SÜLEYMAN DEMİREL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANA BİLİM DALI

İLERİ YAPAY ZEKA DERSİ

SEVDANUR GENÇ - 1740138M62

KONU :

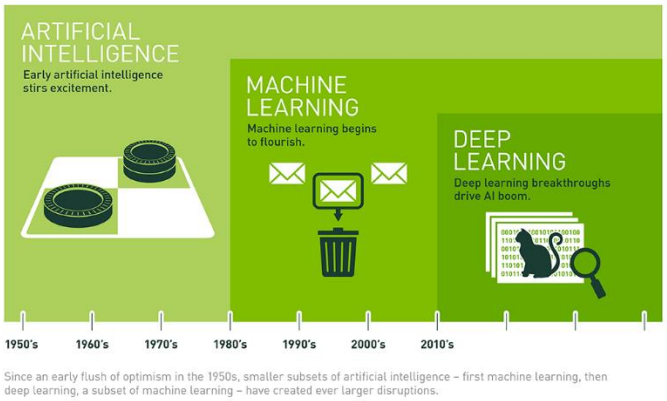
DERİN ÖĞRENME YÖNTEMİ KULLANILARAK GÖRÜNTÜ SINIFLANDIRMA

DERİN ÖĞRENME YÖNTEMİ KULLANILARAK GÖRÜNTÜ SINIFLANDIRMA

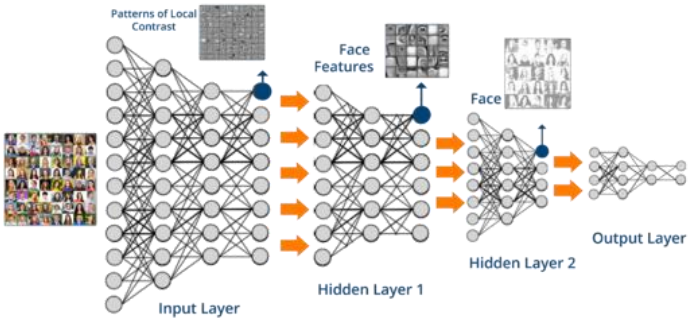
1. GİRİŞ :

1.1. DERİN ÖĞRENME NEDİR?

Derin öğrenme, 2000'lerin başından beri yapay zeka alanında çığır açan ve yaygınlaşmaya başlayan bir alandır. Aslında makine öğrenmesi tekniklerinden sadece birisi olan derin öğrenme, yapay zeka alanında yeni bir teknolojidir. Google, Microsoft ve Apple gibi önemli şirketler ses ve görüntü işleme tanımında derin öğrenme teknolojisi kullanmaktadırlar. Kullanımının asıl amacı gerekli bilgiyi, verilerden ayırt etmek için kullanılır.



Derin öğrenmede, görüntü, ses ve metin gibi verilere anlam kazandıran sunum ve soyutlama seviyelerini öğrenmek için çok katmanlı sinir ağlarını kullanırlar (Deep Neural Networks - DNN).



Derin öğrenmenin kilit noktası bir nesnenin gösterimindeki farklı katmanlardır. Her katman ayrı ayrı önceden eğitilir (Unsupervised pre-training). Bu özellik, derin öğrenme ağlarının diğer sinir ağlarından ayıran en önemli özelliktir. Bunun içinde bir resmi sınıflandırmak istiyorsak resmin en alt katmanlarından işe başlanır yani piksellerden.

1.1.1. DERİN ÖĞRENME NEDEN ÖNEMLİ?

Veri miktarının artması : Dijital ortamlarda artan, üretilen ve saklanan tüm veriler büyük veriyi kullanarak avantaj elde etmektedir.

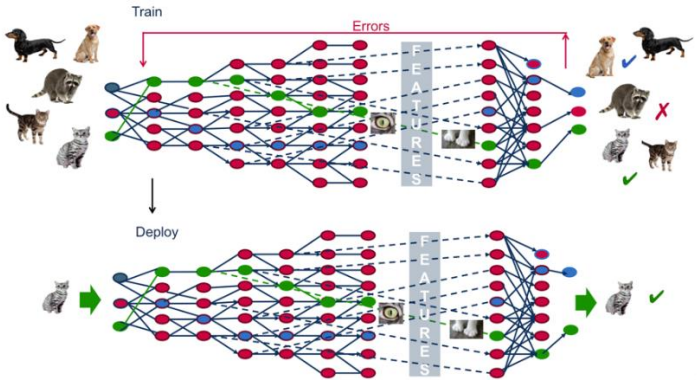
GPU'lar ve işlem gücünün artması : Grafik işlemciler, paralel hesaplama yapma konusunda özelleşmiş donanımlardır. Bu sayede, CPU'nun yavaş kaldığı bazı işlemleri çok daha hızlı yapabiliyorlar.

Derinliğin artması : İşlem gücünün artması sonucu daha derin modellerin pratikte kullanılabilmesine imkan doğdu. Derin öğrenme modelleri çok katmanlı yapılardır.

Kısacası, modellerin daha derin ve karmaşık hale gelebilmesi, bunları eğitebilen algoritmaların keşfi, bu ağların büyük verilerle eğitilebilmesi ve tüm bu sürecin bir bilgisayar veya ucuz/erişilebilir bulut servisleriyle gerçekleştirilebilir hale gelmesiyle derin öğrenme bu kadar ön plana çıkmıştır.

1.1.2. DERİN ÖĞRENMENİN KULLANILDIĞI ALANLAR?

Plaka tanıma sistemleri, yüz tanıma sistemleri, parmak izi okuyucular, iris okuyucular, ses tanıma sistemleri, sürücüsüz arabalar ve spam(istenmeyen) e-posta tespit sistemlerinde derin öğrenme kullanılabilmektedir.



1.1.3. DERİN ÖĞRENMEYE AİT ÖRNEK FRAMEWORK'LER :

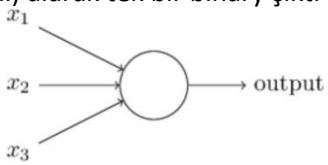
Bir takım açık kaynak proje tarafından derin öğrenme herkesçe kullanılabilir hale gelmiştir. Derin öğrenme algoritmalarını bilgisayar diline çevirmek için birçok framework vardır. Bu kütüphaneler algoritmaların uygulanmasında kolaylık sağlarlar. En çok bilinen frameworkler:

1) Deeplearning4j (DL4j)
    a. Java
    b. Dağıtık ve ölçeklenebilir
2) Theano
    a. Python
    b. Akademik çevrede kullanılır.
3) Torch
4) TensorFlow
5) Caffe

1.2. DERİN ÖĞRENME ALGORİTMALARI :

Perceptronlar 1950'lerde ve 1960'larda Warren McCulloch ve Walter Pitts'ın eserlerinden esinlenerek bilim adamı Frank Rosenblatt tarafından geliştirildi.

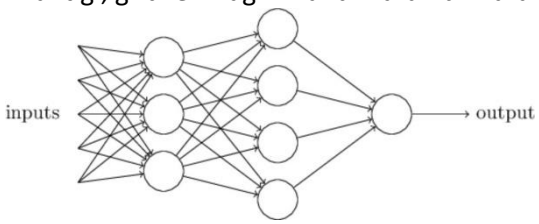
Çok sayıda binary girdi (*x*1,*x*2,…) alarak tek bir binary çıktı üretilir.



Rosenblatt çıktıyı hesaplamak için girdileri önem derecesine göre ağırlıklandırır *w*1,*w*2,… . Nöron çıktısı girdilerin ağırlık değerleriyle çarğımlarının toplamının, belli bir eşik değerinden büyük veya küçük olmasına göre değer alır.

