KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

AN AUTONOMOUS INSTITUTION - ACCREDITED BY NAAC WITH 'A' GRADE

Narayanaguda, Hyderabad.

CONVOLUTIONAL NEURAL NETWORK

OBJECTIVE

- Introduction to Neural Network
- Motivation
- Convolutional Neural Network
- CNN Architecture
- CNN Layers
- Implementation
- Exercises

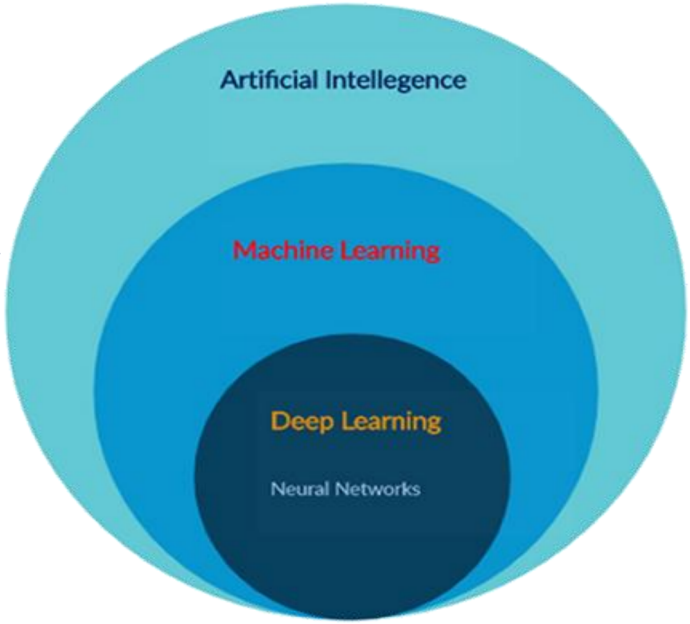
NEURAL NETWORKS

- Neural network is a subfield of **machine learning**

- Neural networks take input data, train themselves to recognize patterns found in the data, and then predict the output for a new set of similar data

- Neural networks, a beautiful biologically-inspired programming paradigm which enables a computer to learn from observational data

- Deep learning, a powerful set of techniques for learning in neural networks



NEURAL NETWORKS ARCHITECTURE

•First layer taking in inputs and the last layer producing outputs.

• The middle layers have no connection with the external world, and hence are called hidden layers.

• Each perceptron in one layer is connected to every perceptron on the next layer.

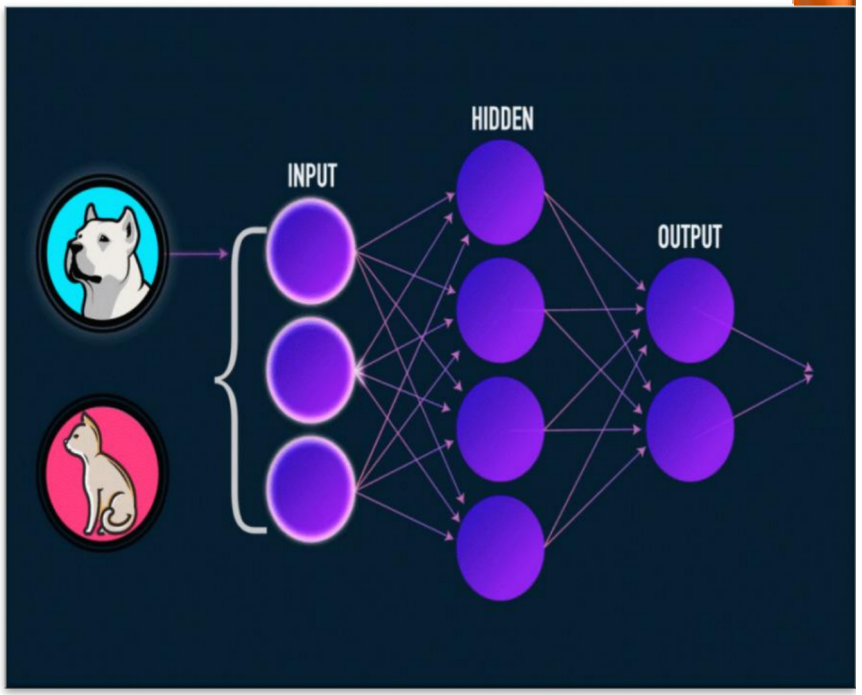
• output:

$$Output = w_1 * i_1 + w_2 * i_2 + w_3 * i_3 + \cdots + w_n * i_n + b$$

Where,

$w_i = weight\ for\ input\ i_i$

$b = bias * weight\ for\ bias$

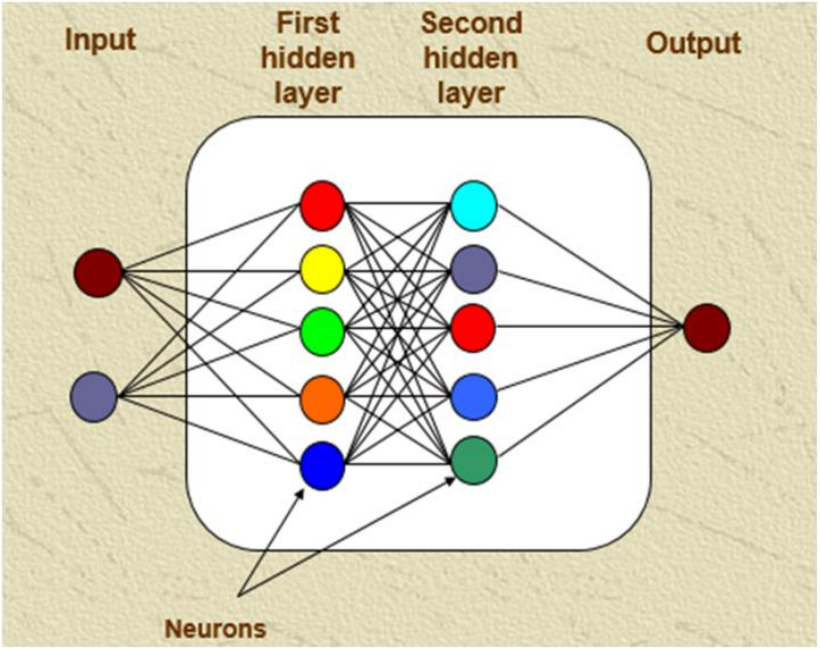


MULTILAYER NEURAL NETWORKS

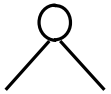
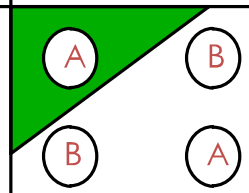
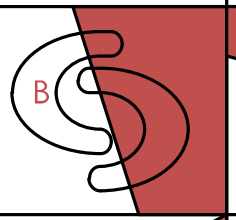
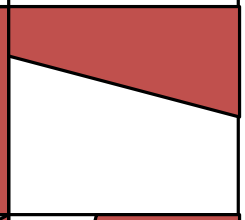
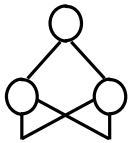
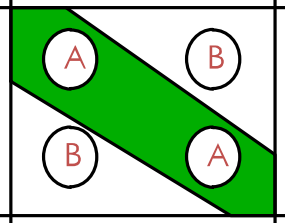
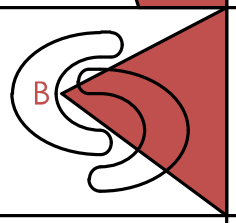
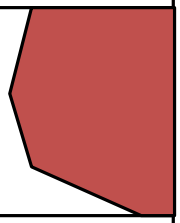
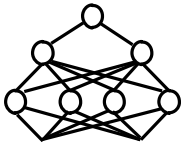
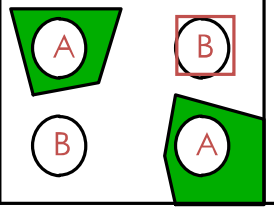
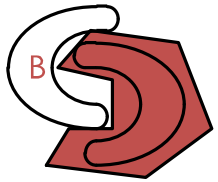
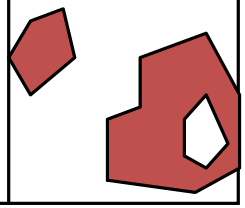
- For computer vision, why can't we just flatten the image and feed it through the neural networks?

- Images are high-dimensional vectors.

- It would take a huge amount of parameters to characterize the network.

