

# Prepoznavanje saobraćajnih znakova

Admir Subašić, Ajdin Ločić i Haris Dajić

December 21, 2018

GitHub url:

[https://github.com/hdajic/traffic\\_sign\\_recognition\\_poos/tree/drugiKraj](https://github.com/hdajic/traffic_sign_recognition_poos/tree/drugiKraj)

1. Izbor modela za prepoznavanje koji odgovara problemu.

Kao model za prepoznavanje odabrali smo *K susjedni klasifikator* (KNeighborsClassifier). Primjeri treninga su vektori u multidimenzionalnom prostoru funkcija, svaka sa oznakom klase. Faza treninga algoritma sastoji se samo od čuvanja vektora funkcija i oznaka klase uzoraka obuke.

U fazi klasifikacije,  $k$  je korisnički definisana konstanta, a neoblježeni vektor (upita ili testna tačka) se klasifikuje dodjeljivanjem etikete koja je najčešća među uzorcima obuke najbližih tački upita.

2. Izbor deskriptora koji odgovara problemu.

Za deskriptor ćemo koristiti *histogram boja* (Color Histogram). Histogram u boji fokusira se samo na procenat broja različitih tipova boja, bez obzira na prostornu lokaciju boja.

Vrednosti histograma boje su iz statistike. Prikazuju statističku distribuciju boja i bitni ton slike.

Koristili smo ovaj histogram iz razloga što se znakovi razlikuju po boji, dok im je oblik isti.

3. Izbor metoda poboljšavanja iz 1. Projektnog zadatka koje će biti primijenjene nad slikama.

Metode koje smo koristili za poboljšavanje performansi modela jesu:

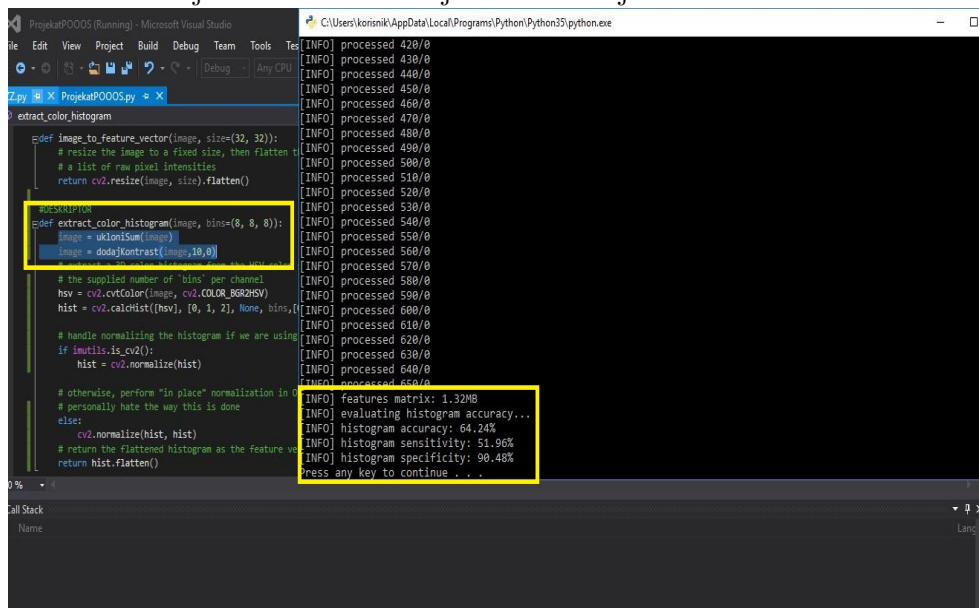
- ukloniSum
- dodajKontrast

5. Poboljšavanje performansi modela za prepoznavanje na osnovu performansi testiranja modela - primijeniti neku od metoda i demonstrirati postignuti rezultat ponovnim testiranjem: (D) -dokumentovati sve isprobane načine poboljšavanja

1. Izmjena parametara odgovarajućeg modela
2. Drugačija podjela podataka na trening/test skup
3. Izbacivanje outlier-a slika
4. Primjena drugih metoda poboljšavanja
5. Isl.