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

Perceptronların temel mantığı, girdilerin ağırlıklandırılarak bir karar oluşturulmasıdır.



Bu ağdaki ilk katman basit kararlar almaya yarar. İkinci katman, birinci katmanda ağırlıklandırılan sonuçları kullanır. Katmanlarda ilerledikçe daha karmaşık kararlar alınır.

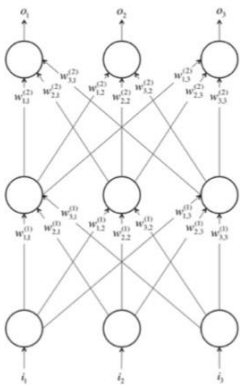
$$w \cdot x \equiv \sum_j w_j x_j$$

W ve X vektörleri ağırlık ve girdi vektörlerini temsil etmektedir. Eşik değerine bias ismini verip denklemin soluna atılıp, Perceptron kuralı tekrar yazılırsa;

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Bias değeri ne kadar büyük olursa, çıktının 0 değerini üretmesi daha kolaydır. Ama bias değeri negatif olursa çıktı değeri 0 olmaya daha yakındır.

İnsan beyni çok daha karmaşık bir yapıya sahiptir ve çok sayıda katman içerir. Cerebral cortex altı tane karman içerir ve bilgiler bir katmandan diğerine anlamlı bir bilgi elde edilene kadar iletilir. Örneğin, en alttaki katman gözlerden görme verilerini alır ve bu veriler bir katmandan diğerine iletilerek, ne görüldüğünü anlamaya yarar.



En alttaki katmanlar verileri alır. En üstteki çıktı katmanı ise sonucu görmemizi sağlar. Ortaki katmanlar “hidden layer” olarak isimlendirilir.

K. Katmandaki i. nöronu temsil eder. J ise bir sonraki katmandaki (k+1) nöronu temsil eder.

Çıktı üretmek için bir katmandan diğerine verilerin iletildiği, bir katmanın çıktısının diğerinin girdisi olarak kabul edildiği networklere *feedforward neural networks* denir.

Ağda hiç döngü yoktur, sürekli ileriye doğru gider.

LINEER NÖRONLAR

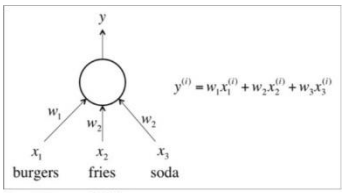


Figure 1-10. An example of a linear neuron

Birçok nöron tipi f fonksiyonuyla tanımlanır.

Lineer olarak f(z) = az+b şeklinde tanımlanır fakat hidden layer'lar girdilerden önemli öznitelikler ve kompleks ilişkiler öğrenmemizi sağladığından doğrusal ağlar yetersiz kalır.

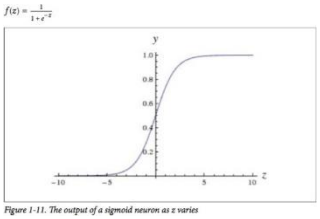


Figure 1-11. The output of a sigmoid neuron as z varies

Non-linearity'i sağlayan üç çeşit nöron tipi vardır.

Logit çok küçük olduğunda, logistic nöronun çıktısı sıfıra yakın olur. Logitler çok büyük olduğunda ise çıktı 1'e yakın olur. Sigmoid fonksiyonlarda aynı zamanda, ağırlık ve bias'taki küçük değişimler, çıktıda da küçük değişimlere sebep olur.

$\sigma(w \cdot x + b)$ şeklinde tanımlanan sigmoid fonksyion,

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}. \quad \text{—>} \quad \frac{1}{1+\exp(-\sum_j w_j x_j - b)}.$$

Z = wx+b negatif olduğunda $e^{-z} \rightarrow \infty$ olur ve $\sigma(z) \approx 0$

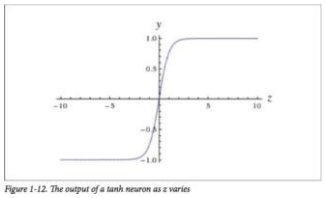


Figure 1-12. The output of a tanh neuron as z varies

Tanh nöronlarında ise çıktılar 0-1 arasında değil, -1 -1 arasında olur.

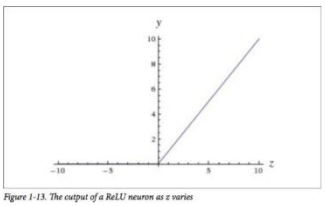


Figure 1-13. The output of a ReLU neuron as z varies

Non-linearity'i sağlayan bir diğer nöron tipi restricted linear unit (ReLU) dir. *f(z)= max(0,z)*

SOFTMAX OUTPUT LAYER

Çıktıları olasılık dağılımı cinsinden elde etmemizi sağlayan fonksyiondur.

$$z_j^L = \sum_k w_{jk}^L a_k^{L-1} + b_j^L$$

Önce girdileri ağırlıklandıralım.

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}},$$

j. Çıktı nöronunun aktivasyon değeri ise

$$\sum_j a_j^L = \frac{\sum_j e^{z_j^L}}{\sum_k e^{z_k^L}} = 1.$$

En optimal ağırlık değerlerini bulmak için, hata değerini minimize etmeliyiz.

Gerçek değer ile tahmin edilen değer arasındaki farklar en az seviyede olmalıdır.

$E = \frac{1}{2}\Sigma_i\left(t^{(i)} - y^{(i)}\right)^2$ E değeri sıfır olduğunda model doğru tahmin edilmiş demektir.

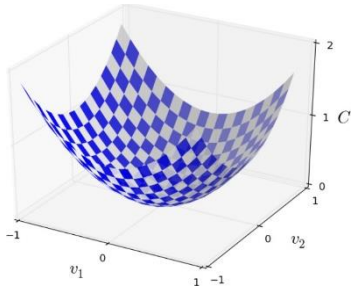
COST FUNCTION

$$C(w,b) \equiv \frac{1}{2n}\sum_x \|y(x) - a\|^2.$$

w -> Networkteki tüm ağırlıklar
b -> Tüm bias değerleri
n -> Eğitim veirlerinin sayısı
a -> X girdisi için oluşan çıktı
C -> Quadratic cost function (mean squared error)
y(x) değeri çıktı olan a değerine eşit olduğunda cost fucntion 0'a eşit veya yakın olur.

Cost function'ı minimum yapacak ağırlık ve bias değerlerini bulmak istiyoruz.

W ve b değerilerine v1 ve v2 dersek, C değerinin global minimum noktasını bulmak istiyoruz.



Türev hesaplayarak tekrar C fonksiyonunun extremum noktalarını hesaplayabiliriz. Fakat çok fazla değişken olduğunda (milyarlarca) türev hesaplamak uzun süre alacağından gradient descent metodu önerilmiştir.

V1 ve v2 eksenlerinde minimum noktasına erişmek için küçük miktarda ilerlemeler yapılır.

$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2.$ V1 ve v2 değerlerinde değişimler Delta C değeri negatif olacak şekilde yapılır. Delta V değeri eksenlerdeki değişimlerin tutulduğu vektördür.

$$\Delta v \equiv (\Delta v_1, \Delta v_2)^T$$

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T.$$

$$\Delta C \approx \nabla C \cdot \Delta v.$$

Delta C değeri v değerindeki değişime göre cost fonksiyonundaki değişimleri gösterir. v eksenlerindeki değişime göre C nin negatif olması için "learning rate" isminde negatif bir parametre eklenmelidir.

$$\Delta v = -\eta \nabla C,$$
$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2.$$

Kare ifadesi pozitif olduğu için c değeri sürekli azalmaktadır.

$$v \rightarrow v' = v - \eta \nabla C.$$

v değerleri (ağırlık ve bias) güncellenir.

$$\Delta v = (\Delta v_1, \ldots, \Delta v_m)^T$$

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m} \right)^T.$$

$$w_k \rightarrow w_k' = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \rightarrow b_l' = b_l - \eta \frac{\partial C}{\partial b_l}.$$

Yukarıda v değeri 2 adet olmasına rağmen, çoğu zaman c fonksyionu çok sayıda değişken içerebilir. Bu durumlarda yerel minimum noktasının bulunması zorlaşabilir. Ayrıca v değerlerinin (w ve b) tekrar tekrar güncellenmesi gerekir.

