model is later known as “**LeNet**”.
- LeNet-5: Input – Convolution – Pooling – Convolution – Pooling – Convolution – Fully Connected Layer.

Alex Krizhevsky et al. 2012 (1)

- Krizhevsky et al. designed a CNN architecture called “AlexNet”.



<https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637>

- Won the ImageNet challenge in 2012:
 - Accuracy of 16.4%.
 - Approximately 10% better than the runner up, AND the winner of 2011 competition.
 - The huge improvement sparked the interest in CNN and deep learning from 2012 until today.

Content

- Introduction to Image Classification
- Commonly-Used Datasets
- MLP for Image Classification
- History of Convolutional Neural Network (CNN)
- **CNN in Details**
- MATLAB & Python Examples
- Examples of CNN Architectures
- Transfer Learning

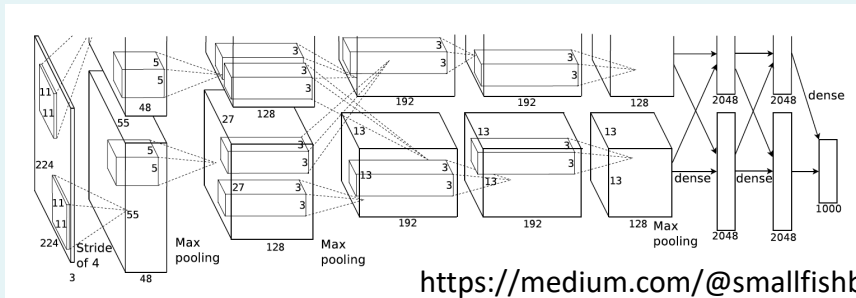
General Structure of CNN (1)

- A convolutional neural network (CNN) consists of several layers in general:
 - Convolution layer
 - ReLU layer
 - Pooling layer
- Followed by fully-Connected layers (i.e. MLP)

Can have different repetitions e.g.

C-R-P-C-R-P-C-R-P....

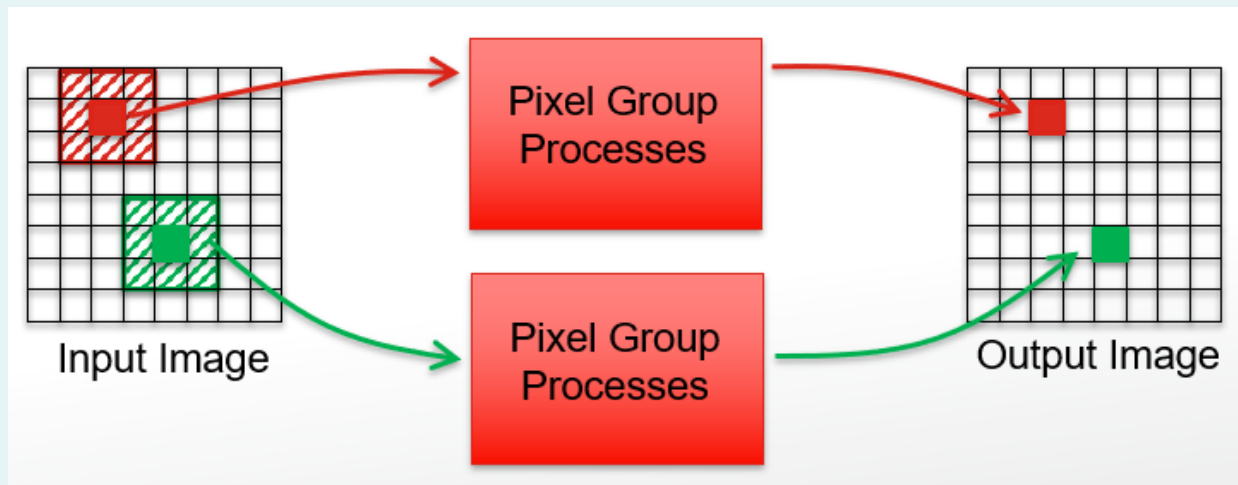
C-R-C-R-P-C-R-C-R-P...



<https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637>

Convolution Layer (1)

- Let's first discuss about the **convolution layer**.
- You have actually already learnt about this in the “image processing” lecture, under topic “**pixel group processing**”.



Convolution Layer (2)

- It is the **elementwise multiplication** of pixel intensities surrounding centre pixel (x, y) with a **convolution mask**.
 - In other words, it's a **dot-product**.

$I(x - 1, y - 1)$	$I(x, y - 1)$	$I(x + 1, y - 1)$
$I(x - 1, y)$	$I(x, y)$	$I(x + 1, y)$
$I(x - 1, y + 1)$	$I(x, y + 1)$	$I(x + 1, y + 1)$

.
(dot product)

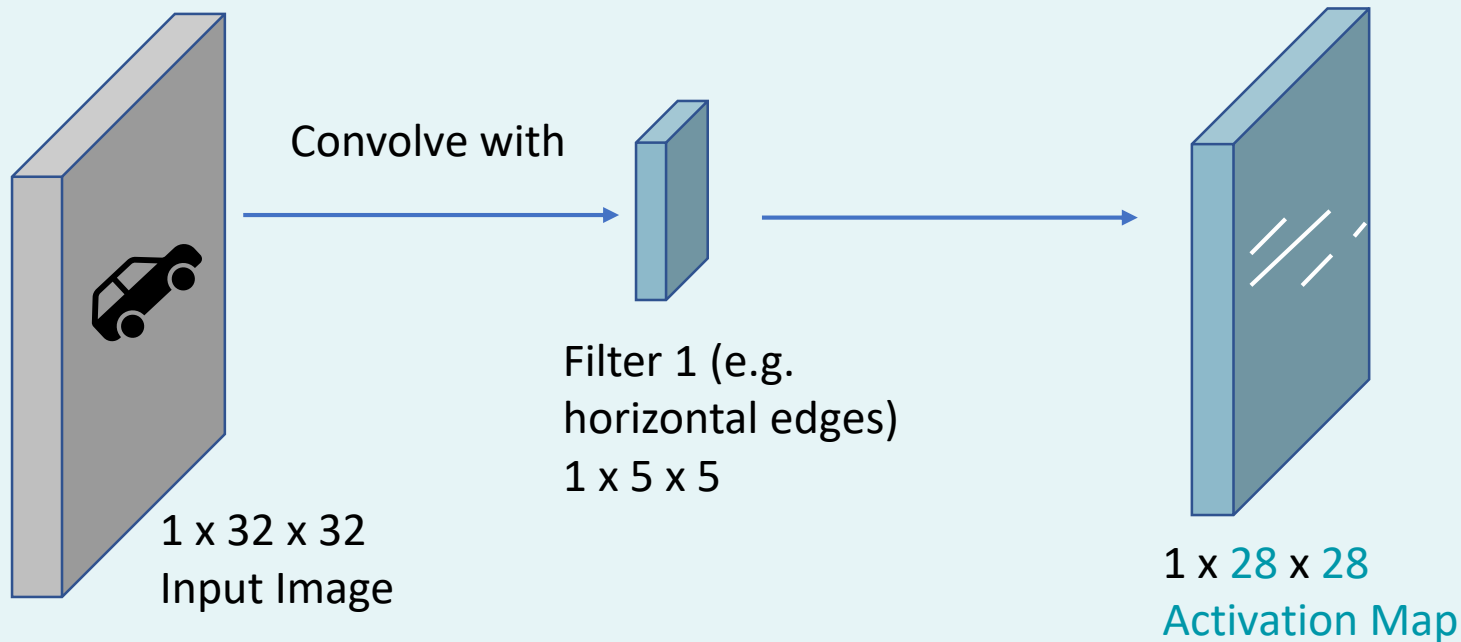
w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

Convolution Layer (3)

- You also learnt that by using some special convolution masks, you can detect **edges**.
- E.g. Sobel filter $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$, $M_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
- So by having some masks, we can **detect simple features** using convolution, just like the “simple cell” discovered by Hubel and Wiesel.
- Note that the convolution **preserves the spatial structure** of the image.

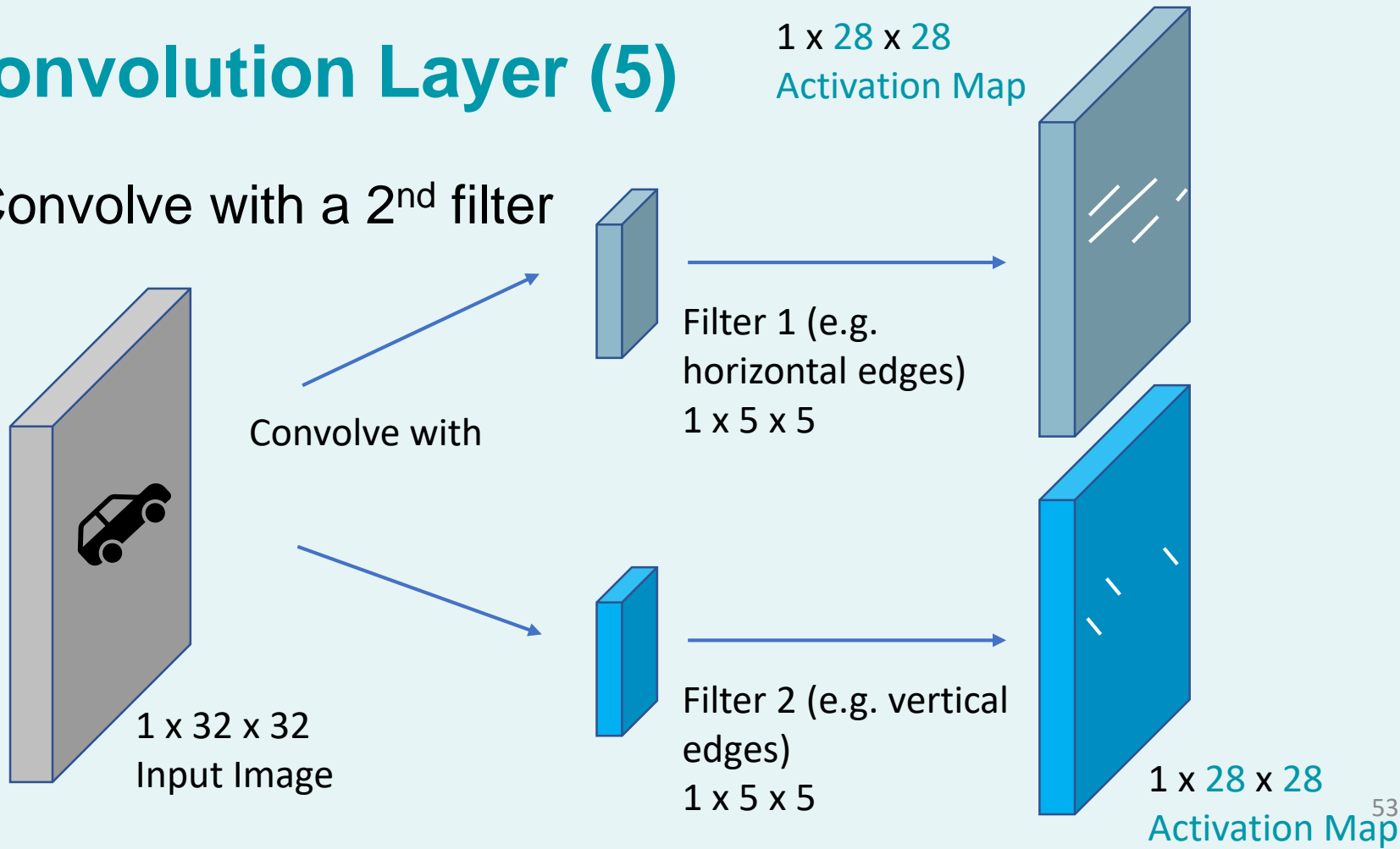
Convolution Layer (4)

