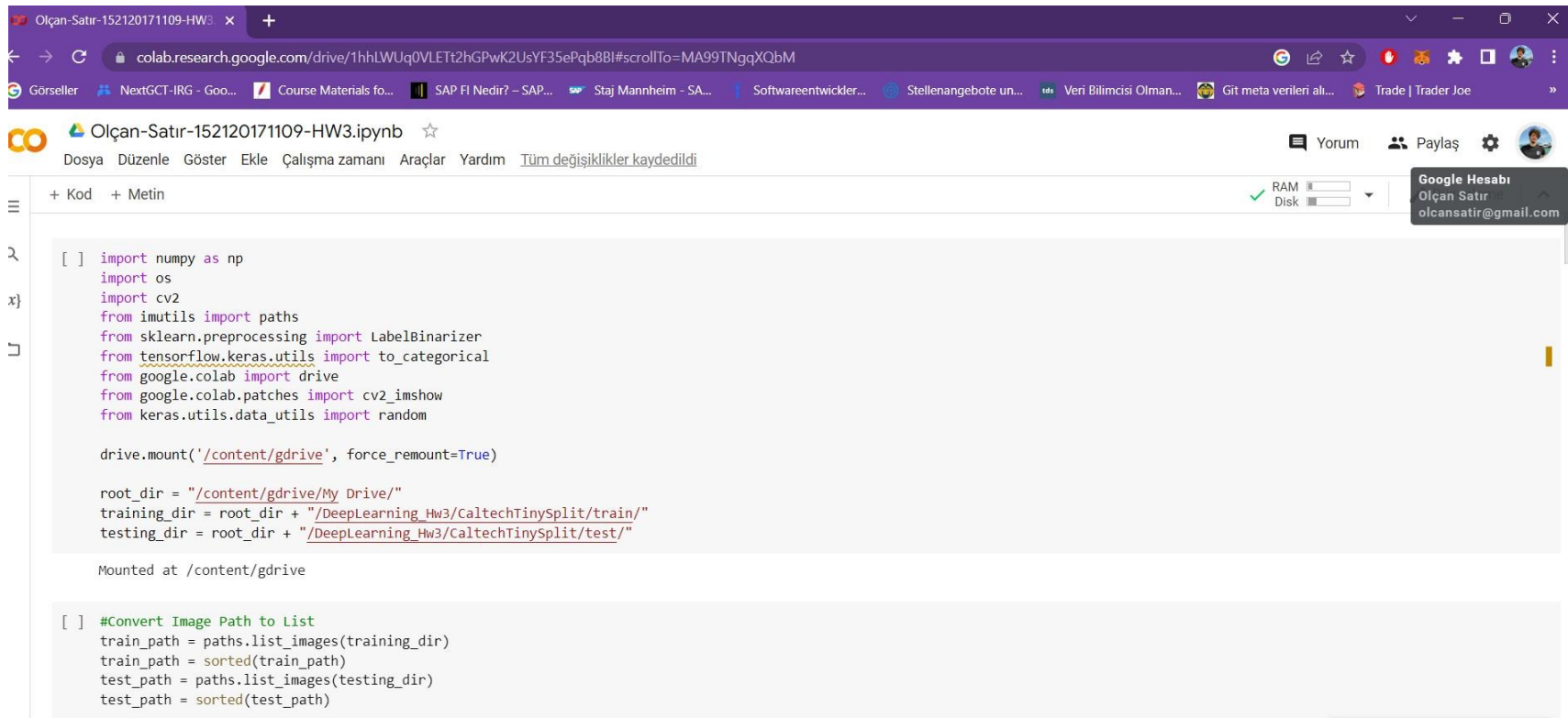


OLÇAN SATIR

152120171109

Deep Learning HW-3

Here, we first do it from the folder we created for the dataset in the drive. Then we create an image list and sort it.



```
[ ] import numpy as np
import os
import cv2
from imutils import paths
from sklearn.preprocessing import LabelBinarizer
from tensorflow.keras.utils import to_categorical
from google.colab import drive
from google.colab.patches import cv2_imshow
from keras.utils.data_utils import random

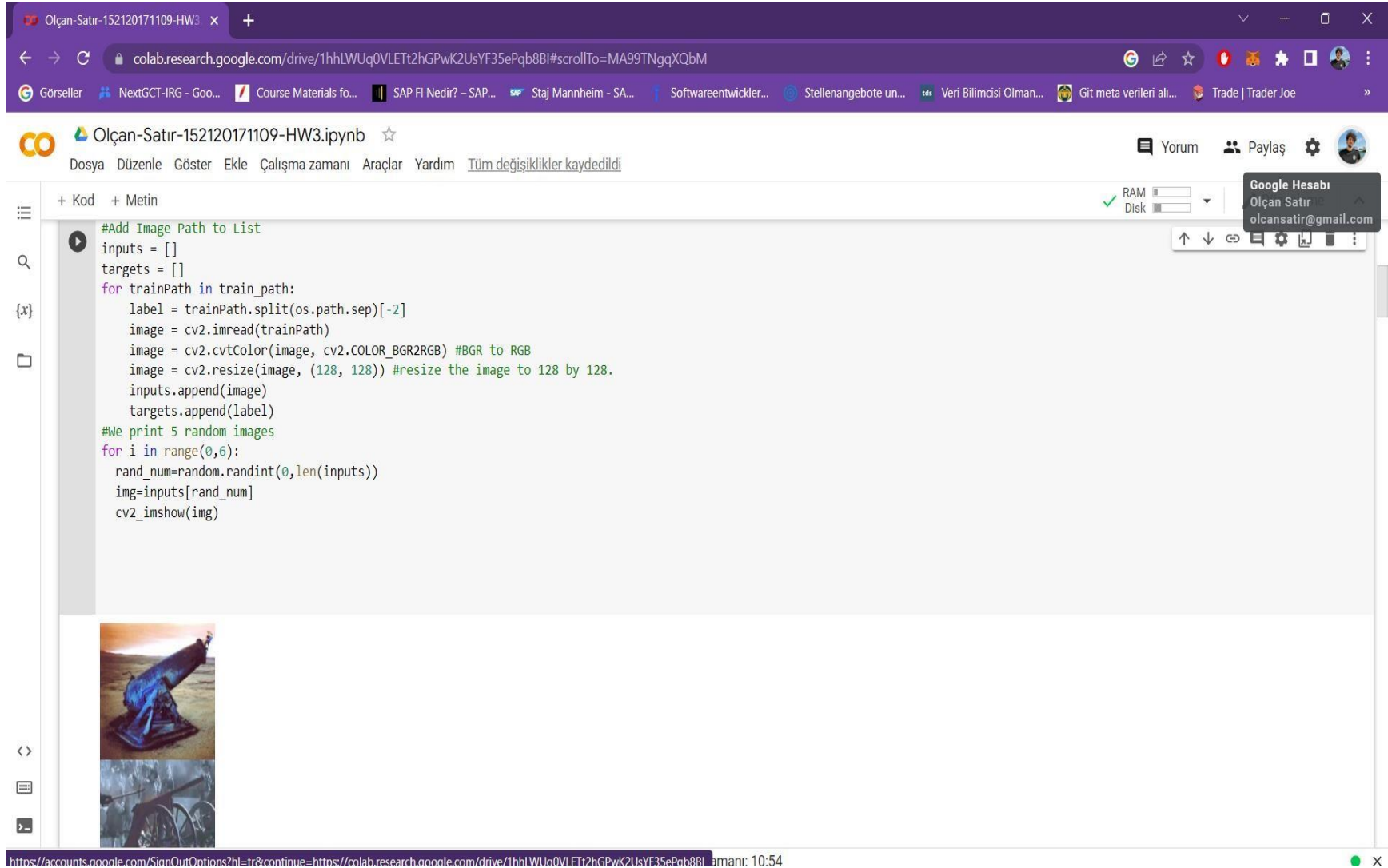
drive.mount('/content/gdrive', force_remount=True)

root_dir = "/content/gdrive/My Drive/"
training_dir = root_dir + "/DeepLearning_Hw3/CaltechTinySplit/train/"
testing_dir = root_dir + "/DeepLearning_Hw3/CaltechTinySplit/test/"

Mounted at /content/gdrive

[ ] #Convert Image Path to List
train_path = paths.list_images(training_dir)
train_path = sorted(train_path)
test_path = paths.list_images(testing_dir)
test_path = sorted(test_path)
```

Then we resize the train pictures as 128 by 128 by converting them from BGR to RGB by naming them as inputs and targets. In the for loop below, we print 5 random pictures to see.