Pratikte tüm cost fonksyionları hesaplamak için tüm eğitim setindeki verilerin cost fonksiyonunu hesaplayıp ortalamalarını almamız gerekir. Fakat eğitim setleri büyüdükçe bu durum çok fazla zaman alabilir.

Bu durumda hızı arttırmak için "stochastic gradient descent" yöntemi kullanılır. Bu yöntemde gradient descent yöntemi rastgele seçilen eğitim örnekleri ile yapılır. Bu rastgele seçilen eğitim örneklerini *X*1,*X* 2,…, *Xm* "mini-batch" olarak temsil edebilirz. M değerini yeterince büyük seçersek bulunan cost değeri, tüm setin değerine yaklaşabilir. Epoch değeri kadar eğitim seti seçilip, farklı mini-batchlerle cost fonksyionları hesaplanabilir.

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C,$$

$$w_k \rightarrow w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_l \rightarrow b_l' = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l},$$

Kısaca her ağırlık ve bias değerini güncellemek için hata fonksiyonunun türevi alınır. Bu adım her iterasyonda ağırlıklar için tekrarlanır.

$$\Delta w_k = -\epsilon \frac{\partial E}{\partial w_k}$$

$$= -\epsilon \frac{\partial}{\partial w_k} \left(\frac{1}{2} \Sigma_i \left(t^{(i)} - y^{(i)} \right)^2 \right)$$

$$= \Sigma_i \epsilon \left(t^{(i)} - y^{(i)} \right) \frac{\partial y_i}{\partial w_k}$$

$$= \Sigma_i \epsilon x_k^{(i)} \left(t^{(i)} - y^{(i)} \right)$$

Sigmoid fonksiyonuna sahip bir sinir ağını düşünürsek logistic nöronlar çıktı değerlerini şu şekilde hesaplar:

$$z = \Sigma_k w_k x_k$$

$$y = \frac{1}{1 + e^{-z}}$$

Ağırlıklarındırılmış girdilerin toplamını logit z olarak hesapladıktan sonra bu değer ile sigmoid fonksiyon hesaplanır. Bu logit değerinin ağırlık ve girdilere göre türevi alınır. Çıktı değerlerinin de logit'e göre türevi alınır.

$$\frac{dy}{dz} = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{1}{1+e^{-z}} \frac{e^{-z}}{1+e^{-z}}$$

$$= \frac{1}{1+e^{-z}} \left(1 - \frac{1}{1+e^{-z}}\right)$$

$$= y(1-y)$$

$$\frac{\partial y}{\partial w_k} = \frac{dy}{dz} \frac{\partial z}{\partial w_k} = x_k y(1-y)$$

$$\frac{\partial E}{\partial w_k} = \Sigma_i \frac{\partial E}{\partial y^{(i)}} \frac{\partial y^{(i)}}{\partial w_k} = -\Sigma_i x_k^{(i)} y^{(i)} \left(1 - y^{(i)}\right) \left(t^{(i)} - y^{(i)}\right)$$

Her hidden katmanı birçok çıktı değerlerini etkileyebilir. Bir hidden katmanının hata oranını hesapladıktan sonra bu hata oranı gerideki katman için kulanılır.

Örneğin, katman j için hata türevlerini hesaplamış olalım. Şimdi katman j'nin bir alt katmanı olan i için olan hata türevini hesaplayalım. Katman i'deki her bilgi katman j'yi de etkilemektedir.

BACKPROPOGATION ALGORITMASI

1. **Input** x**:** Set the corresponding activation a^1 for the input layer.
2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.
3. **Output error** δ^L**:** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.
4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.
5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

1. **Input a set of training examples**
2. **For each training example** x**:** Set the corresponding input activation $a^{x,1}$, and perform the following steps:
   - **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$.
   - **Output error** $\delta^{x,L}$**:** Compute the vector $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$.
   - **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$.
3. **Gradient descent:** For each $l = L, L-1, \ldots, 2$ update the weights according to the rule $w^l \to w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$, and the biases according to the rule $b^l \to b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$.

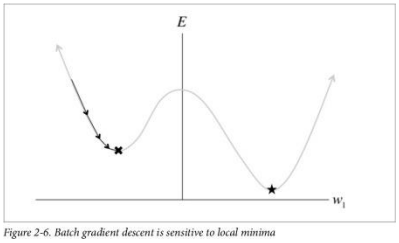


Figure 2-6. Batch gradient descent is sensitive to local minima

Bazı durumlarda optimal minimum noktasını bulmaya çalışırken, yerel minimum noktalarında sıkışıp kalınabilir.

Stochastic gradient descent yaklaşımıyla, her iterasyonda hata fonksiyonu her bir örnek için yeniden hesaplanır. Hata oranı dinamik olarak yeniden oluşur. Fakat her bir örnek için hata oranını hesaplamak zaman açısından uygulanabilir bir yöntem değildir. Bu nedenle mini-batch ile küçük setler halinde hata oranı hesaplanarak yerel minimum noktalarından kaçılır. SGD metodunda küçük grupların hata oranları hesaplanarak ortalamaları alınır. Bu değer azalmayı bıraktığında yerel minimum noktasına ulaşılmış olarak düşünülür.

$\Delta w_{ij} = -\Sigma_{k \in mini-batch} \epsilon y_i^{(k)} y_j^{(k)} \left(1 - y_j^{(k)}\right) \frac{\partial E^{(k)}}{\partial y_j^{(k)}}$ Hata fonksiyonu türevi hesabını tekrar açıklayacak olursak;

$y_k = \sum_i w_{ki} x_i$ Yk değerleri çıktıyı, xk değerleri ise girdiyi temsil etmektedir.

$E_n = \frac{1}{2}\sum_k (y_{nk} - t_{nk})^2$ Hata fonksiyonu.

$$y_{nk} = y_k(\mathbf{x}_n, \mathbf{w}).$$

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})x_{ni}$$

Ağırlıklara göre hata fonksiyonunun gradienti

$a_j = \sum_i w_{ji} z_i$ Z değeri aktivasyon değeridir. Her katmanda girdilerin ağırlıklandırılmış değeri hesaplanır.

$z_j = h(a_j).$ Bir sonraki katmanın aktivasyonu, aj değeri ile hesaplanır.

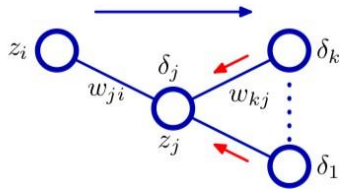
Hata fonksiyonunun türevi almak için;

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}.$$

$\delta_j \equiv \frac{\partial E_n}{\partial a_j}$ —>. $\delta_k = y_k - t_k.$

$\frac{\partial a_j}{\partial w_{ji}} = z_i.$ $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i.$

Illustration of the calculation of δ_j for hidden unit j by backpropagation of the δ's from those units k to which unit j sends connections. The blue arrow denotes the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information.



$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}$ Aj değerindeki değişimler, ak değerindeki değişimler ile hata fonksiyonuna etmektedir. Bu durumda back-propogation algoritması;

$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k$ Şeklinde olur. Yani bir katmandaki sigma değerleri, daha üst katmanlardan back propogation yapılarak elde edilir.

