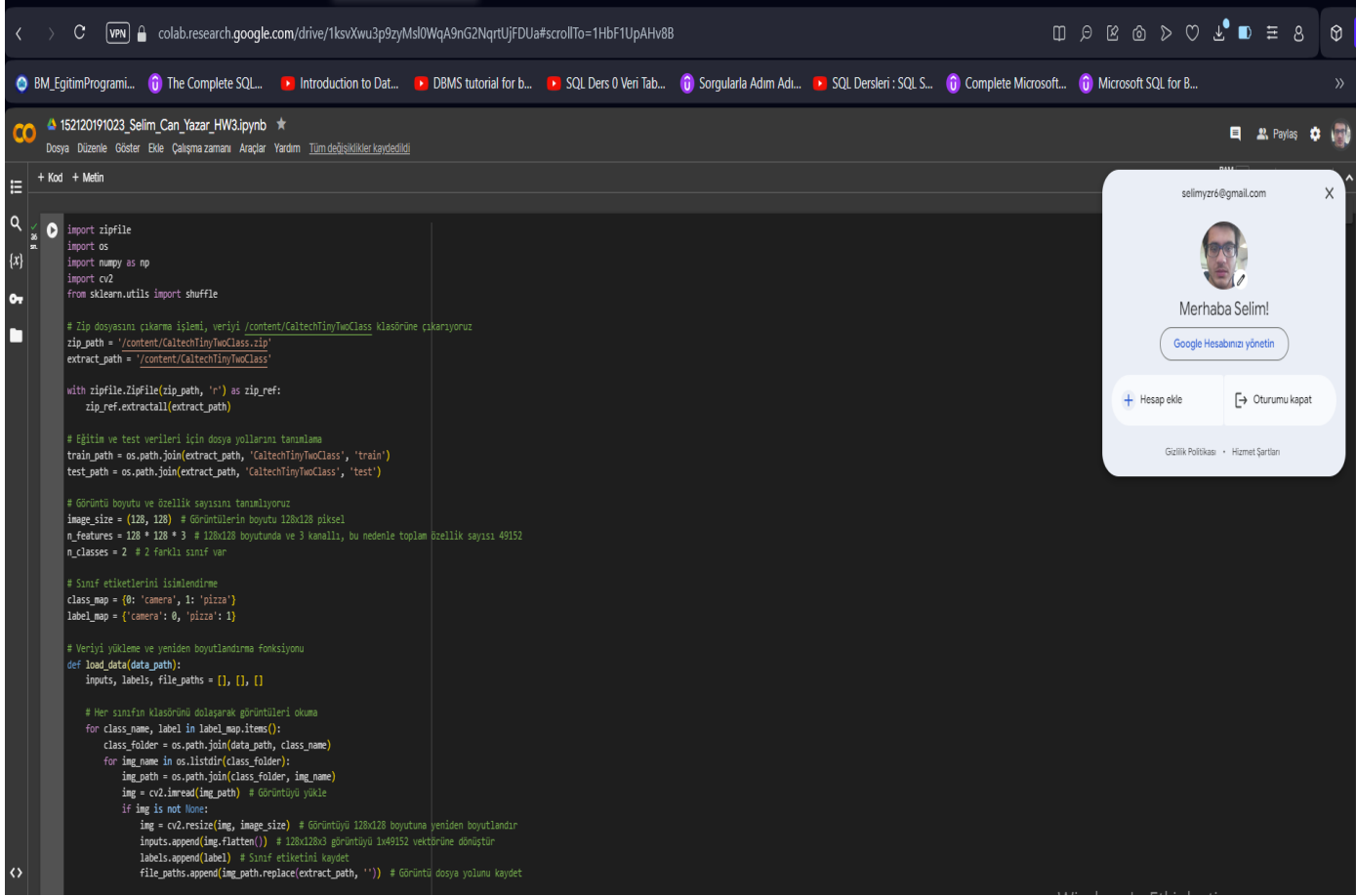


Deep Learning Homework 3 Raporu



```
import zipfile
import os
import numpy as np
import cv2
from sklearn.utils import shuffle

# Zip dosyasını çıkarma işlemi, veriyi /content/CaltechTinyTwoClass klasörüne çıkarıyoruz
zip_path = '/content/CaltechTinyTwoClass.zip'
extract_path = '/content/CaltechTinyTwoClass'

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

# Eğitim ve test verileri için dosya yollarını tanımlama
train_path = os.path.join(extract_path, 'CaltechTinyTwoClass', 'train')
test_path = os.path.join(extract_path, 'CaltechTinyTwoClass', 'test')

# Görüntü boyutu ve özellik sayısını tanımlıyoruz
image_size = (128, 128) # Görüntülerin boyutu 128x128 piksel
n_features = 128 * 128 * 3 # 128x128 boyutunda ve 3 kanallı, bu nedenle toplam özellik sayısı 49152
n_classes = 2 # 2 farklı sınıf var

# Sınıf etiketlerini isimlendirme
class_map = {0: 'camera', 1: 'pizza'}
label_map = {'camera': 0, 'pizza': 1}

# Veriyi yükleme ve yeniden boyutlandırma fonksiyonu
def load_data(data_path):
    inputs, labels, file_paths = [], [], []

    # Her sınıfın klasörünü dolayarak görüntüleri okuma
    for class_name, label in label_map.items():
        class_folder = os.path.join(data_path, class_name)
        for img_name in os.listdir(class_folder):
            img_path = os.path.join(class_folder, img_name)
            img = cv2.imread(img_path) # Görüntüyü yükle
            if img is not None:
                img = cv2.resize(img, image_size) # Görüntüyü 128x128 boyutuna yeniden boyutlandır
                inputs.append(img.flatten()) # 128x128x3 görüntüyü 1x49152 vektörüne dönüştür
                labels.append(label) # Sınıf etiketini kaydet
                file_paths.append(img_path.replace(extract_path, '')) # Görüntü dosya yolunu kaydet