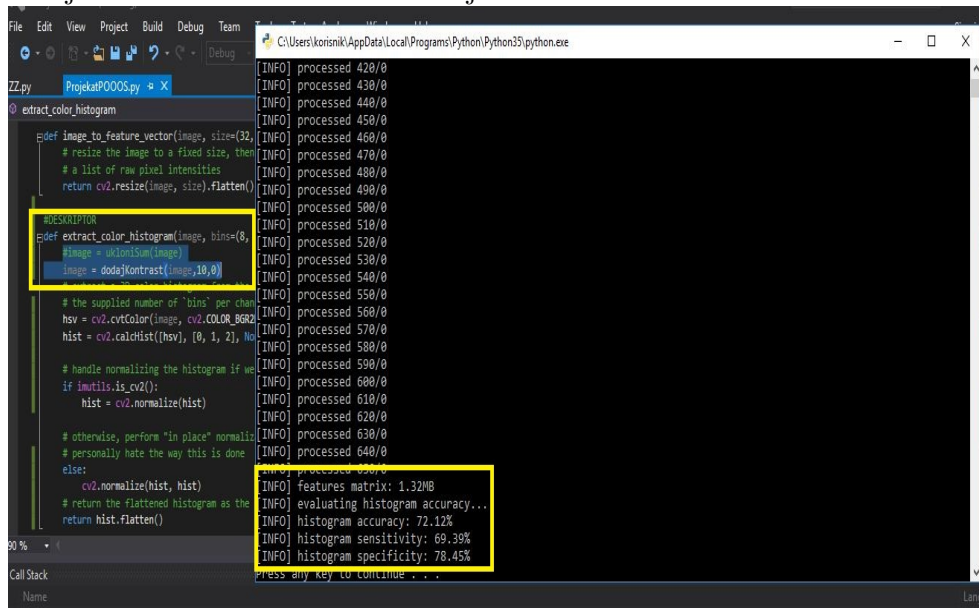
U nastavku ćemo nabrojati koje smo promjene koristili nad našim modelom:

1. U ovom slučaju smo koristili obje metode koje smo naveli u 3. zadatku.



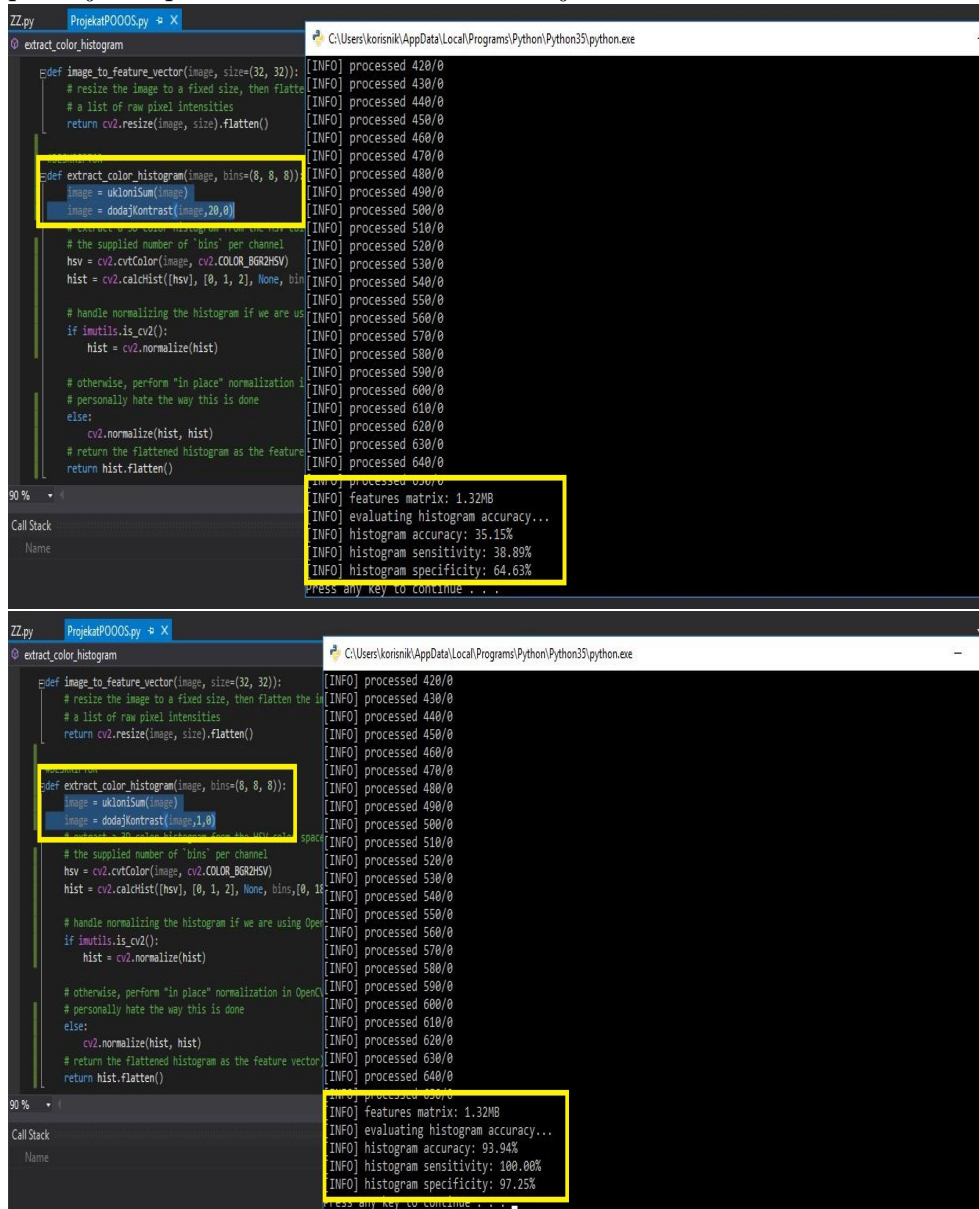
```
def image_to_feature_vector(image, size=(32, 32)):  
    # resize the image to a fixed size, then flatten it  
    # a list of raw pixel intensities  
    return cv2.resize(image, size).flatten()  
  
#DESCRIPTION  
def extract_color_histogram(image, bins=(8, 8, 8)):  
    image = ukloniSum(image)  
    image = dodajKontrast(image, 10, 0)  
    # the supplied number of 'bins' per channel  
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)  
    hist = cv2.calcHist([hsv], [0, 1, 2], None, bins, [0, 0, 0, 256, 256, 256])  
    # handle normalizing the histogram if we are using  
    if hist is not None:  
        hist = cv2.normalize(hist)  
    # otherwise, perform "in place" normalization in O  
    # personally hate the way this is done  
    else:  
        cv2.normalize(hist, hist)  
    # return the flattened histogram as the feature vector  
    return hist.flatten()  
  
[INFO] processed 420/0  
[INFO] processed 430/0  
[INFO] processed 440/0  
[INFO] processed 450/0  
[INFO] processed 460/0  
[INFO] processed 470/0  
[INFO] processed 480/0  
[INFO] processed 490/0  
[INFO] processed 500/0  
[INFO] processed 510/0  
[INFO] processed 520/0  
[INFO] processed 530/0  
[INFO] processed 540/0  
[INFO] processed 550/0  
[INFO] processed 560/0  
[INFO] processed 570/0  
[INFO] processed 580/0  
[INFO] processed 590/0  
[INFO] processed 600/0  
[INFO] processed 610/0  
[INFO] processed 620/0  
[INFO] processed 630/0  
[INFO] processed 640/0  
[INFO] processed 650/0  
[INFO] features matrix: 1.32MB  
[INFO] evaluating histogram accuracy...  
[INFO] histogram accuracy: 64.24%  
[INFO] histogram sensitivity: 51.96%  
[INFO] histogram specificity: 90.48%  
Press any key to continue . . .
```