- E.g. grey scale image:



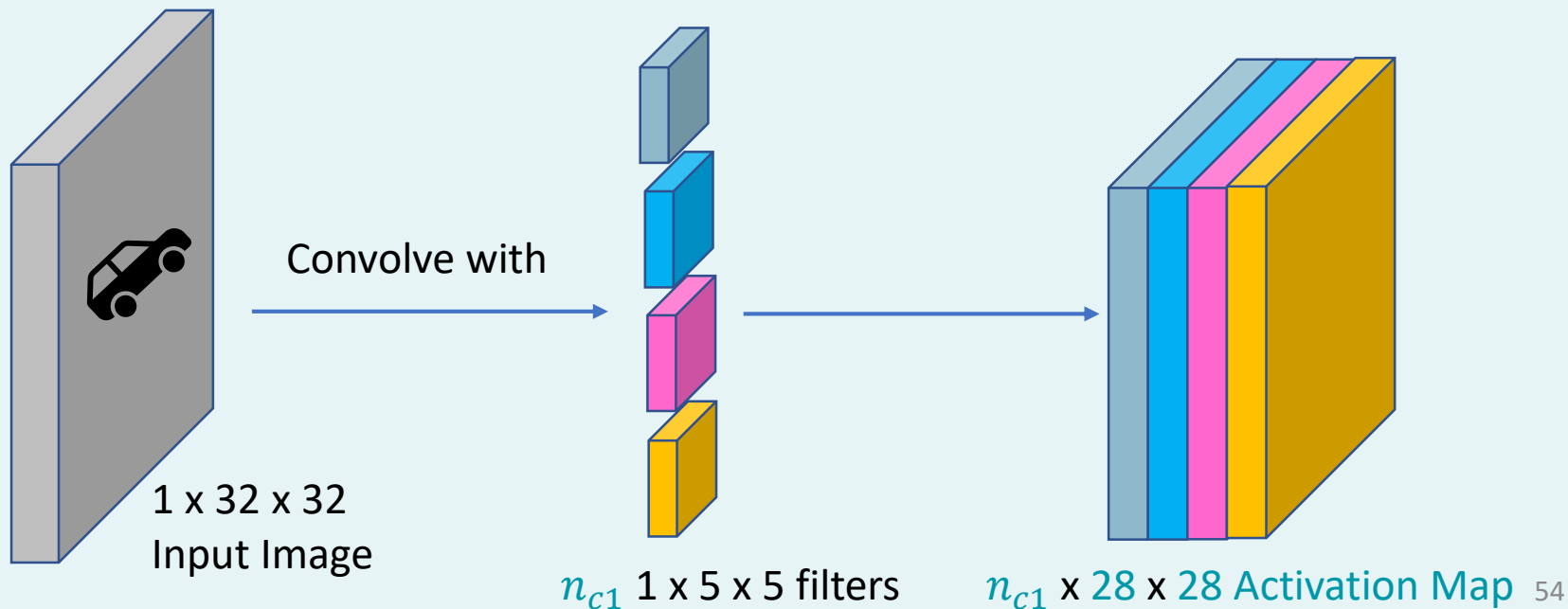
Convolution Layer (5)

- Convolve with a 2nd filter



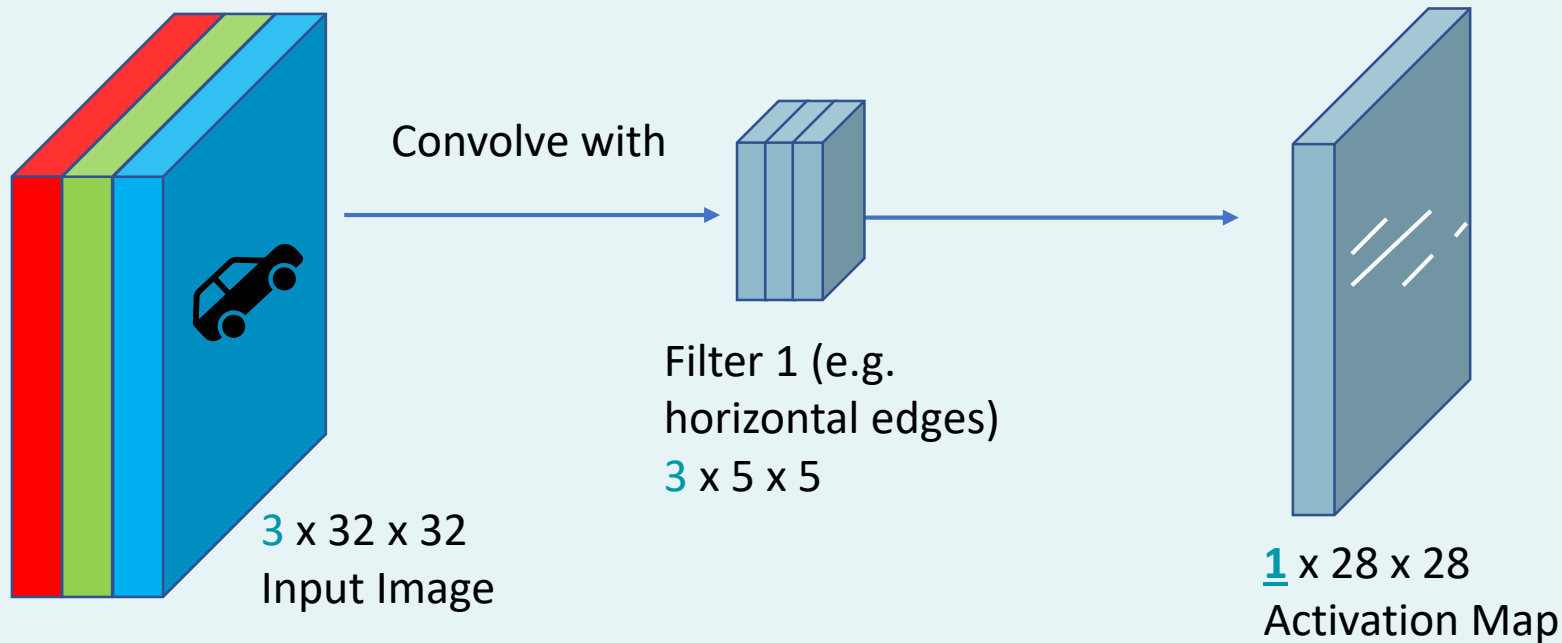
Convolution Layer (6)

- We repeat the convolution with n_{c1} filters, and **stack the activation maps** one after another into a new “image”.



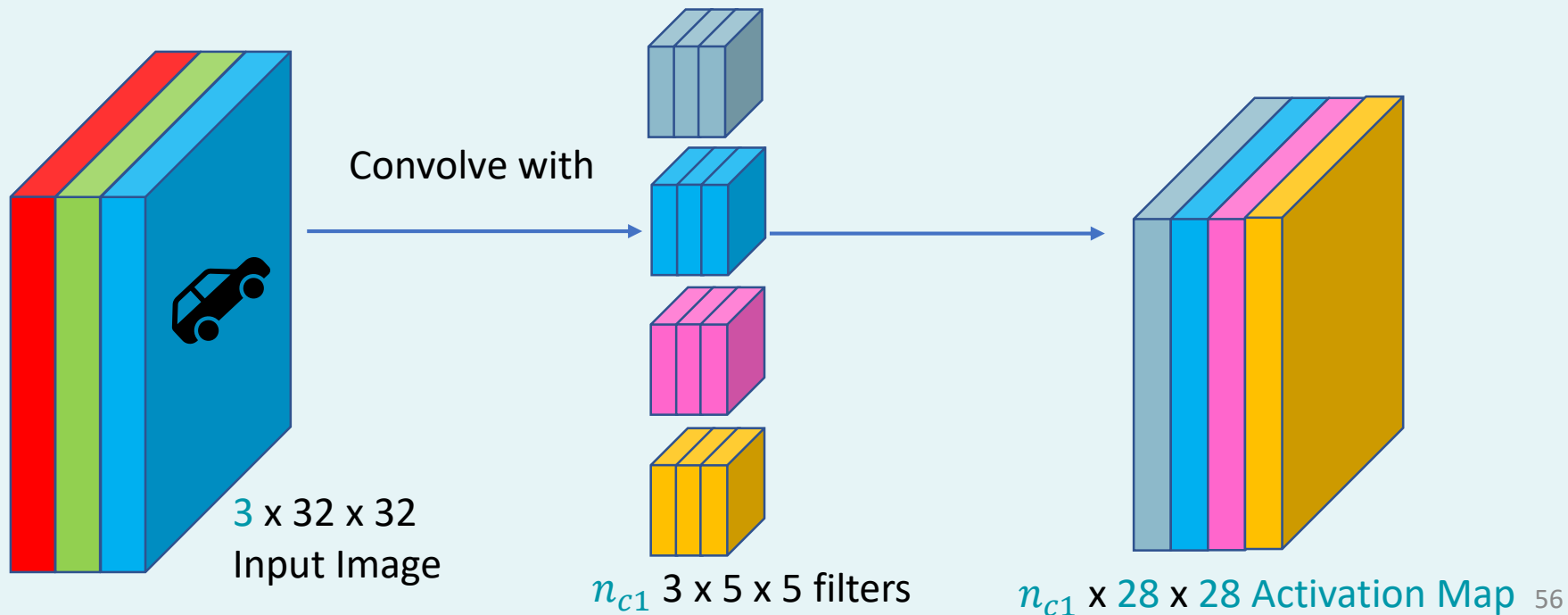
Convolution Layer (7)

- Same idea for RGB images, but **note the dimensions!**



Convolution Layer (8)

- After repeating with n_{c1} filters, we get:



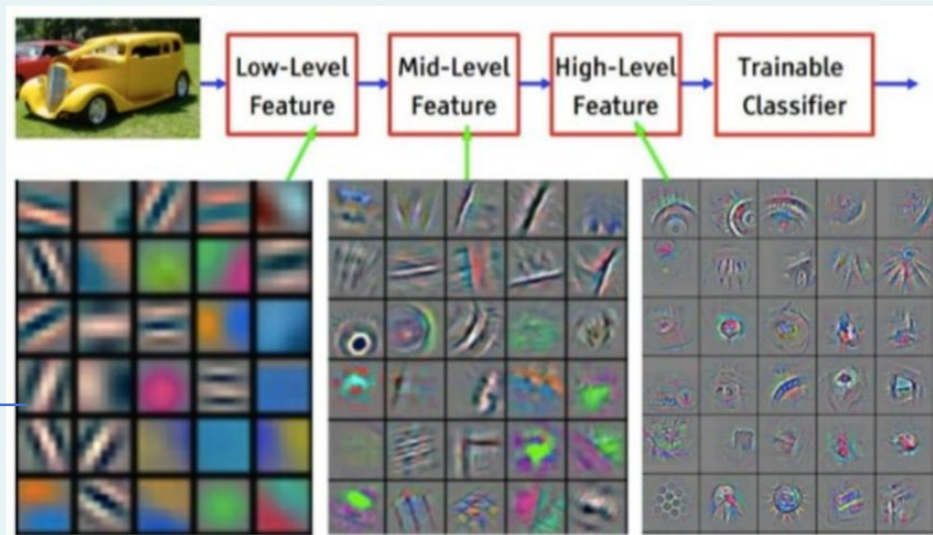
Convolution Layer (9)

- Note: To explain the concepts, the filters were illustrated with specific weights to extract horizontal edges, vertical edges etc.
- In actual fact, the filter weights are learnable parameters, to be updated during the optimization (backpropagation) to minimize classification error!
 - Express loss function in terms of the filter weights, then get the gradient using (a rather long) chain rule.

