BURSA TEKNİK ÜNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ VERİ MADENCİLİĞİ DERSİ PROJE RAPORU

Öğrenci Adı Soyadı: Ali Darı

Öğrenci Numarası: 20360859010

Projede Kullanılan Algoritma: Convolutional Neural Network (CNN)

Projede Kullanılan Veri Seti: CIFAR10

Öğretim Görevlisi: Doç. Dr. Erdem Yavuz

Araştırma Görevlisi: Sena Dikici

Sunum Linki: https://www.youtube.com/watch?v=xuhxakByJ9s

İçindekiler

Problem Özeti

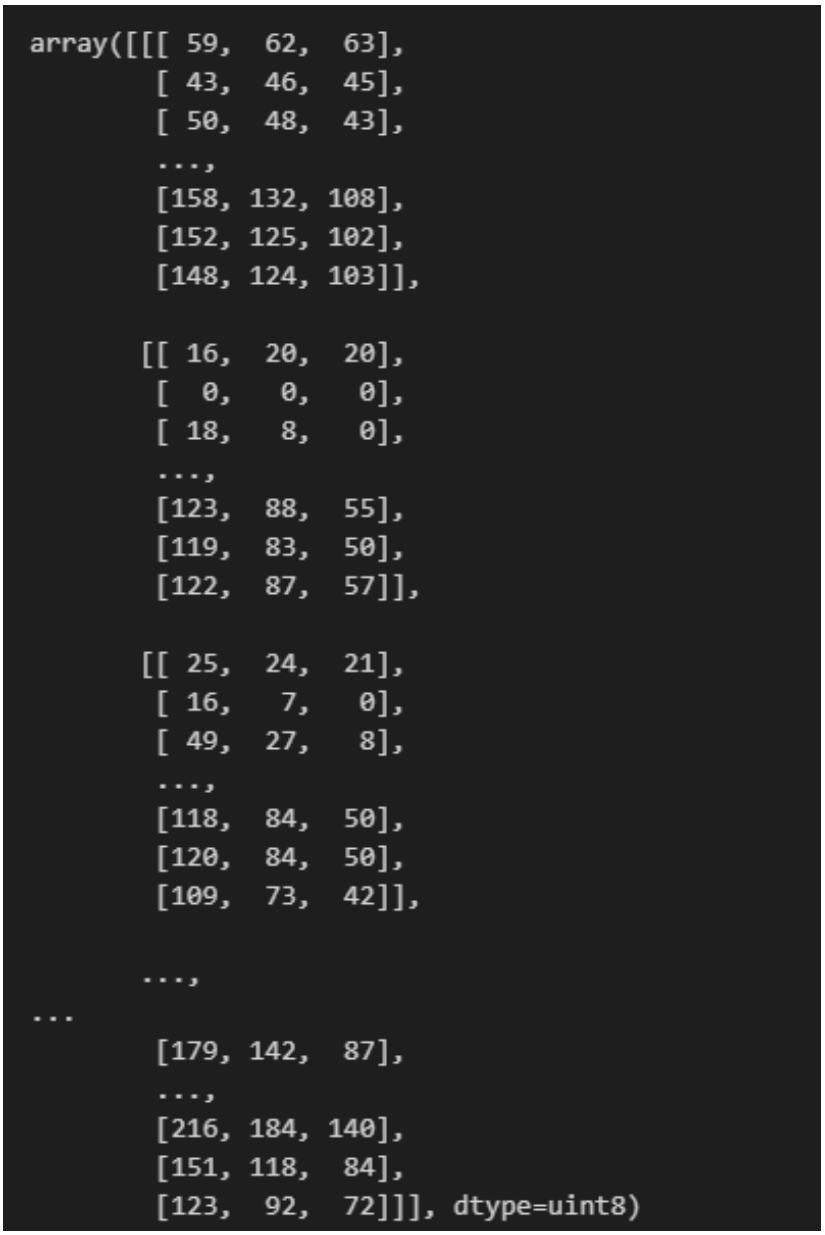
Bilgisayarla görme alanında, görüntülerin doğru bir şekilde sınıflandırılması, birçok uygulama için kritik bir öneme sahiptir. Örneğin, otonom araçlardan, güvenlik sistemlerine, sağlık teşhislerinden, sosyal medya platformlarındaki görsel içerik analizine kadar birçok alanda görüntü sınıflandırma teknikleri kullanılmaktadır. Bu projede, geniş çapta kullanılan CIFAR-10 veri seti üzerinde derin öğrenme tekniklerinden biri olan Convolutional Neural Network (CNN) ile görüntü sınıflandırma işlemi gerçekleştirilmiştir.

