Makine Öğrenmesi ile Elyazısı Rakam Tanıma

Semih Uzun 18290064

Ömer Yıldız 18290735

Ocak, 2022

ÖZET

Günümüz dünyasında yapay zeka ve makine öğrenmesi büyük bir öneme sahiptir. Gün geçtikçe bu alanlardaki gelişmeler hızla artmaktadır. Bilgisayarların işlem kapasitesinin artması ve donanım maliyetlerinin düşmesiyle birlikte makine öğrenmesi çalışmaları hız kazanarak büyümeye devam etmektedir. Ayrıca makine öğrenmesinin bir alt alanı olan derin öğrenme görüntü tanıma, görüntü sınıflandırma ve ses tanıma gibi problemlerin çözümü için önemli bir araştırma konusu olmuştur.

Çalışmamızda derin öğrenme Evrişimsel Sinir Ağı (Convolutional Neural Network-CNN) algoritmasını ve MNIST veri setini kullanarak el yazısı ile rakam tanıma problemi üzerine çalışıldı. MNIST veri seti kullanılarak bir evrişimsel sinir ağı modeli oluşturuldu. Oluşturulan model ile test verisi sınandı. Doğruluk (accuracy) ve kayıp (loss) değerleri görselleştirilerek modelin başarısı gözlemlendi.

İÇİNDEKİLER

1. GİRİŞ

Derin öğrenme, makine öğrenmesinin ve dolayısıyla yapay zekanın bir alt alanıdır. Temelde birden çok katmana sahip olan bir yapay sinir ağıdır ve öğrenme işlemi örnekler üzerinden çalışır. Sinir ağına örnek veriler vererek modelin bu veriler işlemesini sağlanır. Veriler işlendikten sonra test için ayırılan veriler ile doğruluğu kontrol edilir ve daha sonra verdiğimiz veriden farklı olan programın bilmediği yeni veriler verilir. Modelin bu veriler üzerinde tahmin yapmasını isteriz. Bu sebeple bir evrişimsel sinir ağı oluştururken öncelikle verimizi iki parçaya bölmeliyiz. Bu parçalardan biri modeli eğitmek için verdiğimiz örneklerden oluşurken diğer parça ise modeli eğittikten sonra modelin başarısını görmemiz için gereken test verisidir. Yani modelin eğitimi bittikten sonra test verisinin özelliklerini modele göstererek bizim daha önceden bildiğimiz sonuçlarla aynı tahminlerde bulunup bulunmadığı gözlenir. Bunun sonucunda modelin başarısı hesaplanır.

2. PROJE VE KULLANILAN ARAÇLAR

Bu bölümde makine öğrenmesi ile el yazısı tanıma projesinde kullanılan programlama dili, projede kullanılacak kütüphaneler ve programın derleneceği yazılım hakkında bilgi verilmektedir.

2.1. Python, Python Kütüphaneleri ve Jupyter Notebook

Programlama dili olarak python seçildi. Python, öğrenilmesi kolay, makine öğrenmesi alanında çok fazla kullanılan ve popüler bir programlama dilidir. Modüler bir yapıya sahip olduğundan yeni teknolojilerin içine dahil edilmesi kolay bir dildir. Tüm işletim sistemlerinde çalışabilen açık kaynaklı bir yazılımdır.