Convolution Layer (10)

- After training, the convolution layers often learn **low level features (oriented edges)** in the first layer, and combine those features for more **complicated patterns** in subsequent layers.

To visualize first layer, just plot (imshow) the filters, since each filter is 3 x size x size, just like an image.

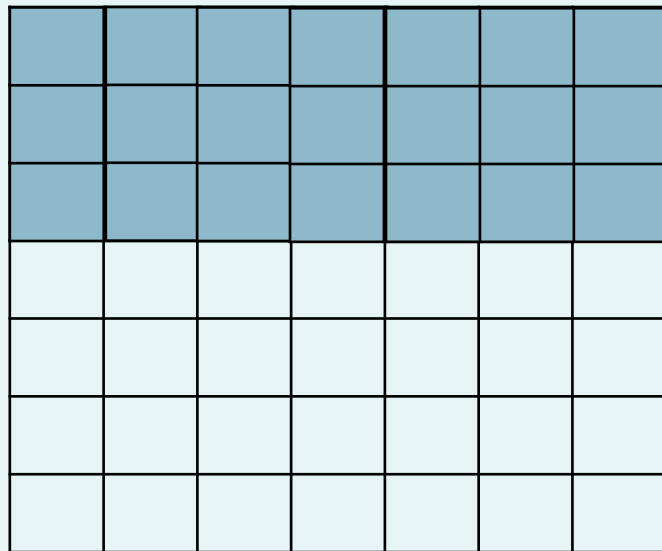


Convolution Layer (11)

- Convolution solves **another problem** of the MLP approach for image classification:
 - It was mentioned that MLP requires a huge number of parameters – Every neuron pair (e.g. input layer – first hidden layer) needs one weight.
 - With convolution layer, each filter (e.g. $5 \times 5 = 25$ parameters) is shared by the whole image. Therefore, convolution greatly **reduces the number of weights** to be learnt!

Convolution Layer – Padding (1)

- In the previous examples, we saw that while the original image has width 32 and height 32, the activation map (after convolution) becomes 28 x 28 (shrinks!)
- Another example:
 - Input: 7 x 7
 - Filter: 3 x 3
 - Can only convolve 5 times horizontally (and vertically)!
 - The output is thus 5 x 5.



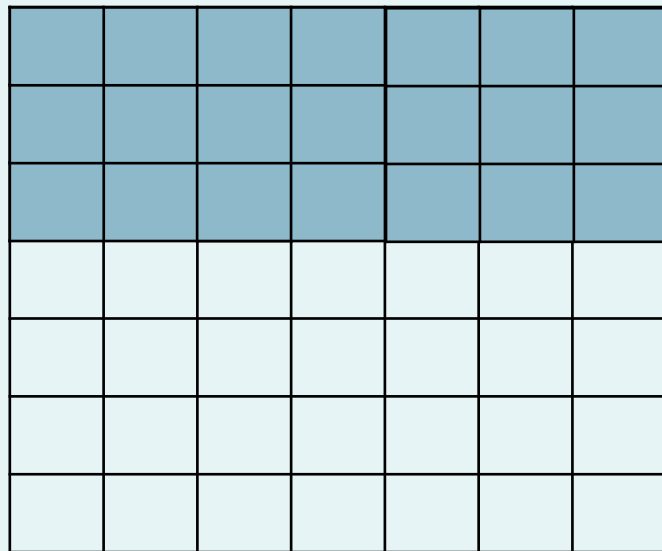
Convolution Layer – Padding (2)

- To prevent shrinking, we can **add zeros** around the input before convolution.
- This is called “**padding**”.
- In MATLAB or Python, this is called “**same**” padding.

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

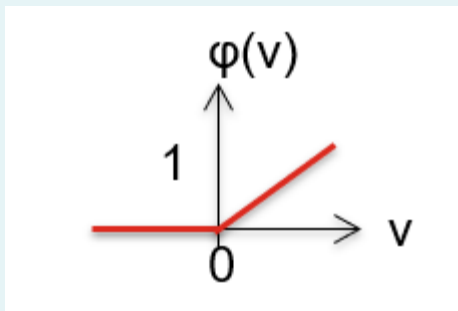
Convolution Layer – Stride (1)

- In the previous examples, the convolution mask slides across the image one-width/height at a time.
- We may also choose to down-sample the output by increasing the “stride”.
- E.g. stride = 2 for image on the right.
 - The output will then be 3 x 3.



ReLU Layer (1)

- After completing the convolution, we then pass the activation maps through ReLU layer.
- You are already familiar with this operation.
- If the pixel value of the activation map is negative, then set it to zero. Otherwise, keep the value.



ReLU Layer (2)

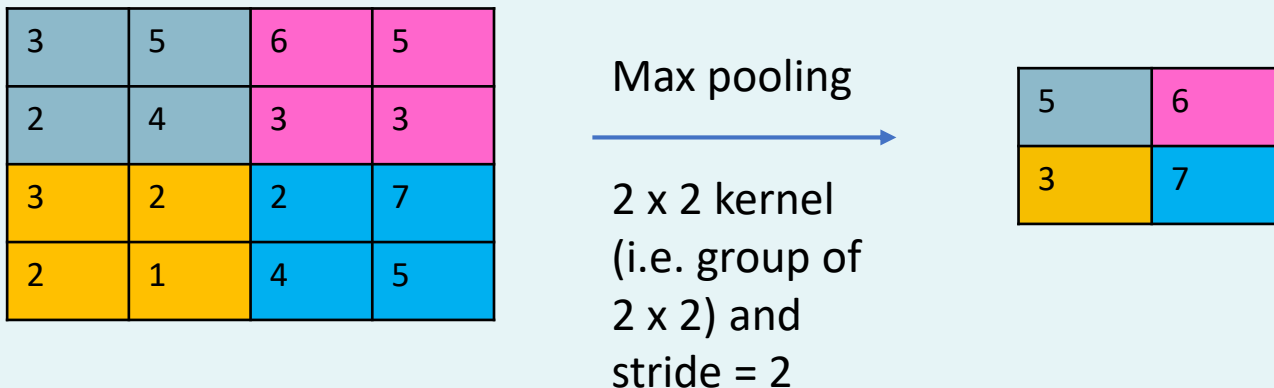
- Note 1: There is no learnable parameters in the ReLU layer.
- Note 2: Convolution layer and ReLU always come in pair.
Example:
 - Conv-ReLU-P-Conv-ReLU-P-Conv-ReLU-P....
 - Conv-ReLU-Conv-ReLU-P-Conv-ReLU-Conv-ReLU-P...

Pooling Layer (1)

- The activation map after convolution and ReLU has one **limitation**: It records the exact position of features in the input.
- Therefore, **small movement** of the features in the input, as a result of translation, rotation etc. will create a different feature map.

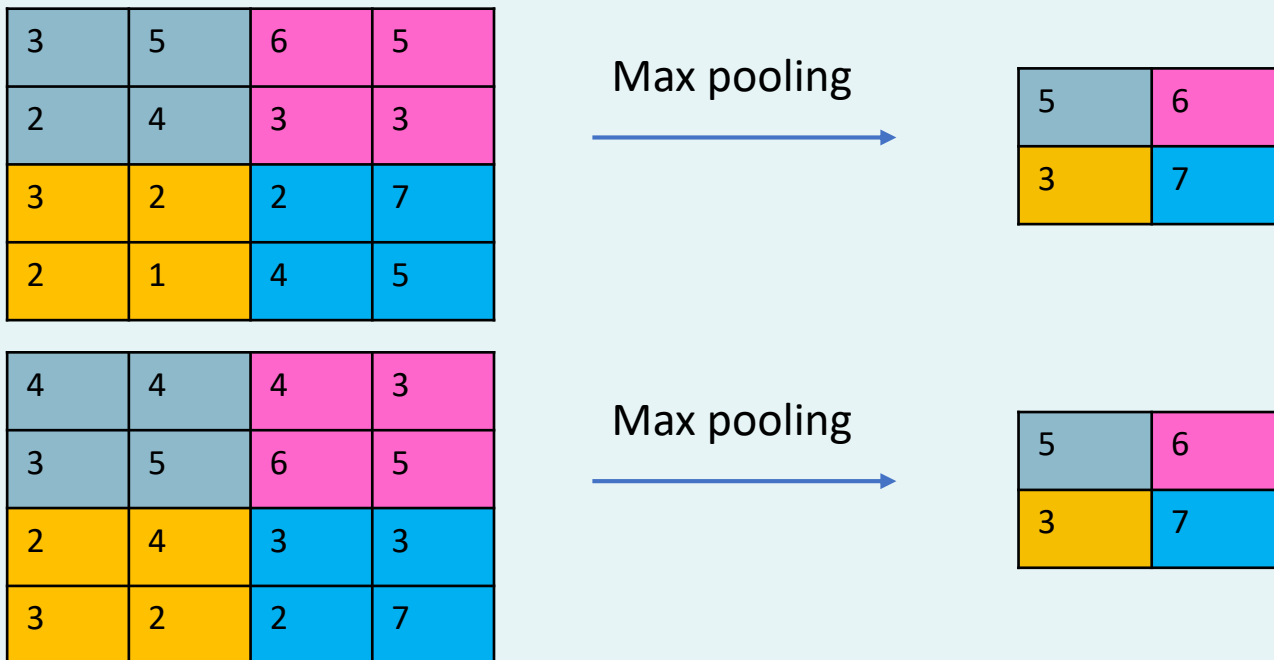
Pooling Layer (2)

- To address this issue, we can “down-sample” the feature map, i.e. using a number to represent a group of pixels.
 - This is called “pooling”.
 - Commonly used: “max pooling” or “average pooling”.



