Convolutional Neural Networks

Introduction to Artificial Intelligence with Mathematics
Lecture Notes

Ganguk Hwang

Department of Mathematical Sciences
KAIST

Convolutional Neural Network (CNN)

- Convolutional Neural Networks (CNNs) are very similar to ordinary Artificial Neural Networks, but
- CNNs take in an input image, learn the weights and biases for various aspects in the image, and differentiate the image from the other images.
- CNNs are shift/space invariant artificial neural networks by sharing the weights (filters) and can successfully capture local features of input images.
- CNNs reduces the image into a form which is easier to process without losing local features which are critical for image classification and prediction.
- CNNs have applications in image/video recognition and classification, image segmentation, medical image analysis, and natural language processing.

Note that

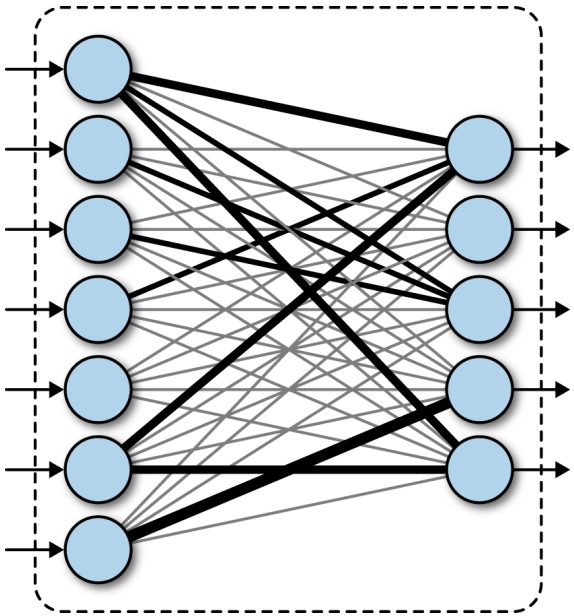
- Images are very high dimensional, e.g., have millions of pixels.
- There are a number of variants for images in, e.g., shape and space.
- It is very difficult to learn images by an artificial neural network that uses all pixel information as an input (e.g., a fully connected network) due to many parameters to learn and not using the spatial structure.

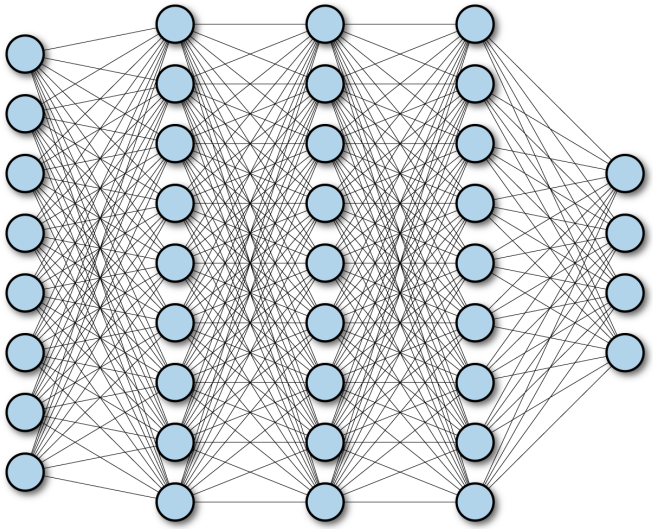
Question: How can we use the spatial structure in the image to learn an artificial neural network?

We can use locally connected layers and the replicated feature approach, which is the motivation of CNNs.

Fully Connected Layer and Network

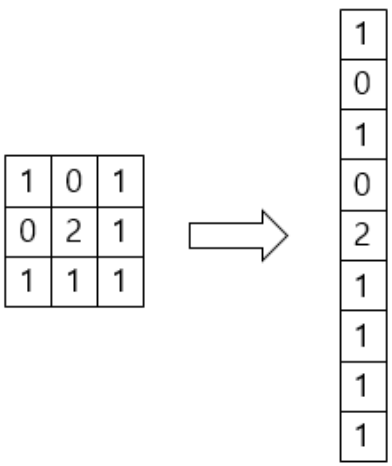
- Many parameters to learn

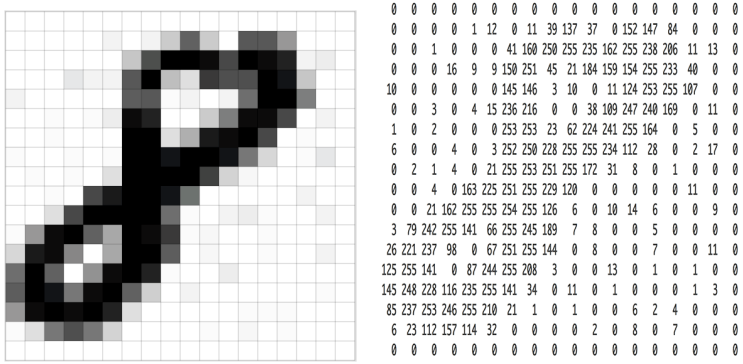




source: R. B. Zadeh, B. Ramsundar, TensorFlow for Deep Learning, O'Reilly Media, Inc., March 2018.

- No spatial information can be used.
  - An image is nothing but a matrix of pixel values.
  - The matrix of pixel values is flattened to be a sequence of pixel values which is fed to a fully connected layer.

