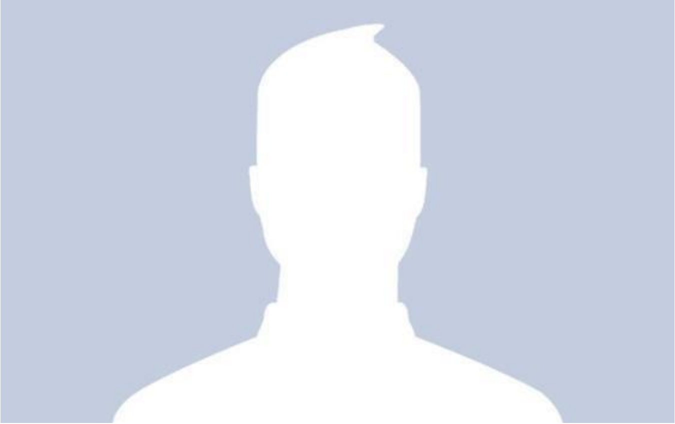
TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ

	
İsim Soyisim	Batu Kaan Özen
Öğrenci Numarası	141201079
Tarih	02/02/2020

Aim of project: In this project, our goal is detecting Uno Card via computer camera.

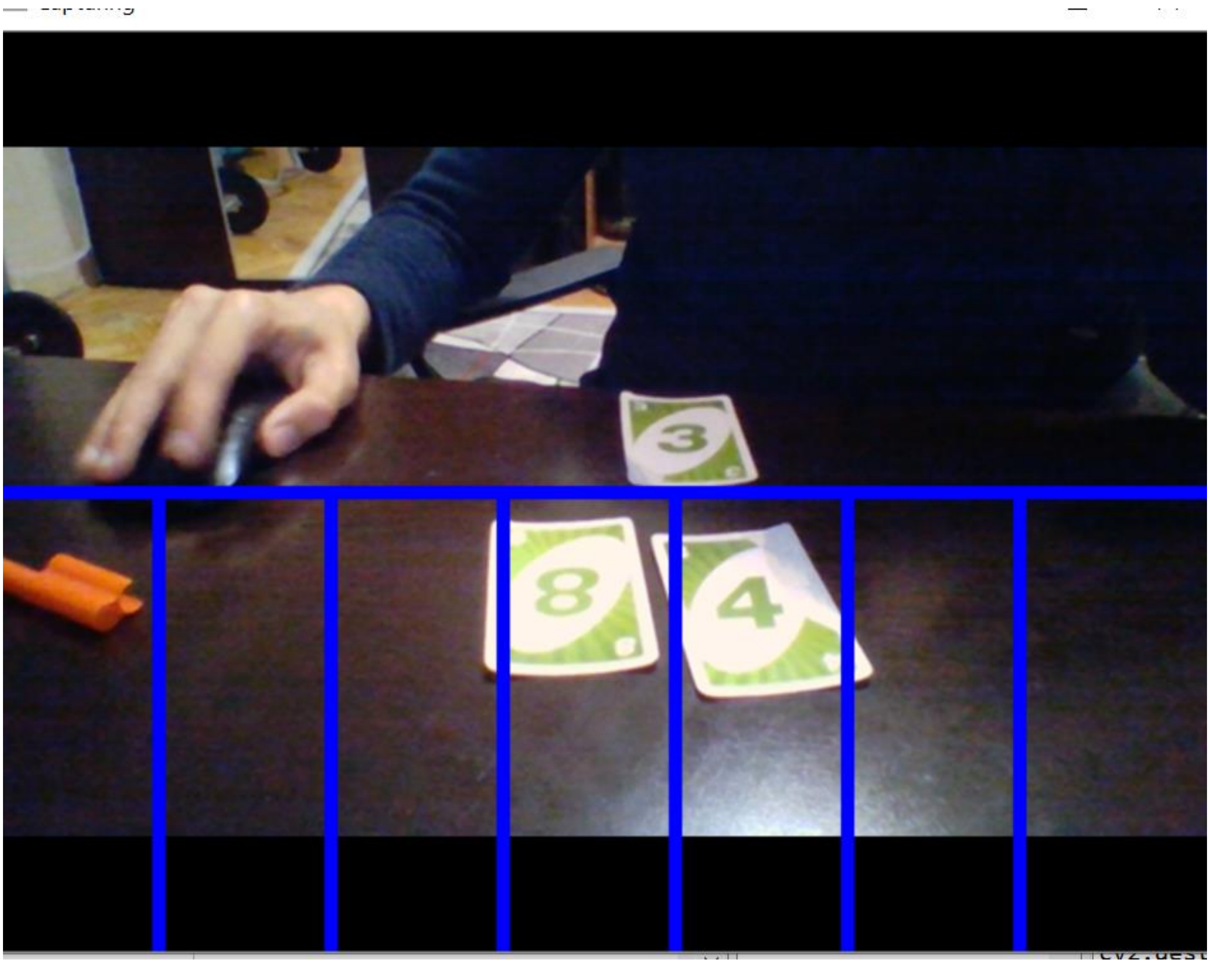
Project Structure:

Our system contains two main parts. First one is Finding uno card on Image and the second part is making decision about card number and card color.

1. Finding Uno Card on Image

Take Image From Camera:

In this part of my code, we took image from camera. You can see this system working in Picture -1- .

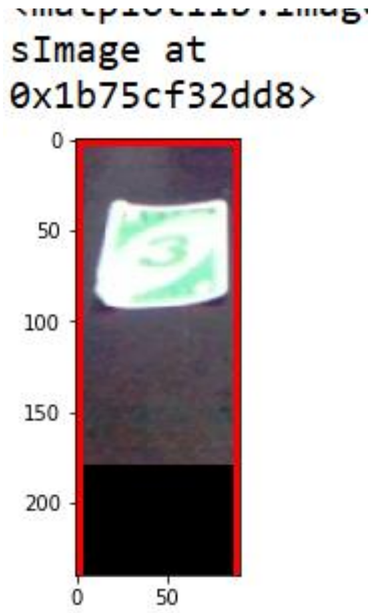


Picture-1-

As you can see in picture-1-. Our frame is divided 8 different areas. Above one is for main area in uno game and other seven small areas for players' hand.

Separate Image To Frames:

After taking Image from camera, our code separate to image 8 different areas. You can see in Picture-2-



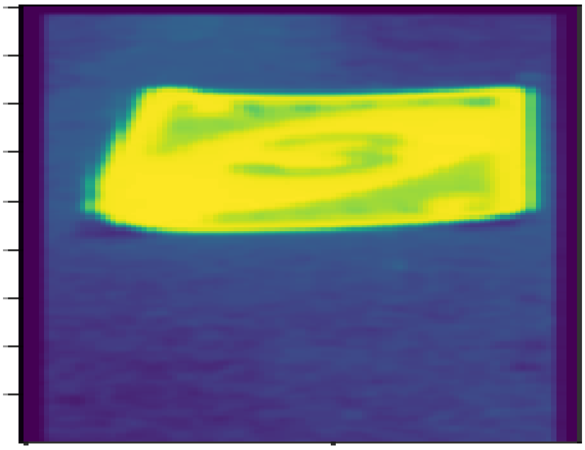
```
<matplotlib.imag
sImage at
0x1b75cf32dd8>
```

Picture-2-

Converting Image to Gray Scale:

Gaussian Blur Algorithm:

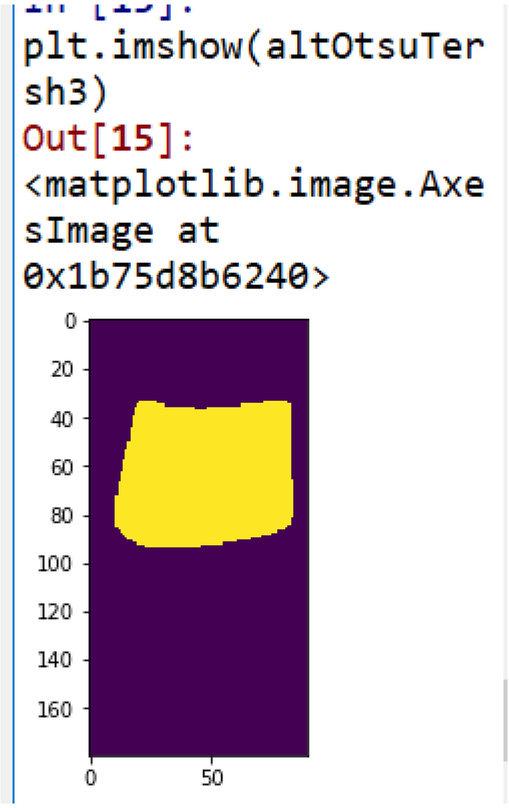
In Gaussian Blur operation, the image is convolved with a Gaussian filter instead of the box filter. The Gaussian filter is a low-pass filter that removes the high-frequency components are reduced.