2. Ovdje smo koristili metodu 'dodajKontrast'.



```
def image_to_feature_vector(image, size=(32, 32)):  
    # resize the image to a fixed size, then flatten it  
    # a list of raw pixel intensities  
    return cv2.resize(image, size).flatten()  
  
#DESCRIPTION  
def extract_color_histogram(image, bins=(8, 8, 8)):  
    image = ukloniSum(image)  
    image = dodajKontrast(image, 10, 0)  
    # the supplied number of 'bins' per channel  
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)  
    hist = cv2.calcHist([hsv], [0, 1, 2], None, bins, [0, 0, 0, 256, 256, 256])  
    # handle normalizing the histogram if we  
    if hist is not None:  
        hist = cv2.normalize(hist)  
    # otherwise, perform "in place" normaliz  
    # personally hate the way this is done  
    else:  
        cv2.normalize(hist, hist)  
    # return the flattened histogram as the  
    return hist.flatten()  
  
[INFO] processed 420/0  
[INFO] processed 430/0  
[INFO] processed 440/0  
[INFO] processed 450/0  
[INFO] processed 460/0  
[INFO] processed 470/0  
[INFO] processed 480/0  
[INFO] processed 490/0  
[INFO] processed 500/0  
[INFO] processed 510/0  
[INFO] processed 520/0  
[INFO] processed 530/0  
[INFO] processed 540/0  
[INFO] processed 550/0  
[INFO] processed 560/0  
[INFO] processed 570/0  
[INFO] processed 580/0  
[INFO] processed 590/0  
[INFO] processed 600/0  
[INFO] processed 610/0  
[INFO] processed 620/0  
[INFO] processed 630/0  
[INFO] processed 640/0  
[INFO] processed 650/0  
[INFO] features matrix: 1.32MB  
[INFO] evaluating histogram accuracy...  
[INFO] histogram accuracy: 72.12%  
[INFO] histogram sensitivity: 60.39%  
[INFO] histogram specificity: 78.45%  
Press any key to continue . . .
```