Pooling Layer (3)

- To illustrate the invariance to small shift:



Pooling Layer (4)

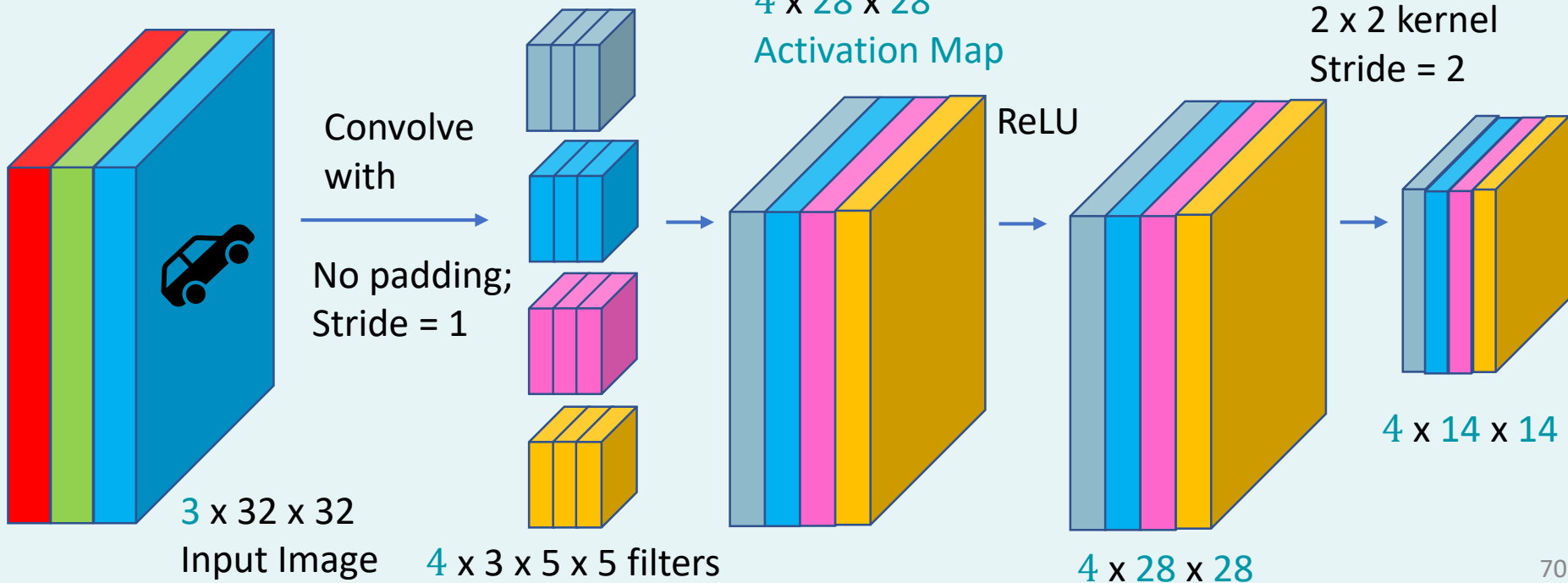
- The MAX pooling function retains large or important elements, while ignoring some finer details which may not be crucial for the image classification.
- Note: Pooling layer also does not have learnable parameters!
 - The only hyperparameter here is the type of pool function.

Conv-ReLU-Pool (1)

- We have now understood the convolution, ReLU and Pool layers.
- As mentioned, we can combine (conv+ReLU) and (pool) freely.
- Let's look at two examples, paying attention to the dimensions.

Conv-ReLU-Pool (2)

- Example 1:



Conv-ReLU-Pool (3)

- Example 2:

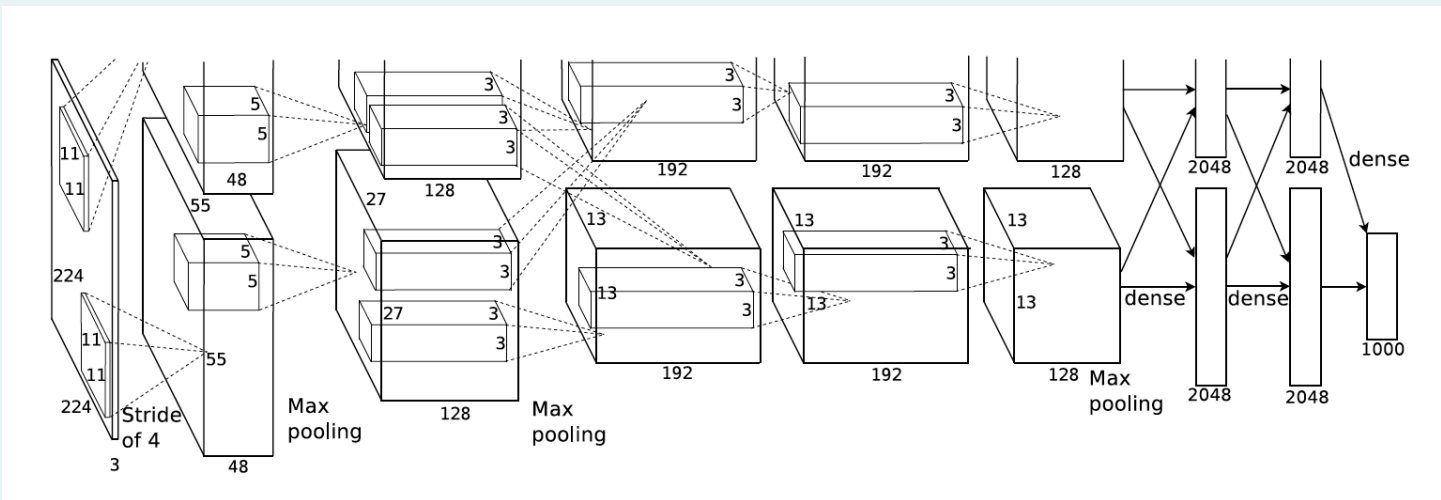
Layer	Setting	Output Size
Input	3 x 32 x 32	3 x 32 x 32
Convolution + ReLU	6 x 3 x 5 x 5 Same padding Stride = 1	6 x 32 x 32
Convolution + ReLU	12 x 6 x 5 x 5 No padding Stride = 1	12 x 28 x 28
Max Pool	4 x 4 kernel Stride = 4	12 x 7 x 7

Fully-Connected Layer (1)

- Finally, after several convolution, ReLU and pool layer, we **flatten** the final activation map, and send the array into a **fully-connected-layer**, which is just another MLP.
- However, unlike the previous CIFAR-10 example with pure MLP, here the activation map is much smaller, after going through down-sampling by strided convolution, or convolution without padding, as well as pooling.

Fully-Connected Layer (2)

- The fully-connected layer is called “Dense” in Python.



<https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637>

Improving CNN Training (1)

- Several techniques have been found to improve the training of CNN:
 - Batch Normalization
 - Data Augmentation
 - Learning Rate Schedule

Batch Normalization (1)

- We have previously discussed the importance of normalizing the inputs.
- It is also important to **normalize the outputs of each neuron** layer so that they have **zero mean** and **unit variance**, before they are fed as inputs to the next layer.
 - This is called “**Batch Normalization**”, which is done on a per-batch-basis.
 - Note: Usually added **after convolution and fully-connected** layer, BEFORE nonlinear function.

Batch Normalization (2)

- Implementation:
 - Step 1: Calculate mean and variance:

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

- $\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j}$ is the **mean** of all x_i (here: 2D pixels) in **channel j** , one minibatch at a time.
- $\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2$ is the **variance** of all x_i in **channel j** , one minibatch at a time.

Batch Normalization (3)

- Step 2: Calculate **normalized output**:

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

- γ_j scales the standard deviation;
 - β_j shifts the mean.
 - γ_j and β_j are both **learnable parameters**, learned during the backpropagation process!
- Step 3: The normalized output $y_{i,j}$ is then fed to the subsequent nonlinear layer.

Batch Normalization (4)

- Note that the mean and variance depends on the minibatch.
- During **testing** with unseen data, we **can't calculate these!**
- The solution is therefore to use the **running average** of mean and variance obtained during training.

Batch Normalization (5)

- Two theories of why batch normalization works:
 - 1) Helps reduce “Internal Covariate Shift”, i.e. the change in the distribution of inputs to layers in the network which causes the learning algorithm to keep chasing a moving target. Batch normalization stabilizes the learning process.
 - 2) Batch normalization smoothens the objective function, which in turn improves performance.
 - The exact theoretical reason is still unknown!

Data Augmentation (1)

- Data augmentation is a technique used to **reduce overfitting** when training CNN.
- Given a labeled image, e.g. “dog”
 - We “**create**” **new labeled images** by flipping horizontally, scaling, cropping, changing brightness etc. and **add into training set**.

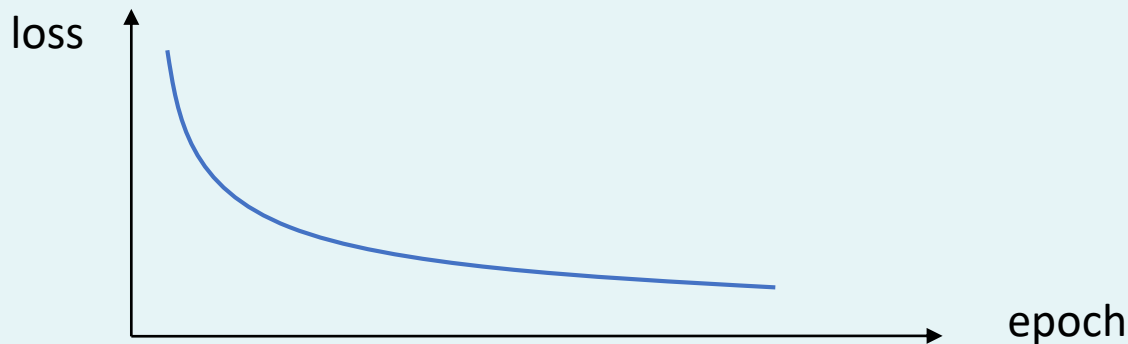
(Given)



- This has a **positive side effect** of increasing amount of data!

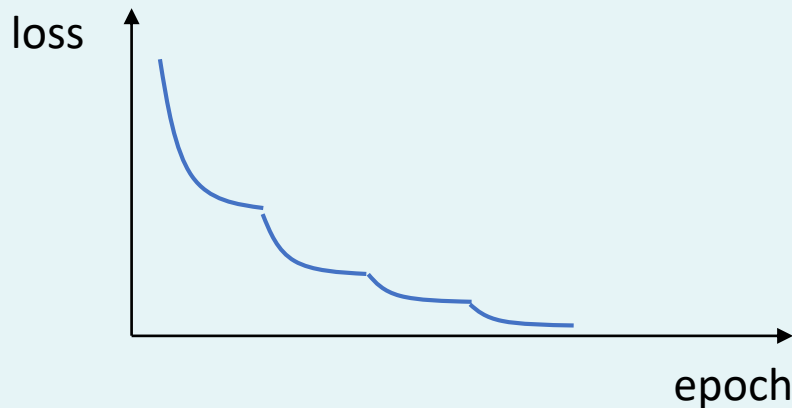
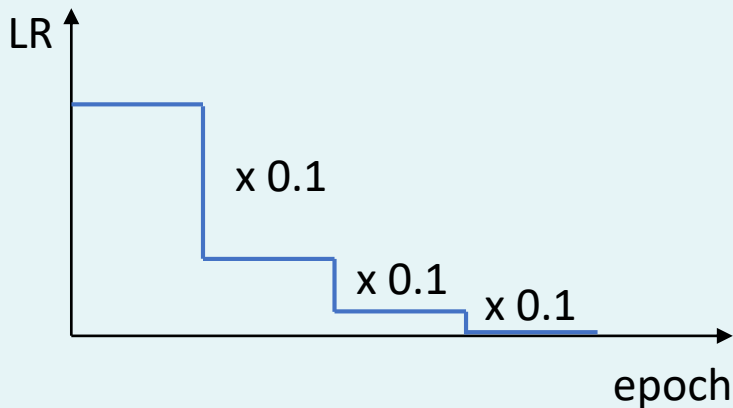
Learning Rate Schedule (1)

- Assume we have chosen a **good learning rate** for our optimization algorithm.
- The loss function would **decrease rapidly** at the start, but **slows down** as it approaches the minimum.



