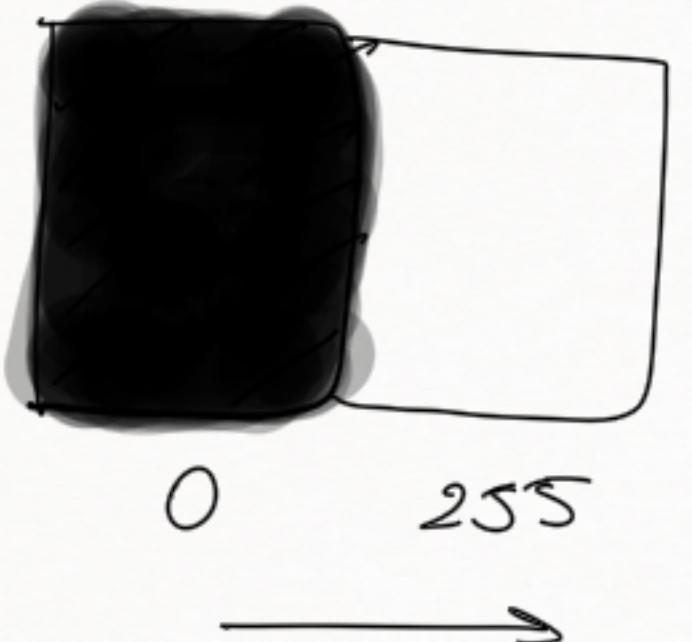


Murat Emre AK

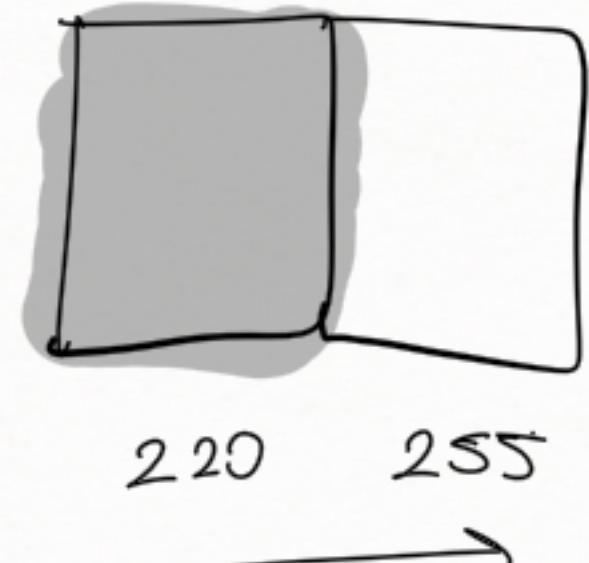
Gradient (değşim)

→ Yolundaki piksellerde, gäre
paralelde değişim



Strong gradient
(Güçlü değişim)

→ Kenar bulunurken bulunur.



Small gradient
(Zayıf değişim)

OPENCV

- Anaconda3
- Conda-forge OpenCV

colismak
için gerekli olur.

Gradient Edge

→ Değişim resmi, kenarları belli
etmek için bulunur.

Strong gradient = beyaz renk } Euren = değişim
Small gradient = siyah renk }



Normal regime



Gradient Edge
(Değişim Regimi)

Greyscale

→ Siyah-Beyaz tabanlı renk.

→ Normalde renkler R-G-B olur. (3 tone)

→ Greyscale'de sadece yapınlık (1 tone)

→ İşlem hızı için.

Blur

→ Bulutluştırma işlemi (keşkinliği düşürme)

→ Her piksel çevresindeki kitle ortalamasına eşit oluyor.

Katsayı		
1	2	1
2	4	2
1	2	1

/16

Canny Kerner Bulma Yöntemi:

- `cv2.Canny(image, low_threshold, high_threshold)`
 - ①
 - ②
- Eğer değişim (gradient) ② den büyükse kerner obrak algılanır.
- ① den küçükse reddedilir. ① - ② arasında ise kernerla karşıyaçsa algılanır. Kesişmeyipse reddedilir.
- Önerilen oran: $\frac{1}{3}$ 'tur. $\frac{1}{2}$ ($-50, 150$)

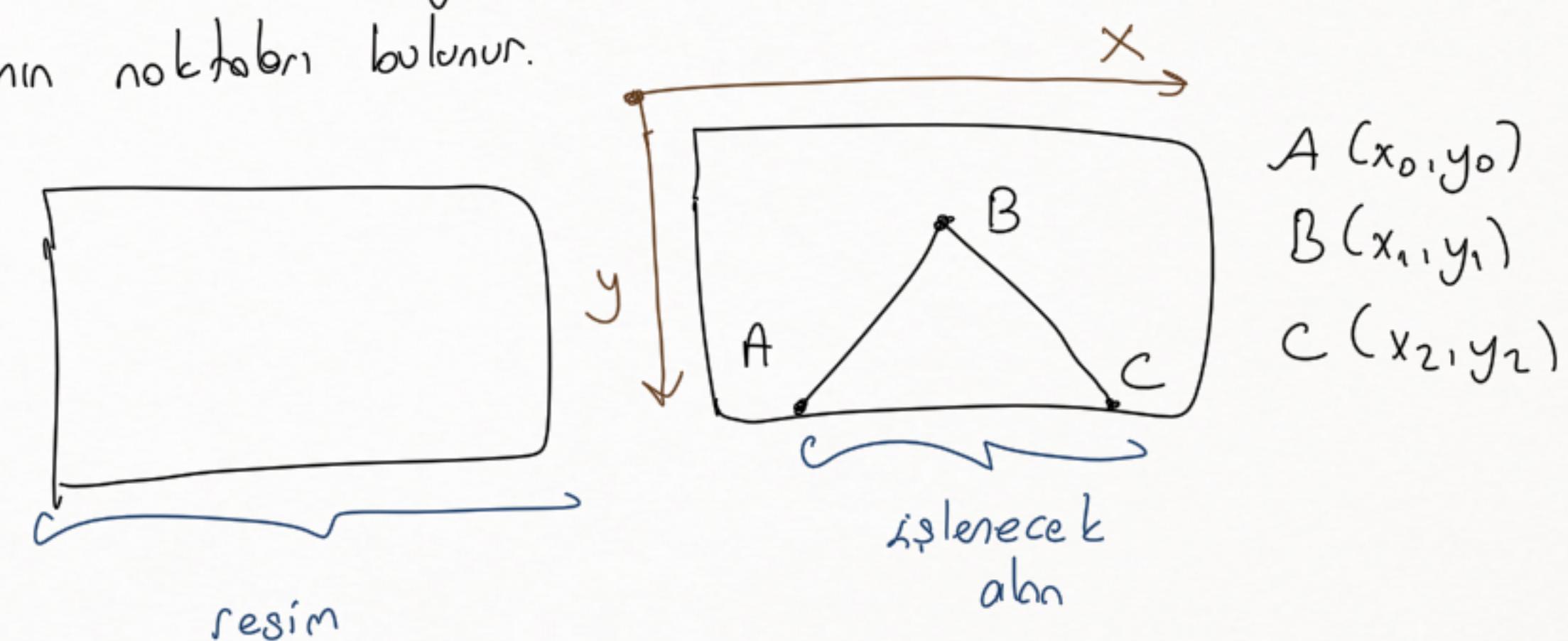
Maskeleme

- Belli bir obrı dışını sıfırlama işlemi
 - Resmin matrisiyle aynı boyutta sıfır dizi oluşturular.
"np.zeros_like(resim)" (kalıp)
 - Belirli bir obrı dısında maskeleme
"cv2.fillPoly(kalıp, [(obr1), (obr2)], int8)"
- 0-255 arası
yögenlik değer
(grı renk)

Belli Bir Abın İşleneceğine .

- "import matplotlib.pyplot" resmi koordinat sistemiyle çizme.

- Koordinat sistemiyle resmi çizdıktan sonra, işlenecek obrın noktaları bulunur.

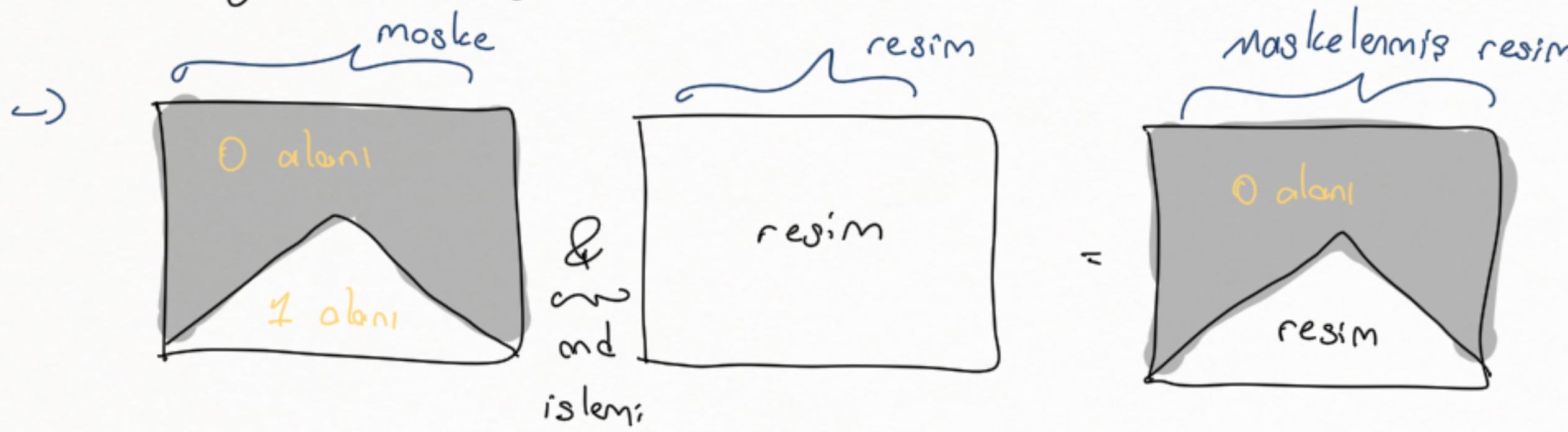


- Bulunan noktaların matrix oluşturular.
"np.array([A, B, C])"

Not: İşlemelerin hepsi orginal resmin kopyasıyla yapılmalıdır.

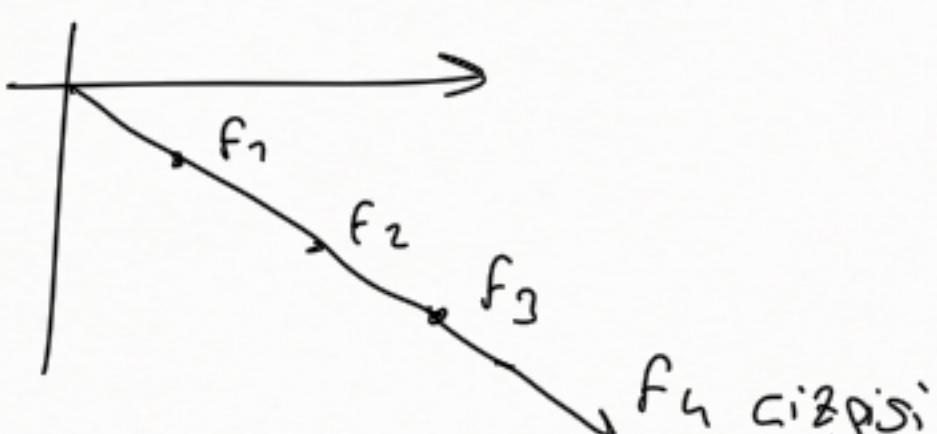
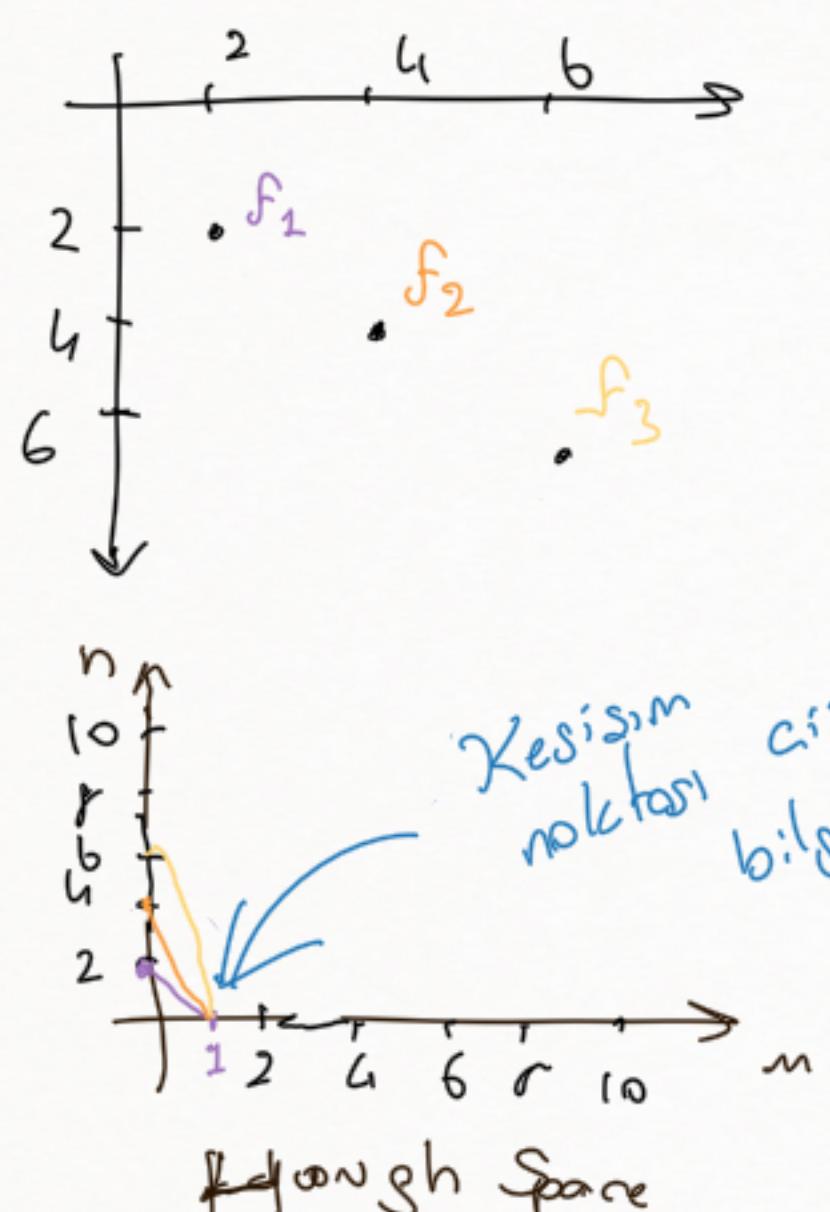
Moskeleme Devamı

