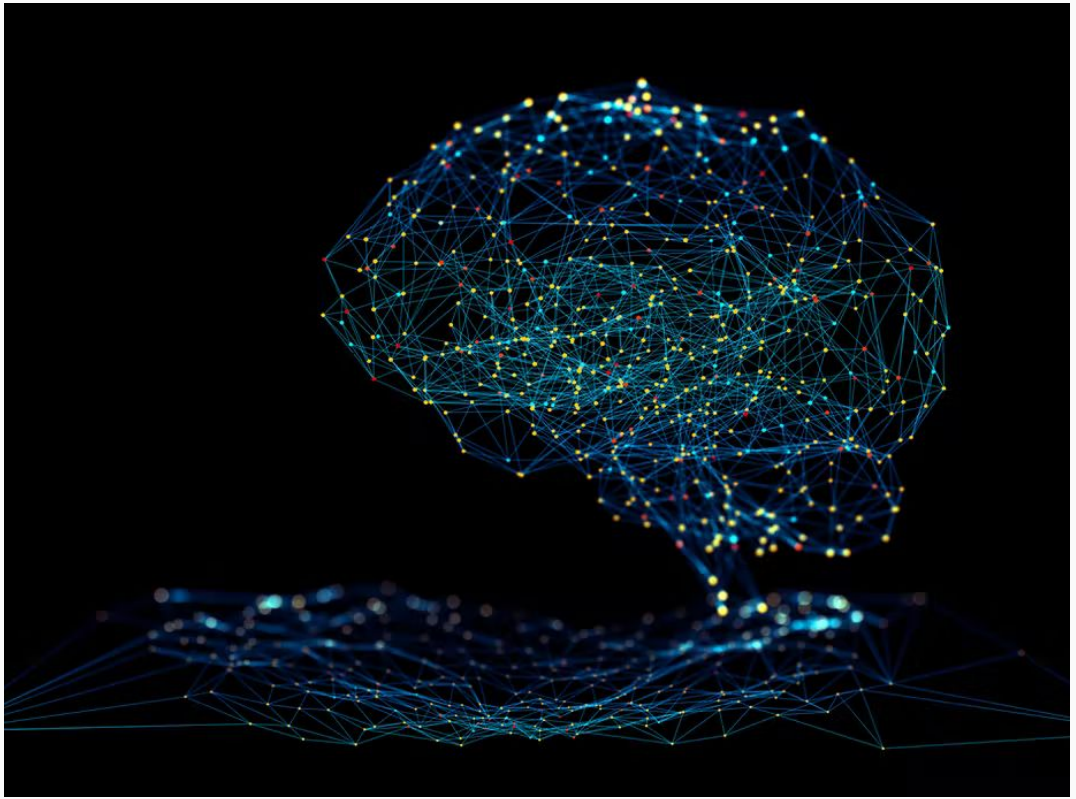
Neural Networks

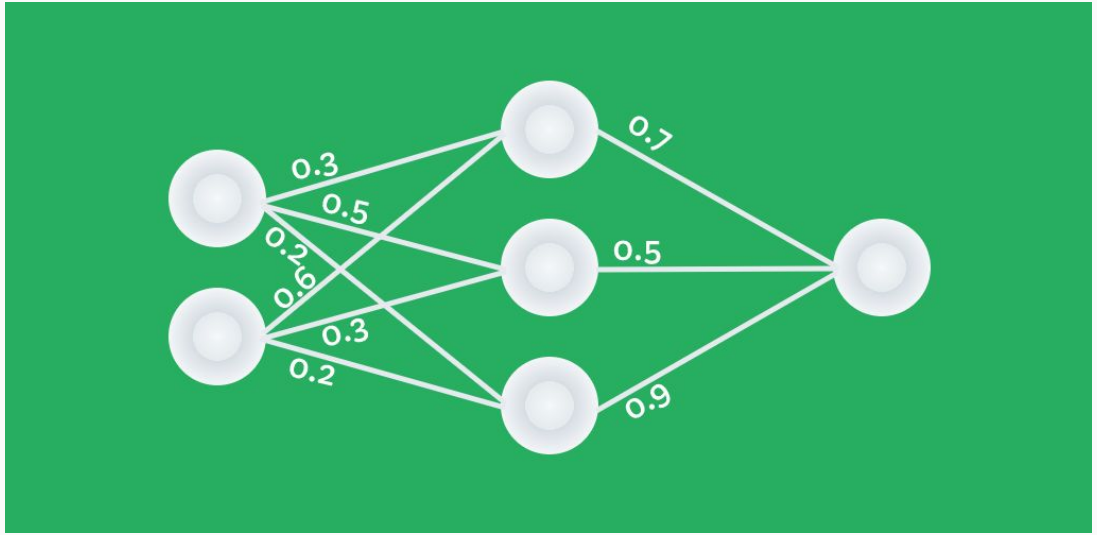
Neural Networks

Günümüzde bir çok makine öğrenmesi modeli mevcuttur. Neural Network bunlardan sadece bir tanesidir. İnsan beyni ve sinir sisteminden esinlenerek keşfedilmiş bir modelidir.

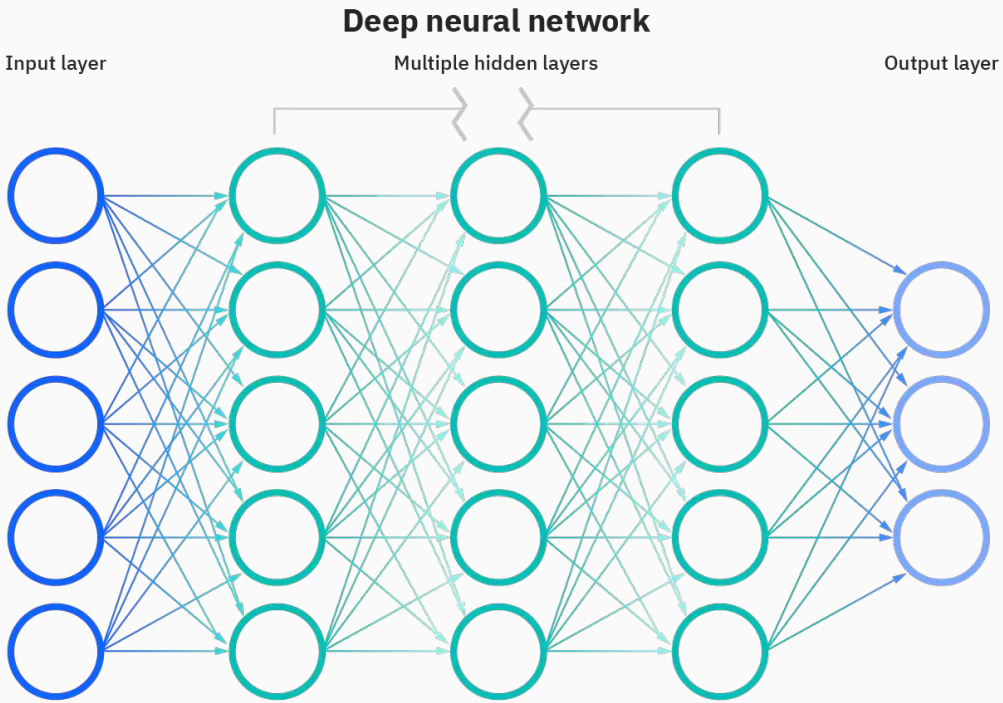


What Are Neural Networks

Neural Network katmanlar şeklinde kurulmuş bir yapıdır. İlk katman giriş, son katman çıkış olarak adlandırılır. Orta kısımda bulunan katmanlar ‘Hidden Layers’ yani gizli katmanlar olarak adlandırılmaktadır. Her katman belli sayıda ‘Neuron’ içerir. Bu neuronlar birbirine ‘Synapse’lar ile bağlıdır. Synapselar bir katsayı barındırır. Bu katsayılar bağlı oldukları neurondaki bilginin ne kadar önemli olduğunu söylemektedir.