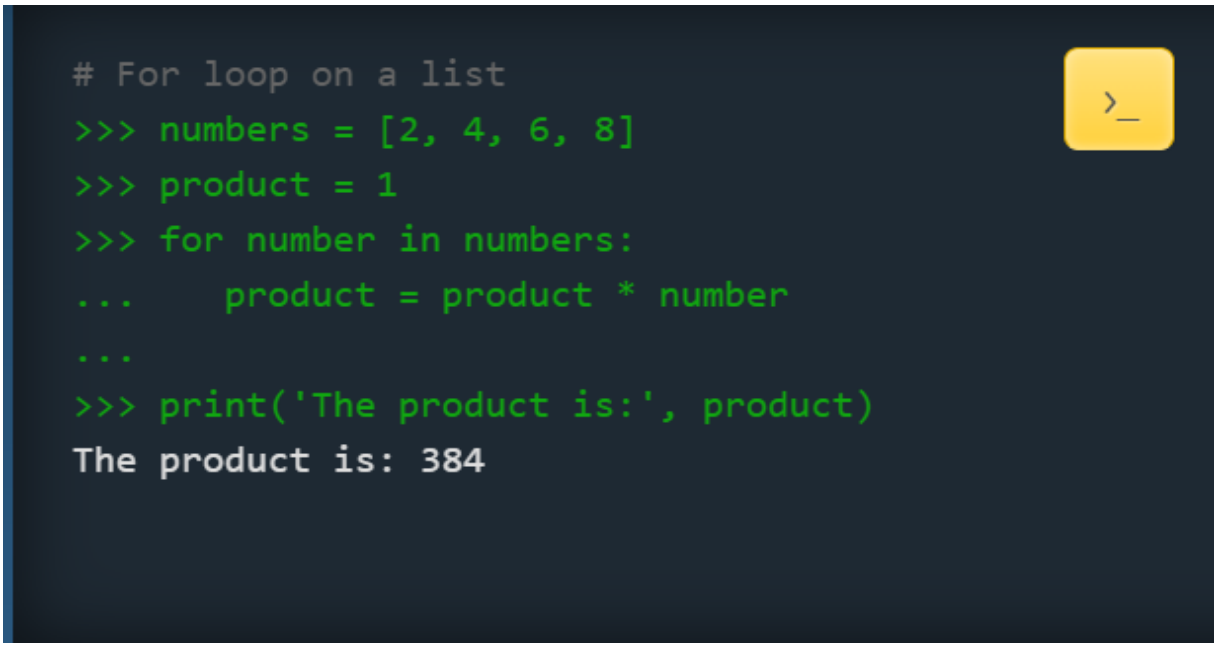
Makine öğrenmesinde bize yardımcı olacak python kütüphaneler ise numpy, pandas, matplotlib, tensorflow, keras ve opencv olarak belirlendi.

2.1.1. Python

Python, 1980'lerin sonuna doğru ABC diline alternatif oluşturmak amacıyla yazıldı. Python 'un ikinci sürümü (2.0), 2000 yılında piyasaya sürüldü. 2008 yılına kadar 2 sürümü kullanılmaya devam edildi. 2008 yılında 3.0 sürümü yayınlandı. Bu sürüm ikinci sürümden farklı olarak tasarlandığından dolayı kodlar birbiriyle uyum içinde çalışamamaktadır. Günümüzde 3.0 sürümü kullanılmaktadır.

Python, nesne tabanlı, yorumlanabilen, modüler ve etkileşime sahip olan yüksek seviyeli bir programlama dilidir. Ayrıca boşluğa duyarlı ve söz dizimi basit bir dildir. Bu da dilin öğrenilmesini kolaylaştırmakla birlikte öğrenilen bilginin zihinde kalmasına fayda sağlar. Bununla birlikte değişken tipini değiştirmeden veri atama işlemi gerçekleştirilir. Programlama dili işimizi basitleştirmekle birlikte bize zaman ve hız kazandırır. Zaman kaybı azaldığı için proje geliştirme sürecine pozitif yönde etki ederek bize yeni özellikler için vakit kazandırır.

Modüler bir yapıya sahip olan bu dil, her türlü alan ve sistemi destekler. Windows, Macos, Linux, Unix, Ios ve Android ekosistemi gibi farklı sistemlerde çalışabilmektedir. Bu işletim sistemleri için sistem programları, kullanıcı programları, mobil uygulama ve veri tabanı gibi birbirinden farklı uygulamalar geliştirilebilir.



Şekil 2.1.1. Python kodu örneği

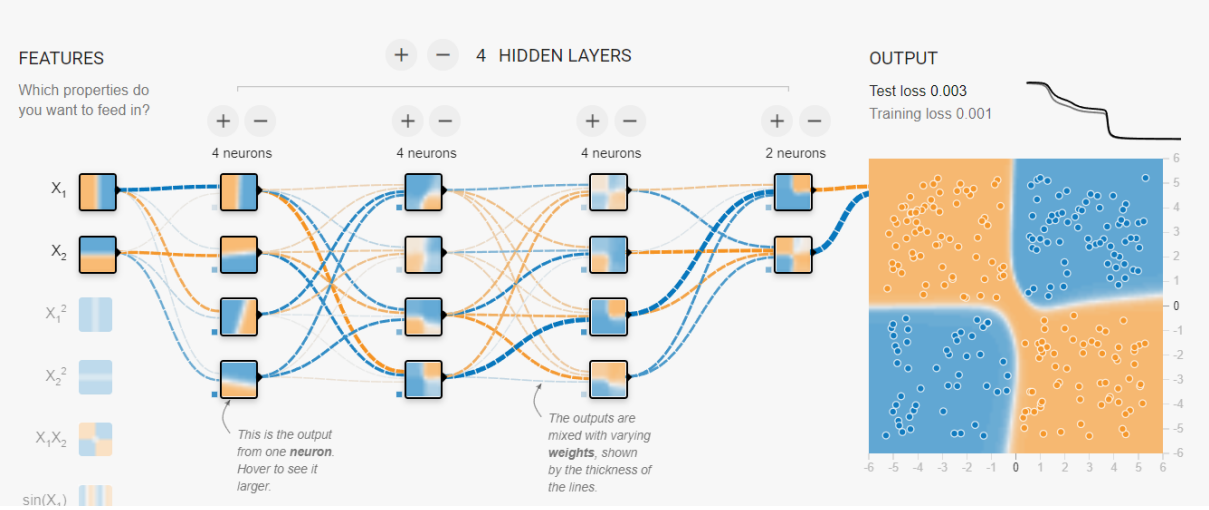
Şekil 2.1.1. de python ile yazılmış kod örneğini görebilirsiniz.