Learning Rate Schedule (2)

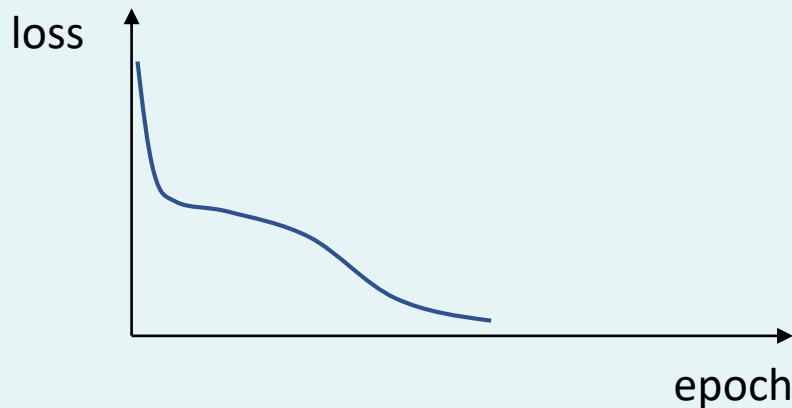
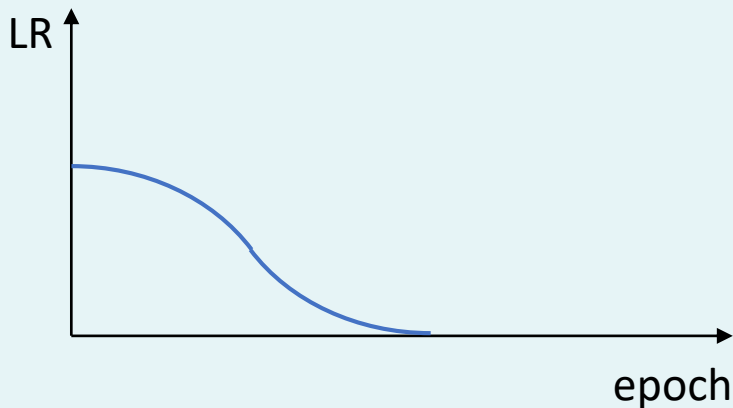
- It has been observed that if we **reduce the learning rate as epoch increases**, we could potentially improve the learning.
- E.g. Step decrease: Multiply by a factor every nth epoch.



Learning Rate Schedule (3)

- E.g. Cosine decrease:

$$\eta(t) = \frac{1}{2} \eta(0) \left(1 + \cos \left(\frac{t\pi}{T} \right) \right)$$



Content

- Introduction to Image Classification
- Commonly-Used Datasets
- MLP for Image Classification
- History of Convolutional Neural Network (CNN)
- CNN in Details
- **MATLAB & Python Examples**
- Examples of CNN Architectures
- Transfer Learning

MATLAB Example (1)

- We shall use the same dataset which we used earlier when training the MLP, namely CIFAR-10.
- Please [download from Moodle](#):
 - CNN_Code.m
 - saveCIFAR10AsFolderOfImages.m
- You will also need cifar-10-batches-mat which we used earlier when training MLP.

MATLAB Example (2)

- The codes in (CNN_Code.m) and (saveCIFAR10AsFolderOfImages.m) are both written by MATLAB.
- However, several comments are added for better understanding, and some minor changes are made due to not having GPU on my laptop.

MATLAB Example (3)

- Please have a look at CNN_Code.m
 - The first few sections are for [preparation of data](#).
 - The “[Define a CNN architecture](#)” section is where you can modify and check the effects of different CNN architecture:

```
%% Define a CNN architecture
conv1 = convolution2dLayer(5,32,'Padding',2,...
    'BiasLearnRateFactor',2); % 2D convolution layer,
    % size 5x5,
    % number of filters = 32,
    % padding = 2 rows on each side
    % learning rate multiplier for bias = 2

% conv1.Weights = gpuArray(single(randn([5 5 3 32])*0.0001)); % comment if no GPU

fc1 = fullyConnectedLayer(64,'BiasLearnRateFactor',2); % output size = 64

% fc1.Weights = gpuArray(single(randn([64 576])*0.1)); % comment if not GPU

fc2 = fullyConnectedLayer(10,'BiasLearnRateFactor',2); % output size = 10

% fc2.Weights = gpuArray(single(randn([10 64])*0.1)); % comment if no GPU
```

MATLAB Example (4)

```
layers = [ ...  
    imageInputLayer([32 32 3]);  
    conv1; % defined earlier for potential use with GPU  
    maxPooling2dLayer(3,'Stride',2); % kernel size 3 x 3, stride 2  
    reluLayer();  
    convolution2dLayer(5,32,'Padding',2,'BiasLearnRateFactor',2);  
    reluLayer();  
    averagePooling2dLayer(3,'Stride',2); % kernel size 3 x 3, stride 2  
    convolution2dLayer(5,64,'Padding',2,'BiasLearnRateFactor',2);  
    reluLayer();  
    averagePooling2dLayer(3,'Stride',2); % kernel size 3 x 3, stride 2  
    fc1; % defined earlier for potential use with GPU  
    reluLayer();  
    fc2; % defined earlier for potential use with GPU  
    softmaxLayer()  
    classificationLayer()];
```


MATLAB Example (5)

- We then train the network:

```
% Define the training options.
opts = trainingOptions('sgdm', ... % stochastic gradient with momentum
    'InitialLearnRate', 0.001, ...
    'LearnRateSchedule', 'piecewise', ... % reduce learning rate
    'LearnRateDropFactor', 0.1, ...      % by a factor of 0.1
    'LearnRateDropPeriod', 8, ...       % every 8 epochs
    'L2Regularization', 0.004, ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 100, ...           % Mini-batch mode
    'Verbose', true);

%% Training the CNN
[net, info] = trainNetwork(XTrain, TTrain, layers, opts);
```

MATLAB Example (6)

- While training, we can see progress on MATLAB:

```
Training on single CPU.
```

```
Initializing input data normalization.
```

=====						
Epoch	Iteration	Time Elapsed	Mini-batch	Mini-batch	Base Learning	
		(hh:mm:ss)	Accuracy	Loss	Rate	
=====						
1	1	00:00:00	8.00%	10.9286	0.0010	
1	50	00:00:11	24.00%	2.1212	0.0010	
1	100	00:00:21	23.00%	2.0984	0.0010	
1	150	00:00:31	27.00%	1.9246	0.0010	

...

10	4850	00:17:51	69.00%	0.8703	0.0001	
10	4900	00:18:01	71.00%	0.9511	0.0001	
10	4950	00:18:13	71.00%	0.8609	0.0001	
10	5000	00:18:24	63.00%	1.1153	0.0001	
=====						

```
Elapsed time is 6.034616 seconds.
```

MATLAB Example (7)

- After training, the network is presented with **unseen test data**

```
%% Run the network on the test set

YTest = classify(net, XTest);

% Calculate the accuracy.
accuracy = sum(YTest == TTest)/numel(TTest)
```

- The **accuracy is 71.08%**, a huge improvement over simple MLP's 40.07%.

```
accuracy =

    0.7108
```

MATLAB Example (8)

- Batch Normalization is then added to see if its effect:

- The accuracy improves to 74.94%!

accuracy =

0.7494

```
layers = [ ...  
    imageInputLayer([32 32 3]);  
    conv1; % defined earlier for potential use with GPU  
    ➔ batchNormalizationLayer; % added to improve training  
    maxPooling2dLayer(3, 'Stride', 2); % kernel size 3 x 3, stride 2  
    reluLayer();  
    convolution2dLayer(5, 32, 'Padding', 2, 'BiasLearnRateFactor', 2);  
    ➔ batchNormalizationLayer; % added to improve training  
    reluLayer();  
    averagePooling2dLayer(3, 'Stride', 2); % kernel size 3 x 3, stride 2  
    convolution2dLayer(5, 64, 'Padding', 2, 'BiasLearnRateFactor', 2);  
    ➔ batchNormalizationLayer; % added to improve training  
    reluLayer();  
    averagePooling2dLayer(3, 'Stride', 2); % kernel size 3 x 3, stride 2  
    fc1; % defined earlier for potential use with GPU  
    reluLayer();  
    fc2; % defined earlier for potential use with GPU  
    softmaxLayer()  
    classificationLayer()];
```

MATLAB Example (9)

- To further improve the result, we can continue trying **different combination** of layers and training parameters.
- Please wait for the next section (Examples of CNN Architectures) for some inspirations.

Python Example (1)

- There are several **deep learning framework** in Python, the most famous ones being:
 - **Pytorch**
 - More “coding from scratch”, good for experienced researchers.
 - **Keras**
 - More “high level”, plug-and-play. Good for rapid prototyping.
 - **Tensorflow**
 - In between Pytorch and Keras. But Keras is actually a high-level library that runs on top of Tensorflow!

Python Example (2)

- Here we will show a **Keras** example [1] on MNIST data.

```
import numpy as np
import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
from tensorflow.keras.utils import to_categorical

train_images = mnist.train_images()
train_labels = mnist.train_labels()
test_images = mnist.test_images()
test_labels = mnist.test_labels()
```

Python Example (3)

```
# Normalize the images.  
train_images = (train_images / 255) - 0.5  
test_images = (test_images / 255) - 0.5  
  
# Reshape the images.  
train_images = np.expand_dims(train_images, axis=3)  
test_images = np.expand_dims(test_images, axis=3)  
  
num_filters = 8  
filter_size = 3  
pool_size = 2
```


Python Example (4)

```
# Build the model.
model = Sequential([
    Conv2D(num_filters, filter_size, input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=pool_size),
    Flatten(),
    Dense(10, activation='softmax'),
])

# Compile the model.
model.compile(
    'adam',
    loss='categorical_crossentropy',
    metrics=['accuracy'],
)
```

Python Example (5)

```
# Train the model.
model.fit(
    train_images,
    to_categorical(train_labels),
    epochs=3,
    validation_data=(test_images, to_categorical(test_labels)),
)

# Save the model to disk.
model.save_weights('cnn.h5')

# Load the model from disk later using:
# model.load_weights('cnn.h5')
```

Python Example (6)