Picture-3-

Otsu+Binary Treshhold:

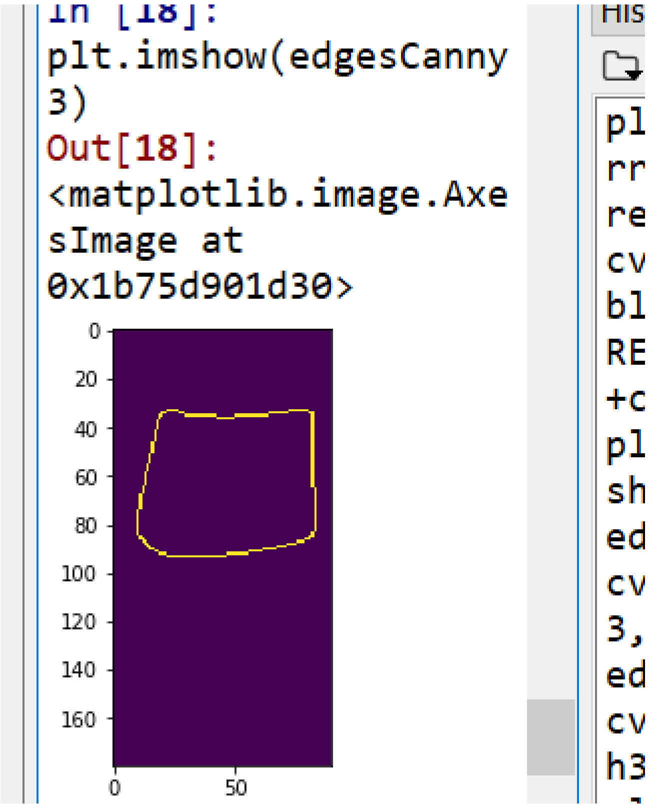
After that Otsu and binary threshold algorithms are applied. You can see result in picture-4- .



```
plt.imshow(altOtsuTersh3)
Out[15]:
<matplotlib.image.AxesImage at
0x1b75d8b6240>
```

Picture-4-

Canny Edge Detector: After thresholding, we applied Canny Edge detection and we detected our systems’ edges . You can see in picture -5-

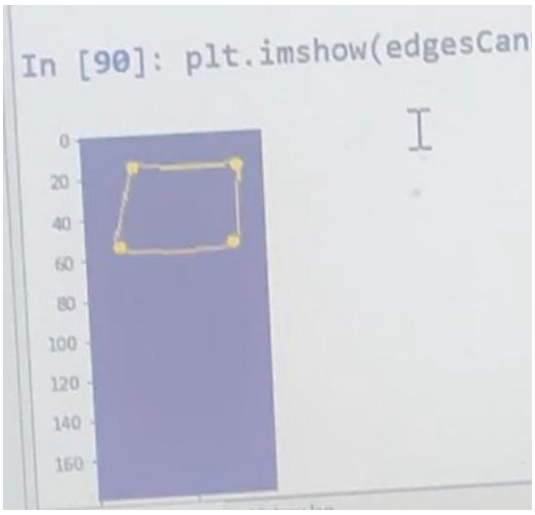


```
In [18]:
plt.imshow(edgesCanny
3)
Out[18]:
<matplotlib.image.Axe
sImage at
0x1b75d901d30>
```

Picture-5-

Corner Detection:

After corner detection, we detected edges via pre-written open cv library (goodFeaturesToTrack). You can see in Picture -6-.

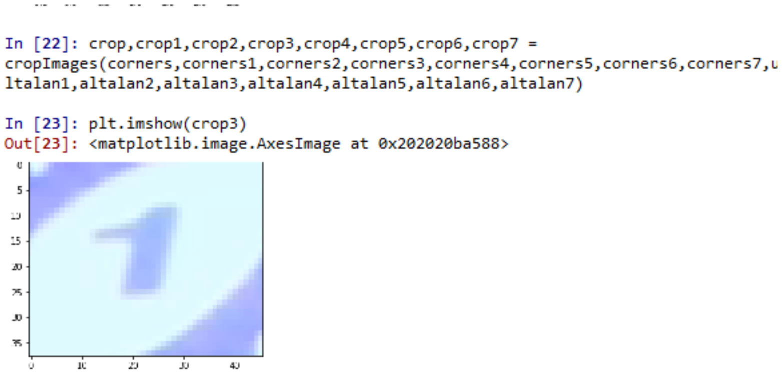


```
In [90]: plt.imshow(edgesCan
```

Picture-6-

Crop Algorithm:

After finding edges on Image , we cropped our image. You can see our result in Picture-7-



```
In [22]: crop,crop1,crop2,crop3,crop4,crop5,crop6,crop7 =
cropImages(corners,corners1,corners2,corners3,corners4,corners5,corners6,corners7,u
ltalan1,altalan2,altalan3,altalan4,altalan5,altalan6,altalan7)

In [23]: plt.imshow(crop3)
Out[23]: <matplotlib.image.AxesImage at 0x202020ba588>
```

Picture-7-

Making Decision about Card

Neural Network Part:

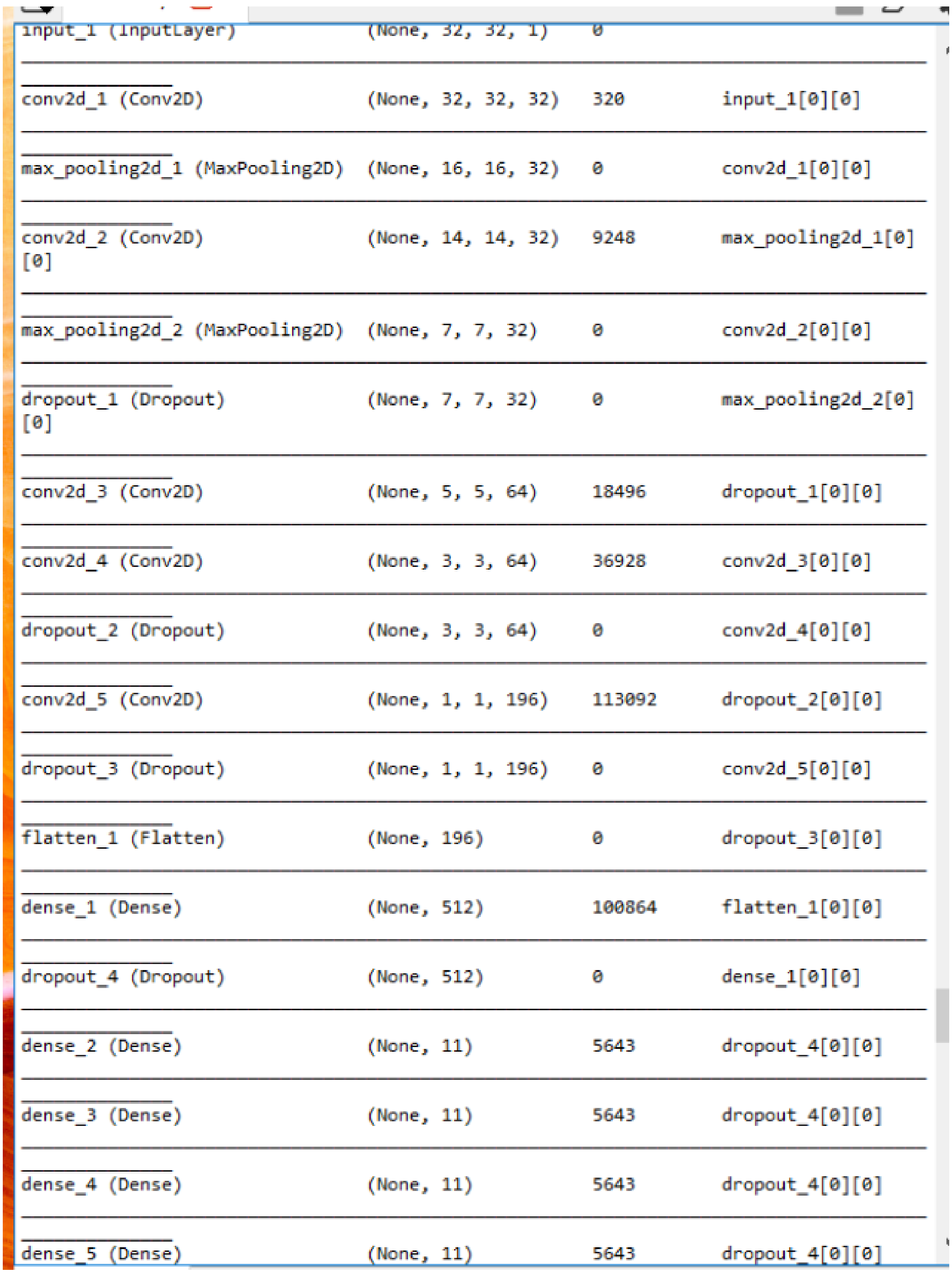
In this part, we used SVHN. Dataset to train our machine learning model for detecting number.