2.1.2. Jupyter Notebook

Makine öğrenmesi ile el yazısı programı geliştirilirken kodların derlenmesi ve çalıştırılması için jupyter notebook kullanıldı. Jupyter notebook kodu hücrelere ayırarak koda yeni parçalar eklendiğinde kodun en baştan derlenmesine gerek kalmadan sadece yeni eklenen kısımların çalıştırılabilmesine imkân tanır. Buda modeli oluşturduğumuz kodda modelin eğitilmesi uzun sürdüğü için o kısmı tekrar tekrar çalıştırmaya gerek kalmadan kod üstünde çalışmamıza imkân tanıdı. Bu sayede kod üstünde çalışırken bize büyük zaman kazandırdı.

2.2. Evrişimsel Sinir Ağları (Convolutional Neural Network-CNN)

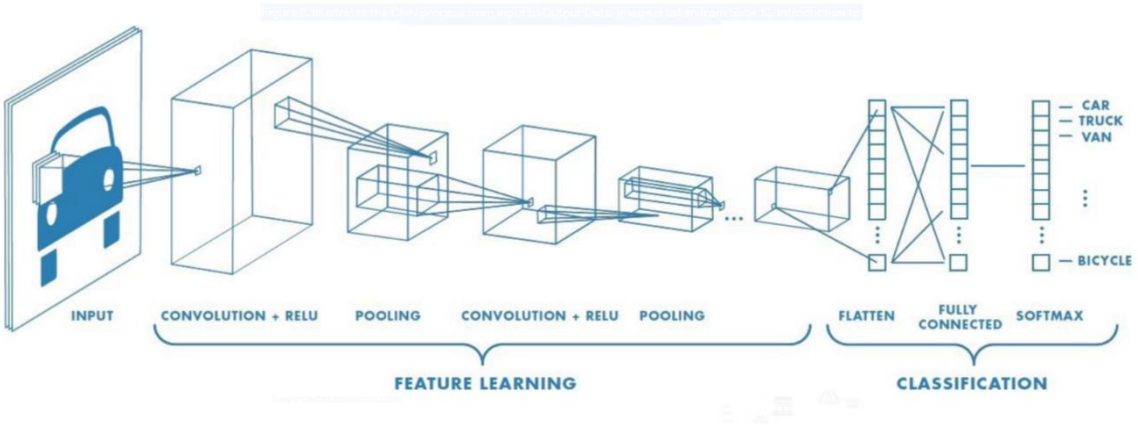
Sinir ağı, insan beyninin çalışma şeklini bilgisayar ile tanımaya çalışan bir algoritmadır. Verilen eğitim veri setleri üzerinde analiz yaparak bir çıkarsımda bulunur. Bunu yaparken insan beynindeki sinirleri taklit eden yapay sinir ağlarını kullanır. Yapay sinir ağlarında nöronlar bulunur. Verilen veriler bu nöronlarda işlenir ve bir çıktı elde edilir. Şekil 2.2.1. üç gizli katmana sahip ve nesneleri iki ayrı sınıfa ayırabilen bir sinir ağı modeli görebilirsiniz.



Şekil 2.2.1. Sinir Ağı Modeli

Evrişimsel sinir ağı, derin öğrenme de yüksek performans gösterdiği için oldukça popüler bir algoritmadır. Görüntüler üzerinde işlem yapabilen bir derin öğrenme algoritmasıdır. Asıl amacı görüntüleri pikseller üzerinde işleyerek, görüntülerin sınıflandırılmasına yardımcı olmaktır. Evrişimsel sinir ağı temel olarak dört katmandan oluşur. Bunlar girdi katmanı, evrişimsel katman, havuzlama katmanı ve sınıflandırma katmanıdır.