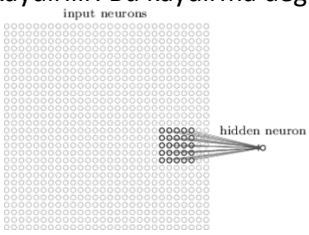
Error Backpropagation

1. Apply an input vector $\mathbf{x}_n$ to the network and forward propagate through the network using (5.48) and (5.49) to find the activations of all the hidden and output units.
2. Evaluate the δ_k for all the output units using (5.54).
3. Backpropagate the δ's using (5.56) to obtain δ_j for each hidden unit in the network.
4. Use (5.53) to evaluate the required derivatives.

CONVOLUTIONAL NEURAL NETWORKS : CNN

Resim tanımada, sıradan sinir ağlarında her piksel ayrı değerlendirilir. Fakat özellikle bu tanıma yönteminde komşu pikseller arasındaki korelasyon önem taşımaktadır.

Konvolisyonal ağlarda katmanlar "feature map" olarak adlandırılan bir yapı içinde organize edilir. Her katman girdi olarak resmin küçük bir bölgesini alır. Feature map'deki her katman aynı ağırlık değerlerini paylaşır. İlk katmandaki her bir nöron, girdi katmanındaki küçük bir bölgeye denk gelir. Girdi resmindeki bu bölgeye "local receptive field" adı verilir. Giriş piksellerindeki küçük bir pencere anlamanı da gelebilir. Her bağlantı bir ağırlık değeri oluşturur. "local receptive field" tüm resim boyunca birer piksel birer piksel kaydırılır. Bu kaydırma değerine "stride" adı verilir.



RELU fonksiyonu konvolisyonal ağlarda non-linearity'i sağlamak için kullanılır. RELU ile negatif pikseller 0'a çevirilir. Diğer doğrusal olmayan fonksiyonlara göre (tanh veya sigmoid), RELU daha iyi sonuç vermektedir.

POOLING

Pooling yöntemi subsampling veya downsapling olarak da bilir. feature map'lerin boyutunu azaltırken önemli bilgileri kaybetmez. max, average ve sum gibi çeşitleri vardır.

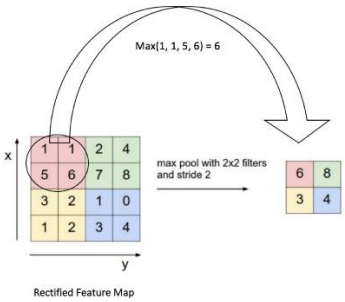


Figure 10: Max Pooling. Source [4]

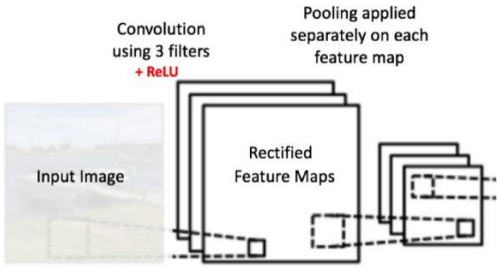


Figure 11: Pooling applied to Rectified Feature Maps

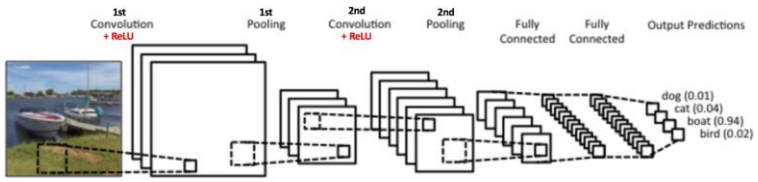


Figure 13

Fully Connected katmanı softmax aktivasyon fonksiyonunu kullanan ve bir önceki katmandaki nöronlarının hepsinin diğer katmandaki nöronlara bağlı olduğu sistemlerdir. Konvolisyon ve pooling işlemlerinden sonra ortaya çıkan öznitelikler, yüksek seviyeli özniteliklerdir. Bu yüzden tüm bu öznitelikler sınıflandırma için kullanılmak istenir.

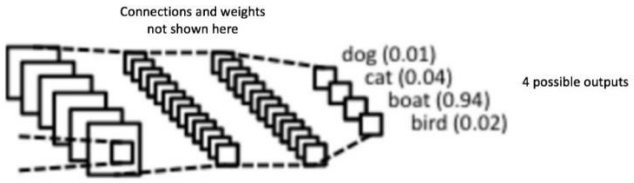
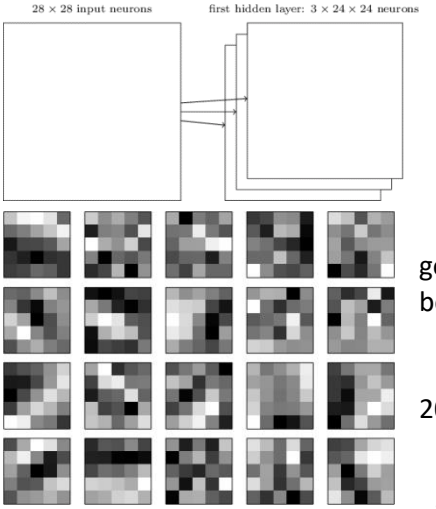


Figure 14: Fully Connected Layer -each node is connected to every other node in the adjacent layer

Pooling + konvolisyon adımları —> feature extractor
Fully Connected —> classification Her hidden layer,
bir bias değerine ve 5x5 (örneğin) adet ağırlık değerine sahiptir.

$$\sigma\left(b+\sum_{l=0}^{4}\sum_{m=0}^{4}w_{l,m}a_{j+l,k+m}\right)$$

Bias değeri ortaktır. w(l,m) ise 5x5 değerinde ağırlık vektörüdür. a ise x,y pozisyonundaki aktivasyon değeridir. Bu ağırlık vektörlerine aynı zamanda "kernel" veya "filtre" de denebilir.



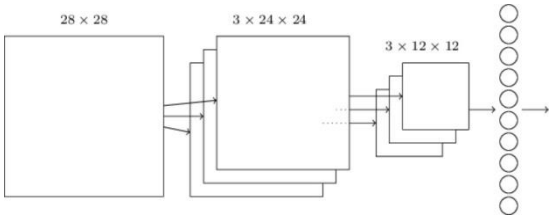
Yandaki örnekte 3 tane feature map vardır. her feature map 5x5 boyutunda ağırlık değeri ve tek bir bias değeri paylaşır.

Sonuç olarak 3 farklı öznitelik çeşidi elde edilir.

Bu örnekte 20 resim 20 farklı feature map'e karşılık gelmektedir. Her map 5x5 boyutundaki ağırlığın resimdeki 5x5 boyutundaki bloğunu temsil etmektedir.

5x5 = 25 ağırlık değeri + bias = 26 parametre 20 resim için 20x26=520 parametre elde edilir.

Fakat 30 tane hidden nöron içeren fully- connected bir ağımız olsaydı, 28x28 boyutundaki bir resim için 28x28x31 adet parametremiz olmuş olacaktı. Bu nedenle konvolisyonal ağların parametreleri azaltması hız açısından oldukça önemlidir.



Yandaki örnekte 28x28 tane giriş nöronu vardır. 3x3 boyutunda feature map ve 5x5 boyutundaki "local receptive field" uygulanmıştır ve 3x24x24 tane hidden öznitelik nöronu elde edilmiştir. Daha sonra bu 3 feature map'e, 2x2 boyutundaki bölge için max pooling yapılmıştır. Sonuç nöronu 3x12x12 boyutundadır.