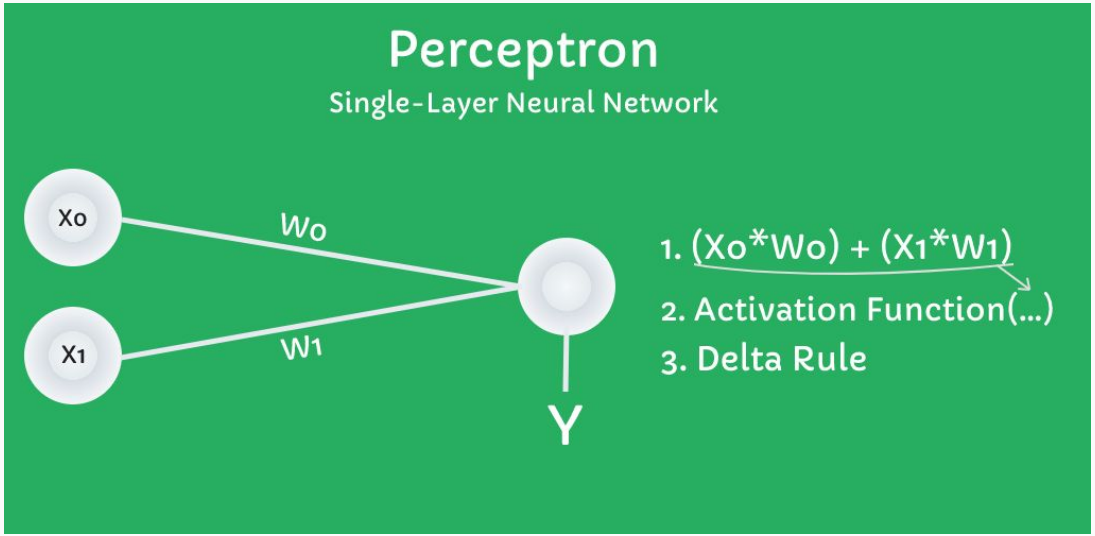


Bir neuronun değeri o neurona gelen girdilerin katsayılar ile çarpılıp toplanması sonucu bulunur. Bulunan bu sonuç bir aktivasyon fonksiyonu içerisine sokulur. Fonksiyondan çıkan sonuca göre o neuronun ateşlenip ateşlenmeyeceğine karar verilir.



Perceptron

Perceptron, tek katmanlı neural networktür. Fazla kompleks olmayan sorunların çözülmesinde yardımcı olmaktadır. Neural networklerde asıl istenen katsayıların eğitilmesidir. Katsayılar eğitilirken delta rule kullanılır.



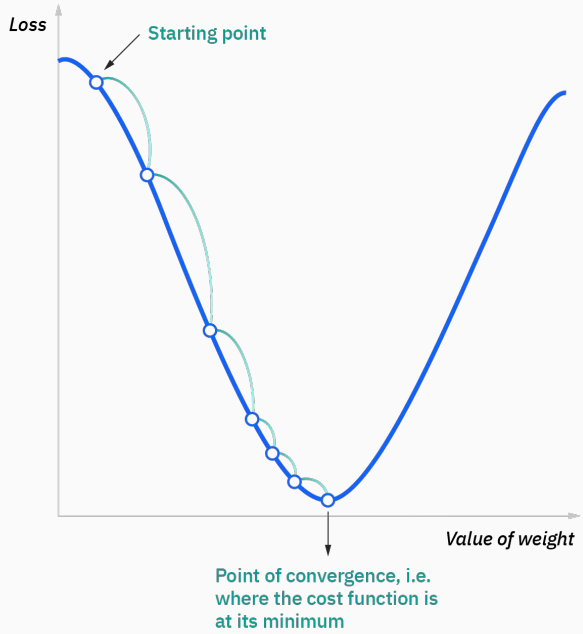
Delta Rule

Delta kuralı, sistemin hata oranını belirleyerek katsayıların güncellenmesidir. Bunu yaparken şöyle bir formül izlenir;

2*μ*Xi (beklenen — gerçekleşen)

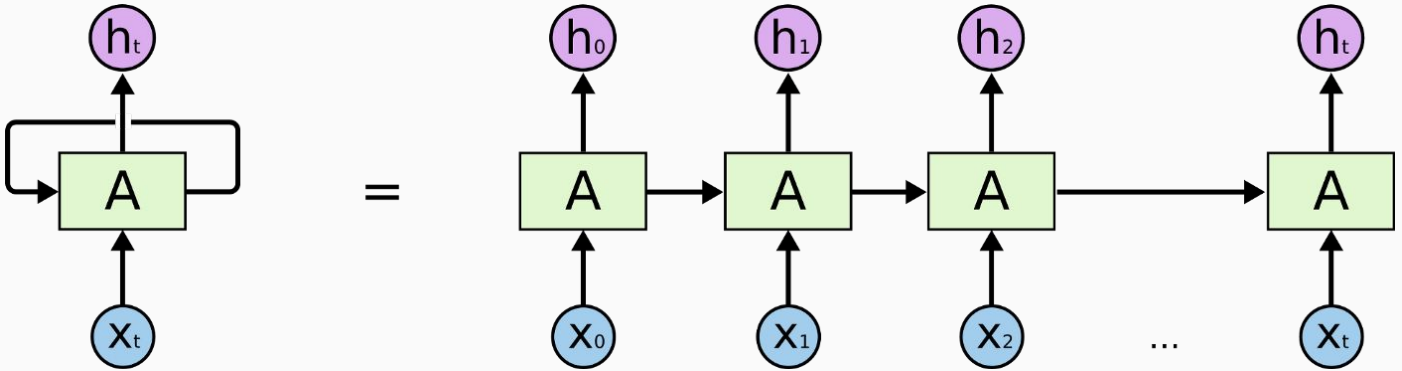
μ: Öğrenme Oranı

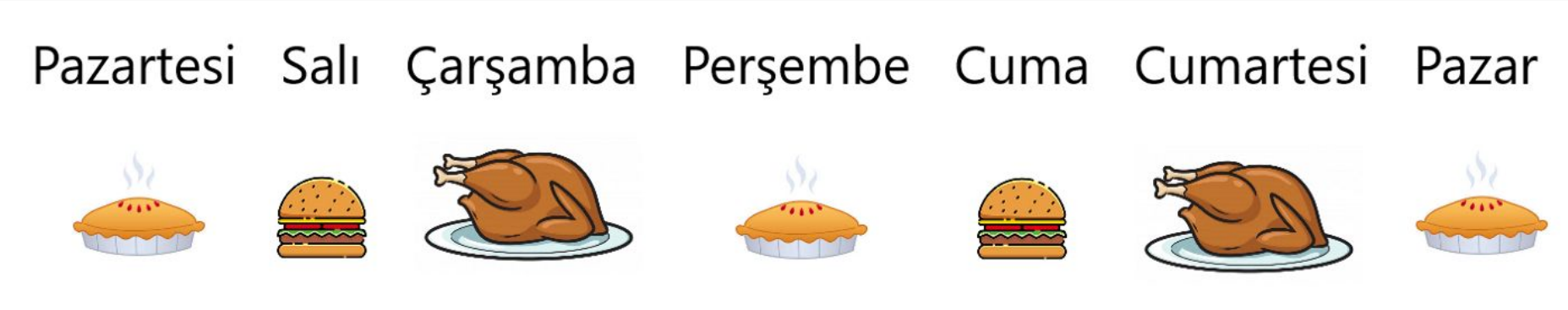
Öğrenme oranı, katsayılar üzerinde değişikliklerin ne kadar etki edeceğini belirtir. Öğrenme oranının çok düşük olması istenen değere uzun sürede ulaşmanızı sağlayabilir. Öğrenme oranını çok yüksek olması istenen değeri es geçmenizi ve tekrar o değere ulaşabileceğiniz süreyi uzatabilir. Öğrenme oranını seçerken dikkatli olunması gerekir.



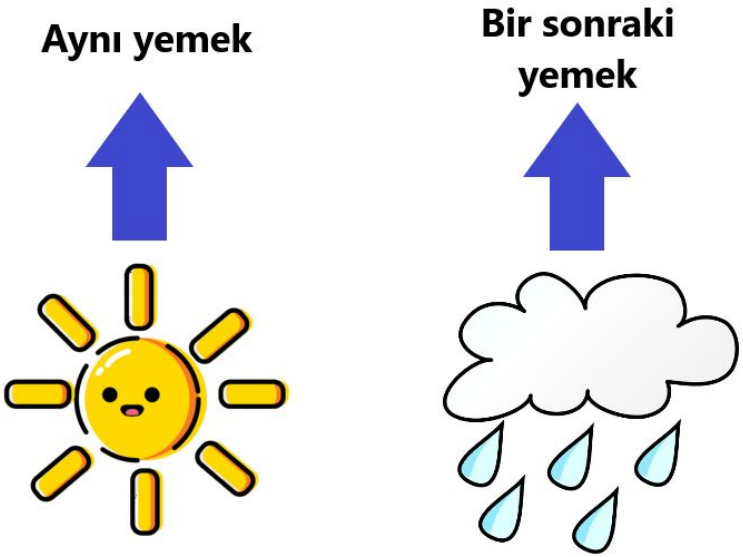
Recurrent Neural Networks

RNN’ler genelde bir sonraki adımı tahmin etmek için kullanılan bir çeşit Derin Öğrenme yapılarıdır. Diğer derin öğrenme yapılarından en büyük farkları ise hatırlamalarıdır.

