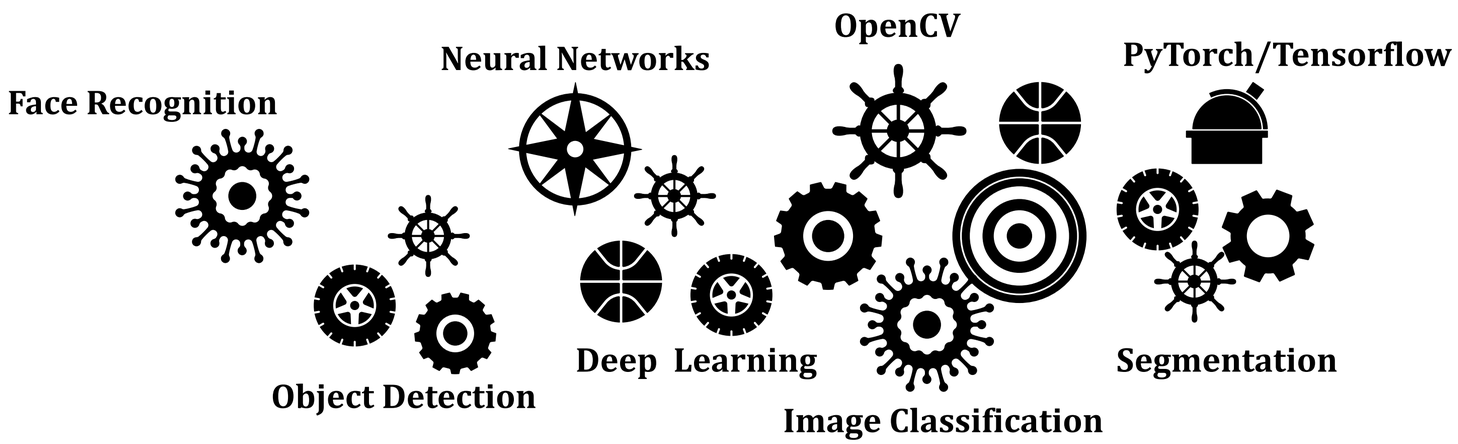
MPSSDEGB
कौशल से रोजगार, समृद्धि अपार

FOUNDATION FOR INNOVATION AND TECHNOLOGY TRANSFER
भारतीय प्रौद्योगिकी संस्थान दिल्ली
Indian Institute of Technology Delhi

Computer Vision



```
[2] import numpy as np
    import tensorflow as tf
    from keras.datasets import mnist
    from tensorflow.keras.models import Sequential
    #from ann_visualizer.visualize import ann_viz
    from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
    from tensorflow.keras.utils import to_categorical
    import matplotlib.pyplot as plt
```

```
[3] from keras.callbacks import CSVLogger
```

```
csv_logger = CSVLogger("model_history_log.csv", append=True)
```

+ Code + Text

```
[6] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
#loding the images
#train_images = mnist.train_images()
#train_labels = mnist.train_labels()
print(len(train_labels))
#test_images = mnist.test_images()
#test_labels = mnist.test_labels()
print(len(test_labels))

# Normalize the images. Pixel values are between 0-255 in image learning it is
# good practice to normalize your data to a smaller range like between 0 and 1.
train_images = (train_images / 255) - 0.5
#print(train_images)
test_images = (test_images / 255) - 0.5
#print(test_images)
```

```
60000
10000
```

```
[10] #lets give the hyper parameters
     num_filters = 1
     filter_size = 9
     pool_size = 2

     # Model is being trained on 1875 batches of 32 images each, not 1875 images. 1875*32 = 60000 images
     # 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'),
     ])
```

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

Cost function used is Cross Entropy Loss

$$J = -\frac{1}{n}\sum_{i=1}^{n} y_i \log p_i$$

$$CE = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} y_i \log p_i$$

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

```
[12] tf.keras.callbacks.EarlyStopping(
           monitor="loss",
           min_delta=0,
           patience=0,
           verbose=0,
           mode="auto",
           baseline=None,
           restore_best_weights=False,
         )
```

```
<keras.src.callbacks.EarlyStopping at 0x7d66cd1b0b80>
```

```
callback_1 = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=2)
```