2. PROJE :

2.1. KULLANILAN EKİPMANLAR :

Geliştirilen projede Matlab R2017a kullanılmıştır. Matlab içerisinde ***Usb WebCam paketi*** ve ***Neural Network Toolbox*** eklentileri kullanılmıştır.

2.2. PROJENİN AMAÇ VE KAPSAMI :

Neural Network Toolbox içerisinde bulunan dataset kullanılarak etrafımızda var olan objelerin dataset'teki image dosyaları ile eşleştirilmesi yapılmaktadır. Bu eşleştirmeler sonucunda objenin ne olduğuna dair gerekli olası ismini ve olasılık değerini ekrana yazdırmak amaçlanmıştır.

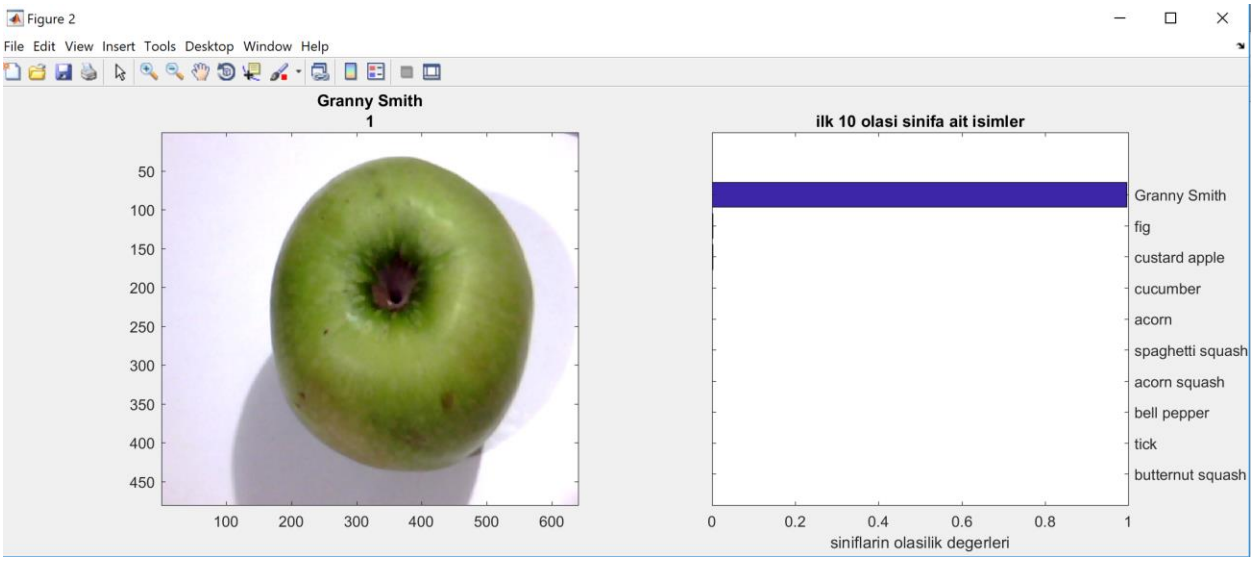
Proje kapsamında kullanılan kodların akışı ise şu şekilde gerçekleşmektedir.

Öncelikle yatay eksende hem kamera görüntüsü için hem de olasılık değerlerinin listelendiği histogram görüntüsünün yan yana gözükmesi için iki adet figure (image) ayarlanmaktadır. Ardından kameranın sürekli görüntü yakalayabilmesi için while döngüsüne true özelliği tanımlanmaktadır. Bu durum, figure ekranının kapatılmasına kadar devam edecektir. Figure ekranı kapatıldığında false değeri alacaktır. Kamera bağlantısı için webcam paketi yüklenilir hemen ardından kullanılacak neural network'e ait dataset yüklenmektedir.

While döngüsü içerisinde kameranın sürekli yakalamış olduğu görüntüleri bir değişkende tutarız ve ardından bu değişkeni neural network'te bulunan sınıf isimleri ve onlara ait olan olasılık değerleriyle karşılaştırıyoruz. Dataset içerisinde bulunan image'larla birebir eşleştirmesi sonucunda ilk image'a benzeyen ilk 10 image'a ait sınıf isimlerini dataset'ten çekeren histograma yazdırıyoruz. Aynı zamanda histogram'ın yatay eksenini 0-1 arasında olasık değerlerine uygun olacak şekilde değerleri yazdırıyoruz.

Sonuç olarak, kameradan gelen görüntülerin derin öğrenme algoritması kullanılarak dataset'te bulunan image'larda derinlemesine araştırma yapar ve olasılık değerlerine göre sonuçları ekrana yazdırmaya çalışır.

2.3. PROJENİN EKRAN ÇIKTILARI :



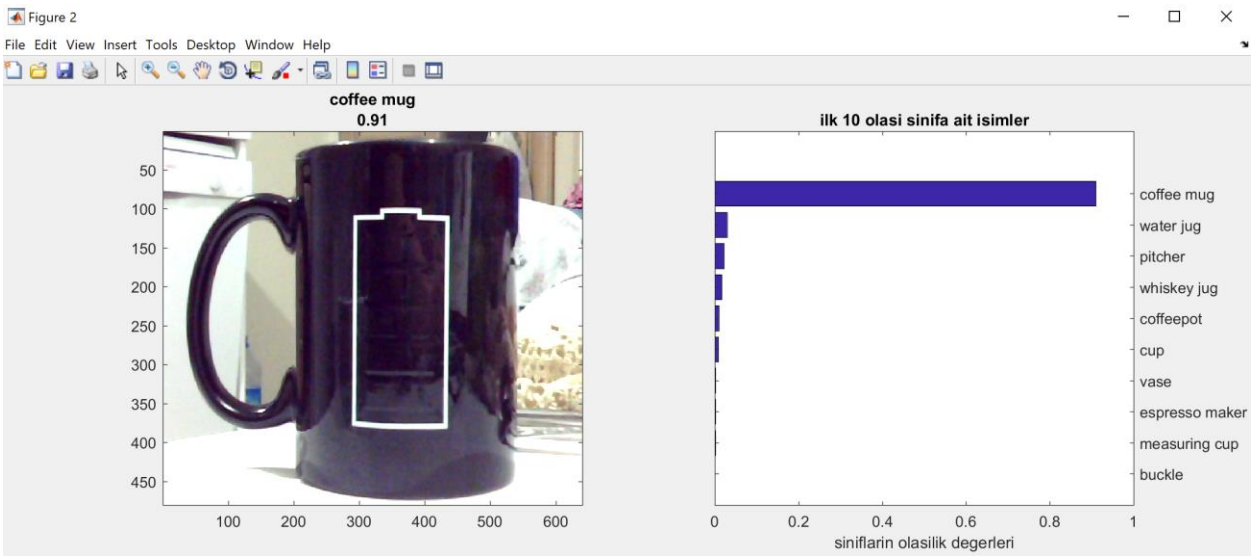


Figure 2
File Edit View Insert Tools Desktop Window Help
coffee mug
0.91
ilk 10 olasi sinifa ait isimler
coffee mug
water jug
pitcher
whiskey jug
coffeepot
cup
vase
espresso maker
measuring cup
buckle
siniflarin olasilik degerleri