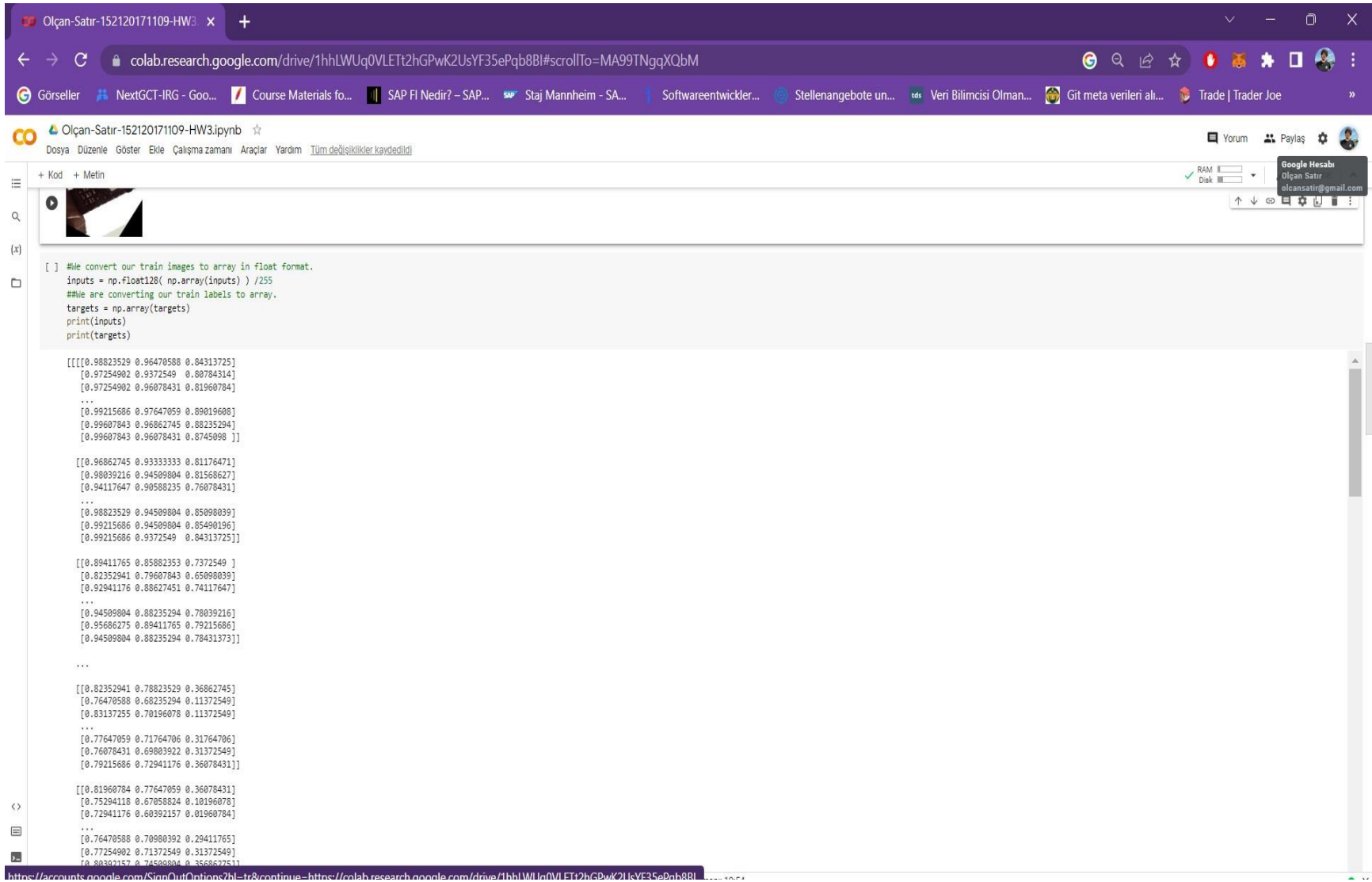
The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: <https://colab.research.google.com/drive/1hhLWUq0VLEt2hGPwK2UsYF35ePqb8BI#scrollTo=MA99TNggXQbM>. The notebook title is "Olçan-Satır-152120171109-HW3.ipynb". The code editor contains the following Python code:

```
#Add Image Path to List
inputs = []
targets = []
for trainPath in train_path:
    label = trainPath.split(os.path.sep)[-2]
    image = cv2.imread(trainPath)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) #BGR to RGB
    image = cv2.resize(image, (128, 128)) #resize the image to 128 by 128.
    inputs.append(image)
    targets.append(label)
#We print 5 random images
for i in range(0,6):
    rand_num=random.randint(0,len(inputs))
    img=inputs[rand_num]
    cv2_imshow(img)
```

Below the code editor, two small image thumbnails are displayed. The top thumbnail shows a blue, metallic, mechanical device, possibly a cannon or a large gun, mounted on a wooden base. The bottom thumbnail shows a close-up of a mechanical component, possibly a wheel or a gear, with a dark, textured background.