The content is added from different sources

```python
# Train the model.
history = model.fit(
  train_images,
  to_categorical(train_labels),
  epochs=10,
  validation_data=(test_images, to_categorical(test_labels)),
  callbacks=[csv_logger, callback_1],
)
```

```
Epoch 1/10
1875/1875 [==============================] - 28s 14ms/step - loss: 2.3017 - accuracy: 0.1086 - val_loss: 2.2996 - val_accuracy: 0.1135
Epoch 2/10
1875/1875 [==============================] - 25s 14ms/step - loss: 2.2979 - accuracy: 0.1148 - val_loss: 2.2921 - val_accuracy: 0.1135
Epoch 3/10
1875/1875 [==============================] - 25s 14ms/step - loss: 2.2795 - accuracy: 0.1437 - val_loss: 2.2528 - val_accuracy: 0.1945
Epoch 4/10
1875/1875 [==============================] - 25s 14ms/step - loss: 2.1817 - accuracy: 0.4567 - val_loss: 2.0541 - val_accuracy: 0.5886
Epoch 5/10
1875/1875 [==============================] - 25s 14ms/step - loss: 1.7909 - accuracy: 0.6685 - val_loss: 1.4641 - val_accuracy: 0.7365
Epoch 6/10
1875/1875 [==============================] - 25s 14ms/step - loss: 1.2079 - accuracy: 0.7600 - val_loss: 0.9762 - val_accuracy: 0.7945
Epoch 7/10
1875/1875 [==============================] - 27s 14ms/step - loss: 0.8720 - accuracy: 0.8036 - val_loss: 0.7503 - val_accuracy: 0.8296
Epoch 8/10
1875/1875 [==============================] - 25s 14ms/step - loss: 0.7021 - accuracy: 0.8265 - val_loss: 0.6252 - val_accuracy: 0.8446
Epoch 9/10
1875/1875 [==============================] - 25s 14ms/step - loss: 0.6035 - accuracy: 0.8419 - val_loss: 0.5480 - val_accuracy: 0.8563
Epoch 10/10
1875/1875 [==============================] - 25s 14ms/step - loss: 0.5459 - accuracy: 0.8520 - val_loss: 0.5043 - val_accuracy: 0.8657
```

```
[15] model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 20, 20, 1)         82

 max_pooling2d (MaxPooling2  (None, 10, 10, 1)         0
 D)

 flatten (Flatten)           (None, 100)               0

 dense (Dense)               (None, 10)                1010

=================================================================
Total params: 1092 (4.27 KB)
Trainable params: 1092 (4.27 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

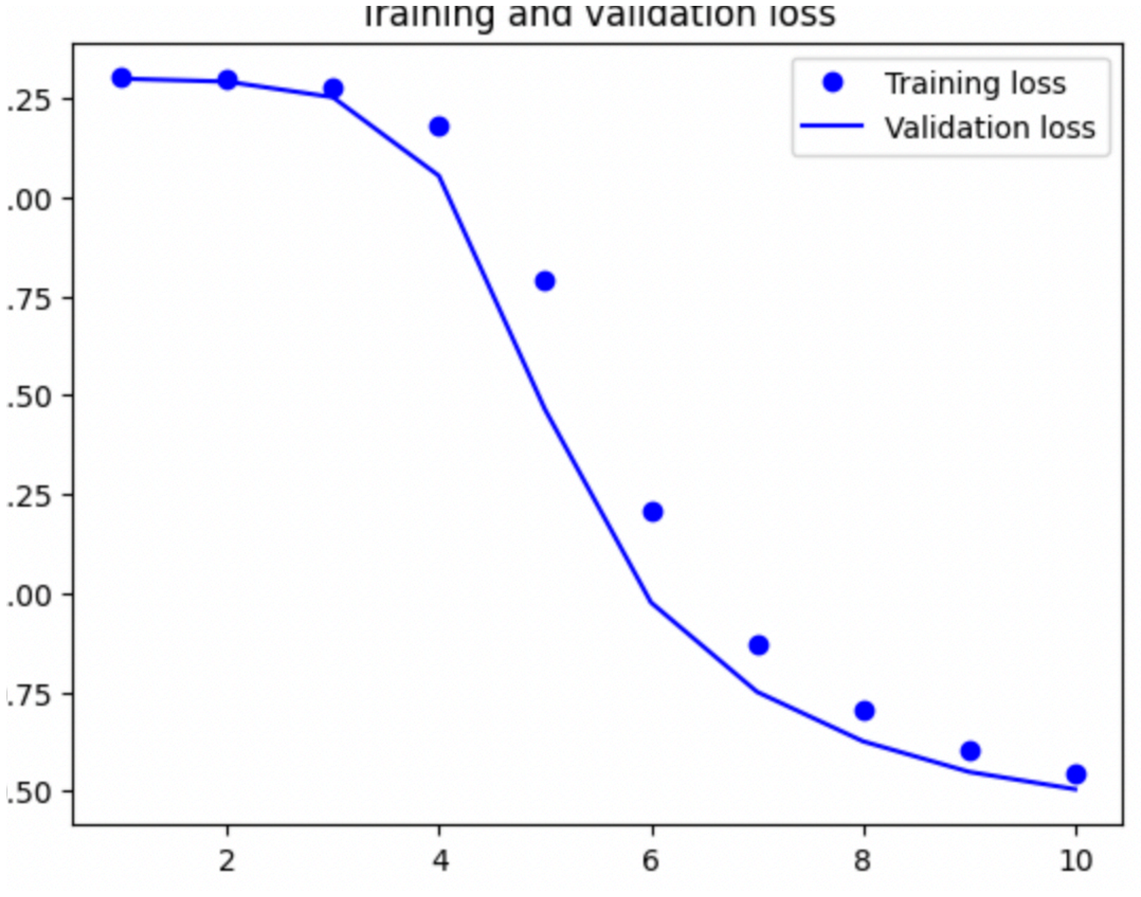
```
[16] #Save the model:
     model.save_weights('cnn.h5')

     # Load the model's saved weights.
     model.load_weights('cnn.h5')
```

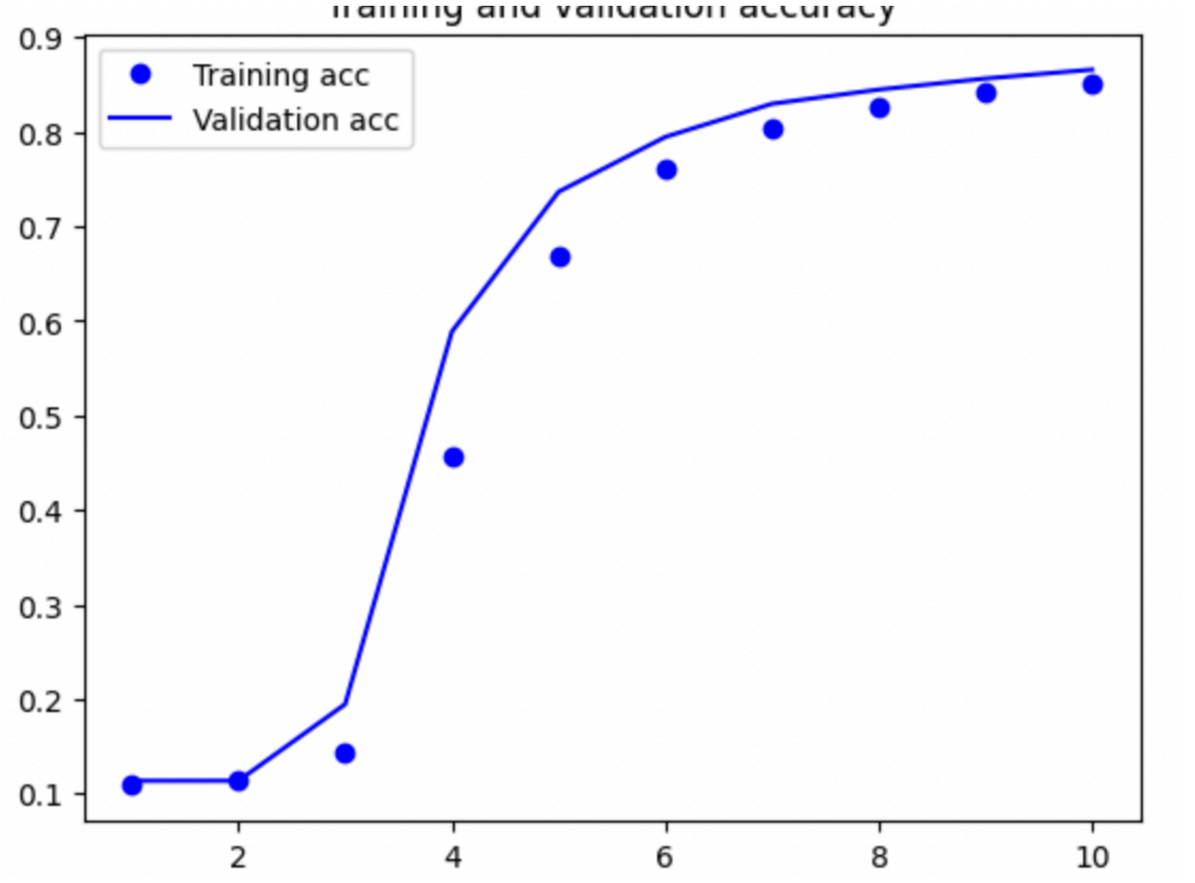
```
     # Predict on the first 10 test images.
     predictions = model.predict(test_images[:10])

