

**Mekatronik Mühendisliği Bölümü**

GÖMÜLÜ SİSTEMLER

ve

UYGULAMALARI

VİZE PROJE ÖDEVİ

MEHMET AKSU – 171214011 FEDİ SALHİ – 170214925

**SIGN2WORD**

**İÇİNDEKİLER**

**Genel Bakış** 3

**DeepLearning ile Sınıflandırma (Image Classifier)** 4

**TensorFlow** 5

**CNN Modeli Oluşturma** 5

**CNN’yi Oluşturan Katmanlar** 7

**Giriş Katmanı (Input Layer)** 7

**Konvolüsyon Katmanı (Convulation Layer)** 7

**Düzleştirilmiş Doğrusal Birim Katmanı (Rectified Lineer Units Layer (ReLu))** 7

**Havuzlama Katmanı (Pooling Layer)** 8

**Tam Bağlantılı Katman (Fully Connected, Dense Layer)** 8

**DropOut Katmanı** 9

**Sınıflandırma Katmanı (Classification Layer)** 9

**OpenCV ile Görüntü İşleme** 9

**Donanım Alt Yapısı** 11

**Yazılım Alt Yapısı** 12

**DATA SET GENERATOR OLUŞTURULMASI** 12

**NETWORK BUILDER OLUŞTURULMASI** 25

**MyFirstCNN MODEL OLUŞTURULMASI** 33

**Modelin Test Dosyasının Oluşturulması** 43

**SONUÇ ve DEĞERLENDİRME** 48

**Programın Eksiklikleri** 48

**KAYNAKÇA** 49

**Projenin Github Adresi** 49

# **Genel Bakış**

Dünyamızda konuşma engeli bulunan insanlar birbirleriyle genellikle işaret dilini kullanarak iletişim kurarlar. Fakat işaret dili toplumda herkes tarafından bilinmediği için, konuşma engeli bulunan insanlar dertlerini anlatamadığı için kendilerini toplumdan yalnızlaştırabiliyorlar.

Biz ise bu problemi giderebilmek adına, konuşma engeli bulunan insanların toplumda ki diğer bireyler ile rahatça iletişime geçmelerini sağlamak amacıyla böylesi bir projeyle ilk adımı atmış oluyoruz.

Daha önceleri bu problemi gidermek amacıyla geliştirilen birçok yöntem yapılmış ve piyasada da hali hazırda bulunmakta olan uygulamaları vardır. Ancak bu uygulamaların daha çok giyilebilir bir ürün olması ve sürekli bunu kullanan kişinin üzerinde taşıma zorunluluğunun olmasının cazip olmadığını düşünerek bizler, dış ortamda ki bir kamera ile bu problemi gidermeye çalıştık.

Görüntü İşleme ve DeepLearning alanlarında yapılan çalışmaları inceleyerek bu uygulamaların bizim sorunumuzu giderebilmesi için kullanılabilirliği açısından araştırmalar yaptık.

Yaptığımız araştırmalardan yola çıkarak şu adımları belirleyerek ilerlemeye karar verdik.

* **OpenCV** kullanarak kamera ile aldığımız görüntüyü bir ekranda el alanı oluşturup sadece bu alan için **BGR2GRAY**, **GaussianBlur** ve son olarak **threshold** işlemlerini kullanarak **binary** görüntüye çevireceğiz.
* Kendi oluşturacağımız **CNN** (Convolutional Neural Networks) için **train** yapabilmek için gerekli **data setleri** hazırlayacağız.
* **TensorFlow** framework kullanarak **CNN** oluşturacağız.
* Oluşturacağımız **CNN**’ den faydalanarak kameradan aldığımız görüntüleri yazıya dönüştüreceğiz.

# **DeepLearning ile Sınıflandırma (Image Classifier)**

Derin öğrenme, geleneksel makine öğrenmesi ve görüntü işleme tekniklerinin aksine öğrenme işlemini ham veri (işlenmemiş veri) üzerinde yapmaktadır. Ham veriyi işlerken gerekli bilgiyi farklı katmanlarda oluşturmuş olduğu yapılarla elde etmektedir.

Derin Öğrenme modelleri kullanıcı tarafından belirlenen özellikler olmadan giriş verisinden öğrenme işlemini kendisi yapmaktadır. Bu öğrenme işlemini farklı katmanlarda veriye ait farklı özellikler keşfetmekle elde etmektedir.

Derin Öğrenme kavramına ait temel mimari **CNN** mimarisi kabul edilir. CNN’ler art arda yerleştirilmiş birden fazla eğitilebilir bölümlerden oluşur.

Derin ağların en üstün tarafı, bir problemin çözümünde sonsuz sayıda modellerin tasarlanmasına olanak sunmasıdır. Bu yüzden kısır bir yönü yoktur ve gelişime her zaman açıktır.

Bu yapıyı bünyesinde bir den fazla parametrenin ayarlanması zorunluluğundan dolayı elde etmektedir.

Bu parametrelerin bazıları, katman sayısı, konvolüsyon katmanı filtre boyutu ve sayısı, havuzlama katmanı filtre boyutu ve adım kayma sayısı, ağın öğrenme oranı örnek olarak verilebilir.

## **TensorFlow**

Tensorflow Google’ın geliştirdiği açık kaynak bir DeepLearning frameworküdür.

Tensorflow, data flow [graphları](https://mcemilg.github.io/general/2017/11/23/tensorflow/%22https:/en.wikipedia.org/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvR3JhcGhfdGhlb3J5%22) kullanarak numerik hesaplama için geliştirilmiş açık kaynak bir kütüphanedir. Graplardaki nodelar matematiksel işlemleri(operation) temsil eder, graph edgeleri(kenarları) birbirleri arasında iletilen çok boyutlu arrayleri(tensor) temsil eder. Esnek yapısı sayesinde, tek bir API ile platform fark etmeksizin hesaplamaları, bir veya birden fazla CPU, GPU kullanarak deploy etmenize olanak sağlar.

Ana fikir: Numerik hesaplamaları graph şeklinde ifade etmek.

* Graph nodeları, herhangi bir sayıda input ve output sahip işlemleri ifade eder.
* Graph edgeleri, nodelar arası akışı sağlayan tensorlardır.

Temel olarak Tensorflow N dimensional arrayler ile graph işlemlerinin otomatik olarak yapılabildiği bir kütüphanedir.

## **CNN Modeli Oluşturma**

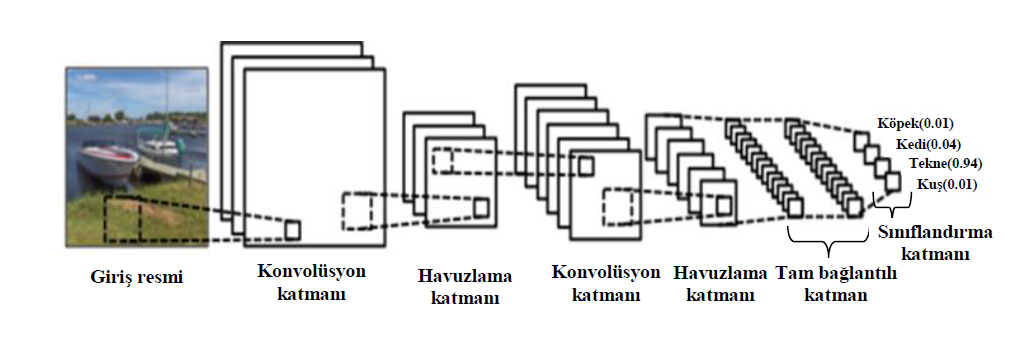
Derin Öğrenme kavramına ait temel mimari **CNN** mimarisi kabul edilir.

Bu mimariye göre ilk birkaç aşama Konvolüsyon (Convulation) ve Havuzlama (Pooling) katmanlarından oluşur.

Son aşama ise Tam Bağlı katmandan (Dense) oluşur ve akabinde Sınıflandırma katmanı mevcuttur.

Özetle CNN’ler art arda yerleştirilmiş birden fazla eğitilebilir bölümlerden oluşur.

* CNN’de giriş verilerini aldıktan sonra katman katman işlemler yapılarak eğitim süreci gerçekleştirilir.
* En sonunda doğru sonuç ile karşılaştırma yapmak için bir final çıktısı verir.
* Üretilen sonuç ile istenen sonucun farkı kadar bir hata oluşur.
* Bu hatanın bütün ağırlıklara aktarılması için geriye yayılım algoritması kullanılır.
* Her bir iterasyonla ağırlıkların güncellenmesi yapılarak hatanın azaltılması sağlanır.

**Resim 1:** CNN ağının Genel Mimarisi

### **CNN’yi Oluşturan Katmanlar**

#### **Giriş Katmanı (Input Layer)**

Bu katman isminden de anlaşılacağı üzere CNN’in ilk katmanını oluşturmaktadır. Bu katmanda veri ham olarak ağa verilmektedir. Tasarlanacak modelin başarımı için bu katmandaki verinin boyutu önem kazanmaktadır.

#### **Konvolüsyon Katmanı (Convulation Layer)**

CNN’nin temelini oluşturan bu katman dönüşüm katmanı olarakta bilinir. Bu dönüşüm işlemi belirli bir filtrenin tüm görüntü üzerinde dolaştırılması işlemine dayanmaktadır. Bu sebeple filtreler katmanlı mimarinin ayrılmaz bir bileşenidir.

Filtreler 2x2, 3x3, 5x5 gibi farklı boyutlarda olabilir. Filtreler, bir önceki katmandan gelen görüntülere konvolüsyon işlemini uygulayarak çıkış verisini oluştururlar. Bu konvolüsyon işlemi sonucu aktivasyon haritası (Özellik haritası) oluşur.

Aktivasyon haritası, her bir filtreye özgü özelliklerin keşfedildiği bölgelerdir. CNN’lerin eğitimi esnasında bu filtrelerin katsayıları, eğitim kümesindeki her öğrenme yinelemesiyle değişir. Böylelikle ağ özelliklerin belirlenmesi için, verinin hangi bölgelerinin önem taşıdığını belirler.

#### **Düzleştirilmiş Doğrusal Birim Katmanı (Rectified Lineer Units Layer (ReLu))**

Bu katman konvolüsyon katmanlarından sonra gelir ve CNN nöronlarının çıktıları için en yaygın şekilde devreye sokulan doğrultucu birim olarak bilinir.

Bu katman aynı zamanda aktifleştirme katmanı olarakta bilinir.

Giriş verisine yapmış olduğu etki negatif değerleri sıfıra çekmesidir. Bu katmandan önce kullanılan konvolüsyon katmanında belirli matematiksel işlemler yapıldığı için ağ doğrusal bir yapıdadır. Bu derin ağı doğrusal olmayan bir yapıya sokmak için bu katman uygulanır. Bu katmanın kullanılması ile ağ daha hızlı öğrenir.

#### **Havuzlama Katmanı (Pooling Layer)**

Havuzlama genellikle ReLu katmanından sonra yerleştirilir.