The bottom status bar of the Colab interface shows the URL: <https://accounts.google.com/SignInOptions?hl=tr&continue=https://colab.research.google.com/drive/1hhLWUq0VLEt2hGPwK2UsYF35ePqb8BI> and the time: 10:54.

Here we convert the inputs and targets into array.

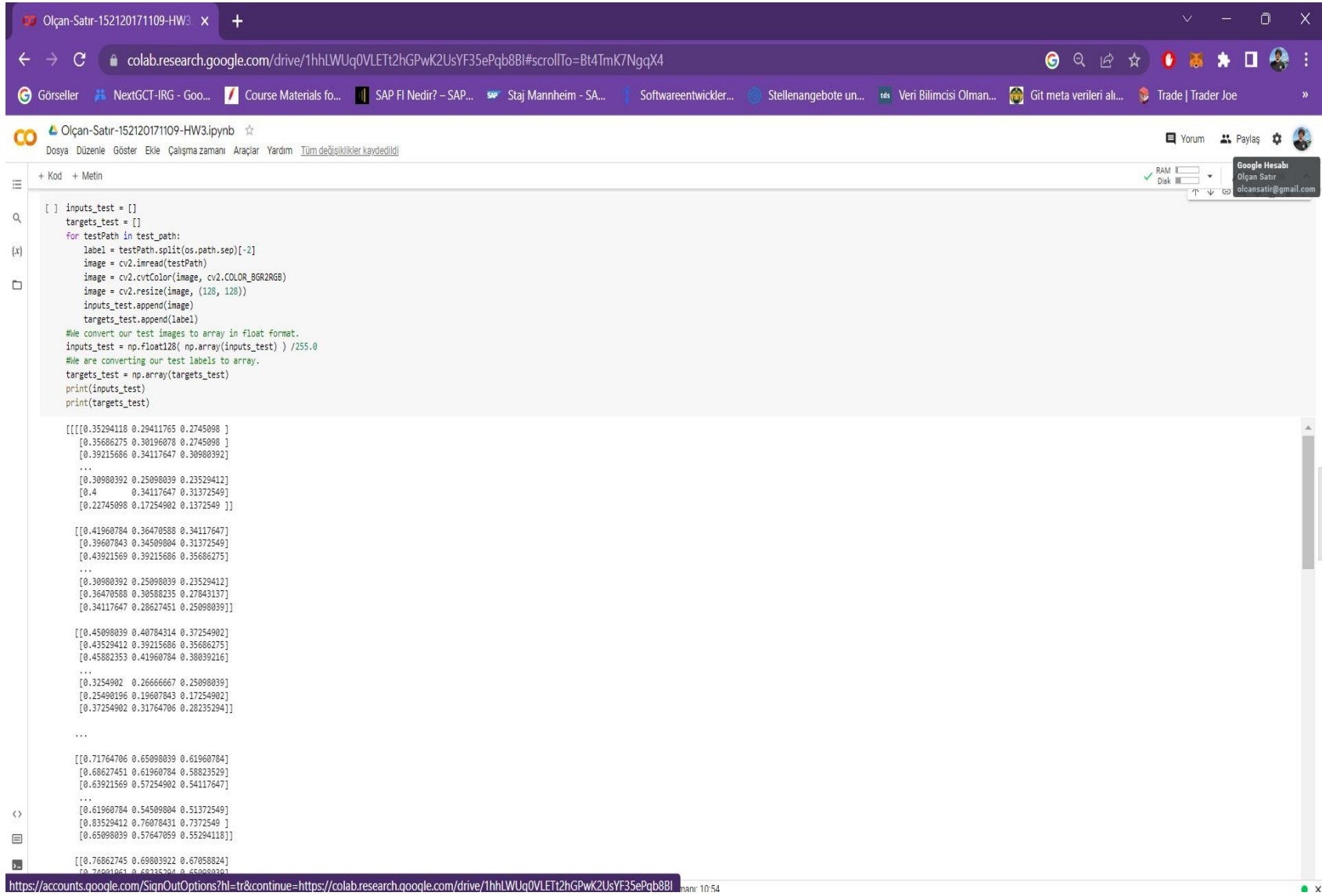


The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: colab.research.google.com/drive/1hhLWUq0VLEt2hGPwK2UsYF35ePqb8Bl#scrollTo=MA99TNggXQbM. The notebook title is "Olçan-Satır-152120171109-HW3.ipynb". The code cell contains the following Python code:

```
[ ] #We convert our train images to array in float format.
inputs = np.float128( np.array(inputs) ) /255
##We are converting our train labels to array.
targets = np.array(targets)
print(inputs)
print(targets)
```

The output of the code is displayed below the code cell, showing two large arrays of numerical values. The first array is the output of `print(inputs)` and the second array is the output of `print(targets)`. Both arrays contain multiple rows of numerical data, representing the converted inputs and targets.

Then we resize the test pictures as 128 by 128 by converting them from BGR to RGB by naming them as inputs and targets. In the for loop below, we print 5 random pictures to see.



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: <https://colab.research.google.com/drive/1hhLWUq0VLEt2hGPwK2UsYF35ePqb8Bl#scrollTo=Bt4TmK7NgqX4>. The notebook is titled "Olçan-Satır-152120171109-HW3.ipynb". The code in the notebook is as follows:

```
[ ] inputs_test = []
targets_test = []
for testPath in test_path:
    label = testPath.split(os.path.sep)[-2]
    image = cv2.imread(testPath)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (128, 128))
    inputs_test.append(image)
    targets_test.append(label)

