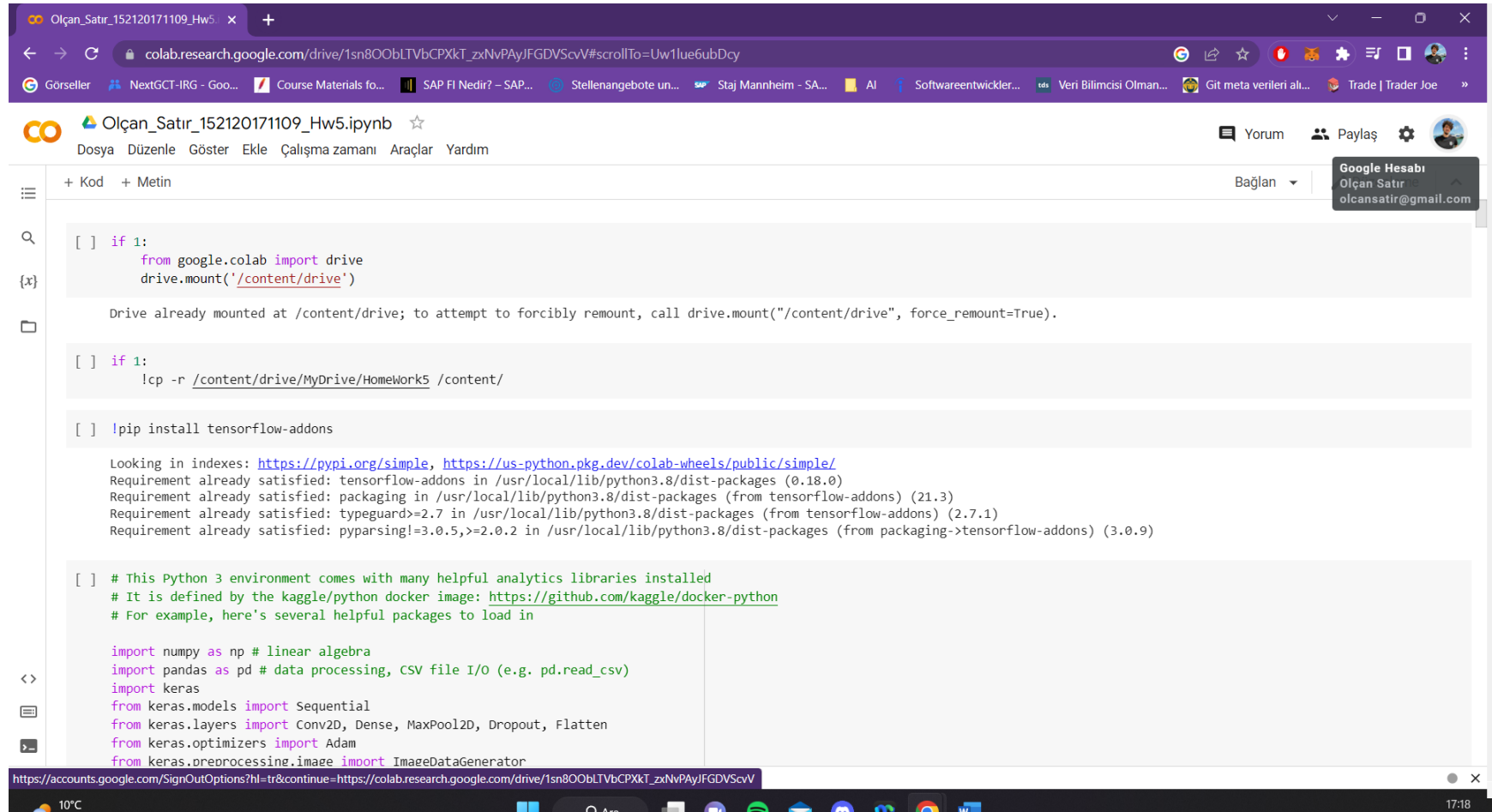


OLÇAN SATIR

152120171109

Deep Learning HW-5

We are establishing our colab connection.



The screenshot shows a Google Colab notebook titled "Olçan_Satir_152120171109_Hw5.ipynb". The notebook is open in a web browser, and the URL bar shows the Colab environment. The notebook content includes the following code cells:

```
[ ] if 1:
    from google.colab import drive
    drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
[ ] if 1:
    !cp -r /content/drive/MyDrive/Homework5 /content/
```

```
[ ] !pip install tensorflow-addons
```

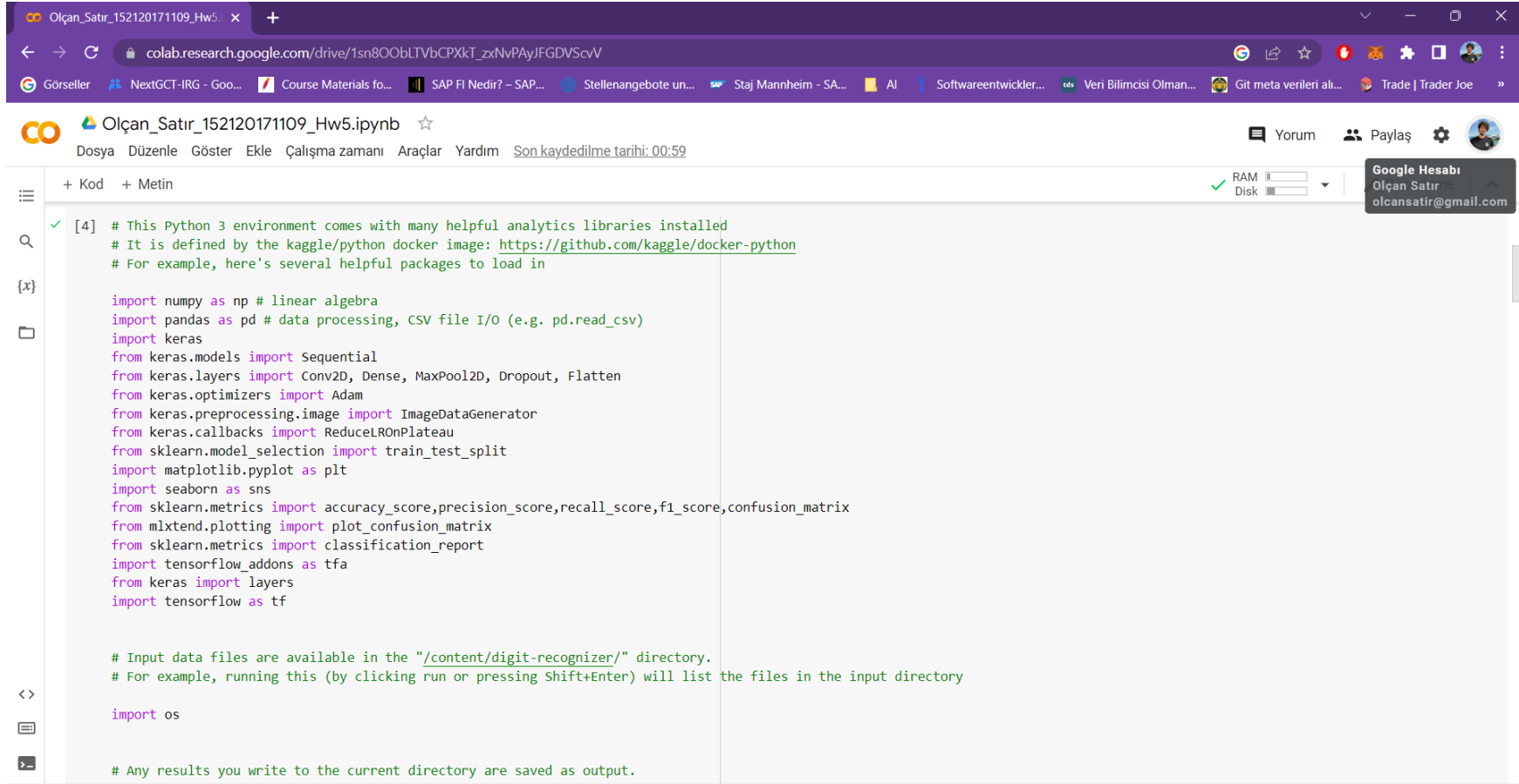
Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/simple/>
Requirement already satisfied: tensorflow-addons in /usr/local/lib/python3.8/dist-packages (0.18.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-packages (from tensorflow-addons) (21.3)
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.8/dist-packages (from tensorflow-addons) (2.7.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.8/dist-packages (from packaging->tensorflow-addons) (3.0.9)

```
[ ] # This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import keras
from keras.models import Sequential
from keras.layers import Conv2D, Dense, MaxPool2D, Dropout, Flatten
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
```