Temel amacı, sonraki konvolüsyon katmanı için giriş boyutunu (Genişlik x Yükseklik) azaltmaktır. Bu işlem veride derinlik boyutunu etkilemez. Bu katmanda gerçekleştirilen işlem, “Aşağı Örnekleme” olarak da adlandırılır.

Bu katman sonucu boyuttaki azalma bilgi kaybına yol açar. Böyle bir kayıp ağ için iki nedenden dolayı faydalıdır. Birincisi, bir sonraki ağ katmanları için daha az hesaplama yükü oluşturur. İkincisi ise sistemin ezberlemesini önler.

CNN’lerde Havuzlama katmanı isteğe bağlıdır ve bazı mimariler bu işlemi gerçekleştirmez.

Genellikle maksimum havuzlama, daha iyi performans gösterdiği için tercih edilir. Havuzlama işlemi, konvolüsyon katmanı sonucu oluşan fitre adedince görüntülerin hepsi için gerçekleştirilir.

#### **Tam Bağlantılı Katman (Fully Connected, Dense Layer)**

CNN mimarisinde art arda gelen konvolüsyon, ReLu ve havuzlama katmanından sonra tam bağlantılı katman gelir. Bu katman kendinden önceki katmanın tüm alanlarına bağlıdır. Farklı mimarilerde bu katmanın sayısı değişebilir.

#### **DropOut Katmanı**

ESA’da büyük veriler ile eğitim işlemi yapıldığı için bazen ağ ezberleme yapar. Ağın ezberlemesinin önüne geçmek için bu katman kullanılır. Bu katmanda uygulanan temel mantık ağın bazı düğümlerinin kaldırılmasıdır.

#### **Sınıflandırma Katmanı (Classification Layer)**

Bu katman tam bağlantılı katmandan sonra gelir. Derin öğrenme mimarilerinin bu katmanında sınıflandırma işlemi yapılmaktadır. Bu katmanın çıkış değeri, sınıflandırması yapılacak nesne sayısına eşittir.

# **OpenCV ile Görüntü İşleme**

OpenCV (Open Source Computer Vision) açık kaynak kodlu görüntü işleme kütüphanesidir. 1999 yılında İntel tarafından geliştirilmeye başlanmış daha sonra Itseez, Willow, Nvidia, AMD, Google gibi şirket ve toplulukların desteği ile gelişim süreci devam etmektedir. İlk sürüm olan OpenCV alfa 2000 yılında piyasaya çıkmıştır. İlk etapta C programlama dili ile geliştirilmeye başlanmış ve daha sonra birçok algoritması C++ dili ile geliştirilmiştir.

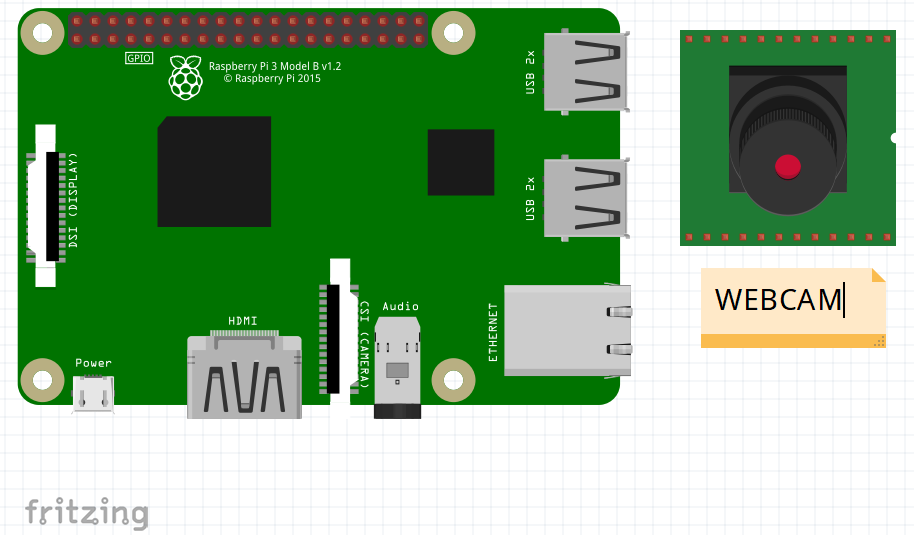
OpenCV platform bağımsız bir kütüphanedir, bu sayede Windows, Linux, FreeBSD, Android, Mac OS ve iOS platformlarında çalışabilmektedir. C++, C, Python, Java, Matlab, EmguCV kütüphanesi aracılığıyla da Visual Basic.Net, C# ve Visual C++ dilleri ile topluluklar tarafından geliştirilen farklı wrapperlar aracılığıyla Perl ve Ruby programlama dilleri ile kolaylıkla OpenCV uygulamaları geliştirilebilir.

OpenCV kütüphanesi içerisinde görüntü işlemeye (image processing) ve makine öğrenmesine (machine learning) yönelik 2500’den fazla algoritma bulunmaktadır. Bu algoritmalar ile yüz tanıma, nesneleri ayırt etme, insan hareketlerini tespit edebilme, nesne sınıflandırma, plaka tanıma, üç boyutlu görüntü üzerinde işlem yapabilme, görüntü karşılaştırma, optik karakter tanımlama OCR (Optical Character Recognition) gibi işlemler rahatlıkla yapılabilmektedir

# **Donanım Alt Yapısı**

Projede kullanılan elemanlar aşağıda ki gibidir:

1. Raspberry Pi 3 Model B v1.2
2. Display
3. Klavye ve Fare
4. Webcam



**Resim 2:** Kullanılan Mikro İşlemci ve Kamera

# **Yazılım Alt Yapısı**

Bu projenin yazılım alt yapısı 4 ana dosyadan;

* DataSetGenerator.py
* NetworkBuilder.py
* MyFirstCNNModel.py
* Test.py

Ve 3 klasörden;

* saved model v2
* summary\_log
* train

Oluşmaktadır.

## **DATA SET GENERATOR OLUŞTURULMASI**

DataSetGenerator.py isimli kod dosyasında oluşturmuş olduğumuz **DataSetGenerator** isimli **class** ile train klasöründe ki verilerimizi (fotoğrafları) istediğimiz format (boyut = 128,128) ve sayıda (batches =10) **return** etmemizi sağlar.

* Gerekli olan modülleri kod dosyamıza import edelim:

1. import cv2
2. import numpy **as** np
3. **from** os.path import isfile, join
4. **from** os import listdir
5. **from** random import shuffle
6. **from** shutil import copyfile
7. import os
8. import pickle

import cv2

OpenCV modülünü eklemeyi sağlar. OpenCV görüntü işlemek için kullanılan bir modüldür.

import numpy **as** np

Numpy modülünü ekleyip np isim değişkeni ile kullanmayı sağlar. Numpy modülü matris işlemlerini kolay ve hızlı bir şekilde yapmamızı sağlayan bir modüldür.

**from** os.path import isfile, join

os.path modülünden isfile ve join fonksiyonlarını eklememizi sağlar.

**isfile** () metodu, parametre olarak bir path alıp onun içinde bir sisteme ait data dosyası varsa **True** return eder.

**Join** () metodu, parametre olarak aldığı 2 veya daha çok pathleri birleştirerek tek bir path (yol adresine) dönüştürür.

**from** os import listdir

os modülünün içerisinden **listdir** () metodunu eklemeyi sağlar. Parametre olarak verilen adres yolunda ki dosya isimlerini return eder.

**from** random import shuffle

random modülünün içerisinde ki **shuffle** () metodunu eklemeyi sağlar. Shuffle metodu parametre olarak verilen liste içeriklerini karıştırmayı sağlar.

**from** shutil import copyfile

shutil modülünün içerisinde ki **copyfile** () metodunu eklemeyi sağlar.

import os

os modülünü eklemeyi sağlar. Yazdığımız programda kullandığımız ifadeleri farklı işletim sistemlerinde tutarlı bir şekilde kullanama bilmemiz için birtakım araçlar sunar.

import pickle

pickle modülünü eklemeyi sağlar. Pickle modülü, herhangi bir python nesnesini binary olarak bir dosyaya yazmasını sağlar.

* Şimdi DataSetGenerator isimli class oluşturalım:

1. class **DataSetGenerator**:
2. **def** \_\_init\_\_(self, data\_dir):
3. self.data\_dir = data\_dir
4. self.data\_labels = self.get\_data\_labels()
5. self.data\_info = self.get\_data\_paths()

**def** \_\_init\_\_(self, data\_dir):

Constructor metodunu oluşturur.

self.data\_dir = data\_dir

self.data\_labels = self.get\_data\_labels()

self.data\_info = self.get\_data\_paths()

data\_dir, data\_labels ve data\_info; DataSetGenerator sınıfının değişkenleridir (Attribute).

data\_dir : verilerin bulunduğu klasör (dizin).

data\_labels : verilerin etiketleri (sınıflar)

data\_info : verilerin path ‘leridir.

* Şimdi aynı class içinde get\_data\_labels () metodunu oluşturmayla devam edelim.

1. **def** **get\_data\_labels**(self):
2. data\_labels = []
3. for filename in listdir(self.data\_dir):
4. if not isfile(join(self.data\_dir, filename)):
5. data\_labels.append(filename)
6. return data\_labels

data\_labels = [ ] isminde boş bir liste oluşturduk.

for filename in listdir(self.data\_dir):

data\_dir değişkeninde bulunan verilerin tarama işlemi yapar.

if not isfile(join(self.data\_dir, filename)):

data\_labels.append(filename)

return data\_labels

Eğer, data\_dir içinde ki veriler sistem dosyası değil ise, bu verileri data\_labels listesine ekle ve en son olarak da data\_labels listesini return et.

* Şimdi aynı class içinde get\_data\_paths () metodunu oluşturmayla devam edelim.

1. **def** get\_data\_paths(self):
2. data\_paths = []
3. for label in self.data\_labels:
4. img\_lists = []
5. path = join(self.data\_dir, label)
6. for filename in listdir(path):
7. tokens = filename.split('.')  if tokens[1] == 'jpg':
8. image\_path = join(path, filename)
9. img\_lists.append(image\_path)
10. shuffle(img\_lists)
11. data\_paths.append(img\_lists)
12. return data\_paths

data\_paths = [ ] isminde boş bir dizi oluşturduk.

for label in self.data\_labels:

img\_lists = []

path = join(self.data\_dir, label)

data\_labels listesinin içindeki etiketleri sırayla label değişkenine atayarak tarama yapıyoruz. Daha sonra img\_lists = [ ] isminde boş bir liste daha oluşturuyoruz.

Sonra da data\_dir değişkeninde ki verileri label ile birleştirerek path değişkenine atıyoruz.

for filename in listdir(path):

tokens = filename.split('.')

if tokens[-1] == 'jpg':

image\_path = join(path, filename)

img\_lists.append(image\_path)

İlk for döngüsünün içerisinde bir for döngüsü daha oluşturduk.