     # 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[:10]) # [7, 2, 1, 0, 4]
```

```
1/1 [==============================] - 0s 91ms/step
[7 2 1 0 4 1 4 9 2 9]
[7 2 1 0 4 1 4 9 5 9]
```

```
[18] #Callback records events into a History object.
     accuracy = history.history['accuracy']
     val_accuracy = history.history['val_accuracy']
     loss = history.history['loss']
     val_loss = history.history['val_loss']
```

```
#The range of epochs, from 1 to the total number of epochs, is often used to plot the learning curves of the model.
#the range function is used to generate a list of integers from 1 to the length of the accuracy list plus one.
# This is because the accuracy list is typically recorded at the end of each epoch, starting from the first epoch.
epochs = range(1, len(accuracy) + 1)

plt.plot(epochs, accuracy, 'bo', label='Training acc')
plt.plot(epochs, val_accuracy, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



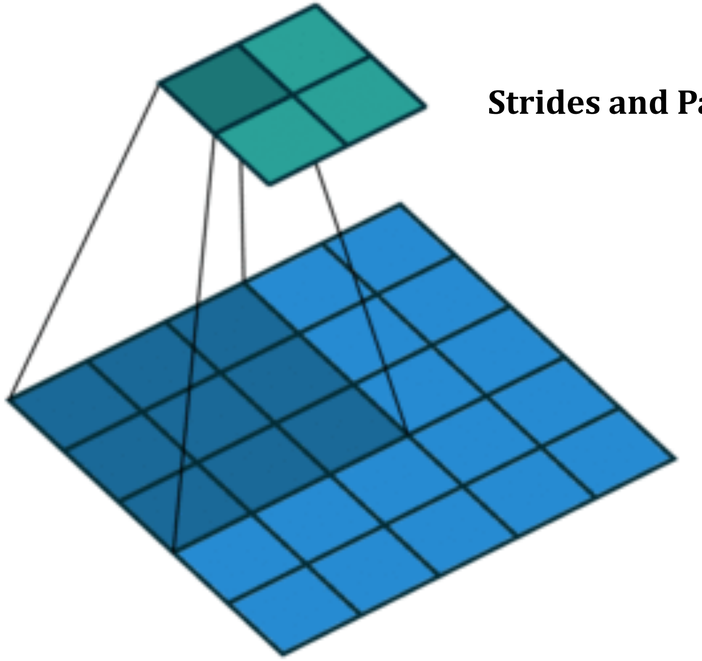


The content is added from different sources

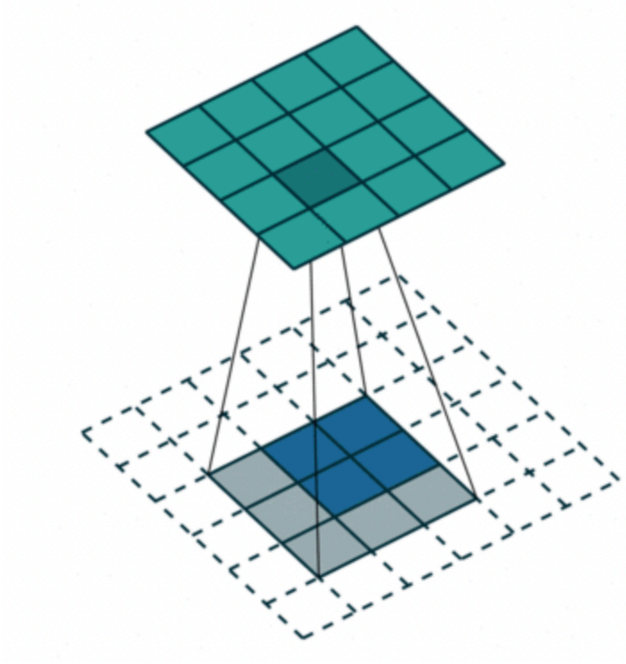
The content is added from different sources

Convolution

Strides and Padding



Convolution



Transpose Convolution

https://medium.com/@marsxiang/convolutions-transposed-and-deconvolution-6430c358a5b6

Convolution

stride=1

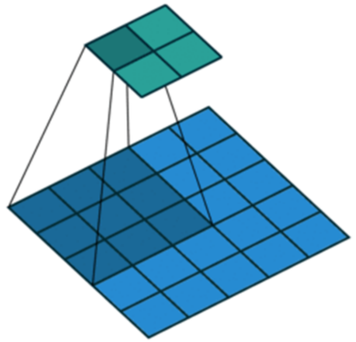
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