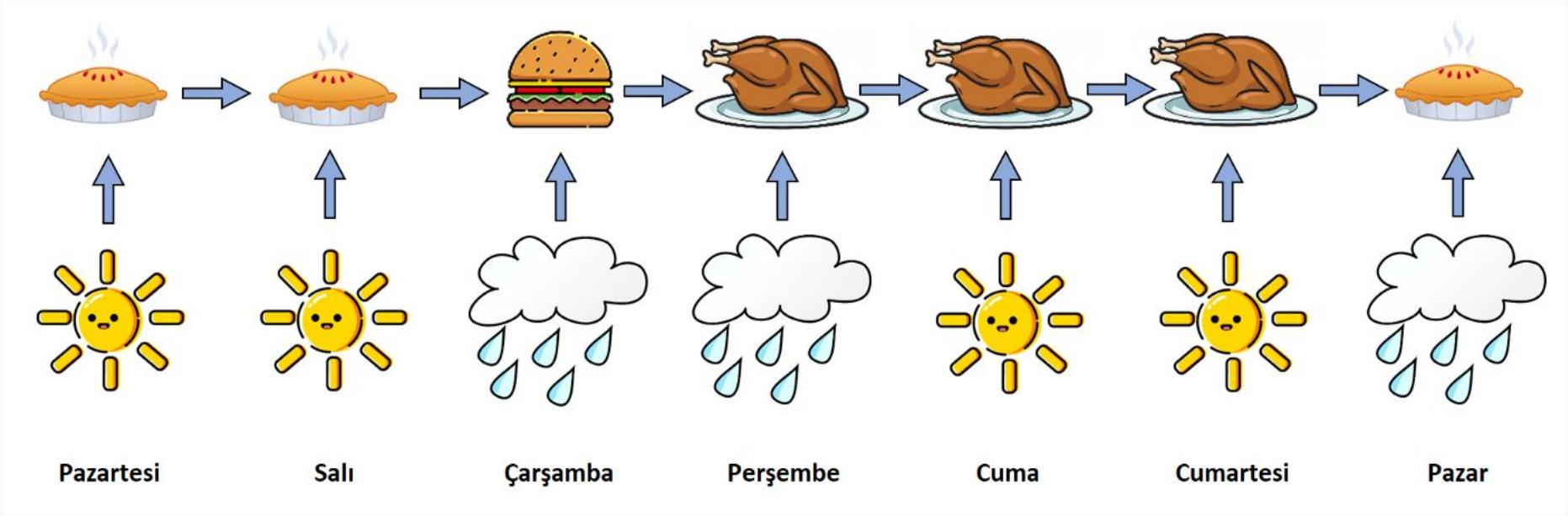




Yemekleri ***elmalı turta***, ***hamburger*** ve ***tavuk*** sırasıyla yapıyoruz



Bir diğer kural olarak hava güneşliyse o da dışarı çıkıyor ve bir önceki gün yemek olarak ne yaptıysa aynısını yapıyor. Eğer hava yağmurluysa dün yaptığı yemeği değiştiriyor.


Pazartesi
Salı
Çarşamba
Perşembe
Cuma
Cumartesi
Pazar

$h_{pazar} = f($  ,  $) =$ 

⚠ $h_{pazar} = f(h_{t-1}, x_t)$

Klasık derin öğrenme modellerinde sonuçlar birbirinden bağımsız çalışır.

Fakat RNN’lerde bir önceki adımın çıkışı, bir sonrakini etkiler.

Önceki adımlarda hesaplanan sonuçları hafızasında(memory) tutmaya çalışır.

Teoride RNN’ler uzun dizilerde(long-sequence) iyi sonuçlar verebiliyor olması gerekirken pratikte elde edilen tecrübeler sonucunda bunu başaramadığı görülür.

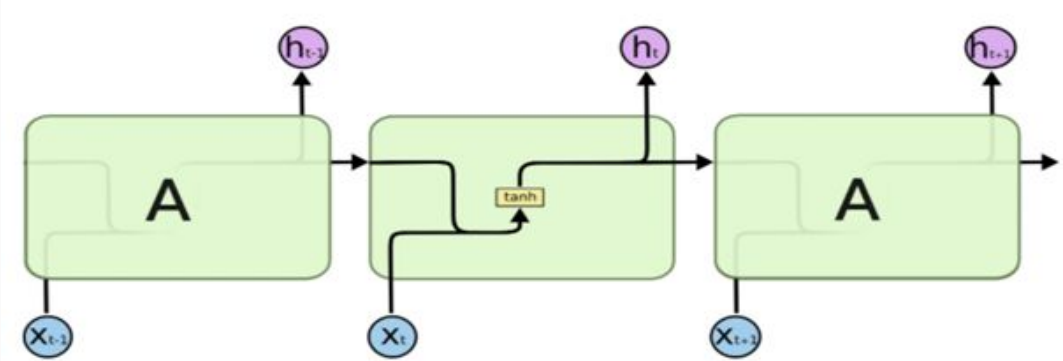
RNN’lerin kısa süreli bir hafızası vardır. Bu sebeple LSTM yöntemi kullanılır.

LSTM

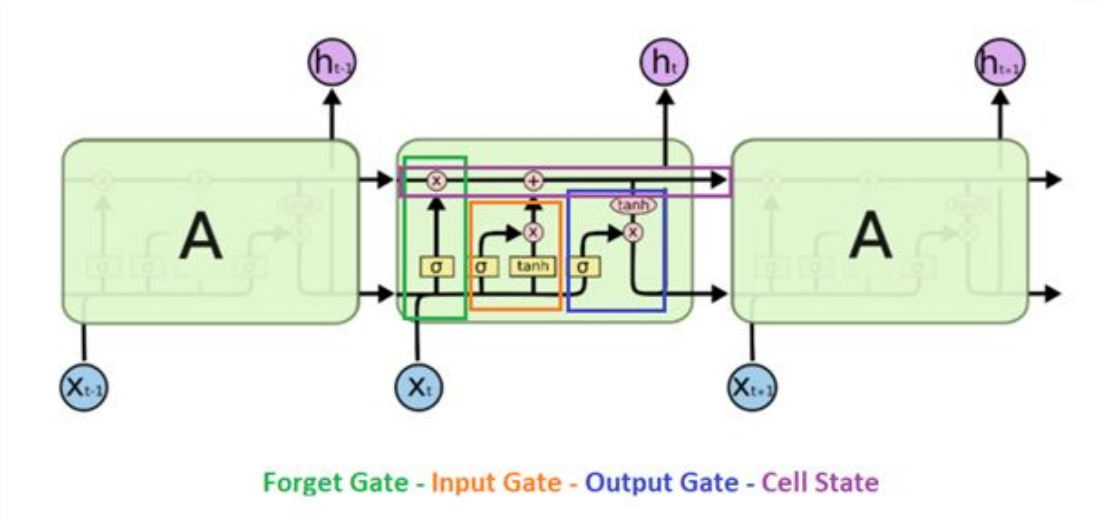
RNN kısa vadeli bir hafızası olmasından dolayı bazı problemlerde iyi çalışamaz.

Giriş verisi olarak yeterince uzun bir cümleyi RNN’e verdiğimizde, geçmiş bilgileri hatırlamakta yetersiz kalır ve bu nedenle tahmin yapamaz.

LSTM, bilgileri daha iyi depolayarak, standart RNN’in kısa vadeli bellek problemini ortadan kaldırmak için tasarlanmıştır.



RNN



LSTM

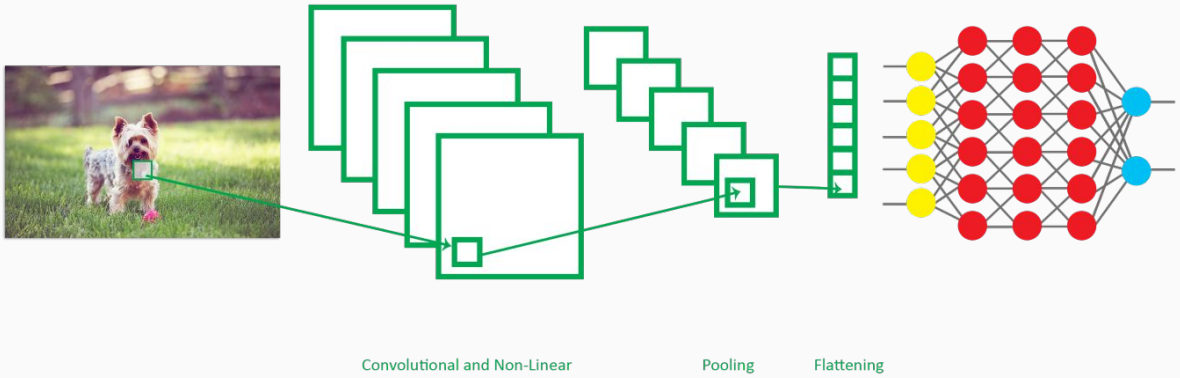
Standart bir RNN tek bir tanh katmanı içerirken, LSTM'ler iletişim halinde olan 4 farklı katman içerirler.

Unutma Kapısı(Forget Gate): Hangi bilgilerin unutulacağı veya tutulacağı kararına varan kapıdır.