Bu for döngüsünde, ilk for içinde oluşturulan path (verilerin bulunduğu adres yolu) isimli değişkenin içinde ki her bir veriyi sırayla filename isimli değişkene atıyoruz. Dosya adından dosya uzantısını ayırdık. Sonra eğer dosya uzantısı ‘jpg’ ise dosya ismi ile path i birleştirerek image\_path değişkenine atıyoruz. Sonra da img\_path i image\_list e ekliyoruz.

Sonra img\_list i shuffle ile karıştırıyoruz. Sonra da data\_path’e ekledik.

Son olarak da bu data\_paths’leri return ettik.

* Şimdi aynı class içinde get\_mini\_batches () metodunu oluşturmayla devam edelim.

1. **def get\_mini\_batches**(self, batch\_size = 10, image\_size = (200, 200), allchannel = **True**):
2. images = []
3. labels = []
4. empty = **False**
5. counter = 0
6. each\_batch\_size = int(batch\_size / **len**(self.data\_info))
7. while True:
8. for i in **range**(**len**(self.data\_labels)):
9. label = np.zeros(**len**(self.data\_labels), dtype =int)
10. label[i] = 1
11. if **len**(self.data\_info[i]) <= counter + 1:
12. empty = **True**
13. continue
14. empty = **False**
15. img = cv2.imread(self.data\_info[i][counter])
16. img = self.resizeAndPad(img, image\_size)
17. if not allchannel:
18. img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)
19. img = np.reshape(img, (img.shape[0], img.shape[1], 1))
20. images.append(img)
21. labels.append(label)
22. counter += 1
23. if empty:
24. break
25. if (counter) % each\_batch\_size == 0:
26. **yield** np.array(images, dtype = np.uint8), np.array(labels, dtype =np.uint8)
27. **del** images
28. **del** labels
29. images = []
30. labels = []

get\_mini\_batches () metodunun parametreleri:

**def get\_mini\_batches**(self, batch\_size = 10, image\_size = (200, 200), allchannel = **True**):

batch\_size = 10; CNN ağına aynı anda verdiğimiz toplam fotoğraf sayısı,

image\_size = (200, 200) ; çıkıştaki fotoğrafın boyutu,

allchannel = True; çıkıştaki fotoğraflar RGB, eğer False ise Gray’dir.

empty = **False** ; default olarak False tanımlıyoruz. Eğer dosyada veri yok ise True olacaktır.

each\_batch\_size = int(batch\_size / **len**(self.data\_info))

each\_batch\_size; her sınıftan kaç adet fotoğraf alınacağını belirler.

1. while True:
2. for i in **range**(**len**(self.data\_labels)):
3. label = np.zeros(**len**(self.data\_labels), dtype =int)
4. label[i] = 1

ilk for döngüsü fotoğraf veri sınıfı sayısına kadar tekrar edecek.

Label = fotoğraf veri sınıfı sayısına kadar sıfır matrisine eişittir.

Label[ i ] = 1; i.inci sınıf için label dizisinin i.inci elemanı 1 olur.

**ÖRNEK:**

A, B, C, D, E sınıflarımız olsun. Mesela B harfi için **label** dizisi;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 0 |

1. if **len**(self.data\_info[i]) <= counter + 1:
2. empty = **True**
3. continue

data\_info değişkeninde ki path sayısı küçük ise counter+1 (yani 0+ 1 = 1) den sınıf klasöründe hiç dosya yok demektir.

O yüzden empty = True olacak ve continue ile bir sonra ki sınıfa geçiyor.

1. img = cv2.imread(self.data\_info[i][counter])
2. img = self.resizeAndPad(img, image\_size)
3. if not allchannel:
4. img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)
5. img = np.reshape(img, (img.shape[0], img.shape[1], 1))

cv2.imread ile sınıftaki görüntüyü okuyup img değişkenine atıyor. Sonra da bu img değişkeni resizeAndPad () metodu yardımıyla boyutlandırılıyor.

1. images.append(img)
2. labels.append(label)

Sonra da img, images listesine ekleniyor. Label da labes listesine ekleniyor.

1. if empty:
2. break
3. if (counter) % each\_batch\_size == 0:
4. **yield** np.array(images, dtype = np.uint8), np.array(labels, dtype =np.uint8)
5. **del** images
6. **del** labels
7. images = []
8. labels = []

Eğer empty == True ise çık, değil ise zaten devam edecek.

Eğer each\_batch\_ size, counter’a tam bölünüyor ise batch\_size tamamlanmış demektir. Ve elde ettiğimiz images ve labels listelerini yield ediliyor.

Sonra da del ile images ve labels listeleri sıfırlanıyor.

**ÖRNEK:**

A, B, C, D, E olmak üzere 5 adet sınıfımız olsun ve batch\_size = 10 olduğu için images ve labes liste boyutu 10 olur.

**İmages listesi:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| img 0 | img 1 | img 2 | img 3 | img 4 | img 5 | img 6 | img 7 | img 8 | img 9 |

**Labes listesi:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Label0 | Label1 | Label2 | Label3 | Label4 | Label5 | Label6 | Label7 | Label8 | Label9 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 1 |

* Şimdi aynı class içerisinde resizeAndPad () metodunu oluşturmayla devam edelim:

1. **def** **resizeAndPad**(self, img, size):
2. h, w = img.shape[: 2]
3. sh, sw = size
4. if h >= sh **or** w >= sw:
5. interp = cv2.INTER\_AREA
6. else :
7. interp = cv2.INTER\_CUBIC
8. aspect = w / h
9. if aspect >= 1:
10. new\_shape = **list**(img.shape)
11. new\_shape[0] = w
12. new\_shape[1] = w
13. new\_shape = **tuple**(new\_shape)
14. new\_img = np.zeros(new\_shape, dtype = np.uint8)
15. h\_offset = int((w - h) / 2)
16. new\_img[h\_offset: h\_offset + h, : , : ] = img.copy()
17. **elif** aspect <= 1:
18. new\_shape = **list**(img.shape)
19. new\_shape[0] = h
20. new\_shape[1] = h
21. new\_shape = **tuple**(new\_shape)
22. new\_img = np.zeros(new\_shape, dtype = np.uint8)
23. w\_offset = int((h - w) / 2)
24. new\_img[: , w\_offset: w\_offset + w, : ] = img.copy()
25. else:
26. new\_img = img.copy()
27. scaled\_img = cv2.resize(new\_img, size, interpolation=interp)
28. **return** scaled\_img

resizeAndPad () metodunun parametreleri:

img, boyutlandırılacak olan fotoğraftır. size ise boyutlandırılacak olan fotoğrafın istenilen boyutudur.

h, w = img.shape[: 2]

sh, sw = size

h: Verilen fotağrafın yüksekliği, w: fotoğrafın genişliği

sh: İstanilen yükseklik, sw:istenilen genişlik

1. if h >= sh **or** w >= sw:
2. interp = cv2.INTER\_AREA
3. else :
4. interp = cv2.INTER\_CUBIC
5. aspect = w / h

Verilen fotoğraf yüksekliği veya genişliği istenilen yükseklik veya genişlikten büyük ise;

İnterp değişkenine cv2.INTER\_AREA atanır.

Eğer küçük ise;

İnterp değişkenine cv2.INTER\_CUBIC atanır.

cv2.INTER\_CUBIC interpolasyon metodu, genişletmek için daha iyi iken cv2.INTER\_AREA metodu, büyütmek için daha iyidir.

1. if aspect >= 1:
2. new\_shape = **list**(img.shape)
3. new\_shape[0] = w
4. new\_shape[1] = w
5. new\_shape = **tuple**(new\_shape)
6. new\_img = np.zeros(new\_shape, dtype = np.uint8)
7. h\_offset = int((w - h) / 2)
8. new\_img[h\_offset: h\_offset + h, : , : ] = img.copy()

Eğer w/h>=1 ise;

Fotoğraf yataydır.

new\_shape değişkenine liste olarak verilen fotoğrafın boyutunu aktarır.

new\_shape dizisinin 0. Ve 1. İndisi w ‘ya eşitlenerek gerçek fotoğrafın en büyük kenarına göre bir kare sıfır matrisi oluşturulur.

Sonra da gerçek fotoğraf (img) bu kare sıfır matrisine kopyalanır. Böylece verilen görüntü cv2.resize () fonksiyonuna kare olarak gönderilmiş olur.

1. **elif** aspect <= 1:
2. new\_shape = **list**(img.shape)
3. new\_shape[0] = h
4. new\_shape[1] = h
5. new\_shape = **tuple**(new\_shape)
6. new\_img = np.zeros(new\_shape, dtype = np.uint8)
7. w\_offset = int((h - w) / 2)
8. new\_img[: , w\_offset: w\_offset + w, : ] = img.copy()

Eğer w/h<=1 ise;

Fotoğraf dikeydir.

Bir önceki if döngüsünde tanımlanan işlemlerin aynısı yapılır fakat, new\_shape matrisinin yüksekliğine ve genişliğine h değeri atanır.

Sonra da gerçek fotoğraf (img) bu kare sıfır matrisine kopyalanır. Böylece verilen görüntü cv2.resize () fonksiyonuna kare olarak gönderilmiş olur.

1. else:
2. new\_img = img.copy()
3. scaled\_img = cv2.resize(new\_img, size, interpolation=interp)
4. **return** scaled\_img

else: işleminde verilen fotoğraf yatay veya dikey değil ise direk olarak gerçek görüntü (img) new\_img ‘e kopyalanır.

Cv2.resize () fonksiyonuna parametre olarak vereceğimiz new\_img, istediğimiz boyut (size) ve isteğimiz interpolasyon metoduna göre boyutlandırma işlemi yapılarak boyutlandırılmış fotoğraf return edilir.

**ÖRNEK:**

resizeAndPad () metoduna örnek olarak img(500,200) fotoğrafı verilmiş olsun. İstenilen size ise size(200,200) olmuş olsun. Şimdi bunu inceleyelim.

Verilen orijinal görüntü (img)

200

500

h = 200, w = 500 olmuştur.

sh = 200, sw = 200 istiyoruz.

İnterpolasyon metodunun belirlenmesi;

w >= sw (500 >= 200 ) olduğu için interpolasyon metodu, cv2.INTER\_AREA olacaktır.

aspect = w / h oranı, yani 500/200 = 2.5 olacaktır.

aspect 1’den büyük olduğu için fotoğraf yataydır.

Fotoğraf yatay olduğu için, en büyük kenarı (yatay olduğu için yatay kenarı) baz alınarak yeni bir şekil oluşturulur (new\_shape).

Sonra new\_shape boyutunda bir sıfır matrisi oluşturularak new\_image ‘e atanır.

SIFIR MATRİSİ

500

500

h\_offset = (w – h ) / 2 yani h\_offset = 150 olur.

h\_offset + h = 150 + 200 = 350 olur.

new\_img sıfır matrisine verilen görüntünün kopyalanmış hali;

200 = (h\_offset + h)- (h\_offset)

500

Cv2.resize () metoduna parametre olarak;

new\_img(500,500),

size(200,200),

interp = cv2.INTER\_AREA gönderiliyor. Çıkış olarak ise aşağıda ki gibi olmaktadır.