Girdi katmanında veri setindeki resimler özellikleri alınır ve matrislere dönüştürülür. Evrişimsel katmanda resimlerin filtre sayısı, boyutu ve piksel sayısı gibi özellikleri belirlenir. Evrişimsel katmandan sonra havuzlama işlemi yapılır. Havuzlamada belirlediğimiz filtreleri pikseller üzerinde gezdirerek daha küçük boyutlu matrisleri oluşturur. Son olarak da aktivasyon fonksiyonumuzla nesneni sınıfını belirleriz. Şekil 2.2.2. de tüm bu aşamaları gösterilen CNN algoritması gösterilmiştir.



Şekil 2.2.2. CNN girdi ve çıktısı.

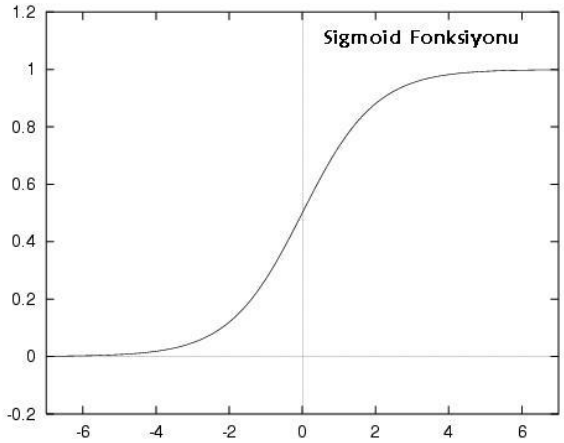
2.3. Aktivasyon Fonksiyonları

Aktivasyon fonksiyonları; sinir ağında bulunan katmanlardaki nöronların çıktı değerlerini sonraki katmana aktarmada kullanılır. Bu çıktı değerinin, diğer katmanlara iletilip iletilmeyeceğine karar verebilmek için bir eşik değerinin belirlenmesi gerekir. Çünkü bir yapay sinir hücresindeki bilginin değeri (−∞, +∞) aralığında olabilir ve nöron gerçek değerin sınırlarını bilmeyebilir. Bu nedenle, nöronun aktif olup-olmaması gerektiğine karar verebilmesi için aktivasyon fonksiyonlarına ihtiyaç duyulur. Böylece bir nöron tarafından üretilen çıktı değerini kontrol edebilecek ve dış bağlantıların nöronu aktif olarak görüp görmeyeceğine karar verilebilecektir. Yapay sinir ağları daha çok doğrusal olmayan sınıflandırmalarda kullanıldığından, aktivasyon fonksiyonu genellikle doğrusal olmayan bir fonksiyon seçilir. Mimarinin öğrenme sürecinde geri yayılım algoritması kullanılırken, aktivasyon fonksiyonunun türevi de kullanıldığı için türevi kolay hesaplanabilen bir aktivasyon fonksiyonunun kullanılması algoritmanın hızı için önemlidir.[1]

En yaygın kullanılan aktivasyon fonksiyonları Sigmoid, ReLU ve Softmax fonksiyonlarıdır.

2.3.1. Sigmoid Fonksiyonu

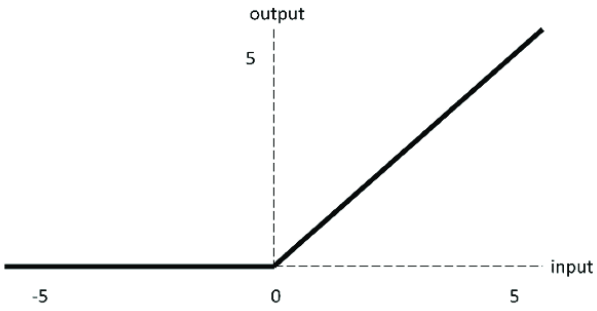
0 ile 1 arasında düzgün bir değer aralığı oluşturan sigmoid dönüşümü, pürüzsüz eğrilere sahiptir. Girdi değerlerindeki küçük değişikliklerin çıktı değerlerinde gözlemlenmesine imkan tanıyan pürüzsüz eğriler, adım fonksiyonlarında tercih edilir.