Giriş Kapısı(Input Gate): Cell State güncellemesi yapar. Önceki ve mevcut bilginin sigmoid işlemi sonucuna göre güncelleme yapıp yapılmayacağı kararına varılır.

Çıkış Kapısı(Output Gate): Çıkış kapısı ise bir sonraki hücrenin girişini(ht+1) belirler. Ayrıca tahmin yapmak için de kullanılır.

Convolutional Neural Networks



Convolutional Layer: Özellikleri saptamak için kullanılır

Non-Linearity Layer: Sisteme doğrusal olmayanlığın (non-linearity) tanıtılması

Pooling (Downsampling) Layer: Ağırlık sayısını azaltır ve uygunluğu kontrol eder

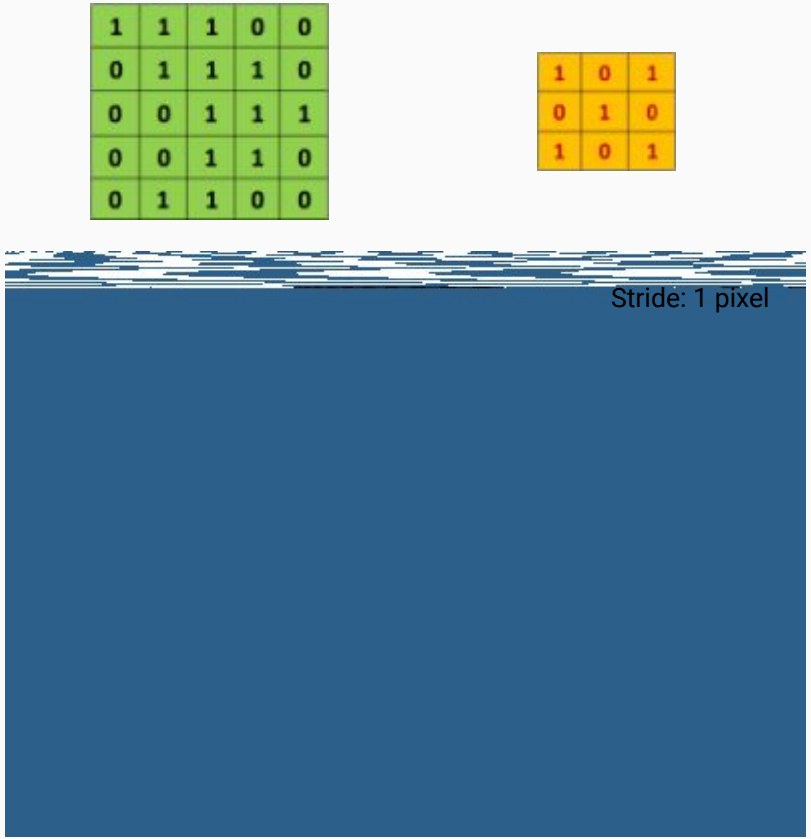
Flattening Layer: Klasik Sinir Ağı için verileri hazırlar

Fully-Connected Layer: Sınıflamada kullanılan Standart Sinir Ağı

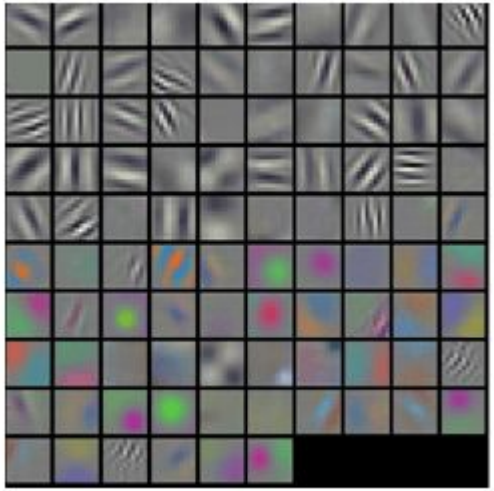
Temel olarak, Cnn, sınıflandırma sorununun çözümü için standart Sinir Ağı kullanır, ancak bilgileri belirlemek ve bazı özellikleri tespit etmek için diğer katmanları kullanır.

Convolutional Layer

Bu katman CNN'nin ana yapı taşıdır. Resmin özelliklerini algılamaktan sorumludur. Bu katman, görüntüdeki düşük ve yüksek seviyeli özellikleri çıkarmak için resme bazı fitreler uygular. Örneğin, bu filtre kenarları algılayacak bir filtre olabilir. Bu filtreler genellikle çok boyutludur ve piksel değerleri içerirler.(5x5x3) 5 matrisin yükseklik ve genişliğini, 3 matrisin derinliğini temsil eder.



İlk filtreyi uyguladığımızda, bir Feature Map oluşturuyor ve bir özellik türünü tespit ediyoruz. Ardından, ikinci bir filtre kullanıp başka bir özellik türünü algılayan ikinci bir Feature Map oluştururuz.



Non-linearity

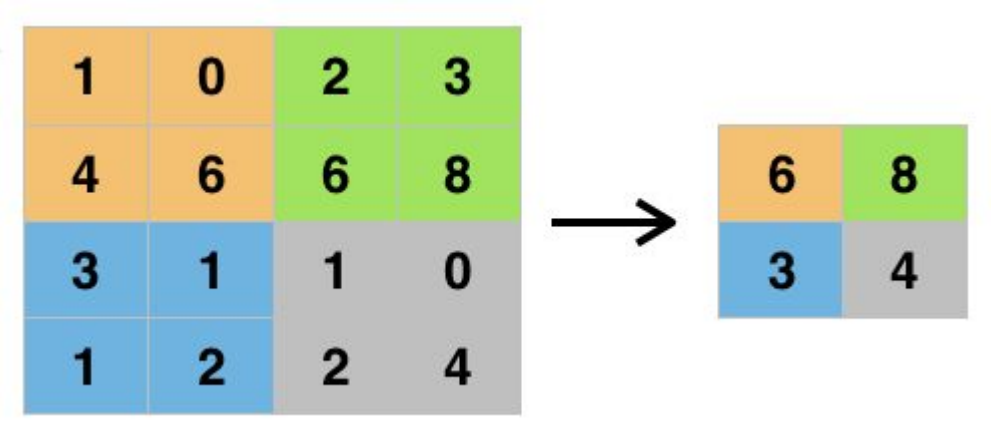
Tüm Convolutional katmanlarından sonra genellikle Non-Linearity(doğrusal olmayan) katmanı gelir.

Peki görüntüdeki doğrusallık neden bir problemdir? Sorun şu ki, tüm katmanlar doğrusal bir fonksiyon olabildiğinden dolayı Sinir Ağı tek bir perception gibi davranır, yani sonuç, çıktıların linear kombinasyonu olarak hesaplanabilir.

Bu katman aktivasyon katmanı (Activation Layer) olarak adlandırılır çünkü aktivasyon fonksiyonlarından birini kullanılır. Geçmişte, sigmoid ve tahn gibi doğrusal olmayan fonksiyonlar kullanıldı, ancak Sinir Ağı eğitiminin hızı konusunda en iyi sonucu Rectifier(ReLu) fonksiyonu verdiği için artık bu fonksiyon kullanılmaya başlanmıştır.

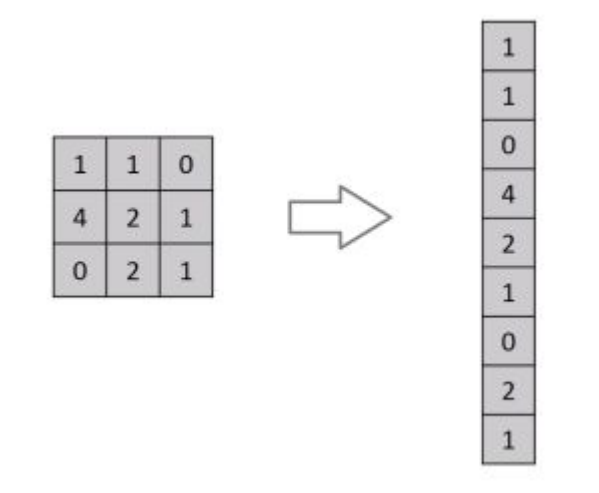
ReLu Fonksiyonu f (x) = max (0, x)