200

200

## **NETWORK BUILDER OLUŞTURULMASI**

NetworkBuilder.py isimli kod dosyasında tanımlanan **NetworkBuilder** isimli **class** ile CNN ağının katmanlarının oluşturulması sağlanır.

1. import tensorflow **as** tf
2. class **NetworkBuilder**:
3. **def** **\_\_init\_\_**(self): #class Constructor başlatıldı.
4. **pass**

import tensorflow **as** tf

TensorFlow kod dosyasına eklendi ve tf isim değişkenine atandı.

* İlk olarak Konvolüsyon Katmanı (Convolutional Layer) tanımlayalım:

1. **def** **attach\_conv\_layer**(self, input\_layer, output\_size = 32, feature\_size = (5, 5), strides = [1, 1, 1, 1], padding = 'SAME', summary = **False**):
2. with tf.name\_scope("Convolution")**as** scope:
3. input\_size = input\_layer.get\_shape().as\_list()[1]
4. weights = tf.Variable(tf.random\_normal([feature\_size[0], feature\_size[1], input\_size, output\_size]),
5. name = 'conv\_weights')
6. if summary:
7. tf.summary.histogram(weights.name, weights)
8. biases = tf.Variable(tf.random\_normal([output\_size]),
9. name = 'conv\_biases')
10. conv = tf.nn.conv2d(input\_layer, weights, strides = stride s, padding = padding) + biases
11. return conv

attach\_conv\_layer () metodunun parametreleri:

input Layer, giriş katmanıdır.

output\_size, çıktı boyutudur.

Feature\_size, özellik boyutudur. Giriş katmanından özellik çıkartmak için kullanılan yapı elemanıdır.

strides, 4 yönde yapı elemanının atacağı adımlar.

Padding = ‘SAME’ sıfırlama metodudur.

Summary değişkeni bu katmanın tensorBoard da kullanılması isteniyor ise True, istenmiyorsa False değerini alır.

Şimdi tensorflow oturumu açalım ve katmanın değişkenlerini tanımlayalım.

1. with tf.name\_scope("Convolution")**as** scope:
2. input\_size = input\_layer.get\_shape().as\_list()[-1]
3. weights = tf.Variable(tf.random\_normal([feature\_size[0], feature\_size[1], input\_size, output\_size]),
4. name = 'conv\_weights')
5. if summary:
6. tf.summary.histogram(weights.name, weights)
7. biases = tf.Variable(tf.random\_normal
8. ([output\_size]), name = 'conv\_biases')
9. conv = tf.nn.conv2d(input\_layer, weights,
10. strides = stride s,
11. padding = padding) + biases
12. return conv

input\_size = input\_layer.get\_shape().as\_list()[-1]

Giriş katmanın boyutu bir liste olarak alınır ve son elemanı input\_size değişkenine atanır. Bu son eleman 3 ise görüntü RGB, 1 ise Gray’dir.

weights = tf.Variable(tf.random\_normal([feature\_size[0],feature\_size[1],

input\_size, output\_size]),name = 'conv\_weights')

Katmanın ağırlıkları ‘conv weights’ ismi ile tanımlanır ve tf.random\_normal fonksiyonu kullanılarak rastgele bir başlangıç değeri seçilerek weights değişkenine atanır.

if summary:

tf.summary.histogram(weights.name, weights)

TensorBoard kullanılması isteniyorsa bu ağırlıklar (weights) diyagrama eklenir.

biases = tf.Variable(tf.random\_normal([output\_size]),

name = 'conv\_biases')

Biases katmanı tanımlanır ve tf.random\_normal fonksiyonu kullanılarak başlangıç değeri atanır.

conv = tf.nn.conv2d(input\_layer, weights, strides = strides,

padding = padding) + biases

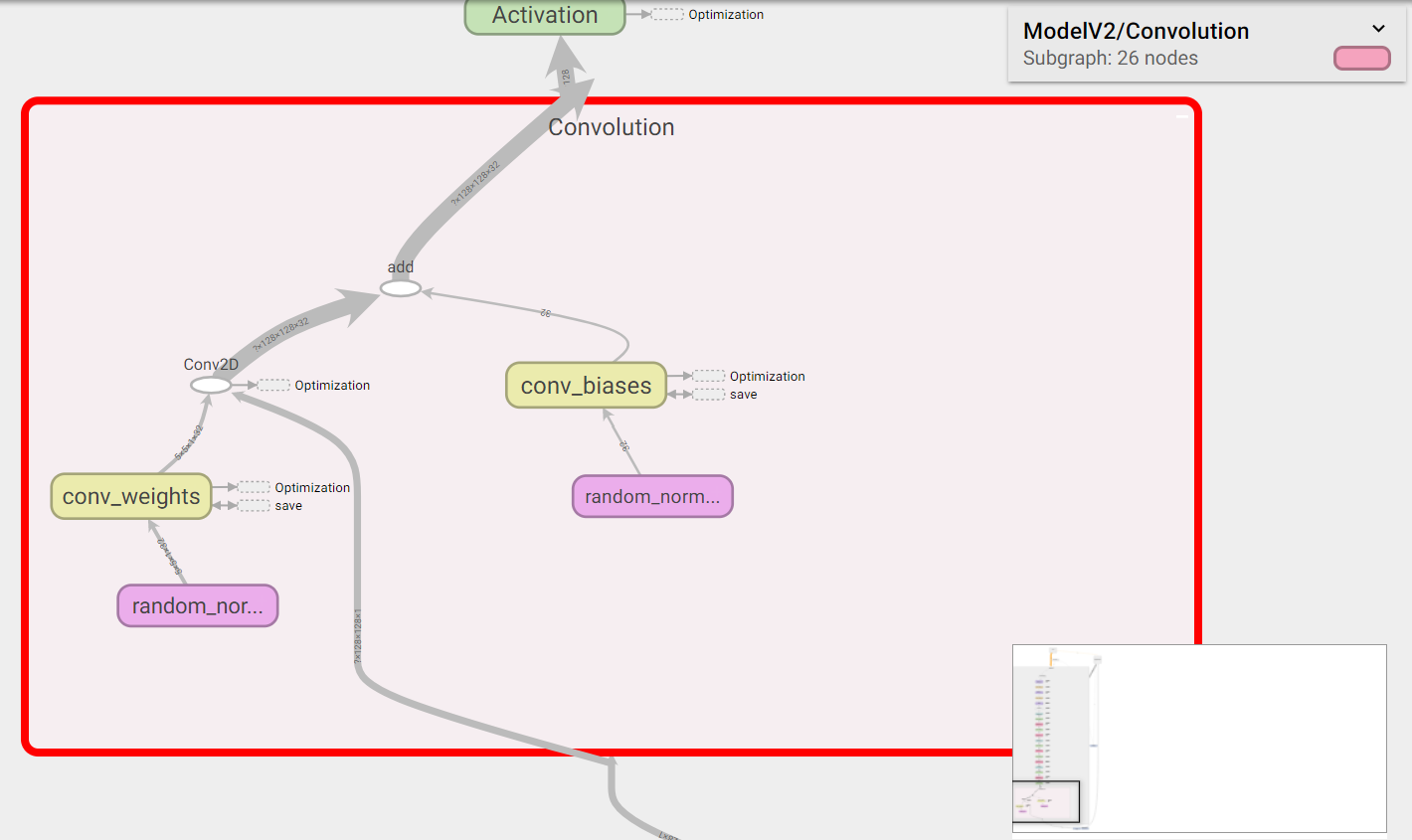
return conv

Son katman oluşturulur ve return edilir.

TensorFlow regresyon yöntemini kullanmaktadır:

y = ax + b

Output Layer = ( weights \* input\_layer ) + biases



**Resim 3:** Convolutional Layer TensorBoard Görüntüsü

* Şimdi ikinci olarak Havuzlama Katmanını (Pooling Layer ) tanımlayalım:

1. **def** **attach\_pooling\_layer**(self, input\_layer, ksize = [1, 2, 2, 1],
2. strides = [1, 2, 2, 1], padding = 'SAME'):
3. with tf.name\_scope("Pooling") **as** scope:
4. return tf.nn.max\_pool(input\_layer, ksize = ksize,
5. strides = str ides, padding = padding)

attach\_poling\_layer () metodun parametreleri;

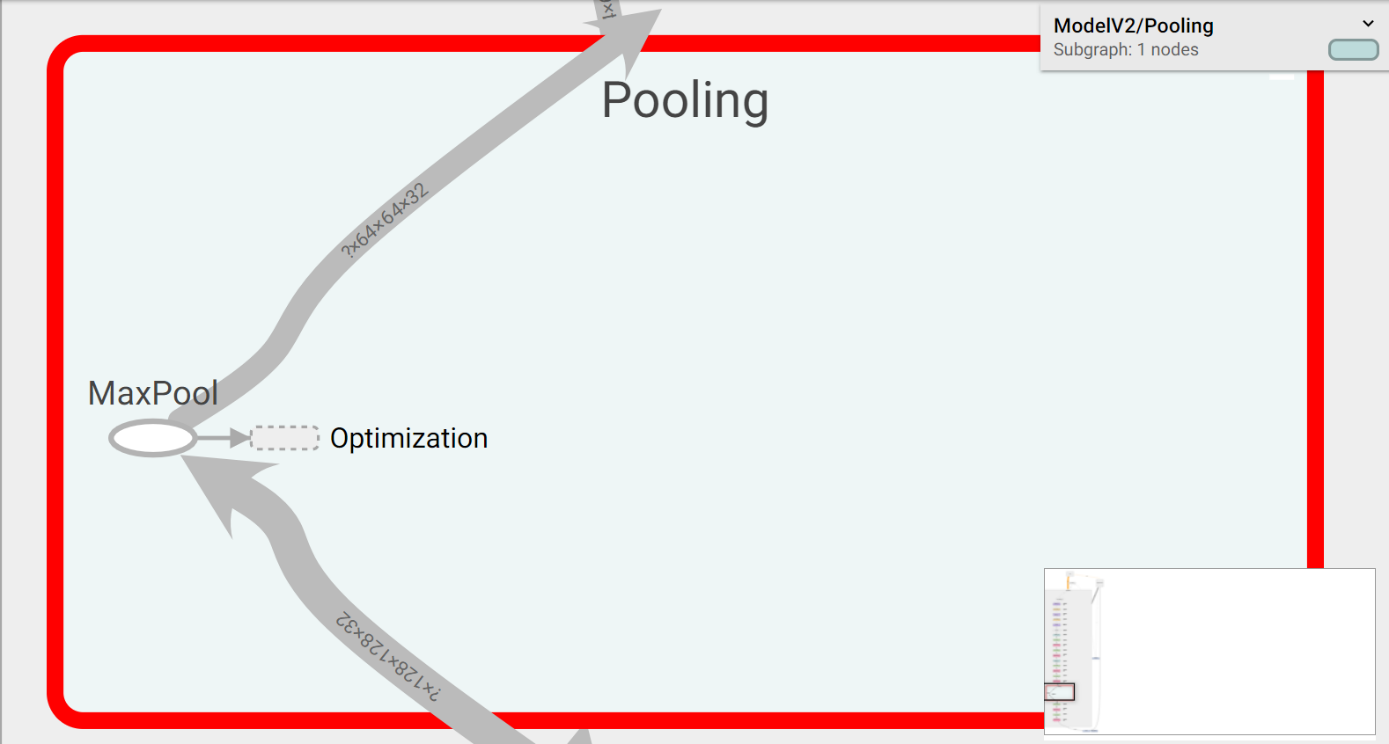
input\_layer, giriş katmanıdır.

ksize, Pooling size’dır. (Havuzlama boyutu)

strides, yapı elemanı boyutudur.

padding ise sıfırlama metdodudur.