Şekil 2.3.1. Sigmoid fonksiyonu grafiği

2.3.1. ReLU Fonksiyonu

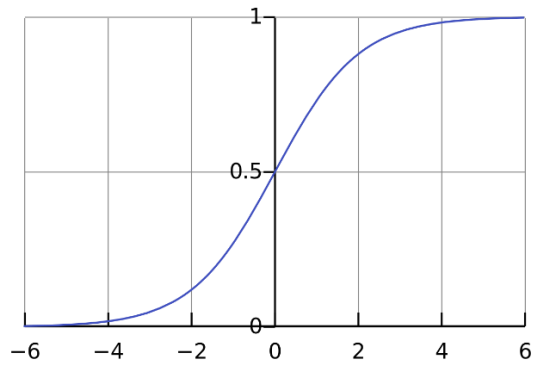
Bu fonksiyonun kullanılmasının en büyük nedeni 0'dan büyük tüm girdiler için ağın daha hızlı eğitilmesini sağlayan sabit türev değerine sahip olmasıdır.



Şekil 2.3.2. ReLU fonksiyonu grafiği

2.3.1. Softmax Fonksiyonu

Bu fonksiyon genellikle çıktı katmanında kullanılmaktadır. Sigmoid fonksiyonuna benzer özellikleri vardır. Sigmoidden farkı çıktıların 0-1 arasında bir sayıya normalize edilmesidir. Çok katmanlı sınıflandırma problemlerinde kullanılır.



Şekil 2.3.3. Softmax fonksiyonu grafiği

2.4. Kayıp Fonksiyonu(Loss) ve Optimizasyon Yöntemleri

Kayıp fonksiyonu, modelimizin yaptığı tahminlerle gerçek değerler arasındaki farkı hesaplar. Bu fark başlarda yüksekken modelin eğitilme süresince azalması beklenir. Optimizasyon fonksiyonları bu farkın azalmasına yardım eder. Bu fonksiyon aynı zamanda modelimizin başarısını ölçer.

Optimizasyon yöntemleri, modelin verdiği tahmin ile gerçek veri arasındaki farkı minimuma indirmeye yarar. Biz modelimizde bu farkın olabildiğince az olmasını isteriz. Optimizasyon fonksiyonları veri setimizin türüne ve büyüklüğüne göre seçmemiz gereken ve modelin tahmin ettiği verilerle gerçek veriler arasındaki farkı minimumda tutmaya yarayan fonksiyonlardır. Günümüzde en çok gradyan inişi (Gradyan Descent) yöntemi kullanılmaktadır. En popüler gradyan iniş fonksiyonları “rmsprop” ve “adam” dır.

2.5. Proje Detayları

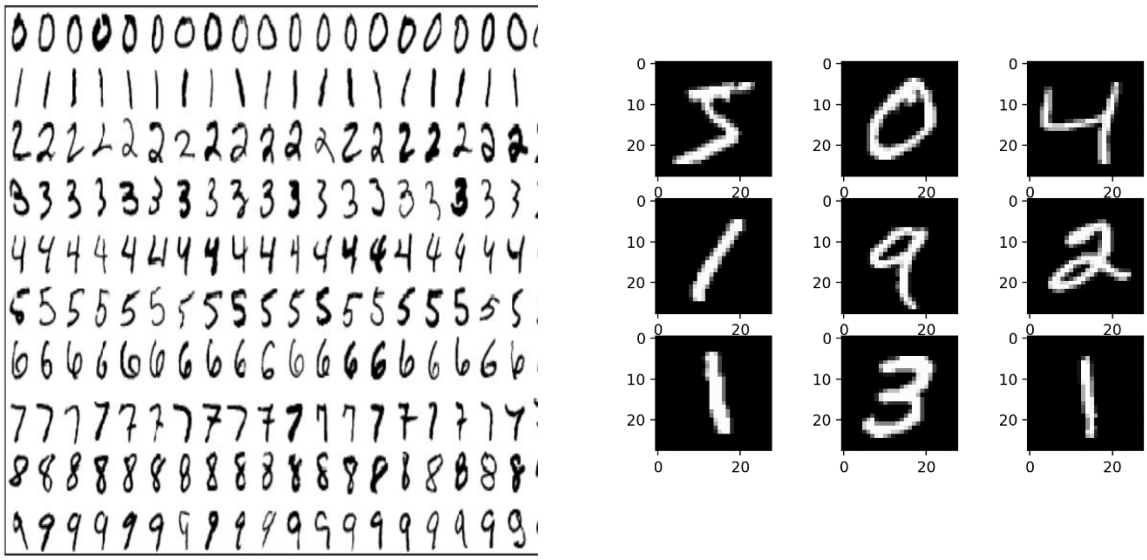
Projemizin geliştirilme sürecinde birçok kütüphane kullanılmıştır. Bu kütüphanelerin bir kısmı şekil 2.5. te gösterilmiştir. Raporumuzun bu kısmında kullandığımız kütüphaneleri daha detaylı anlatmaktayız.

```
import numpy as np
from keras.datasets import mnist
import tensorflow as tf
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import array_to_img
from sklearn.metrics import confusion_matrix
```