The bottom of the screenshot shows the Google account sign-out options and the system clock indicating 10°C and 17:18.

We define our libraries.



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: colab.research.google.com/drive/1sn8OObLTVbCPXkT_zxNvPayJFGDVSqvV. The notebook title is "Olçan_Satır_152120171109_Hw5.ipynb". The code cell [4] contains the following Python code:

```
[4] # This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import keras
from keras.models import Sequential
from keras.layers import Conv2D, Dense, MaxPool2D, Dropout, Flatten
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import classification_report
import tensorflow_addons as tfa
from keras import layers
import tensorflow as tf

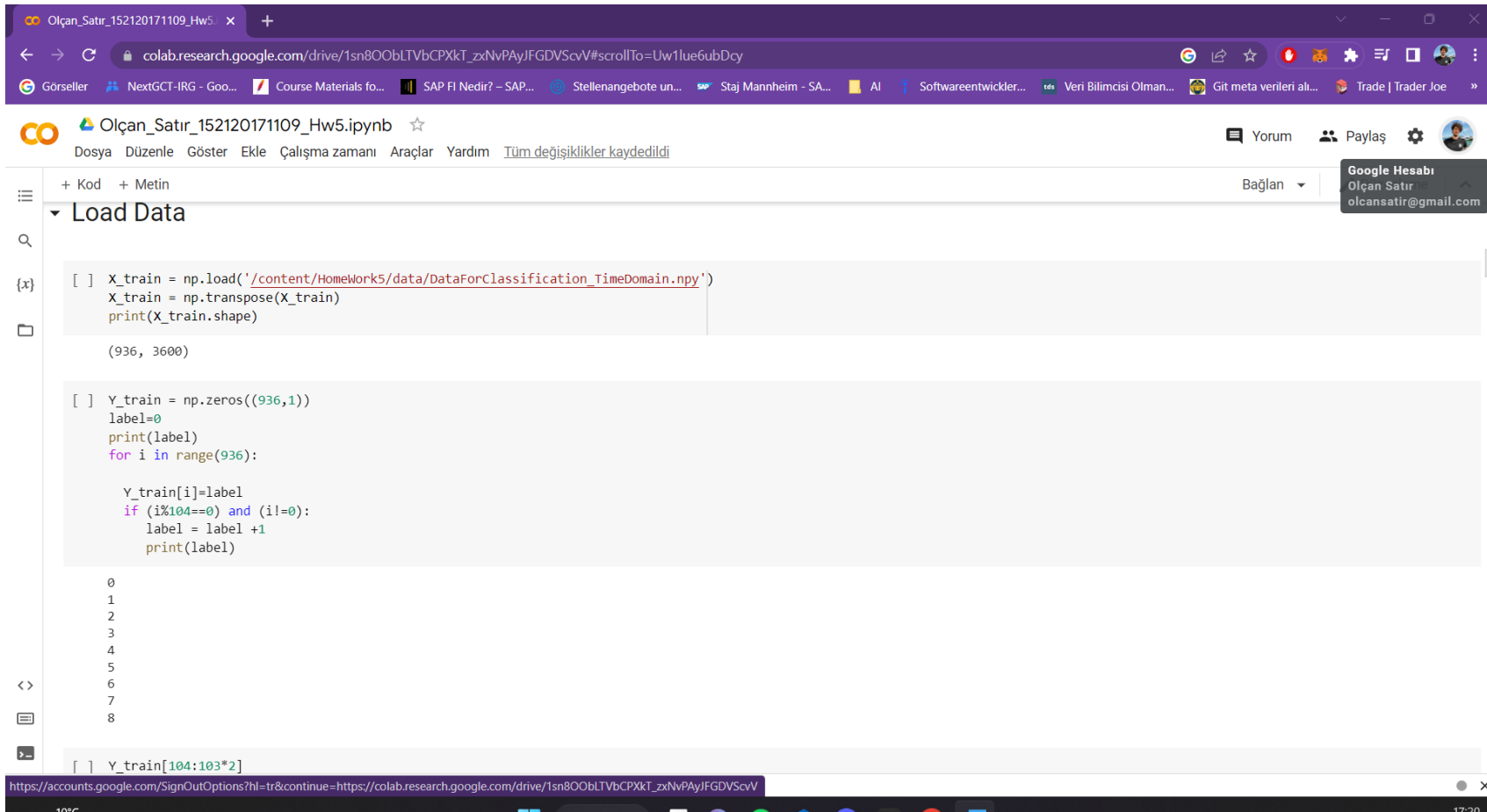
# Input data files are available in the "/content/digit-recognizer/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

import os

# Any results you write to the current directory are saved as output.
```

The interface includes a left sidebar with icons for file explorer, search, and code execution. The top right shows RAM and Disk usage, and a Google account dropdown menu.

With the help of the for loop, we fill the Y_train and print our labels. For this, our i value must be divisible by 104 and equal to 0. When this condition is met, we increase our label value by 1 and suppress it.