Output olarak direk olarak Pooling katmanını gönderir.



**Resim 4:** Pooling Layer TensorBoard Görüntüsü

* Şimdi Dense katmanından önce görüntüyü 3D den 1D ‘ye çevirecek olan Flatten dönüştürücüsünü oluşturalım:

1. **def** **flatten**(self, input\_layer):
2. with tf.name\_scope("Flatten")**as** scope:
3. input\_size = input\_layer.get\_shape().as\_list()
4. new\_size=input\_size[-1] \* input\_size[-2]\*input\_size[-3]
5. return tf.reshape(input\_layer, [-1, new\_size])

flatten () metodunun parametresi input\_layer ‘dir.

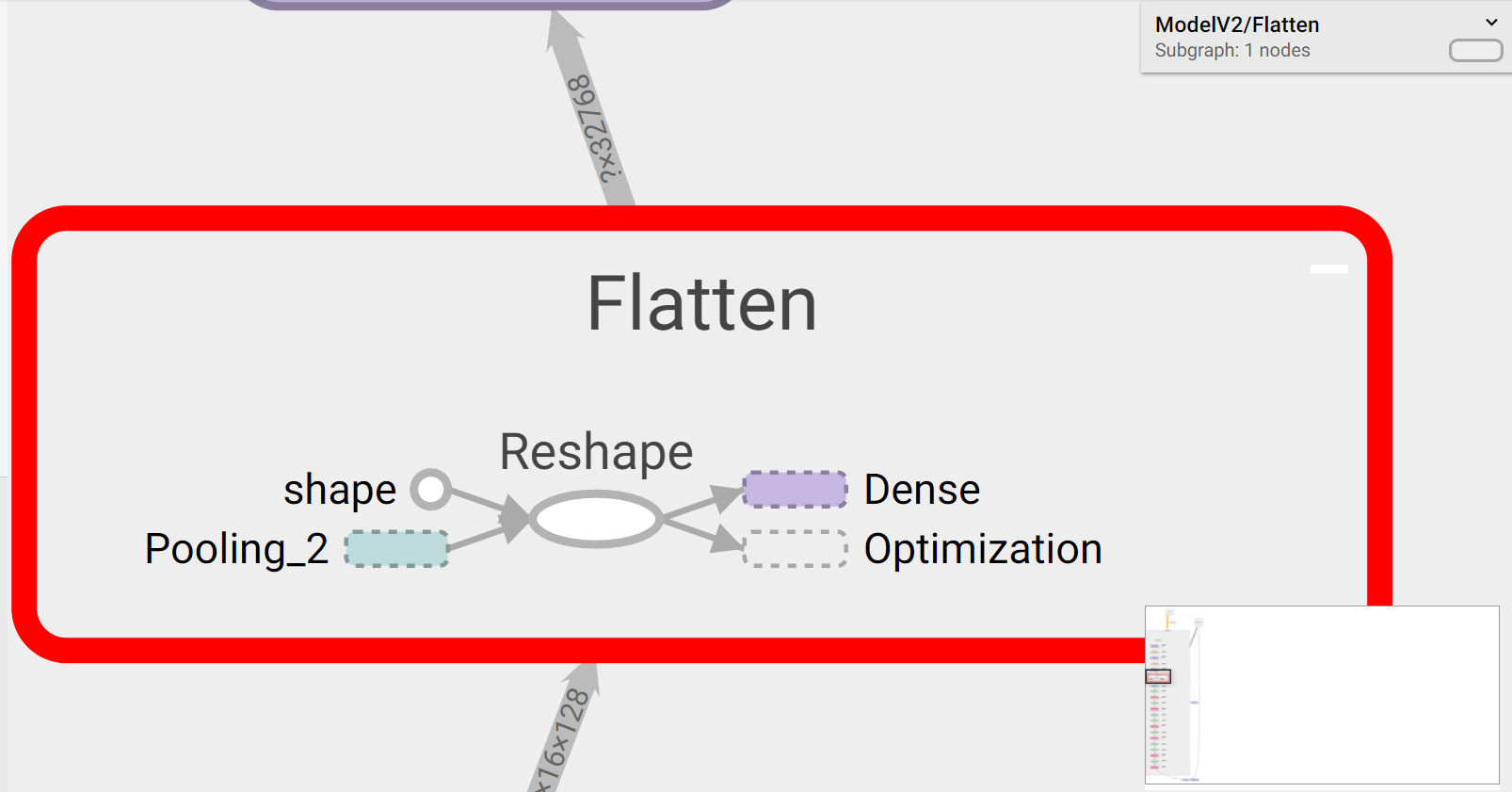
input\_size = input\_layer.get\_shape().as\_list()

get\_shape() fonksiyonu yardımıyla giriş katmanın boyutları alınır ve list olarak input\_size değişkenine atanır.

new\_size=input\_size[-1] \* input\_size[-2]\*input\_size[-3]

return tf.reshape(input\_layer, [-1, new\_size])

input\_size listede ki diğerlerini birbirleri ile çarparak new\_size değişkenine atanır ve sonra da tf.reshape () fonksiyonu yardımıyla input\_layer 3D ‘den 1D ‘ye dönüştürülmesi gerçekleştirilmiş olur.



**Resim 5:** Flatten Layer TensorBoard Görüntüsü

* Şimdi üçüncü olarak Dense katmanını oluşturmayla devam edelim:

1. **def** **attach\_dense\_layer**(self, input\_layer, size, summary = **False**):
2. with tf.name\_scope("Dense") **as** scope:
4. input\_size = input\_layer.get\_shape().as\_list()[1]
5. weights = tf.Variable(tf.random\_normal([input\_size, size]),
6. name = 'dense\_weigh')
7. if summary:
8. tf.summary.histogram(weights.name, weights)
9. biases = tf.Variable(tf.random\_normal([size]),
10. name = 'dense\_biases')
11. dense = tf.matmul(input\_layer, weights) + biases
13. return dense

attach\_dense\_layer () metodu parametre olarak:

input Layer; giriş katmanı,

size; Output size (Çıkış Boyutu).

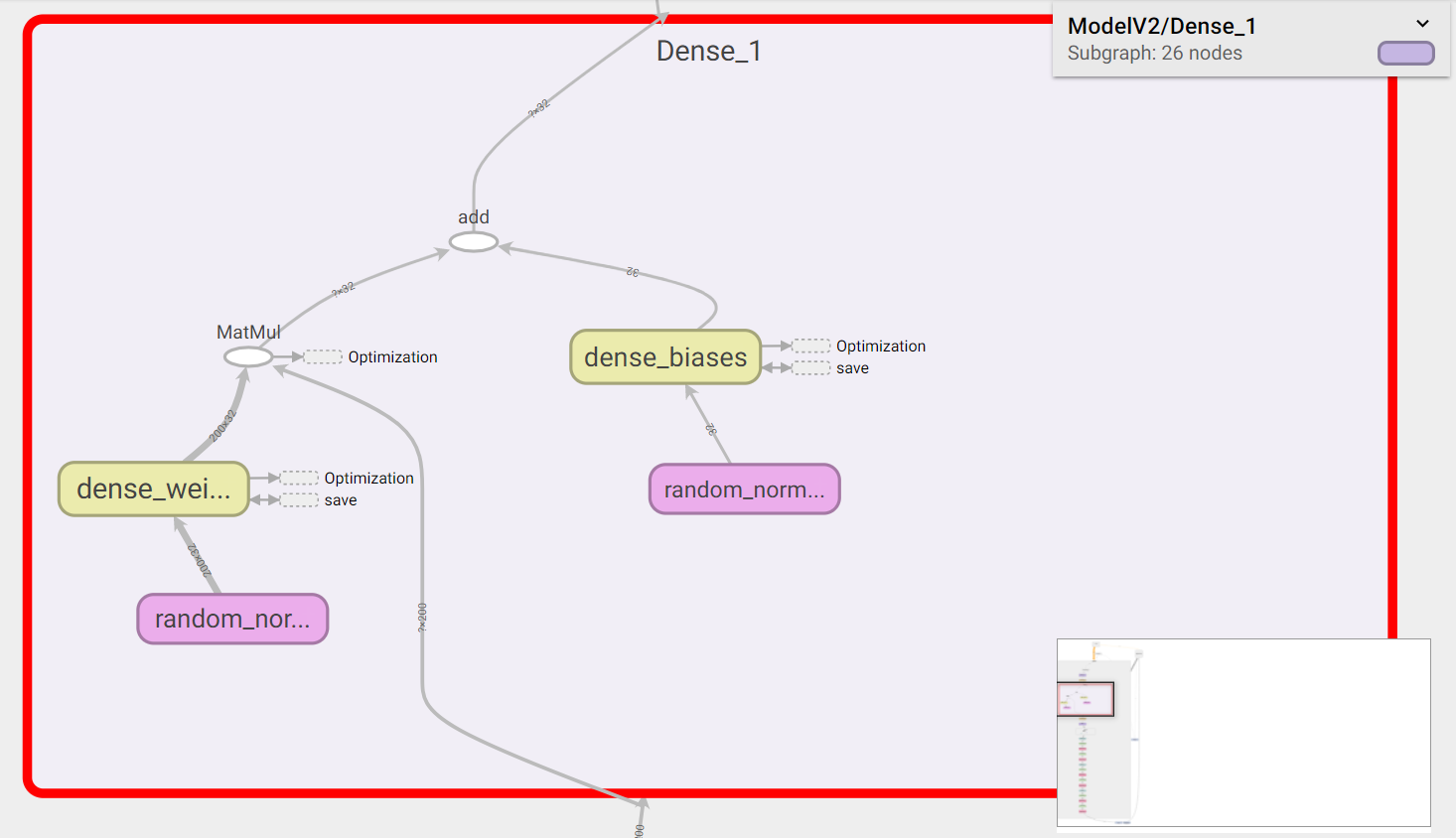
weights ve biases Convolutional Layer da tanımlandığı gibi burada da aynı şekilde tanımlanır.

dense = tf.matmul(input\_layer, weights) + biases

return dense

tf.matmul () fonksiyonu machine learning uygulamalarında çoklu girişleri tekli girişe dönüştürür.

tf.matmul () foksiyonunun return ettiği değer dense değişkenine atanarak dense katmanından return edilir.