What is SVHN?

The Street View House Numbers (SVHN) is a real-world image dataset used for developing machine learning and object recognition algorithms. It is one of the commonly used benchmark datasets as It requires minimal data preprocessing and formatting. Although it shares some similarities with MNIST where the images are of small cropped digits, SVHN incorporates an order of magnitude more labelled data (over 600,000 digit images). It also comes from a significantly harder real world problem of recognising digits and numbers in natural scene images.

Neural Network Architecture:

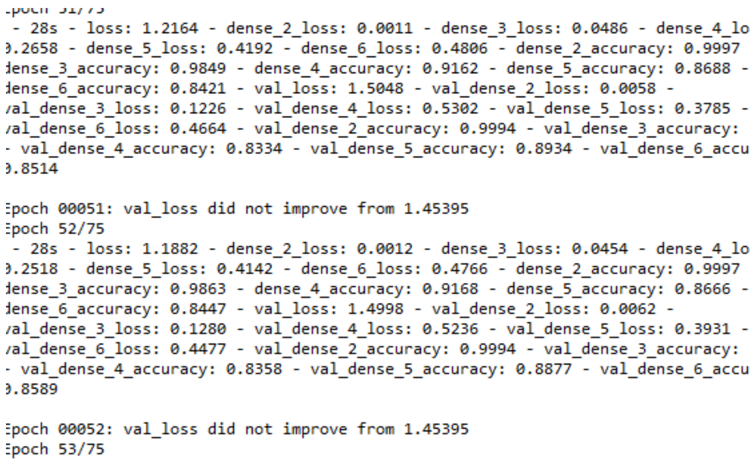
You can see my neural network Architecture in Picuture-7-



```
input_1 (InputLayer)            (None, 32, 32, 1)    0
________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 32, 32, 32)   320         input_1[0][0]
________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 16, 16, 32)   0           conv2d_1[0][0]
________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 14, 14, 32)   9248        max_pooling2d_1[0]
[0]
________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 7, 7, 32)     0           conv2d_2[0][0]
________________________________________________________________________________
dropout_1 (Dropout)             (None, 7, 7, 32)     0           max_pooling2d_2[0]
[0]
________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 5, 5, 64)     18496       dropout_1[0][0]
________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 3, 3, 64)     36928       conv2d_3[0][0]
________________________________________________________________________________
dropout_2 (Dropout)             (None, 3, 3, 64)     0           conv2d_4[0][0]
________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 1, 1, 196)    113092      dropout_2[0][0]
________________________________________________________________________________
dropout_3 (Dropout)             (None, 1, 1, 196)    0           conv2d_5[0][0]
________________________________________________________________________________
flatten_1 (Flatten)             (None, 196)          0           dropout_3[0][0]
________________________________________________________________________________
dense_1 (Dense)                 (None, 512)          100864      flatten_1[0][0]
________________________________________________________________________________
dropout_4 (Dropout)             (None, 512)          0           dense_1[0][0]
________________________________________________________________________________
dense_2 (Dense)                 (None, 11)           5643        dropout_4[0][0]
________________________________________________________________________________
dense_3 (Dense)                 (None, 11)           5643        dropout_4[0][0]
________________________________________________________________________________
dense_4 (Dense)                 (None, 11)           5643        dropout_4[0][0]
________________________________________________________________________________
dense_5 (Dense)                 (None, 11)           5643        dropout_4[0][0]
```

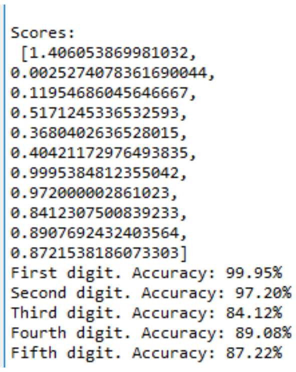
Picuture-7-

After initializing our neural network, we trained with SVHN dataset and applied SVHN test data set. Our result in Test data set is shown in Picuture-9-. In Picuture -8-, you can see our neural network training phase.



```
:poch 51/75
 - 28s - loss: 1.2164 - dense_2_loss: 0.0011 - dense_3_loss: 0.0486 - dense_4_lo
ɜ.2658 - dense_5_loss: 0.4192 - dense_6_loss: 0.4806 - dense_2_accuracy: 0.9997
lense_3_accuracy: 0.9849 - dense_4_accuracy: 0.9162 - dense_5_accuracy: 0.8688 -
lense_6_accuracy: 0.8421 - val_loss: 1.5048 - val_dense_2_loss: 0.0058 -
/al_dense_3_loss: 0.1226 - val_dense_4_loss: 0.5302 - val_dense_5_loss: 0.3785 -
/al_dense_6_loss: 0.4664 - val_dense_2_accuracy: 0.9994 - val_dense_3_accuracy:
- val_dense_4_accuracy: 0.8334 - val_dense_5_accuracy: 0.8934 - val_dense_6_accu
ɜ.8514

:poch 00051: val_loss did not improve from 1.45395
:poch 52/75
 - 28s - loss: 1.1882 - dense_2_loss: 0.0012 - dense_3_loss: 0.0454 - dense_4_lo
ɜ.2518 - dense_5_loss: 0.4142 - dense_6_loss: 0.4766 - dense_2_accuracy: 0.9997
lense_3_accuracy: 0.9863 - dense_4_accuracy: 0.9168 - dense_5_accuracy: 0.8666 -
lense_6_accuracy: 0.8447 - val_loss: 1.4998 - val_dense_2_loss: 0.0062 -
/al_dense_3_loss: 0.1280 - val_dense_4_loss: 0.5236 - val_dense_5_loss: 0.3931 -
/al_dense_6_loss: 0.4477 - val_dense_2_accuracy: 0.9994 - val_dense_3_accuracy:
- val_dense_4_accuracy: 0.8358 - val_dense_5_accuracy: 0.8877 - val_dense_6_accu
ɜ.8589

:poch 00052: val_loss did not improve from 1.45395
:poch 53/75
```

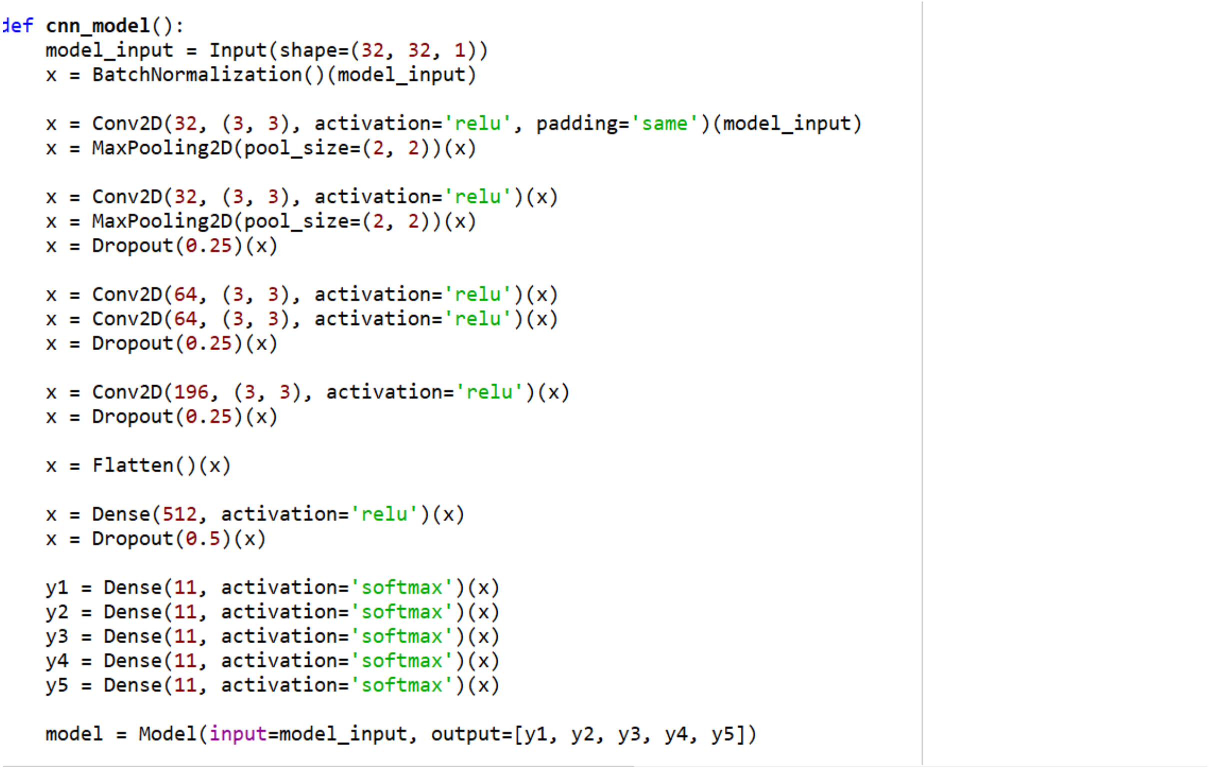
Picture-8-(Training Phase)



```
Scores:
 [1.406053869981032,
0.0025274078361690044,
0.11954686045646667,
0.5171245336532593,
0.3680402636528015,
0.40421172976493835,
0.9995384812355042,
0.972000002861023,
0.8412307500839233,
0.8907692432403564,
0.8721538186073303]
First digit. Accuracy: 99.95%
Second digit. Accuracy: 97.20%
Third digit. Accuracy: 84.12%
Fourth digit. Accuracy: 89.08%
Fifth digit. Accuracy: 87.22%
```