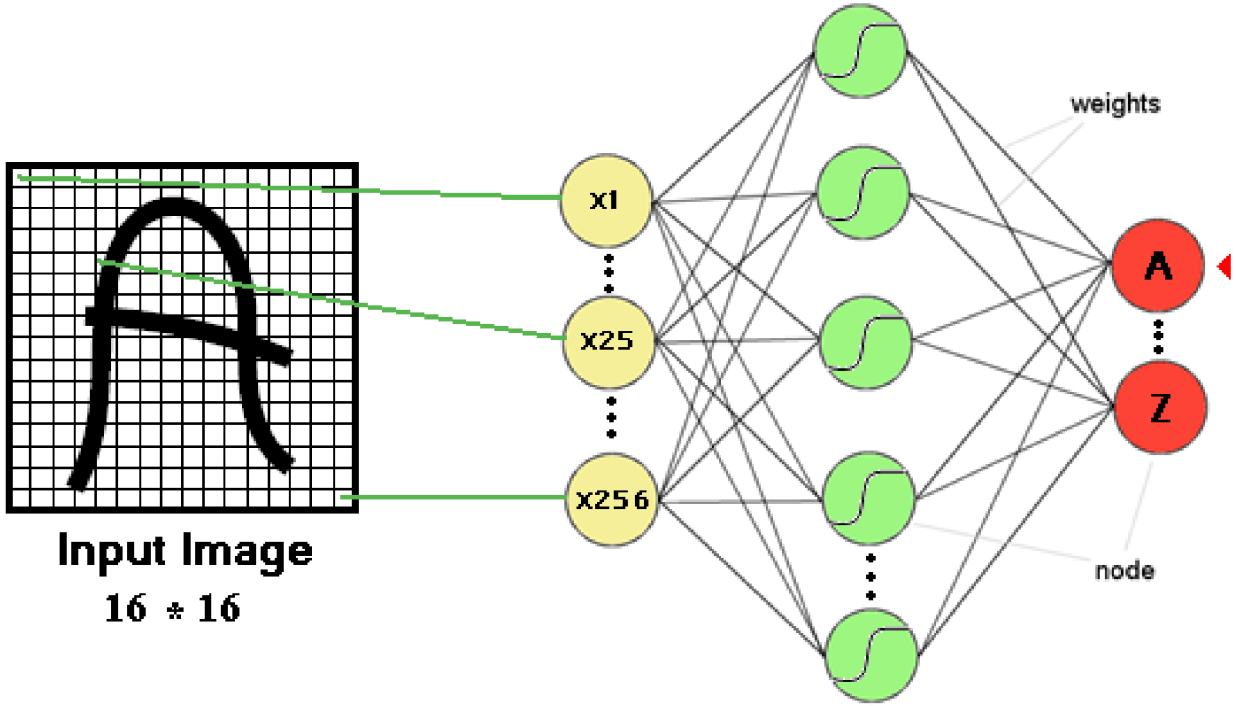


BEHAVIOR OF MULTILAYER NEURAL NETWORKS

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer 	Half Plane Bounded By Hyper plane			
Two-Layer 	Convex Open Or Closed Regions			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			

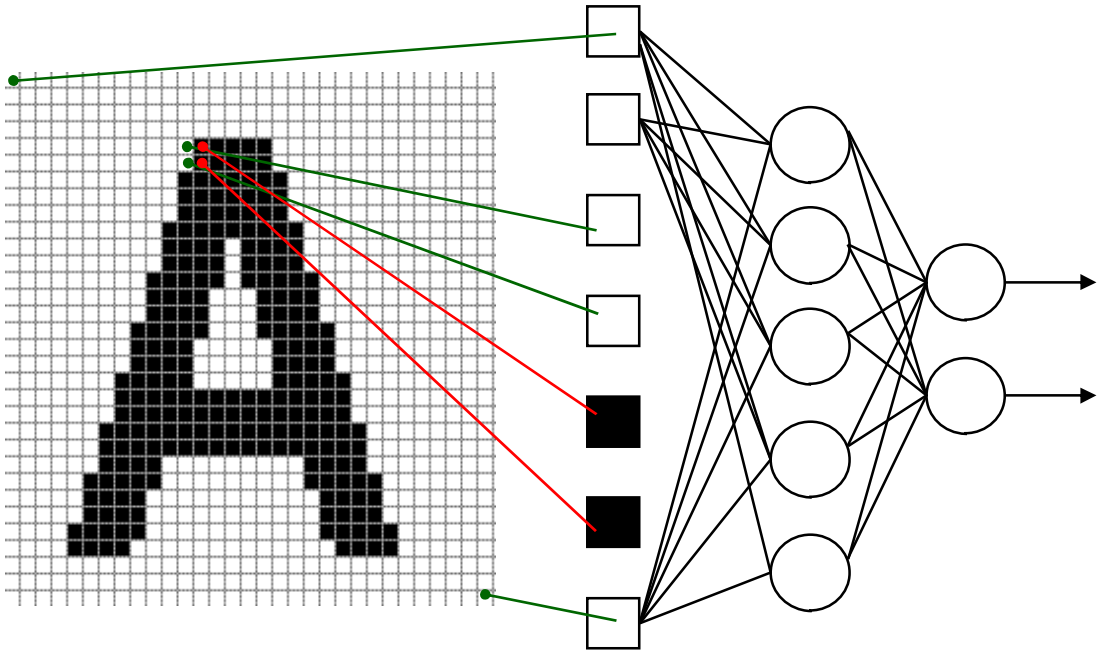
MULTI-LAYER PERCEPTRON AND IMAGE PROCESSING

- One or more hidden layers Sigmoid activations functions



DRAWBACKS OF PREVIOUS NEURAL NETWORKS

- The number of trainable parameters becomes extremely large
- Little or no invariance to shifting, scaling, and other forms of distortion



HISTORY OF CNN

- Yann LeCun, Professor of Computer Science
  The Courant Institute of Mathematical Sciences
  New York University
- Room 1220, 715 Broadway, New York, NY 10003, USA.
  (212)998-3283 yann@cs.nyu.edu

- In 1995, Yann LeCun and Yoshua Bengio introduced the concept of convolutional neural networks.

https://towardsdatascience.com/convolution-neural-networks-a-beginners-guide-implementing-a-mnist-hand-written-digit-8aa60330d022

https://towardsdatascience.com/convolutional-neural-networks-mathematics-1beb3e6447c0

https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9

TERMINOLOGIES USED IN CNN

•**Filter, Kernel**, or **Feature Detector** is a small matrix used for features detection.

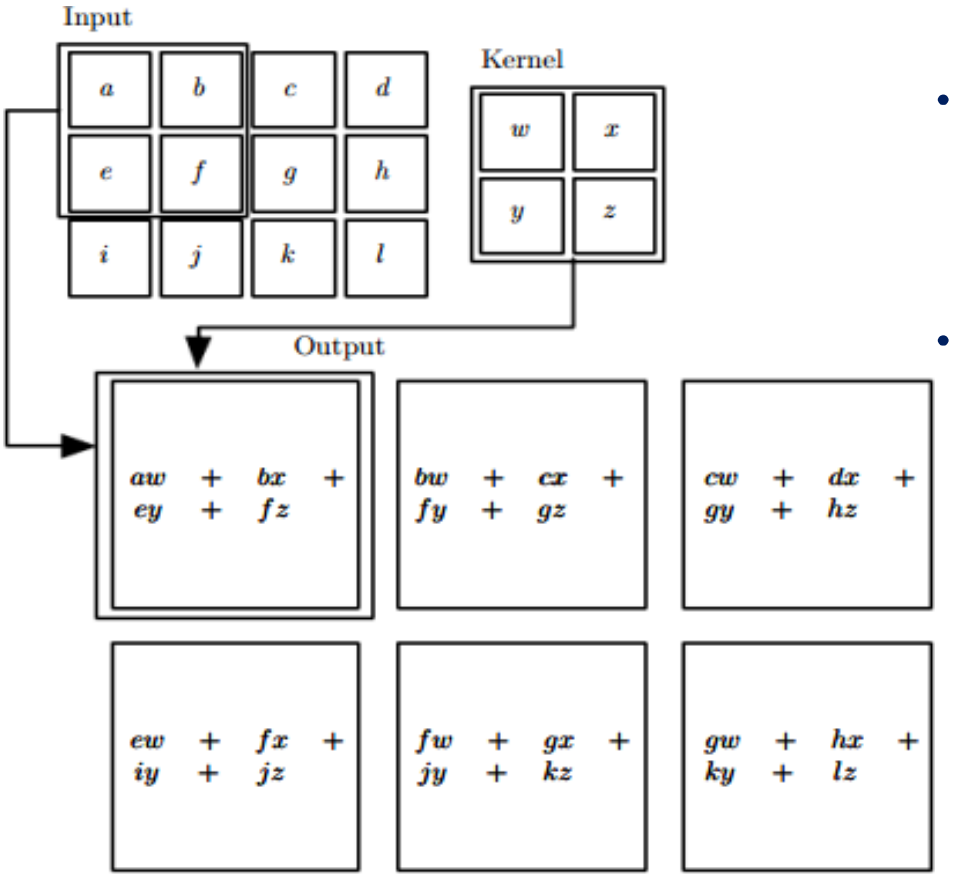
• **Convolved Feature**, **Activation Map** or **Feature Map** is the output volume formed by sliding the filter over the image and computing the dot product.