Pooling Layer

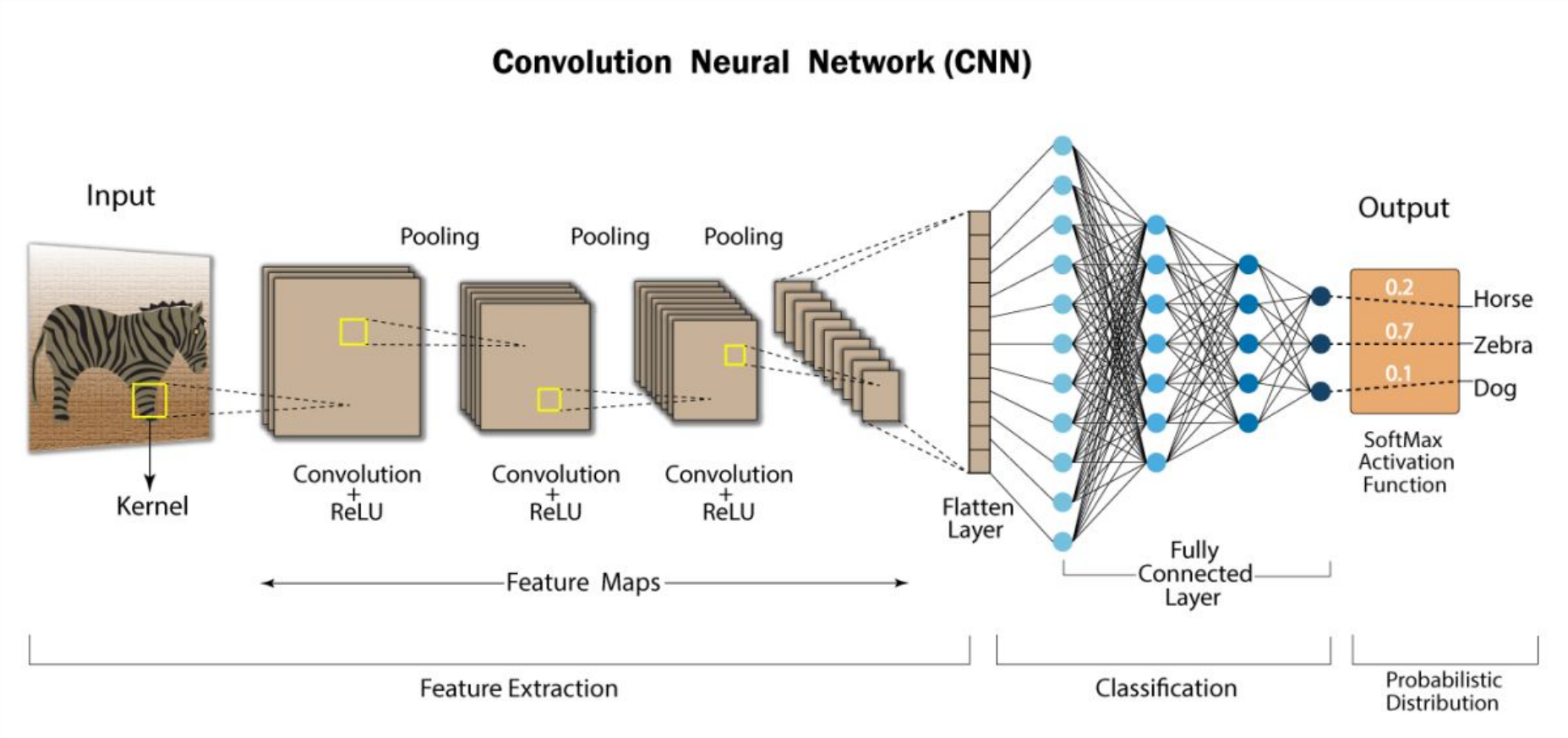


Bu katman, CovNet'teki ardışık convolutional katmanları arasına sıklıkla eklenen bir katmandır. Bu katmanın görevi, gösterimin kayma boyutunu ve ağ içindeki parametreleri ve hesaplama sayısını azaltmak içindir. Bu sayede ağdaki uyumsuzluk kontrol edilmiş olur. Birçok Pooling işlemleri vardır, fakat en popüleri max pooling'dir.

Flattening Layer



Fully-Connected Layer



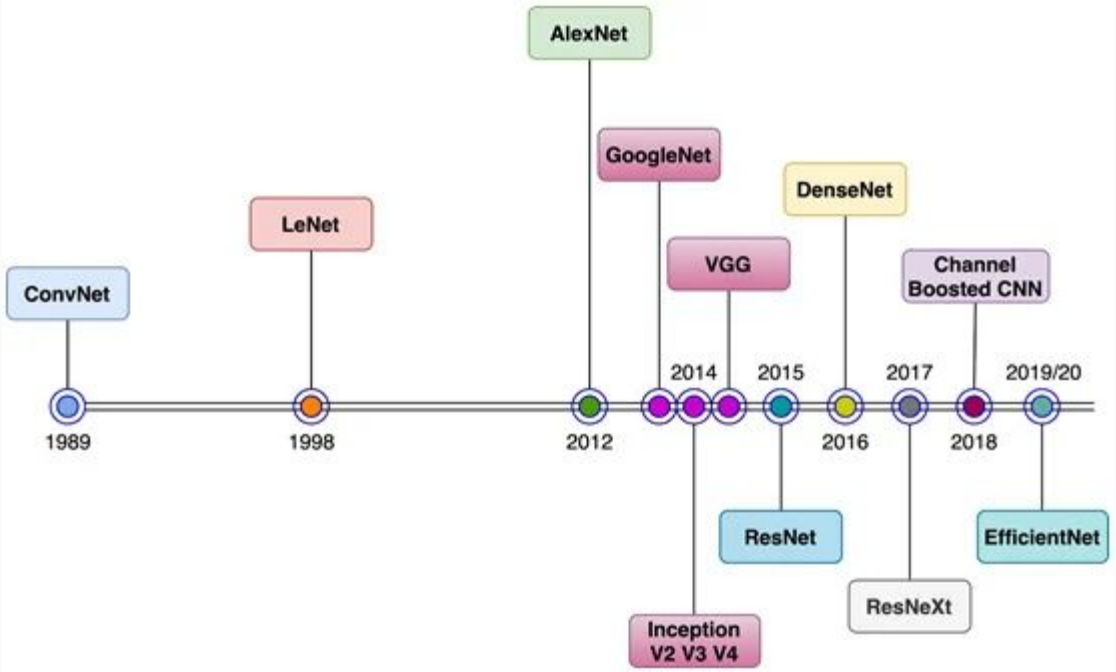
NN -> Her şey
RNN -> Time series
CNN -> Image processing
LSTM -> Time series

BERT

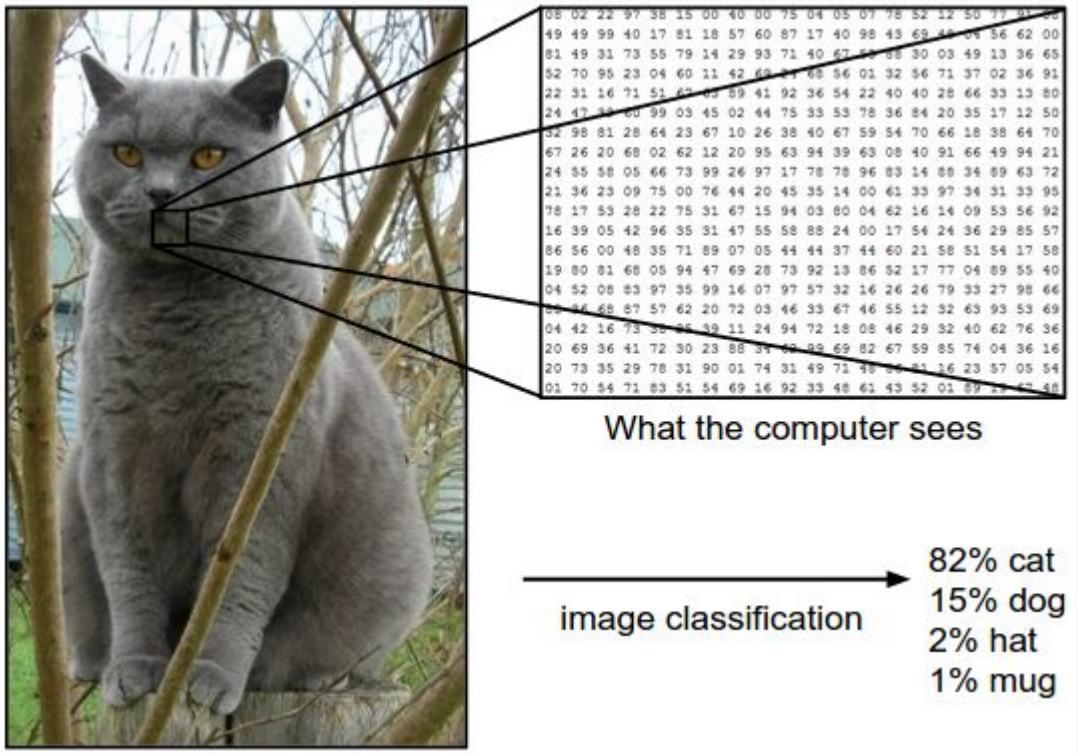
GPT

GAN

Image Classification


AlexNet
GoogleNet
DenseNet
LeNet
VGG
Channel Boosted CNN
ConvNet
2014
2015
2017
2019/20
1989
1998
2012
2016
2018
ResNet
EfficientNet
ResNeXt
Inception V2 V3 V4