#We convert our test images to array in float format.
inputs_test = np.float128( np.array(inputs_test) ) /255.0
#We are converting our test labels to array.
targets_test = np.array(targets_test)
print(inputs_test)
print(targets_test)
```

The output of the code shows two arrays of image data and labels. The first array, `inputs_test`, contains 128x128 pixel images converted to float128 format and normalized by 255.0. The second array, `targets_test`, contains the corresponding labels. The output is displayed as a list of lists, where each inner list represents a single image and its label.

```
[[[0.35294118 0.29411765 0.27450898 ]
 [0.35686275 0.30196078 0.27450898 ]
 [0.39215686 0.34117647 0.30908392 ]
 ...
 [0.30908392 0.25098039 0.23529412 ]
 [0.4         0.34117647 0.31372549 ]
 [0.22745098 0.17254902 0.1372549 ]]]

[[0.41960784 0.36470588 0.34117647]
 [0.39607843 0.34509804 0.31372549]
 [0.43921569 0.39215686 0.35686275]
 ...
 [0.30908392 0.25098039 0.23529412 ]
 [0.36470588 0.30588235 0.27843137]
 [0.34117647 0.28627451 0.25098039]]]

[[0.45098039 0.40784314 0.37254902]
 [0.43529412 0.39215686 0.35686275]
 [0.45882353 0.41960784 0.38039216]
 ...
 [0.3254902 0.26666667 0.25098039]
 [0.25490196 0.19607843 0.17254902]
 [0.37254902 0.31764706 0.28235294]]]

...

[[0.71764706 0.65098039 0.61960784]
 [0.68627451 0.61960784 0.58823529]
 [0.63921569 0.57254902 0.54117647]
 ...
 [0.61960784 0.54509804 0.51372549]
 [0.83529412 0.76078431 0.7372549 ]
 [0.65098039 0.57647059 0.55294118]]]

[[0.76862745 0.69803922 0.67058824]
 [0.74801961 0.68235294 0.65098039]]]
```

The bottom of the screenshot shows the URL: <https://accounts.google.com/SignOutOptions?hl=tr&continue=https://colab.research.google.com/drive/1hhLWUq0VLEt2hGPwK2UsYF35ePqb8Bl>.

Then we label our images as 0 and 1 using the labelencoder.