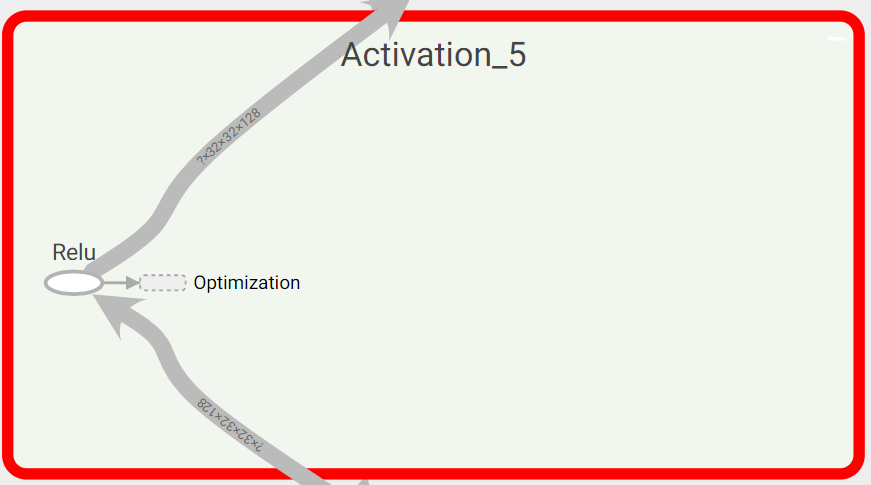


**Resim 6:** Dense Layer TensorBoard Görüntüsü

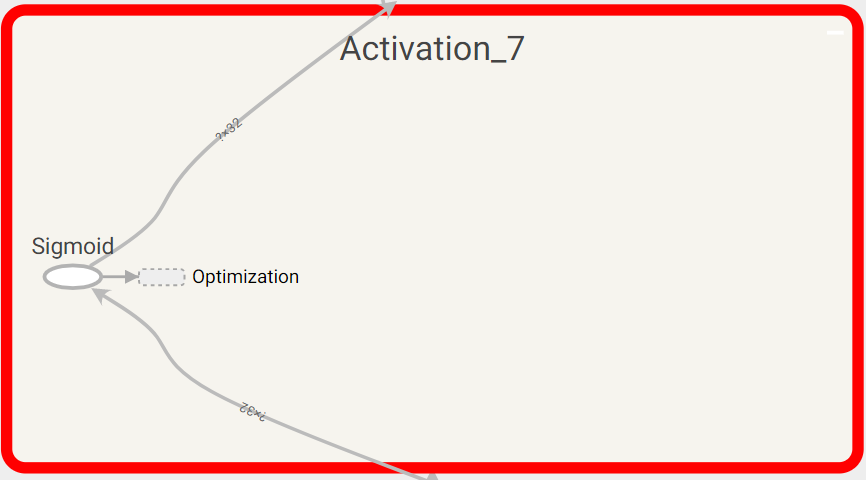
Şimdi Aktivasyon Katmanlarını oluşturarak devam edelim:

relu, sigmoid ve softmax katmanları ağımızda ki doğrusal olmayan(nonlineer) karmaşık işlevleri öğrenmeyi kolaylaştıran Activasyon katmanlarıdır.

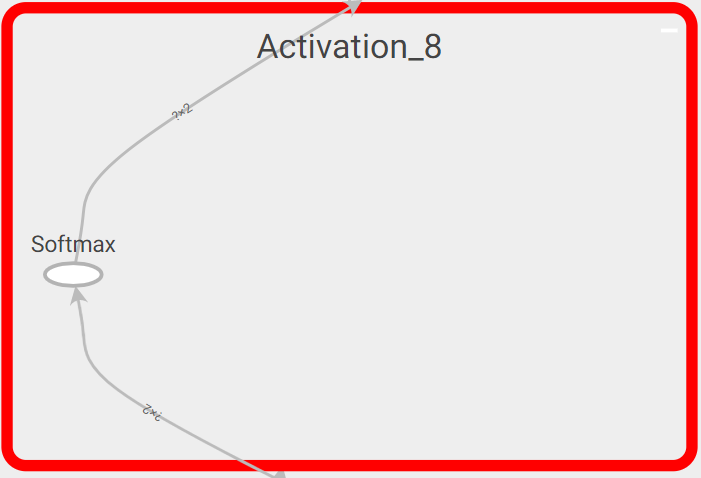
1. **def** **attach\_relu\_layer**(self, input\_layer):
2. with tf.name\_scope("Activation") **as** scope:
3. return tf.nn.relu(input\_layer)
4. **def attach\_sigmoid\_layer**(self, input\_layer):
5. with tf.name\_scope("Activation")**as** scope:
6. return tf.nn.sigmoid(input\_layer)
7. **def** **attach\_softmax\_layer**(self, input\_layer):
8. with tf.name\_scope("Activation") **as** scope:
9. return tf.nn.softmax(input\_layer)



**Resim 7:** Relu Layer TensorBoard Görüntüsü



**Resim 8:** Sigmoid Layer TensorBoard Görüntüsü



**Resim 9:** Softmax Layer TensorBoard Görüntüsü

## **MyFirstCNN MODEL OLUŞTURULMASI**

MyFirstCNNModel.py isimli kod dosyasında **CNN** (**C**onvolutional **N**eural **N**etwork) oluşturulur ve **train** (eğitim) yapılır.

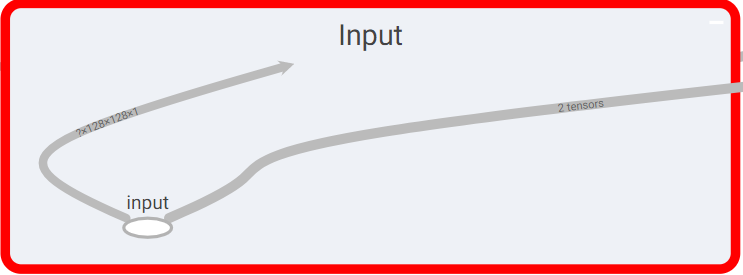
* İlk olarak kod dosyasında kullanacağımız gerekli modülleri ekleyelim.

1. import tensorflow **as** tf
2. **from** NetworkBuilder import NetworkBuilder
3. **from** DataSetGenerator import DataSetGenerator
4. import numpy **as** np
5. import os

* TensorFlow da ki tanımlanan **placeholder** ‘lar veri kapılarıdır. Verilerin bir katmandan girmesini veya çıkmasını sağlarlar. Yani verilen görüntülerin ağ modelimizde ki katmalar arasında ki geçişini sağlayan tutuculardır.
* Şimdi İkinci olarak da Input, Target ve keep\_prob\_input **placeholder**’ ları tanımlayalım:

1. with tf.name\_scope("Input")**as** scope:
2. input\_img = tf.placeholder(dtype = 'float',
3. shape = [**None**, 128,  128, 1], name = "input")

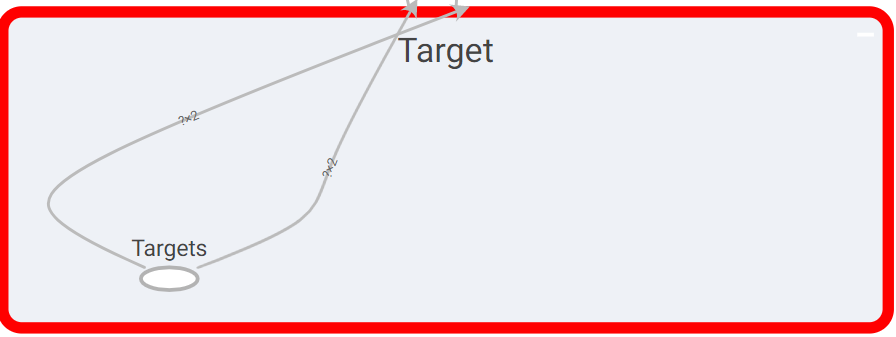
input\_img, verilen görüntünün ağa feed edilmesini sağlayan kapıdır.



**Resim 10:** Input Placeholder TensorBoard Görüntüsü

1. with tf.name\_scope("Target") **as** scope:
2. target\_labels = tf.placeholder(dtype = 'float',
3. shape = [**None**, 5], name = "Targets")

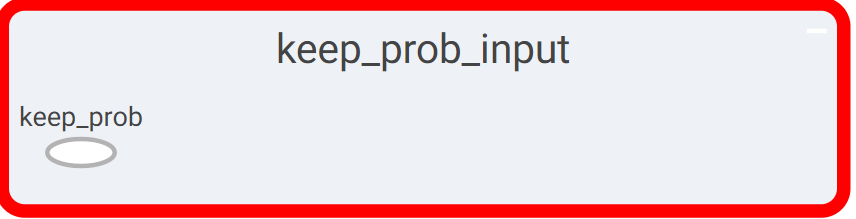
target\_labels, verilen görüntünün hangi sınıftan geldiğini belirleyen listenin ağa feed edilmesini sağlayan kapıdır.



**Resim 11:** Target Placeholder TensorBoard Görüntüsü

1. with tf.name\_scope("keep\_prob\_input"):
2. keep\_prob = tf.placeholder(dtype = 'float', name = 'keep\_prob')

keep\_prob\_input, verilerin takip edilmesi için kullanılır.

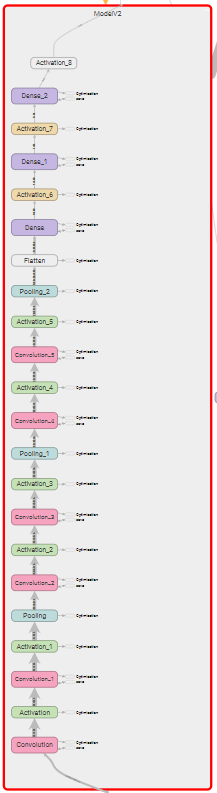


**Resim 12:** keep\_prob\_input Placeholder TensorBoard Görüntüsü

* Şimdi de NetworkBuilder kullanarak sinir ağını oluşturacağız.

1. nb = NetworkBuilder()
2. with tf.name\_scope("ModelV2") **as** scope:
3. model = input\_img
4. model = nb.attach\_conv\_layer(model, 32, summary = **True**)
5. model = nb.attach\_relu\_layer(model)
6. model = nb.attach\_conv\_layer(model, 32, summary = **True**)
7. model = nb.attach\_relu\_layer(model)
8. model = nb.attach\_pooling\_layer(model)
10. model = nb.attach\_conv\_layer(model, 64, summary = **True**)
11. model = nb.attach\_relu\_layer(model)
12. model = nb.attach\_conv\_layer(model, 64, summary = **True**)
13. model = nb.attach\_relu\_layer(model)
14. model = nb.attach\_pooling\_layer(model)
15. model = nb.attach\_conv\_layer(model, 128, summary = **True**)
16. model = nb.attach\_relu\_layer(model)
17. model = nb.attach\_conv\_layer(model, 128, summary = **True**)
18. model = nb.attach\_relu\_layer(model)
19. model = nb.attach\_pooling\_layer(model)
20. model = nb.flatten(model)
21. model = nb.attach\_dense\_layer(model, 200, summary = **True**)
22. model = nb.attach\_sigmoid\_layer(model)
23. model = nb.attach\_dense\_layer(model, 32, summary = **True**)
24. model = nb.attach\_sigmoid\_layer(model)
25. model = nb.attach\_dense\_layer(model, 5)
26. prediction = nb.attach\_softmax\_layer(model)

**nb** isimli NetworkBuilder tipinde bir obje tanımlanır ve onu kullanarak Convolutional Layer, Relu Layer, dense Layer, sigmoid Layer, Softmax Layer birbirine bağlayıp **CNN** modeli oluşturulmuş olur.