• **Receptive field** is a local region of the input volume that has the same size as the filter.

• **Depth** is the number of filters.

• **Stride** has the objective of producing smaller output volumes spatially.

WHY CONVOLUTION ?



A 2D convolution example from deep learning

- Preserves the spatial relationship between pixels by learning image features using small squares of input data
- Detect small,meaningful features such as edges with kernels

WHAT IS CONVOLUTION NEURAL NETWORK

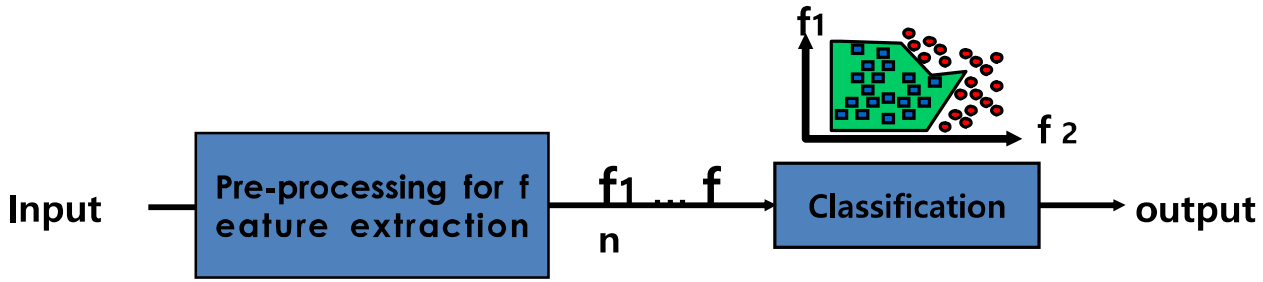
- CNN is a specualized kind of neural network for processing data that has a known, grid-like topology, such as time- series(1D grid), image data(2D grid), etc.
- CNN is a supervised deep learning algorithm, it is used in various fields like speech recognition, image retrieval and face recognition.
- CNN's Were **neurobiological** motivated by the findings of locally sensitive and orientation-selective nerve cells in the visual cortex.
- A network structure designed to extracts relevant features.
- Convolutional Neural Networks are a special kind of **multi-layer neural networks**.

CONVOLUTION NEURAL NETWORK

- CNN is a **feed-forward** network that can extract topological properties from an image.
- Every neural networks they **are trained** with a version of the **back-propagation algorithm.**
- CNNs are designed to **recognize visual patterns** directly from pixel images with minimal preprocessing.
- They can recognize patterns with extreme variability (such as handwritten characters).
- Convolutional neural networks are usually composed by a set of layers that can be grouped by their functionalities.
- Classification can be done with the extraction of features

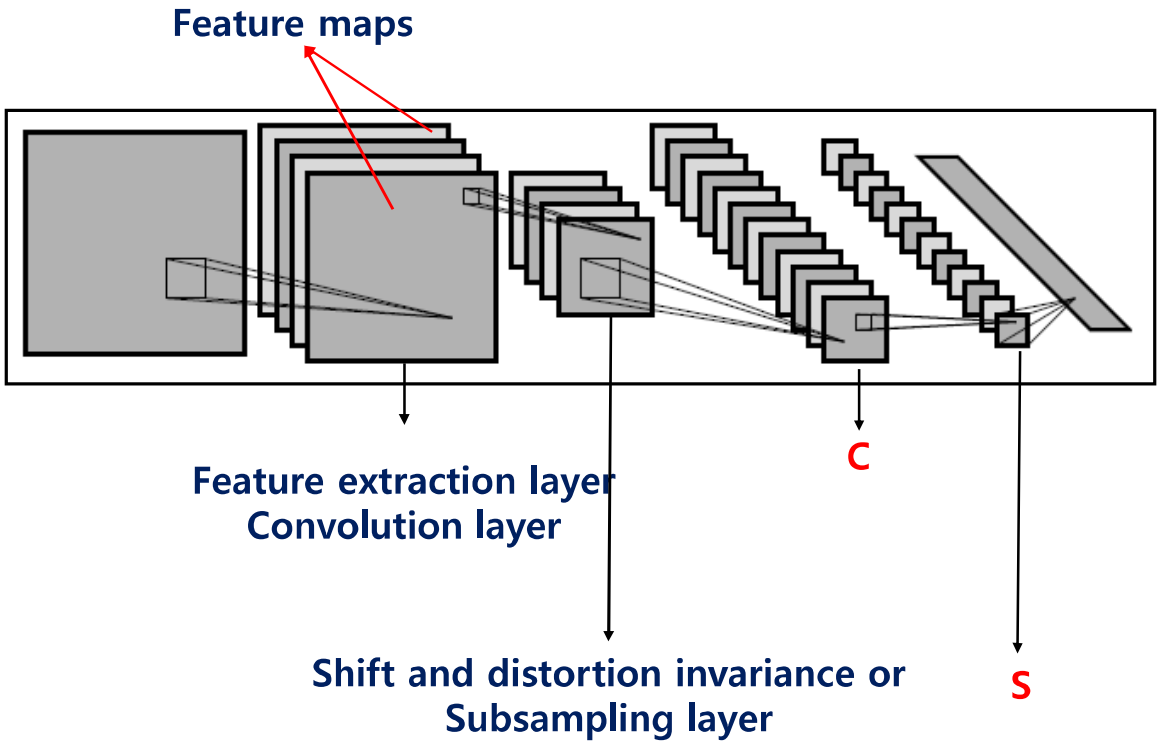
CLASSIFICATION

Classification

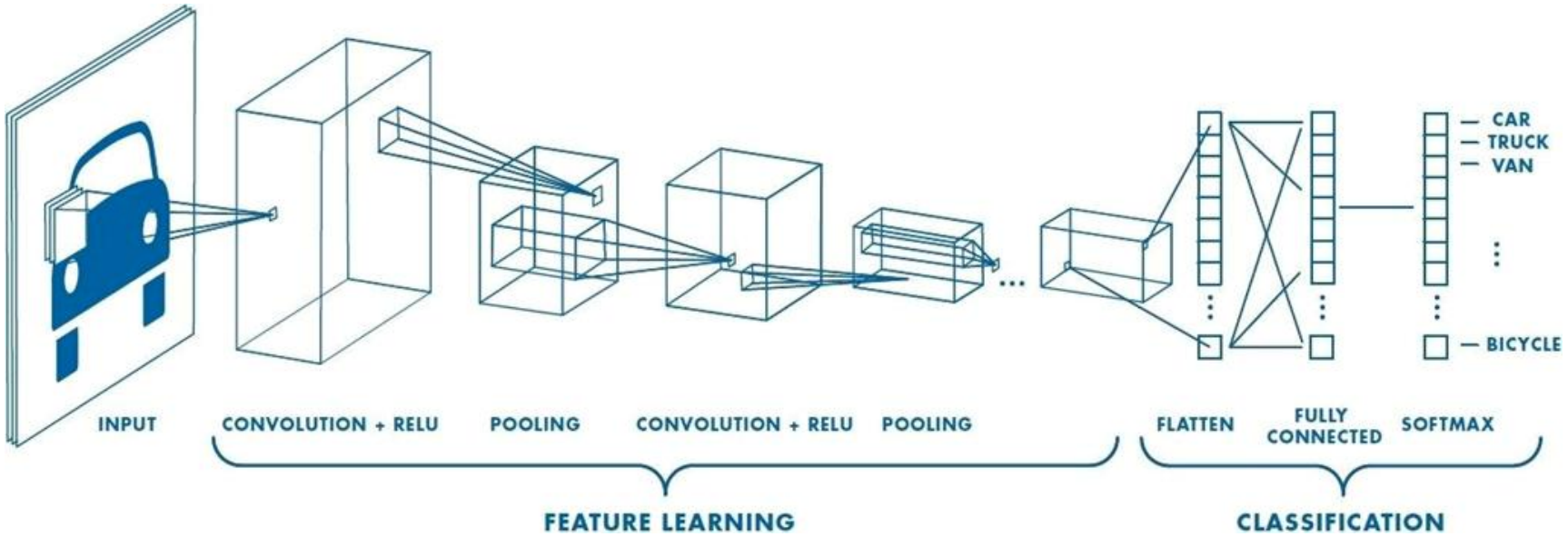


Convolutional neural network

CNN'S TOPOLOGY



CNN Architecture



LAYERS IN CONVOLUTION NEURAL NETWORK

CNN architecture consist of convolution layer, pooling layer and classification layer(Activation, Fully Connected and Loss Layer).