The screenshot shows a Google Colab notebook with the following content:

```
Olcan-Satir-152120171109-HW3.ipynb
Dosya Düzenle Göster Ekle Çalışma zamanı Araçlar Yardım Tüm değişiklikler kaydedildi

+ Kod + Metin
[1] ['cannon' 'cannon' 'cannon' 'cannon' 'cannon' 'cellphone' 'cellphone'
      'cellphone' 'cellphone' 'cellphone' 'cellphone' 'cellphone']

[6] #With the help of the random function, we create our weights with the size of 1*49152.
W = np.random.rand(49152,) / 1000
print(W)

[7.48484256e-04 8.38371281e-04 8.93944631e-05 ... 7.81460936e-04
 9.94778173e-04 3.62843993e-04]

[7] #Create the activation functions.
def tanh(x):
    return (np.exp(x) - np.exp(-x))/(np.exp(x) + np.exp(-x))
def softplus(x):
    return np.log(1 + np.exp(x))
def mish(x):
    return x * tanh(softplus(x))
def dmish(x):
    omega = np.exp(3*x) + 4*np.exp(2*x) + (6+4*x)*np.exp(x) + 4*(1 + x)
    delta = 1 + pow((np.exp(x) + 1), 2)
    derivative = np.exp(x) * omega / pow(delta, 2)
    return derivative

x_train=inputs
y_train=targets
X_test=inputs_test
Y_test=targets_test
#We reshape x_train to shape: 1*49152 (for 81 train)
x_train = np.reshape(x_train, (81,49152))

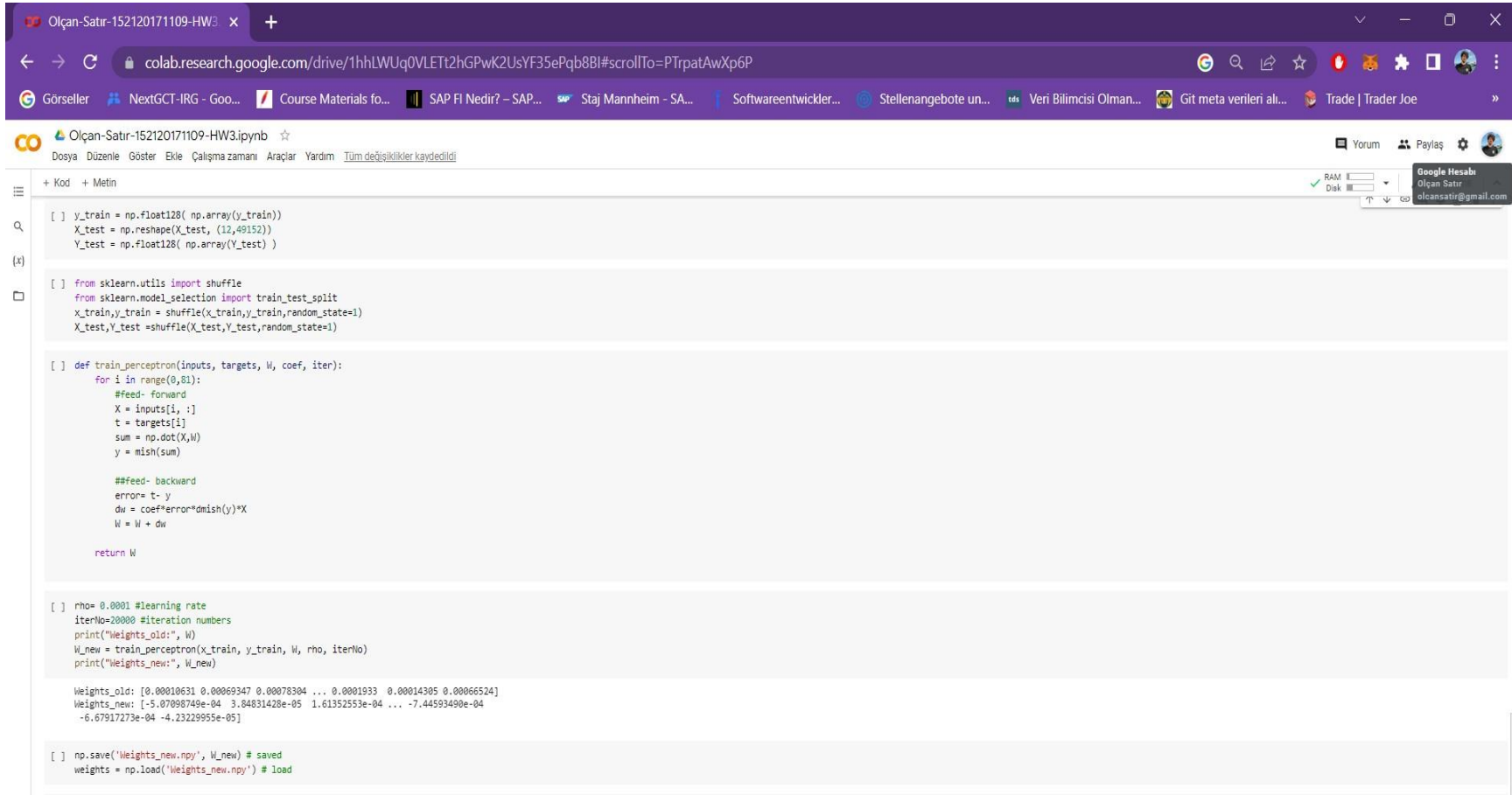
[9] #We convert our labels to 0 and 1 by performing labelEncoder operation.
from sklearn.preprocessing import LabelEncoder , OneHotEncoder
y_labelencoder = LabelEncoder ()
y_train = y_labelencoder.fit_transform (y_train)
#Cannon class the label is 0, cellphone class the label is 1
print ("After label encoding train labels: \n",y_train)
y_labelencoder = LabelEncoder ()
Y_test = y_labelencoder.fit_transform (Y_test)
print(" ")
print ("After label encoding test labels: ",Y_test)

After label encoding train labels:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1]

After label encoding test labels: [0 0 0 0 1 1 1 1 1 1]
```

