Data Mining, Machine Learning and Deep Learning
Lecture-13

By: Somnath Mazumdar
Assistant Professor
sma.digi@cbs.dk

Spring 2023

CDSCO1004U/7.5 ECTS

Overview

- BatchNorm
- Convolutional Neural Network (CNN)
- Adversarial Examples
    - Generative Adversarial Network (GAN)

BatchNorm

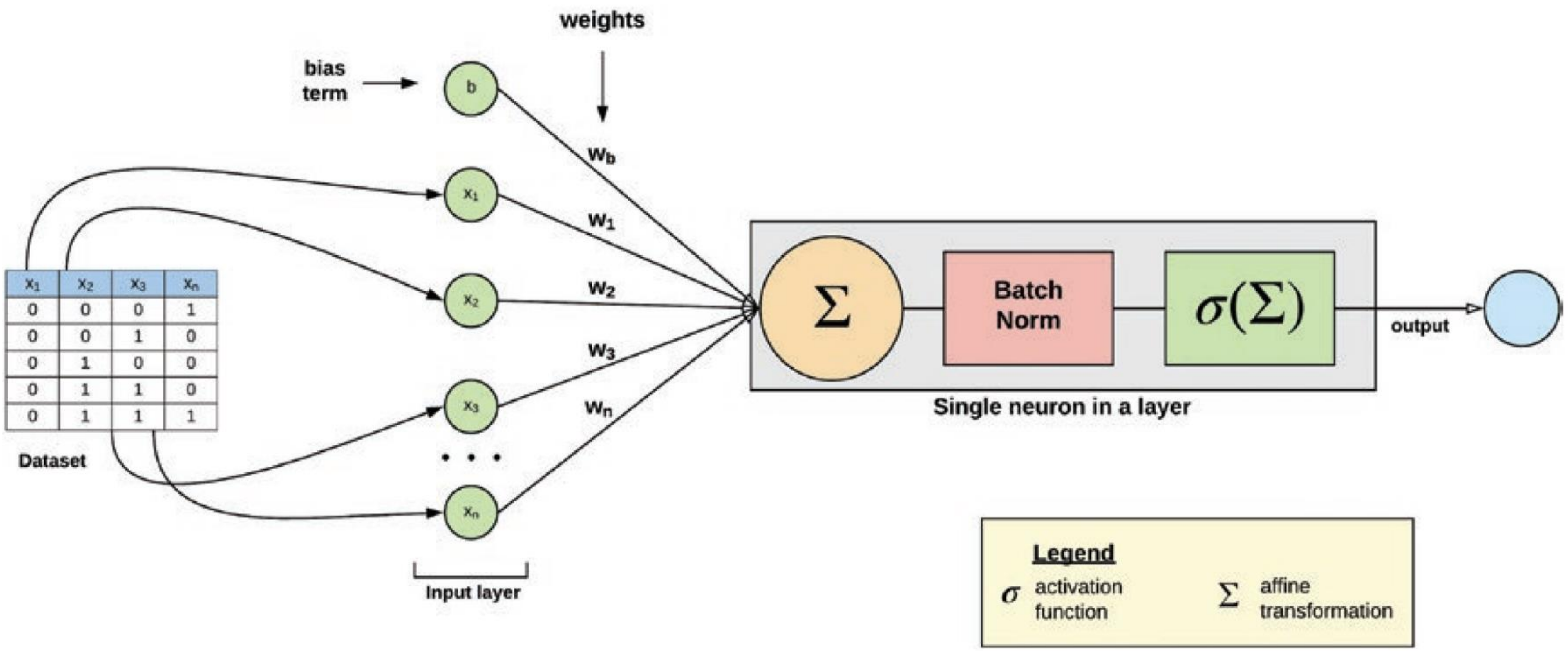
- Vanishing gradients problem can be alleviated with better weight initialization, better optimizers or Batch Normalization[1].
- BatchNorm most successful architectural innovations in deep learning[2].
- BatchNorm aims to stabilize distribution (over a minibatch) of inputs to a given network layer during training.

- BatchNorm Working: Operation lets model learn optimal scale and mean of each of layer's inputs.

1. First add an operation in model just before or after activation function of each hidden layer. This operation simply zero-center and normalizes each input.
2. Next, scales and shifts result using scaling and shifting vectors per layer.

1. Sergey Ioffe and Christian Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Proceedings of the 32nd International Conference on Machine Learning (2015): 448–456.
2. Santurkar, Shibani, et al. "How does batch normalization help optimization?." *Advances in Neural Information Processing Systems*. 2018.

BatchNorm



- Note: If you add a BN layer as the very first layer of your NN, you do not need to standardize your training set; the BN layer will do it for you.
- Vanishing gradients problem can be reduced to a point that activation functions can be used for further solution.
- BN can improve many deep neural networks.

Bisong E. (2019) More on Optimization Techniques. In: Building Machine Learning and Deep Learning Models on Google Cloud Platform. Apress, Berkeley, CA

BatchNorm

- Batch Normalization is tricky to use in RNNs but sufficient for other nets.
  - Gradient Clipping* is often used RNNs to mitigate exploding gradients problem.
  - Gradient clipping does not help with vanishing gradients.
  - Gradient Clipping clips the gradients during backpropagation so that they never exceed some threshold.
- BN acts like a regularizer reducing need for other regularization techniques (such as dropout).
- BN adds runtime penalty to neural network.
  - Training is rather slow because each epoch takes much more time when BN in use.

*Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." *International conference on machine learning*. 2013.

Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNN)

- CNNs emerged from the study of the brain's visual cortex.
- Popular use image recognition (since 1980s).
- Deep neural network does NOT work well for large mage recognition..?
  - For example, a 100×100 pixel image has 10,000 pixels. If the first layer has just 1,000 neurons means a total of 10M connections for just the first layer.
  - CNNs solve it using partially connected layers and weight sharing.
- CNN each layer is represented in 2D which makes it easier to match neurons with their corresponding inputs.

https://poloclub.github.io/cnn-explainer/

Convolutional Neural Networks (CNN)

- Convolution is a mathematical operation that slides one function over another and measures the integral of their pointwise multiplication.
- It has deep connections with Fourier transform + Laplace transform.
- Convolutional layers use cross-correlations (similar to convolutions).

- CNN has three fundamental layers: Convolutional layer, Pooling layer, Fully connected layer.
- Convolutional layer is the most important building block of a CNN.
- During training convolutional layers require a huge amount of RAM.

Convolutional Neural Networks (CNN)

- Colored image consists red, green, and blue with pixel intensity values from 0 to 255.
- A colored image has a matrix shape of [height x width x channel].
- Side image of shape [10 x 10 x 3] indicating a 10 x 10 matrix with three channels.

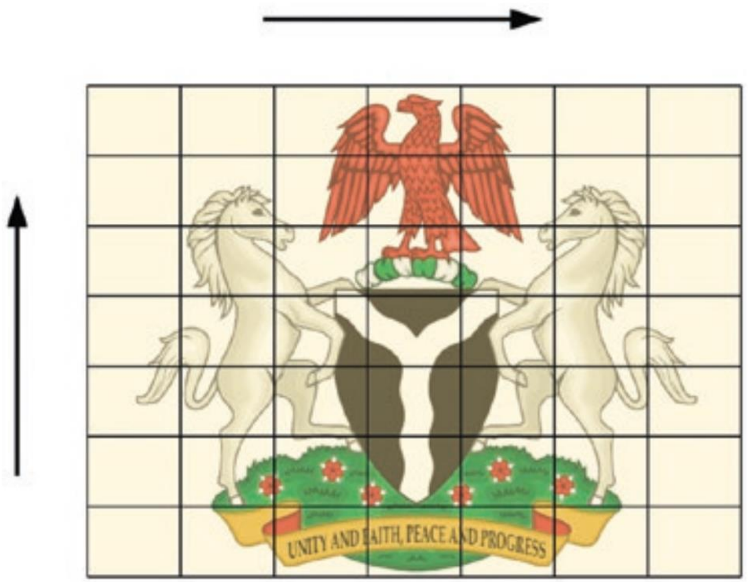
250	250	255	246	249	251	245	251	250	250
249	255	246	206	118	97	183	241	255	250
253	253	218	60	8	6	28	203	254	254
255	242	226	89	37	45	89	214	230	253
254	231	208	235	122	112	235	213	217	255
254	238	203	253	139	111	254	204	228	251
255	234	229	196	114	101	155	230	233	255
253	247	254	55	93	132	0	215	252	253
251	253	236	144	74	74	121	221	255	252
250	255	249	242	218	209	239	246	253	249

250	250	255	236	216	209	231	255	252	250
251	254	234	161	78	52	120	223	255	250
253	255	173	43	5	8	4	148	255	255
255	249	123	0	50	60	2	101	217	255
255	230	94	54	53	25	119	113	179	255
255	226	130	214	49	2	150	136	208	255
255	216	218	205	109	94	147	216	210	255
254	244	237	47	87	122	0	178	243	255
251	254	197	69	61	60	51	165	255	252
248	255	250	203	156	137	188	249	253	249

250	250	253	224	103	97	202	252	251	252
248	255	212	6	8	21	4	183	253	248
253	254	54	35	118	119	64	31	239	255
253	205	2	73	103	103	83	0	175	252
255	165	0	0	58	67	0	0	132	253
253	150	2	23	74	83	65	27	119	255
253	150	50	236	77	42	255	109	106	255
255	229	33	120	53	37	127	35	202	254
251	255	138	2	68	83	0	106	255	252
249	253	252	148	33	29	122	254	253	249

Convolutional Neural Networks (CNN)

- Image is depicted as a matrix of pixel intensity values ranging from 0 to 255.
- Grayscale consists of a single channel with 0 representing the black areas and 255 the white regions with the values in between for various shades of Gray.