- Convolution layer: extract the unique features from the input image
- Activation Layer : To increase non-linearity of the network without affecting receptive fields of conv layers
- Pooling Layer : reduce the dimensionality
- Fully Connected(FC): adaptive to classification/encoding tasks
- Loss Layer: Minimize the Error
- Generally CNN is trained using back-propagation algorithm

CONVOLUTION LAYER

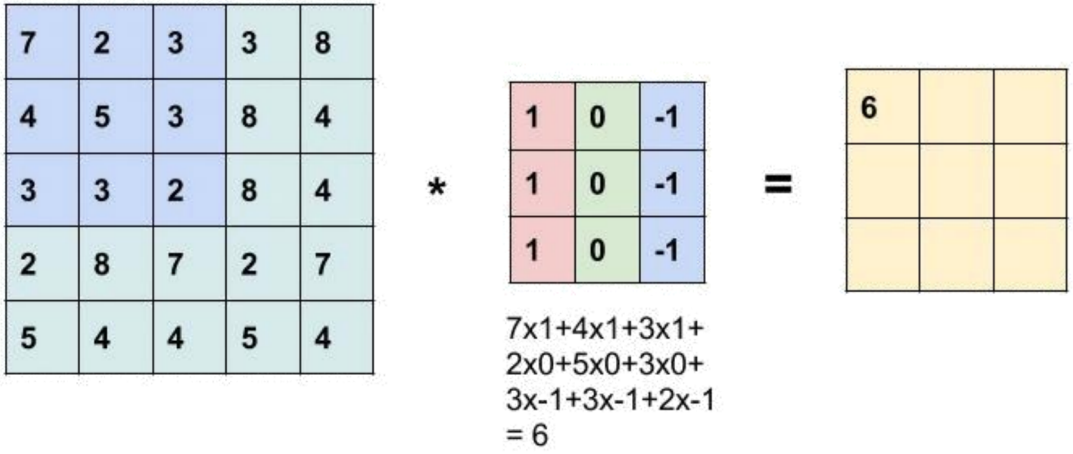
- Three Important process in Convolution
  - The input image,
  - The feature detector, and
  - The feature map.

- The input image is the image being detected. The feature detector is a matrix, usually 3x3 (it could also be 7x7).
- A feature detector is also referred to as a kernel or a filter.
- The matrix representation of the input image is multiplied element-wise with the feature detector to produce a feature map

CONVOLUTION LAYER

- The process is a 2D convolution on the inputs.
- The objective of a Conv layer is to extract features of the input volume
- The “dot products” between weights and inputs are “integrated” across “channels”.
- Filter weights are shared across receptive fields.
- The filter has same number of layers as input volume channels, and output volume has same “depth” as the number offilters.

Feature size = ((Image size − Kernel size) / Stride) + 1

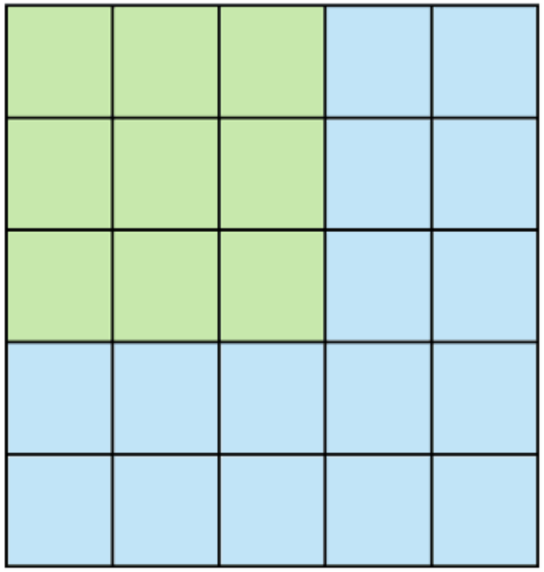


https://youtu.be/KiftWz544_8

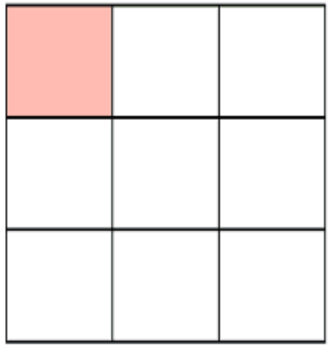
https://youtu.be/f0t-OCG79-U

STRIDE

• Stride denotes how many steps we are moving in each steps in convolution. By default it is one.



Stride 1

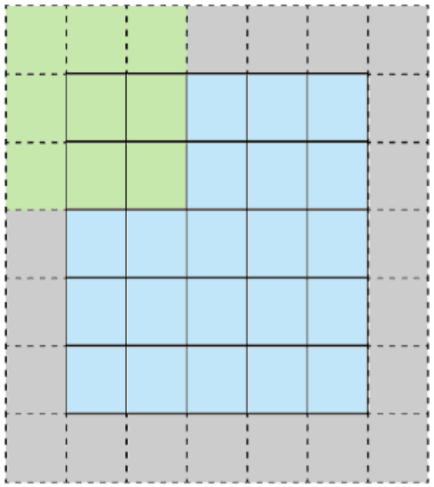


Feature Map

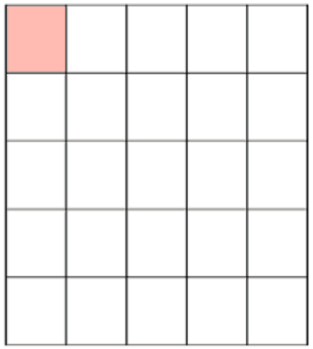
Convolution with Stride1

PADDING

- To maintain the dimension of output as in input
- Padding is a process of adding zeros to the input matrix symmetrically.
- It is used to make the dimension of output same as input.
- The extra grey blocks denote the padding



Stride 1 with Padding



Feature Map

Stride 1 with Padding1

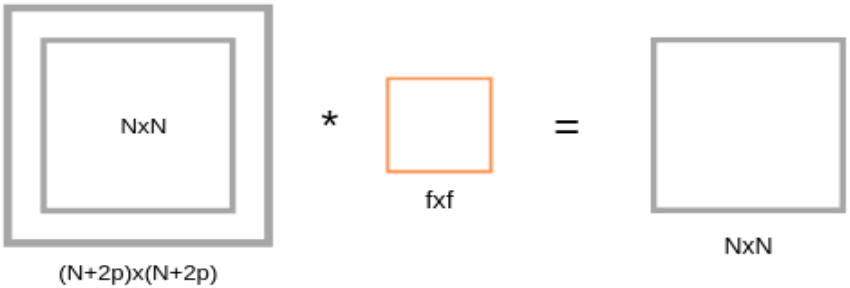
COMPUTING PADDING

Initially(without padding)

$$(N \times N) * (F \times F) = (N-F+1) \times (N-F+1)$$

If we apply filter F x F in (N+2p) x (N+2p) input matrix with padding, then we will get output matrix dimension (N+2p-F+1) x (N+2p-F+1)

Output after applying padding



$(N+2p-F+1) \times (N+2p-F+1)$ equivalent to NxN

$$N+2p-F+1 = N$$
$$p = (F-1)/2$$

Compute the output volume[W2xH2xD2]?