Veri Setini Anlamlandıralım

CIFAR-10 Veri Seti Yapısı

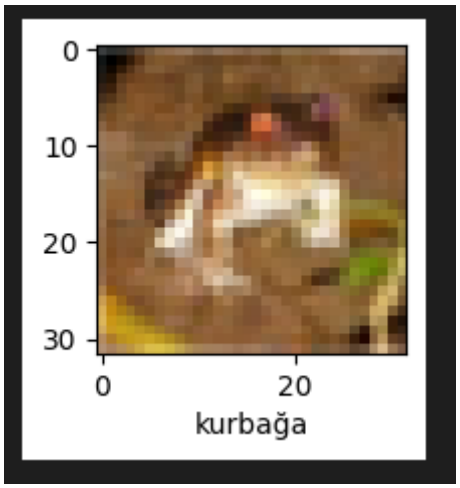
CIFAR-10 veri seti, her biri 32x32 piksel boyutunda 3 farklı renk kanalına (kırmızı, yeşil, mavi) sahiptir ve 10 farklı sınıfa ait 60,000 görüntüden oluşmaktadır. Veri seti 10 sınıfa sahiptir bu sınıflar; uçak, araba, kuş, kedi, geyik, köpek, kurbağa, at, gemi ve kamyondur. Veri seti, her sınıftan 6,000 görüntü içerir ve bu görüntüler eğitim ve test veri seti olarak ikiye ayrılmıştır. Eğitim seti 50,000, test seti ise 10,000 görüntüden oluşmaktadır.

Örnek bir veri şu şekildeki matris yapısına sahiptir *(Şekil 1)*



Şekil 1

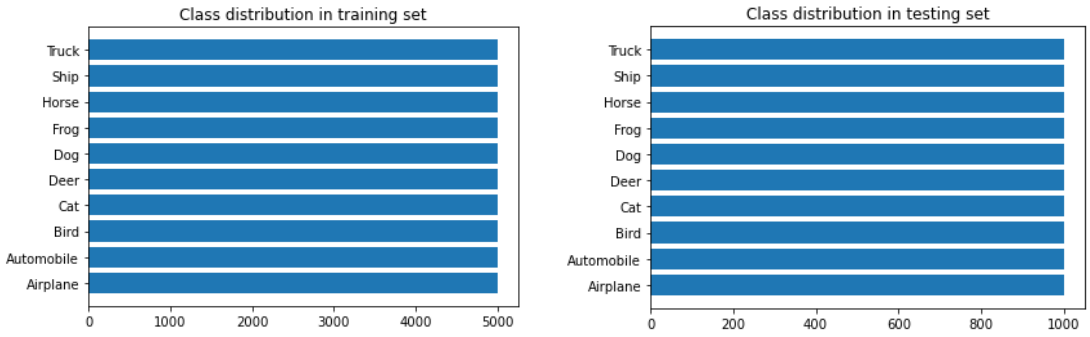
Bunu görselleştirecek olursak bu şekilde 32x32 boyutunda renkli bir görsel ortaya çıkmaktadır *(Şekil 2)*