Şekil 2.5. Kütüphanelerin Tanımlanması

2.5.1. Mnist Veri Seti

Mnist, makine öğrenmesi için hazırlanmış bir veri setidir. Veri setinde sıfır ile dokuz arasındaki rakamlar bulunur. Toplam 70 000 adet el yazısını içeren, siyah beyaz formatta resimden oluşur. Bu resimlerin boyutu 28x28 pikseldir. Genel olarak bu veri setinin 60 000 tanesi eğitim için, geri kalan 10 000 tanesi ise test için kullanılır. Veri setinin içinde çizilmiş bazı rakamlar şekil 2.5.1. in solunda, 28x28 piksele sığdırılmış görüntülerden oluşan rakamlar ise şeklin sağında gösterilmiştir



Şekil 2.5.1. Mnist Veri Seti Örnekleri

2.5.2. Veri Setinin Eklenmesi ve Eğitime Hazırlanması

Model oluşturulmadan önce ilk önce projeye veri setini eklemek gerekir. Mnist veri seti pythonda tanımlı bir veri setidir. Projede bu veriyi setini kullanmak için Mnisti keras kütüphanesinden import etmek yeterlidir. Bu kütüphaneden veri setimizdeki resimler numpy arrayine dönüştürülmüş halde çekilebilir. 28x28 piksel boyutunda olan resimlerimiz 28x28 numpy arrayine dönüştürülür. Bu dönüşüm renk yoğunluğuna göre arrayimizin değerlerini 0-255 arasındaki sayılara dönüştürür. Şekil 2.5.2 de görülen kısımda veri setimizi çağırmayı, modelin eğitilmeden önce istenilen şekle ve işlem kolaylığı için veri setimizin 255'e bölündüğünü görmekteyiz.

```
(X_train, y_train) , (X_test, y_test) =mnist.load_data() #Mnist verilerini çekme
#  VERİLERİ İŞLEM KOLAYLIĞI İÇİN 0-1 NORMALLEŞTİRME VE RESHAPE YAPMA
X_train = X_train / 255
X_test = X_test / 255
X_train = X_train.reshape(-1,28,28,1)     #training set
X_test = X_test.reshape(-1,28,28,1)       #test set
```

Şekil 2.5.2 Verilerin yüklenmesi ve Hazırlanması

2.5.3. Modelin ve Katmanların Oluşturulması

Veri setimiz hazırlandıktan sonra yapılacak ilk iş boş bir model oluşturmak şekil 2.5.3.te oluşturulan CNN Model değişkeni kerasın Sequential metoduyla boş bir modele

eşitlenir. Model oluşturulduktan sonra ise katmanların oluşturulması gereklidir. Bu kısım modelin en önemli kısmıdır kullanılan veri setine, veri setinin büyüklüğüne, veri setinin türüne vs. gibi parametrelere göre katmanların değiştirilmesi gerekir. Bizim verimiz resimlerden oluştuğu için daha çok konvolüsyon ve pooling katmanları kullandık. Çıkış katmanı olarak 10 rakam için 10 tane hücre ekledik ve her bir rakam için bir olasılık hesaplanması için softmax aktivasyon fonksiyonunu kullandık.

```
#MODEL OLUŞTURMA
from tensorflow.keras import models
from tensorflow.keras import layers
cnnModel = models.Sequential()
```

```
#KATMANLAR
cnnModel.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28,28,1)))
cnnModel.add(tf.keras.layers.MaxPooling2D((2, 2)))
cnnModel.add(tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
cnnModel.add(tf.keras.layers.MaxPooling2D((2, 2)))
cnnModel.add(tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
cnnModel.add(tf.keras.layers.MaxPooling2D((2, 2)))
cnnModel.add(tf.keras.layers.Flatten())
cnnModel.add(tf.keras.layers.Dense(32, activation='relu'))
cnnModel.add(tf.keras.layers.Dense(10, activation='softmax'))
```