- $W2 = (W1 - F + 2P)/S + 1$
- $H2 = (H1 - F + 2P)/S + 1$
- $D2 = K$

where:

- [W1xH1xD1] : input volume size
- F: receptive field size
- S: stride
- P: amount of zero padding used on the border.
- K: depth

EXAMPLE

What is the output volume of the first Convolutional Layer
Input size: [227x227x3], W=227, F=11, S=4, P=0, and K=96.

$$W2 = (W1 - F + 2P)/S + 1$$

$$H2 = (H1 - F + 2P)/S + 1$$

$$D2 = K$$

Output:
W2= (227 - 11) / 4 + 1 = 55
H2=(227-11)/4+1=55
D2=K
The size of the Conv layer output volume is [55x55x96]

ACTIVATION LAYER

•In a neural network, numeric data points, called inputs, are fed into the neurons in the input layer.

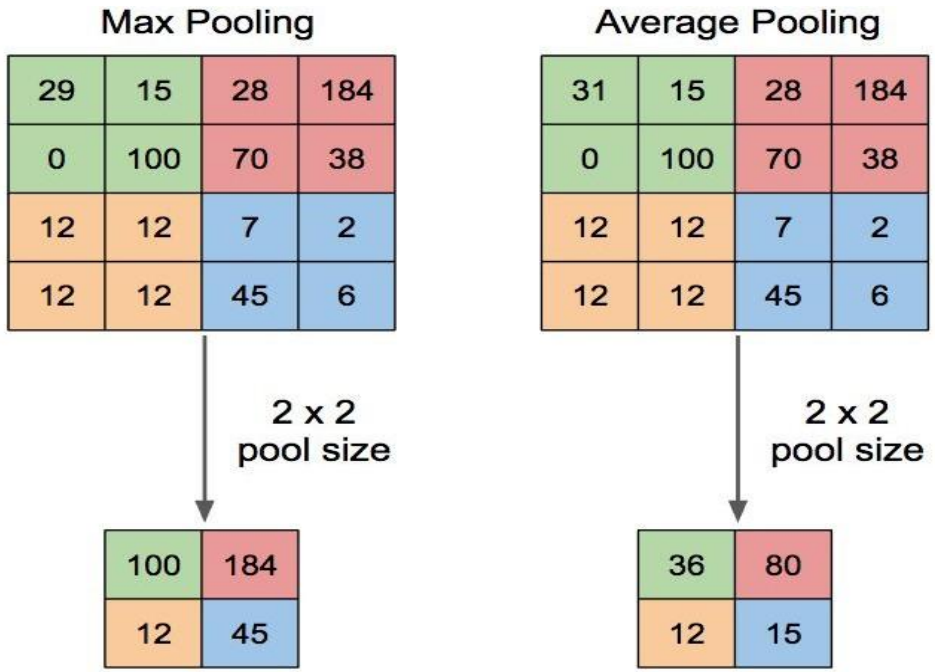
• Each neuron has a weight, and multiplying the input number with the weight gives the output of the neuron, which is transferred to the next layer.

•The activation function is a mathematical **“gate”** in between the input feeding the current neuron and its output going to the next layer.

• It can be as simple as a step function that turns the neuron output on and off, depending on a rule or threshold.

POOLING LAYER

- Convolutional layers provide activation maps.
- Pooling layer applies non-linear downsampling on activation maps.
- Pooling is aggressive (discard info); the trend is to use smaller filter size and abandon pooling
- a pooling layer is to reduce the dimensionality of feature maps. For this reason, it is also called downsampling.



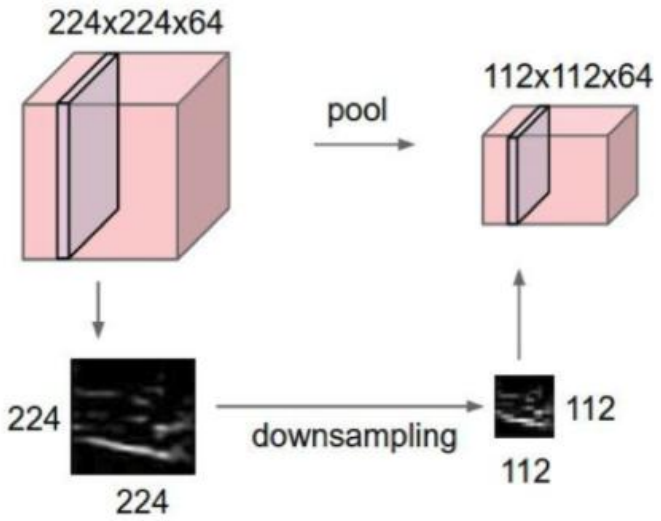
POOLING LAYER

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

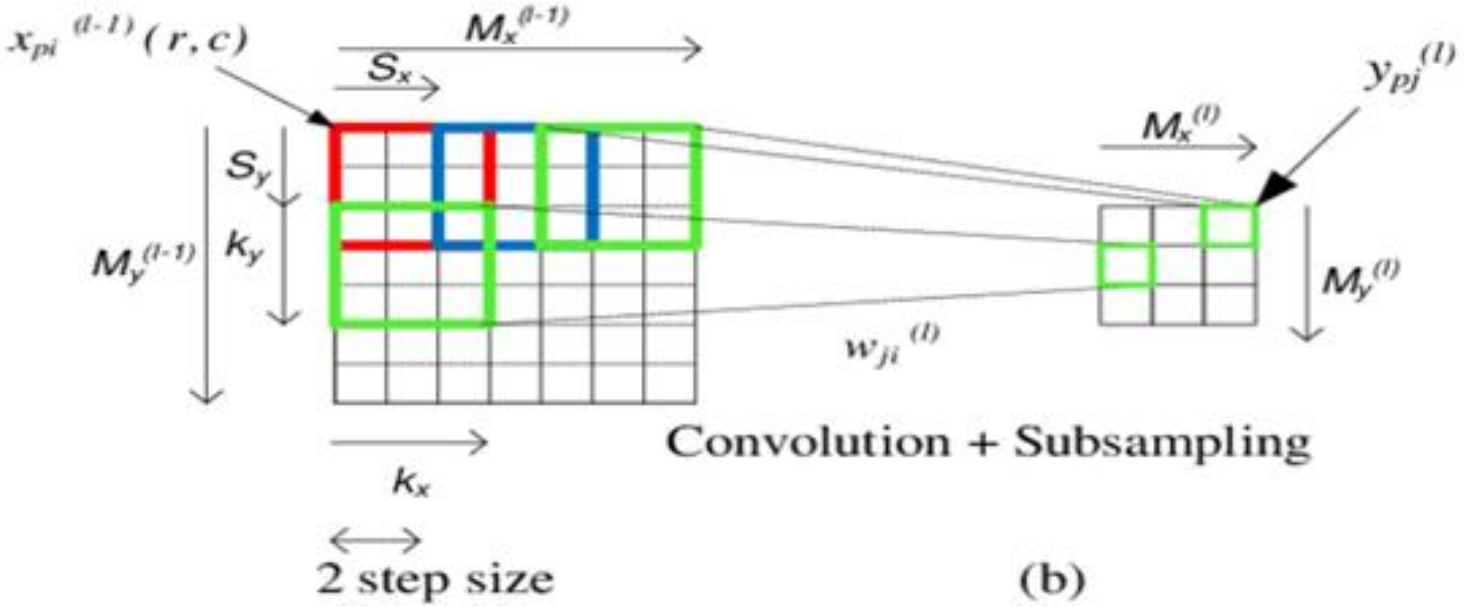
SUBSAMPLING: POOLING

- Makes representation smaller and more manageable
- Operates over each feature map independently.
- Has no activation function

- In standard CNNs, a convolution layer has trainable parameters which are tuned during the the training process, while the sub-sampling layer is a constant operation (usually performed by a max-pooling layer).