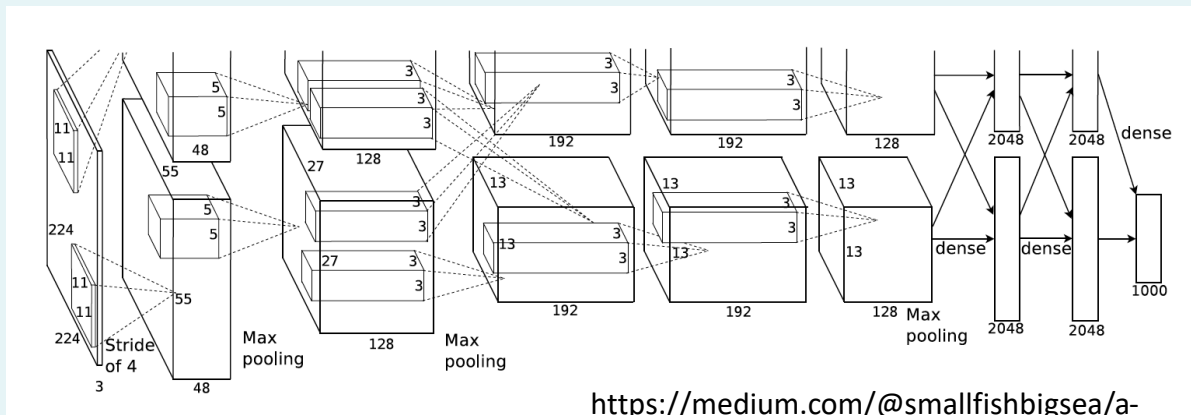
```
# Predict on the first 5 test images.  
predictions = model.predict(test_images[:5])  
  
# Print our model's predictions.  
print(np.argmax(predictions, axis=1)) # [7, 2, 1, 0, 4]  
  
# Check our predictions against the ground truths.  
print(test_labels[:5]) # [7, 2, 1, 0, 4]
```

Content

- Introduction to Image Classification
- Commonly-Used Datasets
- MLP for Image Classification
- History of Convolutional Neural Network (CNN)
- CNN in Details
- MATLAB & Python Examples
- **Examples of CNN Architectures**
- Transfer Learning

AlexNet (1)

- In this section, we will look at several well-known CNN architecture.
- AlexNet, by Krizhevsky et al. 2012:
 - Performance on ImageNet: 16.4%



<https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637>

AlexNet (2)

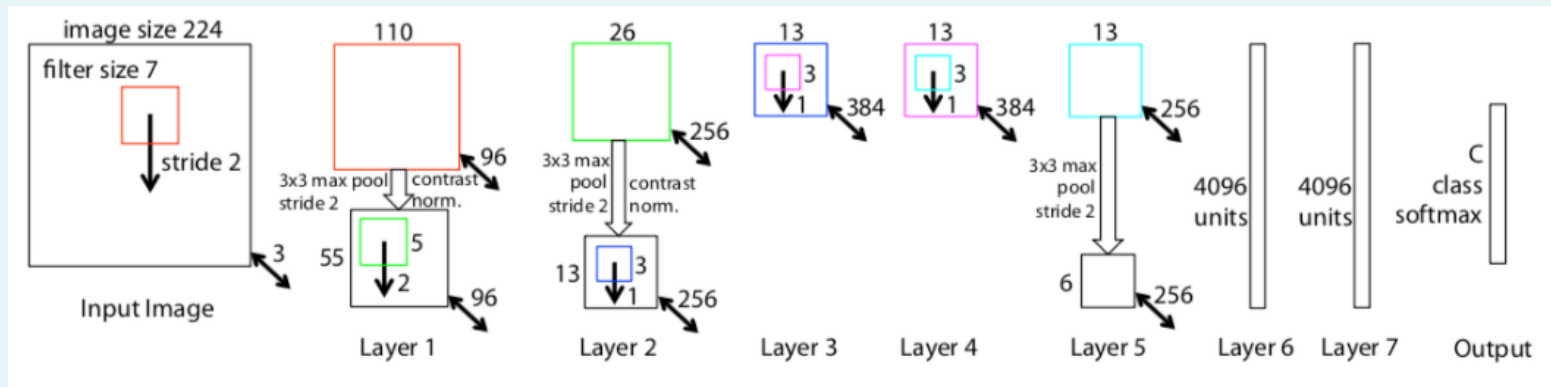
Layer	Size	Stride	Pad	Output Size
Input Image	3 x 224 x 224			3 x 224 x 224
Conv + ReLU	64 x (3 x 11 x 11)	4	2	64 x 56 x 56
Pool	3 x 3	2	0	64 x 27 x 27
Conv + ReLU	192 x (64 x 5 x 5)	1	2	192 x 27 x 27
Pool	3 x 3	2	0	192 x 13 x 13
Conv + ReLU	384 x (192 x 3 x 3)	1	1	384 x 13 x 13
Conv + ReLU	256 x (384 x 3 x 3)	1	1	256 x 13 x 13
Conv + ReLU	256 x (256 x 3 x 3)	2	0	256 x 13 x 13
Pool	3 x 3	2	0	256 x 6 x 6
Flatten				9216
Fully Connect	4096 nodes			4096
Fully Connect	4096 nodes			4096
Fully Connect	1000 nodes			1000

Increasing
channels;
Decreasing sizes

Most weights
are here

ZFNet (1)

- ZFNet, by Zeiler and Fergus, 2013:
 - Performance on ImageNet: 11.7%
 - Optimized version of AlexNet – Bottlenecks of AlexNet removed.



Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

ZFNet (2)

Layer	Size	Stride	Pad	Output Size
Input Image	3 x 224 x 224			3 x 224 x 224
Conv + ReLU	96 x (3 x 7 x 7)	2	2	96 x 110 x 110
Pool	3 x 3	2	0	96 x 55 x 55
Conv + ReLU	256 x (96 x 5 x 5)	2	2	256 x 26 x 26
Pool	3 x 3	2	0	256 x 13 x 13
Conv + ReLU	384 x (256 x 3 x 3)	1	1	384 x 13 x 13
Conv + ReLU	384 x (384 x 3 x 3)	1	1	384 x 13 x 13
Conv + ReLU	256 x (384 x 3 x 3)	1	0	256 x 13 x 13
Pool	3 x 3	2	0	256 x 6 x 6
Flatten				9216
Fully Connect	4096 nodes			4096
Fully Connect	4096 nodes			4096
Fully Connect	1000 nodes			

Increasing
channels;
Decreasing sizes

Most weights
are here

VGG (1)

- VGG, by Simonyan and Zisserman, 2014:
 - Performance by VGG19 on ImageNet: 7.3%
 - AlexNet and ZFNet needed a lot of trial-&-error for structures.
 - Simonyan and Zisserman proposed some design rules:
 - All convolution layers: size 3 x 3; stride = 1; pad = 1;
 - All max pool layers: size 2 x 2; stride = 2;
 - After pool layer, double the number of channels (but cap at 512).
 - Using these rules, the network will be deeper rather than wider.
 - Why? E.g. receptive field of one 5 x 5 filter is the same as that of two 3 x 3 filters, but the latter uses less parameters (25 vs 2 x 9).

VGG (2)

• VGG16:

Layer	Size	Stride	Pad	Output Size
Input Image	3 x 224 x 224			3 x 224 x 224
Conv + ReLU	64 x (3 x 3 x 3)	1	1	64 x 224 x 224
Conv + ReLU	64 x (64 x 3 x 3)	1	1	64 x 224 x 224
Pool	2 x 2	2		64 x 112 x 112
Conv + ReLU	128 x (64 x 3 x 3)	1	1	128 x 112 x 112
Conv + ReLU	128 x (128 x 3 x 3)	1	1	128 x 112 x 112
Pool	2 x 2	2		128 x 56 x 56
Conv + ReLU	256 x (128 x 3 x 3)	1	1	256 x 56 x 56
Conv + ReLU	256 x (256 x 3 x 3)	1	1	256 x 56 x 56
Pool	2 x 2	2		256 x 28 x 28



A

A



Layer	Size	Stride	Pad	Output Size
Conv + ReLU	512 x (256 x 3 x 3)	1	1	512 x 28 x 28
Conv + ReLU	512 x (512 x 3 x 3)	1	1	512 x 28 x 28
Conv + ReLU	512 x (512 x 3 x 3)	1	1	512 x 28 x 28
Pool	2 x 2	2		512 x 14 x 14
Conv + ReLU	512 x (512 x 3 x 3)	1	1	512 x 14 x 14
Conv + ReLU	512 x (512 x 3 x 3)	1	1	512 x 14 x 14
Conv + ReLU	512 x (512 x 3 x 3)	1	1	512 x 14 x 14
Pool	2 x 2	2		512 x 7 x 7
Flatten				25088
Fully Connect	4096 nodes			4096
Fully Connect	4096 nodes			4096
Fully Connect	1000 nodes			1000

VGG (3)

• VGG19:

Layer	Size	Stride	Pad	Output Size
Input Image	3 x 224 x 224			3 x 224 x 224
Conv + ReLU	64 x (3 x 3 x 3)	1	1	64 x 224 x 224
Conv + ReLU	64 x (64 x 3 x 3)	1	1	64 x 224 x 224
Pool	2 x 2	2		64 x 112 x 112
Conv + ReLU	128 x (64 x 3 x 3)	1	1	128 x 112 x 112
Conv + ReLU	128 x (128 x 3 x 3)	1	1	128 x 112 x 112
Pool	2 x 2	2		128 x 56 x 56
Conv + ReLU	256 x (128 x 3 x 3)	1	1	256 x 56 x 56
Conv + ReLU	256 x (256 x 3 x 3)	1	1	256 x 56 x 56
Pool	2 x 2	2		256 x 28 x 28



A

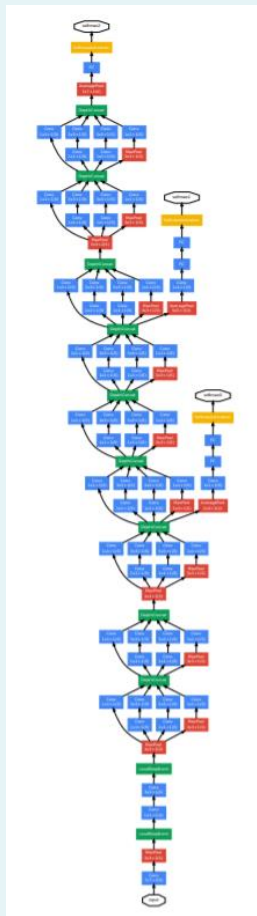


Layer	Size	Stride	Pad	Output Size
Conv + ReLU	512 x (256 x 3 x 3)	1	1	512 x 28 x 28
Conv + ReLU	512 x (512 x 3 x 3)	1	1	512 x 28 x 28
Conv + ReLU	512 x (512 x 3 x 3)	1	1	512 x 28 x 28
Conv + ReLU	512 x (512 x 3 x 3)	1	1	512 x 28 x 28
Pool	2 x 2	2		512 x 14 x 14
Conv + ReLU	512 x (512 x 3 x 3)	1	1	512 x 14 x 14
Conv + ReLU	512 x (512 x 3 x 3)	1	1	512 x 14 x 14
Conv + ReLU	512 x (512 x 3 x 3)	1	1	512 x 14 x 14
Conv + ReLU	512 x (512 x 3 x 3)	1	1	512 x 14 x 14
Pool	2 x 2	2		512 x 7 x 7
Flatten				25088
Fully Connect	4096 nodes			4096
Fully Connect	4096 nodes			4096
Fully Connect	1000 nodes			1000

GoogLeNet (1)

- GoogLeNet, by Szegedy et al., 2015:
 - Performance by GoogLeNet on ImageNet: 6.7%
 - Clever name! Google (created by researchers at Google) + LeNet (the basis of CNN)
 - Focus on efficiency: Reduce parameter, memory usage and computation.
 - 22 layers in total.