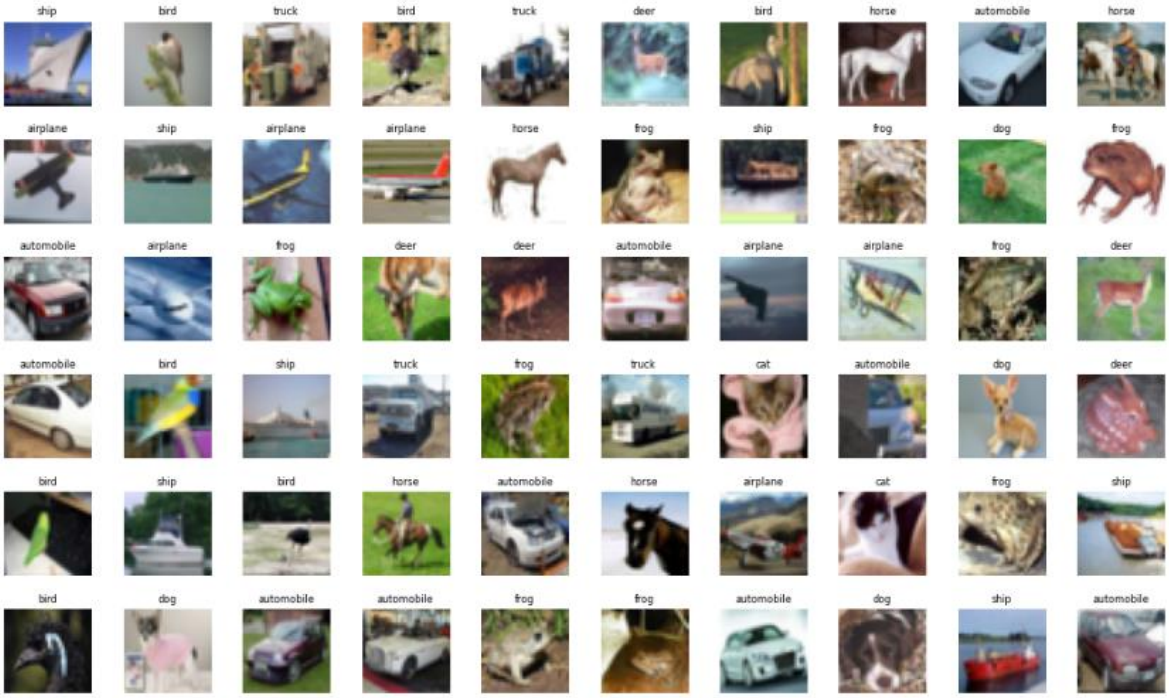
Şekil 2

Veri Seti Dağılımı

Veri setinde, bahsetmiş olunduğu üzere 10 sınıf mevcuttur. Veri setinde bu sınıflar eşit bir dağılım sergilemektedir. Her sınıfımızda, eğitim için 5000 ve test için 1000 adet olmak üzere bir sınıf için toplam 6000 örnek bulunmaktadır. *(Şekil 3)*



Şekil 3



Şekil 4

Görsellerin Özellikleri ve Zorluklar

Veri setindeki görseller, gerçek dünyadan alınmış ve uçak, araba, kuş, kedi, geyik, köpek, kurbağa, at, gemi ve kamyon olarak ayrılmıştır. Her sınıf, oldukça çeşitli görseller içermektedir; bu da veri setinin zengin ve gerçek dünya senaryolarını yansıtmasını sağlar. Ancak, görsellerin düşük çözünürlüklü ve bazen gürültülü olması, sınıflandırma görevini zorlaştıran önemli bir faktördür. Ayrıca, bazı sınıflar arasındaki görsel benzerlikler (örneğin, köpek ve kedi) modelin doğruluğunu olumsuz etkileyebilir. Görsellerdeki farklı açılar, pozlar ve arka plan varyasyonları da sınıflandırmayı zorlaştıran diğer etmenlerdir. Bu zorluklar, derin öğrenme modellerinin, özellikle CNN modelinin, bu veri seti üzerinde etkili bir şekilde çalışmasını gerektiren önemli unsurlardır.

Neden CNN?

Convolutional Neural Network (CNN) Tanımı

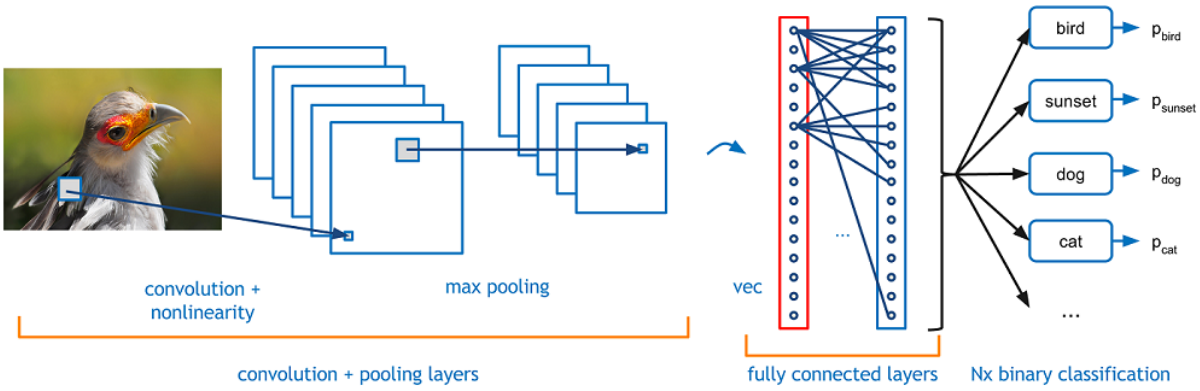
Convolutional Neural Networks (CNN), özellikle görüntü ve video işleme gibi veriler üzerinde etkili olan bir tür derin öğrenme mimarisidir. CNN'ler, görsellerin uzamsal ve derinliksel özelliklerini öğrenme yetenekleri sayesinde