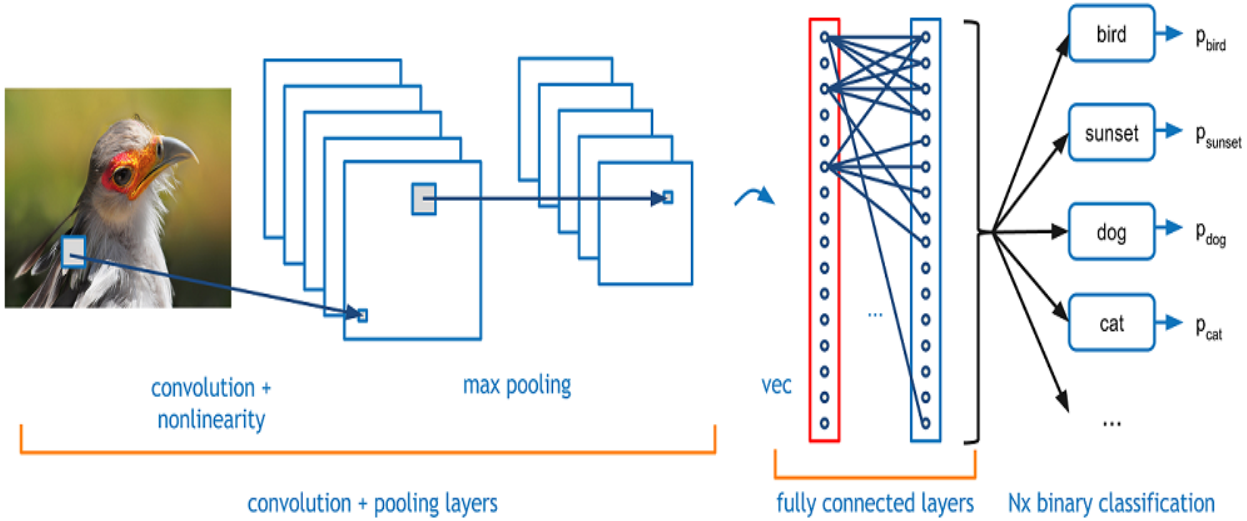
SUB SAMPLING



convolution/subsampling in CNN

FC LAYER

- Regular neural network
- Can view as the final learning phase, which maps extracted visual features to desired outputs
- Usually adaptive to classification/encoding tasks
- Common output is a vector, which is then passed through softmax to represent confidence of classification
- The outputs can also be used as “bottleneck”



LOSS LAYER

A loss function is a method of evaluating how well the model models the dataset. The loss function will output a higher number if the predictions are off the actual target values whereas otherwise it will output a lower number.

- Cross-Entropy loss (works well for classification, e.g., image classification)
- Hinge Loss
- Huber Loss, more resilient to outliers with smooth gradient
- Minimum Squared Error (works well for regression task, e.g., Behavioral Cloning)

$$H(p,q) = -\sum_{x} p(x)\ \log q(x)$$

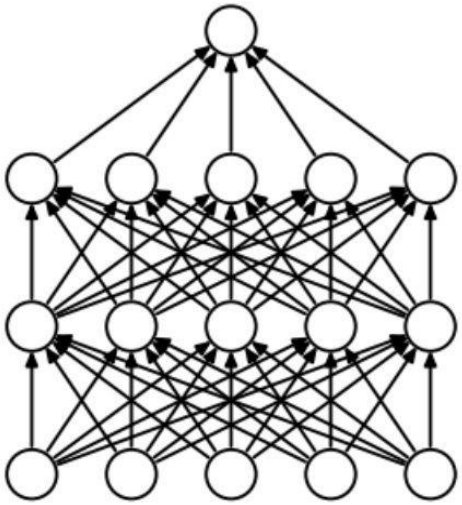
Binary case $-\,y \log \hat{y} - (1-y)\log(1-\hat{y})$

General case $-\sum_{i} p_i \log q_i$

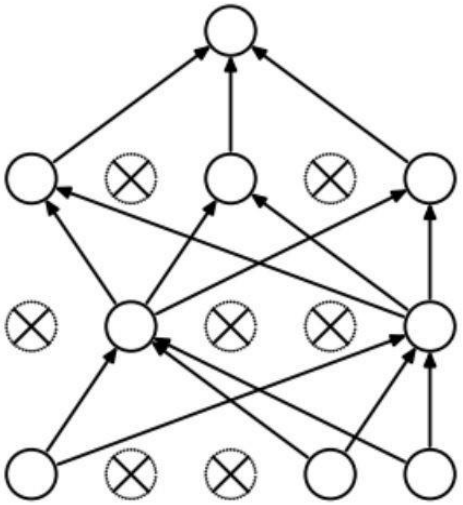
DROPOUT

Dropout is a regularization technique used to reduce over-fitting on neural networks.

- During training, randomly ignore activations by probability ***p***
- During testing, use all activations but scale them by ***p***
- Effectively prevent overfitting by reducing correlation between neurons



(a) Standard Neural Net

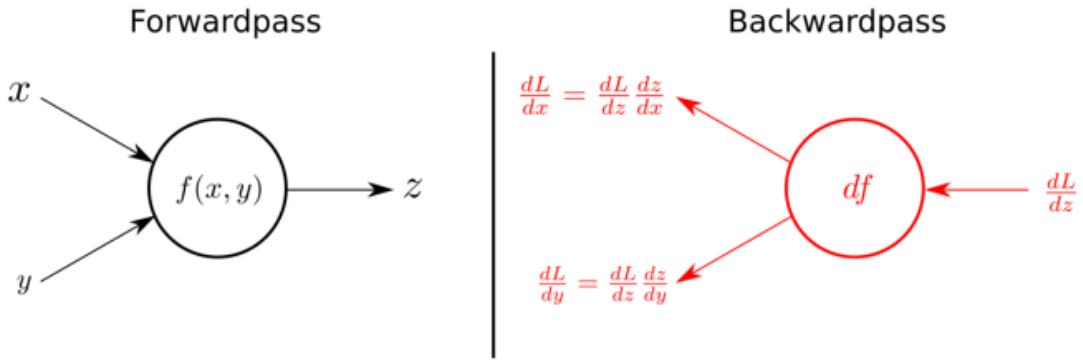


(b) After applying dropout.

BACK PROPAGATION

- “It is easy to fall into the trap of abstracting away the learning process —believing that you can simply stack arbitrary layers together and backprop will "magically make them work" on your data"

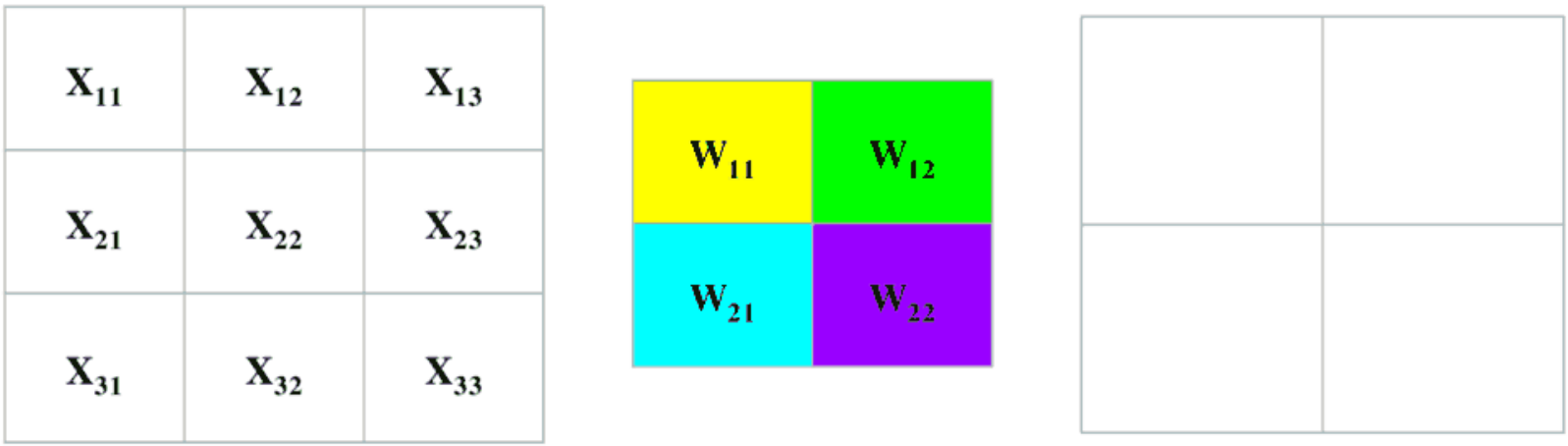
BACK PROPAGATION



The forward pass on the left calculates **z** as a function **f(x,y)** using the input variables **x** and **y.**

- The right side of the figures shows the backward pass.
- Receiving **dL/dz**, the gradient of the loss function with respect to **z** from above, the gradients of **x** and **y** on the loss function can be calculate by applying the chain rule

FORWARD PASS



Convolution Operation (Forward Pass)

Performing the convolution operation without flipping the filter. This is also referred to as the cross-correlation operation in literature.The above animation is provided just for the sake of clarity.