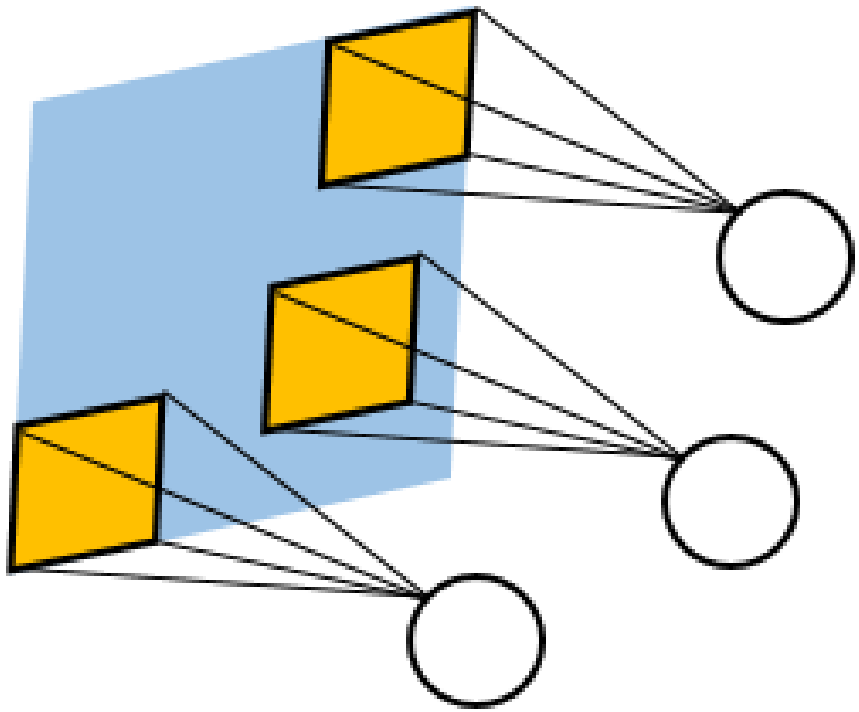




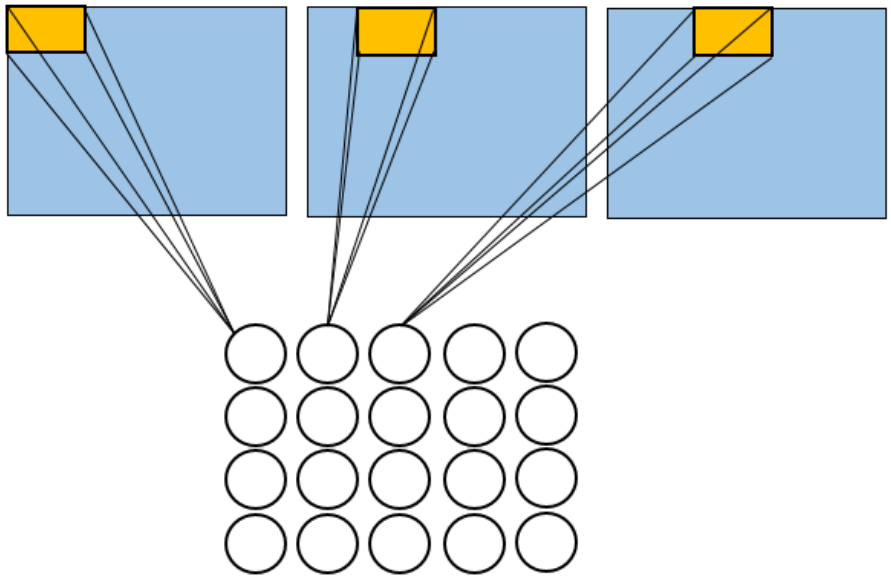
source: https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721

Locally Connected Layer

- The pixel values in an image are dependent, which contain the spatial structure.
- So, to use the spatial structure, each neuron in a hidden layer is connected to a small input patch.



- We use a sliding window to define connections.
- How can we weight the patch to extract proper features?
- To determine the set of weights, called a filter, learning is necessary.

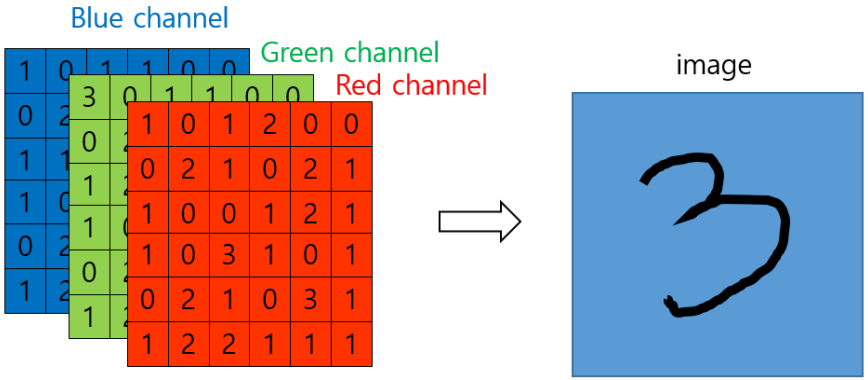


In CNNs, we need the following:

- channel
- convolution
- filter (or kernel)
- stride
- padding
- feature map
- pooling

Channels

- A color image consists of three channels (Red, Green, and Blue channels)



The Convolution Operation and Feature Map

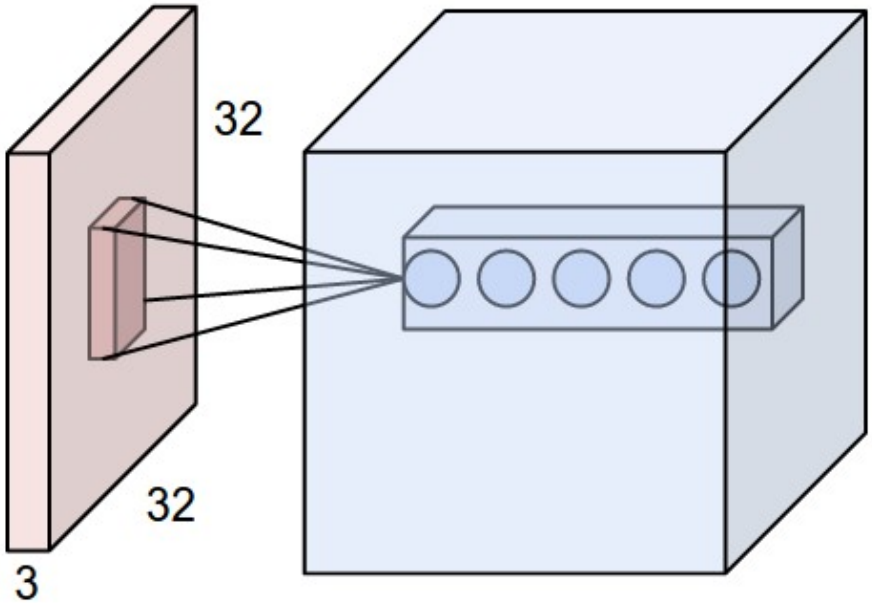
- element-wise multiplication (and bias addition) with sliding window