Szegedy et al. "Going Deeper with Convolutions", CVPR 2015



GoogLeNet (2)

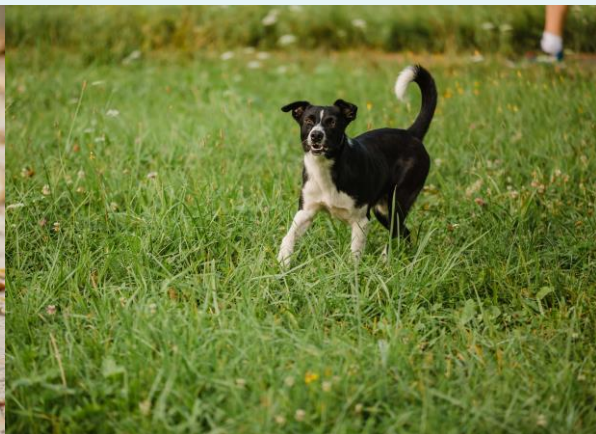
- To achieve efficiency, reduce size aggressively at the beginning.
- E.g. first few layers:

Layer	Size	Stride	Pad	Output Size
Input Image	3 x 224 x 224			3 x 224 x 224
Conv + ReLU	64 x (3 x 7 x 7)	2	3	64 x 112 x 112
Pool	3 x 3	2	1	64 x 56 x 56
Conv + ReLU	64 x (64 x 1 x 1)	1	0	64 x 56 x 56
Conv + ReLU	192 x (64 x 3 x 3)	1	1	192 x 56 x 56
Pool	3 x 3	2	1	192 x 28 x 28

- Compare with VGG16, after 5 layers, the output size was 118 x 112 x 112.

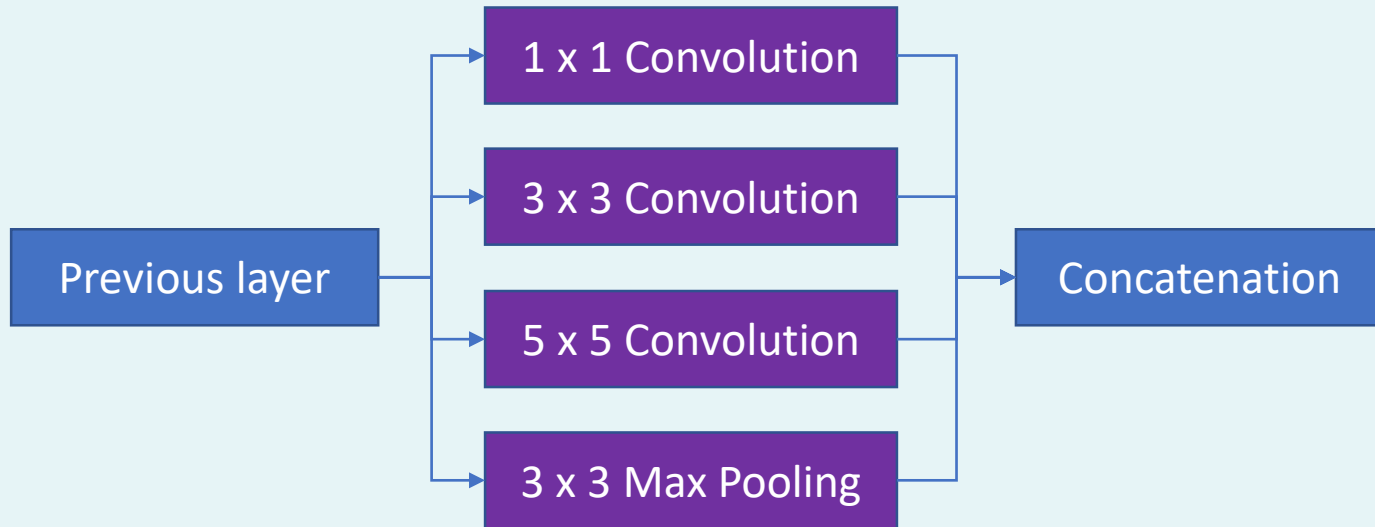
GoogLeNet (3)

- GoogLeNet also uses the “inception module”, to handle objects at different scales.
- E.g. objects at different scales:
 - Hard to get a filter which works well for all!



GoogLeNet (4)

- The “inception module” helps to solve the problem by using **filters of different sizes** at the **same level**:



- The network becomes **wider** instead of deeper.

ResNet (1)

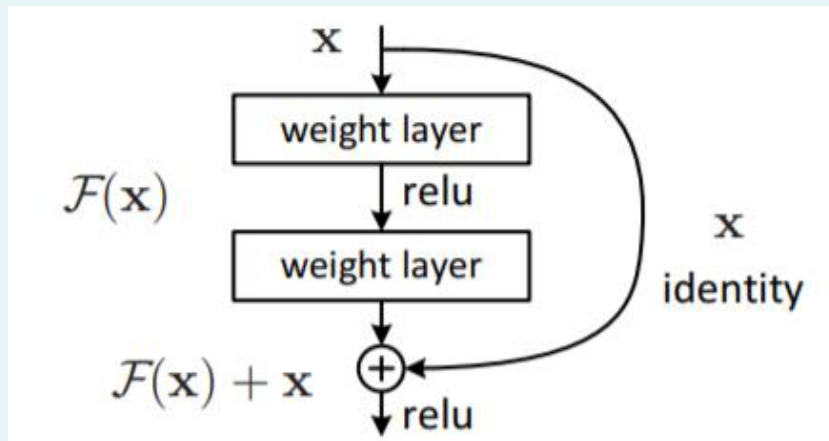
- ResNet, by He et al., 2015:
 - Performance by ResNet on ImageNet: 3.6%
 - Number of layers: 152!
 - Some background: It was initially observed that the deeper the network, the better the accuracy becomes. (AlexNet 8 → GoogLeNet 22).
 - Researchers therefore thought that by simply increasing depth, even better accuracy can be obtained.
 - But alas, that is not the case!

ResNet (2)

- The worse performance by deeper network doesn't only show up on test sets, but also on the training set!
- This suggests that the problem is not of overfitting, but something else.
- Now, deeper networks should theoretically be at least as good as shallower networks, because the “redundant” layers could simply learn to approach identity.
- Since this is not the case (i.e. extra layers do not become identity), researchers hypothesized that it is an **optimization problem**.

ResNet (3)

- The solution is to create some “shortcuts” for learning of identity.
- They proposed the following “Residual Block”.

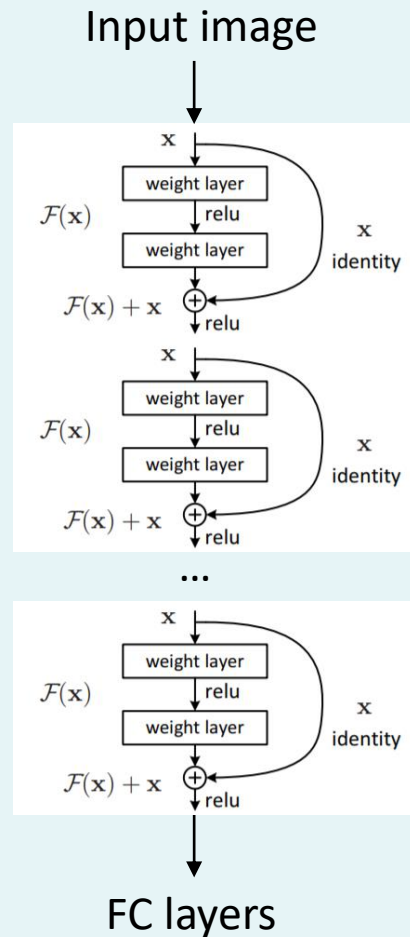


He et al. “Deep Residual Learning for Image Recognition”, CVPR 2016.

- If the weight layers are zero, the block will give identity.

ResNet (4)

- The build-up of the complete ResNet is similar to VGG – Use the **Residual Block repeatedly**.
- Each convolution filter is 3 x 3.
- With this structure, the network becomes very deep (e.g. 152) while achieving very high accuracy (error 3.6%).

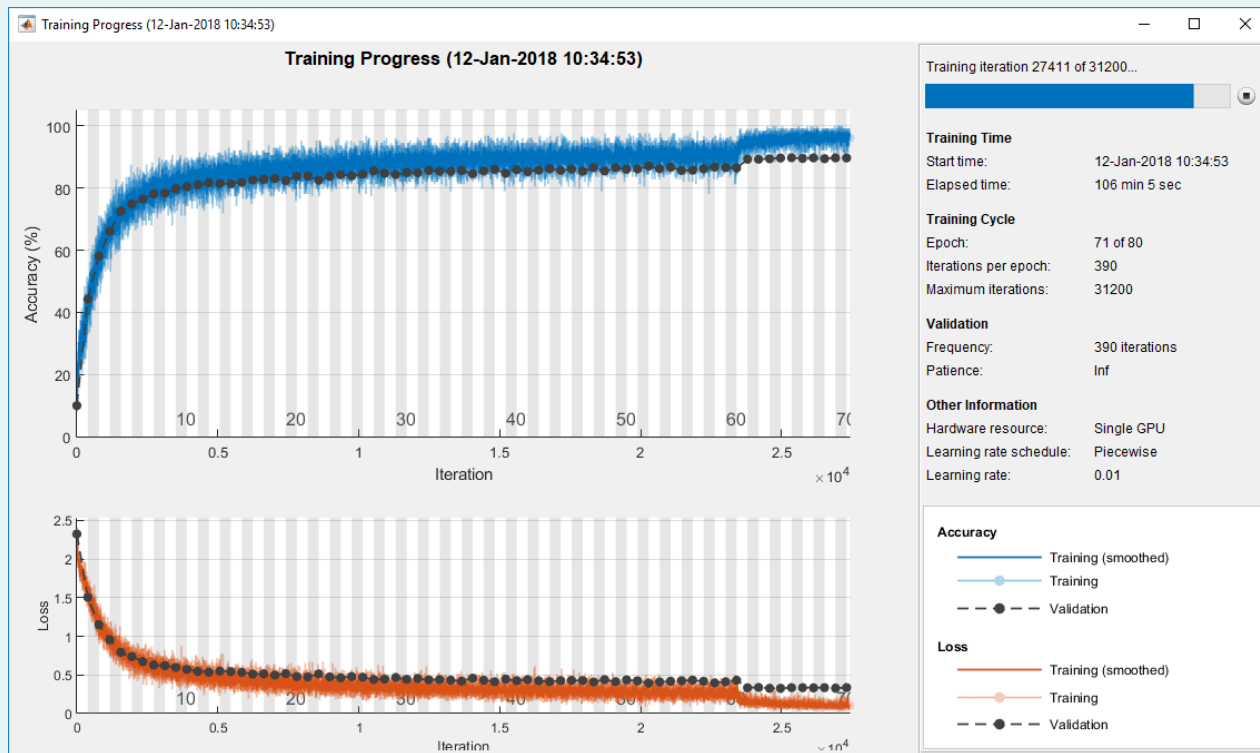


MATLAB ResNet on CIFAR-10 (1)

- The training of larger and deeper networks becomes significantly longer without GPU.
- As such, we will directly look at an example from Matlab's documentation page:
<https://www.mathworks.com/help/deeplearning/ug/train-residual-network-for-image-classification.html>
- ResNet on CIFAR-10 data.