FORWARD PASS

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

W_{11}	W_{12}
W_{21}	W_{22}

h_{11}	h_{12}
h_{21}	h_{22}

Input Size : 3x3, **Filter Size** : 2x2, **Output Size** : 2x2

Output Equations

$$h_{11} = W_{11}X_{11} + W_{12}X_{12} + W_{21}X_{21} + W_{22}X_{22}$$

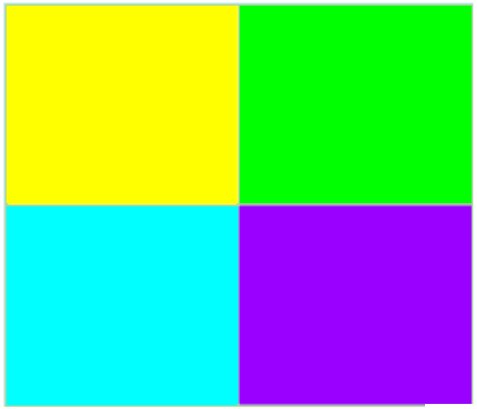
$$h_{12} = W_{11}X_{12} + W_{12}X_{13} + W_{21}X_{22} + W_{22}X_{23}$$

$$h_{21} = W_{11}X_{21} + W_{12}X_{22} + W_{21}X_{31} + W_{22}X_{32}$$

$$h_{22} = W_{11}X_{22} + W_{12}X_{23} + W_{21}X_{32} + W_{22}X_{33}$$

BACK PROPAGATION

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



∂h_{11}	∂h_{12}
∂h_{21}	∂h_{22}

Derivative Computation (Backward pass)

Derivatives after performing back propagation

$$\partial W_{11} = X_{11}\partial h_{11} + X_{12}\partial h_{12} + X_{21}\partial h_{21} + X_{22}\partial h_{22}$$

$$\partial W_{12} = X_{12}\partial h_{11} + X_{13}\partial h_{12} + X_{22}\partial h_{21} + X_{23}\partial h_{22}$$

$$\partial W_{21} = X_{21}\partial h_{11} + X_{22}\partial h_{12} + X_{31}\partial h_{21} + X_{32}\partial h_{22}$$

$$\partial W_{22} = X_{22}\partial h_{11} + X_{23}\partial h_{12} + X_{32}\partial h_{21} + X_{33}\partial h_{22}$$

CNN IMPLEMENTATION

- CNN is a supervised deep learning algorithm,
- it is used in various fields like image classification, speech recognition, image retrieval and face recognitio
- Demonstrates training a simple Convolutional Neural Network (CNN) to classify

CIFAR-10(collection of images) images, the data set which is available in tensorflow.

CNN can be implemented using following steps:

- import required Libraries
- Load Data
- Data Validation
- Create the Base Convolution
- Design the New Architecture(By Adding Dense Layer)
- Train the model
- Evaluate the model

REQUIRED LIBRARIES

- The following Libraries are required to classify the CIFAR-10 images

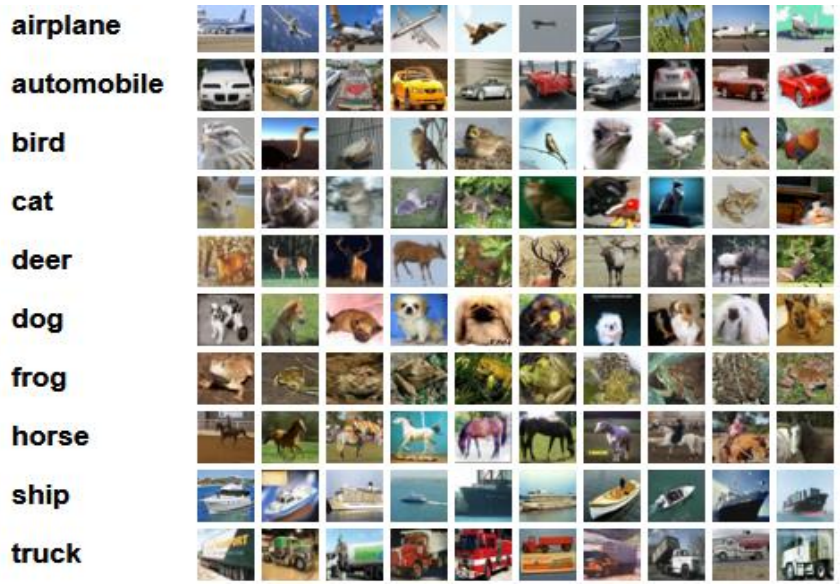
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

CIFAR10 DATA DESCRIPTION

- The CIFAR10 dataset contains 60,000 color images in 10 classes, with 6,000 images in each class.

The dataset is divided into 50,000 training images and 10,000 testing images.

- The classes in the dataset, as well as 10 random images from each:



LOAD THE DATA AND PREPARE THE DATA

- Load the Data

```
#load the data from tensorflow
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
#display first 2 rows
print(train_images[:2])
```

```
[[[177 144 116]
  [168 129  94]
  [179 142  87]
  ...
  [216 184 140]
  [151 118  84]
  [123  92  72]]

 [[154 177 187]
  [126 137 136]
  [105 104  95]
  ...
  [ 91  95  71]
  [ 87  90  71]
  [ 79  81  70]]

 [[140 160 169]
  [145 153 154]
  [125 125 118]
  ...
  [ 96  99  78]
  [ 77  80  62]
  [ 71  73  61]]

 [[140 155 164]
  [139 146 149]
  [115 115 112]
  ...
  [ 79  82  64]
  [ 68  70  55]
  [ 67  69  55]]

 ...

 [[175 167 166]
  [156 154 160]
  [154 160 170]
  ...
  [ 42  34  36]
  [ 61  53  57]
  [ 93  83  91]]

 [[165 154 128]
  [156 152 130]
  [159 161 142]
  ...
  [103  93  96]
  [123 114 120]
  [131 121 131]]

 [[163 148 120]
  [158 148 122]
  [163 156 133]
  ...
  [143 133 139]
  [143 134 142]
  [143 133 144]]]]
```

PREPARE THE DATA

- Normalize the data

```
# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
#Data after Normalization
print(train_images[0:2])
```

```
[[[[0.00090734 0.00095348 0.00096886]
   [0.00066128 0.00070742 0.00069204]
   [0.00076894 0.00073818 0.00066128]
   ...
   [0.00242983 0.00202999 0.0016609 ]
   [0.00233756 0.00192234 0.00156863]
   [0.00227605 0.00190696 0.00158401]]

  [[0.00024606 0.00030757 0.00030757]
   [0.         0.         0.        ]
   [0.00027682 0.00012303 0.        ]
   ...
   [0.00189158 0.00135333 0.00084583]
   [0.00183007 0.00127643 0.00076894]
   [0.0018762  0.00133795 0.00087659]]

  [[0.00038447 0.00036909 0.00032295]
   [0.00024606 0.00010765 0.        ]
   [0.00075356 0.00041522 0.00012303]
   ...
   [0.00181469 0.00129181 0.00076894]
   [0.00184544 0.00129181 0.00076894]
   [0.00167628 0.00112265 0.00064591]]
```

VALIDATE THE DATA

- To verify that the dataset looks correct ,
- plot the first 10 images from the training set and
- display the class name below each image.

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
plt.figure(figsize=(10,10))
for i in range(10):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

CREATE THE BASE CONVOLUTION

- CNN takes tensors of shape (image_height, image_width, color_channels)
- CNN process inputs of shape (32, 32, 3), which is the format of CIFAR images.
- passes the argument input_shape to our first layer.
- will get out_shape and no of parameters in that layer

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
```

Output of the first Layer:

```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 30, 30, 32)        896
=================================================================
Total params: 896
Trainable params: 896
Non-trainable params: 0
_________________________________________________________________
```

CREATE THE CONVOLUTION

Out shape will be calculated using following formula

formula $[(W-K+2P)/S]+1$.

where
W is the input volume size,
F is the receptive field size,
S is the stride, and
P is the amount of zero padding used on the border

```
In our case
w1=32,h1=32,kernel=3,p=0,s=1,filter=32
Output shape:
w2=((32-3+0)/1)+1=29+1
h2=((32-3+0)/1)+1=29+1
D2=filter
So, o/p:(None,30,30,32)
```

Note: None is batch size

CREATE THE CONVOLUTION

No.of paramters of CONV2D will be calculated as follows:

Parameters in the first CONV2(filter shape =3*3, stride=1) layer is:
((shape of width of filter * shape of height filter * number of filters in the previous layer+1)
* number of filters) = (((3*3*3)+1)*32) = 896.

CREATE THE CONVOLUTION

The convolutional base using a common pattern:
a stack of Conv2D and MaxPooling2D layers.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

- The output of every Conv2D and MaxPooling2D layer is a 3D tensor of shape (height, width, channels).
- The width and height dimensions tend to shrink as you go deeper in the network.