3. U sljedeća dva slučaja pozivamo kao i u prvom obje metode, s tim što smo ovdje promijenili parametar kod metode 'dodajKontrast'.



```
def image_to_feature_vector(image, size=(32, 32)):
    # resize the image to a fixed size, then flatten it
    # a list of raw pixel intensities
    return cv2.resize(image, size).flatten()

def extract_color_histogram(image, bins=(8, 8, 8)):
    image = ukloniSum(image)
    image = dodajKontrast(image, 20.0)
    # extract a color histogram from the HSV color space
    # the supplied number of 'bins' per channel
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hist = cv2.calcHist([hsv], [0, 1, 2], None, bins)

    # handle normalizing the histogram if we are using OpenCV
    if imutils.is_cv2():
        hist = cv2.normalize(hist)

    # otherwise, perform "in place" normalization in OpenCV
    # personally hate the way this is done
    else:
        cv2.normalize(hist, hist)

    # return the flattened histogram as the feature vector
    return hist.flatten()

[INFO] processed 420/0
[INFO] processed 430/0
[INFO] processed 440/0
[INFO] processed 450/0
[INFO] processed 460/0
[INFO] processed 470/0
[INFO] processed 480/0
[INFO] processed 490/0
[INFO] processed 500/0
[INFO] processed 510/0
[INFO] processed 520/0
[INFO] processed 530/0
[INFO] processed 540/0
[INFO] processed 550/0
[INFO] processed 560/0
[INFO] processed 570/0
[INFO] processed 580/0
[INFO] processed 590/0
[INFO] processed 600/0
[INFO] processed 610/0
[INFO] processed 620/0
[INFO] processed 630/0
[INFO] processed 640/0
[INFO] processed 650/0
[INFO] features matrix: 1.32MB
[INFO] evaluating histogram accuracy...
[INFO] histogram accuracy: 35.15%
[INFO] histogram sensitivity: 38.89%
[INFO] histogram specificity: 64.63%
Press any key to continue . . .
```

```
def image_to_feature_vector(image, size=(32, 32)):
    # resize the image to a fixed size, then flatten it
    # a list of raw pixel intensities
    return cv2.resize(image, size).flatten()

def extract_color_histogram(image, bins=(8, 8, 8)):
    image = ukloniSum(image)
    image = dodajKontrast(image, 1.0)
    # extract a color histogram from the HSV color space
    # the supplied number of 'bins' per channel
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hist = cv2.calcHist([hsv], [0, 1, 2], None, bins, [0, 180, 256, 256])

    # handle normalizing the histogram if we are using OpenCV
    if imutils.is_cv2():
        hist = cv2.normalize(hist)

    # otherwise, perform "in place" normalization in OpenCV
    # personally hate the way this is done
    else:
        cv2.normalize(hist, hist)

    # return the flattened histogram as the feature vector
    return hist.flatten()

[INFO] processed 420/0
[INFO] processed 430/0
[INFO] processed 440/0
[INFO] processed 450/0
[INFO] processed 460/0
[INFO] processed 470/0
[INFO] processed 480/0
[INFO] processed 490/0
[INFO] processed 500/0
[INFO] processed 510/0
[INFO] processed 520/0
[INFO] processed 530/0
[INFO] processed 540/0
[INFO] processed 550/0
[INFO] processed 560/0
[INFO] processed 570/0
[INFO] processed 580/0
[INFO] processed 590/0
[INFO] processed 600/0
[INFO] processed 610/0
[INFO] processed 620/0
[INFO] processed 630/0
[INFO] processed 640/0
[INFO] processed 650/0
[INFO] features matrix: 1.32MB
[INFO] evaluating histogram accuracy...
[INFO] histogram accuracy: 93.94%
[INFO] histogram sensitivity: 100.00%
[INFO] histogram specificity: 97.25%
Press any key to continue . . .
```

4. Ovdje ćemo pokazati rezultate prilikom različite podjele podataka na train i test.

The image displays two screenshots of a Python IDE (likely PyCharm) showing the execution of a script named 'ProjektatPOOOS.py'. The code implements a k-NN classifier on a dataset of raw images.