→ int8 genelde 255 yazılır. $(11111\dots)_2 = 255$



"np.bitwise_and(resim, mask)"

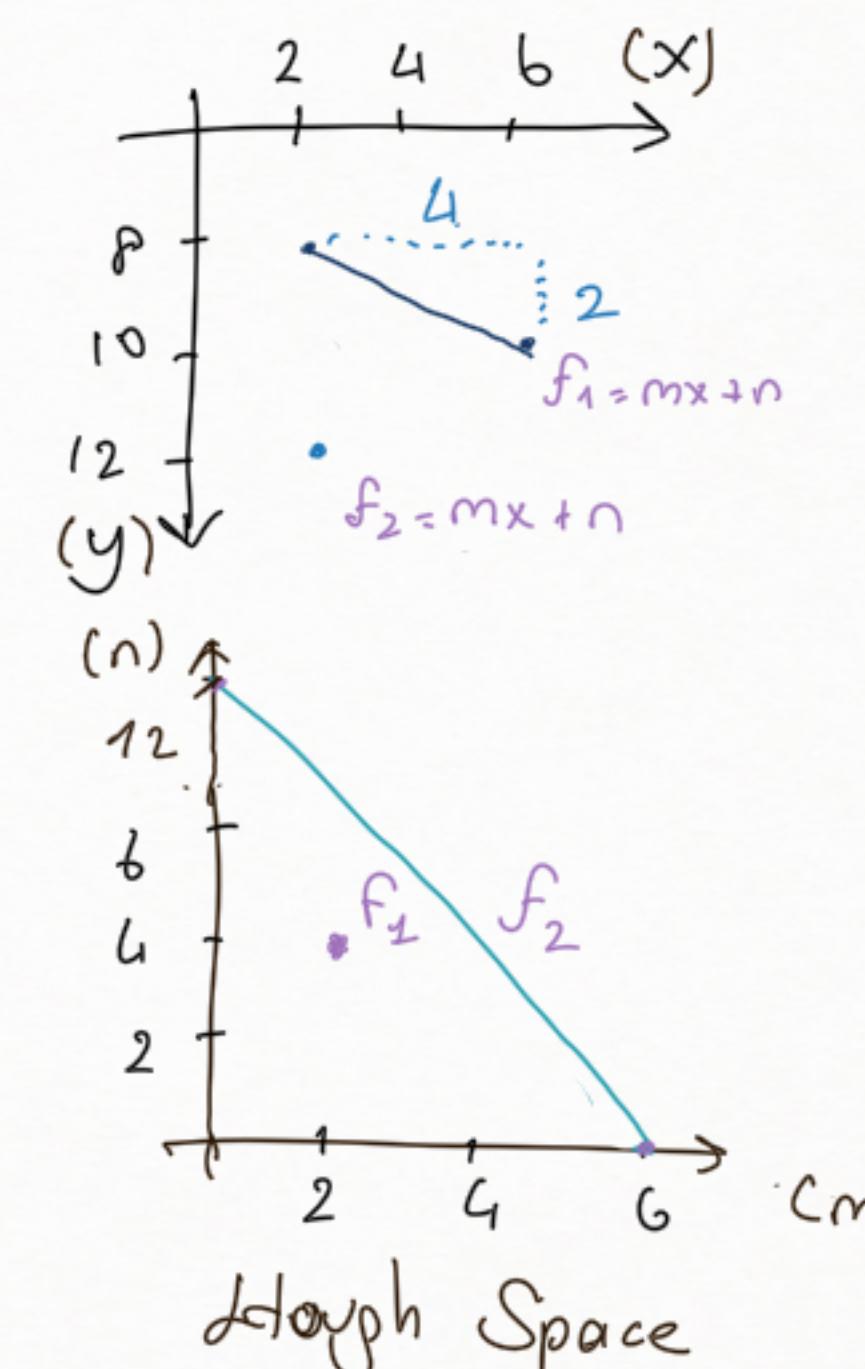
Hough Space ile Noktaları Boğluğun Cizgisi Bulma



$$f_4 = \underset{\text{(Noktaları barınduran doğrular)}}{x} \quad \left(\begin{matrix} m \\ n \end{matrix} \right) = f_4$$

Hough Space

→ Kenarları düzgün bulmak için geliştirilmiş bir yöntem



$$\begin{aligned} & f_1 \\ & m: \text{Doğrunun türü} \\ & n: \text{Eşitliği sağlayon sabit değer} \\ & m=2 \quad (n_1, 2) \quad n=4 \quad (f(2), 8) \end{aligned}$$

Eşitliği sağlayan m,n değerleri:

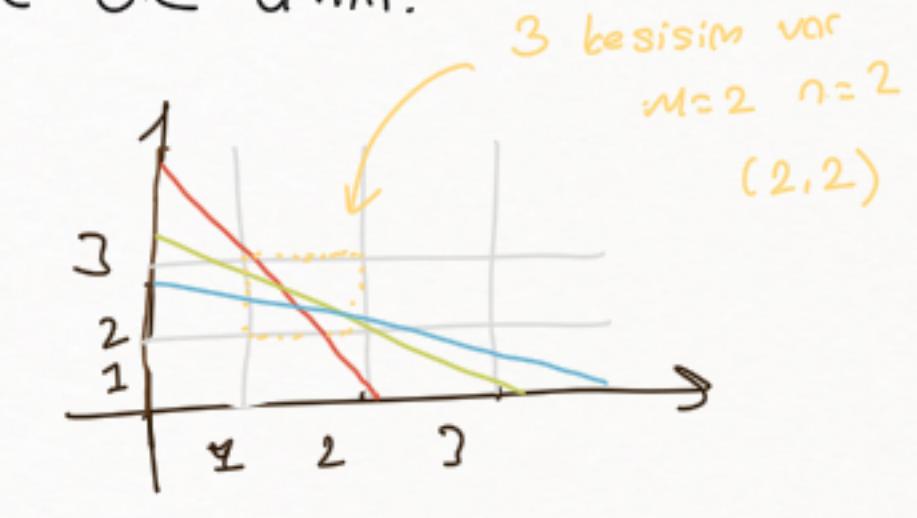
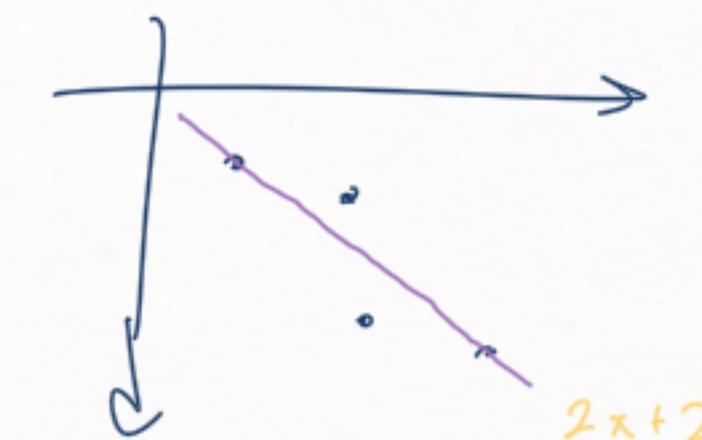
$$\begin{aligned} f_2 &= 6x + 0 & (6, 0) \\ f_2 &= 4x + 2 & (4, 4) \\ f_2 &= 3x + 6 & (3, 6) \\ f_2 &= \dots & \vdots \\ f_2 &= 12 & (0, 12) \end{aligned}$$

Cok Fazla ve Dözeniz Noktaları İçin

→ Hough space karelerde bölünür.

→ Karelerin değeri içinde bulunan kesiminin değerini tutur.

→ En yüksek değere sahip kare ele alınır.

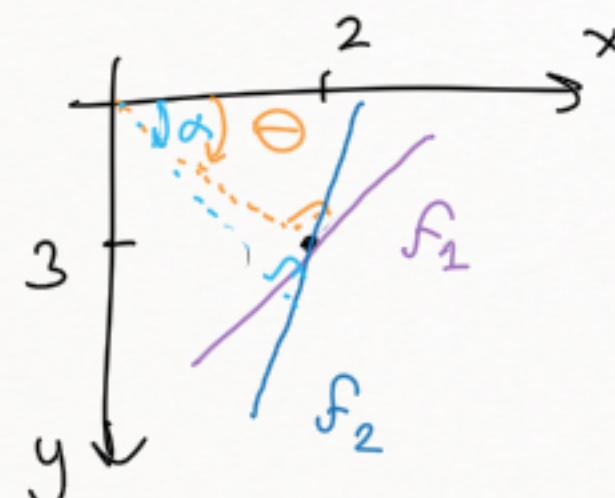


Radyon Tabanlı Hough Space

$$\rightarrow f = mx + n \Rightarrow f = \tan(\theta)x + n \text{ olduğundan}$$

ve $\tan(\alpha)$ her değerde tanımı olmazlarından hata
sonucu engellemek adına radyon tabanlı sistem
kullanılır.

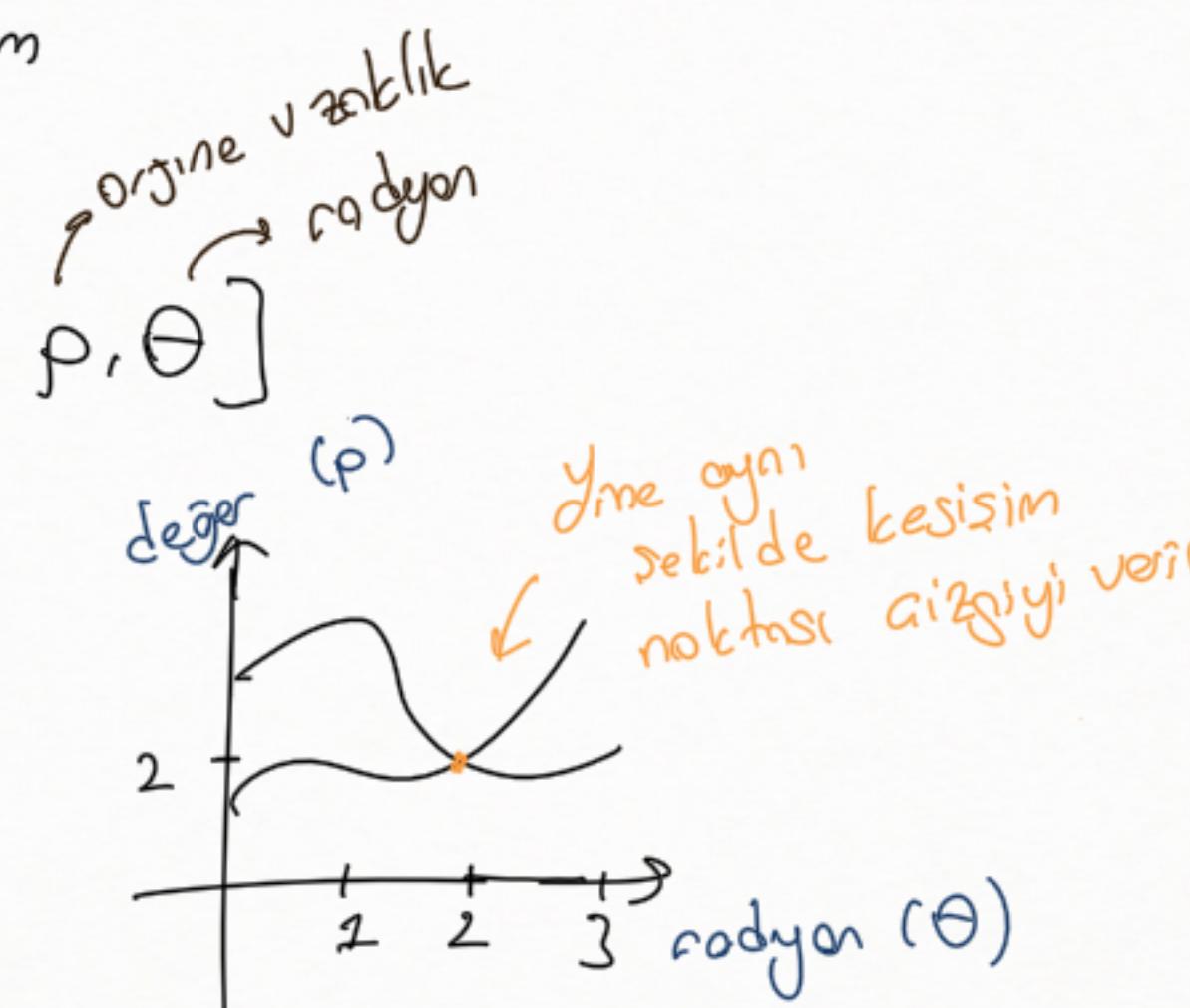
$$\rightarrow \rho = x \cos(\theta) + y \sin(\theta) \quad [m, n \text{ yerine } \rho, \theta]$$



$$f_1 = 2 \cdot \cos(\theta) + 3 \cdot \sin(\theta)$$

$$f_2 = 2 \cdot \cos(\alpha) + 3 \cdot \sin(\alpha)$$

;



Radyon Tabanlı Hough Space Çizgisi

$$\rightarrow \rho = 2 \quad \theta = 2$$

ρ : Orjine olan uzaklık

θ : X eksen ile arasındaki açı (saat yönü)



Kodlu Kullanımı

\rightarrow "cv2. houghLine P (①, ②, ③, ④, ⑤, ⑥)"

\rightarrow Çizgi verilerini 2D dizi olarak döndürür.

\rightarrow Çizgiler düzensiz ve aralıklı gelebilir. Bu yüzden çizgi düzeltme teknigi uygulanır.