**Resim 13:** Model V2 TensorBoard Görüntüsü

* Şimdi de Optimizasyon katmanını tanımlayalım:

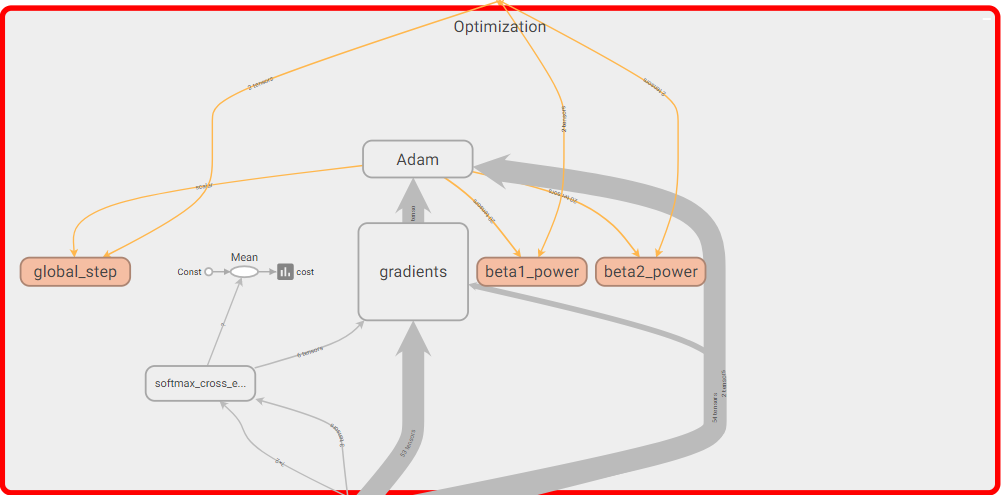
1. with tf.name\_scope("Optimization")**as** scope:
2. global\_step = tf.Variable(0, name = 'global\_step',
3. trainable = **False**)
4. cost = tf.nn.softmax\_cross\_entropy\_with\_logits(logits = model,  labels = target\_labels)
5. cost = tf.reduce\_mean(cost)
6. tf.summary.scalar("cost", cost)
7. optimizer = tf.train.AdamOptimizer().
8. minimize(cost,global\_step=global\_step

global\_step toplam eğitilmiş batch sayısıdır. Her yeni batch için weights değişkeni kayıplar en az olacak şekilde güncellenir

cost değişkeni hata miktarıdır.

tf.train.AdamOptimizer () fonksiyonu hatanın azaltılmasını sağlar.

tf.summary.scalar () fonksiyonu, cost değişkenini TensorBoard da görünmesini sağlar.



**Resim 14:** Optimization Katmanı TensorBoard Görüntüsü

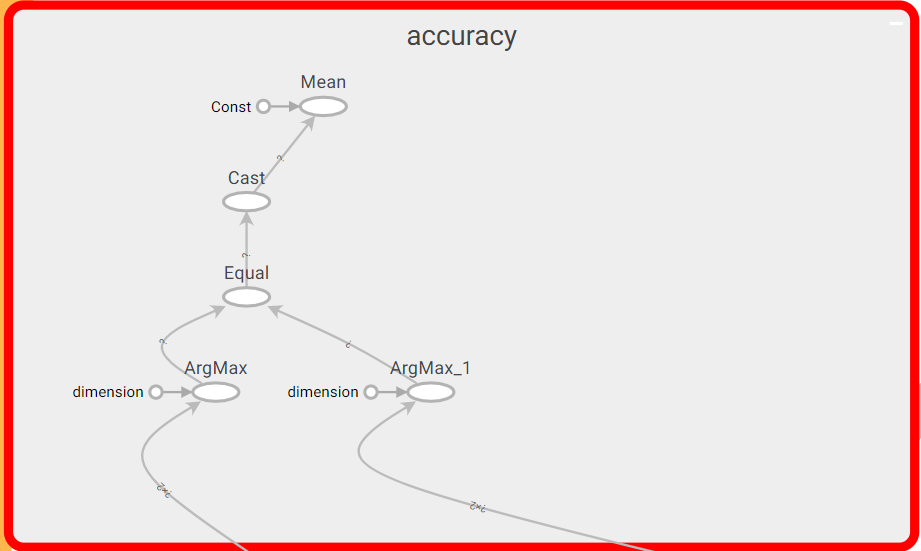
* Şimdi de Accuracy katmanını inceleyelim:

1. with tf.name\_scope('accuracy') **as** scope:
2. correct\_pred = tf.equal(tf.argmax(prediction, 1),
3. tf.argmax(target\_labels, 1))
4. accuracy = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32))

Tahminin, doğruluk derecesini verir.

correct\_pred değişkeni, doğru olan tahminidir.

Accuracy değişkenine, CNN ağının çıkışı olan doğru tahmin ile kıyaslayarak doğruluk derecesi atanır.



**Resim 15:** Accuracy Katmanı TensorBoard Görüntüsü

* Şimdi modelimizi eğitelim:

1. dg = DataSetGenerator("Train dosyasının path’i")
2. epochs = 50
3. batchSize = 10
4. saver = tf.train.Saver()
5. model\_save\_path = "Modelin kaydedileceği dizin"
6. model\_name = 'model'

**dg;** DataSetGenerator () clasınının göndereceği batch,

**epochs**; eğitimin tekrar sayısı,

**batchSize**; 1 batch içinde ki fotoğraf sayısıdır,

**saver**; modelin kaydedilmesini sağlar,

**model\_save\_path**; modelin kaydedileceği dizin,

**model\_name**; modelin ismidir.

* Şimdi TensorFlow Oturumunu Başlatalım:

1. with tf.Session() **as** sess:
2. summaryMerged = tf.summary.merge\_all()
3. filename = "TensorBoard için Kullanılacak dosyanın Dizini"
4. if os.path.exists(model\_save\_path + "checkpoint"):
5. saver.restore(sess,
6. tf.train.latest\_checkpoint(model\_save\_path))
7. writer = tf.summary.FileWriter(filename, sess.graph)
8. tf.global\_variables\_initializer().run()

tf.summary.merge\_all () fonksiyonu, modelin tensorboard da grafiğinin oluşturulmasını sağlar.

if os.path.exists(model\_save\_path + "checkpoint"):

saver.restore(sess,

tf.train.latest\_checkpoint(model\_save\_path))

Eğer eski eğitilmiş bir model varsa tensorflow da ki değişkenlerin değerleri restore edilir.

writer = tf.summary.FileWriter(filename, sess.graph)

Modelin tensorboard da yazılmasını sağlar.

tf.global\_variables\_initializer().run()

Global değişkenlerin başlangıç değerleri initsalizasyon edilir.

* Şimdi eğitme döngüsünü başlatıyoruz:

1. for epoch in **range**(epochs):
2. batches = dg.get\_mini\_batches(batchSize, (128, 128),
3. allchannel = **False**)
4. for imgs, labels in batches:
5. imgs = np.divide(imgs, 255)
6. error, sumOut, acu, steps, \_ =
7. sess.run([cost, summaryMerged, accuracy, global\_step,
8. optimizer], feed\_dict = {input\_img: imgs, target\_labels: labels})
9. writer.add\_summary(sumOut, steps)
10. **print**("Döngü=", epoch, "\*\*\*",  "Toplam Resim=",
11. steps \* batchSize, "\*\*\*", "Hata=", error, "\*\*\*",  "Doğruluk=", acu)
12. if steps % 100 == 0:
13. **print**("Model Kaydediliyor...")
14. saver.save(sess, model\_save\_path + "\\" + model\_name,
15. global\_step = steps)
16. **print**("Model Kaydedildi")

for epoch in **range**(epochs):

Tanımlanan epochs sayısına kadar train yapacağız.

batches = dg.get\_mini\_batches(batchSize, (128, 128),

allchannel = **False**)

Modele feed edilecek batch ’lerde ki fotoğrafların boyutu 128,128 ve Gray olacaktır.

for imgs, labels in batches:

1. imgs = np.divide(imgs, 255)
2. error, sumOut, acu, steps, \_ =
3. sess.run([cost, summaryMerged, accuracy, global\_step,
4. optimizer], feed\_dict =
5. {input\_img: imgs, target\_labels: labels})

Batch içinde ki imgs ve labels objelerini sess.run () fonksiyonuyla modele feed edilir.

**cost**, tahminde ki hatadır.

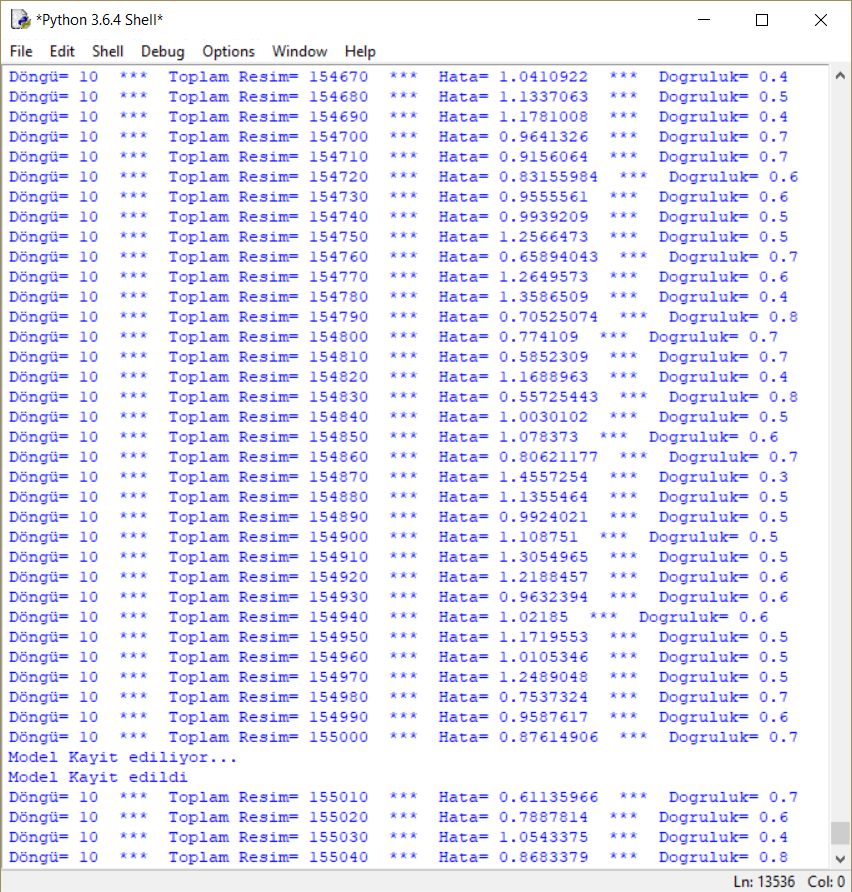
**summaryMerged**, TensorBoard özetidir. TensorBoard için kullanılacak özettir.