**Top Screenshot:** The code uses `train_test_split` with `test_size=0.25`. The output shows the following metrics:

- features matrix: 1.32MB
- evaluating histogram accuracy...
- histogram accuracy: 87.27%
- histogram sensitivity: 92.45%
- histogram specificity: 93.75%

**Bottom Screenshot:** The code is modified to use `train_test_split` with `test_size=0.15`. The output shows improved metrics:

- features matrix: 1.32MB
- evaluating histogram accuracy...
- histogram accuracy: 94.95%
- histogram sensitivity: 100.00%
- histogram specificity: 98.44%

5. Ovdje smo koristili metodu 'ukloniSum'.

```

def extract_color_histogram(image, bins=(8, 8, 8)):
    # image = ukoniSum(image)
    # image = dotaContrast(image, 10, 0)
    # extract a 3D color histogram from the image
    # the supplied number of bins per channel
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hist = cv2.calcHist([hsv], [0, 1, 2], None, bins)

    # handle normalizing the histogram if we are not using cv2
    if imutils.is_cv2():
        hist = cv2.normalize(hist)

    # otherwise, perform "in place" normalization
    # personally hate the way this is done
    else:
        cv2.normalize(hist, hist)

    # return the flattened histogram as the feature vector
    return hist.flatten()

```

```

[INFO] processed 420/0
[INFO] processed 430/0
[INFO] processed 440/0
[INFO] processed 450/0
[INFO] processed 460/0
[INFO] processed 470/0
[INFO] processed 480/0
[INFO] processed 490/0
[INFO] processed 500/0
[INFO] processed 510/0
[INFO] processed 520/0
[INFO] processed 530/0
[INFO] processed 540/0
[INFO] processed 550/0
[INFO] processed 560/0
[INFO] processed 570/0
[INFO] processed 580/0
[INFO] processed 590/0
[INFO] processed 600/0
[INFO] processed 610/0
[INFO] processed 620/0
[INFO] processed 630/0
[INFO] processed 640/0
[INFO] processed 650/0
[INFO] features matrix: 1.32MB
[INFO] evaluating histogram accuracy...
[INFO] histogram accuracy: 93.94%
[INFO] histogram sensitivity: 100.00%
[INFO] histogram specificity: 97.25%
press any key to continue . . .

```

6. Ovdje smo promijenili parametar prilikom pozivanja algoritma modela.

```

def model():
    # split the data for training and the remaining test set
    (trainRI, testRI, trainRL, testRL) = train_test_split(
        rawImages, labels, test_size=0.25, random_state=42)
    (trainFeat, testFeat, trainLabels, testLabels) = train_test_split(
        features, labels, test_size=0.15, random_state=42)

    # train and evaluate a k-NN classifier on the raw pixel intensities
    #print("[INFO] evaluating raw pixel accuracy...")
    #model = KNeighborsClassifier(n_neighbors=20)
    #model.fit(trainRI, trainRL)
    #acc = model.score(testRI, testRL)
    #print("[INFO] raw pixel accuracy: {:.2f}%".format(acc))

    # train and evaluate a k-NN classifier on the histogram representations
    #print("[INFO] evaluating histogram accuracy...")
    model = KNeighborsClassifier(n_neighbors=40)
    model.fit(trainFeat, trainLabels)
    y_test = model.predict(testFeat)
    acc = model.score(testFeat, testLabels)
    print("[INFO] histogram accuracy: {:.2f}%".format(acc))

    matrica = metrics.confusion_matrix(testLabels, y_test)
    TP = matrica[0, 0]
    TN = matrica[1, 1] + matrica[1, 2] + matrica[2, 1]
    FP = matrica[0, 1] + matrica[0, 2]
    FN = matrica[1, 0] + matrica[2, 0]
    sensitivity = TP / (TP + FN)
    specificity = TN / (TN + FP)

```

```

[INFO] processed 420/0
[INFO] processed 430/0
[INFO] processed 440/0
[INFO] processed 450/0
[INFO] processed 460/0
[INFO] processed 470/0
[INFO] processed 480/0
[INFO] processed 490/0
[INFO] processed 500/0
[INFO] processed 510/0
[INFO] processed 520/0
[INFO] processed 530/0
[INFO] processed 540/0
[INFO] processed 550/0
[INFO] processed 560/0
[INFO] processed 570/0
[INFO] processed 580/0
[INFO] processed 590/0
[INFO] processed 600/0
[INFO] processed 610/0
[INFO] processed 620/0
[INFO] processed 630/0
[INFO] processed 640/0
[INFO] processed 650/0
[INFO] features matrix: 1.32MB
[INFO] evaluating histogram accuracy...
[INFO] histogram accuracy: 86.87%
[INFO] histogram sensitivity: 89.47%
[INFO] histogram specificity: 96.72%
press any key to continue . . .

```