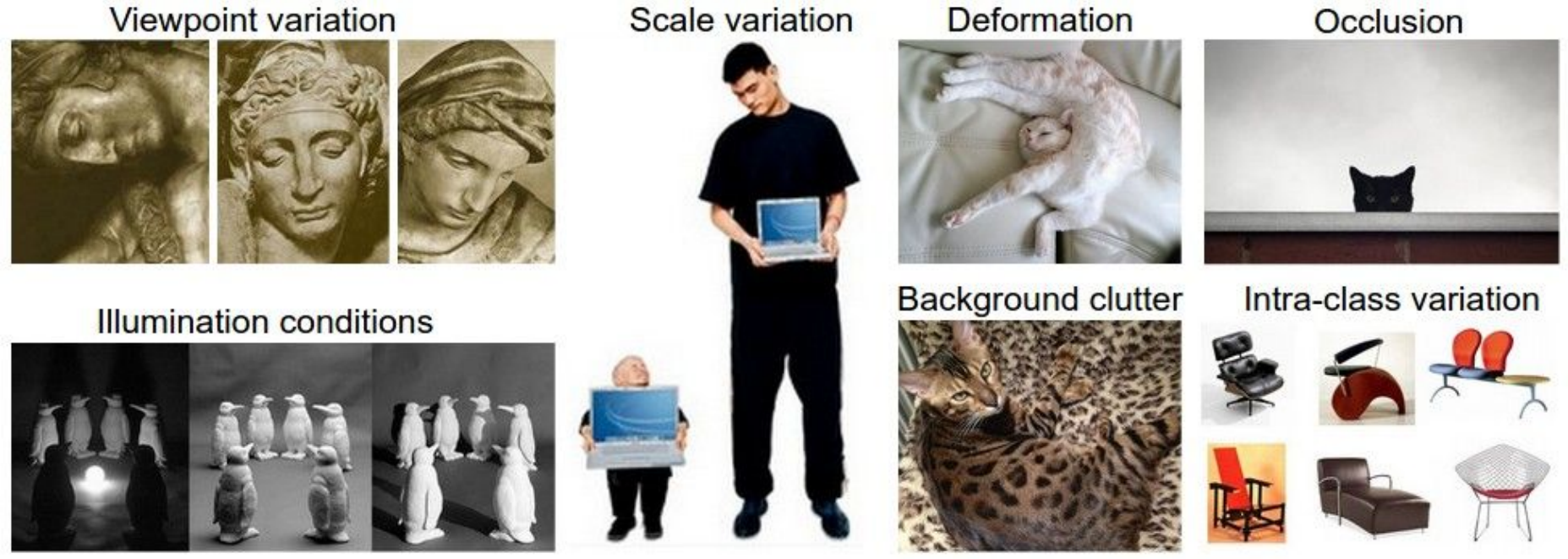
Example



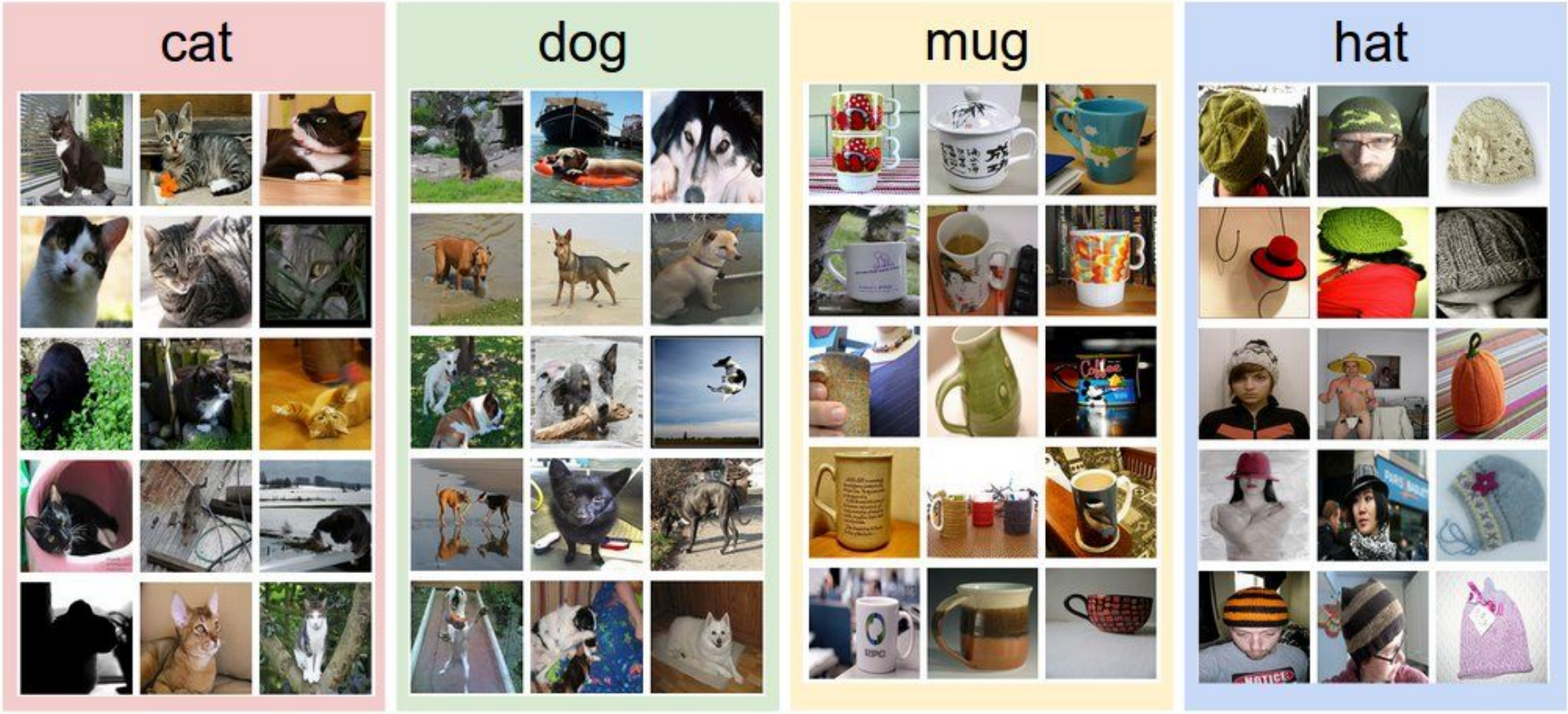
The task in Image Classification is to predict a single label (or a distribution over labels as shown here to indicate our confidence) for a given image. Images are 3-dimensional arrays of integers from 0 to 255, of size Width x Height x 3.

The 3 represents the three color channels Red, Green, Blue.

Challenges


Viewpoint variation
Scale variation
Deformation
Occlusion
Illumination conditions
Background clutter
Intra-class variation

Data-driven approach



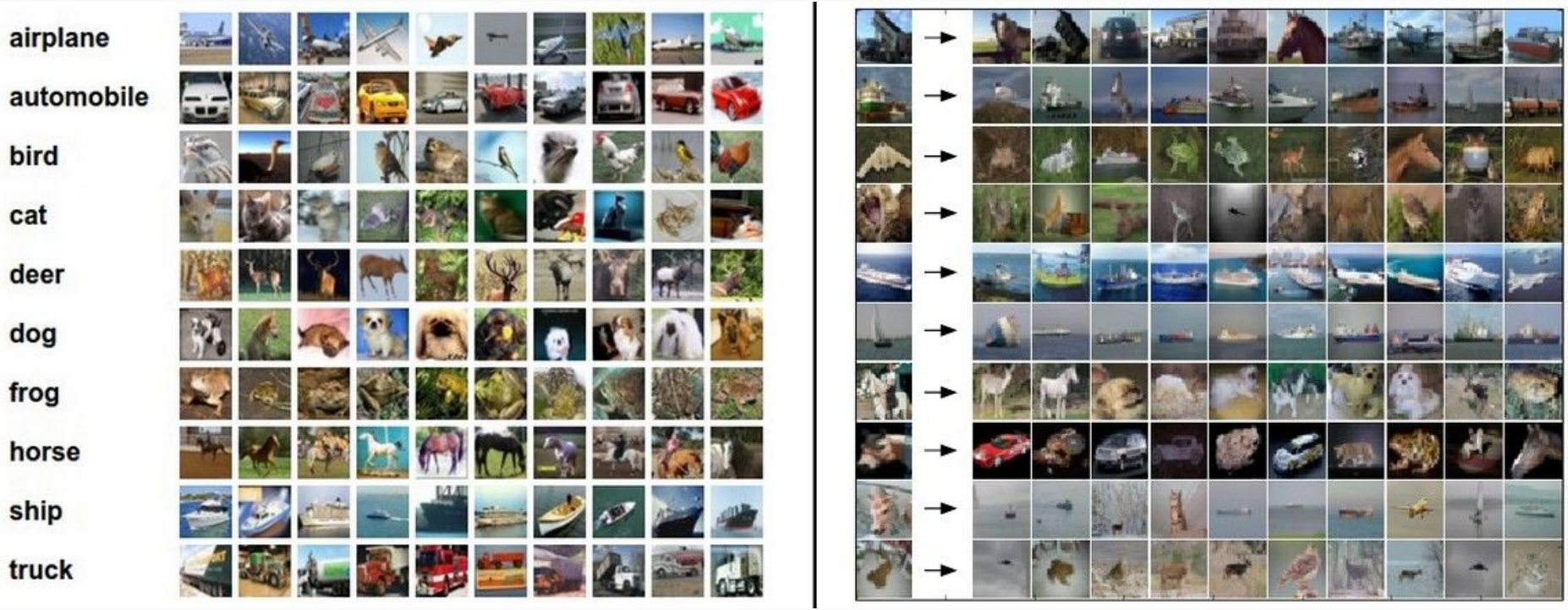
Pipeline

Input: Our input consists of a set of N images, each labeled with one of K different classes. We refer to this data as the training set.