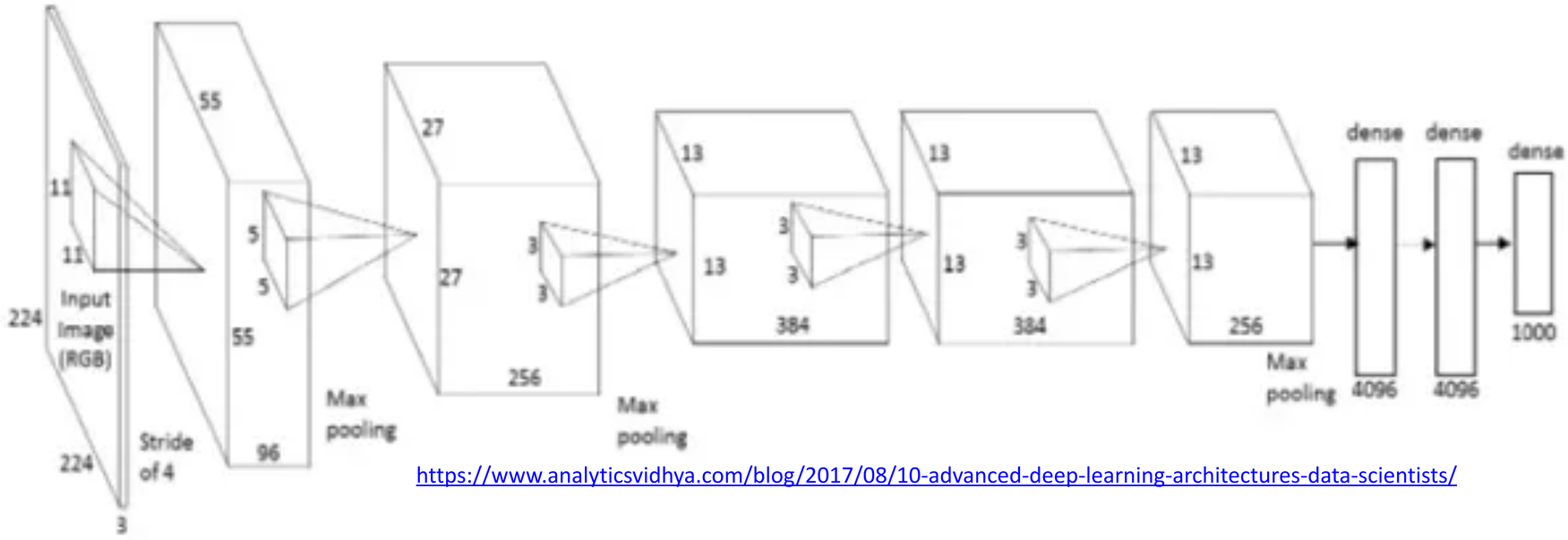
Dot product

1	-1	-1
-1	1	-1
-1	-1	1

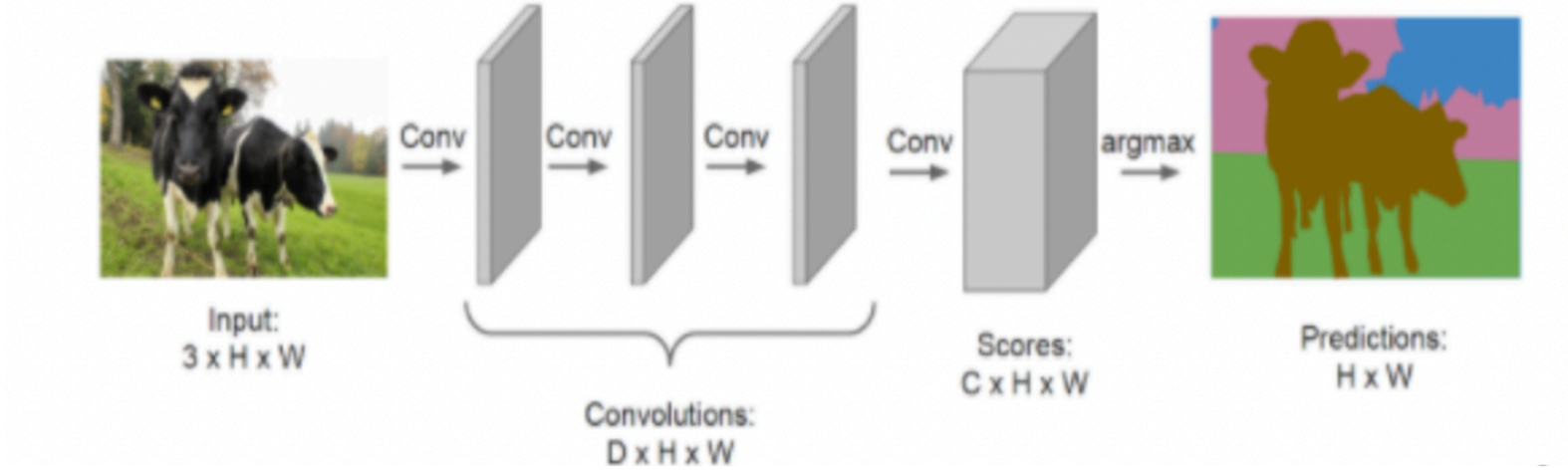
1*1 + 0*(-1) + 0*(-1) +
0*(-1) + 1*1 + 0*(-1) +
0*(-1) + 0*(-1) + 1*1



Fully Convolutional Network



https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/

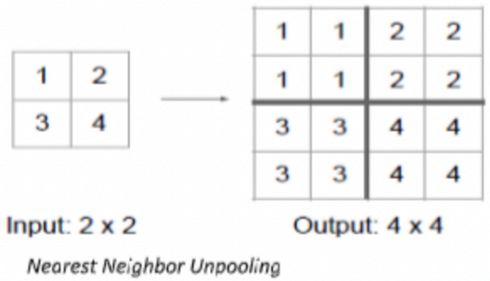


https://www.mygreatlearning.com/blog/fcn-fully-convolutional-network-semantic-segmentation/

The content is added from different sources

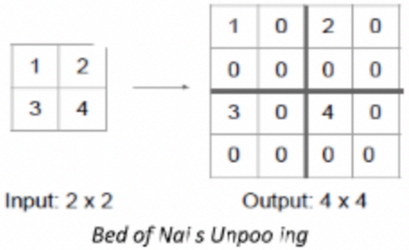
Fully Convolutional Network

1. Nearest Neighbor



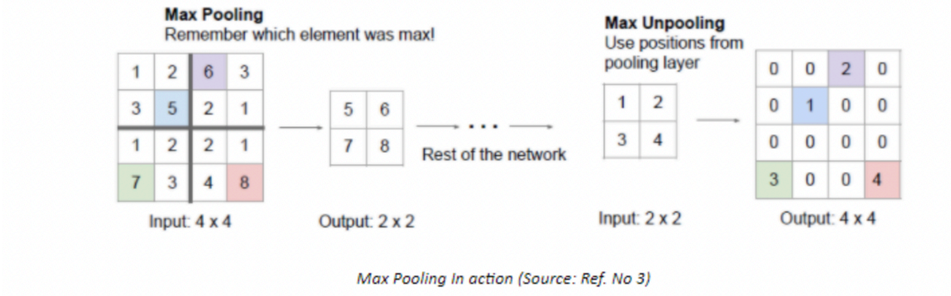
Nearest Neighbor Unpooling

2. Bed of Nails



Bed of Nails Unpooling

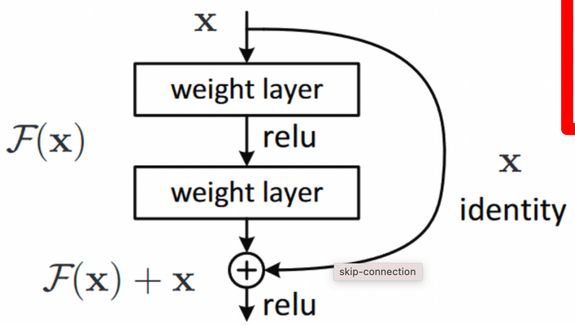
3. Max Unpooling



Max Pooling In action (Source: Ref. No 3)

https://www.mygreatlearning.com/blog/fcn-fully-convolutional-network-semantic-segmentation/ The content is added from different sources