① Çizgilerin olduğu resim

② "2"

③ "np. pi / 180"

④ Çizgi kabul edilmesi için en az kaç noktası olsun (100 mirel)

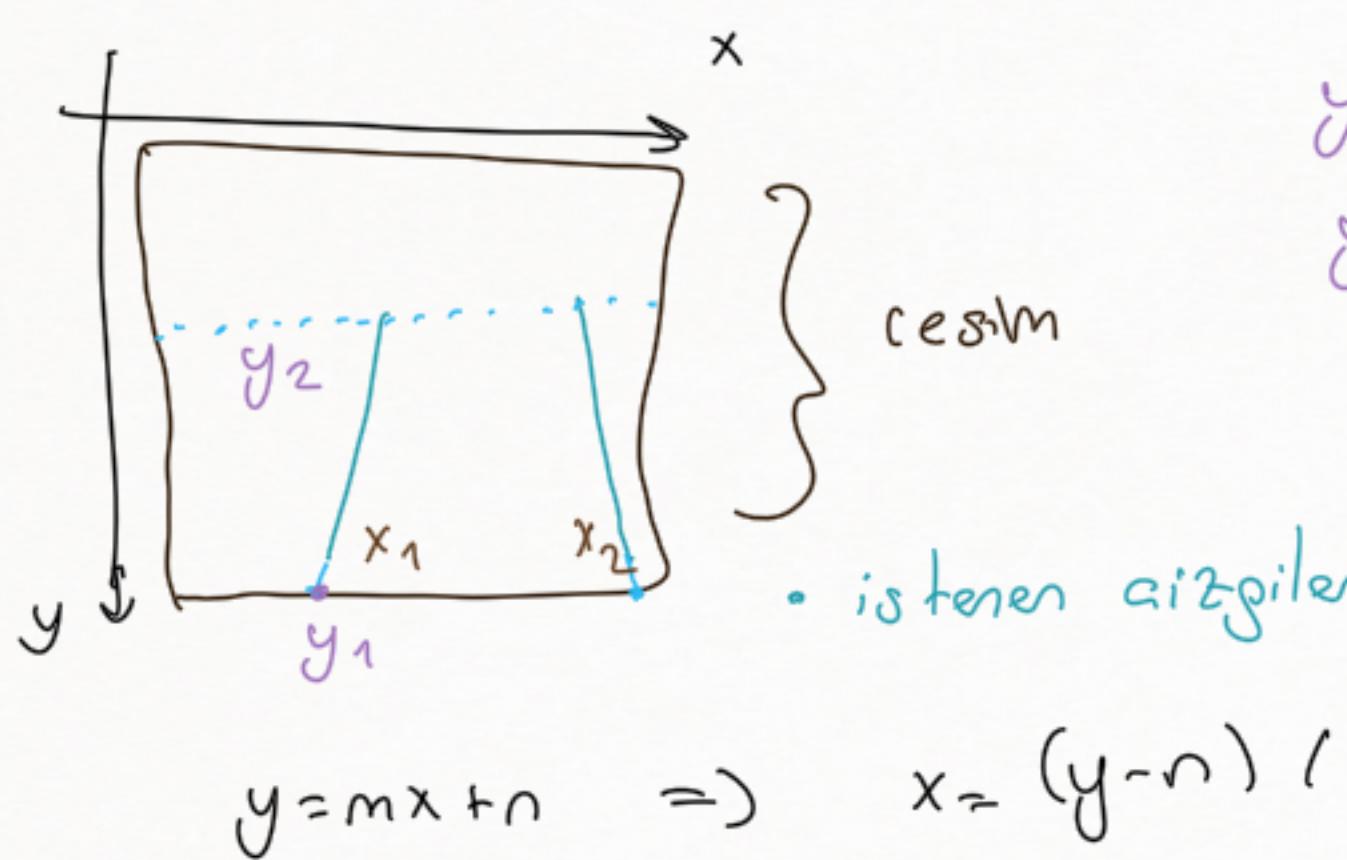
⑤ "np. array []"

⑥ "minLineLength = 40" 40, en kısa çizgi uzunluğu

⑦ "maxLineGap = 5" iki çizgi arasındaki başlangıç 5 ise 0 çizgileri birleştirir.

Gizgir Düzeltme Tekniği

- "cv2.HoughLineP(...)" ile oluşturulan çizgilerden başlangıç ve bitiş noktaları alınır. $(x_1, y_1), (x_2, y_2)$
- Alınan noktalarla m ve n degeri hesaplanır. $(mx + n)$
- Tüm değerlerin ortalaması alınır. (m, n)
- Ortalama m ve n değerleri, resmin uzunluğun ve yükseliğiyile, düzeltilmis (ortalaması alınmış) çizginin başlangıç ve bitiş noktaları bulunur.



y_1 = resim yükseliği

y_2 = y_1 . oran

$$x_1 = (y_1 - n) / m$$

$$x_2 = (y_2 - n) / m$$

- Sol veya sağ çizgi olduğunu eğim (m) ile karar veririz
 m negatifse sol; pozitifse sağ çizgidir.

Kod Karşılaştırması

- for çizgi in çizgiler
 - $x_1, y_1, x_2, y_2 = çizgi$ açı degeri
- "cv2.polyfit((x1, y1), (x2, y2), 1)" kodu
[m, n] dizisini döndürür. veriler [0] = m veriler [1] = n
- "np.average(çizgiler, axis=0)" kodu ortalamayı döndürür.
- "return np.array([sol-çizgi, sağ-çizgi])" ile koordinatları hesaplanan çizgiler döndürülür.

Cizginin koordinatları, Hesaplama

- resim ve (m, n) değerleri gerekli dir.
- "resim.shape[0]" ile resmin tabanı bulunur.
- Soldaki islemek yapıılır.
- "np.array([(x1, y1, x2, y2)])" ile koordinatlar döndürülür.

Video İşleme Tekniği

- Videolar resim topluluğu olduğu için resim işleme teknikleri kullanılır.
- "cv2.VideoCapture(video_yolu)" ile video yokalanır.
- Yokalanan video "isOpened()" olduğu surece işlenir. " -, video_karesi = cap.read()" ile fotoğraf alınır.
yokalanan video'nun değişken adı.

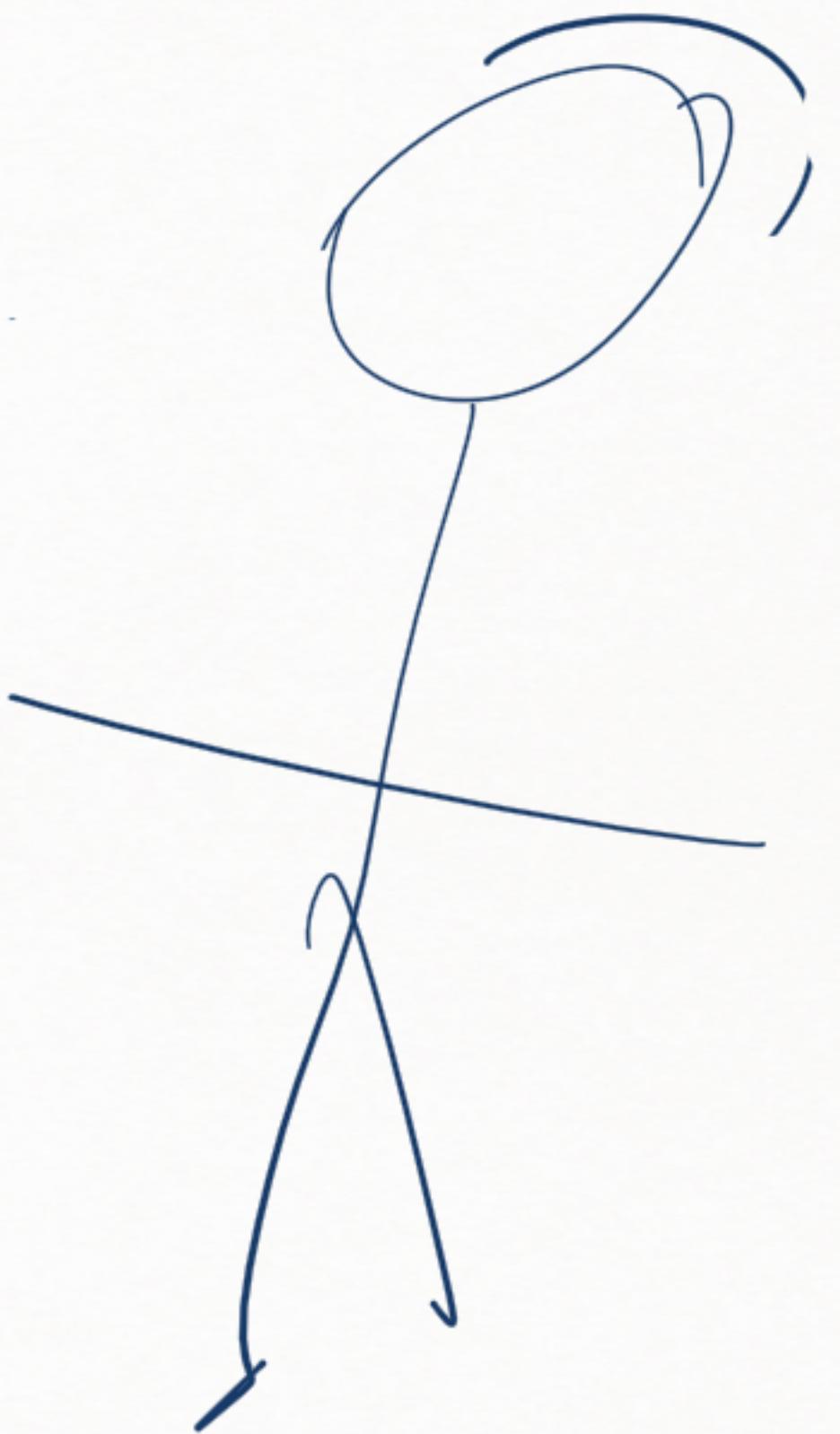
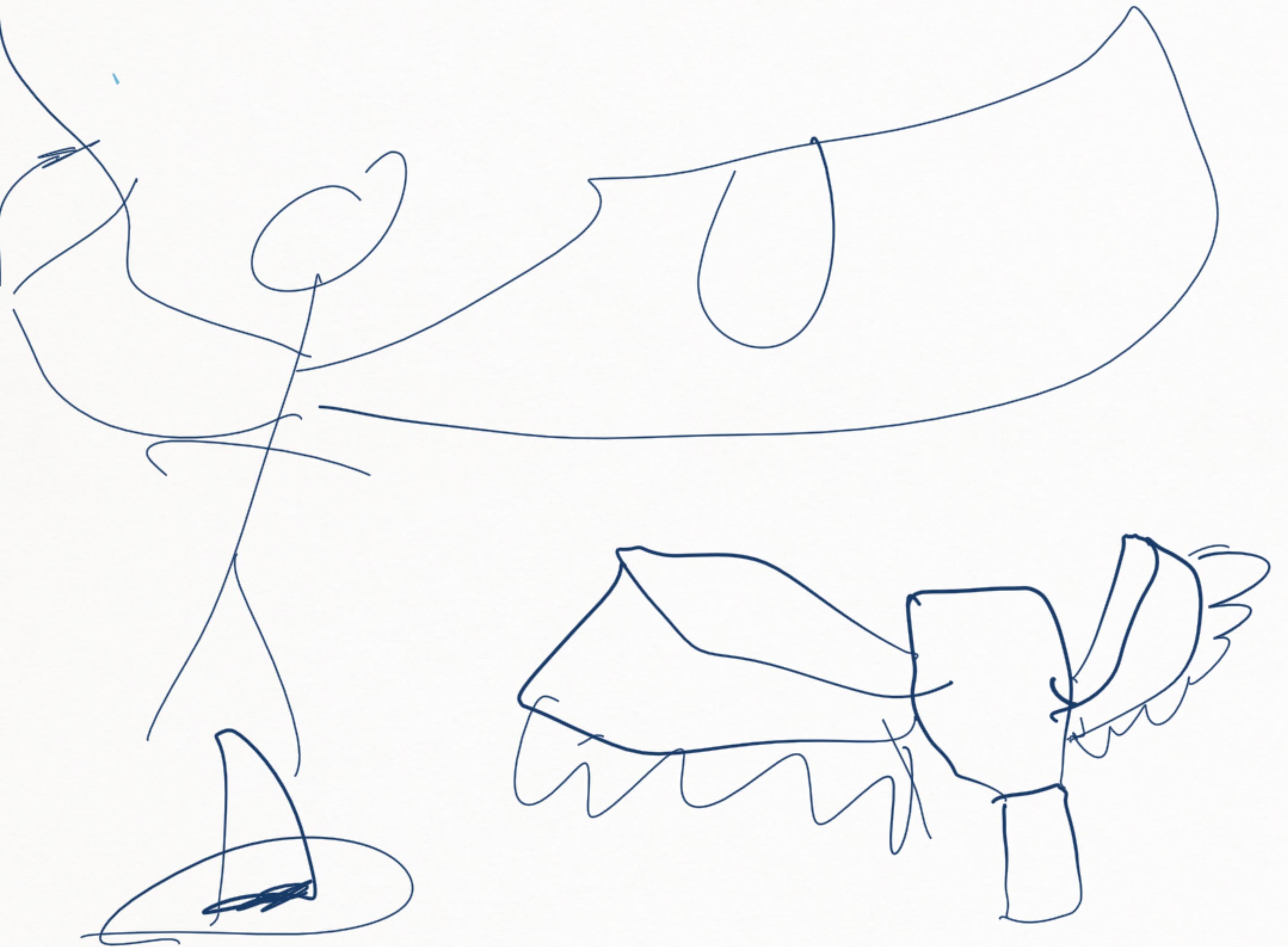
Video İşleme Sonlandırma

→ "cv2.waitKey(1) & 0xFF == ord('q')" ile q'ya basıldığında video işleme sonlandırılır.

→ "release()" ile serbest bırakılır.

→ "cv2.destroyAllWindows()" ile tüm ekranlar sonlandırılır.