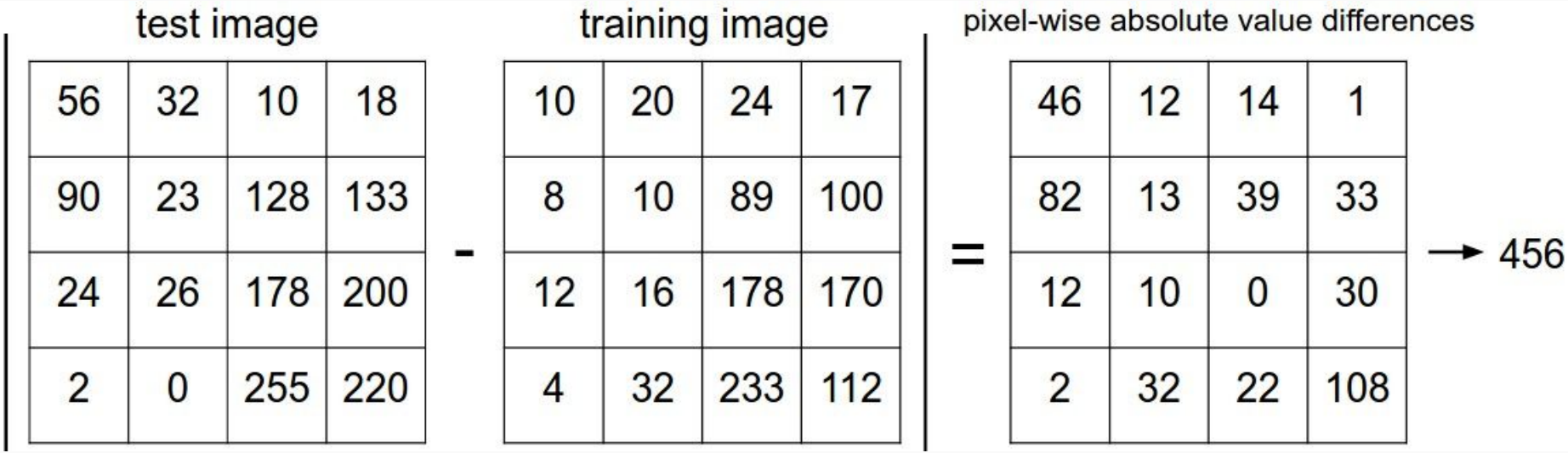
Learning: Our task is to use the training set to learn what every one of the classes looks like. We refer to this step as training a classifier, or learning a model.

Evaluation: In the end, we evaluate the quality of the classifier by asking it to predict labels for a new set of images that it has never seen before. We will then compare the true labels of these images to the ones predicted by the classifier. Intuitively, we're hoping that a lot of the predictions match up with the true answers (which we call the ground truth).

Nearest Neighbor Classifier



Left: Example images from the CIFAR-10 dataset. Right: first column shows a few test images and next to each we show the top 10 nearest neighbors in the training set according to pixel-wise difference.



An example of using pixel-wise differences to compare two images with L1 distance (for one color channel in this example). Two images are subtracted elementwise and then all differences are added up to a single number. If two images are identical the result will be zero. But if the images are very different the result will be large.

Let's also look at how we might implement the classifier in code. First, let's load the CIFAR-10 data into memory as 4 arrays: the training data/labels and the test data/labels. In the code below, `Xtr` (of size 50,000 x 32 x 32 x 3) holds all the images in the training set, and a corresponding 1-dimensional array `Ytr` (of length 50,000) holds the training labels (from 0 to 9):

```
Xtr, Ytr, Xte, Yte = load_CIFAR10('data/cifar10/') # a magic function we provide
# flatten out all images to be one-dimensional
Xtr_rows = Xtr.reshape(Xtr.shape[0], 32 * 32 * 3) # Xtr_rows becomes 50000 x 3072
Xte_rows = Xte.reshape(Xte.shape[0], 32 * 32 * 3) # Xte_rows becomes 10000 x 3072
```

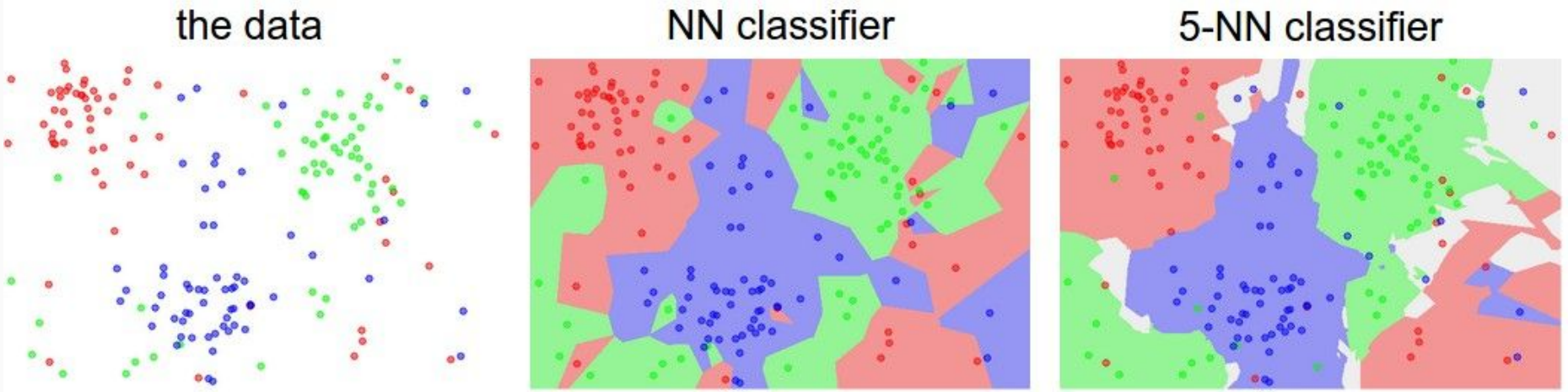
Now that we have all images stretched out as rows, here is how we could train and evaluate a classifier:

```
nn = NearestNeighbor() # create a Nearest Neighbor classifier class
nn.train(Xtr_rows, Ytr) # train the classifier on the training images and labels
Yte_predict = nn.predict(Xte_rows) # predict labels on the test images
# and now print the classification accuracy, which is the average number
# of examples that are correctly predicted (i.e. label matches)
print 'accuracy: %f' % ( np.mean(Yte_predict == Yte) )
```