Şekil 2.5.3. Model ve Katmanların Oluşturulması

2.5.4. Modelin Eğitilmesi ve Eğitim Sonuçları

Modelin eğitimine başlamadan önce kullanılacak optimize ve kayıp fonksiyonları belirlenmelidir. Biz Modelimizde optimizasyon fonksiyonu olarak en çok kullanılan fonksiyonlardan olan rmsprop fonksiyonunu kullandık. Kayıp fonksiyonu olarak ise 2'den fazla olan sınıflandırma problemlerinde sıkça kullanılan crossentropy fonksiyonunu kullandık. Bu yöntemleri şekil 2.5.4. te görülen compile metodunun içinde parametre olarak belirtiyoruz. Bu işlemi bitirdikten sonra ise modeli eğitmeye başlayabiliriz. Yine şekil 2.5.4. te görülen fit metodunun içine eğitim için ayrılmış verilerimizin özellikleri olan X_train değişkeniyle bu özelliklerin sonucunda bulması tahmin etmesi gereken y_train değişkenlerini ekliyoruz. Epoch değeri ise modelimizin veri setimizin üzerinden kaç defa geçeceğini belirtir. Aşırı besleme olayının olmaması için epoch değerinin çok yüksek verilmemesi gerekir. Aşırı besleme durumunda model eğitim için ayrılan veriler üzerinde iyi tahminde bulunurken yeni veriler üzerinde yanlış tahminde bulunma olasılığı fazladır.

Eğitim tamamlandıktan sonra şekil 2.5.5. te görülen kısımda evaluate metoduyla modelimizin test ve train verileri üzerindeki kayıp ve doğruluk oranlarını görmekteyiz.

Genellikle test ve train verilerinin doğruluk ve kayıp oranlarının birbirine yakın çıkması beklenmektedir.

```
#MODELIN DERLENMESI VE EGITILMESI
cnnModel.compile(optimizer = 'rmsprop', loss= 'sparse_categorical_crossentropy',metrics=['accuracy'])
cnnModel.fit(X_train,y_train,epochs=2)
```

```
Epoch 1/2
1875/1875 [==============================] - 20s 10ms/step - loss: 0.2559 - accuracy: 0.9198
Epoch 2/2
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0777 - accuracy: 0.9765
```

Şekil 2.5.4. Modelin Compile Edilmesi ve Eğitime Başlanması

```
#MODEL TEST VE TRAİN SONUÇLARI
print("Model train data sonuçları Sonuçları")
cnnModel.evaluate(X_train, y_train)
print("Model test data sonuçları Sonuçları")
cnnModel.evaluate(X_test, y_test)
```

```
Model train data sonuçları Sonuçları
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0653 - accuracy: 0.9800
Model test data sonuçları Sonuçları
313/313 [==============================] - 1s 4ms/step - loss: 0.0707 - accuracy: 0.9772:
```

Şekil 2.5.5. Modelin Eğitim Sonuçları

2.5.5. Modelin Test verileri üzerindeki tahminleri

Daha önce modelimizden bahsederken çıkış katmanında 10 tane hücremizin olduğundan bahsetmiştik. Şekil 2.5.6. daki ilk çıktıda modelimizin ilk test verimiz üzerindeki tahmin sonuçlarını görmekteyiz. Model her bir rakam için bir olasılık değeri döndürür. Biz ise argmax metodunu kullanarak olasılığı en yüksek olan değerin indexini alırız bu ise yapılan tahminimiz olarak varsayarız. Şekil 2.5.6. da ilk test verimizin nasıl göründüğü ve modelimizin üstünde yaptığı tahmini görmekteyiz.

Şekil 2.5.7. de görülen kısımda test verilerimiz üzerindeki tüm tahminlerin tam sayı karşılıklarını değişkenimize atıyoruz. Daha sonra ise Şekil 2.5.8. te görüllen kısımdaki confusion_matrix isimli metoda bu değişkenimizi ve test verilerinin gerçek değerlerini vererek confusion matrixi oluşturuyoruz. Bu matrixin asal köşegeni üzerindeki kısımlar sırasıyla 0-9'a kadar olan sayıların kaçının doğru bilindiğidir. Diğer kısımlar ise o sütundaki sayının hangi sayı ile karıştırıldığı hakkında bize bilgi verir.

```
testTahminleri = cnnModel.predict(X_test)
print(testTahminleri[0])
print(F"TAHMİN SONUCU:{np.argmax(testTahminleri[0])}")
array_to_img(X_test[0])
```

```
[1.4886750e-06 2.1355655e-07 3.3572374e-05 1.0376549e-05 7.2752286e-08
 6.4442916e-06 5.3565100e-13 9.9988306e-01 6.4580966e-08 6.4681590e-05]
TAHMİN SONUCU:7
```

Şekil 2.5.6. Test Verilerinin Model Üzerinde Test Edilmesi

```
#TEST TAHMİNLERİNİ INTE ÇEVİRME
testTahminleriInt=[]
for i in range(10000):
    testTahminleriInt.append(np.argmax(testTahminleri[i]))
testTahminleriInt=np.array(testTahminleriInt)
testTahminleriInt=testTahminleriInt.astype('uint8')
```