Bilds VS filas



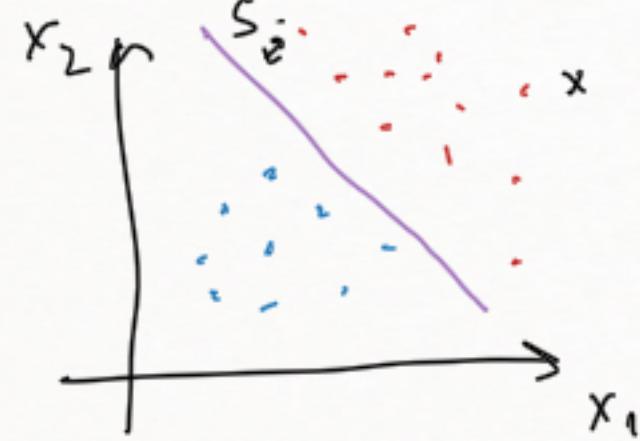
Machine Learning

Supervised Learning

- Örnek giriş ve çıkış verileri programa veriliyor.
- Ön bilgi ile program, genel kuralları aktarıyor.

Lineer Regression
Classification

Classification



negatif veya
pozitif olabilir
(skor)
 s : Girişye göre puanı
 x : girdiler
 y : çıktı { 1 0 }

Giriş: Hesabı

$$S = 0$$

$[x_1, x_2]$: giriş

$$x_1 \left\{ \begin{array}{l} (x_{\max}, y_{\min}) \end{array} \right.$$

w_1, w_2, b { **İçeri:**

$$\begin{aligned} S &= 0 \\ 0 &= w_1 \cdot x_1 + w_2 \cdot x_2 + b \\ x_2 \cdot w_2 &= -b - x_1 \cdot w_1 \\ &\frac{-b - x_1 \cdot w_1}{w_2} \end{aligned}$$

Ağzıye Göre Puanı Hesaplama

$$S = w_1 \cdot x_1 + w_2 \cdot x_2 + b$$

- w_i : ağırlıklar
- b : ortam sabiti
- Negatif ve pozitif değerler
göre çıktı değiş.

$$x = [x_1, x_2, 1]^T$$

$$w = \begin{pmatrix} w_1 \\ w_2 \\ b \end{pmatrix}$$

$$S = x \cdot w$$

Matriks çarpımı

Cross Entropy Formülü

$$E(w, x, y) = -\sum_{i=1}^n (y_i \ln(S(x \cdot w)) + (1-y_i) \ln(1-S(x \cdot w)))$$

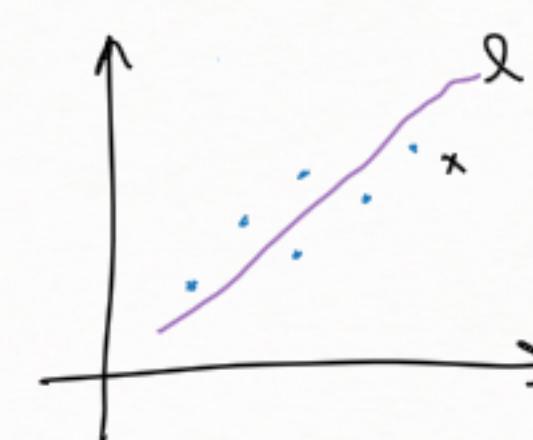
- E : Cross Entropy, lata to puanı

Ortalama Hata

$$E_0 = \frac{E(w, x, y)}{n}$$

- n : noktası sayısı
- Toplam y sayısı $= m$

Lineer Regression



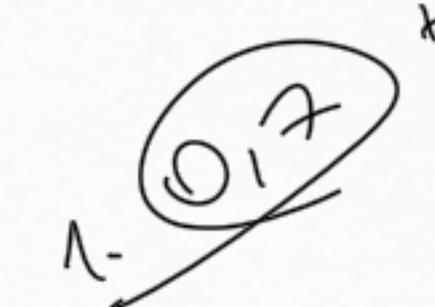
l : çizgi
 x : girdi

y : çıktı
cizgiyi çizdir.

- Anlaşılabilirdir

- x 'lere en uygun optimum

$0,3$
 $0,17$



Sigmoid Fonksiyonu

$$S(s) = \frac{1}{1+e^{-s}}$$

- S : obsalık değeri döndürür ($0 < < 1$)
 - $S(s) = 0,80 \Rightarrow \{ \cdot 1,80, \cdot \% 20 \}$
 - $\lim_{s \rightarrow -\infty} S(s) = 0$
 - $\lim_{s \rightarrow \infty} S(s) = 1$
- Lacivert

$$S = x \cdot w$$

Matriks çarpımı

$$w = \begin{pmatrix} w_1 \\ w_2 \\ b \end{pmatrix}$$

Not: Bütün veriler matriks
formunda olmalı

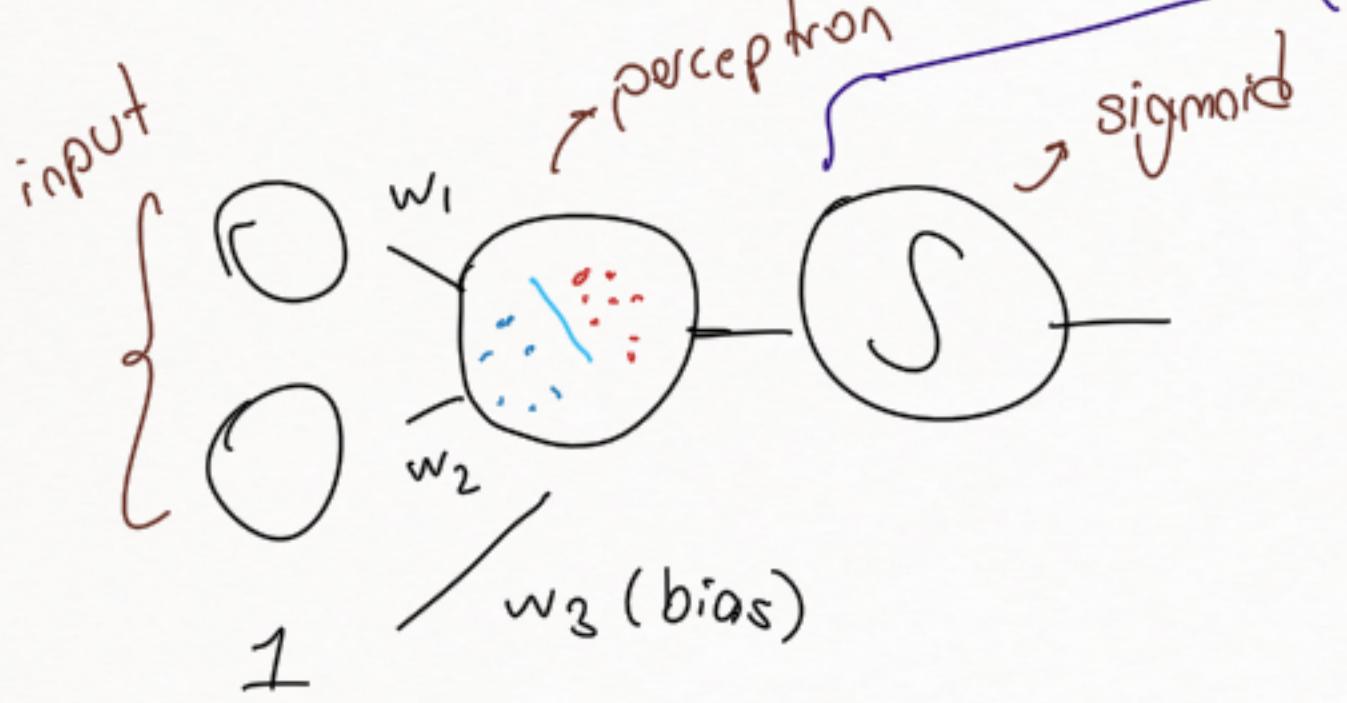
Gradient Descent

$$\nabla E = \frac{x \cdot (S(x \cdot w) - y)}{m} \cdot \alpha$$

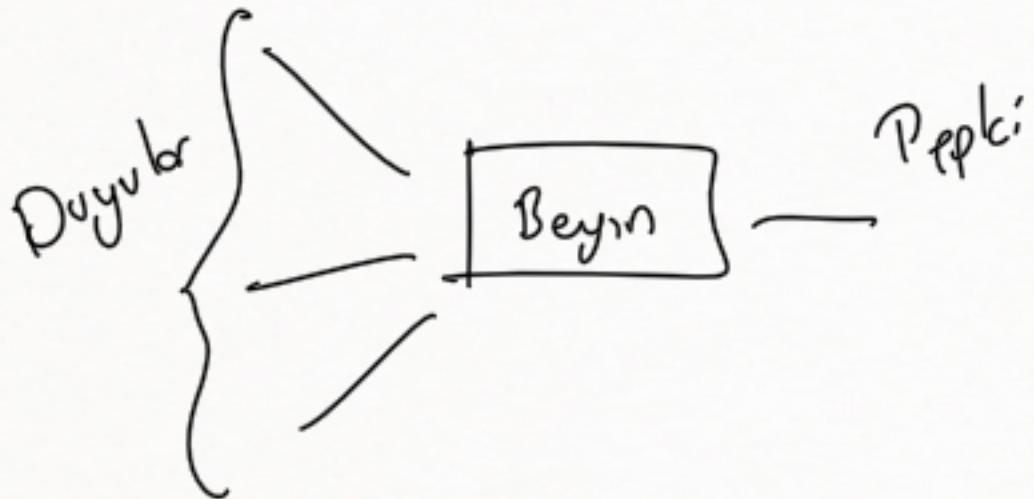
∇E : Degisim
 x: girildiler (input, points)
 P: Sigmoid
 y: Çıktı (output, label)
 m: Nokta sayısı
 w: Ağırlıklar
 α : Adım kat sayısı (öncek 0,01)

* playground tensorflow

Neural Network



Basitçe Classification yapısı.



ciğiden kemi

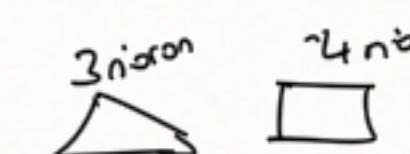
$$w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot 1$$

- Problem type: Classification
- Aktivasyon: Sigmoid ($\frac{1}{1+e^{-s}}$)
- Her bir nöron çıkışını temsil eder.
- Epoch: Devir sayısı

Degisimin Uygulanması

$$w = w - \nabla E$$

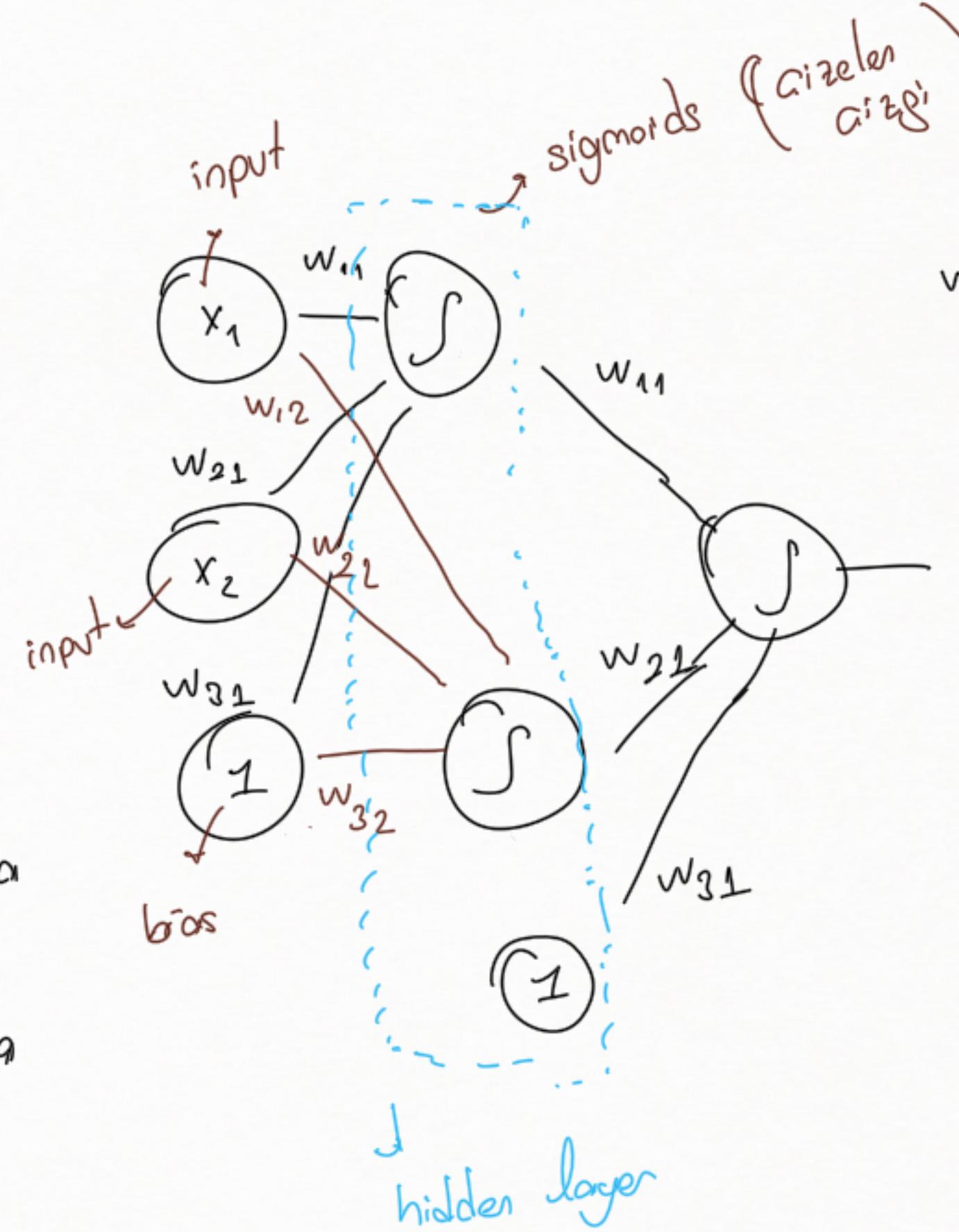
- Yeni w değerlerine göre x değerleri hesaplanır.