7 x 7 pixels

2-D representation of an image

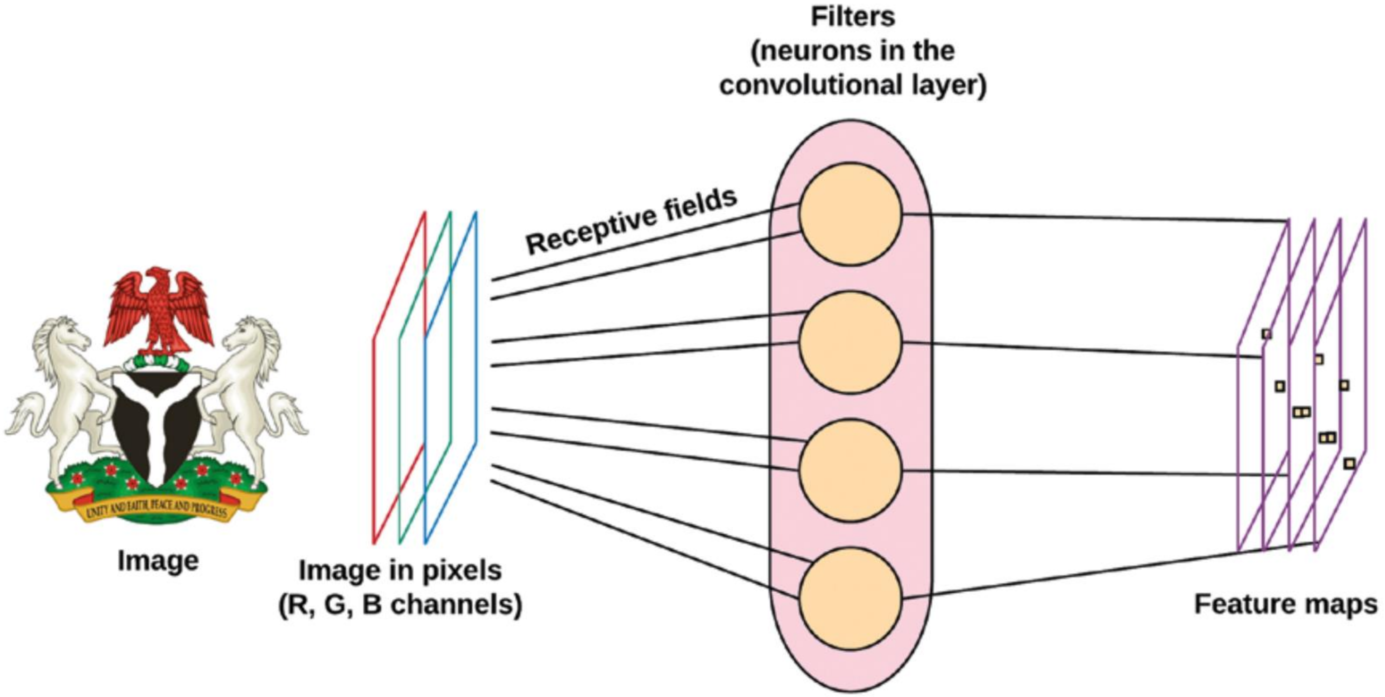
251	251	255	233	182	179	224	254	251	250
250	255	229	120	66	56	96	215	255	249
253	254	144	47	29	31	32	122	248	255
255	229	113	65	56	62	68	106	204	255
255	203	102	106	82	78	118	108	178	255
254	199	109	154	95	78	158	120	179	255
255	196	156	207	98	77	173	181	179	255
254	241	163	67	76	90	25	135	230	255
251	254	190	72	72	72	59	164	255	252
249	253	251	193	127	115	179	250	254	249

10 x 10 grayscale image with its matrix representation.

E. Bisong, Building Machine Learning and Deep Learning Models on Google Cloud Platform,

CNN Components

- Convolution layer is made up of filters and feature maps.
  - Filter is passed over input image pixels to capture a specific set of features in a process called convolution.
    - Convolution is the process by which a function is applied to a matrix to extract specific information from the matrix.
  - Feature maps are outputs of a filter in a convolutional layer.



Design Convolutional layer

- Considerations to design convolutional layer:
    - Filter size: Neuron’s weights can be represented as a small image size of receptive field.
        - Filters are also known as convolution kernels.
    - Stride of filter: determines how many pixel steps filter makes when moving from one image activation to another (typical to use a stride of 1).
    - Padding for input layer: Zero padding is used to pad borders of image pixels with a defined layer of zeros.

without zero padding

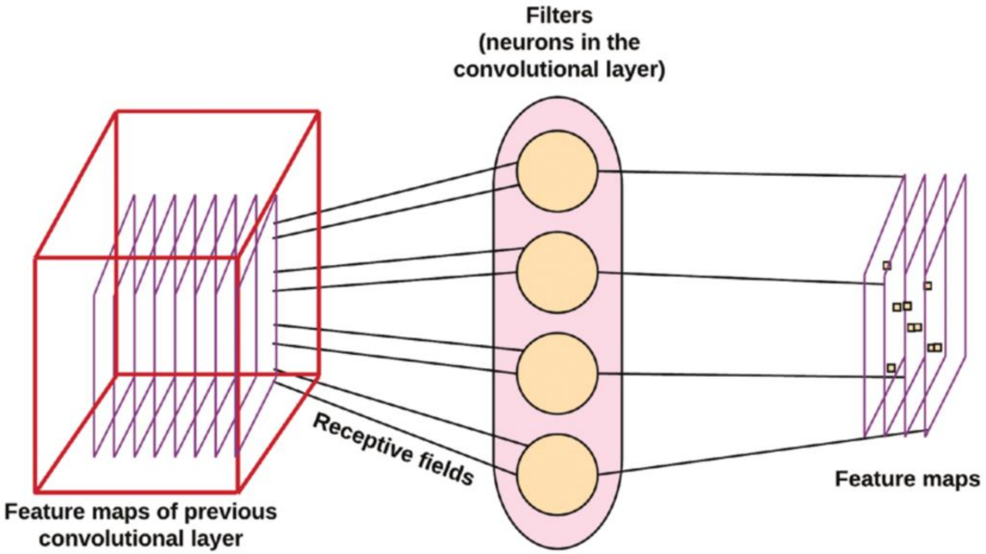
251	251	255	233	182	179
250	255	229	120	66	56
253	254	144	47	29	31
255	229	113	65	56	62
255	203	102	106	82	78
254	199	109	154	95	78

with zero padding

0	0	0	0	0	0	0	0
0	251	251	255	233	182	179	0
0	250	255	229	120	66	56	0
0	253	254	144	47	29	31	0
0	255	229	113	65	56	62	0
0	255	203	102	106	82	78	0
0	254	199	109	154	95	78	0
0	0	0	0	0	0	0	0