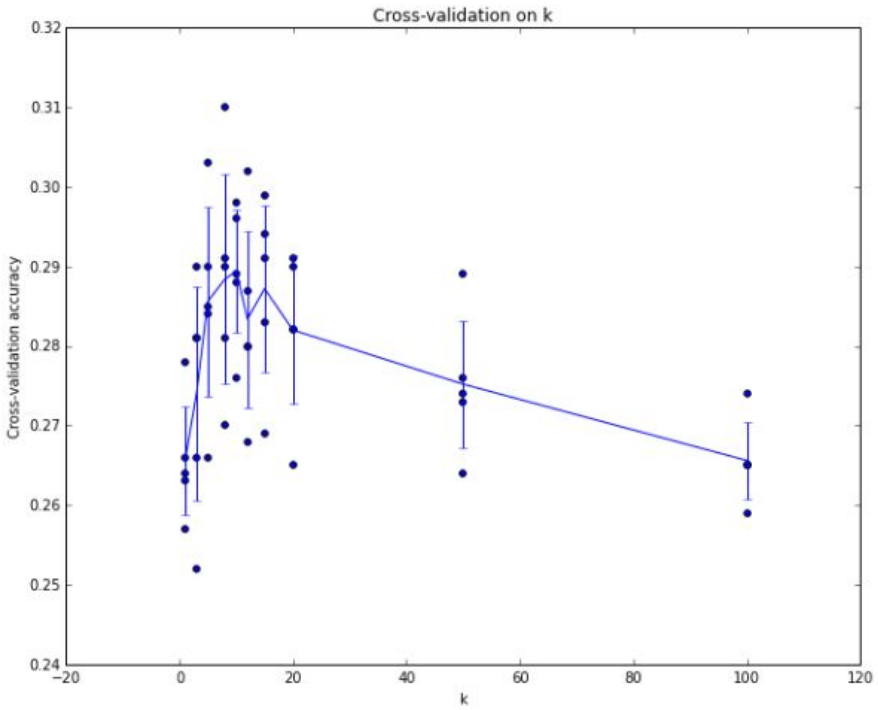
```python
import numpy as np

class NearestNeighbor(object):
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in range(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

If you ran this code, you would see that this classifier only achieves 38.6% on CIFAR-10. That's more impressive than guessing at random (which would give 10% accuracy since there are 10 classes), but nowhere near human performance (which is estimated at about 94%) or near state-of-the-art Convolutional Neural Networks that achieve about 95%, matching human accuracy (see the leaderboard of a recent Kaggle competition on CIFAR-10).

k - Nearest Neighbor Classifier

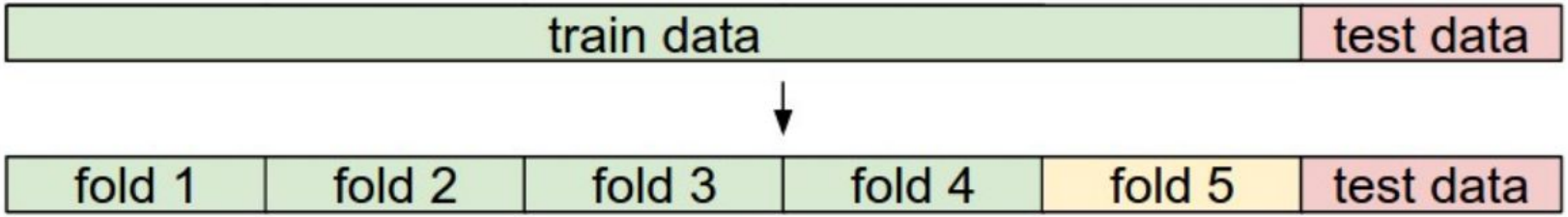


Cross Validation

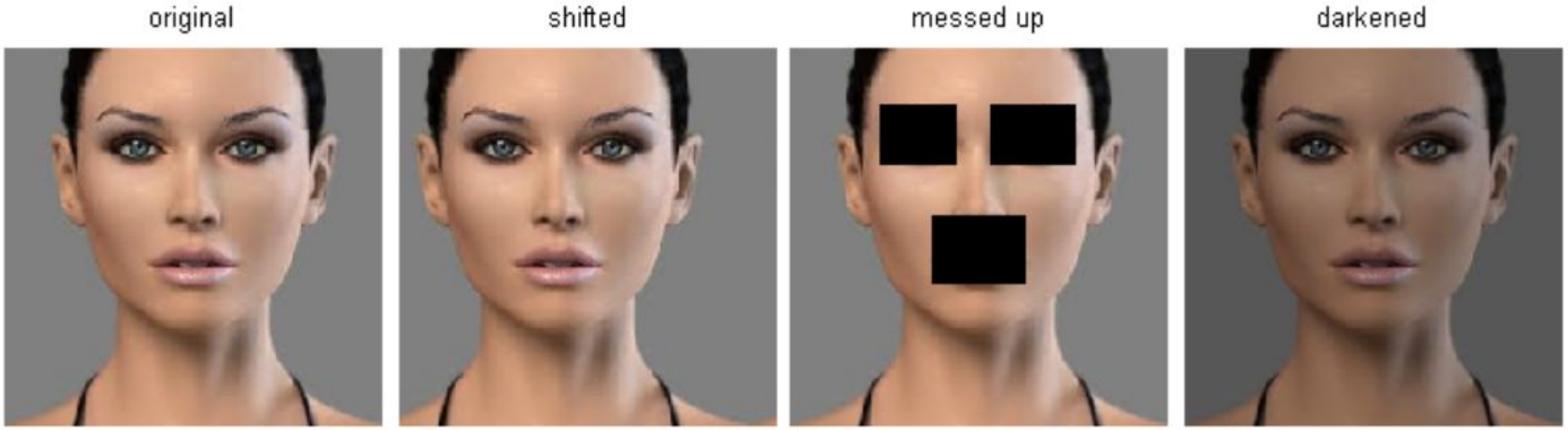


Example of a 5-fold cross-validation run for the parameter **k**. For each value of **k** we train on 4 folds and evaluate on the 5th. Hence, for each **k** we receive 5 accuracies on the validation fold (accuracy is the y-axis, each result is a point). The trend line is drawn through the average of the results for each **k** and the error bars indicate the standard deviation. Note that in this particular case, the cross-validation suggests that a value of about **k** = 7 works best on this particular dataset (corresponding to the peak in the plot). If we used more than 5 folds, we might expect to see a smoother (i.e. less noisy) curve.

In practice. In practice, people prefer to avoid cross-validation in favor of having a single validation split, since cross-validation can be computationally expensive. The splits people tend to use is between 50%-90% of the training data for training and rest for validation. However, this depends on multiple factors: For example if the number of hyperparameters is large you may prefer to use bigger validation splits. If the number of examples in the validation set is small (perhaps only a few hundred or so), it is safer to use cross-validation. Typical number of folds you can see in practice would be 3-fold, 5-fold or 10-fold cross-validation.

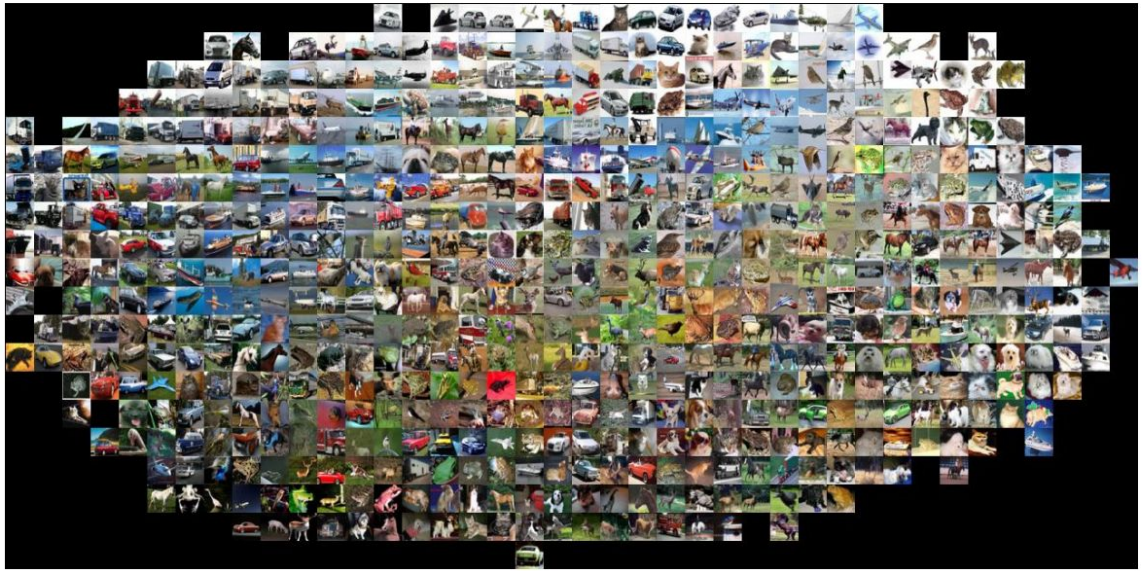


Common data splits. A training and test set is given. The training set is split into folds (for example 5 folds here). The folds 1-4 become the training set. One fold (e.g. fold 5 here in yellow) is denoted as the Validation fold and is used to tune the hyperparameters. Cross-validation goes a step further and iterates over the choice of which fold is the validation fold, separately from 1-5. This would be referred to as 5-fold cross-validation. In the very end once the model is trained and all the best hyperparameters were determined, the model is evaluated a single time on the test data (red).



Pixel-based distances on high-dimensional data (and images especially) can be very unintuitive. An original image (left) and three other images next to it that are all equally far away from it based on L2 pixel distance. Clearly, the pixel-wise distance does not correspond at all to perceptual or semantic similarity.

Here is one more visualization to convince you that using pixel differences to compare images is inadequate. We can use a visualization technique called t-SNE to take the CIFAR-10 images and embed them in two dimensions so that their (local) pairwise distances are best preserved. In this visualization, images that are shown nearby are considered to be very near according to the L2 pixelwise distance we developed above:



CIFAR-10 images embedded in two dimensions with t-SNE. Images that are nearby on this image are considered to be close based on the L2 pixel distance. Notice the strong effect of background rather than semantic class differences. Click here for a bigger version of this visualization.

Teşekkürler!

https://cs231n.github.io/classification