```

Ödevde başlarken ilk önce google colaba .zip formatında Caltech veri setimizi yüklüyor ve ön işlemlere başlıyoruz. İlk olarak, 'zipfile', 'os', 'numpy' (np), 'cv2', ve 'sklearn.utils' kütüphaneleri kodun başında import ediliyor. Bu kütüphanelerin her biri, kodun farklı bölümlerinde önemli bir rol oynuyor. Örneğin, 'zipfile' kütüphanesi verilerin sıkıştırılmış bir zip dosyasından çıkarılmasında kullanılırken, 'os' kütüphanesi dosya yollarını dinamik olarak yönetmekte kullanılıyor. 'numpy' kütüphanesi, verilerin matrisler ve vektörler şeklinde işlenmesini sağlarken, 'cv2' (OpenCV) görüntü işleme adımlarında kullanılıyor. 'sklearn.utils' kütüphanesi ise verilerin karıştırılmasında (shuffle) kullanılarak modelin aşırı yüklenmesini (overfitting) engelleyerek daha genel ve çeşitli verilerle eğitilmesine yardımcı oluyor.

Kodun ilk bölümünde, '/content/CaltechTinyThreeClass.zip' adlı zip dosyası '/content/CaltechTinyThreeClass' yoluna çıkarılıyor. Bu işlem 'with zipfile.ZipFile()' yöntemi kullanılarak gerçekleştiriliyor ve zip dosyasındaki tüm veriler belirtilen yola çıkarılıyor. Bu adımdan sonra, çıkarılan veriler arasında eğitim ve test verilerinin

bulunduđu iki ayrı klasörün dosya yolları tanımlanıyor: 'train_path' ve 'test_path'. Bu dizinlerde, eğitim ve test için ayrılmış görüntüler yer alıyor.

Görüntülerin boyutu 128x128 piksel olarak belirlenmiş. Kodda 'image_size' değişkeni bu boyutu tanımlıyor ve görüntüler üç kanallı (RGB) olduğundan, her bir görüntü toplamda 49152 piksel değerine sahip. 'n_features' değişkeni bu özellik sayısını temsil ederken, 'n_classes' ise veride yer alan sınıf sayısını belirtiyor (burada 2 sınıf var: 'camera' ve 'pizza'). Bu sınıflar, 'class_map' adlı bir sözlük yapısında 0 ve 1 etiketleri ile isimlendirilmiş durumda.