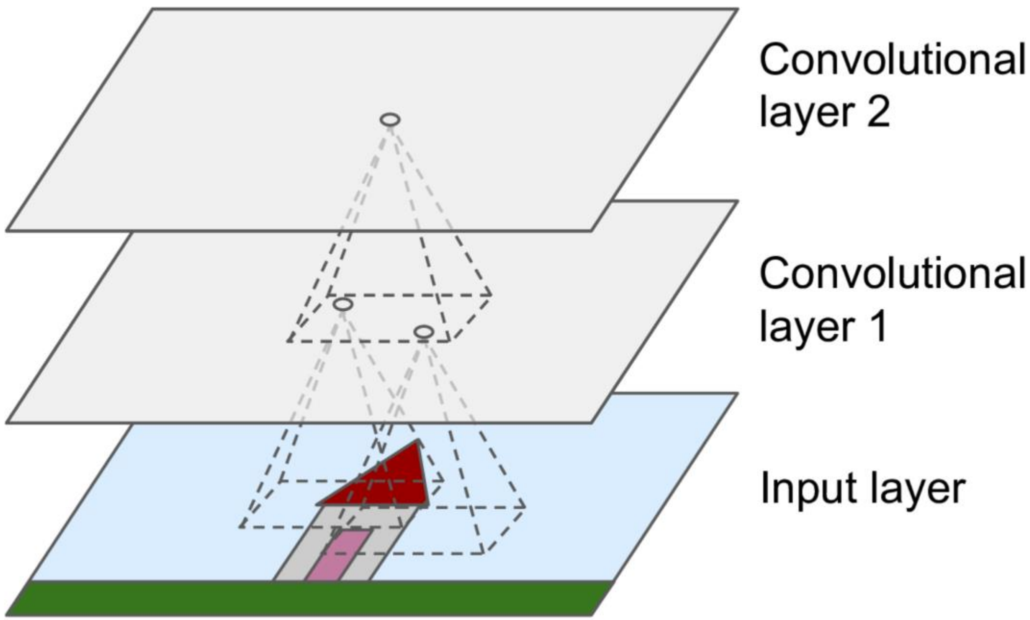
CNN Components

- Feature Maps are outputs of a filter in a convolutional layer.
    - Expose certain patterns of input image (such as horizontal lines, vertical lines).
    - Deeper CNN: Inputs to a deeper convolutional layer are feature maps of previous layer.
- Pooling layer summarizes image features learned in previous network layers.
    - Pooling Layer: follow one or more convolutional layers.
    - Common type of pooling layer is Max pooling layer.
    - Goal: to reduce feature map of convolutional layer.



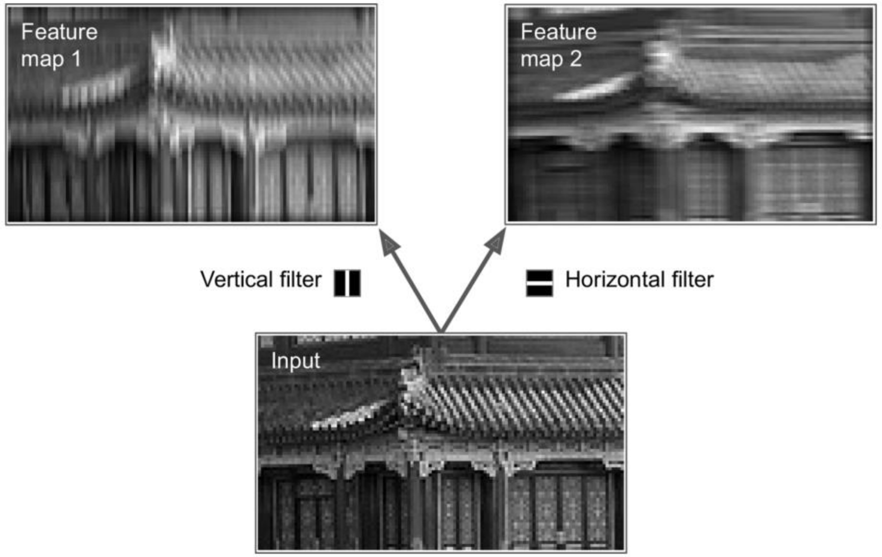
Convolutional Layer

- Neurons in first convolutional layer are connected only to pixels in their receptive fields (BUT not to all pixels).
- Each neuron in second convolutional layer is connected only to neurons located within a small rectangle in first layer.
- It allows network to concentrate on small low-level features in first hidden layer, then assemble them into larger higher-level features in next hidden layer, and so on.



Convolutional Layer

- Neuron’s weights can be represented as a small image size of receptive field.
- Feature map highlights areas in an image that activate filter (or convolution kernels) most.
- All neurons in a feature map share same parameters which dramatically reduces number of parameters in model.
- Once CNN has learned to recognize a pattern in one location, it can recognize it in any other location.
    - DNN can recognize only at particular location.