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: `colab.research.google.com/drive/1sn8OObLTVbCPXkT_zxNvPayJFGDVScvV#scrollTo=Uw1lue6ubDcy`. The notebook title is "Olçan_Satır_152120171109_Hw5.ipynb". The left sidebar shows a file explorer with a folder named "Load Data". The main code area contains the following Python code:

```
[ ] X_train = np.load('/content/HomeWork5/data/DataForClassification_TimeDomain.npy')
X_train = np.transpose(X_train)
print(X_train.shape)

(936, 3600)

[ ] Y_train = np.zeros((936,1))
label=0
print(label)
for i in range(936):

    Y_train[i]=label
    if (i%104==0) and (i!=0):
        label = label +1
        print(label)

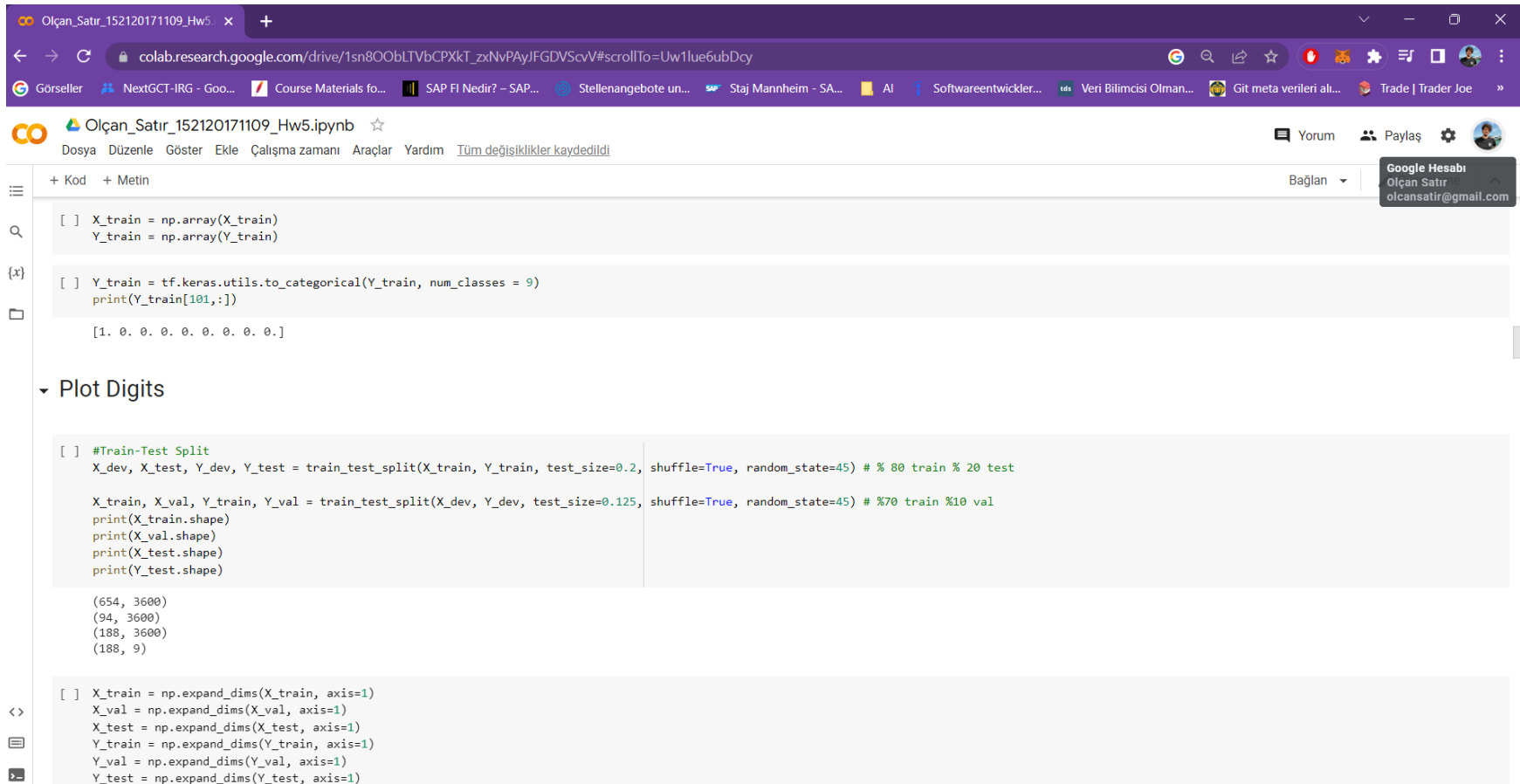
0
1
2
3
4
5
6
7
8

[ ] Y_train[104:103*2]
```

The output of the code shows the shape of X_train as (936, 3600) and the values of label from 0 to 8, which are printed at intervals of 104 iterations.

Here we print the values from index 104 to index 206 of the Y_train data. (not including 206)

Here we convert X_train and Y_train to array type. Then we make our Y_train data with labels categorically as 9 classes. Then we print our Y_train data 101 and beyond. Afterwards, we first separate our data into 80% train and 20% test. Afterwards, we allocate 12.5% of this 80% train data as our validation data. In the last case, 70% of our data is reserved as train, 20% as test and 10% as validation.



```
Olcun_Satir_152120171109_Hw5.ipynb
Dosya Düzenle Göster Ekle Çalışma zamanı Araçlar Yardım Tüm değişiklikler kaydedildi

+ Kod + Metin
Bağlan

[ ] X_train = np.array(X_train)
    Y_train = np.array(Y_train)

[ ] Y_train = tf.keras.utils.to_categorical(Y_train, num_classes = 9)
    print(Y_train[101,:])

[1. 0. 0. 0. 0. 0. 0. 0. 0.]

Plot Digits

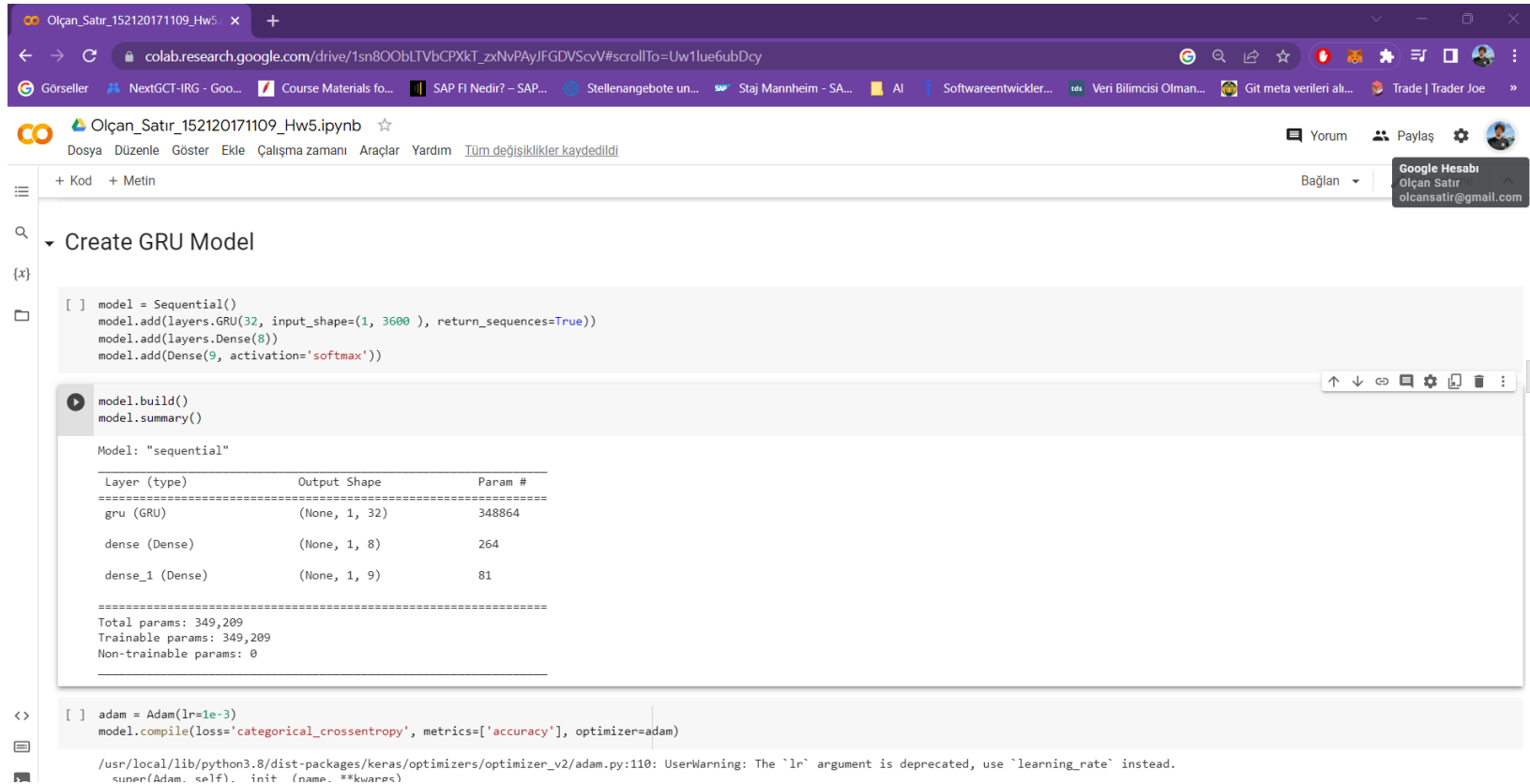
[ ] #Train-Test Split
    X_dev, X_test, Y_dev, Y_test = train_test_split(X_train, Y_train, test_size=0.2, shuffle=True, random_state=45) # % 80 train % 20 test

    X_train, X_val, Y_train, Y_val = train_test_split(X_dev, Y_dev, test_size=0.125, shuffle=True, random_state=45) # %70 train %10 val
    print(X_train.shape)
    print(X_val.shape)
    print(X_test.shape)
    print(Y_test.shape)

(654, 3600)
(94, 3600)
(188, 3600)
(188, 9)

[ ] X_train = np.expand_dims(X_train, axis=1)
    X_val = np.expand_dims(X_val, axis=1)
    X_test = np.expand_dims(X_test, axis=1)
    Y_train = np.expand_dims(Y_train, axis=1)
    Y_val = np.expand_dims(Y_val, axis=1)
    Y_test = np.expand_dims(Y_test, axis=1)
```