$$\begin{array}{|c|c|c|c|c|}\hline 1&1&1&0&0\\\hline 0&1&1&1&0\\\hline 0&0&1&1&1\\\hline 0&0&1&1&0\\\hline 0&1&1&0&0\\\hline\end{array} \odot \begin{array}{|c|c|c|}\hline 1&0&1\\\hline 0&1&0\\\hline 1&0&1\\\hline\end{array} +0 = \begin{array}{|c|c|c|}\hline 4&3&4\\\hline 2&4&3\\\hline 2&3&4\\\hline\end{array}$$

3×3 filter, bias 0, stride 1

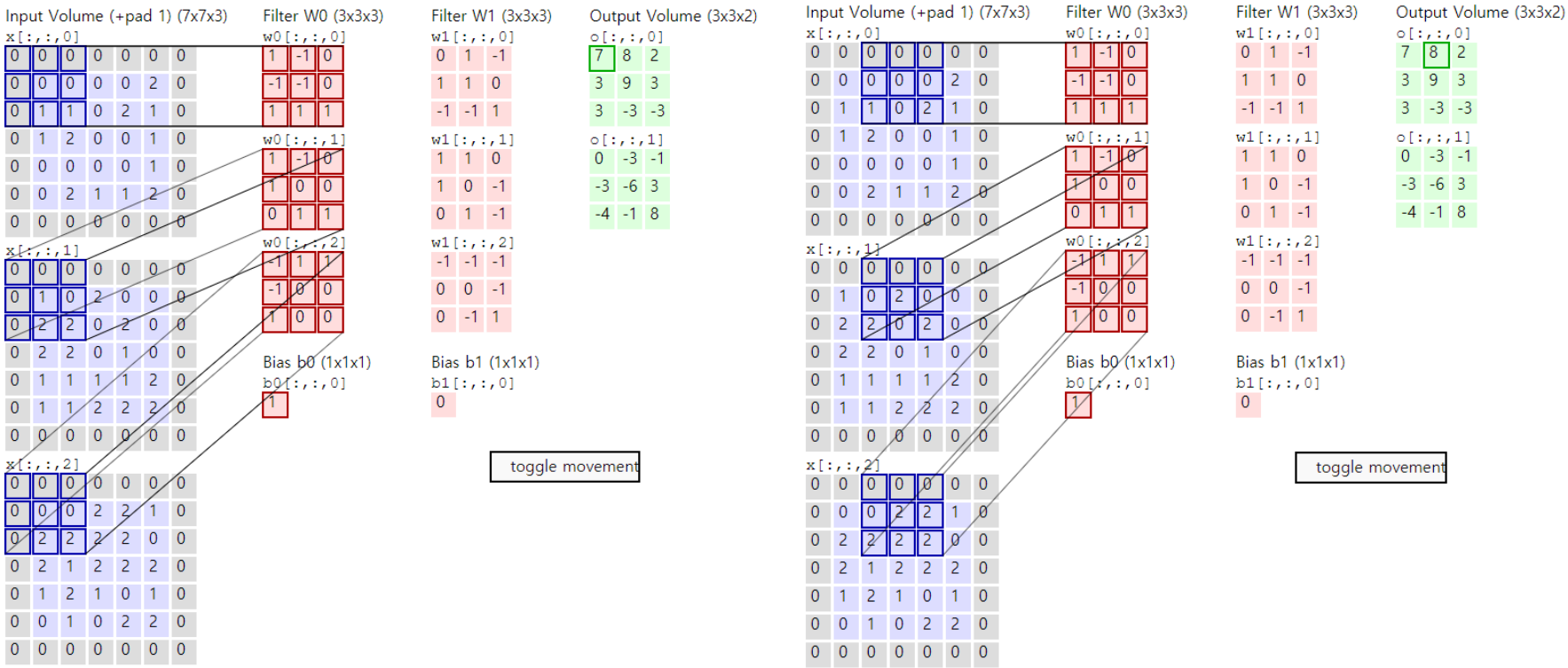
(input channel: 1, output channel: 1)

Example: Suppose that the input volume has size $[32 \times 32 \times 3]$ (e.g. an RGB CIFAR-10 image).