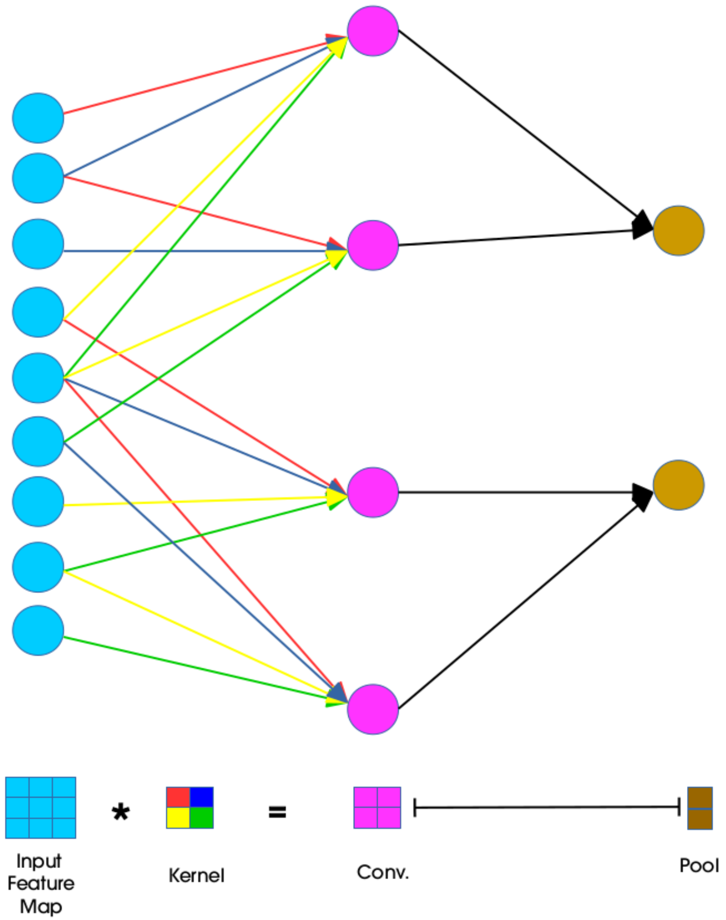
CNN Components

- Fully Connected Network (FCN) layer is feedforward neural network or multilayer perceptron (MLP).
  - These layers typically have a non-linear activation function (softmax activation).
  - FCN is the final layer of CNN.

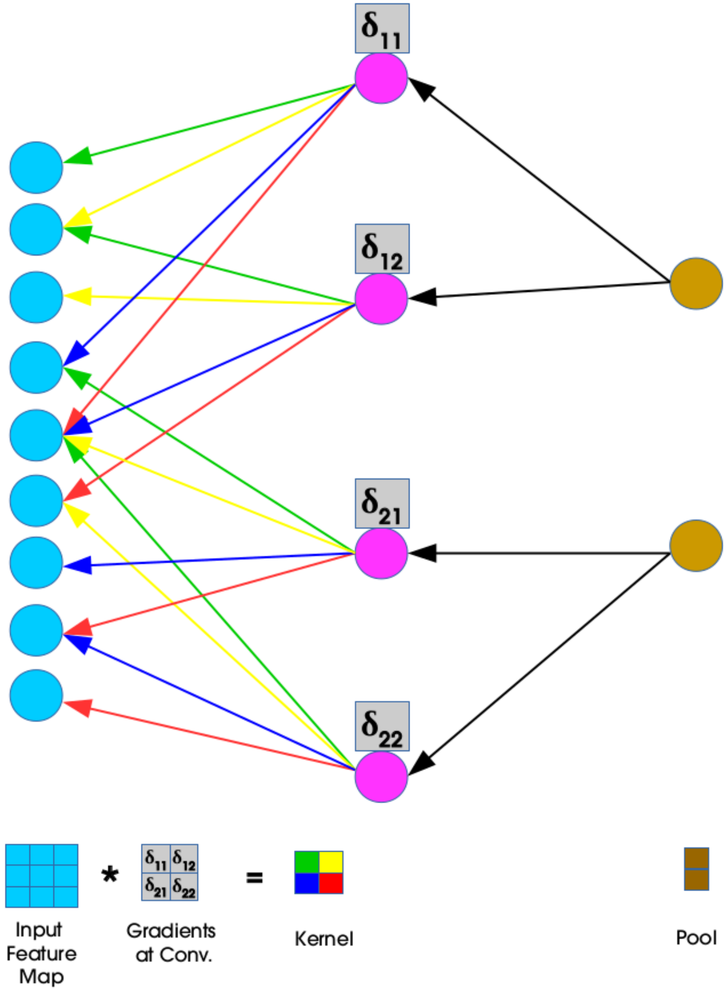
- CNN Modeling:
  - First layer following input layer of images must be a convolutional layer for extracting image features.
  - Pooling layers typical follow a set of one or more convolutional layers.

Convolution Layer Training Process

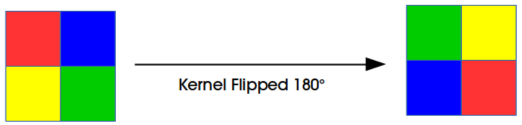
Forward Propagation



Backpropagation



No learning takes place on the pooling layers!!