Here we create our GRU model. Then we add Dense. Since we have 9 classes, we give the value 9. Then we compile our model, give the loss function of our model as categorical_crossentropy and we used Adam as the optimization algorithm. Our learning rate: 1e-3



```
[ ] model = Sequential()
model.add(layers.GRU(32, input_shape=(1, 3600 ), return_sequences=True))
model.add(layers.Dense(8))
model.add(Dense(9, activation='softmax'))

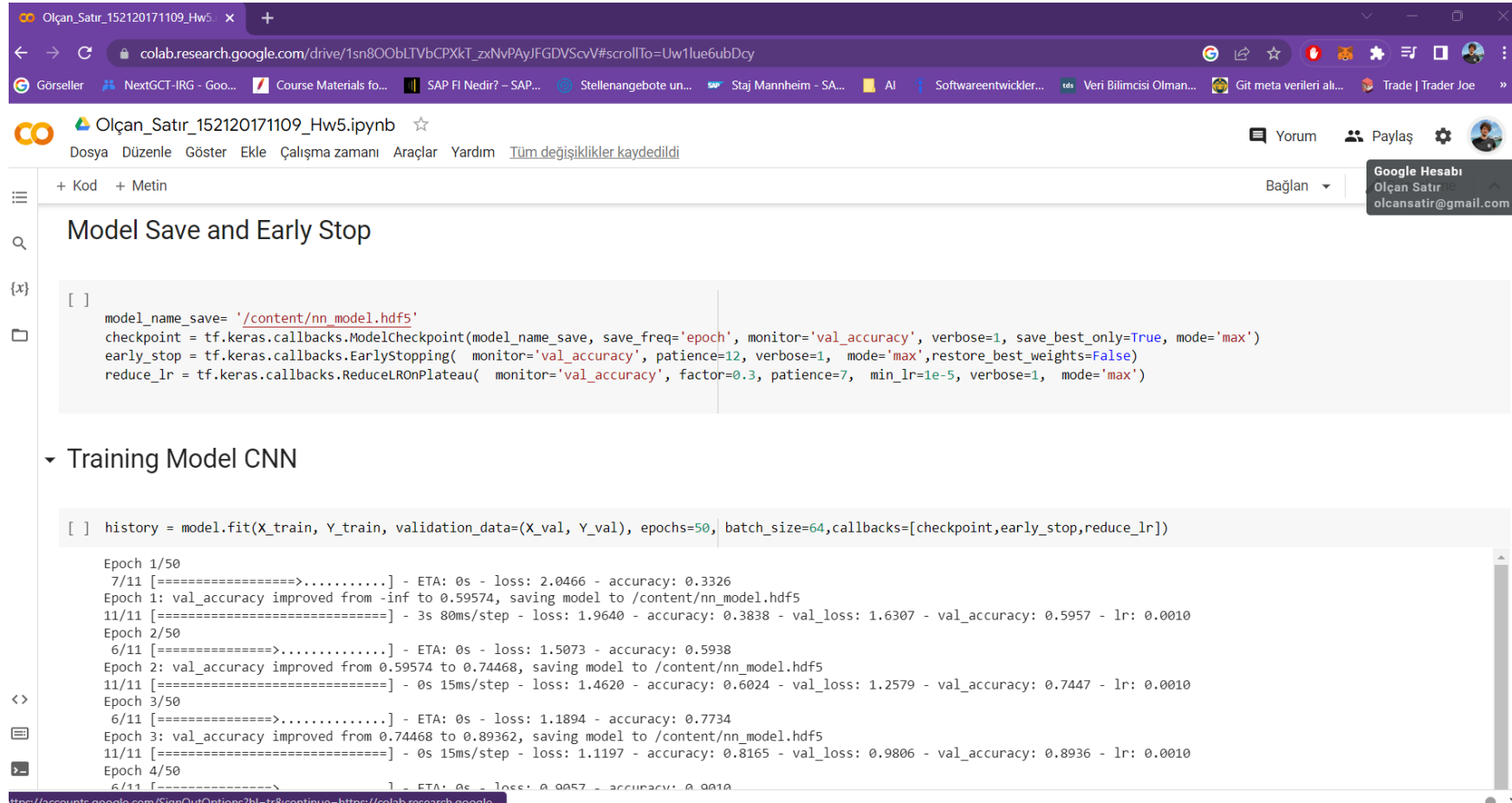
model.build()
model.summary()

Model: "sequential"
Layer (type)                Output Shape                Param #
=====
gru (GRU)                    (None, 1, 32)              348864
dense (Dense)                (None, 1, 8)               264
dense_1 (Dense)              (None, 1, 9)               81
=====
Total params: 349,209
Trainable params: 349,209
Non-trainable params: 0

[ ] adam = Adam(lr=1e-3)
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer=adam)

/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/adam.py:110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).init(name, **kwargs)
```