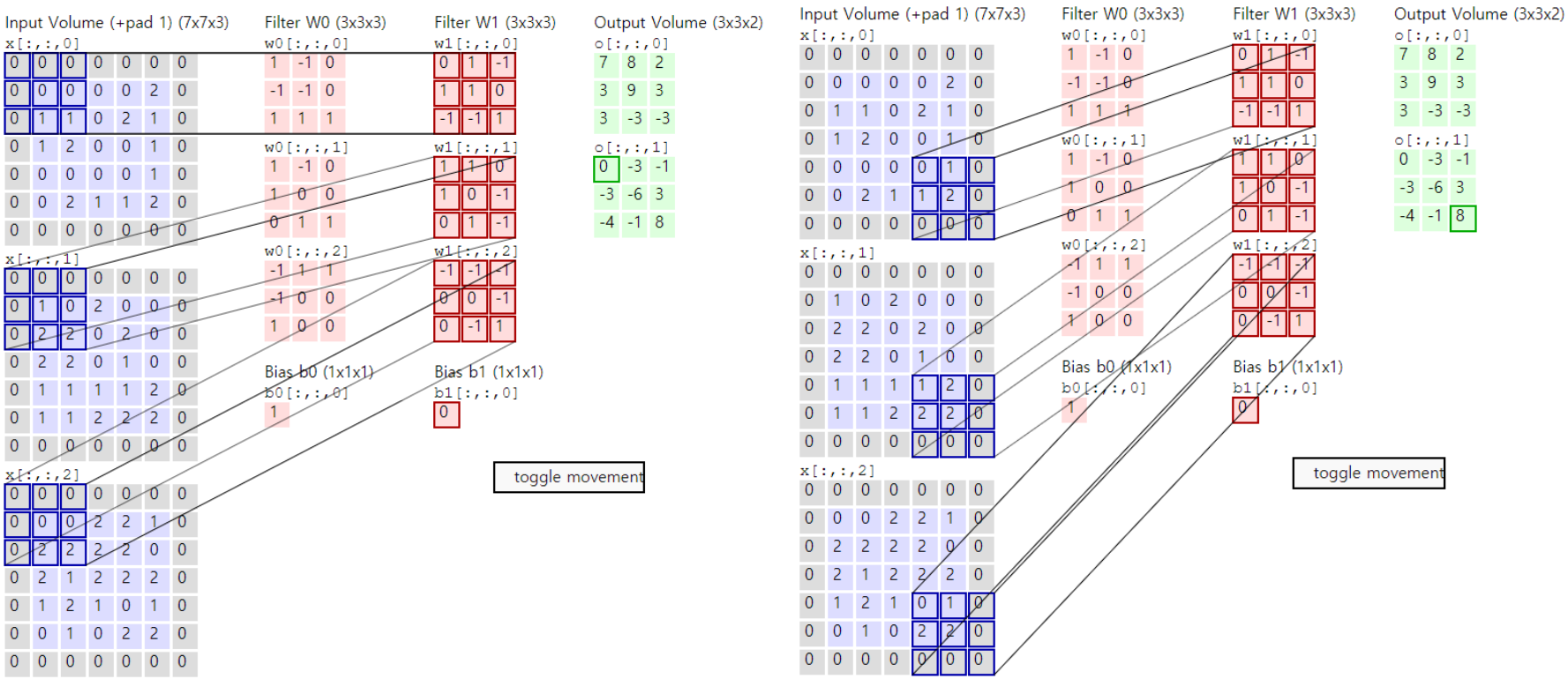
source: http://cs231n.github.io/convolutional-networks/

Filter size: 3×3, Stride: 2, Two Filters (W_0 and W_1), zero-padding

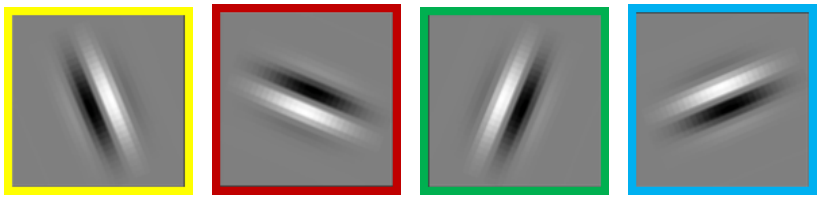


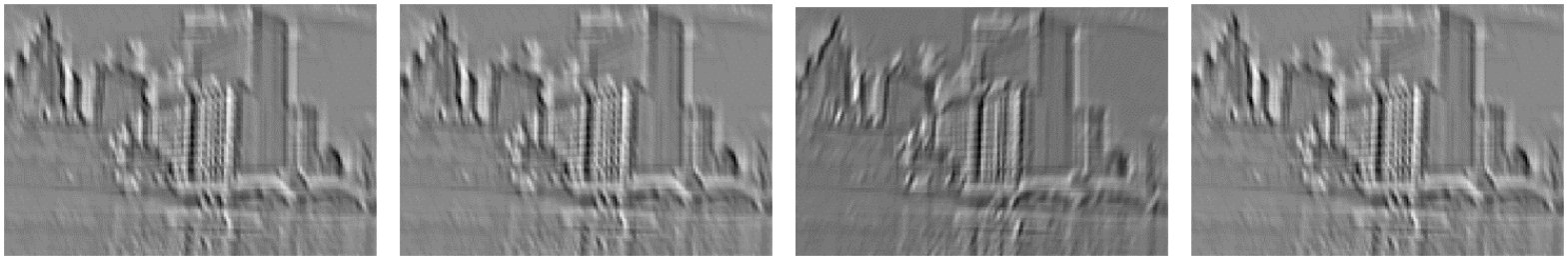
source: http://cs231n.github.io/convolutional-networks/
(input channel: 3, output channel: 2)

Filter size: 3×3, Stride: 2, Two Filters (W_0 and W_1), zero-padding



source: http://cs231n.github.io/convolutional-networks/
(input channel: 3, output channel: 2)

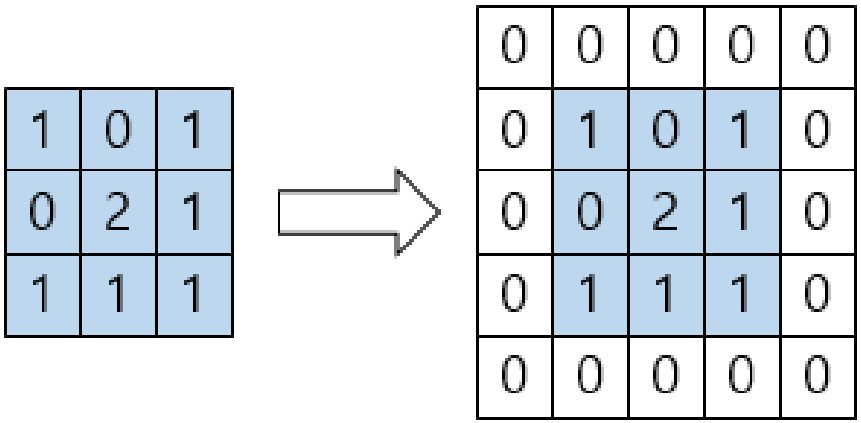




source: R. Fergus et. al., Deep Learning Methods for Vision, CVPR 2012 Tutorial

Padding

- The size of a feature map is always smaller than the size of the input data due to filter and stride.
- To avoid the reduction in size, we use padding.