Here we first convert `y_train` to an array of float128 type values. We then resize `X_test` for 1x49152. Then we convert `Y_test` into an array of float128 type values in the same way, then we perform shuffle operation for our data in the train dataset. After that, we apply feed forward to our training data using mish function, then we update the weights by calculating the error and applying feed backward.

Then, we define the learning rate and the number of iterations that we will use in the testing phase, save and load the weights.



```
Olcun-Satir-152120171109-HW3 x +
colab.research.google.com/drive/1hhLWUq0VLEtt2hGPwK2UsYF35ePqb8BI#scrollTo=PTtptAwXp6P
Görseller NextGCT-IRG - Goo... Course Materials fo... SAP FI Nedir? - SAP... Staj Mannheim - SA... Softwareentwickler... Stellenangebote un... Veri Bilimcisi Olman... Git meta verileri ali... Trade | Trader Joe »

Olcun-Satir-152120171109-HW3.ipynb ☆
Dosya Düzenle Göster Ekle Çalışma zamanı Araçlar Yardım Tüm değişiklikler kaydedildi
+ Kod + Metin RAM Disk Google Hesabı Olcun Satir alcansatir@gmail.com

[ ] y_train = np.float128( np.array(y_train))
X_test = np.reshape(X_test, (12,49152))
Y_test = np.float128( np.array(Y_test) )

[ ] from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
x_train,y_train = shuffle(x_train,y_train,random_state=1)
X_test,Y_test =shuffle(X_test,Y_test,random_state=1)

[ ] def train_perceptron(inputs, targets, W, coef, iter):
    for i in range(0,81):
        #feed- forward
        X = inputs[i, :]
        t = targets[i]
        sum = np.dot(X,W)
        y = mish(sum)

        ##feed- backward
        error= t- y
        dw = coef*error*mish(y)*X
        W = W + dw

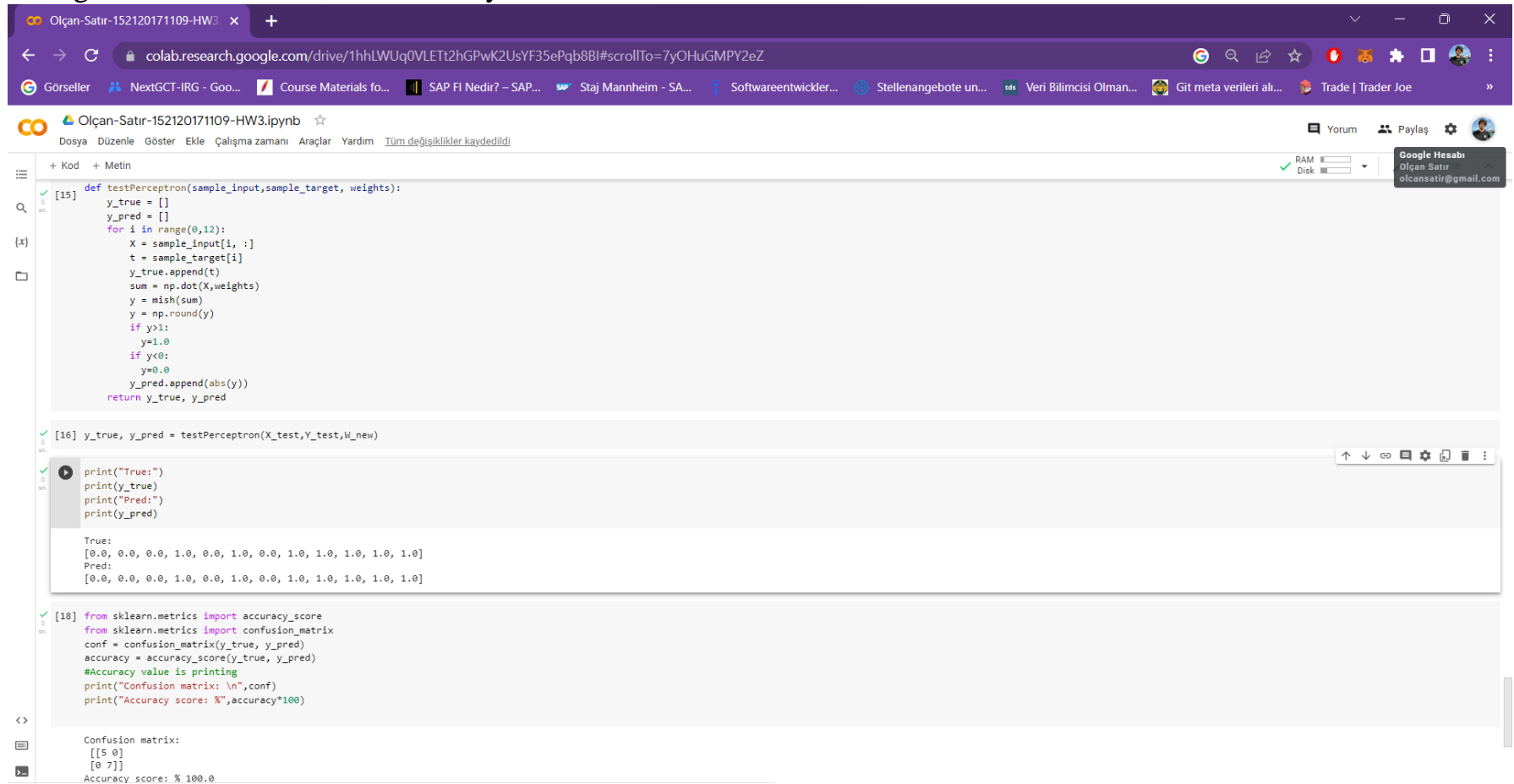
    return W

[ ] rho= 0.0001 #learning rate
iterNo=20000 #iteration numbers