We used early stop here. If our model does not improve accuracy 12 times in a row, it stops and saves the model. Here, we work with an early stop in the 21st training for our model, which we run with 50 epochs, and save the model.



```
[ ] model_name_save= '/content/nn_model.hdf5'
checkpoint = tf.keras.callbacks.ModelCheckpoint(model_name_save, save_freq='epoch', monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=12, verbose=1, mode='max', restore_best_weights=False)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy', factor=0.3, patience=7, min_lr=1e-5, verbose=1, mode='max')
```

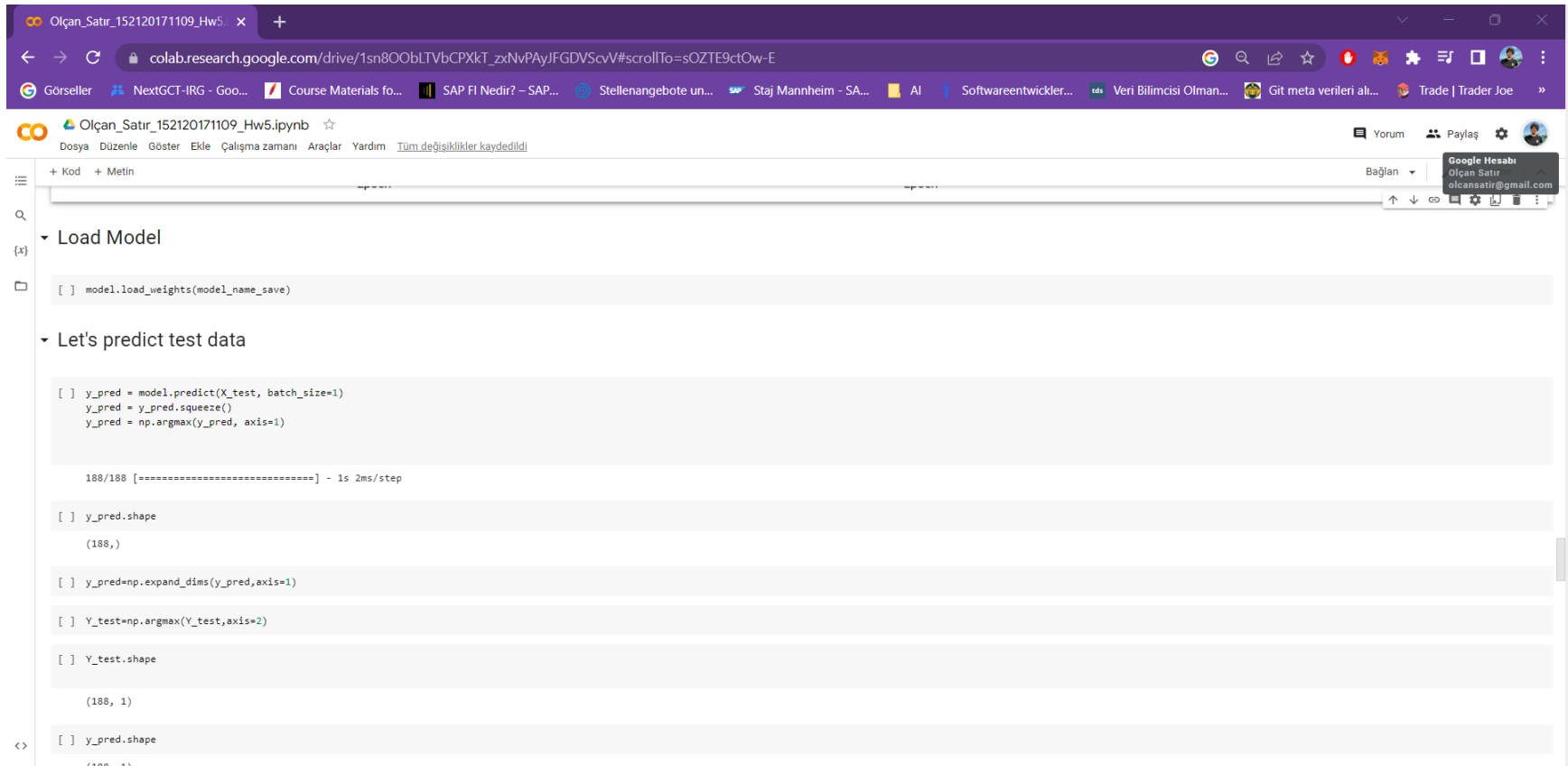
Training Model CNN

```
[ ] history = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), epochs=50, batch_size=64, callbacks=[checkpoint, early_stop, reduce_lr])
```

```
Epoch 1/50
7/11 [=====>.....] - ETA: 0s - loss: 2.0466 - accuracy: 0.3326
Epoch 1: val_accuracy improved from -inf to 0.59574, saving model to /content/nn_model.hdf5
11/11 [=====] - 3s 80ms/step - loss: 1.9640 - accuracy: 0.3838 - val_loss: 1.6307 - val_accuracy: 0.5957 - lr: 0.0010
Epoch 2/50
6/11 [=====>.....] - ETA: 0s - loss: 1.5073 - accuracy: 0.5938
Epoch 2: val_accuracy improved from 0.59574 to 0.74468, saving model to /content/nn_model.hdf5
11/11 [=====] - 0s 15ms/step - loss: 1.4620 - accuracy: 0.6024 - val_loss: 1.2579 - val_accuracy: 0.7447 - lr: 0.0010
Epoch 3/50
6/11 [=====>.....] - ETA: 0s - loss: 1.1894 - accuracy: 0.7734
Epoch 3: val_accuracy improved from 0.74468 to 0.89362, saving model to /content/nn_model.hdf5
11/11 [=====] - 0s 15ms/step - loss: 1.1197 - accuracy: 0.8165 - val_loss: 0.9806 - val_accuracy: 0.8936 - lr: 0.0010
Epoch 4/50
6/11 [=====>.....] - ETA: 0s - loss: 0.9057 - accuracy: 0.9010
```

```
11/11 [-----] - 0s 12ms/step - loss: 0.0014 - accuracy: 0.9999 - val_loss: 0.1412 - val_accuracy: 0.9681 - lr: 3.0000e-04
Epoch 19/50
 7/11 [=====>.....] - ETA: 0s - loss: 0.0519 - accuracy: 0.9978
Epoch 19: val_accuracy did not improve from 0.96809
11/11 [=====>.....] - 0s 12ms/step - loss: 0.0590 - accuracy: 0.9939 - val_loss: 0.1391 - val_accuracy: 0.9681 - lr: 3.0000e-04
Epoch 20/50
 7/11 [=====>.....] - ETA: 0s - loss: 0.0474 - accuracy: 0.9978
Epoch 20: val_accuracy did not improve from 0.96809
11/11 [=====>.....] - 0s 13ms/step - loss: 0.0568 - accuracy: 0.9939 - val_loss: 0.1378 - val_accuracy: 0.9681 - lr: 3.0000e-04
Epoch 21/50
 7/11 [=====>.....] - ETA: 0s - loss: 0.0558 - accuracy: 0.9933
Epoch 21: val_accuracy did not improve from 0.96809
11/11 [=====>.....] - 0s 12ms/step - loss: 0.0547 - accuracy: 0.9939 - val_loss: 0.1362 - val_accuracy: 0.9681 - lr: 3.0000e-04
Epoch 21: early stopping
```


Here we load our model that we recorded using early stopping and have it predicted. Then we make our model fit by adding a dimension because there is a mismatch in the dimensions.