MATLAB ResNet on CIFAR-10 (2)

- Results:
 - Validation accuracy close to 90%



Python ResNet (1)

- For coding of ResNet in Python, here is a good example:
 - <https://machinelearningknowledge.ai/keras-implementation-of-resnet-50-architecture-from-scratch/>
 - Please ignore the paragraph section “Quick Concept about Transfer Learning”, in between code blocks “In [12]” and “In [13]”.
 - Please also ignore code blocks “In [16]”, “In [17]” and “in [18]”.
 - This means, you should jump straight from “In [15]” to “In [19]”.
 - We will discuss about transfer learning in the next section.

Content

- Introduction to Image Classification
- Commonly-Used Datasets
- MLP for Image Classification
- History of Convolutional Neural Network (CNN)
- CNN in Details
- MATLAB & Python Examples
- Examples of CNN Architectures
- **Transfer Learning**

Transfer Learning (1)

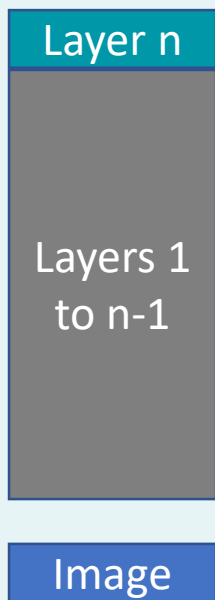
- Assume that you have already trained a CNN model to classify x number of classes.
- You are then given a new set of images belonging to a new class ($x+1$).
- Do you need to add the new images into the training set and retrain the whole model again?

Transfer Learning (2)

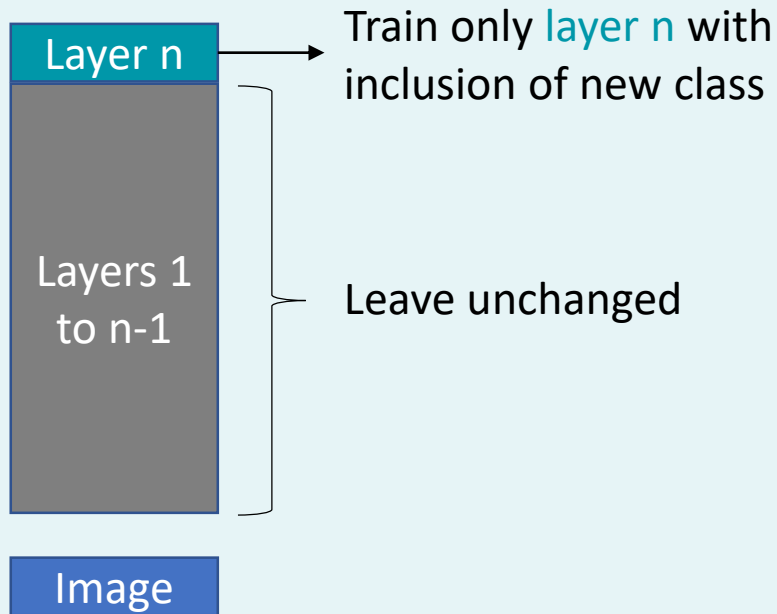
- Fortunately not! 😊
- We can take the **already-trained network**, leave the 1st layer up until the 2nd last layer as they are (no changes), and **re-train only the final layer**.
- This method is called “**Transfer Learning**”.
 - The 1st layer until the 2nd last layer acts as “**feature extractor**”,
 - Whereas the final layer re-learns the **classification**.

Transfer Learning (3)

- In pictorial form:

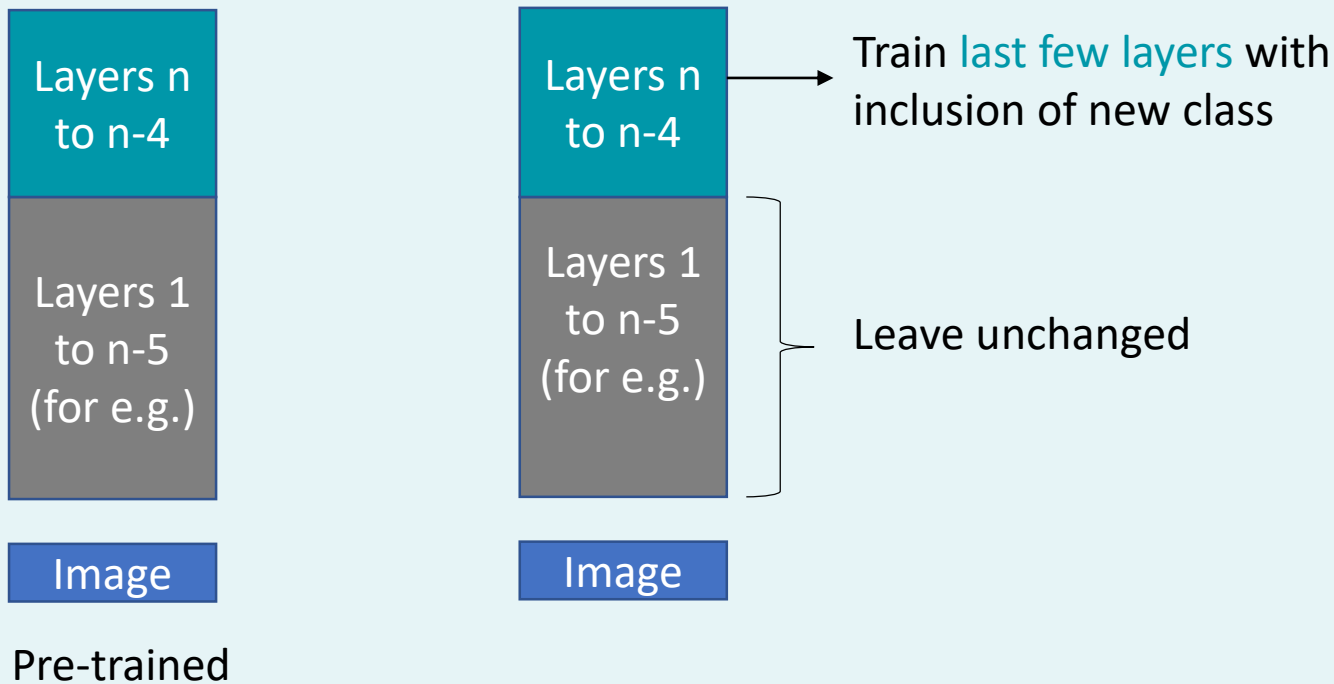


Pre-trained



Transfer Learning (4)

- If you have more data, you may train a few more layers.



Transfer Learning (5)

- Note: It is common to use a network which is pre-trained on [ImageNet](#) (instead of CIFAR-10), as there are more images, more classes, and higher resolutions in the former.
 - This gives a better starting point for transfer learning.

MATLAB Transfer Learning Example (1)

- Example: In ImageNet, there are some flowers (such as daisy) but there are not too many categories.
- Here, we want to create a network to classify 12 categories of flowers, including bluebell, buttercup, crocus, daffodil, daisy, dandelion, iris, lilyvalley, pansy, snowdrop, sunflower and tulip.
- We will perform transfer learning using a pre-trained GoogLeNet.

MATLAB Transfer Learning Example (2)

- Get training images:

```
flowerds = imageDatastore("Flowers","IncludeSubfolders",true,"LabelSource","foldernames");
```

- Split into training and testing sets:

```
[trainImgs,testImgs] = splitEachLabel(flowerds,0.6);
```

- Determine the number of flower species:

```
numClasses = numel(categories(flowerds.Labels));
```

MATLAB Transfer Learning Example (3)

- Create a network by modifying GoogLeNet.

```
net = googlenet;  
lgraph = layerGraph(net)
```

- **Modify** the classification and output layers:

```
newFc = fullyConnectedLayer(12,"Name","new_fc")  
lgraph = replaceLayer(lgraph,"loss3-classifier",newFc)  
newOut = classificationLayer("Name","new_out")  
lgraph = replaceLayer(lgraph,"output",newOut)
```

MATLAB Transfer Learning Example (4)

- Lower the learning rate for transfer learning:

```
options = trainingOptions("sgdm","InitialLearnRate", 0.001);
```

- Perform training:

```
[flowernet,info] = trainNetwork(trainImgs, lgraph, options);
```

Training on single CPU.

Initializing input data normalization.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Mini-batch Loss	Base Learning Rate
1	1	00:00:25	7.03%	5.1655	0.0010
13	50	00:20:58	96.88%	0.1043	0.0010
25	100	00:45:23	100.00%	0.0111	0.0010
30	120	00:56:21	100.00%	0.0088	0.0010

MATLAB Transfer Learning Example (5)

- Use the trained network to classify test images:

```
testpreds = classify(flowernet,testImgs);
```

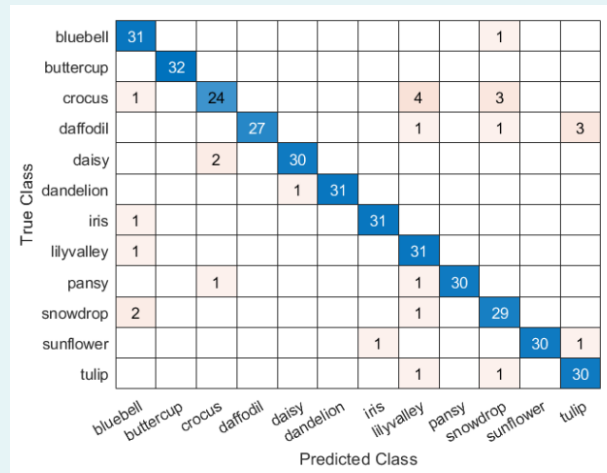
- Evaluate the results – Calculate the accuracy:

```
nnz(testpreds == testImgs.Labels)/numel(testpreds)
```

ans = 0.9271

- Visualize the confusion matrix:

```
confusionchart(testImgs.Labels,testpreds);
```



Python Transfer Learning Example (1)

- You may refer to the same Python example earlier on ResNet, but without ignoring “In [16]”, “In [17]” and “in [18]”.
 - <https://machinelearningknowledge.ai/keras-implementation-of-resnet-50-architecture-from-scratch/>
- Another more-direct example:
 - <https://chroniclesofai.com/transfer-learning-with-keras-resnet-50/>

Free MATLAB DL On-Ramp Course (1)

- What you have learnt in the lecture was the concept and theory of deep learning and CNN.
- In practice, there are some **subtleties to handle** e.g.:
 - Managing **collections of image** data.
 - Pre-processing images which are of “**wrong**” **sizes** (e.g. GoogLeNet takes in square images of size 224 x 224 only).
 - Getting **labels** of images.
 - **Splitting data** into training, validation and test sets.
 - Obtaining **pre-trained networks** from the internet.

Free MATLAB DL On-Ramp Course (2)

- You are thus strongly encouraged to take the **free Matlab Deep Learning On-Ramp** course, which can be completed in 2 hours.
- You will also receive a **certificate** for completing the course.
- <https://www.mathworks.com/learn/tutorials/deep-learning-onramp.html>
- Sign in to your **Matlab account** (registered using your UCL email address).
- We **may** be able to do this during the workshop this week.

Thank you for your attention!

Any questions?