Şekil 2.5.7. Tahmin Sonuçlarının Int Değişkene Dönüşümü

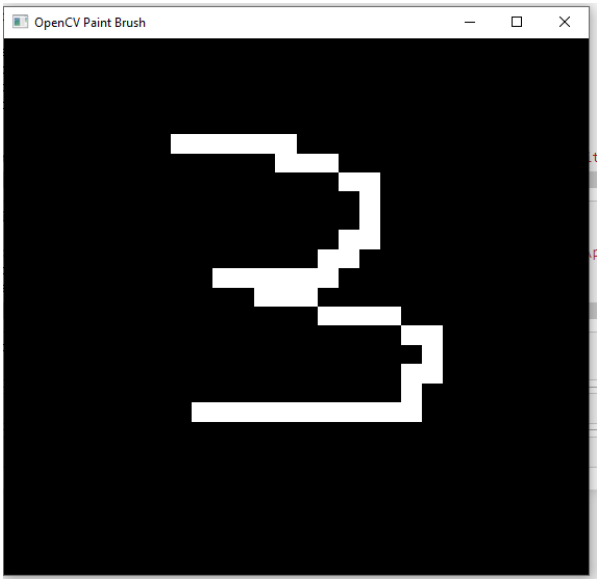
```
cm = confusion_matrix(testTahminleriInt,y_test)
print(cm)
```

```
[[ 971    0    3    0    1    4    3    0    1    4]
 [   1 1108    0    0    0    0    2    1    0    0]
 [   1    2 1000    2    0    0    0    9    1    0]
 [   1    2    3  988    0    4    1    1    2    0]
 [   1    2    4    1  956    0    1    0    3    1]
 [   0    4    0    9    0  870    5    1    1    1]
 [   1    1    1    0    3    1  944    0    0    0]
 [   3    3   14    4    0    1    0 1000    1    2]
 [   0    7    4    0    0    4    2    2  935    1]
 [   1    6    3    6   22    8    0   14   30 1000]]
```

Şekil 2.5.8. Confusion Matrix Gösterimi

2.5.6. OpenCv ile Girdi Alma

OpenCv kütüphanesi pythonda görsel işler yapmakta kullanılır. Şekil 2.5.9 da görülen pencere openCv ile oluşturuluyor ve bizim input girmemize olanak sağlıyor. Girilen inputu kaydederek modelimizin o input hakkında tahminde bulunmasını sağlayabiliyoruz.



Şekil 2.5.9. OpenCv Input Alma

2.5.7. Modele İnput Tahmini Yaptırma

Modele input vermeden önce inputu modelimizin anlayacağı şekle sokmamız lazım. Şekil 2.6.1 de görülen kısımda input dosyası çağrılarak önce inputumuz alınır. Daha sonra input resmimizi siyah beyaz forma ve 28x28 piksel boyutuna alırız img_to_array metoduylada resmimizi numpy arayine dönüştürerek model için uygun hale getiririz. Şekil 2.6.1 in son kısmında verdiğimiz inputun resim olarak görünümü ve modelimizin yaptığı tahmin görünmektedir.

```
exec(open("./drawing.py").read()) #çizim dosyasını çağırma
# image yükleme
Path="C:\\Users\\semih\\Desktop\\python\\Jupyter Notebook\\Bitirme Projesi\\img\\painted_image.png"
img = load_img(Path,color_mode='grayscale',target_size=(28,28))
img_array = img_to_array(img) # image to np.array
img_array=img_array.reshape(-1,28,28,1)
```

```
print(np.argmax(model.predict(img_array)))
img
```

```
3
```

Şekil 2.6.1. Modelin İnput Üzerindeki Tahmini

3. SONUÇ

Rapor boyunca kullandığımız programlama dillerinden o dile ait birçok kütüphaneden bahsettik. Daha sonra ise projemiz için gerekli olan evrişimsel sinir ağları modellerinden ve bu modele ait aktivasyon fonksiyonlarından, kayıp fonksiyonlarından ve optimizasyon yöntemlerinden bahsettik. Daha sonra ise bu bahsettiğimiz kısımları projemiz içinde kullandığımız kısımları göstermiş bulunmaktayız.

Projemizin bu kısmı evrişimsel sinir ağlarını ve derin öğrenme tekniklerini anlamamız için bize büyük yardımda bulundu. Projemizin sonraki aşamasında ise bu öğrendiklerimizi daha karmaşık problemleri ve farklı veri setleri üzerinde çalışmayı çözmek için kullanmayı hedeflemekteyiz.