Veri yükleme işlemi, 'load_data()' fonksiyonu aracılığıyla gerçekleştiriliyor. Bu fonksiyon, belirtilen 'data_path' dizinine giderek her sınıfın alt klasöründeki görüntüleri yüklüyor ve bu görüntüleri 128x128 boyutuna yeniden boyutlandırıyor. 'cv2.imread()' fonksiyonu kullanılarak her bir görüntü dosyası okunuyor ve 'cv2.resize()' ile yeniden boyutlandırılıyor. Görüntüler, daha sonra 'flatten()' yöntemi ile vektör haline getiriliyor (49152 elemanlı tek boyutlu bir vektör). Bu vektörleştirme işlemi, görüntülerin ileride bir sınıflandırma modeli ile işlenebilmesi için gerekli. Bu modelin daha verimli öğrenmesini sağlıyor ve hesaplamaları daha hızlı hale getiriyor.

Fonksiyon ayrıca her görüntüye uygun sınıf etiketini de 'labels' listesine ekliyor. Sınıf etiketleri, her sınıfın adını 'label_map' üzerinden alıyor ve görüntülerin yüklenmesi sırasında bu sınıflara karşılık gelen etiketler atanıyor. Bu etiketleme işlemi, daha sonra sınıflandırma modelinin hangi görüntünün hangi sınıfa ait olduğunu öğrenmesi için önemli.

Sonuç olarak kodun bu kısmında görüntü verilerini uygun bir şekilde yükleyip, boyutlandırıp, normalize ederek modelin eğitimi için hazır hale getiriyor. Böylece eğitim ve test süreçlerinde kullanılacak olan görüntüler, hem belirli bir boyut standardına hem de format uygunluğuna getiriliyor. Bu detaylı ön işleme süreci, veri kalitesinin yüksek olmasını ve sınıflandırma modelinin daha yüksek doğruluk oranı ile çalışmasını sağlamayı amaçlıyor.

```
# Tüm piksel değerlerini 0-1 arasında olacak şekilde normalleştir
inputs = np.array(inputs, dtype=np.float32) / 255.0
labels = np.array(labels, dtype=np.int32)
return inputs, labels, file_paths

# Eğitim verilerini yükle ve ön işleme
train_inputs, train_labels, _ = load_data(train_path)
train_inputs = np.hstack((train_inputs, np.ones((train_inputs.shape[0], 1)))) # Girişlere bias (1) ekleniyor (49152 + 1 = 49153 özellik)
train_inputs, train_labels = shuffle(train_inputs, train_labels) # Eğitim verilerini karıştırıyoruz

# Softplus aktivasyon fonksiyonunu tanımlama
def softplus(x):
    # Tasmaları önlemek için x değerini -500 ile 500 arasında sınırlandırıyoruz
    x = np.clip(x, -500, 500)
    return np.log(1 + np.exp(x)) # Softplus aktivasyon fonksiyonu

# Softplus fonksiyonunun türevini tanımlama
def der_softplus(x):
    # Tasmaları önlemek için x değerini -500 ile 500 arasında sınırlandırıyoruz
    x = np.clip(x, -500, 500)
    return 1 / (1 + np.exp(-x)) # Softplus türevi döndürülüyor

# Perceptron eğitim fonksiyonu - eğitim süreci burada başlıyor
def trainPerceptron(inputs, t, weights, rho, iterNo):
    # İterasyon sayısına göre eğitim döngüsünü başlat
    for _ in range(iterNo):
        inputs, t = shuffle(inputs, t) # Her iterasyonda veriyi karıştırıyoruz

        # Her bir eğitim örneğini sırayla işleme
        for i in range(inputs.shape[0]):
            x = inputs[i] # Giriş vektörü (49153 özellik, bias dahil)
            target = t[i] # Hedef etiket, -1 veya 1 olarak ayarlanacak

            # İleri besleme (feed-forward) adımı - ağırlık tahmini için ağırlıklar ve giriş vektörünü çarpıyoruz
            z = np.dot(weights, x) # z: giriş ve ağırlıkların çarpımı sonucu (bias dahil)

            # Geri besleme (feed-backward) adımı - gradyan inişi ile ağırlık güncellemesi
            gradient = rho * (-target) * der_softplus(-target * z) * x # x: Gradyan hesaplama
            weights -= gradient # Ağırlıkları gradyan değeriyle güncelliyoruz

        return weights

# Tek model eğitmek için ağırlıkları başlatma (rastgele küçük değerlerle)
weights = np.random.randn(n_features + 1) # 49153 ağırlık, bias dahil
```

Ödevin bu kısmında eğitim verilerinin yüklenmesi ve işlenmesi sonrası yapılan işlemler ve model eğitimi görülmektedir. İlk olarak eğitim verileri 'load_data()' fonksiyonu ile yüklenir ve ön işleme tabi tutulur. Bu adım sayesinde 'train_inputs' değişkenine yüklenen verinin 'np.hstack()' fonksiyonu kullanılarak girişlerine bir "bias" eklenir. Bias terimi lineer modele kaydırma ekleyerek modelin veriyi daha esnek bir şekilde öğrenmesini sağlar. Bu eklenen terim modelin doğruluğunu artırmamıza yardımcı olur. Daha sonra veriler 'shuffle()' fonksiyonu ile karıştırılarak eğitim sürecinde çeşitliliği sağlamak adına her iterasyonda tekrar tekrar rastgele bir şekilde sıralanır ve bu da overfitting (aşırı öğrenme) riskini azaltır.

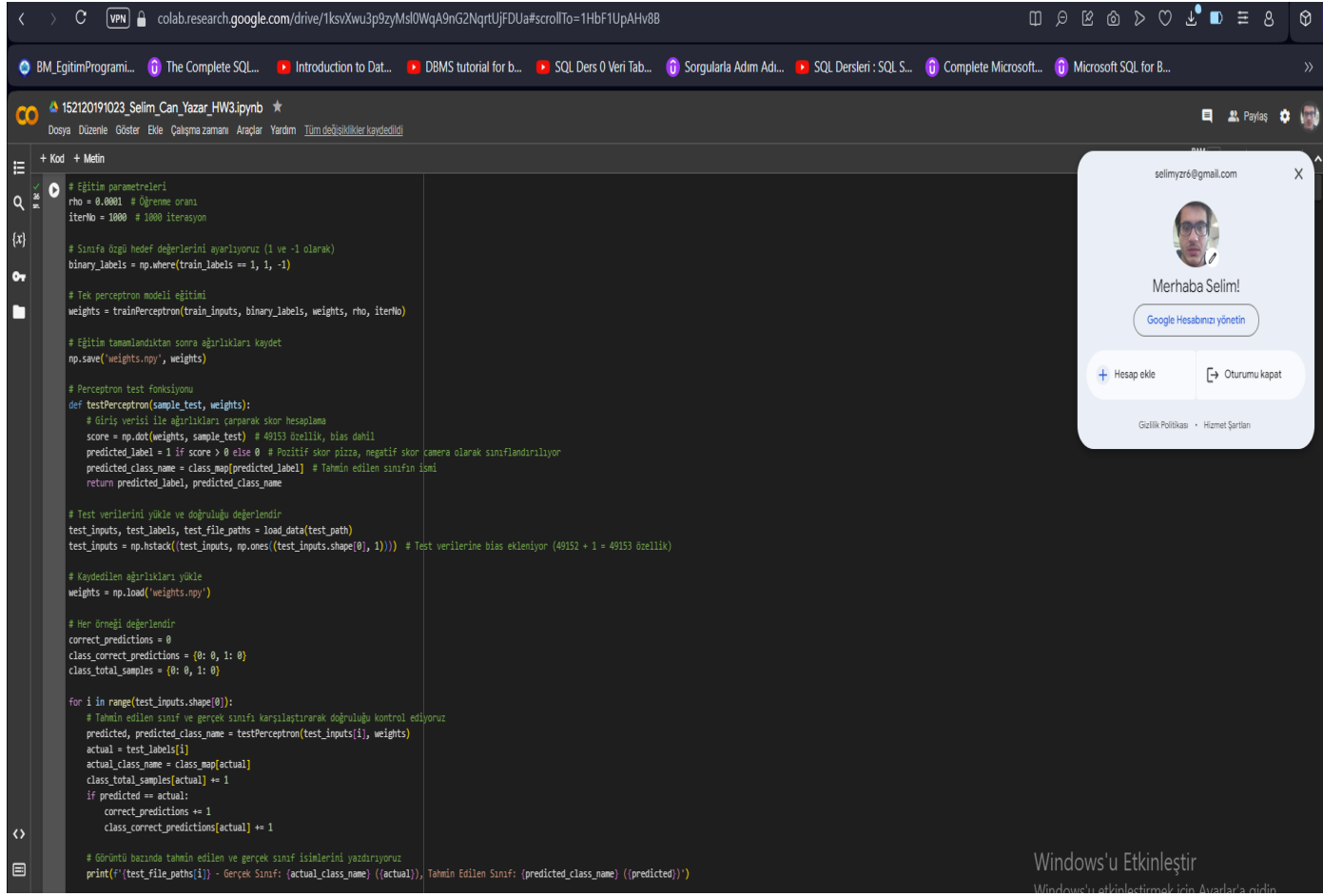
Sonraki kısımda, 'softplus' aktivasyon fonksiyonu ve bu fonksiyonun türevinin fonksiyonu tanımlanmıştır. Softplus aktivasyon fonksiyonu 'ReLU'(Rectified Linear Unit) aktivasyon fonksiyonuna benzeyen bir fonksiyondur, ancak daha yumuşak bir versiyonudur ve ağırlık hatalara karşı daha toleranslı olmasını sağlar. Bu fonksiyonun tanımı sırasında, taşmaları önlemek için 'x' değeri -500 ile 500 arasında sınırlandırılır ve ardından 'np.log(1 + np.exp(x))' formülü uygulanır. Bu sınırlandırma, çok büyük veya çok küçük değerlerin hesaplanmasında karşılaşılan sayısal taşmaları önlemek adına yapılmaktadır. Ayrıca, bu fonksiyonun türevi olan 'der_softplus(x)' tanımlanmış ve bu da ağırlık güncelleme adımlarında kullanılmıştır. Softplus'un türevi, sigmoid aktivasyon

fonksiyonuna benzeyen bir davranış gösterir ve ağın çıkışlarının türevsel olarak nasıl değişeceğini ifade eder.

Algılayıcı (Perceptron) modelin eğitimi 'trainPerceptron()' fonksiyonu ile gerçekleştirilir. Bu fonksiyon sayesinde belirli bir iterasyon sayısınca ('iterNo') çalışarak modelin parametrelerini günceller. Eğitim sürecinde ise her iterasyonda veriler rastgele karıştırılır, bu da modelin farklı örneklerle daha iyi genelleme yapmasına olanak tanır. Ardından, her eğitim örneği tek tek işlenir ve 'x' ile ifade edilen giriş vektörüne göre hedef etiket ('target') belirlenir. Bu hedef etiket belirli bir sınıf için 1 veya -1 olarak ayarlanır.

İleri besleme (feed-forward) aşamasında, giriş vektörü ve mevcut ağırlıklar çarpılarak bir 'z' değeri hesaplanır. Bu değer, modelin tahmin ettiği sonucu temsil eder. Geri besleme (feed-backward) aşamasında ise, gradyan inişi yöntemi kullanılarak ağırlık güncellemeleri yapılır. Bu adımda, 'gradient' değişkeni hesaplanarak mevcut ağırlıklar bu değere göre güncellenir. Gradyan değeri, öğrenme oranı ('rho') ile birlikte ağırlıkları ne kadar değiştirmemiz gerektiğini belirler. Softplus aktivasyon fonksiyonunun türevi, gradyan hesaplamasında kritik bir rol oynar; bu türev, ağın hatalarını minimize etmeye yönelik bir optimizasyon sağlar.

Bu ağırlık güncelleme adımları her iterasyonda tekrar edilir ve bu sayede model verileri daha iyi öğrenerek her sınıf için ağırlıkları günceller. Bu süreç, modelin veriye uygun şekilde optimize edilmesini ve farklı sınıfları ayırt edebilmesini sağlar. Sonuç olarak, bu kod bölümü, verilerin uygun şekilde ön işlenmesi, modelin eğitimine hazırlanması ve perceptron modelinin iteratif olarak güncellenmesi süreçlerini içerir. Modelin her iterasyonda verileri yeniden karıştırarak ve uygun aktivasyon fonksiyonları kullanarak daha iyi öğrenmesi sağlanır.



```
# Eğitim parametreleri
rho = 0.0001 # Öğrenme oranı
iterNo = 1000 # 1000 iterasyon

# Sınıfla ilgili hedef değerlerini ayarlıyoruz (1 ve -1 olarak)
binary_labels = np.where(train_labels == 1, 1, -1)

# Tek perceptron modeli eğitimi
weights = trainPerceptron(train_inputs, binary_labels, weights, rho, iterNo)

# Eğitim tamamlandıktan sonra ağırlıkları kaydet
np.save('weights.npy', weights)

# Perceptron test fonksiyonu
def testPerceptron(sample_test, weights):
    # Giriş verisi ile ağırlıkları çarparak skor hesaplama
    score = np.dot(weights, sample_test) # 49153 özellik, bias dahil
    predicted_label = 1 if score > 0 else 0 # Pozitif skor pizza, negatif skor camera olarak sınıflandırılıyor
    predicted_class_name = class_map[predicted_label] # Tahmin edilen sınıfın ismi
    return predicted_label, predicted_class_name

# Test verilerini yükle ve doğruluğu değerlendir
test_inputs, test_labels, test_file_paths = load_data(test_path)
test_inputs = np.hstack((test_inputs, np.ones((test_inputs.shape[0], 1)))) # Test verilerine bias ekleniyor (49152 + 1 = 49153 özellik)

# Kaydedilen ağırlıkları yükle
weights = np.load('weights.npy')

# Her örneği değerlendir
correct_predictions = 0
class_correct_predictions = {0: 0, 1: 0}
class_total_samples = {0: 0, 1: 0}

for i in range(test_inputs.shape[0]):
    # Tahmin edilen sınıf ve gerçek sınıf karşılaştırarak doğruluğu kontrol ediyoruz
    predicted, predicted_class_name = testPerceptron(test_inputs[i], weights)
    actual = test_labels[i]
    actual_class_name = class_map[actual]
    class_total_samples[actual] += 1
    if predicted == actual:
        correct_predictions += 1
        class_correct_predictions[actual] += 1

# Görüntü bazında tahmin edilen ve gerçek sınıf isimlerini yazdırıyoruz
print('?', test_file_paths[i]) - Gerçek Sınıf: (actual_class_name) (actual), Tahmin Edilen Sınıf: (predicted_class_name) ((predicted))'
```

Bu bölüm de veriler için bir Perceptron modelin oluşturulmasını ve bu modelin eğitilmesini ve test edilmesi anlatılmaktadır. İlk olarak, 'weights_list' adlı bir liste oluşturulur ve her sınıf için başlangıç ağırlıkları belirlenir. Bunlar rastgele değerlerdir ve her sınıf için ortak bir perceptron modeli eğitilir. Ağırlıklar her biri giriş vektörünün boyutuna (yani, (n_features + 1, önyargı terimi dahil) uygun olan 'np.random.randn()' fonksiyonu tarafından rastgele ayarlanır. Daha küçük rastgele değerler, modelin ilk başta belirli bir noktadan başlamasını ve eğitimin daha sorunsuz ilerlemesini sağlar.

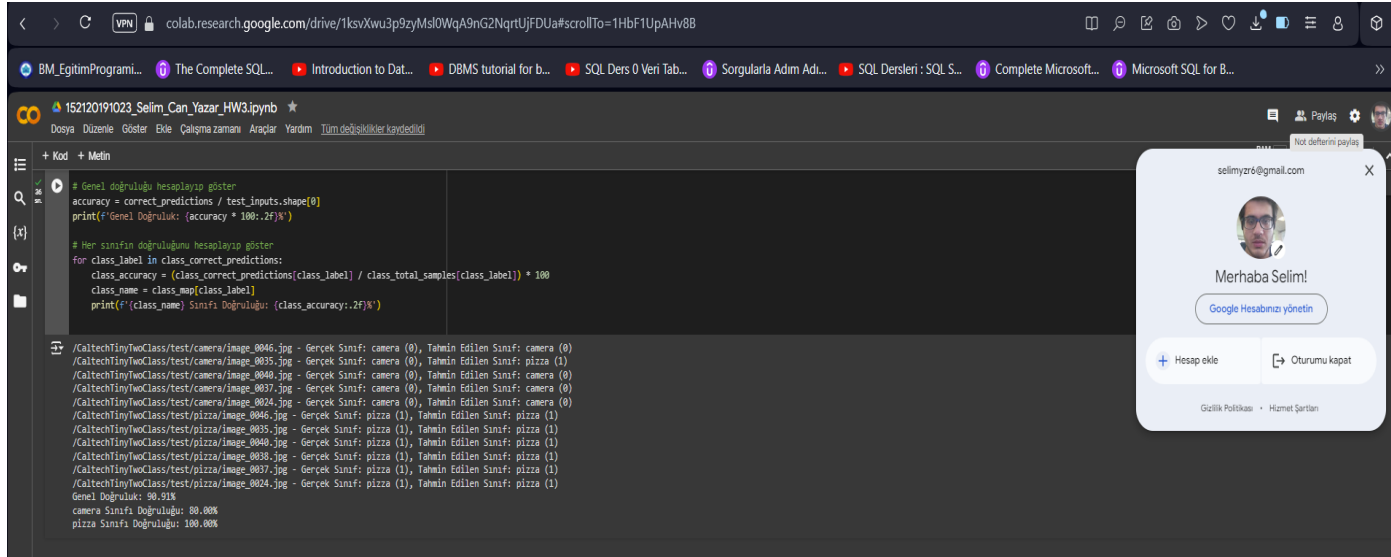
Ardından eğitim parametreleri gelir. 'rho' değişkeni öğrenme oranını tanımlar ve ağırlıkların gradyan inişi sırasında ne kadar güncellenmesi gerektiğini söyler. 'iterNo', toplam yineleme sayısının kısaltmasıdır ve modelin eğitim için sağlanan verileri kaç kez incelemesi gerektiğini söyler. Yukarıdaki iki parametre, modelin eğitimi sırasında performansı ve doğruluğu etkileyen önemli faktörlerden ikisi arasındadır.

Eğer örnek o sınıfa aitse etiket +1, değilse -1 olarak ayarlanır. Bu şekilde her sınıf için ikili bir sınıflandırma problemi yaratılır. 'trainPerceptron()' fonksiyonunun çağrıldığı yer burasıdır; işlemeden sonra, her sınıf 'weights_list'te saklanacak bir ağırlık vektörü üretir.

'np.save()' fonksiyonu ise eğitim tamamlandıktan sonra dosya sistemindeki her sınıf için elde edilen ağırlıkları kaydeder. Bu kayıtlar daha sonra modeli yeniden eğitmeden testler veya diğer işlemler için kullanabilmek için önemlidir. Bu şekilde, her sınıfın ağırlıkları daha sonra kullanılmak üzere saklanan 'weights.npy' adlı dosyalara kaydedilecektir.

Kodun devamında 'testPerceptron()' fonksiyonunu çağırarak testi gerçekleştirir. Bu fonksiyon sayesinde her sınıf için bir puan döndürmek üzere örnek test verisi ('sample_test') ve daha önce eğitilmiş ağırlıkların girdisini alır. Giriş vektörünün her sınıf ağırlığıyla çarpılmasıyla elde edilen değerler 'scores' listesinde saklanır. 'np.argmax(scores)' kullanımıyla, argmax en yüksek puanın indeksini döndürecek, dolayısıyla hangi sınıf olduğunu döndürecek ve bu etiketi 'predicted_label' olarak döndürecektir. Bu da bize bu etiketin sınıf adını ('predicted_class_name') verir. Bu şekilde test verilerini uygulayarak bir sınıf tahmin edilir.

Yine test verileri 'load_data()' kullanılarak yüklenir ve her girdiye bias eklenir. Bundan sonra, kaydedilen ağırlıklar 'np.load()' ile tekrar yüklenir ve test işlemleri bu ağırlıklarla yapılır. Test sırasında tahmin edilen sınıf ve doğru sınıf her örnek için eşleştirilir ve doğru tahminlerin sayısı sayılır. Yukarıdaki kod parçasında, 'correct_predictions' değişkeninde doğru tahmin edilen örneklerin sayısı ve 'class_correct_predictions' adlı sözlükteki her sınıfta doğru tahmin edilen örneklerin sayısı tutulur.



```
# Genel doğruluğu hesaplayıp göster
accuracy = correct_predictions / test_inputs.shape[0]
print("Genel Doğruluk: (accuracy * 100:.2f)%")

# Her sınıfın doğruluğunu hesaplayıp göster
for class_label in class_correct_predictions:
    class_accuracy = (class_correct_predictions[class_label] / class_total_samples[class_label]) * 100
    class_name = class_map[class_label]
    print("class_name Sınıfı Doğruluğu: (class_accuracy:.2f)%")

/Caltech101TwoClass/test/camera/image_0046.jpg - Gerçek Sınıf: camera (0), Tahmin Edilen Sınıf: camera (0)
/Caltech101TwoClass/test/camera/image_0035.jpg - Gerçek Sınıf: camera (0), Tahmin Edilen Sınıf: pizza (1)
/Caltech101TwoClass/test/camera/image_0040.jpg - Gerçek Sınıf: camera (0), Tahmin Edilen Sınıf: camera (0)
/Caltech101TwoClass/test/camera/image_0037.jpg - Gerçek Sınıf: camera (0), Tahmin Edilen Sınıf: camera (0)
/Caltech101TwoClass/test/camera/image_0024.jpg - Gerçek Sınıf: camera (0), Tahmin Edilen Sınıf: camera (0)
/Caltech101TwoClass/test/pizza/image_0046.jpg - Gerçek Sınıf: pizza (1), Tahmin Edilen Sınıf: pizza (1)
/Caltech101TwoClass/test/pizza/image_0035.jpg - Gerçek Sınıf: pizza (1), Tahmin Edilen Sınıf: pizza (1)
/Caltech101TwoClass/test/pizza/image_0040.jpg - Gerçek Sınıf: pizza (1), Tahmin Edilen Sınıf: pizza (1)
/Caltech101TwoClass/test/pizza/image_0038.jpg - Gerçek Sınıf: pizza (1), Tahmin Edilen Sınıf: pizza (1)
/Caltech101TwoClass/test/pizza/image_0037.jpg - Gerçek Sınıf: pizza (1), Tahmin Edilen Sınıf: pizza (1)
/Caltech101TwoClass/test/pizza/image_0024.jpg - Gerçek Sınıf: pizza (1), Tahmin Edilen Sınıf: pizza (1)
Genel Doğruluk: 90.91%
camera Sınıfı Doğruluğu: 80.00%
pizza Sınıfı Doğruluğu: 100.00%
```

Bu bölüm de modeli teste tabi tuttuk ve ne kadar iyi performans gösterdiğini inceledik. İkinci olarak ise model tahmini yapmak için her test örneği döngüden geçer. 'for i in range(testinputs.shape[0])' döngüsü test veri kümesindeki her örnek üzerinde işlemler gerçekleştirmek için kullanılır. Döngü tüm test görüntüleri için yinelenir ve her biri için görüntüyü “testPerceptron” fonksiyonuna geçirir ve döndürülen değer tahmin edilen sınıfın etiketi ve karşılık gelen sınıfın adıdır. Tahmin edilen 'predicted' sınıf etiketi, gerçek sınıf etiketi 'actual' ile karşılaştırılır ve doğru tahminlerin sayısı 'correctpredictions' değişkenine toplanır. Eğer tahmin doğruysa bu değişken artırılır. Ayrıca sınıf başına doğru tahminlerin toplamı 'classcorrectpredictions' sözlük yapısında saklanır. Böylece doğruluk yüzdeleri de sınıf başına hesaplanabilir.

Gerçek Sınıf : camera (0), Tahmin Edilen Sınıf: kamera (0) gibi çıktılar bize verilir . Bu bize modelin her örnekteki performansına daha yakından bakmamızı sağlar. Her tahminin çıktısını gözlemleyerek, modelin hangi sınıfları doğru veya yanlış sınıflandırdığını anlayabiliriz. Genel doğruluk, doğru tahmin edilen örneklerin toplam sayısının test setindeki toplam örnek sayısına bölünmesiyle hesaplanır. Toplam doğruluk olan 'accuracy' değişkeni, ekranda yüzde olarak gösterilir. Bu doğruluk, modelin test verileri üzerindeki ortalama performansdır ve modelin görülmemiş verilere ne kadar iyi genelleştirildiğinin bir ölçüsü olarak kullanılır. Burada modelin tüm model için doğru olduğu düşünüldüğünden, genel doğruluk oranı %90,91'dir, bu da birçok örneğin model tarafından doğru şekilde sınıflandırılabilceği anlamına gelir.

Daha sonra, her sınıfın doğruluğu belirlenir ve ekrana tek tek yazdırılır. Her sınıfın doğruluğu, “for classlabel in classcorrectpredictions” döngüsüyle hesaplanır ve “classaccuracy” değişkenine atanır. Bu hassasiyet, söz konusu sınıfın doğru tahmin edilen örneklerinin sayısının o sınıfın toplam örnek sayısına oranı olarak hesaplanır. Bu

şekilde, modelin sınıf başına performansını gözlemleyebiliriz. Örneğin, 'kamera' sınıfı için doğruluk puanı %80 iken, 'pizza' sınıfı için doğruluk %100'dür. Bu değerler, modelin her sınıftaki performansının göreceli bir karşılaştırmasını yapmamızı sağlar. Bazı sınıflar için, daha yüksek doğruluk, modelin bu sınıfları daha iyi seçmeyi öğrendiği anlamına gelir ve diğerleri için de bunu yapması gerekir.