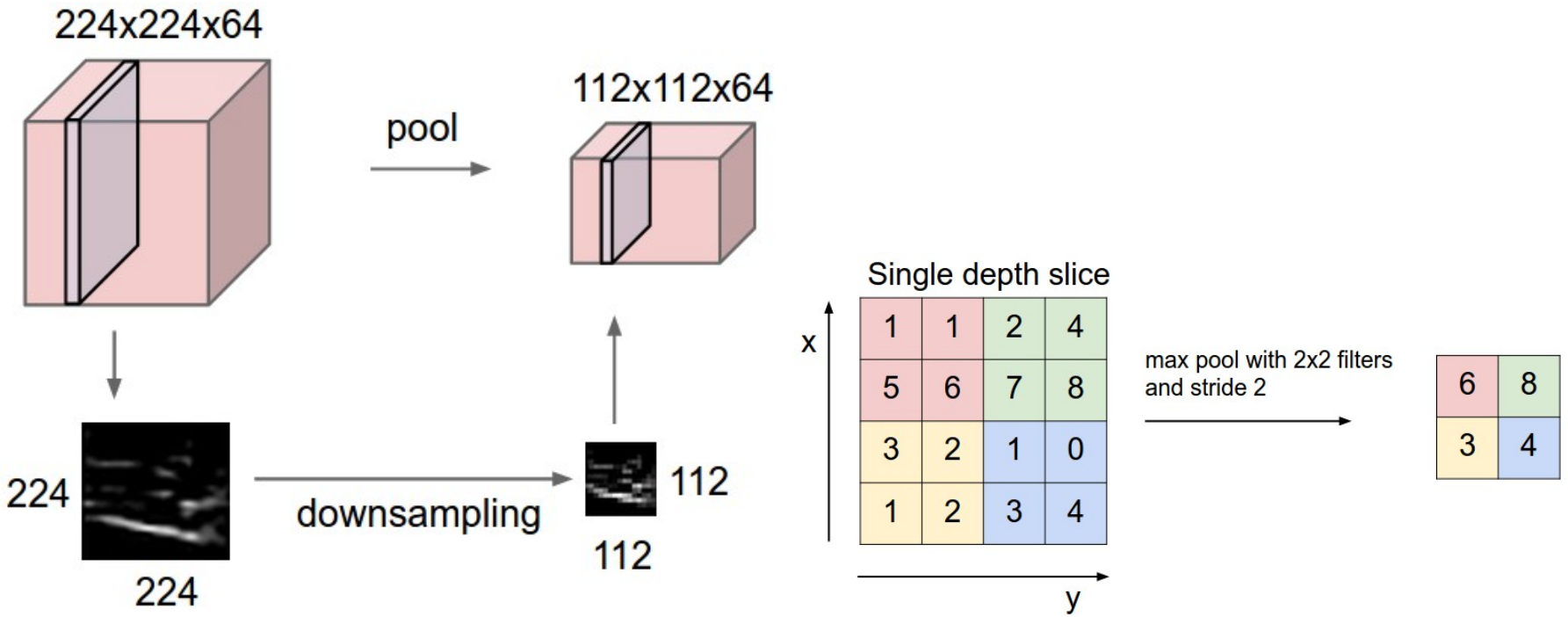
Non-linearity with activation function

- After every convolution operation, apply a given activation function.
- The ReLU function is often used as an activation function.
- When the ReLU function is used, all negative values are replaced by zero.
- The result after applying the activation function, is called an activation map.

Pooling

- Pooling reduces the spatial size of the output volume by the convolution layer.
- Max Pooling returns the maximum value from the portion of the output covered by the filter.
- Average Pooling returns the average of all the values from the portion of the output covered by the filter.
- It is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.
- The hyperparameters of pooling: the filter size and the stride