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: `colab.research.google.com/drive/1sn80ObLTVbCPXkT_zxNvPAyJFGDVScvV#scrollTo=sOZTE9ctOw-E`. The notebook title is `Olcan_Satir_152120171109_Hw5.ipynb`. The left sidebar contains icons for file management, search, and a variable explorer. The main code area is divided into sections: **Load Model** and **Let's predict test data**. The code in the **Load Model** section is:

```
[ ] model.load_weights(model_name_save)
```

The code in the **Let's predict test data** section is:

```
[ ] y_pred = model.predict(X_test, batch_size=1)
y_pred = y_pred.squeeze()
y_pred = np.argmax(y_pred, axis=-1)

188/188 [=====] - 1s 2ms/step

[ ] y_pred.shape

(188,)
```

The output of the `y_pred.shape` command is `(188,)`. The next line of code is:

```
[ ] y_pred=np.expand_dims(y_pred,axis=1)
```

The output of this command is:

```
[ ] Y_test=np.argmax(Y_test,axis=2)

[ ] Y_test.shape

(188, 1)
```

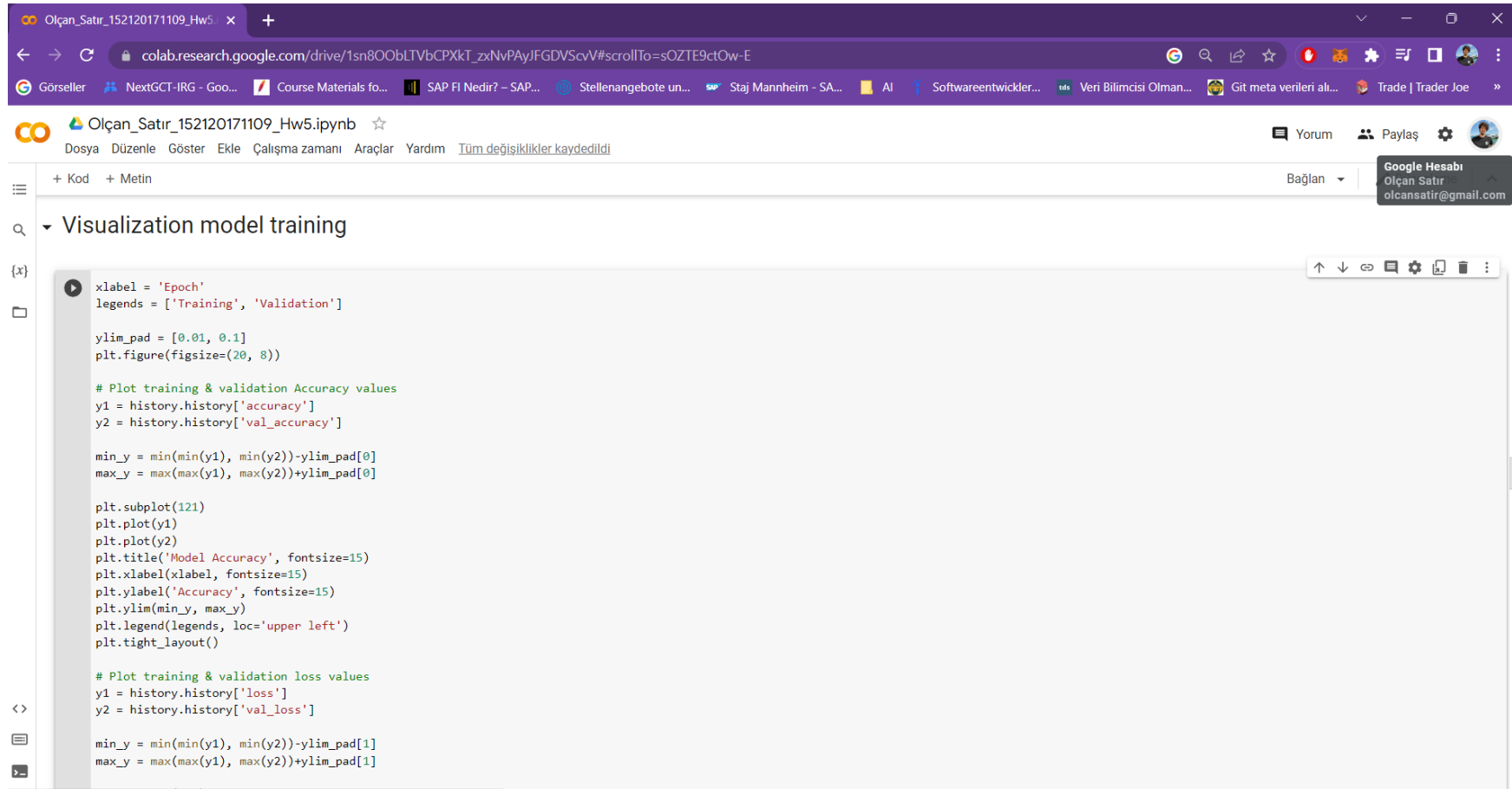
The final line of code is:

```
[ ] y_pred.shape

(188, 1)
```

The output of the final command is `(188, 1)`. The right sidebar shows a Google account dropdown menu with the name `Olcan Satir` and email `olcansatir@gmail.com`.

Here we plot the accuracy and loss values of our model's train and validation data.