```
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 4, 4, 64)          36928
=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
```

ADDING DENSE LAYERS

- Feed the last output tensor from the convolutional base (of shape (4, 4, 64)) into one or more Dense layers to perform classification.
- Dense layers take vectors as input (which are 1D), while the current output is a 3D tensor
- First, flatten (or unroll) the 3D output to 1D, then add one or more Dense layers on top.
- CIFAR has 10 output classes, so we can use a final Dense layer with 10 outputs and a softmax activation.

ADDING DENSE LAYERS

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

COMPLETE ARCHITECTURE OF CNN

- As our (4, 4, 64) outputs were flattened into vectors of shape (1024) before going through two Dense layers.

```
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 4, 4, 64)          36928
_________________________________________________________________
flatten_2 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                65600
_________________________________________________________________
dense_5 (Dense)              (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
```

COMPILE AND TRAIN THE MODEL

• **Metric**: How to measure the performance of our model using a ***metric***. We used *accuracy* as the metric in our experiments.

• **Loss function**: A function that is used to calculate a loss value that the training process then attempts to minimize by tuning the network weights. For classification problems, cross-entropy loss works well.

• **Optimizer**: A function that decides how the network weights will be updated based on the output of the loss function. We used the popular Adam optimizer in our experiments.

COMPILE AND TRAIN THE MODEL

- Train the model with 10 epochs

Learning parameter	Value
Metric	accuracy
Loss function - binary classification	binary_crossentropy
Loss function - multi class classification	sparse_categorical_crossentropy
Optimizer	adam

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
```

COMPILE AND TRAIN THE MODEL

Output:

```
Epoch 1/10
1563/1563 [==============================] - 65s 42ms/step - loss: 1.4969 - accuracy: 0.4508 - val_loss: 1.2199 - val_accuracy: 0.5652
Epoch 2/10
1563/1563 [==============================] - 65s 41ms/step - loss: 1.1216 - accuracy: 0.6040 - val_loss: 1.0938 - val_accuracy: 0.6113
Epoch 3/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.9659 - accuracy: 0.6619 - val_loss: 0.9353 - val_accuracy: 0.6733
Epoch 4/10
1563/1563 [==============================] - 65s 42ms/step - loss: 0.8770 - accuracy: 0.6916 - val_loss: 0.9198 - val_accuracy: 0.6776
Epoch 5/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.8059 - accuracy: 0.7182 - val_loss: 0.8872 - val_accuracy: 0.6927
Epoch 6/10
1563/1563 [==============================] - 67s 43ms/step - loss: 0.7495 - accuracy: 0.7387 - val_loss: 0.9116 - val_accuracy: 0.6876
Epoch 7/10
1563/1563 [==============================] - 65s 41ms/step - loss: 0.6991 - accuracy: 0.7569 - val_loss: 0.8549 - val_accuracy: 0.7130
Epoch 8/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.6551 - accuracy: 0.7700 - val_loss: 0.8902 - val_accuracy: 0.7022
Epoch 9/10
1563/1563 [==============================] - 65s 41ms/step - loss: 0.6176 - accuracy: 0.7828 - val_loss: 0.9544 - val_accuracy: 0.6906
Epoch 10/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.5819 - accuracy: 0.7956 - val_loss: 0.9039 - val_accuracy: 0.7063
```

LOSS FUNCTIONS

- loss functions are helpful to train a neural network.
- Error is calculated as the difference between the actual output and the predicted output.
- The output variable in classification problem is usually a probability value f(x), called the score for the input x

Classification algorithms are:

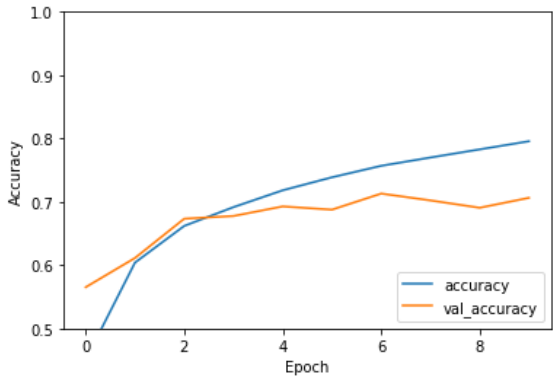
- Binary Cross Entropy /SparseCrossEntropy
- Negative Log Likelihood
- Margin Classifier
- Soft Margin Classifier

EVALUATE THE MODEL

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 4s - loss: 0.9039 - accuracy: 0.7063
```

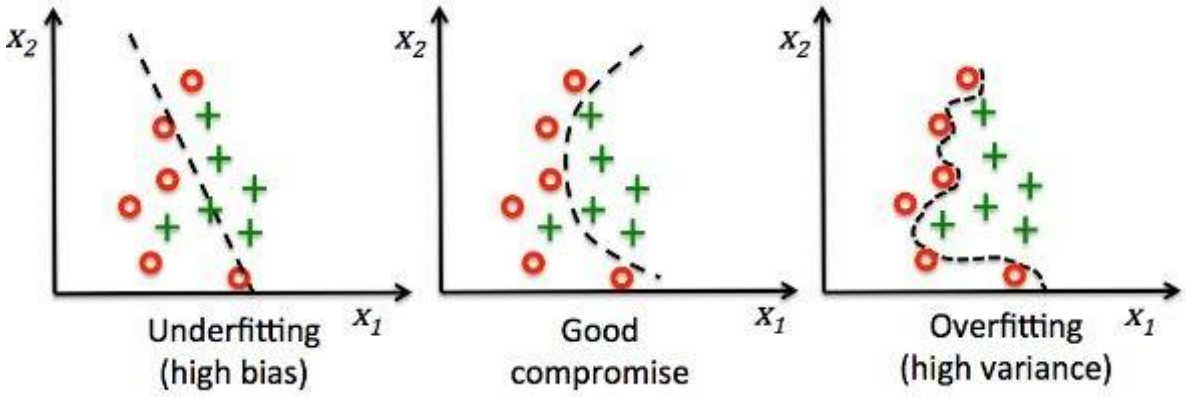


Testing accuracy and validation accuracy of our model,

REGULARIZATION

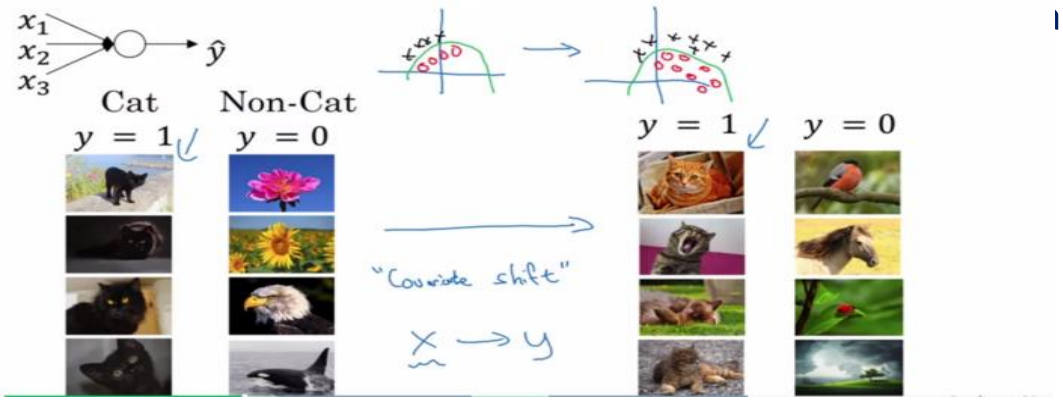
- L1 / L2
- Dropout
- Batch norm
- Gradient clipping
- Max norm constraint

To prevent overfitting with huge amount of trainingdata



BATCH NORMALIZATION

- Batch normalization is a layer that allows every layer of the network to do learning more independently.
- Normalize the input layer by adjusting and scaling the activations
- It is used to normalize the output of the previous layers.
- The activations scale the input layer in normalization.
- Using batch normalization learning becomes efficient also it can be used as regularization to avoid overfitting of the model.
- The layer is added to the sequential model to standardize the input or the outputs.
- Batch normali ıift around (covariance shift).



BATCH NORMALIZATION

- Makes networks robust to bad initialization of weights
- Usually inserted right before activation layers
- Reduce covariance shift by normalizing and scaling inputs
- The scale and shift parameters are trainable to avoid losing stability of the network

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

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.