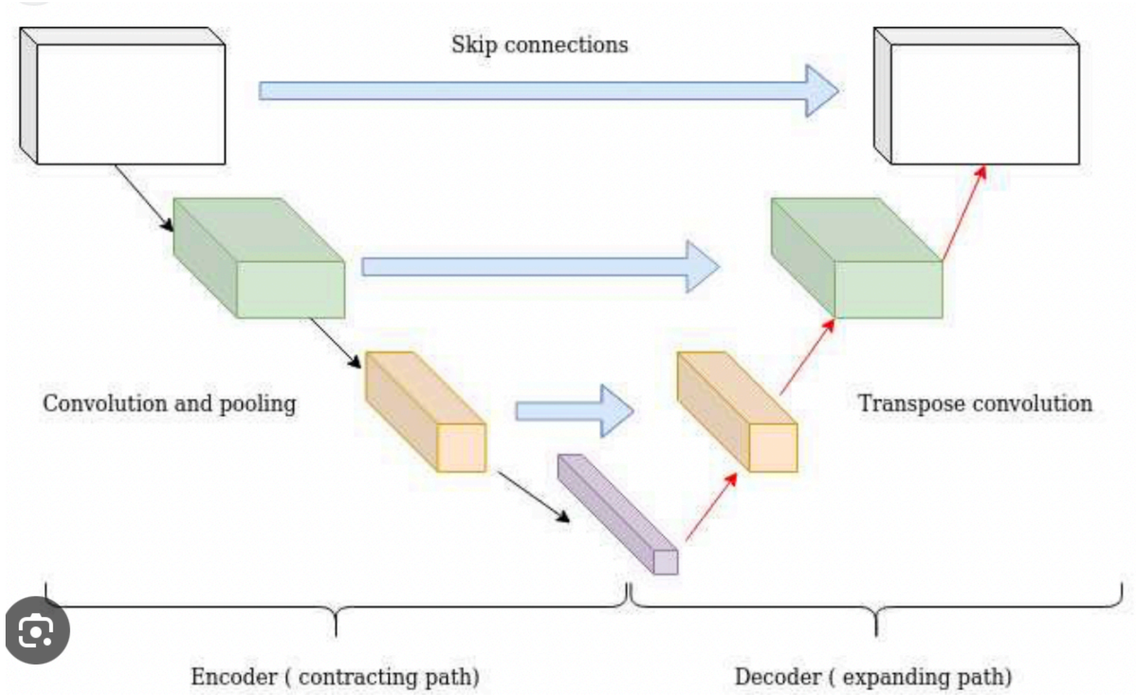
Fully Convolutional Network



x
identity

a) **short** skip connections

b) **long** skip connections.

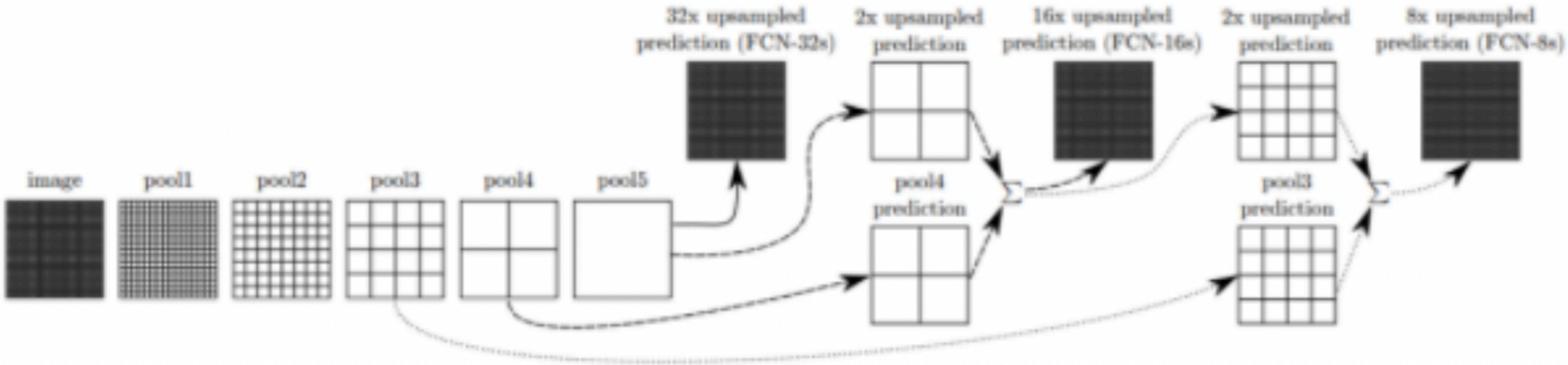


https://theaisummer.com/skip-connections/

https://www.mygreatlearning.com/blog/fcn-fully-convolutional-network-semantic-segmentation/

The content is added from different sources

Fully Convolutional Network



https://www.mygreatlearning.com/blog/fcn-fully-convolutional-network-semantic-segmentation/

The content is added from different sources

Generative adversarial Networks

- Generative Adversarial Networks (GANs) are a powerful class of neural networks that are used for unsupervised learning. It was developed and introduced by Ian J. Goodfellow in 2014. GANs are basically made up of a system of two competing neural network models which compete with each other and are able to analyze, capture and copy the variations within a dataset.
- GANs can create anything whatever we feed to them, as it Learn-Generate-Improve.

- The generative model can be thought of as analogous to a team of counterfeiters,trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the fake currency.
- Competition in this game drivesboth teams to improve their methods until the counterfeits are indistiguishable from the genuine articles.

https://www.slideshare.net/PrakharRastogi23/generative-adversarial-network-gan

The content is added from different sources

Generative adversarial Networks

- Most of the mainstream neural nets can be easily fooled into misclassifying things by adding only a small amount of noise into the original data. Sometimes the model after adding noise has higher confidence in the wrong prediction than when it predicted correctly. The reason for such adversary is that most machine learning models learn from a limited amount of data, which is a huge drawback, as it is prone to overfitting. Also, the mapping between the input and the output is almost linear and even a small change in a point in the feature space might lead to misclassification of data.

- Generative:

To learn a generative model, which describes how data is generated in terms of a probabilistic model.

- Adversarial:

The training of a model is done in an adversarial setting.

- Networks:

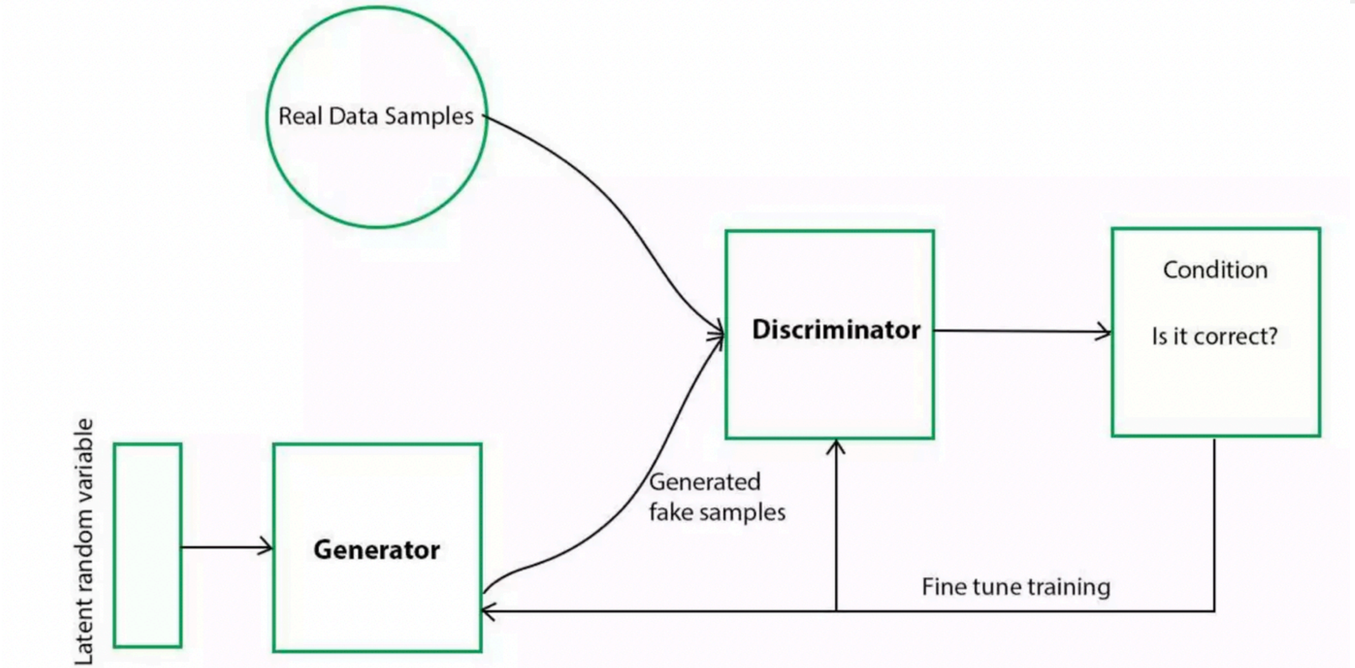
Use deep neural networks as the artificial intelligence (AI) algorithms for training purpose.

https://www.slideshare.net/PrakharRastogi23/generative-adversarial-network-gan

The content is added from different sources

Generative adversarial Networks

- In GANs, there is a generator and a discriminator. The Generator generates fake samples of data and tries to fool the Discriminator. The Discriminator, on the other hand, tries to distinguish between the real and fake samples. The Generator and the Discriminator are both Neural Networks and they both run in competition with each other in the training phase. The steps are repeated several times and in this, the Generator and Discriminator get better and better in their respective jobs after each repetition.

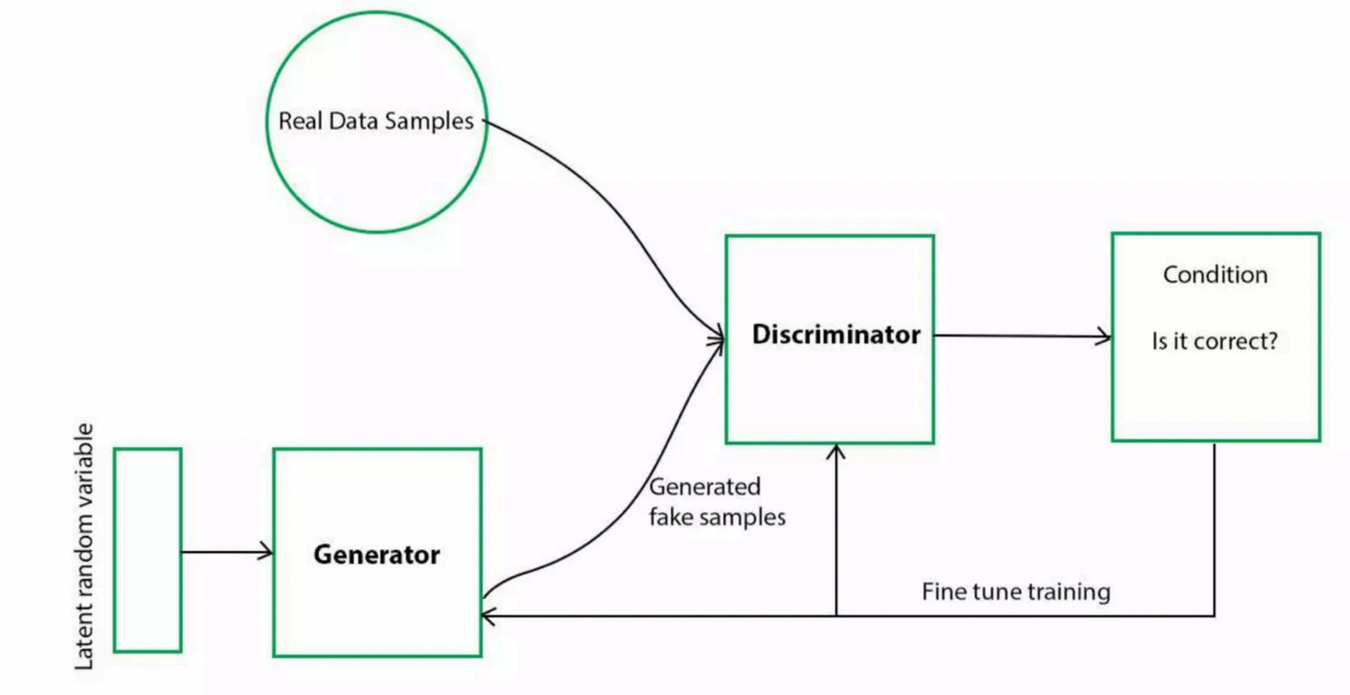


https://www.slideshare.net/PrakharRastogi23/generative-adversarial-network-gan

...e content is added from different sources

Generative adversarial Networks

- Here, the generative model captures the distribution of data and is trained in such a manner that it tries to maximize the probability of the Discriminator in making a mistake. The Discriminator, on the other hand, is based on a model that estimates the probability that the sample that it got is received from the training data and not from the Generator.