print("Weights_old:", W)
W_new = train_perceptron(x_train, y_train, W, rho, iterNo)
print("Weights_new:", W_new)

Weights_old: [0.00010631 0.00069347 0.00078304 ... 0.0001933 0.00014305 0.00066524]
Weights_new: [-5.07098749e-04  3.84831428e-05  1.61352553e-04 ... -7.44593490e-04
 -6.67917273e-04 -4.2329955e-05]

[ ] np.save('Weights_new.npy', W_new) # saved
weights = np.load('Weights_new.npy') # load
```

Then we test our images with the testPerceptron function here. Here, if the relevant values are greater than 1, it takes it as 1, and if it is less than 0, it takes it as 0. When we test our 12 images, we see that they are all correctly categorized. In this case, our accuracy value is 100%.



```
[15] def testPerceptron(sample_input, sample_target, weights):
      y_true = []
      y_pred = []
      for i in range(0, 12):
          X = sample_input[i, :]
          t = sample_target[i]
          y_true.append(t)
          sum = np.dot(X, weights)
          y = mish(sum)
          y = np.round(y)
          if y > 1:
              y = 1.0
          if y < 0:
              y = 0.0
          y_pred.append(abs(y))
      return y_true, y_pred

[16] y_true, y_pred = testPerceptron(X_test, Y_test, W_new)

print("True:")
print(y_true)
print("Pred:")
print(y_pred)

True:
[0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Pred:
[0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0]

[18] from sklearn.metrics import accuracy_score
      from sklearn.metrics import confusion_matrix
      conf = confusion_matrix(y_true, y_pred)
      accuracy = accuracy_score(y_true, y_pred)
      #Accuracy value is printing
      print("Confusion matrix: \n", conf)
      print("Accuracy score: %", accuracy*100)

Confusion matrix:
[[5 0]
 [0 7]]
Accuracy score: % 100.0
```