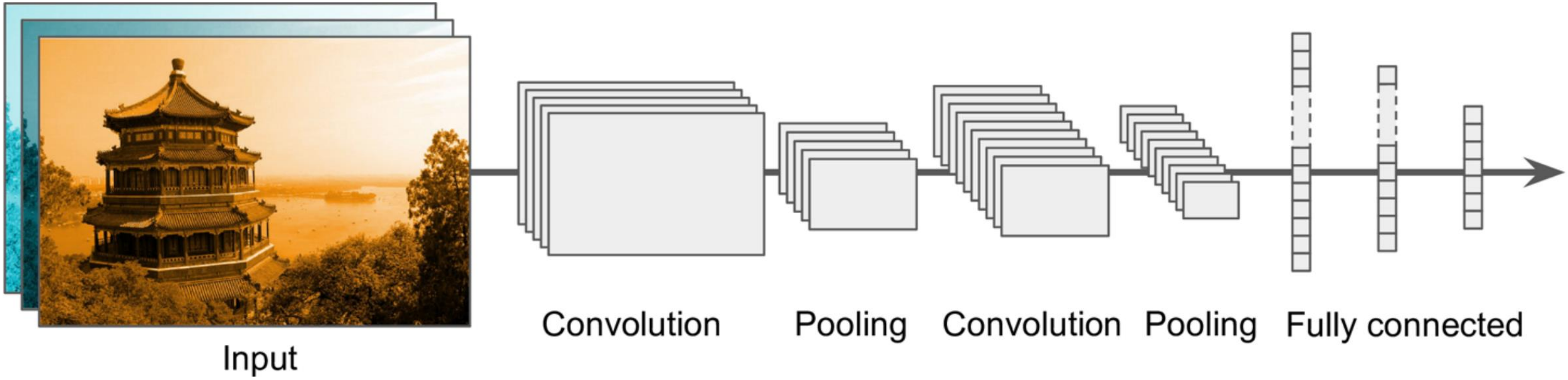


Gradients ==> $\delta_{11}, \delta_{12}, \delta_{21}, \delta_{22}$

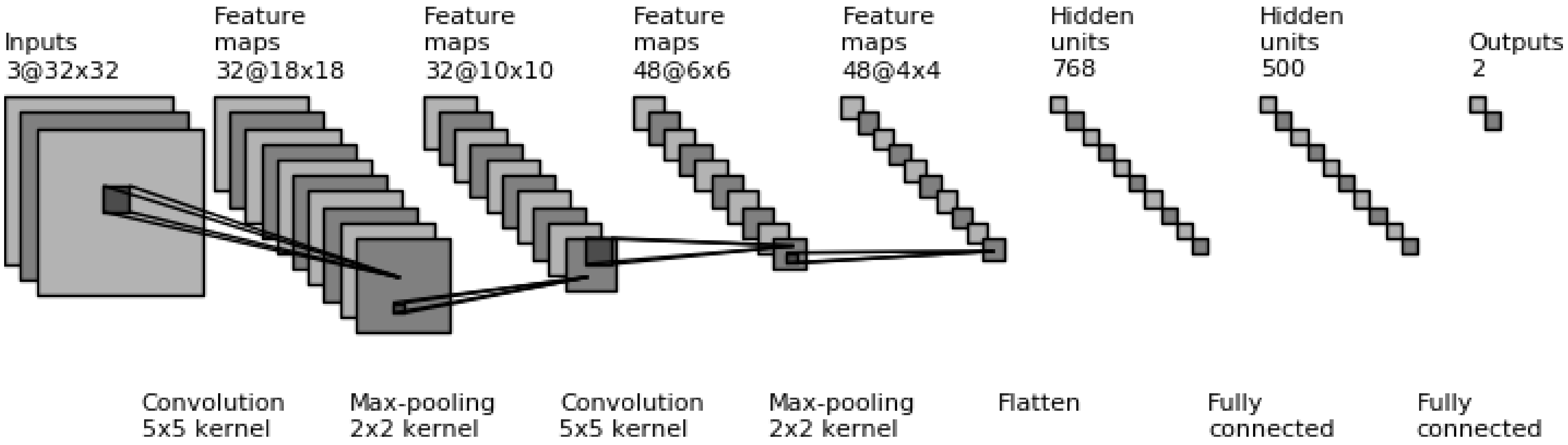
https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

CNN Components

- CNN Modeling:
  - Fully connected layer must be final layer of CNN (called dense layer).
    - Contains ReLU, softmax activation function to give probabilities of class membership.
  - CNN may include one or more Dropout layers to prevent network overfitting.



CNN Example


Inputs
3@32x32
Feature
maps
32@18x18
Feature
maps
32@10x10
Feature
maps
48@6x6
Feature
maps
48@4x4
Hidden
units
768
Hidden
units
500
Outputs
2
Convolution
5x5 kernel
Max-pooling
2x2 kernel
Convolution
5x5 kernel
Max-pooling
2x2 kernel
Flatten
Fully
connected
Fully
connected

Code Snippet

Convolutional base using a common pattern: a stack of Conv2D and MaxPooling2D layers.
CNN takes tensors of shape (image_height, image_width, color_channels),
color_channels refers to (R,G,B) to support the format of CIFAR images.

```
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'))
```

Adversarial Examples

Putting into the context

Panda

Gibbon

Misclassification

$+ .007 \times$

$=$

$\boldsymbol{x}$

$y =$"panda"
w/ 57.7%
confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
w/ 8.2%
confidence

$\boldsymbol{x} + \epsilon\, \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
w/ 99.3%
confidence

Adversarial Examples

- ML models consistently misclassify adversarial examples from dataset
  - such that perturbed input results in model outputting an incorrect answer with high confidence.
  - ML models misclassify examples that are only slightly different from correctly classified examples drawn from data distribution.
  - Adversarial examples are inputs formed by applying small but intentionally worst-case perturbations.