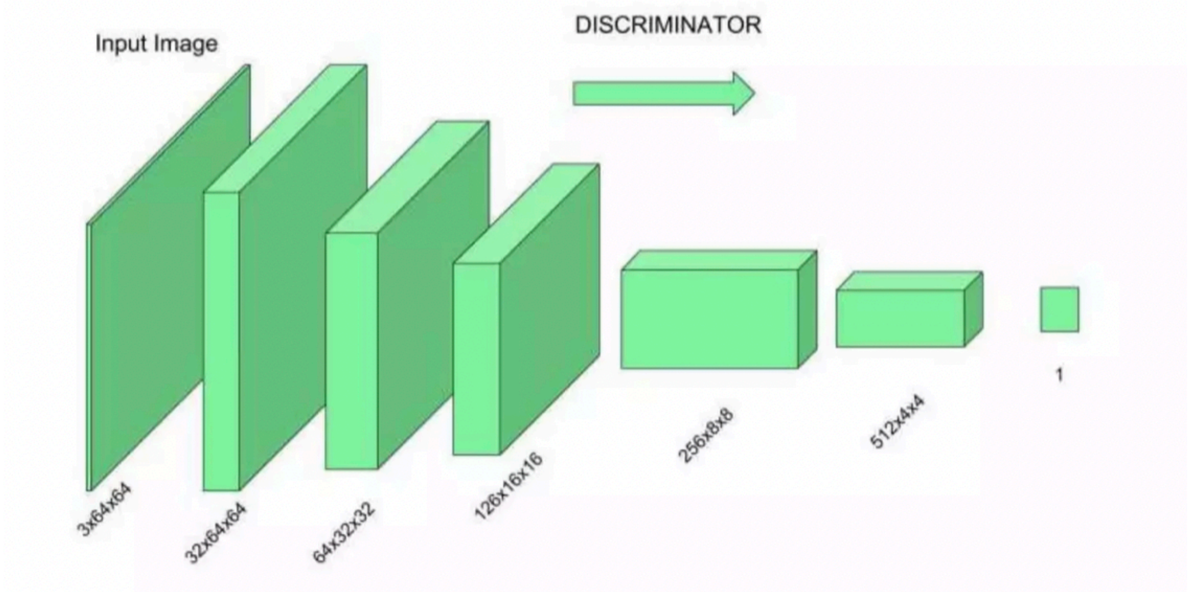


https://www.slideshare.net/PrakharRastogi23/generative-adversarial-network-gan

The content is added from different sources

Discriminator

- This part of GANs can be considered similar to what CNNs does. Discriminator is a Convolutional Neural Network consisting of many hidden layers and one output layer, the major difference here is the output layer of GANs can have only two outputs, unlike CNNs.
- The output of the discriminator can either be 1 or 0 because of a specifically chosen activation function for this task, if the output is 1 then the provided data is real and if the output is 0 then it refers to it as fake data.
- Discriminator is trained on the real data so it learns to recognize how actual data looks like and what features should the data have to be classified as real.



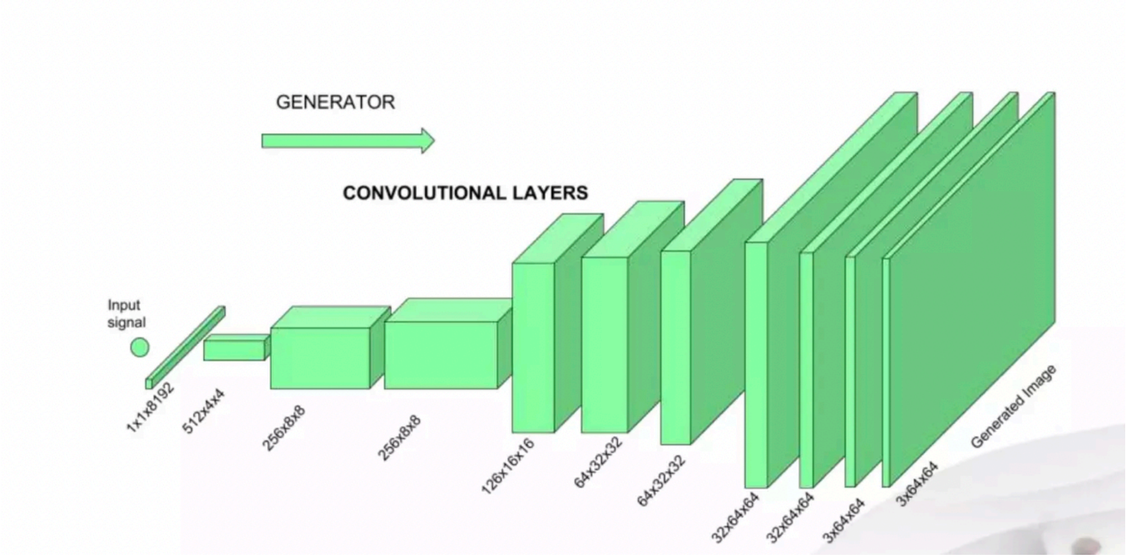
https://www.slideshare.net/PrakharRastogi23/generative-adversarial-network-gan

The content is added from different sources

Generator

Generator is an Inverse Convolutional Neural Net, it does exactly opposite of what a CNN does, because in CNN an actual image is given as an input and a classified label is expected as an output but in Generator, a random noise (a vector having some values to be precise) is given as an input to this Inverse CNN and an actual image is expected as an output.

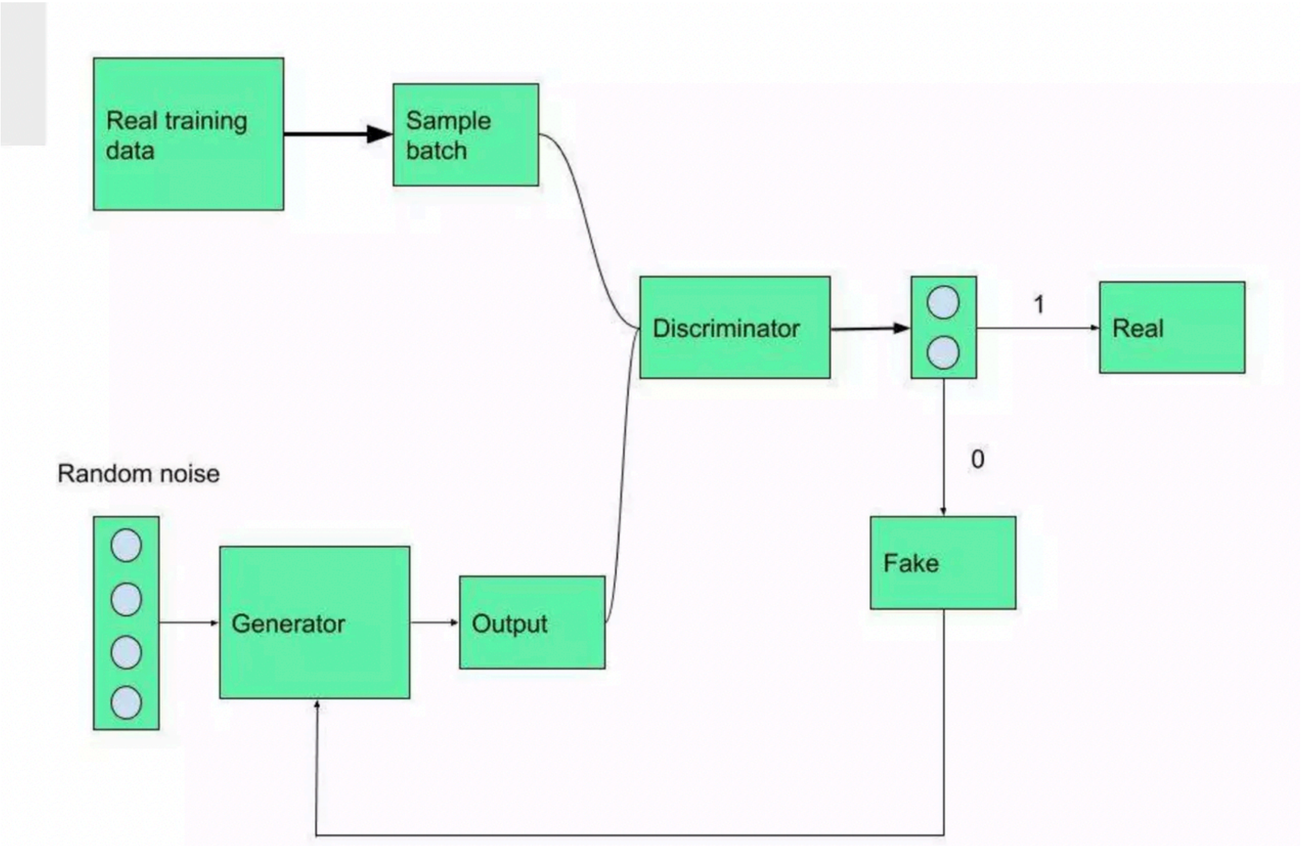
In simple terms, it generates data from a piece of data using its own imagination.