**accuracy**, tahminin doğruluk derecesidir.

**global\_step**, eğitilmiş batch sayısıdır.

**optimizer**, optimizasyon değişkeni

**feed\_dict**, görüntüleri ve etiketleri içeren sözlüktür.



**Resim 16:** Modelin Train Edilmesine Ait Görüntü

## **Modelin Test Dosyasının Oluşturulması**

test.py dosyası web cam den alınan görüntüyü test edilmesini sağlar.

Gerekli modüller import edildikten sonra modelin kaydedildiği dizini tanımlıyoruz.

1. model\_save\_path = "Modelin kaydedildiği dizin adresi"

Şimdi cap isimli bir VideoCapture tipinde bir obje tanımlayalım ve web cam den alınan görüntüleri daha sonra sayabilmek için frame\_counter isimli değişken tanımlayalım.

1. cap = cv2.VideoCapture(0)
2. frame\_counter = 0

Şimdi TensorFlow ortamını sess ismiyle başlatıyoruz.

1. with tf.Session() **as** sess:
2. saver = tf.train.import\_meta\_graph('model-xxxx.meta dosyasının bulunduğu klasör')
3. saver.restore(sess, tf.train.latest\_checkpoint(model\_save\_path))
4. input\_img = tf.get\_default\_graph().get\_tensor\_by\_name('Input/input:0')
6. prediction = tf.get\_default\_graph().get\_tensor\_by\_name('ModelV2 /Activation\_8/Softmax:0')
7. target\_labels = tf.get\_default\_graph().get\_tensor\_by\_name('Targ et/Targets:0')
8. frame\_counter = 0
9. eski\_Harf = 'X'

saver = tf.train.import\_meta\_graph('model-xxxx.meta dosyasının bulunduğu klasör')

saver.restore(sess, tf.train.latest\_checkpoint(model\_save\_path))

tf.train.import\_meta\_graph () u ve saver.restore() fonksiyonları, test için kullanılacak modeli restore etmemizi sağlar.

1. input\_img = tf.get\_default\_graph().get\_tensor\_by\_name('Input/input:0')
3. prediction = tf.get\_default\_graph().get\_tensor\_by\_name('ModelV2 /Activation\_8/Softmax:0')
4. target\_labels = tf.get\_default\_graph().get\_tensor\_by\_name('Targ et/Targets:0')

Önceden train edilmiş modelden input\_img, prediction ve target\_labels değişkenleri restore edilir.

Şimdi de web cam den alınan görüntüyü işleyelim:

1. while **True**:
3. ret, frame = cap.read()
4. frame = cv2.flip(frame, 1)
5. frame\_counter = frame\_counter + 1
6. cv2.rectangle(frame, (100, 100), (420, 420), (0, 0, 100), 8)
7. font = cv2.FONT\_HERSHEY\_TRIPLEX
8. cv2.putText(frame, 'Hand Zone', (120, 80), font, 1.5, (0, 0, 10 0), 2, cv2.LINE\_AA)
9. cv2.imshow('frame', frame)
10. elAlani = frame[100: 420, 100: 420]
11. grayElAlani = cv2.cvtColor(elAlani, cv2.COLOR\_BGR2GRAY)
12. gaussianElAlani = cv2.GaussianBlur(grayElAlani, (5, 5), 0)
13. ret1, thresholdingElAlani = cv2.threshold(gaussianElAlani, 0, 2 55, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)
14. cv2.imshow('El Alani', thresholdingElAlani)
15. img = thresholdingElAlani
16. img.resize((320, 320, 3), refcheck = **False**)
17. image = []
18. img = resizeAndPad(img, (128, 128))
19. img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)
20. img = np.reshape(img, (img.shape[0], img.shape[1], 1))
21. image.append(img)
22. testImage = np.array(image, dtype = np.uint8)
23. np.reshape(testImage, [128, 128, 1])

ret, frame = cap.read()

read () fonksiyonuyla görüntü alınır ve ret ve frame değişkenlerine atanır.

frame = cv2.flip(frame, 1)

cv2.flip() fonksiyonuyla alınan görüntüden ayna etkisi çıkarılır.

frame\_counter = frame\_counter + 1

Her yeni alınan görüntü için frame\_counter değişkeni 1 artırılır.

cv2.rectangle(frame, (100, 100), (420, 420), (0, 0, 100), 8)

cv2.rectangle() fonksiyonuyla frame içinde dikdörtgen çizdirilir.

(100, 100) sol üst noktanın koordinatları,

(420, 420) sağ alt noktanın koordinatları,

(0, 0, 100) RGB değerleri yani dikdörtgenin rengi,

8 ise dikdörtgenin çerçeve kalınlığıdır.

font = cv2.FONT\_HERSHEY\_TRIPLEX

cv2.putText(frame, 'EL ALANI', (120, 80), font, 1.5,

(0, 0, 100), 2, cv2.LINE\_AA)

FONT\_HERSHEY\_TRIPLEX yazı tipiyle görüntüde ‘El Alanı’ yazısı yazdırılır.

cv2.imshow('frame', frame)

Görüntüyü göstermek için kullanılır.

elAlani = frame[100: 420, 100: 420]

grayElAlani = cv2.cvtColor(elAlani, cv2.COLOR\_BGR2GRAY)

gaussianElAlani = cv2.GaussianBlur(grayElAlani, (5, 5), 0)

ret1, thresholdingElAlani = cv2.threshold(gaussianElAlani, 0, 255,

cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

cv2.imshow('El Alani', thresholdingElAlani)

Alınan görüntüden elAlani isimli bir alan tanımlanır ve o alan Gray’e çevrilir.

Sonra da GaussianBlur ve thresholding işlemleri görüntüye uygulanır.

Son olarak da elAlani gösterilir.

img = thresholdingElAlani

İşlenmiş el alanı, img değişkenine atanır.

img.resize((320, 320, 3), refcheck = **False**)

img = resizeAndPad(img, (128, 128))

resizeAndPad () fonksiyonunun kabul edebileceği formata img değişkenini boyutlandırılıp bu fonksiyona verilir.

img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

verilen görüntünün GRAY’e çevrilmesi

img = np.reshape(img, (img.shape[0], img.shape[1], 1))

image.append(img)

np.reshape () fonksiyonuyla verilen görüntü yeniden boyutlandırılıp append() fonksiyonuyla da image listesine eklenir.

testImage = np.array(image, dtype = np.uint8)

np.reshape(testImage, [128, 128, 1])

np.array () fonksiyonu ile img listesi matrise çevrilir ve tekrar np.reshape () fonksiyonuyla boyutlandırılır.

Şimdi de CNN çıkış katmanından son tahminleri alıyoruz:

1. Son\_Tahmin=sess.run([prediction],feed\_dict = {input\_img: testImage})
2. tahminler = [Son\_Tahmin[0][0][0],
3. Son\_Tahmin[0][0][1],
4. Son\_Tahmin[0][0][2],
5. Son\_Tahmin[0][0][3],
6. Son\_Tahmin[0][0][4]]

Şimdi de tahminlere göre çıktıları (Harf) gösterelim:

1. if frame\_counter % 50 == 0:
2. if max(tahminler) == Son\_Tahmin[0][0][0] and eski\_Harf is not 'A':  print('A')
3. eski\_Harf = 'A'
4. if max(tahminler) == Son\_Tahmin[0][0][1] and eski\_Harf is not 'B':
5. print('B')
6. eski\_Harf = 'B'
8. if max((tahminler) == Son\_Tahmin[0][0][2]) and(eski\_Harf != 'C'):
9. print('C')
10. eski\_Harf = 'C'
11. if max(tahminler) == (Son\_Tahmin[0][0][3]) and eski\_Harf is not 'Ç':
12. print('Ç')
13. eski\_Harf = 'D'
14. if max(tahminler) == (Son\_Tahmin[0][0][4]) and eski\_Harf is not 'D':
15. print('D')
16. eski\_Harf = 'E'

# **SONUÇ ve DEĞERLENDİRME**



**Resim 17:** Projenin Run Edilmesi ve Çıktısı

Resim 17 ‘de gördüğünüz gibi El Alanında ki görüntü işlenerek DeepLearning ile hangi harf olduğuna karar verilip komut satırına tahmini sonuç yazdırılmaktadır.

## **Programın Eksiklikleri**

1. Öğrenme işleminin hızı: train esnasında 1000 görüntünün feed edilmesi 10 dakika sürmektedir. Her sınıf için 3000 görüntü bulunmaktadır ve toplamda 29 adet sınıf bulunmaktadır. İyi bir öğrenme işlemi için train tekrar sayısı (epoch) yaklaşık 50 olması gerekmektedir. O nedenle basit bir hesap yaparsak 29 sınıftan oluşan bir veri setinin öğrenme süresi: (29\*3000) \*50 ~30 gün sürmektedir. (Yaklaşık Değer)
2. O nedenle projemizde 29 sınıf bulunmasına rağmen bu derece uzun bir süre olmadığı için projede yapmış olduğumuz algoritmanın doğru çalıştığını göstermek için 29 sınıf yerine 5 sınıf kullanarak gerçekleştirdik.

# **KAYNAKÇA**

1. <https://thecodacus.com/>
2. <https://www.tensorflow.org/>
3. <https://opencv.org/>
4. <https://docs.python.org/2.7/>
5. <https://www.quora.com/>
6. <http://mesutpiskin.com/blog/wp-content/uploads/2017/01/OpenCV%20Kitap.pdf>
7. <http://dergipark.gov.tr/download/article-file/380999>
8. <https://mcemilg.github.io/general/2017/11/23/tensorflow/>

# **Projenin Github Adresi**

<https://github.com/mmtaksuu/sign2word.git>

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| https://ci4.googleusercontent.com/proxy/za3R5VMRl7xRmYDEOhCbX2B1XmO81OSjamkEgnS4fOvaEaLM0raiFBNExv1vvzHW5l9caNlhisqzILG5fh_oOuyGUmbuDwwQhXGCbTHWUzs3yZOXHqnEty2ukj03Xns=s0-d-e1-ft#https://mysignature.io/images/photos/ecdd0ea1dc6f71e6dbe371f45bc74080.png | |  | | --- | | **Mehmet AKSU** | | Mechatronics Engineer Student | Maker | | **M:**[05347356989](tel:05347356989) | | **E:**[mmtaksu.25@gmail.com](mailto:mmtaksu.25@gmail.com) | | **A:**Kadıköy / İstanbul | | [www.mechatronictech.com](http://www.mechatronictech.com/) | | |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | | |