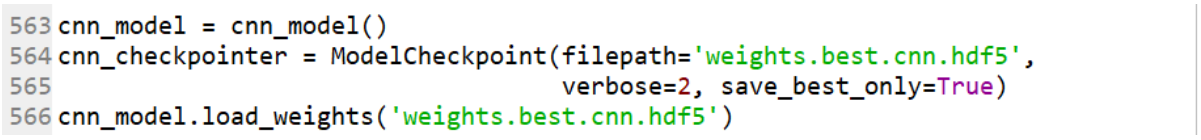
Picture-9- (%90 neural network success on test set)

After training neural network, we saved neural network weigh for avoiding time lose for example training neural network takes 1 hour but if you save the weigh, you can just initialize your model structure and weight and you can start to use your neural network. Because of this situation we have two different code one for image detection system and one for training neural network. Our detection system reads the weight from hdf.5. You can see neural network initializing code in Picture-10 and neural network weight initializing in Picture -11-



```
def cnn_model():
    model_input = Input(shape=(32, 32, 1))
    x = BatchNormalization()(model_input)

    x = Conv2D(32, (3, 3), activation='relu', padding='same')(model_input)
    x = MaxPooling2D(pool_size=(2, 2))(x)

    x = Conv2D(32, (3, 3), activation='relu')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Dropout(0.25)(x)

    x = Conv2D(64, (3, 3), activation='relu')(x)
    x = Conv2D(64, (3, 3), activation='relu')(x)
    x = Dropout(0.25)(x)

    x = Conv2D(196, (3, 3), activation='relu')(x)
    x = Dropout(0.25)(x)

    x = Flatten()(x)

    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)

    y1 = Dense(11, activation='softmax')(x)
    y2 = Dense(11, activation='softmax')(x)
    y3 = Dense(11, activation='softmax')(x)
    y4 = Dense(11, activation='softmax')(x)
    y5 = Dense(11, activation='softmax')(x)

    model = Model(input=model_input, output=[y1, y2, y3, y4, y5])
```

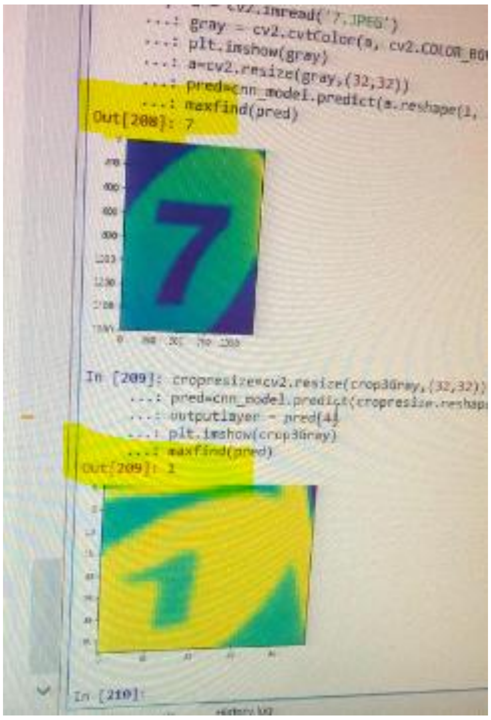
Picture-10-



```
cnn_model = cnn_model()
cnn_checkpointer = ModelCheckpoint(filepath='weights.best.cnn.hdf5',
                                   verbose=2, save_best_only=True)
cnn_model.load_weights('weights.best.cnn.hdf5')
```

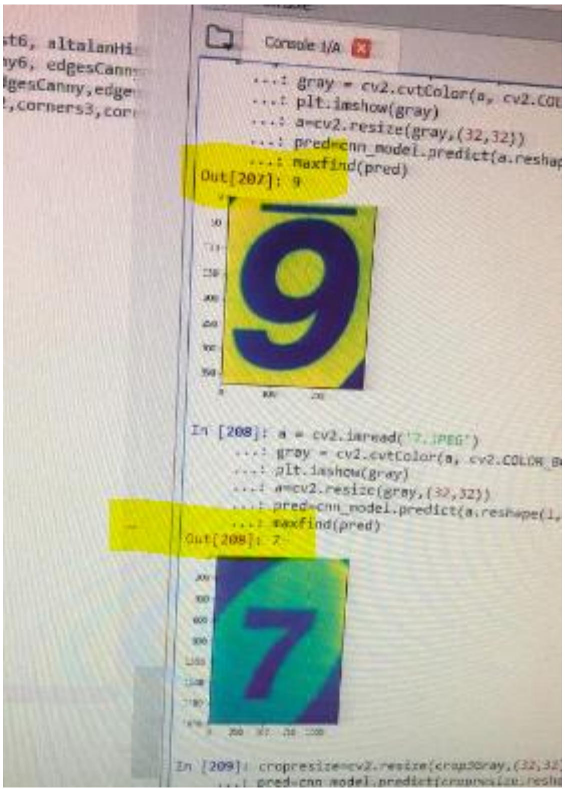
Picture-11-

You can see our neural network prediction examples in picture-12 and picture -13-. Out is our system result



```
   ...: a = cv2.imread('7.JPEG')
   ...: gray = cv2.cvtColor(a, cv2.COLOR_BG
   ...: plt.imshow(gray)
   ...: a=cv2.resize(gray,(32,32))
   ...: pred=cnn_model.predict(a.reshape(1,
   ...: maxfind(pred)
Out[208]: 7

In [209]: cropresize=cv2.resize(crop3Gray,(32,32))
     ...: pred=cnn_model.predict(cropresize.reshap
     ...: outputlayer = pred[4]
     ...: plt.imshow(crop3Gray)
     ...: maxfind(pred)
Out[209]: 1

In [210]:
```

Picutre-13



```
t6, altalanHi
y6, edgesCann
gesCanny,edge
,corners3,cor
```

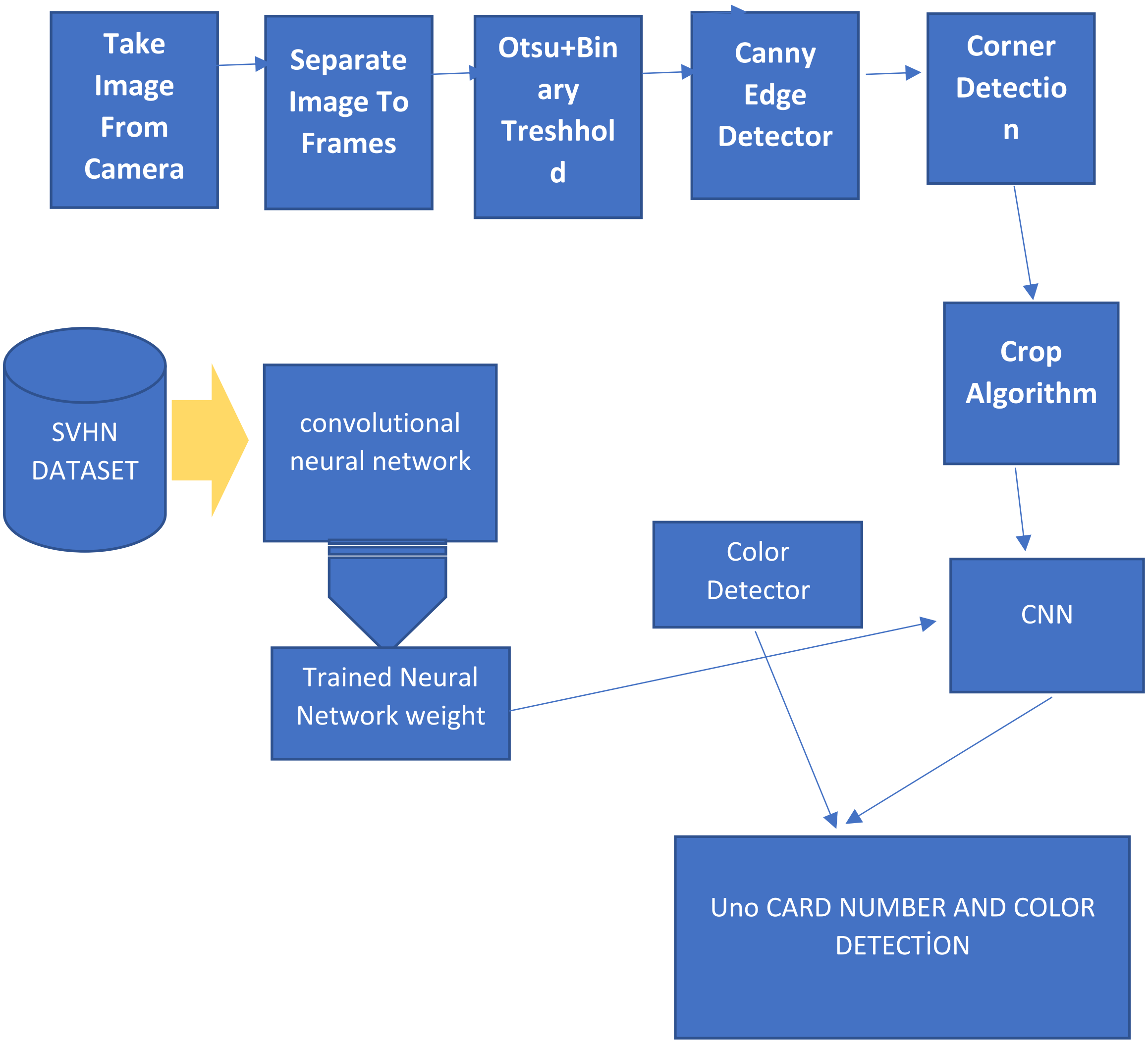
```
   ...: gray = cv2.cvtColor(a, cv2.CO
   ...: plt.imshow(gray)
   ...: a=cv2.resize(gray,(32,32))
   ...: pred=cnn_model.predict(a.resha
   ...: maxfind(pred)
Out[207]: 9

In [208]: a = cv2.imread('7.JPEG')
     ...: gray = cv2.cvtColor(a, cv2.COLOR_B
     ...: plt.imshow(gray)
     ...: a=cv2.resize(gray,(32,32))
     ...: pred=cnn_model.predict(a.reshape(1,
     ...: maxfind(pred)
Out[208]: 7

In [209]: cropresize=cv2.resize(crop3Gray,(32,32
     ...: pred=cnn_model.predict(cropresize.resh
```

Picture-14

Color Detection:

The main principle of color detection is based on HSV values. After cropping image, it is founded the left bottom corner and it is looked the pixel value of the HSV. Then we make decision about our card collor.

Flow Chart of Code:



Resources:

1. https://www.tutorialspoint.com/opencv/opencv_gaussian_blur.htm

2. https://agi.io/2018/01/31/getting-started-street-view-house-numbers-svhn-dataset/

CODES:

You can see Main code(Uno card detector) and CNN code for finding neural network weighs.

Main Code(Uno card Detector):

```
import numpy as np

import cv2

from matplotlib import pyplot as plt

from sklearn.externals import joblib

from sklearn import datasets

from skimage.feature import hog

from sklearn.svm import LinearSVC

import numpy as np

import tensorflow as tf

from keras.models import Sequential

from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D

from sklearn import datasets, svm, metrics

from sklearn.model_selection import train_test_split

# baseline cnn model for mnist

from numpy import mean

from numpy import std

from matplotlib import pyplot

from sklearn.model_selection import KFold
from keras.datasets import mnist

from keras.utils import to_categorical

from keras.models import Sequential

from keras.layers import Conv2D

from keras.layers import MaxPooling2D

from keras.layers import Dense

from keras.layers import Flatten

from keras.optimizers import SGD

import numpy

import pandas

import glob

import matplotlib.pylab as plt

import matplotlib.cm as cm

import warnings

warnings.filterwarnings('ignore')

from keras.models import Sequential, Model

from keras.optimizers import SGD, RMSprop, Adam, Nadam

from keras.callbacks import ModelCheckpoint

from keras.preprocessing.image import ImageDataGenerator

from keras.utils import to_categorical

from keras.layers import Dense, Dropout, LSTM

from keras.layers import Activation, Flatten, Input, BatchNormalization

from keras.layers import Conv1D, MaxPooling1D

from keras.layers import Conv2D, MaxPooling2D

from keras.layers import GlobalAveragePooling2D, GlobalMaxPooling2D


def takePictureFromVideo():

   video = cv2.VideoCapture(0)

   while True:
        check,frame = video.read()

        frame2=frame

        cv2.line(img=frame2, pt1=(0, 240), pt2=(720,240), color=(255, 0, 0), thickness=5, lineType=8, shift=0)

        cv2.line(img=frame2, pt1=(90, 240), pt2=(90,480), color=(255, 0, 0), thickness=5, lineType=8, shift=0)

        cv2.line(img=frame2, pt1=(180, 240), pt2=(180,480), color=(255, 0, 0), thickness=5, lineType=8, shift=0)

        cv2.line(img=frame2, pt1=(270, 240), pt2=(270,480), color=(255, 0, 0), thickness=5, lineType=8, shift=0)

        cv2.line(img=frame2, pt1=(360, 240), pt2=(360,480), color=(255, 0, 0), thickness=5, lineType=8, shift=0)

        cv2.line(img=frame2, pt1=(450, 240), pt2=(450,480), color=(255, 0, 0), thickness=5, lineType=8, shift=0)

        cv2.line(img=frame2, pt1=(540, 240), pt2=(540,480), color=(255, 0, 0), thickness=5, lineType=8, shift=0)

        cv2.imshow('Capturing',frame2)

        key = cv2.waitKey(1)

        if key == ord('q'):

            break

    video.release()

    cv2.destroyAllWindows()

    return frame



def SeperateTwoImage(frame):


    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    ust = frame[0:240,:]

    altalan1 = frame[240:,0:90,:]

    altalan2 = frame[240:,90:180,:]

    altalan3 = frame[240:,180:270,:]
    altalan4 = frame[240:,270:360,:]

    altalan5 = frame[240:,360:450,:]

    altalan6 = frame[240:,450:540,:]

    altalan7 = frame[240:,540:640,:]

    return ust,altalan1,altalan2,altalan3,altalan4,altalan5,altalan6,altalan7


def preprocessing():

   frame = takePictureFromVideo()

   gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

   plt.imshow(gray)

   ust,altalan1,altalan2,altalan3,altalan4,altalan5,altalan6,altalan7 =
SeperateTwoImage(frame)

   gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)


   ustgray = gray[60:240,:]

   altalan1gray = gray[240:420,0:90]

   altalan2gray = gray[240:420,90:180]

   altalan3gray = gray[240:420,180:270]

   altalan4gray = gray[240:420,270:360]

   altalan5gray = gray[240:420,360:450]

   altalan6gray = gray[240:420,450:540]

   altalan7gray = gray[240:420,540:640]


   ustblurred = cv2.GaussianBlur(ustgray,(3,3),0)

   altalanblurred1 = cv2.GaussianBlur(altalan1gray,(3,3),0)

   altalanblurred2 = cv2.GaussianBlur(altalan2gray,(3,3),0)

   altalanblurred3 = cv2.GaussianBlur(altalan3gray,(3,3),0)

   altalanblurred4 = cv2.GaussianBlur(altalan4gray,(3,3),0)

   altalanblurred5 = cv2.GaussianBlur(altalan5gray,(3,3),0)

   altalanblurred6 = cv2.GaussianBlur(altalan6gray,(3,3),0)
```

```
    altalanblurred7 = cv2.GaussianBlur(altalan7gray,(3,3),0)


    ret,ustOtsuTersh =
cv2.threshold(ustblurred,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    ret,altOtsuTersh1 =
cv2.threshold(altalanblurred1,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    ret,altOtsuTersh2 =
cv2.threshold(altalanblurred2,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    ret,altOtsuTersh3 =
cv2.threshold(altalanblurred3,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    ret,altOtsuTersh4 =
cv2.threshold(altalanblurred4,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    ret,altOtsuTersh5 =
cv2.threshold(altalanblurred5,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    ret,altOtsuTersh6 =
cv2.threshold(altalanblurred6,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    ret,altOtsuTersh7 =
cv2.threshold(altalanblurred7,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    return ustOtsuTersh,altOtsuTersh1,altOtsuTersh2,altOtsuTersh3,altOtsuTersh4,
altOtsuTersh5, altOtsuTersh6,
altOtsuTersh7,ust,altalan1,altalan2,altalan3,altalan4,altalan5,altalan6,altalan7



def
edgeDetectorandFindCounter(ustHist,altalanHist1,altalanHist2,altalanHist3,altalanHist4,al
talanHist5,altalanHist6,altalanHist7):
    edgesCanny = cv2.Canny(ustHist,100,200)
    edgesCanny1 =  cv2.Canny(altalanHist1,100,200)
    edgesCanny2 =  cv2.Canny(altalanHist2,100,200)
    edgesCanny3 =  cv2.Canny(altalanHist3,100,200)
    edgesCanny4 =  cv2.Canny(altalanHist4,100,200)
    edgesCanny5 =  cv2.Canny(altalanHist5,100,200)
    edgesCanny6 =  cv2.Canny(altalanHist6,100,200)
    edgesCanny7 =  cv2.Canny(altalanHist7,100,200)
    return edgesCanny, edgesCanny1, edgesCanny2, edgesCanny3, edgesCanny4, edgesCanny5, edgesCanny6, edgesCanny7



def findCorners(edgesCanny,edgesCanny1, edgesCanny2, edgesCanny3, edgesCanny4, edgesCanny5, edgesCanny6, edgesCanny7):
    corners = cv2.goodFeaturesToTrack(edgesCanny,4,0.001,10)
    corners = np.int0(corners)
    corners1 = cv2.goodFeaturesToTrack(edgesCanny1,4,0.001,10)
    corners1 = np.int0(corners1)
    corners2 = cv2.goodFeaturesToTrack(edgesCanny2,4,0.001,10)
    corners2 = np.int0(corners2)
    corners3 = cv2.goodFeaturesToTrack(edgesCanny3,4,0.001,10)
    corners3 = np.int0(corners3)
    corners4 = cv2.goodFeaturesToTrack(edgesCanny4,4,0.001,10)
    corners4 = np.int0(corners4)
    corners5 = cv2.goodFeaturesToTrack(edgesCanny5,4,0.001,10)
    corners5 = np.int0(corners5)
    corners6 = cv2.goodFeaturesToTrack(edgesCanny6,4,0.001,10)
    corners6 = np.int0(corners6)
    corners7 = cv2.goodFeaturesToTrack(edgesCanny7,4,0.001,10)
    corners7 = np.int0(corners7)
    if (np.size(corners)!=8):
        corners=np.zeros((4,2))
    if (np.size(corners1)!=8):
        corners1=np.zeros((4,2))
    if (np.size(corners2)!=8):
        corners2=np.zeros((4,2))
    if (np.size(corners3)!=8):
        corners3=np.zeros((4,2))
    if (np.size(corners4)!=8):
        corners1=np.zeros((4,2))
    if (np.size(corners5)!=8):
        corners5=np.zeros((4,2))
    if (np.size(corners6)!=8):
        corners6=np.zeros((4,2))
    if (np.size(corners7)!=8):
        corners7=np.zeros((4,2))

    return corners,corners1,corners2,corners3,corners4,corners5,corners6,corners7



def cropImages(corners,corners1,corners2,corners3,corners4,corners5,corners6,corners7,ust,altalan1,altalan2,altalan3,altalan4,altalan5,altalan6,altalan7):


    x = np.zeros((4,1))
    y = np.zeros((4,1))
    c=0
    for i in (0,2,4,6):
        a=corners3.item(i)
        x[c,0]=corners3.item(i)
        y[c,0]=corners3.item(i+1)
        c=c+1
    xmax = int(np.max(x))
    xmin = int(np.min(x))
    ymax = int(np.max(y))
    ymin = int(np.min(y))
    aramesafex = int((xmax-xmin)/4)
    aramesafey = int((ymax-ymin)/8)
    crop3 = altalan3[xmin+aramesafex:xmax-aramesafex, ymin+aramesafey:ymax-aramesafey]

    x = np.zeros((4,1))
    y = np.zeros((4,1))
    c=0
    for i in (0,2,4,6):
        a=corners2.item(i)
        x[c,0]=corners2.item(i)
        y[c,0]=corners2.item(i+1)
        c=c+1
    xmax = int(np.max(x))
    xmin = int(np.min(x))
    ymax = int(np.max(y))
    ymin = int(np.min(y))
    aramesafex = int((xmax-xmin)/4)
    aramesafey = int((ymax-ymin)/8)
    crop2 = altalan2[xmin+aramesafex:xmax-aramesafex, ymin+aramesafey:ymax-aramesafey]

    x = np.zeros((4,1))
    y = np.zeros((4,1))
    c=0
    for i in (0,2,4,6):
        a=corners1.item(i)
        x[c,0]=corners1.item(i)
        y[c,0]=corners1.item(i+1)
        c=c+1
    xmax = int(np.max(x))
    xmin = int(np.min(x))
    ymax = int(np.max(y))
    ymin = int(np.min(y))
    aramesafex = int((xmax-xmin)/4)
    aramesafey = int((ymax-ymin)/8)
    crop1 = altalan1[xmin+aramesafex:xmax-aramesafex, ymin+aramesafey:ymax-aramesafey]

    x = np.zeros((4,1))
    y = np.zeros((4,1))
    c=0
    for i in (0,2,4,6):
        a=corners4.item(i)
        x[c,0]=corners4.item(i)
        y[c,0]=corners4.item(i+1)
        c=c+1
    xmax = int(np.max(x))
    xmin = int(np.min(x))
    ymax = int(np.max(y))
    ymin = int(np.min(y))
    aramesafex = int((xmax-xmin)/4)
    aramesafey = int((ymax-ymin)/8)
    crop4 = altalan4[xmin+aramesafex:xmax-aramesafex, ymin+aramesafey:ymax-aramesafey]

    x = np.zeros((4,1))
    y = np.zeros((4,1))
    c=0
    for i in (0,2,4,6):
        a=corners5.item(i)
        x[c,0]=corners5.item(i)
        y[c,0]=corners5.item(i+1)
        c=c+1
    xmax = int(np.max(x))
    xmin = int(np.min(x))
    ymax = int(np.max(y))
    ymin = int(np.min(y))
    aramesafex = int((xmax-xmin)/4)
    aramesafey = int((ymax-ymin)/8)
    crop5 = altalan5[xmin+aramesafex:xmax-aramesafex, ymin+aramesafey:ymax-aramesafey]

    x = np.zeros((4,1))
    y = np.zeros((4,1))
    c=0
    for i in (0,2,4,6):