Goodfellow et al. Explaining and Harnessing Adversarial Examples

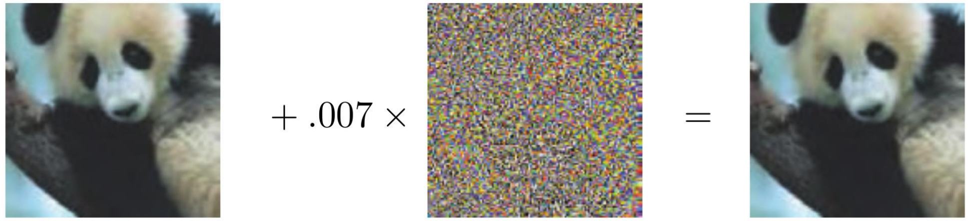
Adversarial Examples

- ML models such as DNN, clustering, Naive Bayes, Decision tree, Multilayer Perceptron, SVM are not attacked resilient.
- Injection of adversarial/malicious data into training datasets that can caused decreased performance/ model failure is known as poisoning.
  - Malicious users add malicious data with similar features of original data and wrong labels.
  - Malicious users might know the training data distribution; also learning algorithm.

Goodfellow et al. Explaining and Harnessing Adversarial Examples

Adversarial Examples

- A model is performing a task (e.g. classification) with some level of success.
- An adversary is attacking that model.
- Objective: to perform the task (maintaining a similar accuracy) under attack.



$\boldsymbol{x}$	$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$	$\boldsymbol{x} + \epsilon\, \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
$y =$"panda" w/ 57.7% confidence	"nematode" w/ 8.2% confidence	"gibbon" w/ 99.3% confidence

- Forms of adversarial image attacks:
  - Untargeted adversarial attacks: *cannot* control output label of adversarial image.
  - Targeted adversarial attacks: *can* control output label of image.

Goodfellow et al. Explaining and Harnessing Adversarial Examples

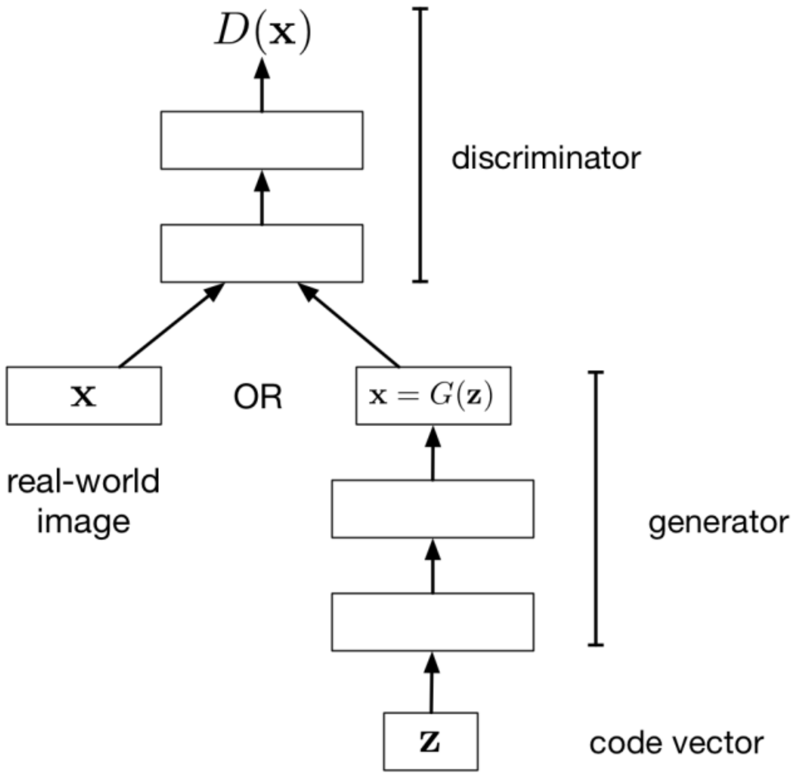
Adversarial Examples

- Two types of attacks in DL: White-Box and Black-Box attacks
  - White-Box: Attacker has access to training method (data/network initialization/algorithm/hyperparameter).
    - Small perturbations --> bad performance.
  - Black-Box attacks: Attacker does not have complete access to network training method.

- We need robust models: weight decay and dropout do not work.
- Approach: Brute force method to generate adversarial examples and train model using them (Adversarial training).

Generative Adversarial Networks

- Generative Adversarial Networks (GANs): train two different networks
  - Generator network tries to produce realistic-looking samples
  - Discriminator network tries to figure out whether an image came from training set or generator network
  - Generator network tries to fool the discriminator network



Adversarial Examples

- Fast Gradient Sign Method (FGSM) effective method to generate adversarial images.

1. Take input image --> Make prediction (using CNN).
2. Compute loss of prediction based on *true* class label.
3. Calculate gradients of loss with respect to input image.
4. Compute gradient sign --> Use to construct adversarial image (output).