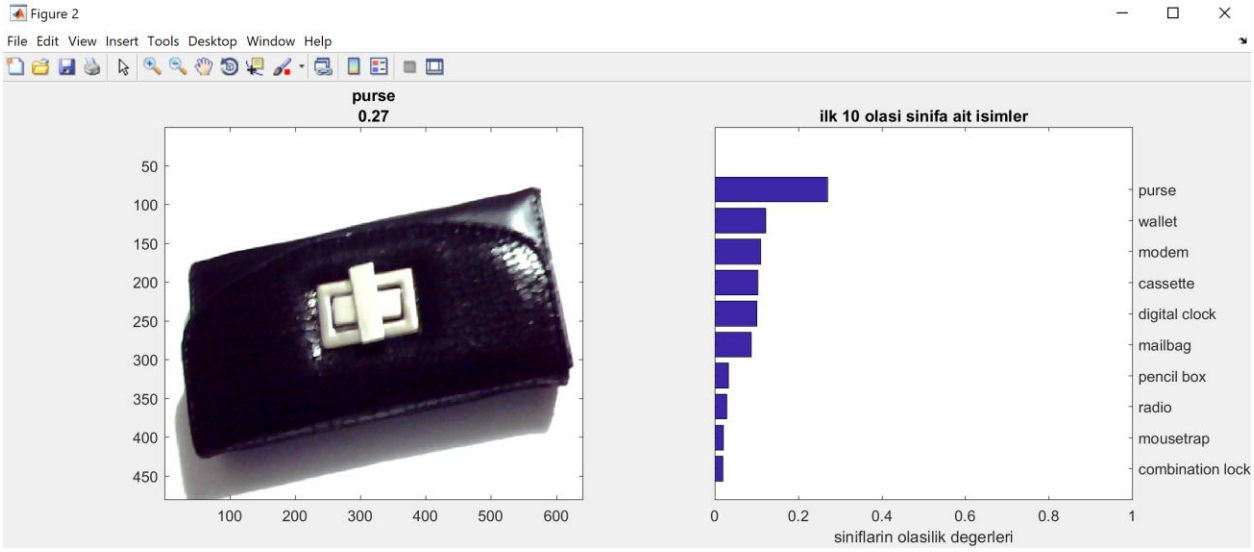


Figure 2
File Edit View Insert Tools Desktop Window Help
purse
0.27
ilk 10 olasi sinifa ait isimler
purse
wallet
modem
cassette
digital clock
mailbag
pencil box
radio
mousetrap
combination lock
siniflarin olasilik degerleri

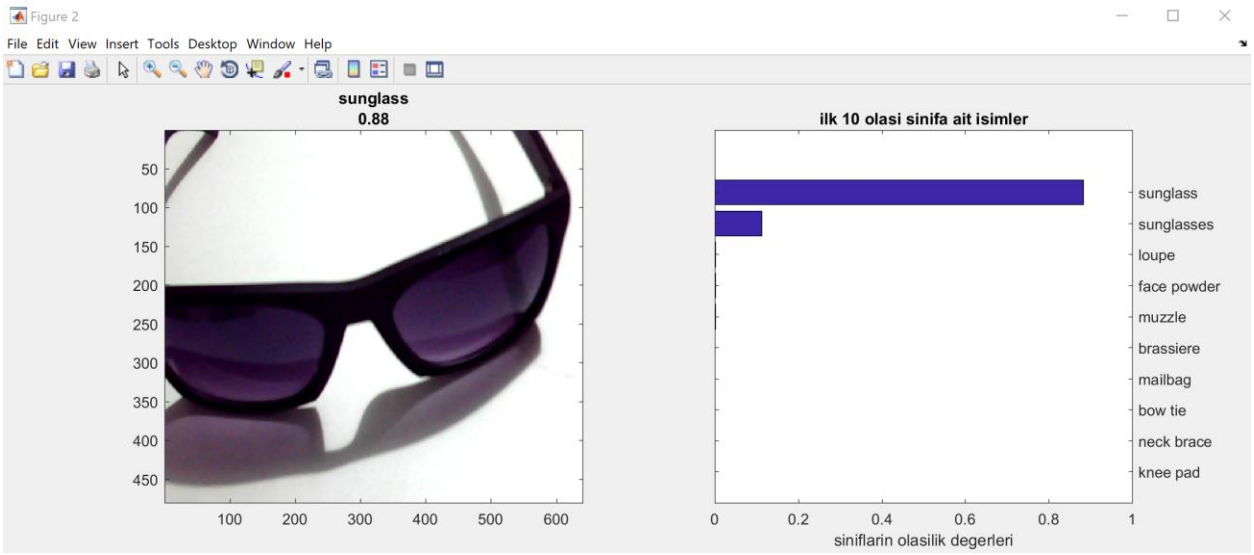


Figure 2
File Edit View Insert Tools Desktop Window Help
sunglass
0.88
ilk 10 olasi sinifa ait isimler
sunglass
sunglasses
loupe
face powder
muzzle
brassiere
mailbag
bow tie
neck brace
knee pad
siniflarin olasilik degerleri

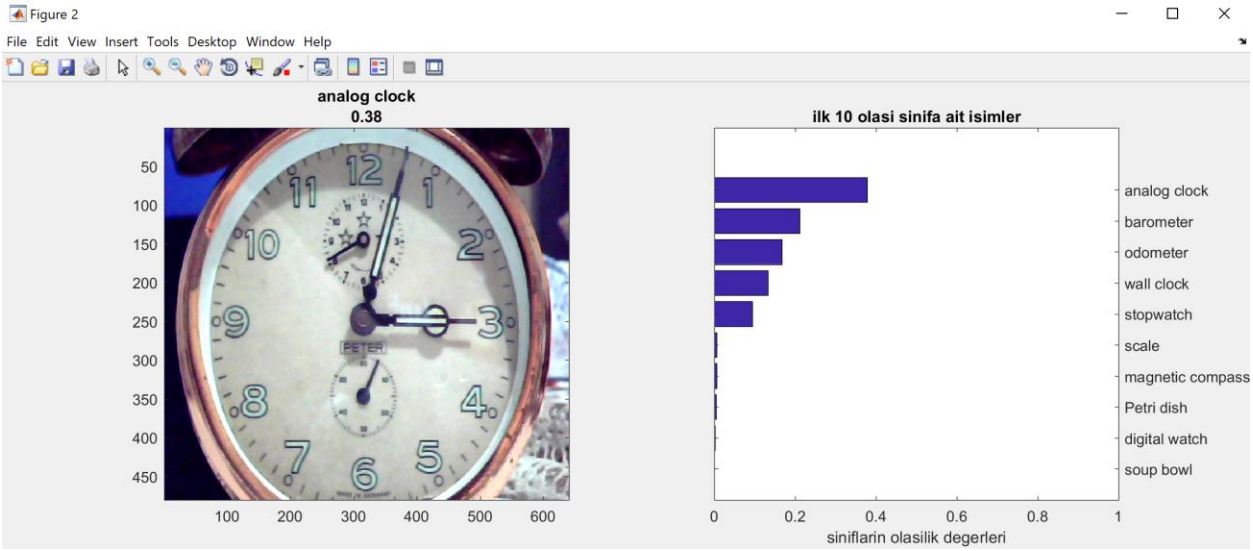


Figure 2
File Edit View Insert Tools Desktop Window Help
analog clock
0.38
ilk 10 olasi sinifa ait isimler
analog clock
barometer
odometer
wall clock
stopwatch
scale
magnetic compass
Petri dish
digital watch
soup bowl
siniflarin olasilik degerleri