Example: Max Pooling

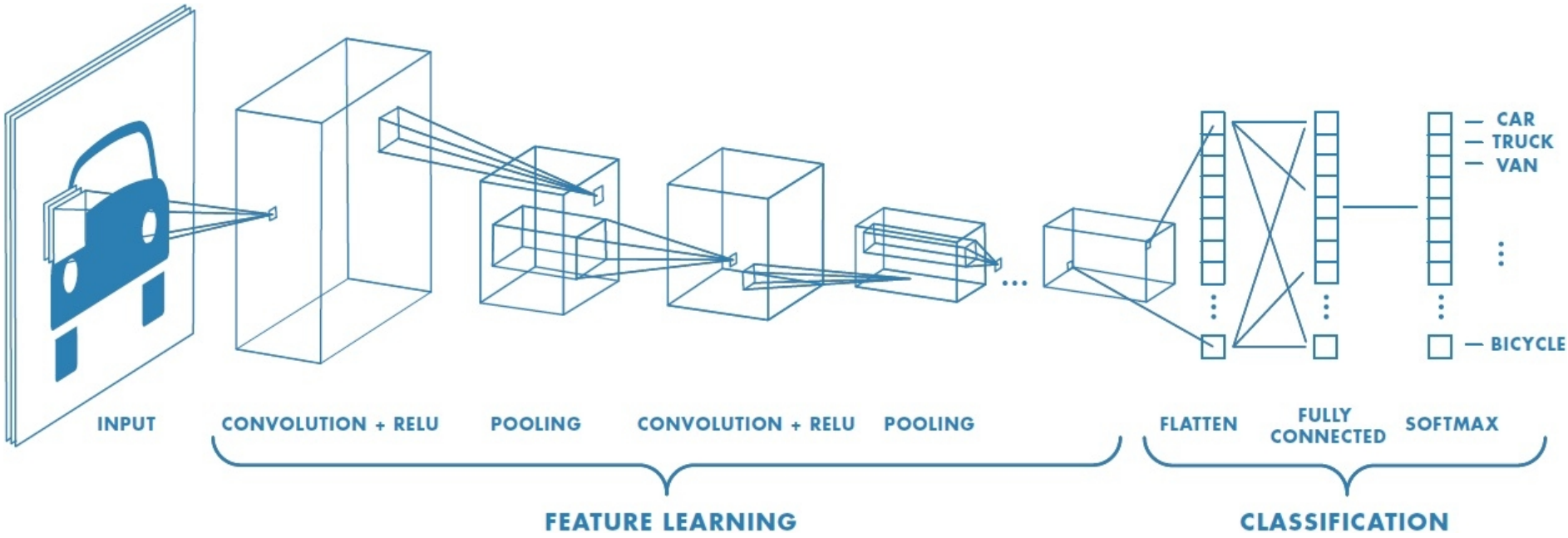


source: http://cs231n.github.io/convolutional-networks/

Backpropagation with Pooling

- The backward pass for a $\max(x_1, x_2, \cdots, x_n)$ operation has a simple interpretation as only routing the gradient to the input that had the highest value in the forward pass.
- Hence, during the forward pass of a pooling layer it is common to keep track of the index of the max activation (sometimes also called the switches).
- When average pooling is used, we multiply $\frac{1}{m}$ for the backward pass.

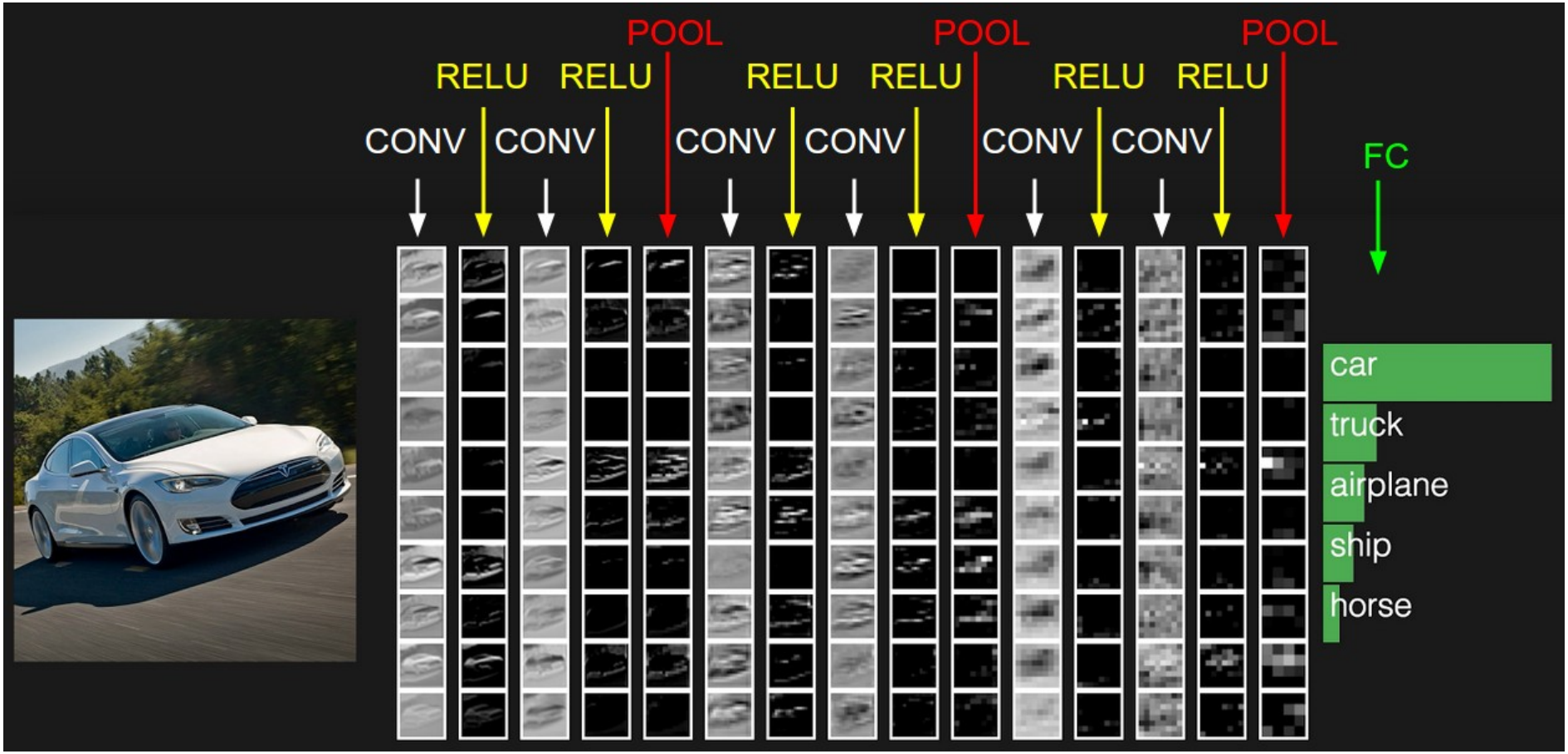
Architecture of a CNN for Classification



source: https://kr.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

Classification Through Fully Connected Layer

- First flatten the input into a column vector.
- The flattened column vector is fed to a feed-forward neural network.
- Classify using the softmax classification.



source: http://cs231n.github.io/convolutional-networks/

Example: CIFAR-10 Data Set

- We're going to use CNN for classifying CIFAR-10.
- CIFAR-10 has 60000 images and 10 labels.
- 0:airplane, 1:automobile, 2:bird, 3:cat, 4:deer, 5:dog, 6:frog, 7:horse, 8:ship and 9:truck
- The set of 50000 images is used for training and the set of the remaining images is used for validation.