klasik yapay sinir ağlarından (ANN) farklılık gösterirler. Bu ağlar, genellikle üç ana katman tipinden oluşur: evrişim (convolutional) katmanları, havuzlama (pooling) katmanları ve tam bağlı (fully connected) katmanlar. Evrişim katmanları, görüntüler üzerindeki yerel özellikleri algılamak için filtreler kullanır ve bu filtreler sayesinde kenar, köşe gibi düşük seviyeli özellikler ortaya çıkarılır. Havuzlama katmanları ise, uzamsal boyutları küçülterek hesaplama maliyetini azaltır ve modelin doğruluğunu artırır. Tam bağlı katmanlar, son aşamada, yüksek seviyeli özellikleri kullanarak sınıflandırma işlemini gerçekleştirir. Bu yapı sayesinde CNN'ler, görüntülerdeki yerel bağlantıları ve hiyerarşileri etkili bir şekilde öğrenir ve karmaşık görsel verileri yüksek doğrulukla sınıflandırabilir.

Evrişim katmanları, CNN'lerin temel yapı taşlarından biridir ve görüntülerdeki yerel özellikleri yakalamak için kullanılır. Bu katmanlarda, belirli bir filtre (kernel) matrisi, giriş görüntüsü üzerinde kaydırılarak (convolve edilerek) geçilir. Bu işlem sonucunda, giriş görüntüsünün farklı bölgelerindeki özellikleri (kenarlar, köşeler, dokular vb.) vurgulayan bir dizi özellik haritası (feature maps) oluşturulur. Her filtre farklı bir özellik öğrenir ve bu sayede model, görüntülerdeki karmaşık yapıları ve desenleri tanıyabilir. Evrişim katmanları, parametre paylaşımı ve lokal bağlantılar sayesinde hesaplama verimliliğini artırır ve modelin daha genel özellikler öğrenmesini sağlar.

Havuzlama katmanları, evrişim katmanlarının ardından gelerek uzamsal boyutları küçültür ve böylece modelin hesaplama yükünü azaltır. En yaygın kullanılan havuzlama türü, maksimum havuzlama (max pooling) olup, belirli bir bölgedeki maksimum değeri alır. Alternatif olarak, ortalama havuzlama (average pooling) da kullanılabilir ve bu yöntem, bölgedeki ortalama değeri alır. Havuzlama katmanları, modelin uzamsal hiyerarşileri öğrenmesine yardımcı olur ve pozisyon, ölçek veya deformasyon gibi değişikliklere karşı daha dayanıklı olmasını sağlar. Bu katmanlar ayrıca, overfitting riskini azaltarak modelin genelleme yeteneğini artırır.

Tam bağlı katmanlar, CNN'lerin son aşamasında yer alır ve sınıflandırma görevini gerçekleştiren katmanlardır. Bu katmanlarda, önceki katmanlardan elde edilen özellik haritaları, tek boyutlu bir vektör haline getirilir ve bu vektördeki her bir nöron, bir önceki katmandaki tüm nöronlara bağlanır. Bu katmanlar, yüksek seviyeli özellikleri bir araya getirerek, nihai sınıflandırma kararını verir. Tam bağlı katmanlar genellikle, softmax aktivasyon fonksiyonu kullanılarak, her bir sınıf için olasılık dağılımı sağlar. Bu katmanlar, modelin öğrenme kapasitesini artırır ve nihai tahminler için gerekli olan karmaşık ilişkileri öğrenir.



Şekil 5

CNN'nin Diğer Modellerden Üstünlükleri

Convolutional Neural Networks (CNN), özellikle görüntü işleme ve bilgisayarla görme alanlarında diğer derin öğrenme modellerine göre çeşitli üstünlüklere sahiptir. Bu üstünlükler, CNN'lerin mimarisinden ve veri ile etkileşim şekillerinden kaynaklanmaktadır.

CNN'ler, evrişim katmanları sayesinde yerel özellikleri öğrenir ve bu özellikleri tüm görüntü boyunca paylaşır. Bu durum, modelin daha az parametre ile daha verimli öğrenmesini sağlar. Parametre paylaşımı, aynı filtrelerin farklı yerlerde kullanılması anlamına gelir ve bu da hesaplama yükünü azaltır ve overfitting riskini düşürür.