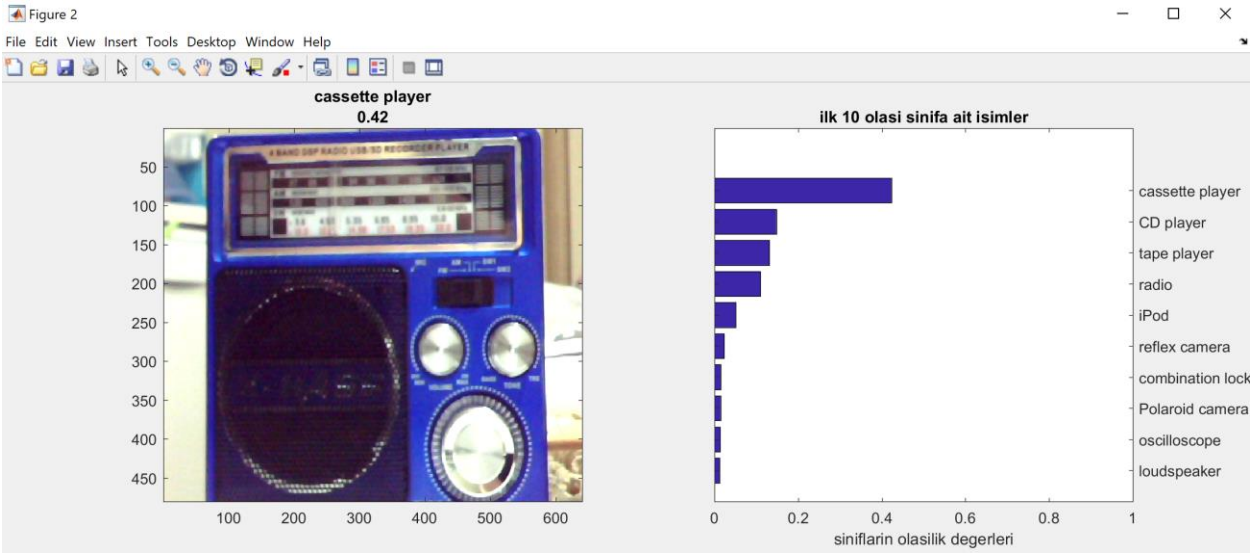


Figure 2
File Edit View Insert Tools Desktop Window Help
cassette player
0.42
ilk 10 olasi sinifa ait isimler
cassette player
CD player
tape player
radio
iPod
reflex camera
combination lock
Polaroid camera
oscilloscope
loudspeaker
siniflarin olasilik degerleri

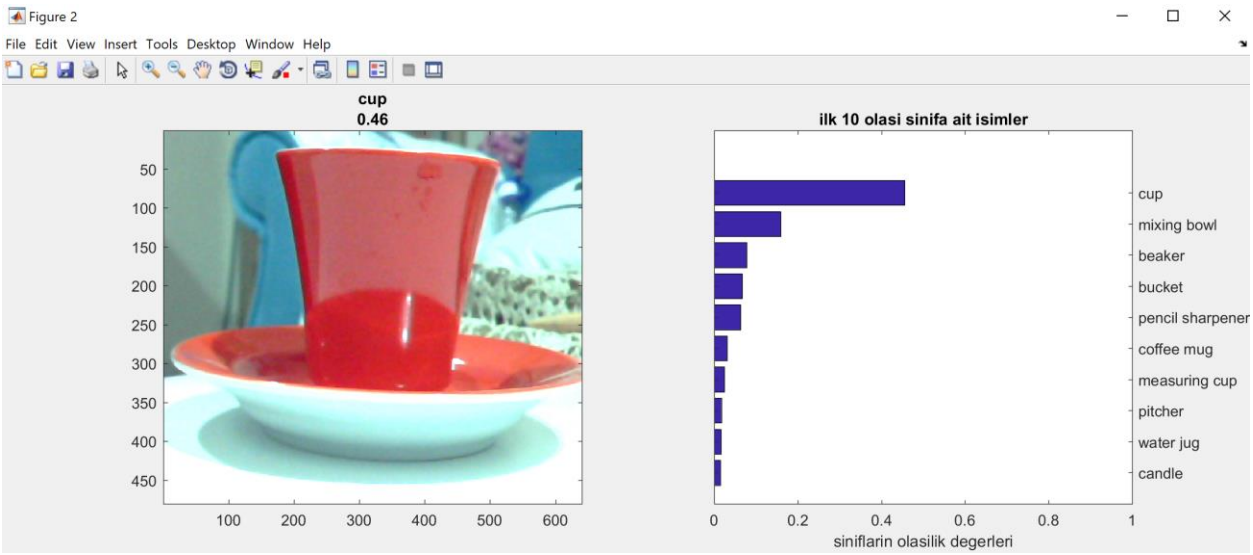


Figure 2
File Edit View Insert Tools Desktop Window Help
cup
0.46
ilk 10 olasi sinifa ait isimler
cup
mixing bowl
beaker
bucket
pencil sharpener
coffee mug
measuring cup
pitcher
water jug
candle
siniflarin olasilik degerleri

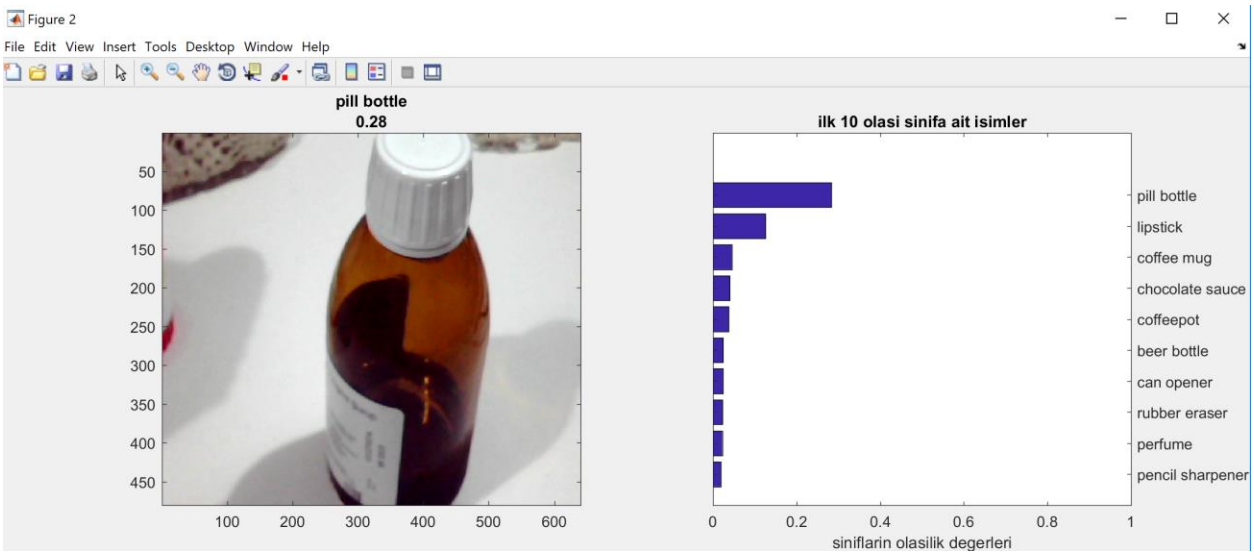


Figure 2
File Edit View Insert Tools Desktop Window Help
pill bottle
0.28
ilk 10 olasi sinifa ait isimler
pill bottle
lipstick
coffee mug
chocolate sauce
coffeepot
beer bottle
can opener
rubber eraser
perfume
pencil sharpener
siniflarin olasilik degerleri