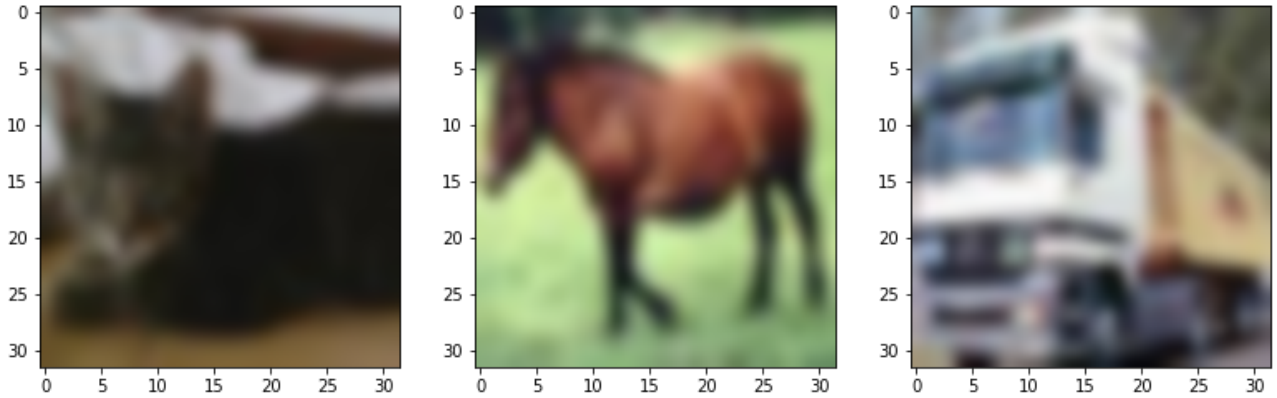


Figure: Samples from CIFAR-10 (3:cat, 7:horse, 9:truck)

CNN Structure

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 32)	896
activation_1 (Activation)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 30, 30, 32)	9248
activation_2 (Activation)	(None, 30, 30, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 15, 15, 32)	0
dropout_1 (Dropout)	(None, 15, 15, 32)	0
conv2d_3 (Conv2D)	(None, 15, 15, 64)	18496
activation_3 (Activation)	(None, 15, 15, 64)	0
conv2d_4 (Conv2D)	(None, 13, 13, 64)	36928
activation_4 (Activation)	(None, 13, 13, 64)	0
max_pooling2d_2 (MaxPooling2	(None, 6, 6, 64)	0
dropout_2 (Dropout)	(None, 6, 6, 64)	0
flatten_1 (Flatten)	(None, 2304)	0
dense_1 (Dense)	(None, 512)	1180160
activation_5 (Activation)	(None, 512)	0
dropout_3 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 10)	5130
activation_6 (Activation)	(None, 10)	0

Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0

Figure: The structure of CNN for training

CNN Structure

The shape of input datum is $(32, 32, 3)$. The number of filters and the filter size for each convolutional layer are as follows:

	number of filters	filter size
conv2d_1	32	(3, 3, 3)
conv2d_2	32	(3, 3, 32)
conv2d_3	64	(3, 3, 32)
conv2d_4	64	(3, 3, 64)

All activation functions except the last layer is 'relu'. The activation function for the last layer is 'softmax'. All strides are set to be the default value, 1.

CNN Structure (continued)

- Pooling is used twice. The max pooling is used and the filter size of it is $(2, 2)$.
- Dropout: randomly setting a fraction of input units to 0 at each update during training time, which helps prevent overfitting. Here, the fraction is set to be 0.25 and 0.5.
- The flatten layer flattens the inputs to make them the inputs of the fully connected layer.
- The dense layer means the fully connected layer.

The Loss Function

- The cross entropy is used for the loss function.
- Suppose that there are n data and C classes.
- $L \in \{0,1\}^C$ is a one-hot vector for true label and $P \in [0,1]^C$ is C-dimensional predicted probability vector.
- Then, the cross entropy loss function is defined as follows:

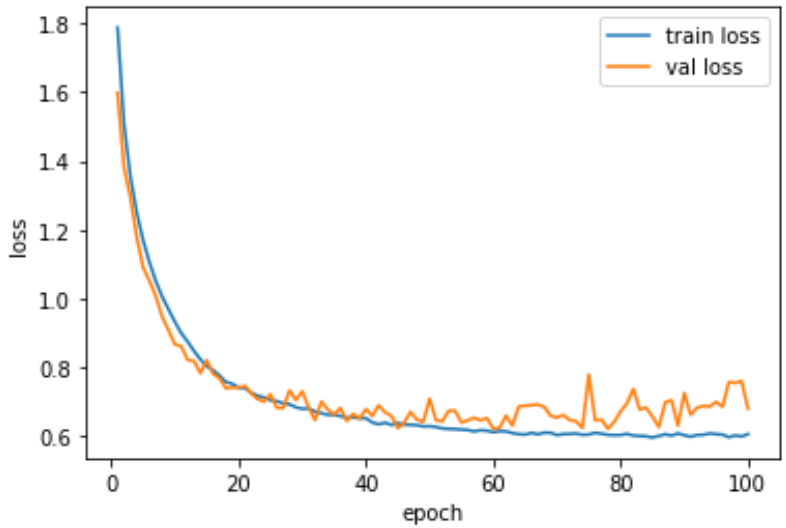
$$-\frac{1}{n}\sum_{i=1}^{n} L_i^T \log(P_i)$$