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: `colab.research.google.com/drive/1sn80ObLTVbCPXkT_zxNvPAyJFGDVScvV#scrollTo=sOZTE9ctOw-E`. The notebook title is "Olçan_Satr_152120171109_Hw5.ipynb". The code is written in a Python cell and is titled "Visualization model training". The code plots training and validation accuracy and loss values.

```
xlabel = 'Epoch'
legends = ['Training', 'Validation']

ylim_pad = [0.01, 0.1]
plt.figure(figsize=(20, 8))

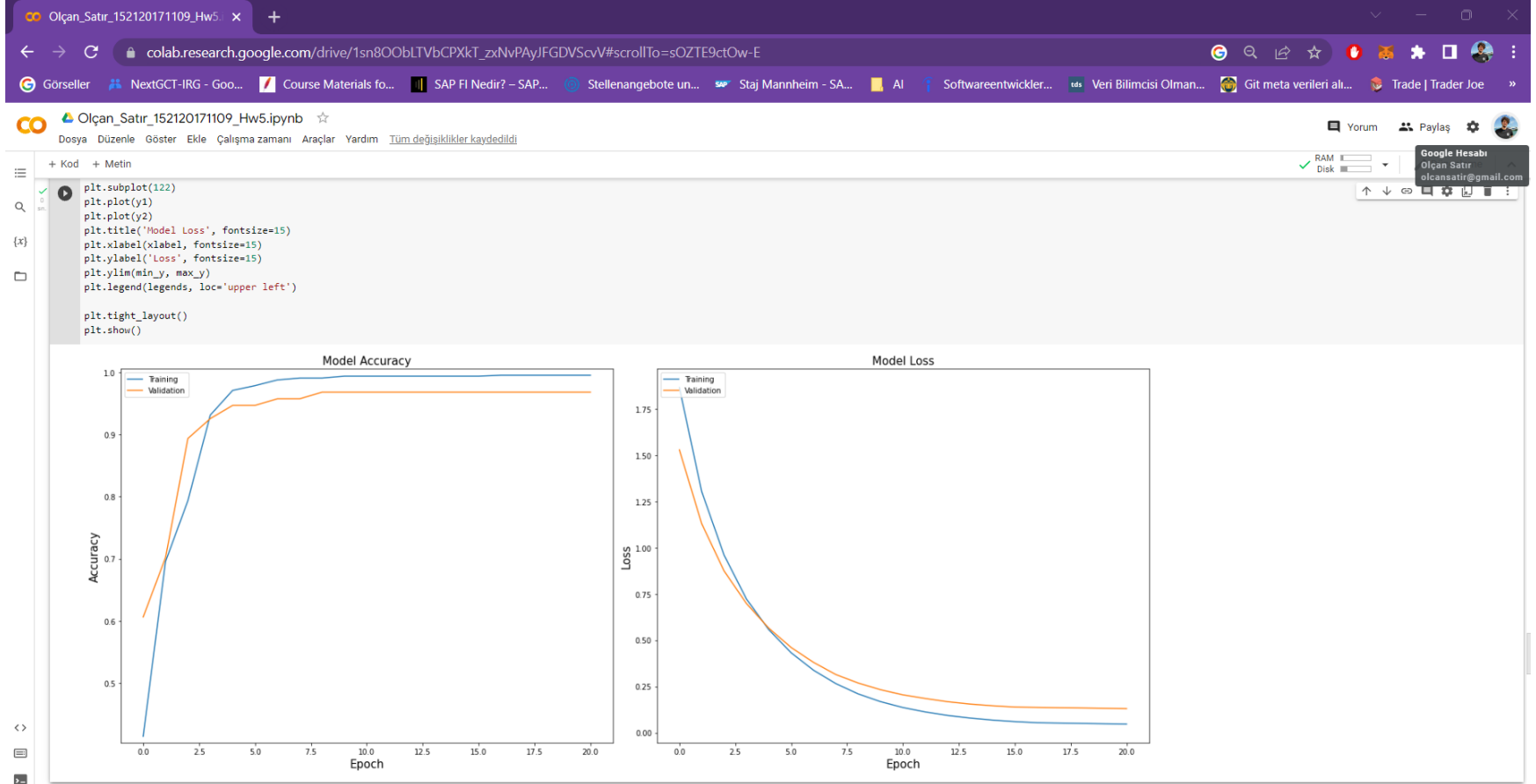
# Plot training & validation Accuracy values
y1 = history.history['accuracy']
y2 = history.history['val_accuracy']

min_y = min(min(y1), min(y2))-ylim_pad[0]
max_y = max(max(y1), max(y2))+ylim_pad[0]

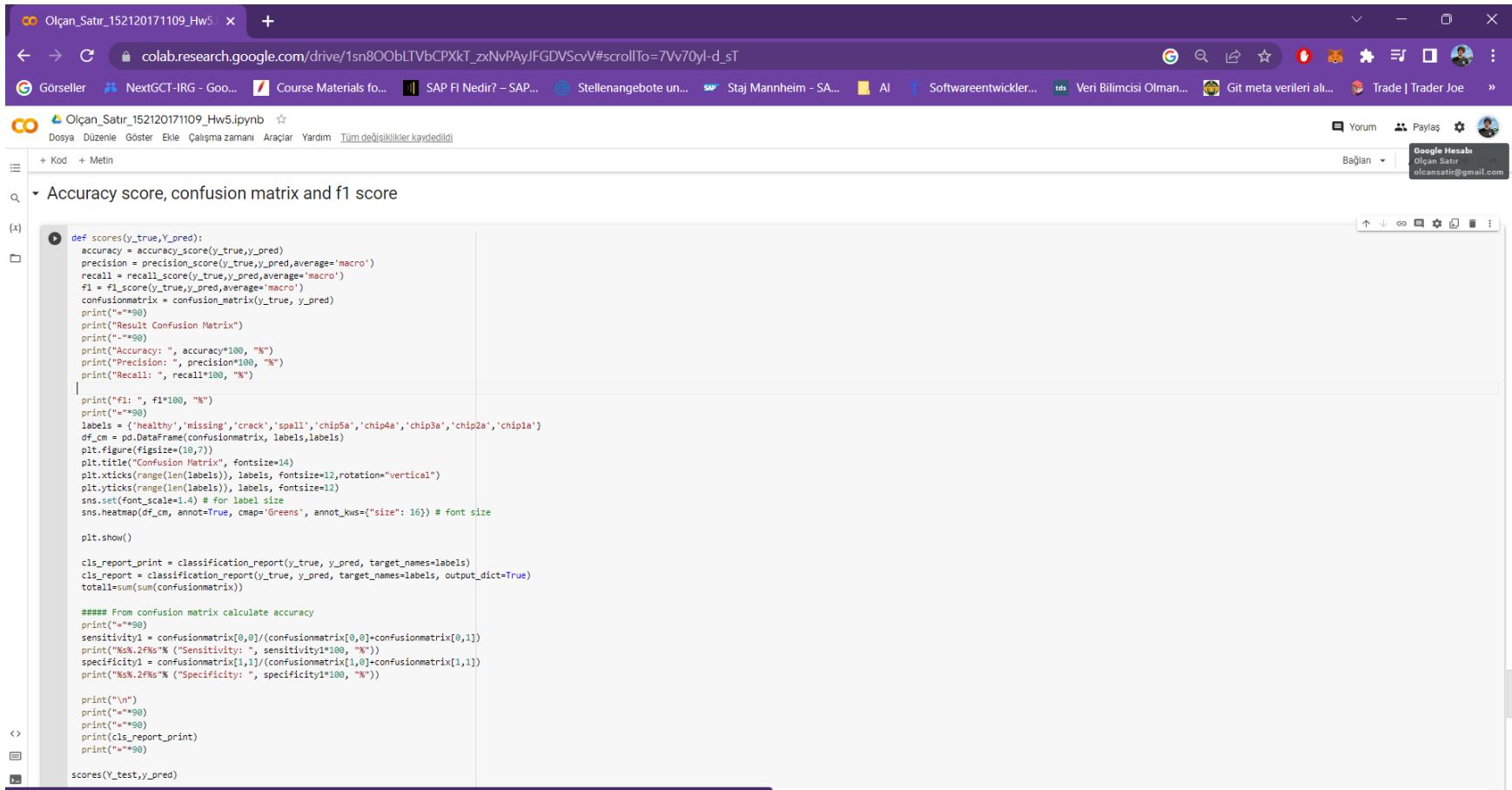
plt.subplot(121)
plt.plot(y1)
plt.plot(y2)
plt.title('Model Accuracy', fontsize=15)
plt.xlabel(xlabel, fontsize=15)
plt.ylabel('Accuracy', fontsize=15)
plt.ylim(min_y, max_y)
plt.legend(legends, loc='upper left')
plt.tight_layout()

# Plot training & validation loss values
y1 = history.history['loss']
y2 = history.history['val_loss']

min_y = min(min(y1), min(y2))-ylim_pad[1]
max_y = max(max(y1), max(y2))+ylim_pad[1]
```



Here we suppress our accuracy, precision, recall and f1 scores within the def scores function. Then we create our confusion matrix and give our labels. We use the seaborn library as requested.



```
def scores(y_true, Y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='macro')
    recall = recall_score(y_true, y_pred, average='macro')
    f1 = f1_score(y_true, y_pred, average='macro')
    confusionmatrix = confusion_matrix(y_true, y_pred)
    print("==90")
    print("Result Confusion Matrix")
    print("==90")
    print("Accuracy: ", accuracy*100, "%")
    print("Precision: ", precision*100, "%")
    print("Recall: ", recall*100, "%")
    print("F1: ", f1*100, "%")

    labels = ('healthy', 'missing', 'crack', 'spall', 'chip5a', 'chip4a', 'chip3a', 'chip2a', 'chip1a')
    df_cm = pd.DataFrame(confusionmatrix, labels, labels)
    plt.figure(figsize=(10,7))
    plt.title("Confusion Matrix", fontsize=14)
    plt.xticks(range(len(labels)), labels, fontsize=12, rotation="vertical")
    plt.yticks(range(len(labels)), labels, fontsize=12)
    sns.set(font_scale=1.4) # for label size
    sns.heatmap(df_cm, annot=True, cmap='Greens', annot_kws={"size": 16}) # font size

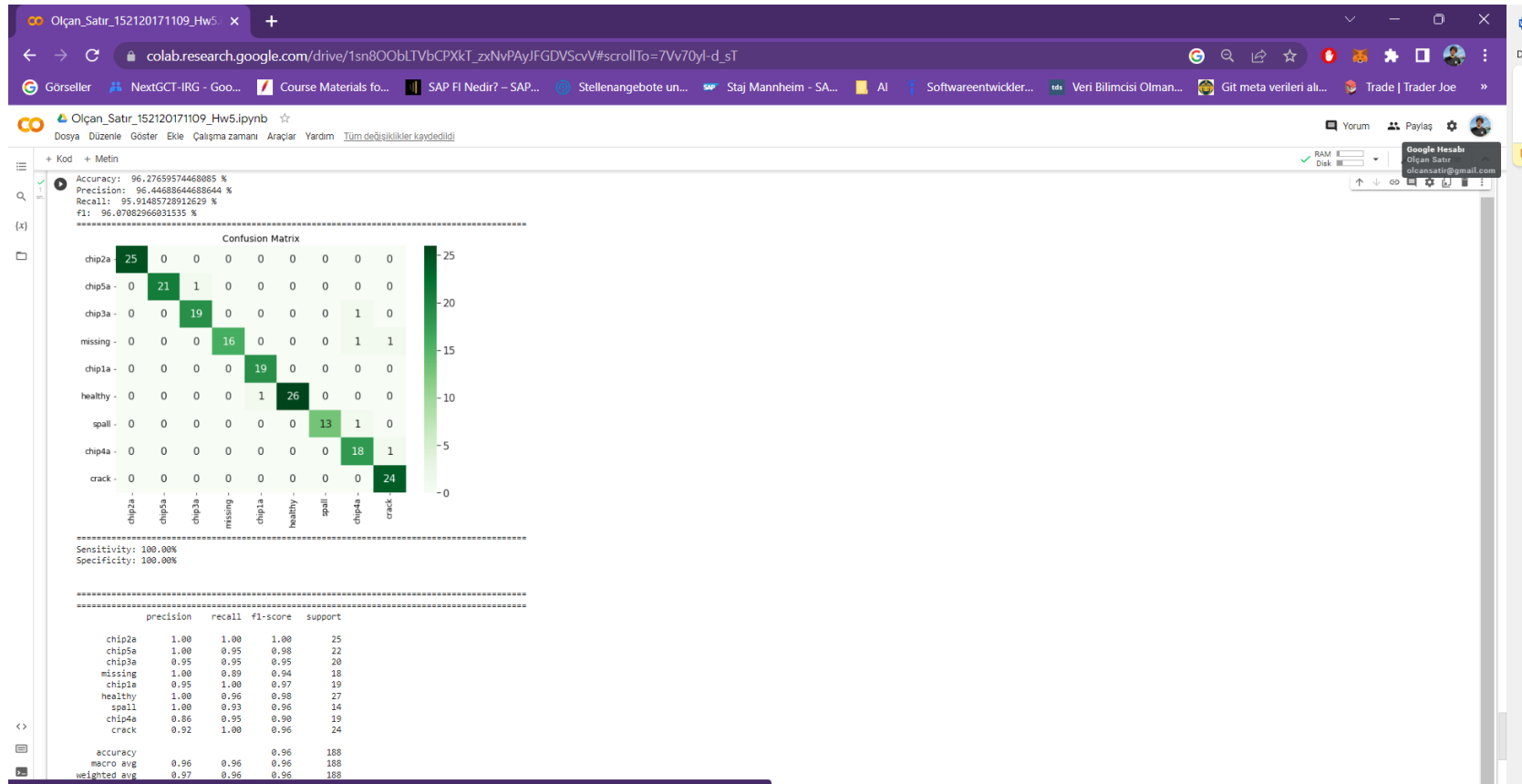
    plt.show()

    cls_report_print = classification_report(y_true, y_pred, target_names=labels)
    cls_report = classification_report(y_true, y_pred, target_names=labels, output_dict=True)
    totall=sum(sum(confusionmatrix))

    ##### From confusion matrix calculate accuracy
    print("==90")
    sensitivity1 = confusionmatrix[0,0]/(confusionmatrix[0,0]+confusionmatrix[0,1])
    print("%s%.2f%%" % ("Sensitivity: ", sensitivity1*100, "%"))
    specificity1 = confusionmatrix[1,1]/(confusionmatrix[1,0]+confusionmatrix[1,1])
    print("%s%.2f%%" % ("Specificity: ", specificity1*100, "%"))

    print("\n")
    print("==90")
    print("==90")
    print(cls_report_print)
    print("==90")

scores(Y_test, y_pred)
```



When we look at our confusion matrix, our model classified 25 of 25 test data belonging to the chip2a class correctly. It correctly classified 21 of the 22 test data belonging to the Chip5a class. It correctly classified 19 of the 20 test data belonging to the Chip3a class. He correctly classified 16 of the 18 test data belonging to the Missing class. It correctly classified 19 of our 19 test data belonging to Chip1a class. He classified 26 of our 27 test data belonging to the Healthy class correctly. He classified 13 of our 14 data belonging to the Spall class correctly. It correctly classified 18 of 19 test data belonging to Chip4a class. It correctly classified 24 of the 24 test data belonging to the Crack class. Our model has a 96% accuracy.