Goodfellow et al. Explaining and Harnessing Adversarial Examples

Adversarial Training

- Models learns to classify correctly adversarial examples.
- Classifiers uses a loss function to minimize model prediction errors.
- After training, attacker uses loss function to maximize model prediction error
  - 1. Compute its gradient with respect to model input
  - 2. Take the sign of gradient and multiply it by a threshold

- Normal (top)
- Noisy (middle)
- Adversarial (bottom) MNIST dataset

Adversarial training

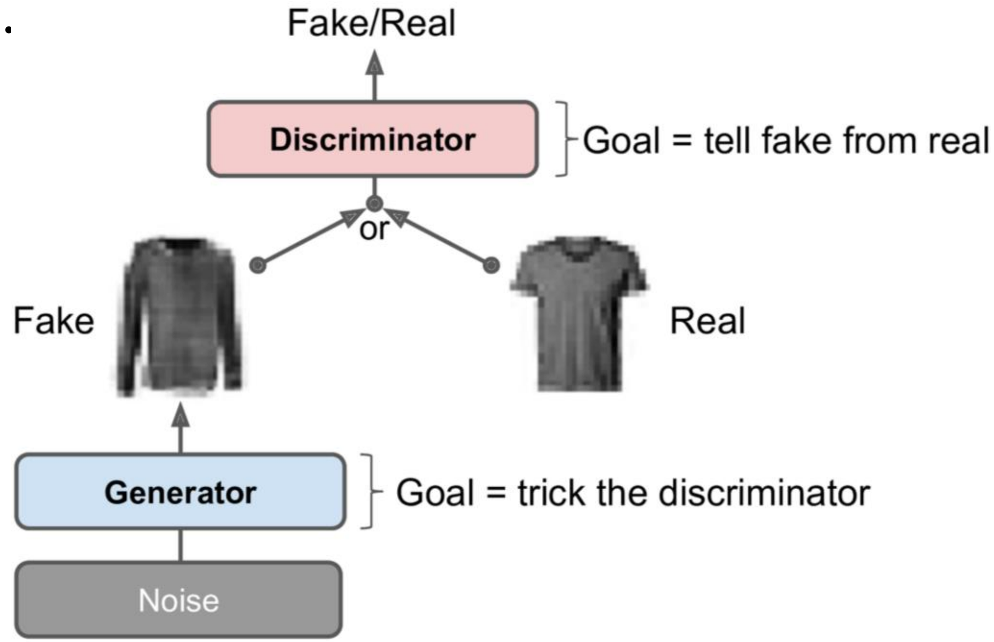
- Adversarial training is process of training a model to correctly classify both unmodified examples and adversarial examples.
  - Adv: robust adversarial examples, generalize performance for original examples.
- Virtual adversarial training* extends idea of adversarial training to semi-supervised regime and unlabelled examples.
- Traditional adversarial and virtual adversarial training can be interpreted as regularization strategy and as defence against malicious inputs.

*Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. Distributional smoothing with virtual adversarial training. In ICLR, 2016.

Generative Adversarial Networks (GANs)

- Idea: let NNs to compete against each other to perform better.

- GAN composed of two neural networks with different objectives
  - Generator: Takes a random distribution as input (typically Gaussian) and outputs some data typically, an image.
  - Discriminator: Takes either a fake image from generator or a real image from training set as input and guess whether input image is fake or real.



Ian Goodfellow et al., "Generative Adversarial Nets," Proceedings of the 27th International Conference on Neural Information Processing Systems 2 (2014): 2672–2680.

Generative Adversarial Networks

- Each training iteration is divided into two phases:
  - Discriminator training: Batch of real images (label 1) is sampled from training set and is completed with an equal number of fake images (label 0) produced by the generator. [Uses binary cross-entropy loss].
  - Generator training: Produce another batch of fake images, and once again discriminator is used to tell whether images are fake or real. Do not add real images in the batch, and all labels are set to 1 (real).
    - Never actually sees any real images.

```
codings_size = 30
generator = keras.models.Sequential([
keras.layers.Dense(100, activation="selu",
input_shape=[codings_size]),
keras.layers.Dense(150, activation="selu"),
keras.layers.Dense(28 * 28, activation="sigmoid"),
keras.layers.Reshape([28, 28])
])
```

```
discriminator = keras.models.Sequential([
keras.layers.Flatten(input_shape=[28, 28]),
keras.layers.Dense(150, activation="selu"),
keras.layers.Dense(100, activation="selu"),
keras.layers.Dense(1, activation="sigmoid")
])

gan = keras.models.Sequential([generator, discriminator])
```

Difficulties of Training GANs

1. Generator and discriminator constantly try to outsmart each other lead to no player would be better off changing their own strategy, assuming other players do not change theirs.

2. Generator produces perfectly realistic images and discriminator is forced to guess (50% real, 50% fake) (Not guaranteed).

3. Generator's outputs gradually become less diverse (called mode collapse).

4. Generator and discriminator are constantly pushing against each other, parameters may end up oscillating and becoming unstable.

Deep Convolutional GANs: DCGANs

- DCGANs (2015): build GANs based on deeper convolutional nets for larger images.
- Guidelines to building DCGANs:
    - Replace any pooling layers with strided convolutions (discriminator) and transposed convolutions (generator).
    - Use Batch Normalization in both generator and discriminator, except in generator's output layer and discriminator's input layer.
    - Remove fully connected hidden layers for deeper architectures.
    - Use ReLU activation in generator for all layers except output layer, which should use tanh.
    - Use leaky ReLU activation in discriminator for all layers.

Reference

- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Aurélien Géron.
- Building Machine Learning and Deep Learning Models on Google Cloud Platform By E. Bisong.