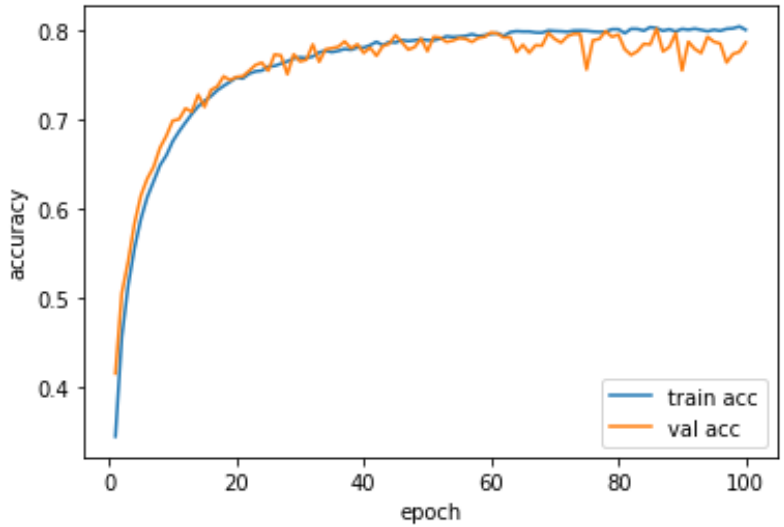
where L_i is the probability vector of the true label and P_i is the predicted probability vector of data i.

c.f. The cross entropy $= -E_{L_i}[\log(P_i)] = -L_i^T \log(P_i)$

Experiment Results

- The number of epochs is 100.
- The training loss and accuracy are 0.6064 and 0.7996, respectively.
- The validation loss and accuracy are 0.6806 and 0.7854, respectively.
- The figures given below plot the loss and accuracy, respectively.





Experiment Results

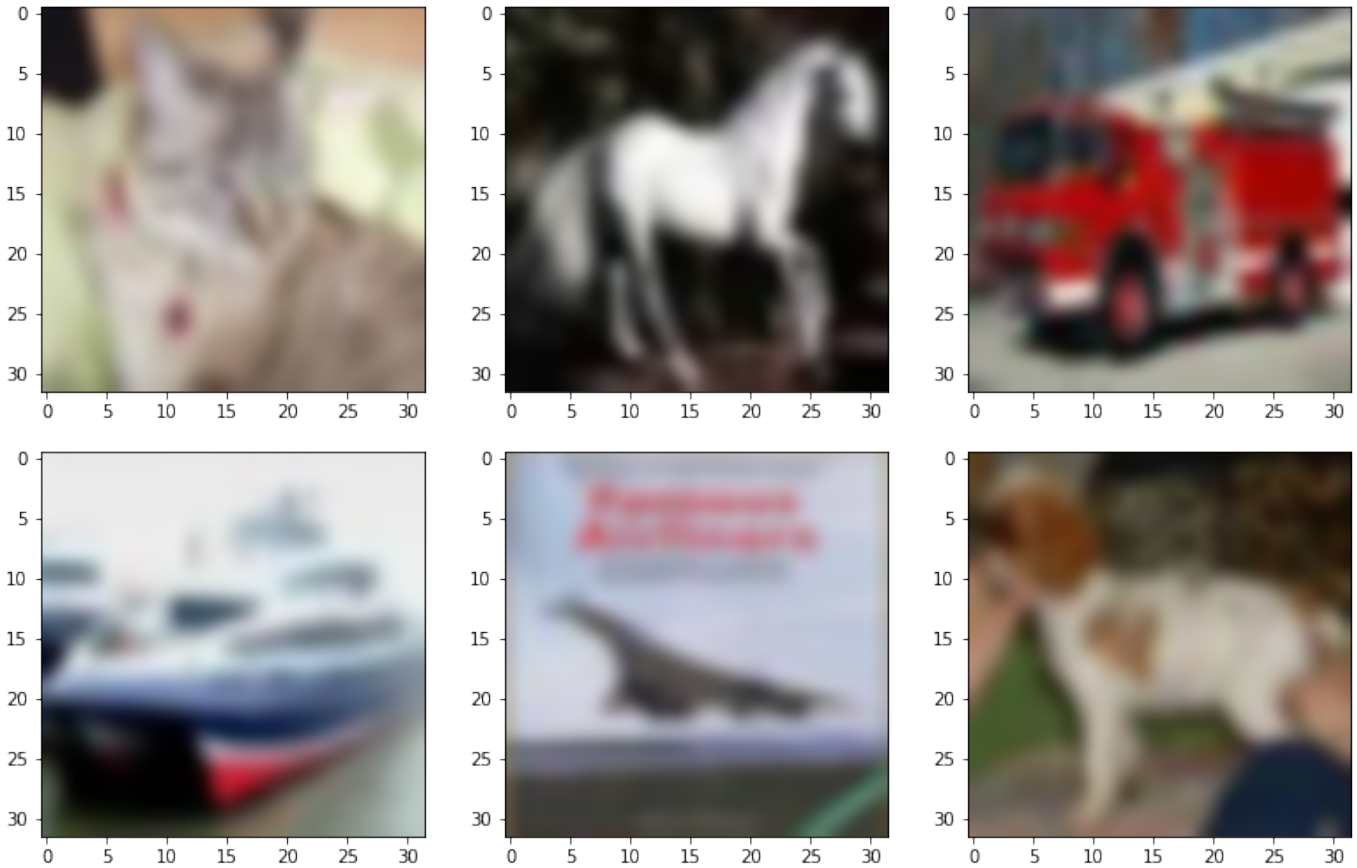


Figure: Prediction results. true label:prediction - cat:cat, horse:horse, truck:truck, ship:ship, airplane:ship, dog:cat