        a=corners6.item(i)
        x[c,0]=corners6.item(i)
        y[c,0]=corners6.item(i+1)
        c=c+1
    xmax = int(np.max(x))
    xmin = int(np.min(x))
    ymax = int(np.max(y))
    ymin = int(np.min(y))
    aramesafex = int((xmax-xmin)/4)
    aramesafey = int((ymax-ymin)/8)
    crop6 = altalan6[xmin+aramesafex:xmax-aramesafex, ymin+aramesafey:ymax-aramesafey]
```

```
x = np.zeros((4,1))
y = np.zeros((4,1))
c=0

for i in (0,2,4,6):
    a=corners.item(i)
    y[c,0]=corners.item(i)
    x[c,0]=corners.item(i+1)
    c=c+1

xmax = int(np.max(x))
xmin = int(np.min(x))
ymax = int(np.max(y))
ymin = int(np.min(y))
aramesafex = int((xmax-xmin)/4)
aramesafey = int((ymax-ymin)/8)
crop = ust[xmin+aramesafex:xmax-aramesafex, ymin+aramesafey:ymax-aramesafey]

x = np.zeros((4,1))
y = np.zeros((4,1))
c=0
for i in (0,2,4,6):
    x[c,0]=corners7.item(i)
    y[c,0]=corners7.item(i+1)
    c=c+1
xmax = int(np.max(x))
xmin = int(np.min(x))
ymax = int(np.max(y))
ymin = int(np.min(y))
```

```
    aramesafex = int((xmax-xmin)/4)
    aramesafey = int((ymax-ymin)/8)
    crop7 = altalan7[xmin+aramesafex:xmax-aramesafex, ymin+aramesafey:ymax-aramesafey]



    return crop,crop1,crop2,crop3,crop4,crop5,crop6,crop7
#color Matrix 1 for  2 for red  3 for  4 for blue
color = np.zeros((8,1))


def colorfind(corners,corners1,corners2,corners3,corners4,corners5,corners6,corners7,ust,altalan1,altalan2,altalan3,altalan4,altalan5,altalan6,altalan7):
    x = np.zeros((4,1))
    y = np.zeros((4,1))
    c=0
    for i in (0,2,4,6):
        x[c,0]=corners3.item(i)
        y[c,0]=corners3.item(i+1)
        c=c+1
    xmax = int(np.max(x))
    xmin = int(np.min(x))
    ymax = int(np.max(y))
    ymin = int(np.min(y))
    crop3 = altalan3[xmin+2:xmin+5, ymin+2:ymin+5]
    point = crop3[:][2][2]
    hsv = cv2.cvtColor(crop3, cv2.COLOR_BGR2HSV)
    point = hsv[:][2][2]
    if (point[0]<20):
        color[4]=1
```

```
if (20<point[0] and point[0]<70):
    color[4]=2
if (70<point[0] and point[0]<155 ):
    color[4]=3
if (155<point[0]):
    color[4]=4



x = np.zeros((4,1))
y = np.zeros((4,1))
c=0
for i in (0,2,4,6):
    x[c,0]=corners.item(i)
    y[c,0]=corners.item(i+1)
    c=c+1
xmax = int(np.max(x))
xmin = int(np.min(x))
ymax = int(np.max(y))
ymin = int(np.min(y))
crop = ust[xmin+2:xmin+5, ymin+2:ymin+5]
point = crop[:][2][2]
hsv = cv2.cvtColor(crop3, cv2.COLOR_BGR2HSV)
point = hsv[:][2][2]
if (point[0]<20):
    color[0]=1
if (20<point[0] and point[0]<70):
    color[0]=2
if (70<point[0] and point[0]<155 ):
    color[0]=3
```

```
if (155<point[0]):
    color[0]=4


x = np.zeros((4,1))
y = np.zeros((4,1))
c=0
for i in (0,2,4,6):
    x[c,0]=corners1.item(i)
    y[c,0]=corners1.item(i+1)
    c=c+1
xmax = int(np.max(x))
xmin = int(np.min(x))
ymax = int(np.max(y))
ymin = int(np.min(y))
crop1 = altalan1[xmin+2:xmin+5, ymin+2:ymin+5]
point = crop1[:][2][2]
hsv = cv2.cvtColor(crop1, cv2.COLOR_BGR2HSV)
point = hsv[:][2][2]
if (point[0]<20):
    color[1]=1
if (20<point[0] and point[0]<70):
    color[1]=2
if (70<point[0] and point[0]<155 ):
    color[1]=3
if (155<point[0]):
    color[1]=4



x = np.zeros((4,1))
```

```
y = np.zeros((4,1))
c=0
for i in (0,2,4,6):
   x[c,0]=corners2.item(i)
   y[c,0]=corners2.item(i+1)
   c=c+1

xmax = int(np.max(x))
xmin = int(np.min(x))
ymax = int(np.max(y))
ymin = int(np.min(y))
crop2 = altalan2[xmin+2:xmin+5, ymin+2:ymin+5]
point = crop2[:][2][2]
hsv = cv2.cvtColor(crop2, cv2.COLOR_BGR2HSV)
point = hsv[:][2][2]
if (point[0]<20):
   color[2]=1
if (20<point[0] and point[0]<70):
   color[2]=2
if (70<point[0] and point[0]<155 ):
   color[2]=3
if (155<point[0]):
   color[2]=4

x = np.zeros((4,1))
y = np.zeros((4,1))
c=0
for i in (0,2,4,6):
   x[c,0]=corners4.item(i)
```

```
    y[c,0]=corners4.item(i+1)
    c=c+1
xmax = int(np.max(x))
xmin = int(np.min(x))
ymax = int(np.max(y))
ymin = int(np.min(y))
crop4 = altalan4[xmin+2:xmin+5, ymin+2:ymin+5]
point = crop4[:][2][2]
hsv = cv2.cvtColor(crop4, cv2.COLOR_BGR2HSV)
point = hsv[:][2][2]
if (point[0]<20):
    color[4]=1
if (20<point[0] and point[0]<70):
    color[4]=2
if (70<point[0] and point[0]<155 ):
    color[4]=3
if (155<point[0]):
    color[4]=4

x = np.zeros((4,1))
y = np.zeros((4,1))
c=0
for i in (0,2,4,6):
    x[c,0]=corners5.item(i)
    y[c,0]=corners5.item(i+1)
    c=c+1
xmax = int(np.max(x))
xmin = int(np.min(x))
ymax = int(np.max(y))
```

```
ymin = int(np.min(y))

crop5 = altalan5[xmin+2:xmin+5, ymin+2:ymin+5]

point = crop5[:][2][2]

hsv = cv2.cvtColor(crop5, cv2.COLOR_BGR2HSV)

point = hsv[:][2][2]

if (point[0]<20):

   color[5]=1

if (20<point[0] and point[0]<70):

   color[5]=2

if (70<point[0] and point[0]<155 ):

   color[5]=3

if (155<point[0]):

   color[5]=4



x = np.zeros((4,1))

y = np.zeros((4,1))

c=0

for i in (0,2,4,6):

   x[c,0]=corners6.item(i)

   y[c,0]=corners6.item(i+1)

   c=c+1

xmax = int(np.max(x))

xmin = int(np.min(x))

ymax = int(np.max(y))

ymin = int(np.min(y))

crop6 = altalan6[xmin+2:xmin+5, ymin+2:ymin+5]

point = crop5[:][2][2]

hsv = cv2.cvtColor(crop6, cv2.COLOR_BGR2HSV)
```

```
point = hsv[:][2][2]
if (point[0]<20):
   color[6]=1
if (20<point[0] and point[0]<70):
   color[6]=2
if (70<point[0] and point[0]<155 ):
   color[6]=3
if (155<point[0]):
   color[6]=4

x = np.zeros((4,1))
y = np.zeros((4,1))
c=0
for i in (0,2,4,6):
   x[c,0]=corners7.item(i)
   y[c,0]=corners7.item(i+1)
   c=c+1
xmax = int(np.max(x))
xmin = int(np.min(x))
ymax = int(np.max(y))
ymin = int(np.min(y))
crop7 = altalan7[xmin+2:xmin+5, ymin+2:ymin+5]
point = crop7[:][2][2]
hsv = cv2.cvtColor(crop7, cv2.COLOR_BGR2HSV)
point = hsv[:][2][2]
if (point[0]<20):
   color[7]=1
if (20<point[0] and point[0]<70):
   color[7]=2
```

```
    if (70<point[0] and point[0]<155 ):
        color[7]=3
    if (155<point[0]):
        color[7]=4



    return color



def cnn_model():
    model_input = Input(shape=(32, 32, 1))
    x = BatchNormalization()(model_input)

    x = Conv2D(32, (3, 3), activation='relu', padding='same')(model_input)
    x = MaxPooling2D(pool_size=(2, 2))(x)

    x = Conv2D(32, (3, 3), activation='relu')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Dropout(0.25)(x)

    x = Conv2D(64, (3, 3), activation='relu')(x)
    x = Conv2D(64, (3, 3), activation='relu')(x)
    x = Dropout(0.25)(x)

    x = Conv2D(196, (3, 3), activation='relu')(x)
    x = Dropout(0.25)(x)

    x = Flatten()(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)

    y1 = Dense(11, activation='softmax')(x)
    y2 = Dense(11, activation='softmax')(x)
    y3 = Dense(11, activation='softmax')(x)
    y4 = Dense(11, activation='softmax')(x)
    y5 = Dense(11, activation='softmax')(x)

    model = Model(input=model_input, output=[y1, y2, y3, y4, y5])

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

def maxfind(pred):
    outputlayer = pred[4][0]
    maxpossibilty = np.max(outputlayer)
    prediction=0
    for i in range(10):
        a=outputlayer[i]
        if (maxpossibilty==a):
            prediction = i
    return prediction



ustHist,altalanHist1,altalanHist2,altalanHist3,altalanHist4, altalanHist5, altalanHist6,
altalanHist7,ust,altalan1,altalan2,altalan3,altalan4,altalan5,altalan6,altalan7 =
preprocessing()
```

```
edgesCanny,edgesCanny1, edgesCanny2, edgesCanny3, edgesCanny4, edgesCanny5, edgesCanny6, edgesCanny7 = edgeDetectorandFindCounter(ustHist,altalanHist1,altalanHist2,altalanHist3,altalanHist4,altalanHist5,altalanHist6,altalanHist7)

corners,corners1,corners2,corners3,corners4,corners5,corners6,corners7=findCorners(edgesCanny,edgesCanny1, edgesCanny2, edgesCanny3, edgesCanny4, edgesCanny5, edgesCanny6, edgesCanny7)

crop,crop1,crop2,crop3,crop4,crop5,crop6,crop7 = cropImages(corners,corners1,corners2,corners3,corners4,corners5,corners6,corners7,ust,altalan1,altalan2,altalan3,altalan4,altalan5,altalan6,altalan7)


#cropGray =   cv2.cvtColor(crop, cv2.COLOR_BGR2GRAY)
#crop1Gray =   cv2.cvtColor(crop1, cv2.COLOR_BGR2GRAY)
#crop2Gray =   cv2.cvtColor(crop2, cv2.COLOR_BGR2GRAY)
crop3Gray =   cv2.cvtColor(crop3, cv2.COLOR_BGR2GRAY)
#crop4Gray =   cv2.cvtColor(crop4, cv2.COLOR_BGR2GRAY)
#crop5Gray =   cv2.cvtColor(crop5, cv2.COLOR_BGR2GRAY)
#crop6Gray =   cv2.cvtColor(crop6, cv2.COLOR_BGR2GRAY)
#crop7Gray =   cv2.cvtColor(crop7, cv2.COLOR_BGR2GRAY)



cnn_model = cnn_model()
cnn_checkpointer = ModelCheckpoint(filepath='weights.best.cnn.hdf5',
                               verbose=2, save_best_only=True)
cnn_model.load_weights('weights.best.cnn.hdf5')
results = np.zeros((8,1))



#cropresize=cv2.resize(cropGray,(32,32))
#pred=cnn_model.predict(cropresize.reshape(1, 32, 32, 1))
#outputlayer = pred[4]
#results[0]=maxfind(pred)
```

```
#cropresize=cv2.resize(crop1Gray,(32,32))

#pred=cnn_model.predict(cropresize.reshape(1, 32, 32, 1))

#outputlayer = pred[4]

#results[1]=maxfind(pred)


#cropresize=cv2.resize(crop2Gray,(32,32))

#pred=cnn_model.predict(cropresize.reshape(1, 32, 32, 1))

#outputlayer = pred[4]

#results[2]=maxfind(pred)



cropresize=cv2.resize(crop3Gray,(32,32))

pred=cnn_model.predict(cropresize.reshape(1, 32, 32, 1))

outputlayer = pred[4]

results[3]=maxfind(pred)

results[3]

plt.imshow(crop3Gray)

results[3]

#cropresize=cv2.resize(crop4Gray,(32,32))

#pred=cnn_model.predict(cropresize.reshape(1, 32, 32, 1))

#outputlayer = pred[4]

#results[4]=maxfind(pred)


#cropresize=cv2.resize(crop5Gray,(32,32))

#pred=cnn_model.predict(cropresize.reshape(1, 32, 32, 1))

#outputlayer = pred[4]

#results[5]=maxfind(pred)
```

```
#cropresize=cv2.resize(crop6Gray,(32,32))
#pred=cnn_model.predict(cropresize.reshape(1, 32, 32, 1))
#outputlayer = pred[4]
#results[6]=maxfind(pred)



#cropresize=cv2.resize(crop7Gray,(32,32))
#pred=cnn_model.predict(cropresize.reshape(1, 32, 32, 1))
#outputlayer = pred[4]
#results[7]=maxfind(pred)


#color=colorfind(corners,corners1,corners2,corners3,corners4,corners5,corners6,corners7,ust,altalan1,altalan2,altalan3,altalan4,altalan5,altalan6,altalan7)
```