https://www.slideshare.net/PrakharRastogi23/generative-adversarial-network-gan

Learning

Generator starts to generate data from a random input and then that generated data is passed to Discriminator as input now Discriminator analyzes the data and checks how close it is to be classified as real, if the generated data does not contain enough features to be classified as real by the Discriminator, then this data and weights associated with it are sent back to the Generator using backpropagation.

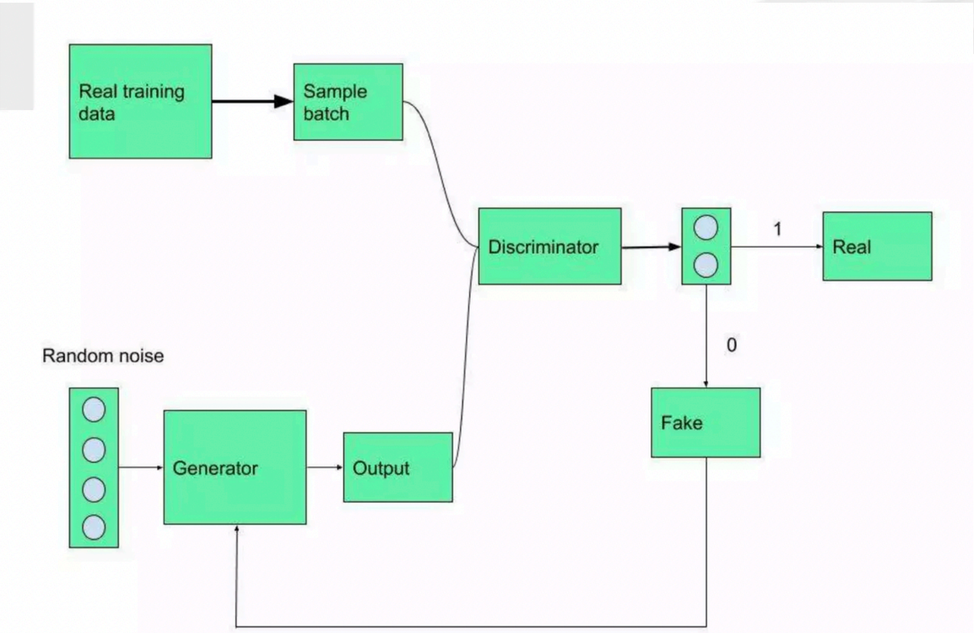


https://www.slideshare.net/PrakharRastogi23/generative-adversarial-network-gan

content is added from different sources

Learning

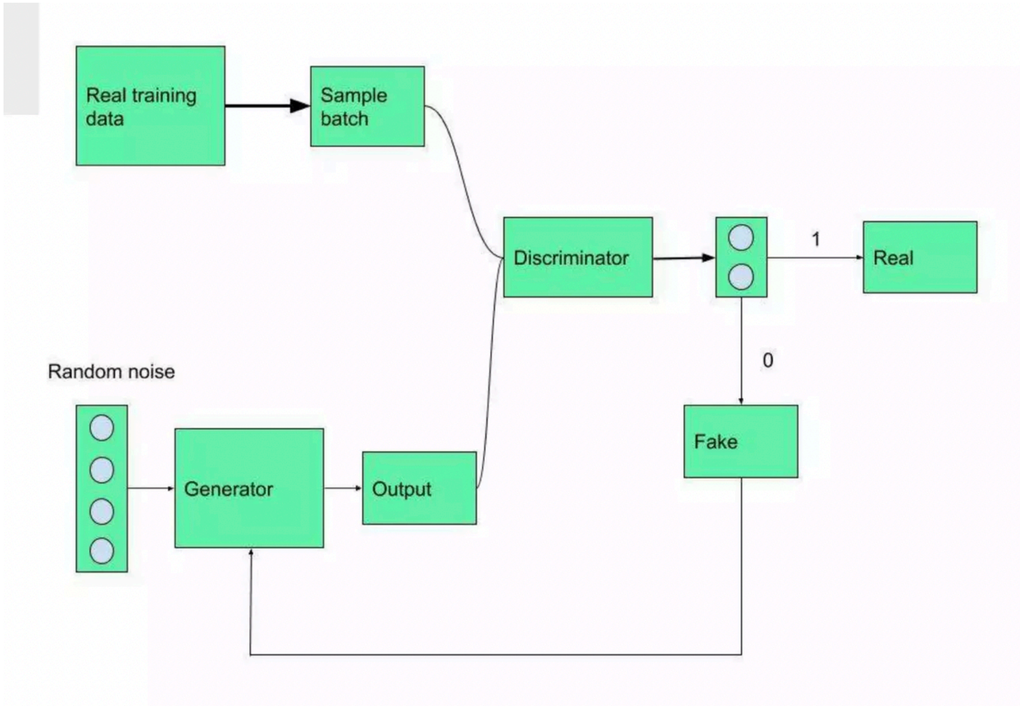
- so that it can re-adjust the weights associated with the data and create new data which is better than the previous one. This freshly generated data is again passed to the Discriminator and it continues.
- This process keeps repeating as long as the Discriminator keeps classifying the generated data as fakes, for every time data is classified as fake and with every backpropagation the quality of data keeps getting better and better and there comes a time when the Generator becomes so accurate that it becomes tough to distinguish between the real data and the data generated by the Generator.



https://www.slideshare.net/PrakharRastogi23/generative-adversarial-network-gan

Learning

- Train Two models-
- a generative model (G) that captures the data distribution, and a discriminative model (D) that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake.



https://www.slideshare.net/PrakharRastogi23/generative-adversarial-network-gan