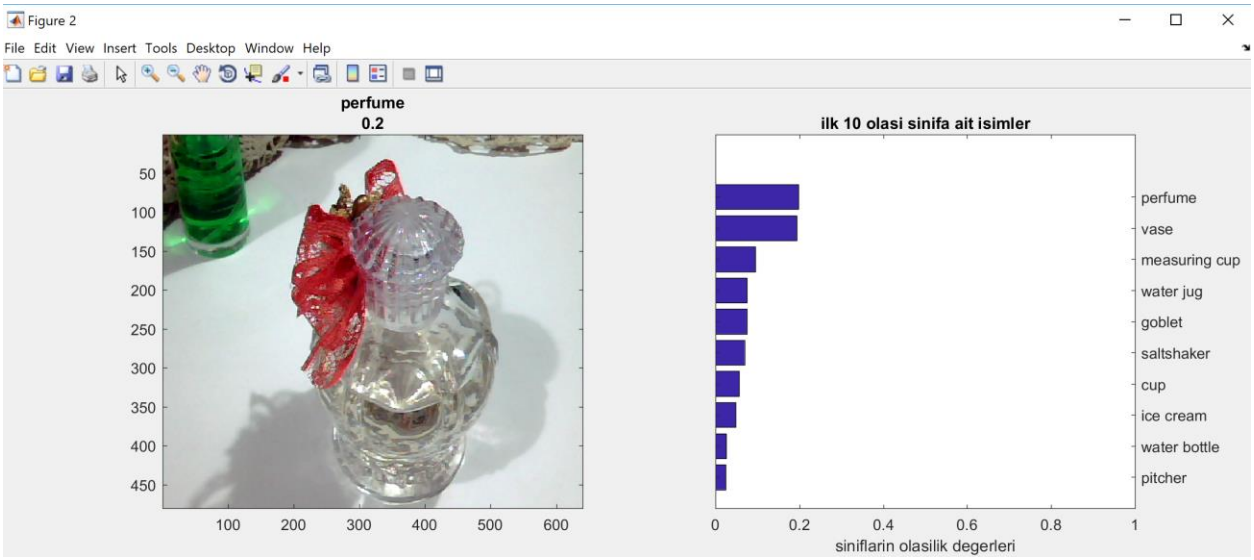


Figure 2
File Edit View Insert Tools Desktop Window Help
perfume
0.2
ilk 10 olasi sinifa ait isimler
perfume
vase
measuring cup
water jug
goblet
saltshaker
cup
ice cream
water bottle
pitcher
siniflarin olasilik degerleri

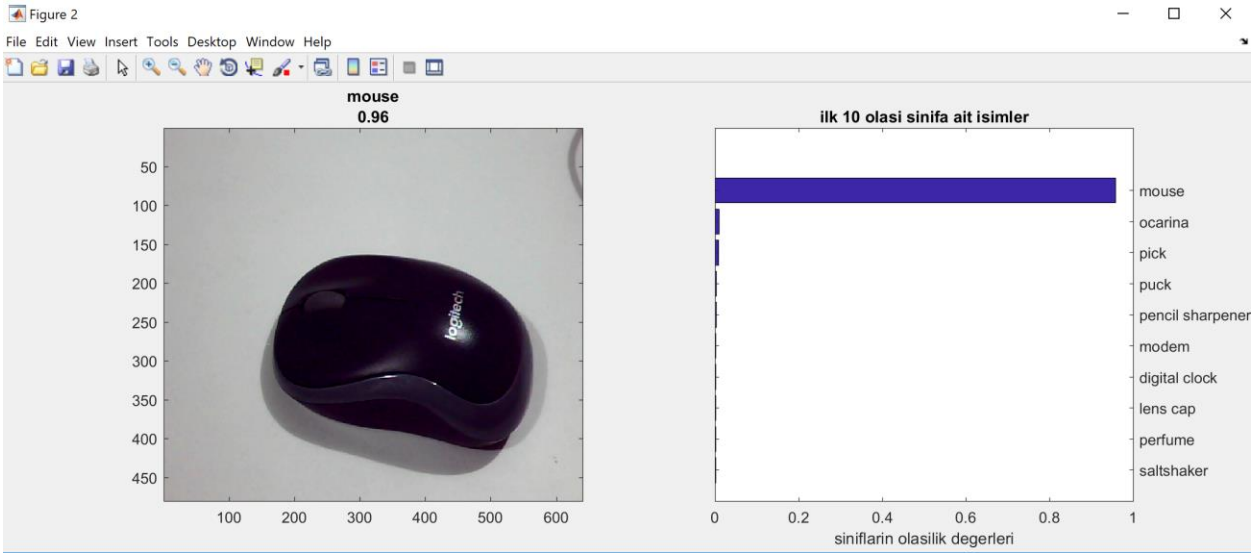


Figure 2
File Edit View Insert Tools Desktop Window Help
mouse
0.96
ilk 10 olasi sinifa ait isimler
mouse
ocarina
pick
puck
pencil sharpener
modem
digital clock
lens cap
perfume
saltshaker
siniflarin olasilik degerleri

2.4. PROJEYE AİT MATBAL KODLARI :

```
SurekliEkranGoruntusuYakala = true;
set(gcf,'CloseRequestFcn','SurekliEkranGoruntusuYakala = false; closereq');

YatayDegeri = figure;
YatayDegeri.Position(3) = 2*YatayDegeri.Position(3);
birinciEkran = subplot(1,2,1);
ikinciEkran = subplot(1,2,2);

KameraBaglantisiSagla = webcam;
NeuralNetworkDataSetYukle = alexnet;

while true
    YakalananGoruntu = snapshot(KameraBaglantisiSagla);
    image(birinciEkran,YakalananGoruntu);

    YakalananGoruntu = imresize(YakalananGoruntu,[227 227]);
    [EkranEtiketi,GoruntuDegeri] =
classify(NeuralNetworkDataSetYukle,YakalananGoruntu);
    title(birinciEkran,{char(EkranEtiketi),num2str(max(GoruntuDegeri),2)});

    [~,KacAdetOlasilik] = sort(GoruntuDegeri,'descend');
    KacAdetOlasilik = KacAdetOlasilik(10:-1:1);
    SinifIsimleri = NeuralNetworkDataSetYukle.Layers(end).ClassNames;
    SecilenSinifIsmi = SinifIsimleri(KacAdetOlasilik);
    SecilenSinifaAitOlasilikDegeri = GoruntuDegeri(KacAdetOlasilik);

    barh(ikinciEkran,SecilenSinifaAitOlasilikDegeri);
    title(ikinciEkran,'ilk 10 olasi sinifa ait isimler');
    xlabel(ikinciEkran,'siniflarin olasilik degerleri');
    xlim(ikinciEkran,[0 1]);

    yticklabels(ikinciEkran,SecilenSinifIsmi)
    ikinciEkran.YAxisLocation = 'right';

    drawnow
end
```

Neural Network'e ait matlab dosyasının içeriği :

	1
1	*1x1 ImageInputLayer*
2	*1x1 Convolution2DLayer*
3	*1x1 ReLULayer*
4	*1x1 CrossChannelNormalizationLayer*
5	*1x1 MaxPooling2DLayer*
6	*1x1 Convolution2DLayer*
7	*1x1 ReLULayer*
8	*1x1 CrossChannelNormalizationLayer*
9	*1x1 MaxPooling2DLayer*
10	*1x1 Convolution2DLayer*
11	*1x1 ReLULayer*
12	*1x1 Convolution2DLayer*
13	*1x1 ReLULayer*
14	*1x1 Convolution2DLayer*
15	*1x1 ReLULayer*
16	*1x1 MaxPooling2DLayer*
17	*1x1 FullyConnectedLayer*
18	*1x1 ReLULayer*
19	*1x1 DropoutLayer*
20	*1x1 FullyConnectedLayer*
21	*1x1 ReLULayer*
22	*1x1 DropoutLayer*
23	*1x1 FullyConnectedLayer*
24	*1x1 SoftmaxLayer*
25	*1x1 ClassificationOutputLayer*

	1
1	tench
2	goldfish
3	great white shark
4	tiger shark
5	hammerhead
6	electric ray
7	stingray
8	cock
9	hen
10	ostrich
11	brambling
12	goldfinch
13	house finch
14	junco
15	indigo bunting
16	robin
17	bulbul
18	jay
19	magpie
20	chickadee
21	water ouzel
22	kite
23	bald eagle
24	vulture
25	great grey owl
26	European fire salamander
27	common newt