Code for training CNN:

```
import numpy
import pandas

import glob

import matplotlib.pylab as plt
import matplotlib.cm as cm

import warnings
warnings.filterwarnings('ignore')
from keras.models import Sequential, Model
from keras.optimizers import SGD, RMSprop, Adam, Nadam
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
from keras.layers import Dense, Dropout, LSTM
from keras.layers import Activation, Flatten, Input, BatchNormalization
from keras.layers import Conv1D, MaxPooling1D
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import GlobalAveragePooling2D, GlobalMaxPooling2D

def digit_to_categorical(data):
    n = data.shape[1]
    data_cat = numpy.empty([len(data), n, 11])
    for i in range(n):
        data_cat[:, i] = to_categorical(data[:, i], num_classes=11)
    return data_cat


def cnn_model():
    model_input = Input(shape=(32, 32, 1))
    x = BatchNormalization()(model_input)

    x = Conv2D(32, (3, 3), activation='relu', padding='same')(model_input)
    x = MaxPooling2D(pool_size=(2, 2))(x)

    x = Conv2D(32, (3, 3), activation='relu')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Dropout(0.25)(x)

    x = Conv2D(64, (3, 3), activation='relu')(x)
    x = Conv2D(64, (3, 3), activation='relu')(x)
    x = Dropout(0.25)(x)

    x = Conv2D(196, (3, 3), activation='relu')(x)
    x = Dropout(0.25)(x)

    x = Flatten()(x)

    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)

    y1 = Dense(11, activation='softmax')(x)
    y2 = Dense(11, activation='softmax')(x)
    y3 = Dense(11, activation='softmax')(x)
    y4 = Dense(11, activation='softmax')(x)
    y5 = Dense(11, activation='softmax')(x)

    model = Model(input=model_input, output=[y1, y2, y3, y4, y5])

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model


train_images = pandas.read_csv('housenumbers/train_images.csv')
train_labels = pandas.read_csv('housenumbers/train_labels.csv')
test_images = pandas.read_csv('housenumbers/test_images.csv')
test_labels = pandas.read_csv('housenumbers/test_labels.csv')
extra_images = pandas.read_csv('housenumbers/extra_images.csv')
extra_labels = pandas.read_csv('housenumbers/extra_labels.csv')

train_images.ix[:10,:10]
train_labels.ix[:10,:]
train_images = train_images.ix[:,1:].as_matrix().astype('float32')
train_labels = train_labels.ix[:,1:].as_matrix().astype('int16')
```

```
test_images = test_images.ix[:,1:].as_matrix().astype('float32')
test_labels = test_labels.ix[:,1:].as_matrix().astype('int16')

extra_images = extra_images.ix[:,1:].as_matrix().astype('float32')
extra_labels = extra_labels.ix[:,1:].as_matrix().astype('int16')
print('Label: ', train_labels[100])
plt.imshow(train_images[100].reshape(32,32), cmap=plt.cm.bone)

x_train = numpy.concatenate((train_images.reshape(-1, 32, 32, 1),
                test_images.reshape(-1, 32, 32, 1)),
               axis=0)
y_train = numpy.concatenate((digit_to_categorical(train_labels),
                digit_to_categorical(test_labels)),
               axis=0)

x_valid = extra_images.reshape(-1, 32, 32, 1)
y_valid = digit_to_categorical(extra_labels)

n = int(len(x_valid)/2)
x_test, y_test = x_valid[:n], y_valid[:n]
x_valid, y_valid = x_valid[n:], y_valid[n:]

x_train.shape, x_test.shape, x_valid.shape, \
y_train.shape, y_test.shape, y_valid.shape

y_train_list = [y_train[:, i] for i in range(5)]
y_test_list = [y_test[:, i] for i in range(5)]
y_valid_list = [y_valid[:, i] for i in range(5)]

cnn_model = cnn_model()
cnn_checkpointer = ModelCheckpoint(filepath='weights.best.cnn.hdf5',
                              verbose=2, save_best_only=True)
cnn_history = cnn_model.fit(x_train, y_train_list,
                        validation_data=(x_valid, y_valid_list),
                        epochs=75, batch_size=128, verbose=2,
                        callbacks=[cnn_checkpointer])

cnn_model.load_weights('weights.best.cnn.hdf5')
cnn_scores = cnn_model.evaluate(x_test, y_test_list, verbose=0)

print("CNN Model 1. \n")
print("Scores: \n" , (cnn_scores))
print("First digit. Accuracy: %.2f%%" % (cnn_scores[6]*100))
print("Second digit. Accuracy: %.2f%%" % (cnn_scores[7]*100))
print("Third digit. Accuracy: %.2f%%" % (cnn_scores[8]*100))
print("Fourth digit. Accuracy: %.2f%%" % (cnn_scores[9]*100))
print("Fifth digit. Accuracy: %.2f%%" % (cnn_scores[10]*100))

print(cnn_model.summary())

plt.figure(figsize=(14, 7))

plt.plot(cnn_history.history['val_dense_2_acc'][35:], label = 'First digit')
plt.plot(cnn_history.history['val_dense_3_acc'][35:], label = 'Second digit')
plt.plot(cnn_history.history['val_dense_4_acc'][35:], label = 'Third digit')
plt.plot(cnn_history.history['val_dense_5_acc'][35:], label = 'Fourth digit')
plt.plot(cnn_history.history['val_dense_6_acc'][35:], label = 'Fifth digit')

plt.legend()
plt.title('Accuracy');

a = cv2.imread('7.JPEG')
```

```
gray = cv2.cvtColor(a, cv2.COLOR_BGR2GRAY)

plt.imshow(gray)

a=cv2.resize(gray,(32,32))

pred=cnn_model.predict(a.reshape(1, 32, 32, 1))
```