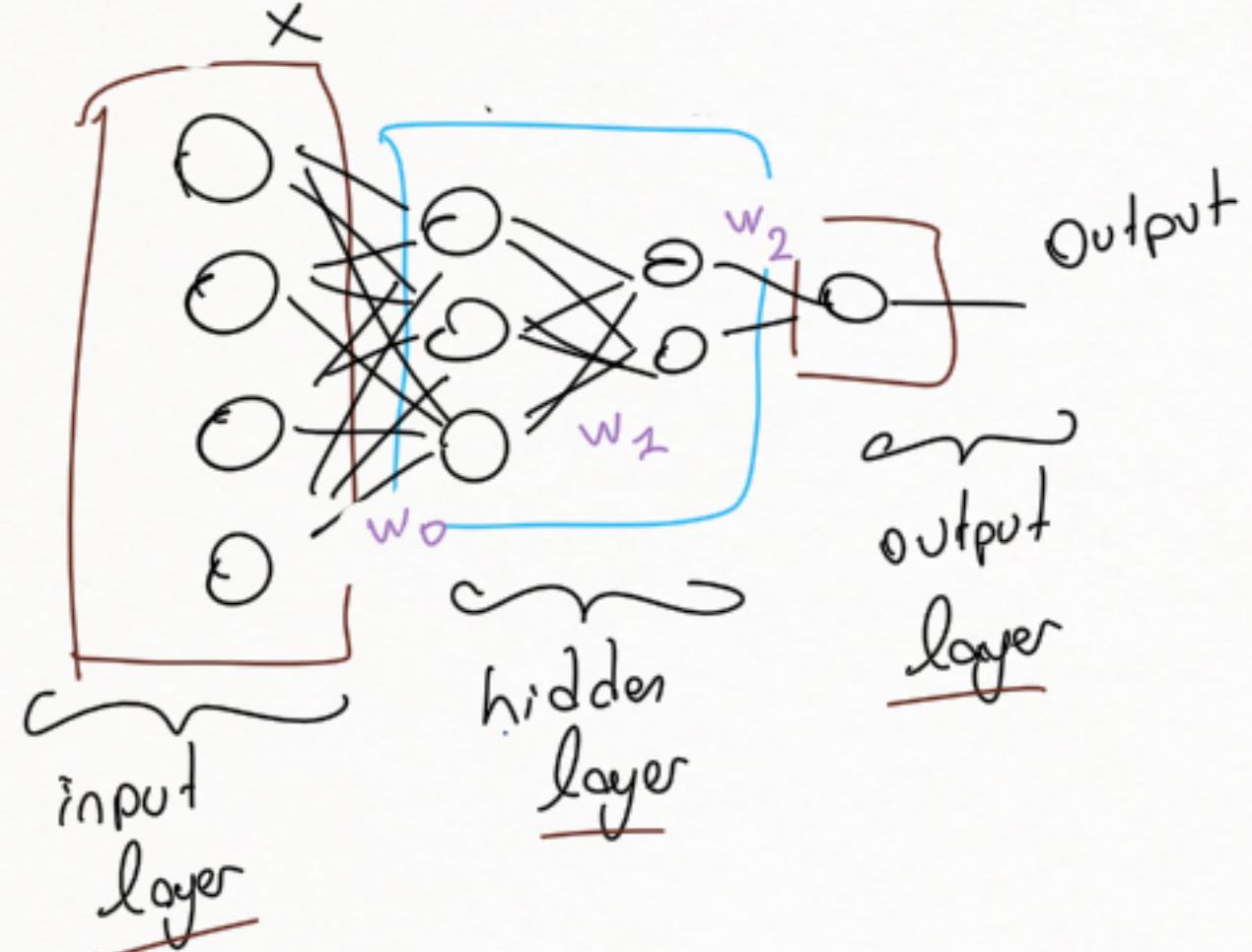
Karmaşık

hidden layer 1

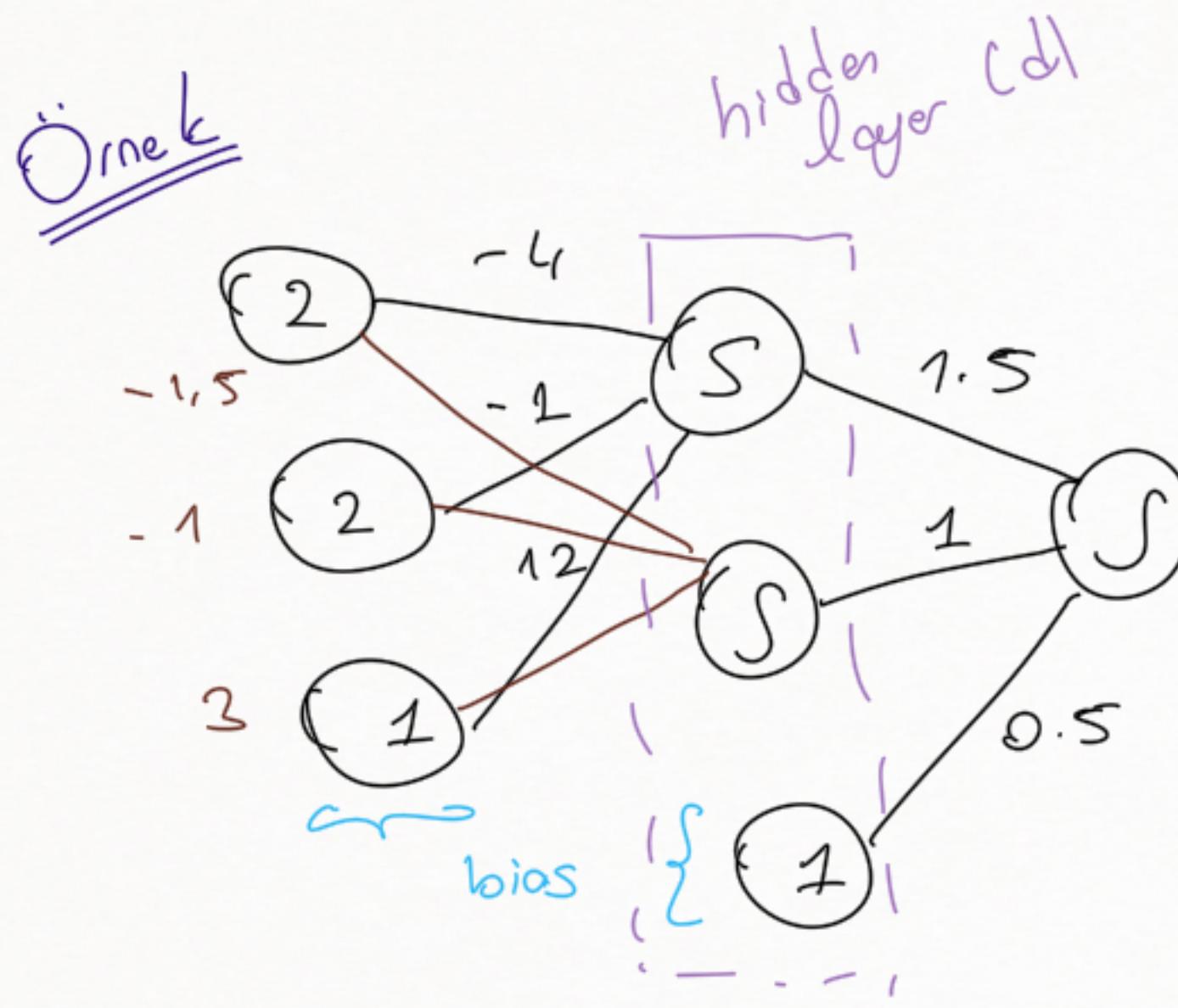
- Geldiği adresi benzerdir.
- w_{11} fazla olursa
 1. çıkışa benzer.
 w_{21} fazla olursa
 2. çıkışa benzer



Neural Network Devamı



• Depth (Dörrlik) = Hidden Layer sayısi (2)



input saysisi kadar
 $x = [x_1, x_2, \dots]$

$w = \begin{bmatrix} w_{11} & w_{21} & \dots \\ w_{12} & w_{22} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$

Sonraki Sigmoid saysisi kadar
 Sonraki Sigmoid
 saysisi kadar
 $s = [s_1, s_2, s_3, \dots]$

$S(s) = [p_{11}, p_{12}, p_{13}, \dots]$

Sonraki Sigmoid
 saysisi

Gözüm

$d=1$ $F = S(s_1)$

$x = [2, 2, 1]$

$w_0 = \begin{bmatrix} -4 & -1.5 \\ -1 & -1 \\ 1.5 & 1 \end{bmatrix}$

$w_1 = \begin{bmatrix} 1.5 \\ 1 \\ 0.5 \end{bmatrix}$

$s_0 = x \cdot w_0$

$s_n = S(s_{n-1}) \cdot w_n$

$F = S(s_d)$

- $n \leq d$
- d : dörrlik

Örnek

$F = S(s_2)$

$s_2 = S(s_1) \cdot w_2$

$s_1 = S(s_0) \cdot w_1$

$s_0 = x \cdot w_0$

input $\times 1$
 Hidden $\times 2$
 Output $\times 1$

$d+1$ tone

$F = S(S(S(x \cdot w_0) \cdot w_1) \cdot w_2)$

$s_1 = S(s_0) \cdot w_1$

$s_0 = x \cdot w_0$

$S(s_1) = [0.92]$

$F = 0.92$

$s_0 = [2, 2, 1] \cdot w_0 = [2, 0.6]$

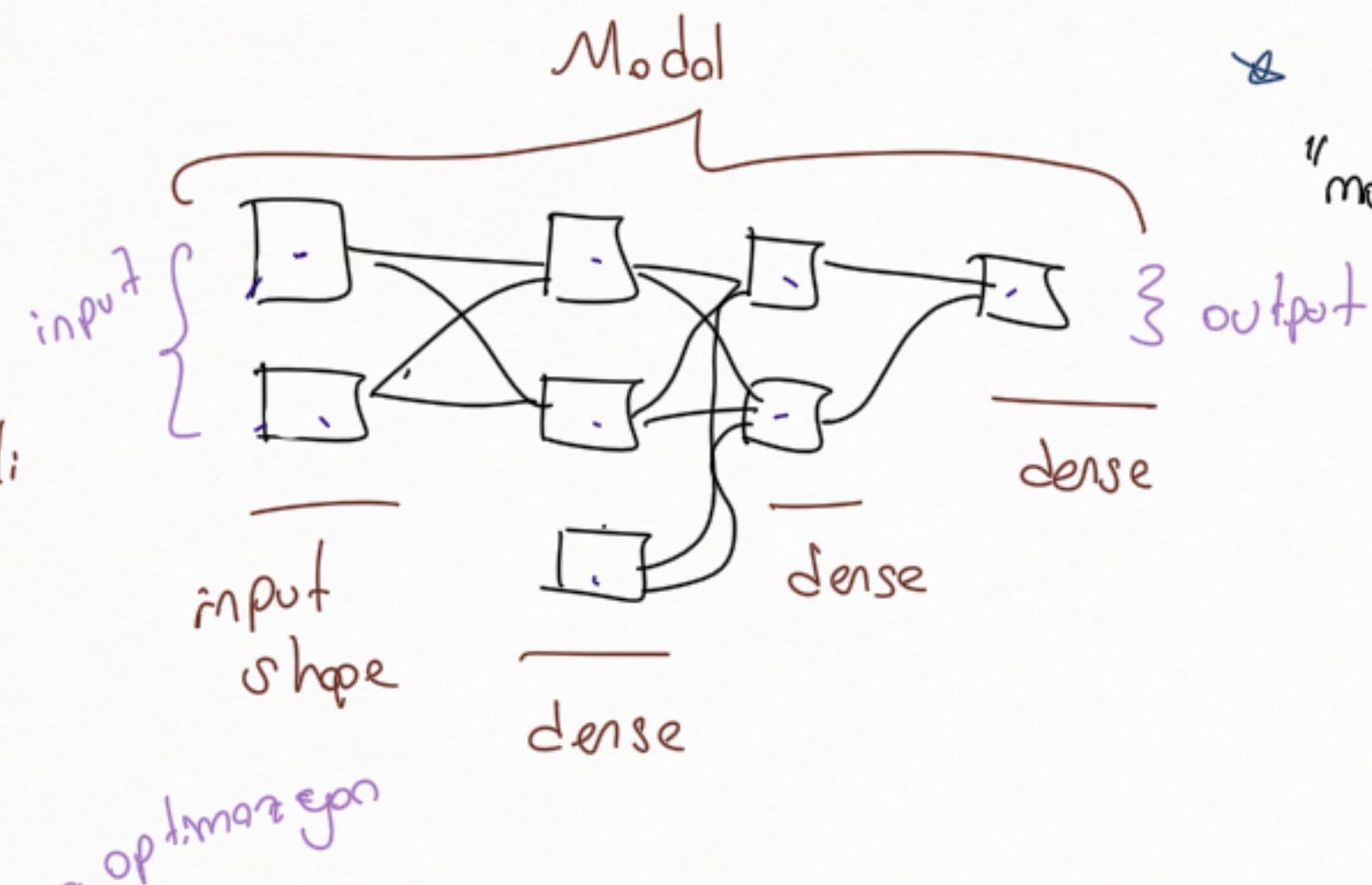
$S(s_0) = \frac{1}{1 + e^{-s_0}} = [0.88, 0.66]$

{ 0.92 degruvut }

$s_1 = [0.88, 0.66] \cdot w_1 = [2.46]$

Code 'do' hâlbârm

- Keras
- TensorFlow } gerekli



» "model.compile (Adam (lr = ①, ②, metrics = ③))"

① Learning rate { 0.001, 0.01, 0.3, 1, 3, 10 ... }

② Hata fonksiyonu { 'binary_crossentropy', 'categorical_crossentropy' ... }

③ 'accuracy'

* "model.fit (x = ④, y = ⑤, verbose = ③, batch_size = ④, epochs = ⑤, shuffle = ⑥,)"

① Input matrix

② Output (label) matrix

③ Ekrana çıktı yazılınır m $0,1$

④ 20 (?)

⑤ Devir sayısı

⑥ 'true' (?)

"model.add (Dense (① , input_shape = ②))"
activation = ③

① Katmandaki nöron sayısı

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \hline 2 & 3 & 2 & 1 \\ - & - & - & - \end{array}$$

② Sadece ilk katman için
grdi dizisi.

③ Aktivasyon fonksiyonu
(S = sigmoid)

$$S \quad S \quad S \quad S$$

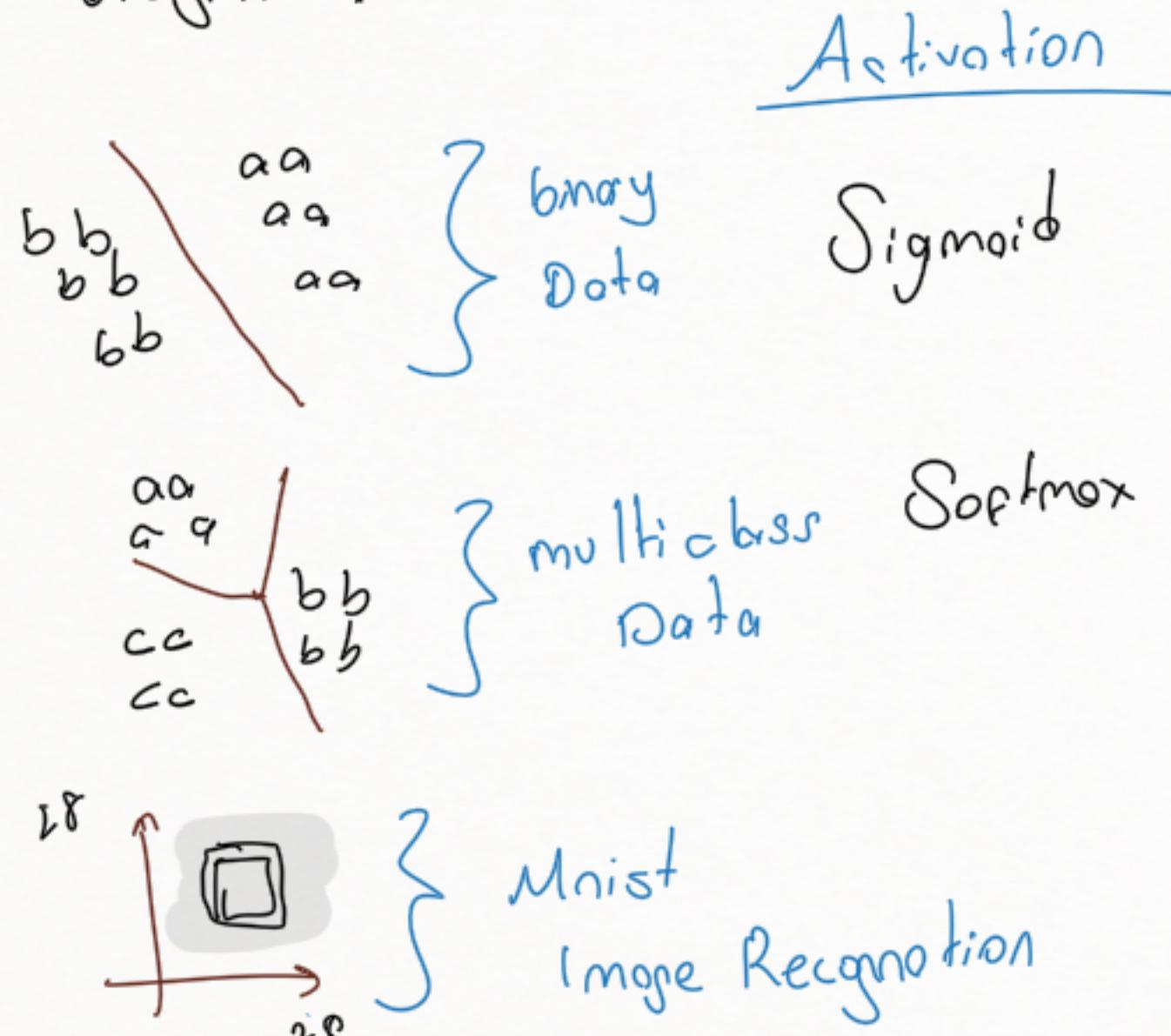
Deep Neural Network başlıca teknikler;

• Gradient Descent

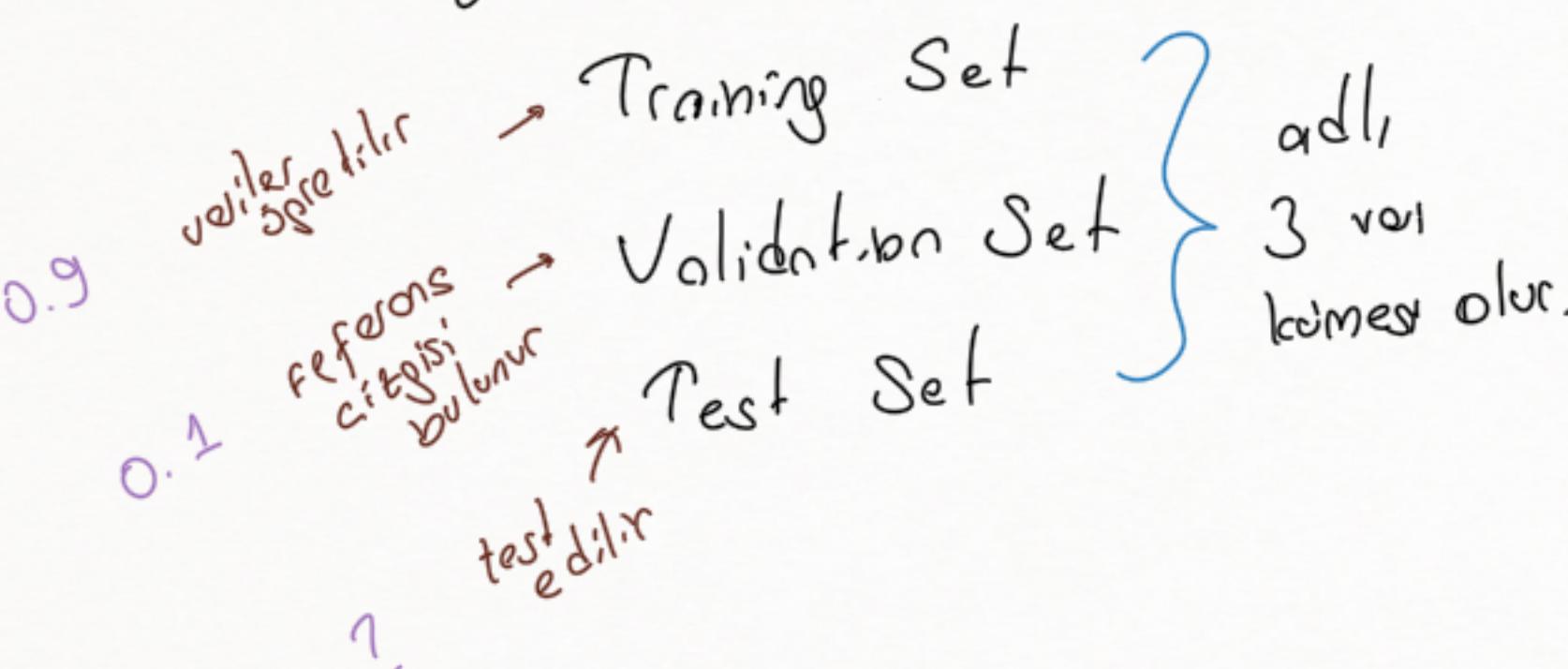
* Back propagation (Katman yarısı)

Multiclass Classification

- 2'den fazla sepetek olduğunda.



- Büyük veri kümelerinde



Binary / Multi Class

Binary Data

$$S(x) = \frac{1}{1+e^{-x}}$$

One Encoded

1: kirmizi 0: mavili

Binary Cross Entropy

$$E(w, x, y)$$

|0| veya |1|

$$\sum_1^m \sum_J y_j \ln(p_j)$$

"model.predict(nokta)"

18



Multi Class Data

$$S(x) = e^x \cdot \sum_{i=1}^n e^{-i}$$

Hot encoded

|1001|: kirmizi

|0101|: mavili

|0011|: turuncu

Activation

Olasılık dan durur
x:skor → P

$$\begin{pmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{pmatrix}$$

Hot probability

Categorical Cross Entropy

$$E = -\sum_i^n \sum_J y_j \ln(p_j)$$

$$1001 \quad 14001 \quad 10101$$

$$\sum_i^n \sum_J y_j \ln(p_j)$$

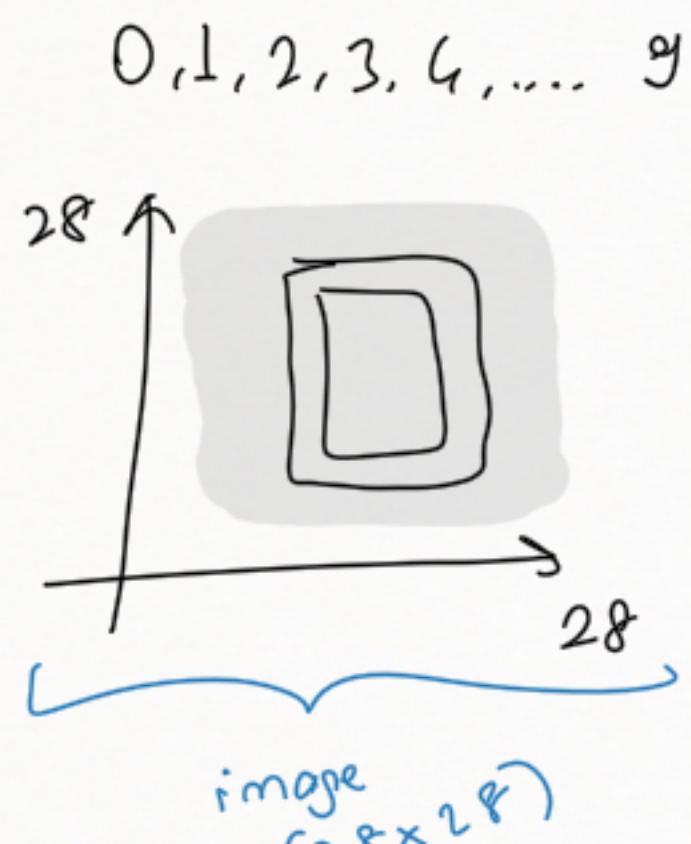
sorun var (?)

"model.predict_classes(nokta)"

Mnist Image Recognition

- Activation = relu (sigmoid, softmax yerine)
- Output activation = softmax
- "To_categorical C)" kullanılır.

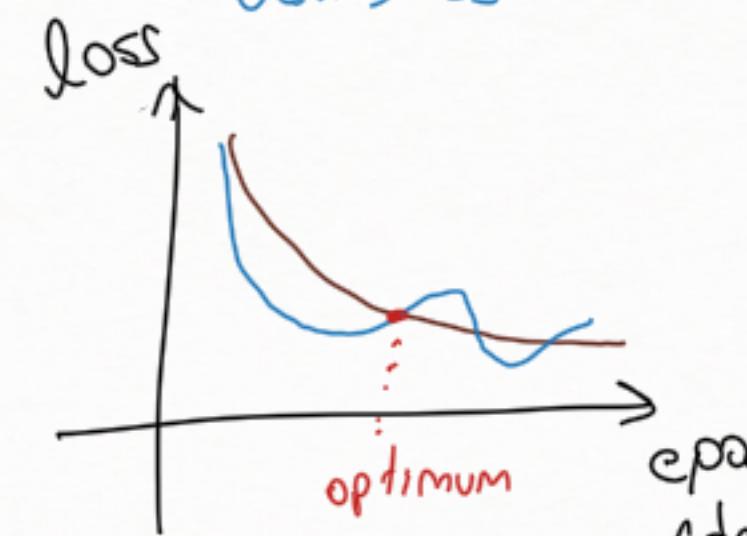
Mnist Image Recognition



» Siyah - beyaz yapılır.

Optimum Epochs

loss
vol-loss



loss: Train Set Kaybı

vol-loss: Validation Set Kaybı

» ilk kesişimden sonrakiler overfit durumu oluşturur.

» ilk kesişim optimum devirdir.

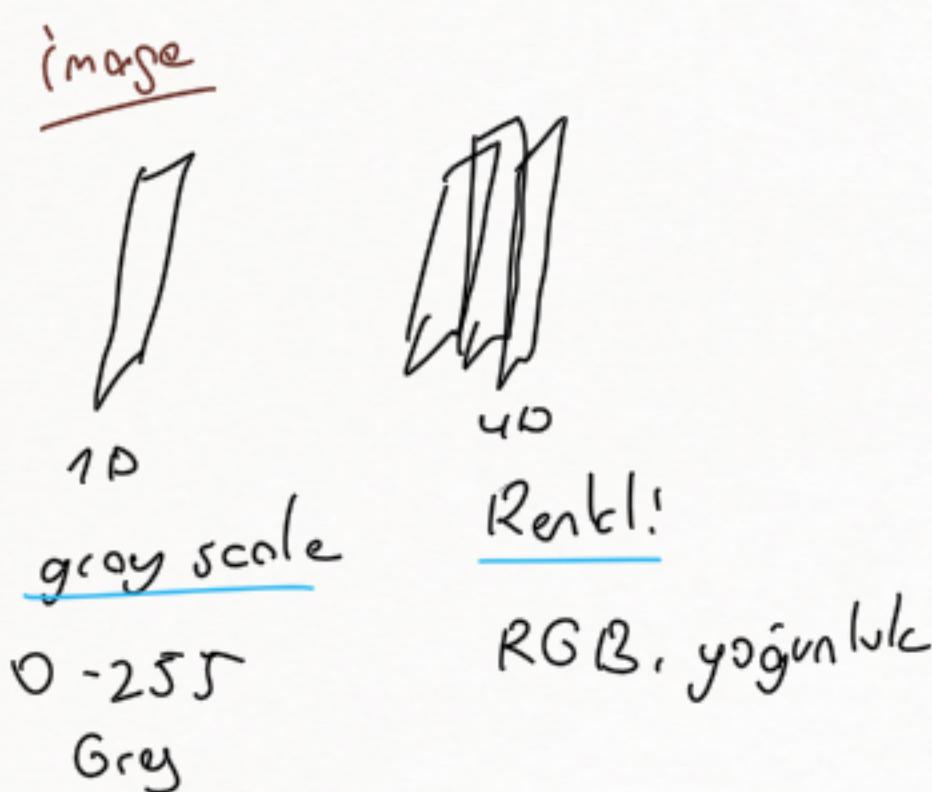
» Train set'in omaci max esitlidiktir. Validation Set'in omaci genel esitlidiktir.

» En uygun epoch bulunana kadar, neural network test edilir.

Overfit

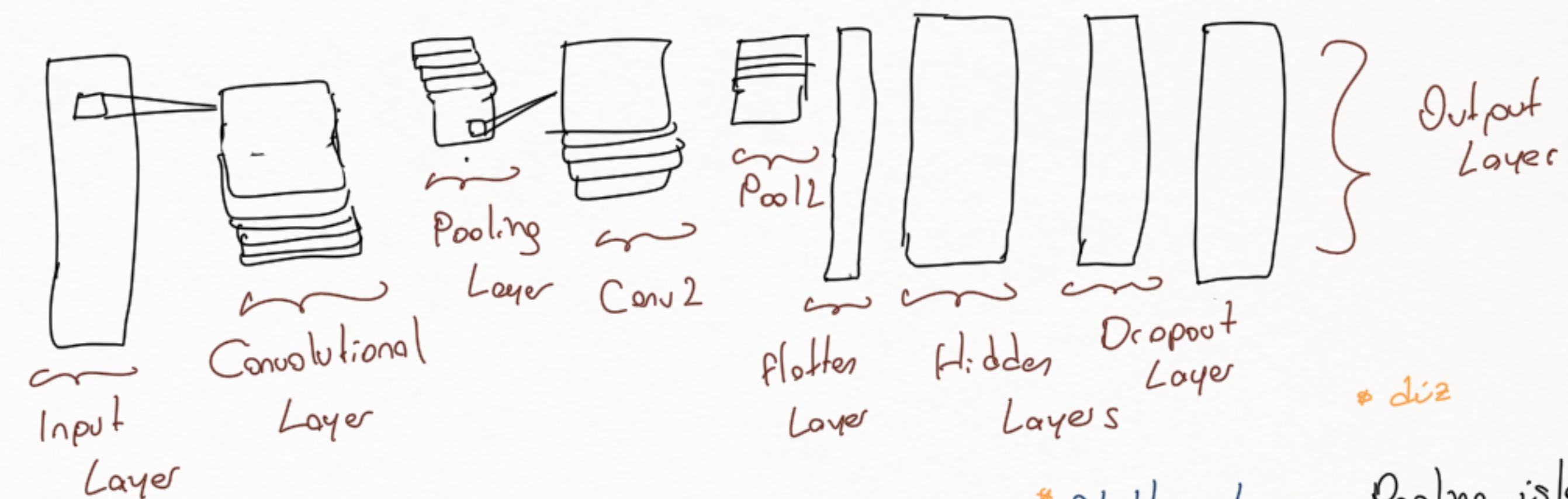
» Grizgi çok doldurulur, yamulur.

» Genelleştirme yapmaz, kesinlik arar. Sağlıklı olmaz.



Not: Ön işleme ve verileri hazırlama oldukça önemlidir.

Convolutional Neural Networks (CNN)



Convolutional Layer: Resimdeki özelliklerin haritası çıkarılır (features map)

Pooling Layer: Features Map'deki max değerleri ele alma konusun.

* flatten Layer: Pooling işleminde oluşan varyi 1D dizeye çevirme

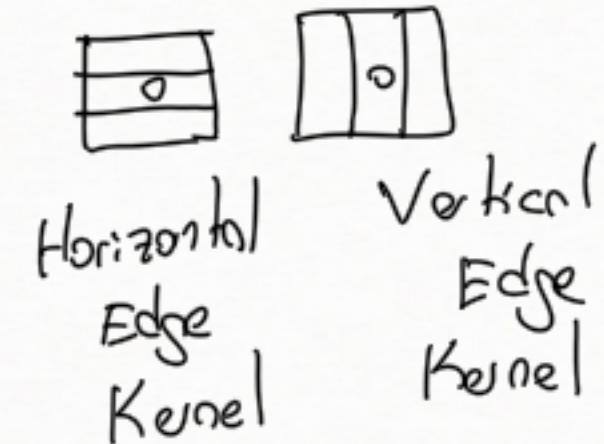
Dropout Layer: Overfitting engelleyen katman

Convolutional Layer

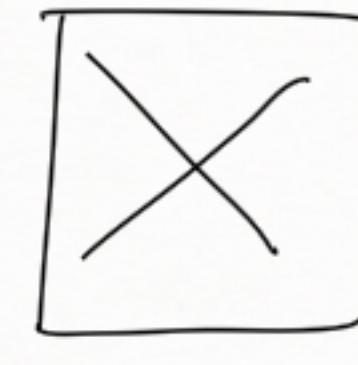
- » filtreleme işlemi yapılır.
- » Özelliklere aroma işlemidir.
- » Her pikselle uygubılır.
- » Çıktısı feature map olur.

0	-1	0
-1	5	-1
0	-1	0

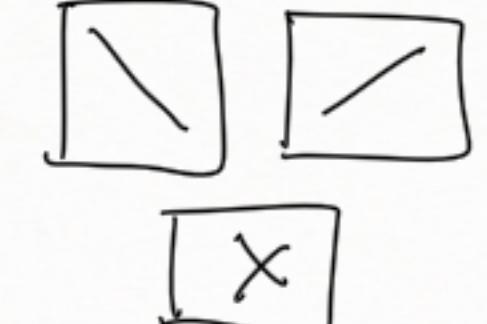
Kernel Matrix
(Filter)



ornek
olank
eşdeğerdirler.



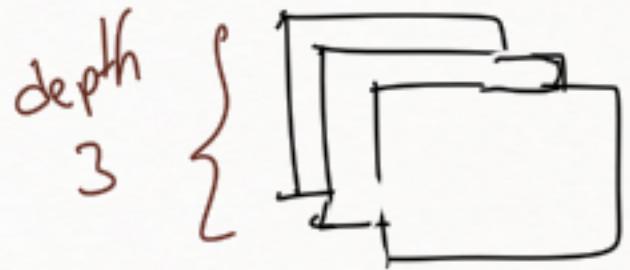
image



Kernels

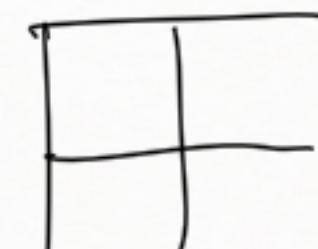
Feature Map

- » Özelliklerin bulunduğu katman
- » Kernerler ile oluşturular.
- » Kernel sayısı depth değerini verir.
- » Özellik haritası pooling layer'a sokulur.



Pooling Layer

- » Feature Map üzerinde filtreleme işlemi
- » Matrisin boyutunuOLFATIR, incelemeyi kolaylaştırır.
- » Overfit olmasını engellemek için de kullanılmaktadır.
- » Genelleştirmeyi artırmır.



2 x 2 Kernel

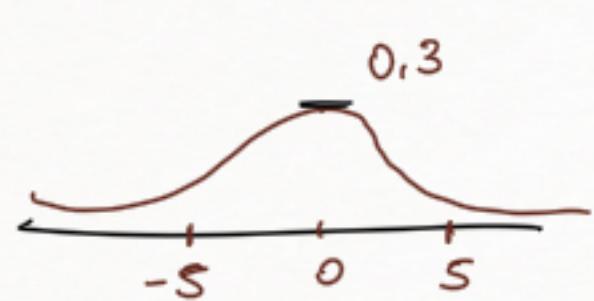
Pooling Operations

- » Sum
- » Average
- » Max

buna odaklıyız.

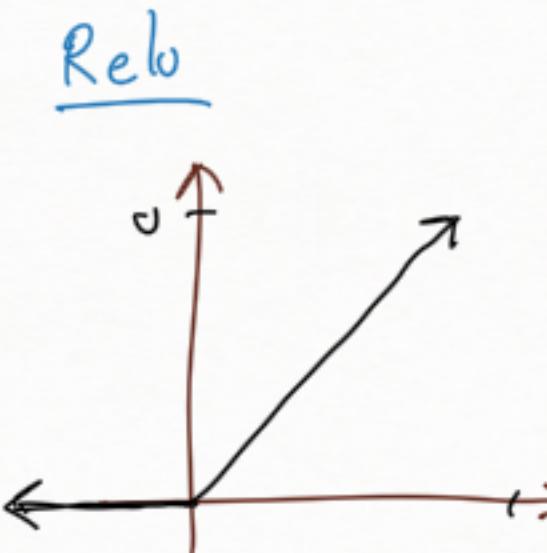
! Eğer renkli ise (3D) kernel boyutu
3D olmak zorundadır. ($3 \times 3 \times 3$)

Activation Function



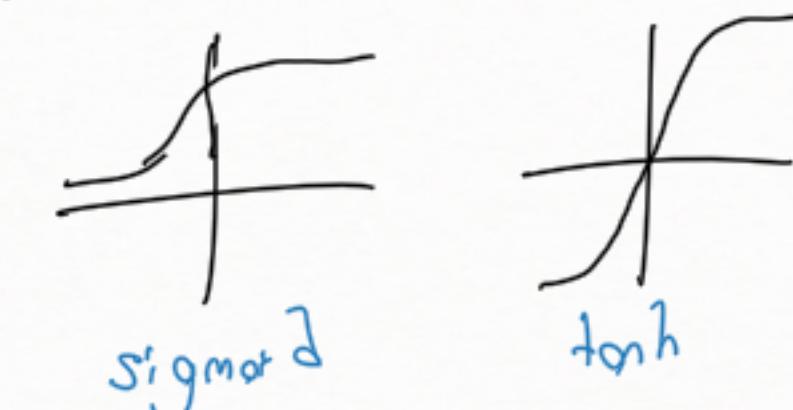
- » Sigmoid
- » Küçüklerde türəv = 0
(yani: deyisim algılanmaz.)

Relu
Pozitif iñin türəv > 1.
(Sadece oranın deyisimləri göstərir.)



$$R = \begin{cases} x, & x > 0 \\ 0, & \text{diger} \end{cases}$$

- » Relu ile deyisimler çox belli olur.
- » Sigmoid ve Tanh gibi Vanishing Gradient olmaz.
- » Vanishing Gradient: Deyisimi dəha zər onlasılır hale getirir, ögrenmeyi yarasaşdırır.



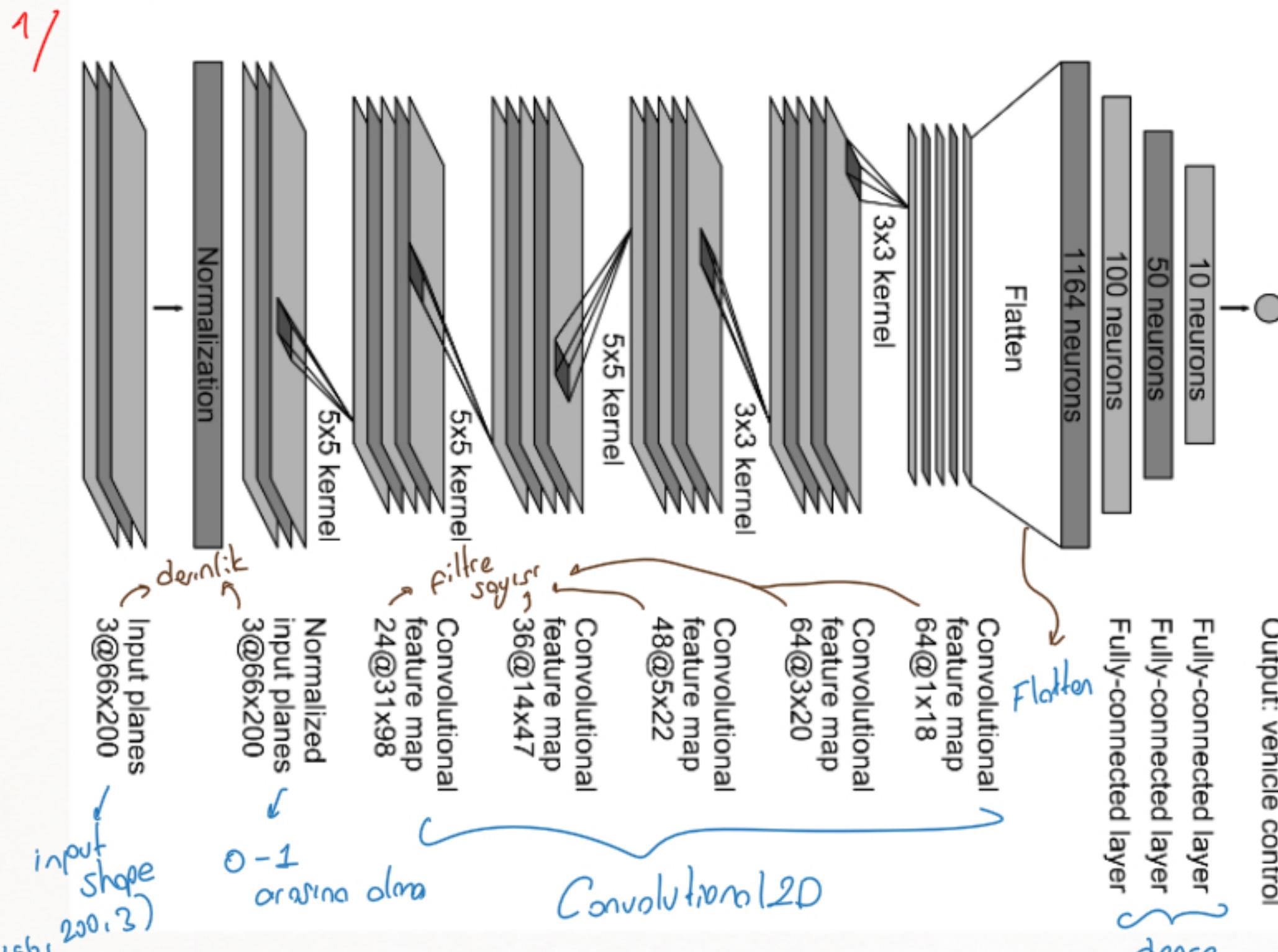
Behavioral Cloning



Nvidia Model

- Nvidia end-to-end self-driven car.

Temel Nvidia Model Yapısı



Son Kod Hali

```
def nvidia_model():
    model = Sequential()
    # Filtre sayısı, kernel boyutu,
    model.add(Convolution2D(24, 5, 5, subsample=(2, 2), input_shape=(66, 200, 3), activation='relu'))
    model.add(Convolution2D(36, 5, 5, subsample=(2, 2), activation='elu'))
    model.add(Convolution2D(48, 5, 5, subsample=(2, 2), activation='elu'))
    model.add(Convolution2D(64, 3, 3, activation='elu'))

    model.add(Convolution2D(64, 3, 3, activation='elu'))
    model.add(Dropout(0.5))

    model.add(Flatten())

    model.add(Dense(100, activation='elu'))
    model.add(Dropout(0.5))

    model.add(Dense(50, activation='elu'))
    model.add(Dropout(0.5))

    model.add(Dense(10, activation='elu'))
    model.add(Dropout(0.5))

    model.add(Dense(1)) # output

    model.compile(loss='mse', optimizer=Adam(lr=1e-3)) # 1e-3 = 10^-3
    return model
```

Kodda Kullanım

```
def nvidia_model():
    model = Sequential()
    # Filtre sayısı, kernel boyutu,
    model.add(Convolution2D(24, 5, 5, subsample=(2, 2), input_shape=(66, 200, 3), activation='relu'))
    model.add(Convolution2D(36, 5, 5, subsample=(2, 2), activation='relu'))
    model.add(Convolution2D(48, 5, 5, subsample=(2, 2), activation='relu'))
    model.add(Convolution2D(64, 3, 3, activation='relu'))
    model.add(Convolution2D(64, 3, 3, activation='relu'))
    model.add(Dropout(0.5))

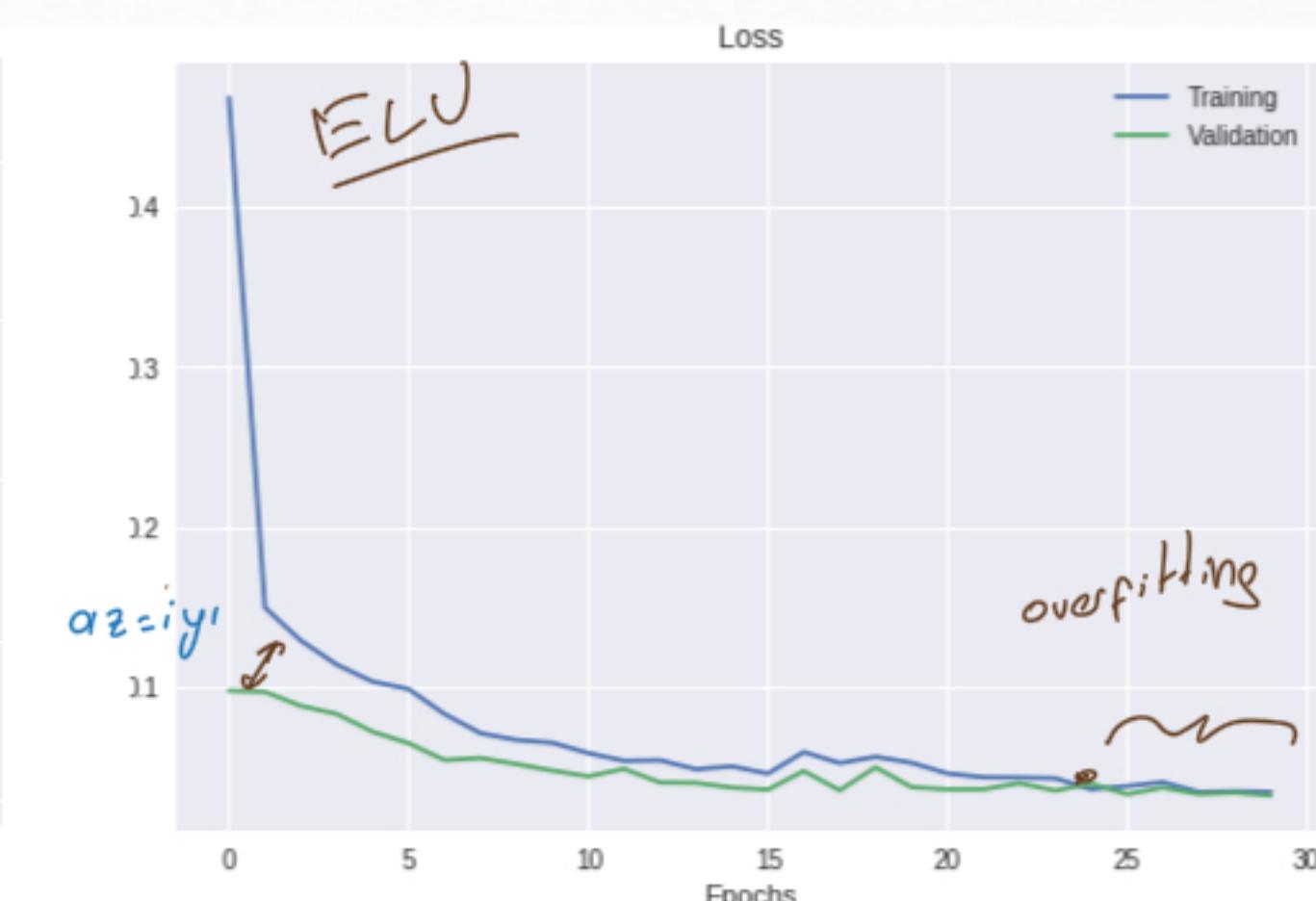
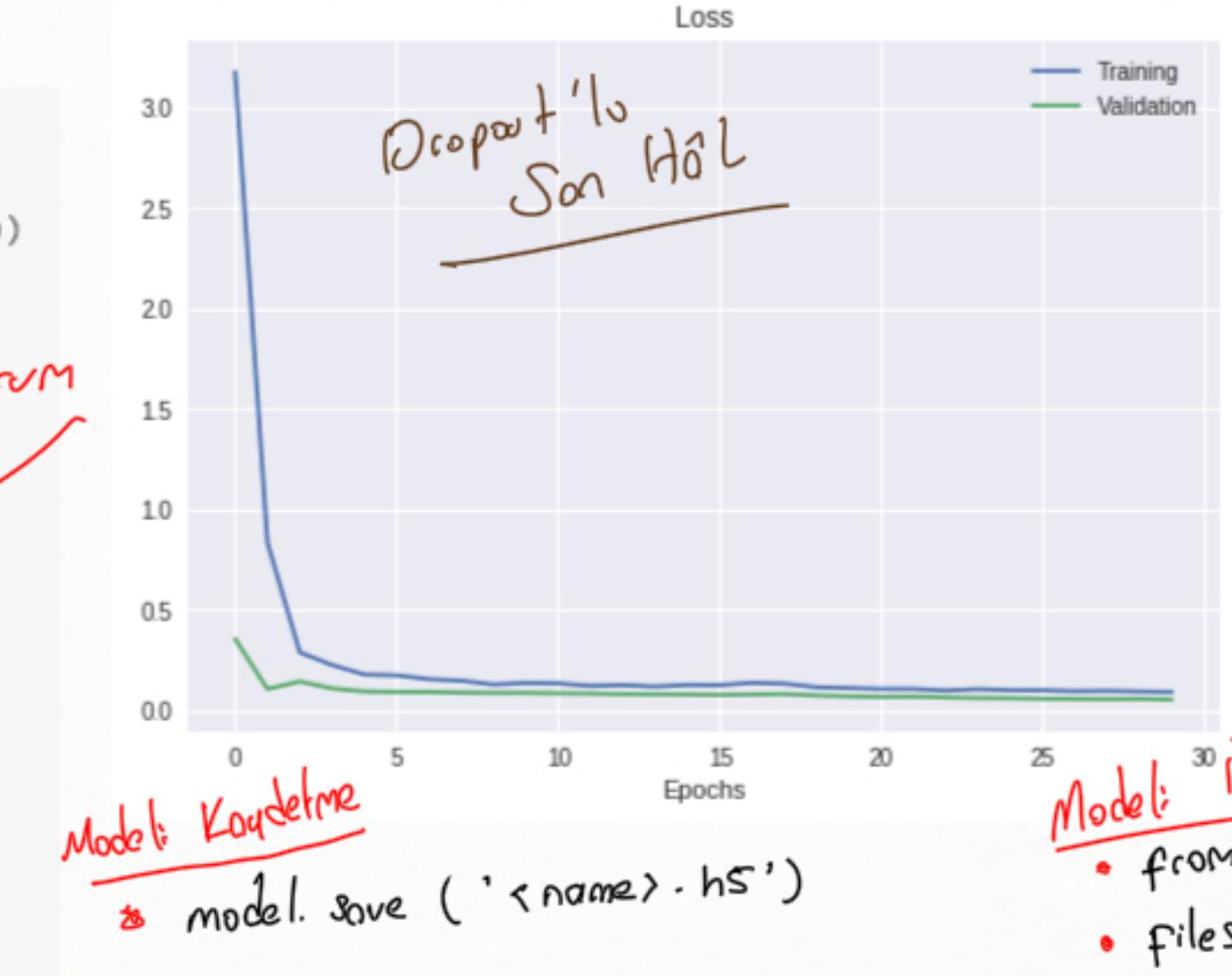
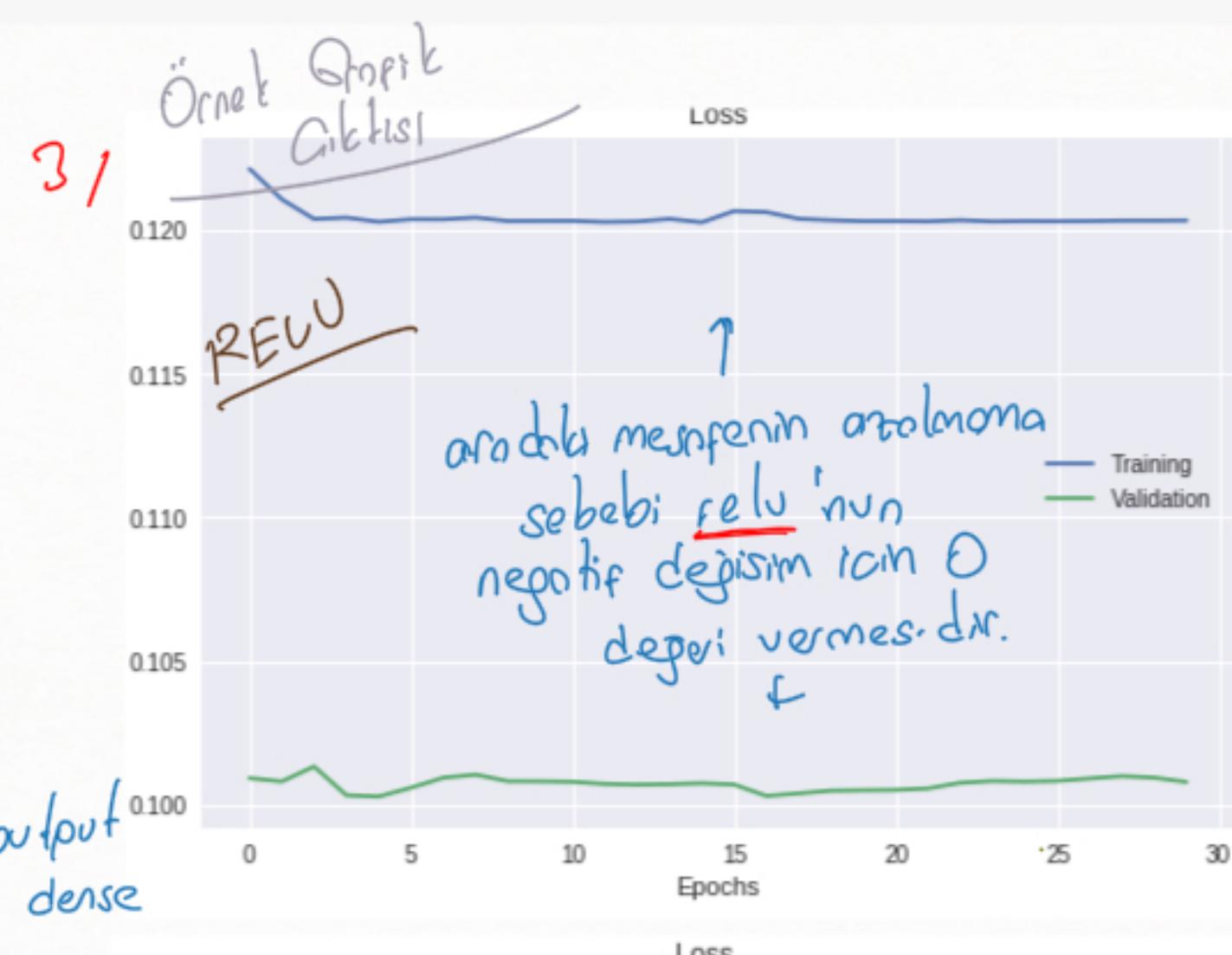
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dropout(0.5))

    model.add(Dense(50, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(1)) # output

    model.compile(loss='mse', optimizer=Adam(lr=1e-3)) # 1e-3 = 10^-3
    return model
```

! relu ile eğitim yapıldığında negatif değerler için öğrenme (back propagation) olmaz. (0'dır.) ! Relu → Elu !

hızlı ve güvenli olmasına rağmen



! Elu activation ile negatif değerlerde az da olsa sonuc geldiğinden değerler olur ve val_loss ile loss orasındaki fark azdır.

» Overfitting durumunda "Dropout" katmanı eklen. (0.5 değer ideal olabilir.)

- from google.colab import files
- files.download (<model>)

$$\text{relu} = \begin{cases} 0 & n < 0 \\ n & n \geq 0 \end{cases}$$

$$\text{elu} = \begin{cases} \lim_{n \rightarrow -\infty} e^n & n < 0 \\ n & n \geq 0 \end{cases} = -1 & n < 0 \\ n & n \geq 0 \end{cases}$$