Veri Ön İşleme

Verilerin Hazırlanması

Veri ön işleme işine başlamadan önce verilerimizi tanımamız gerekmektedir. Verilerin nasıl saklandığı, kaç boyutlu matrislerde tutulduğu, sınıflandırmalarının nasıl olduğu, … vb. gibi etkenler önemli noktalardır.

Verimizi tanımaya öncelikle verilerin boyutundan başlanıldı.

```
(X_train, y_train) , (X_test, y_test) = datasets.cifar10.load_data()
```

Şekil 6

Verileri yüklerken “Test” ve “Train” şeklinde varsayılan olarak bölümlenmiş gelmektedir. Test ve Train setlerinin içerisinde ise veriler ve onların sınıflandırmaları bulunmaktadır.

Verilerin boyutlarına bakacak olursak; train (eğitim) setinde 50.000 veri, test setinde ise 10000 veri bulunmaktadır

```
X_train.shape #32x32 pixel ve 3 renk kanalına sahip eğitim seti
```

```
(50000, 32, 32, 3)
```

```
X_test.shape #32x32 pixel ve 3 renk kanalına sahip test seti
```

```
(10000, 32, 32, 3)
```

Şekil 7

Veriler 32x32x3 boyutunda matrislerde tutulmaktadır. Matris içindeki her bir veri ise 0-255 arasındaki tam sayılardan oluşmaktadır.

Her bir verinin sınıflandırılması sayı cinsinden yapılmaktadır. Train seti için ilk 5 değerin sınıflandırmasına bakacak olursak her bir veri için ayrı bir matriste cevap yazmaktadır.

```
y_train[:5] #eğirim veri setinin hedef indisleri'nin ilk 5 değeri.
```

```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

Şekil 8

Her bir sayı bir sınıfı temsil etmektedir ve sınıf temsilleri ise şu şekildedir:

```
['0 : uçak', '1 : otomobil', '2 : kuş', '3 : kedi', '4 : geyik', '5 : köpek', '6 : kurbağa', '7 : at', '8 : gemi', '9 : kamyon']
```

Şekil 9

Normalizasyon ve Kategorize İşlemleri

Başlangıç olarak giriş verilerinin ölçeğini küçülterek normalizasyon uygulandı ve veriler 0-255 aralığından 0-1 aralığına ölçeklendi.

Veri setindeki değerleri normalize etme

```
X_train[0]/255 #normalde değerlerimiz 32x32x3 boyutlu dizi içerisinde 0-255 arasındaydı,
#bu değerleri 0-1 arasına getirmek için normalizasyon işlemi uyguladık
```

```
array([[[0.00090734, 0.00095348, 0.00096886],
        [0.00066128, 0.00070742, 0.00069204],
        [0.00076894, 0.00073818, 0.00066128],
        ...,
        [0.00242983, 0.00202999, 0.0016609 ],
        [0.00233756, 0.00192234, 0.00156863],
        [0.00227605, 0.00190696, 0.00158401]],

       [[0.00024606, 0.00030757, 0.00030757],
        [0.        , 0.        , 0.        ],
        [0.00027682, 0.00012303, 0.        ],
        ...,
        [0.00189158, 0.00135333, 0.00084583],
        [0.00183007, 0.00127643, 0.00076894],
        [0.0018762 , 0.00133795, 0.00087659]],

       [[0.00038447, 0.00036909, 0.00032295],
        [0.00024606, 0.00010765, 0.        ],
        [0.00075356, 0.00041522, 0.00012303],
        ...,
        [0.00181469, 0.00129181, 0.00076894],
        [0.00184544, 0.00129181, 0.00076894],
        [0.00167628, 0.00112265, 0.00064591]],

       ...,
...
        [0.00275279, 0.00218378, 0.00133795],
        ...,
        [0.0033218 , 0.00282968, 0.00215302],
        [0.00232218, 0.00181469, 0.00129181],
        [0.00189158, 0.00141484, 0.00110727]]])
```

Şekil 10

Bu işlem hem eğitim setine hem de test setine uygulandı.

Çok sınıflı sınıflandırma problemlerinde, modelin çıktısı her sınıf için bir olasılık değerinden oluşur. Bu durumda, etiketlerin de aynı formatta olması gerekir. Bu sebepten dolayı etiketler kategorize edilmelidir. Bu çalışmada "one-hot encoding" yöntemiyle etiketler kategorize edildi. Bu yöntemde sayılar 0 ve 1 cinsinden bit gösterimi şeklinde gösterilmektedir.

Kendi örneğimiz için 10 sınıflı bir problemde ilk sınıfımızın etiketi "6" olarak belirtilmiş, bu sayıyı kategorize edersek şu şekilde gösterilir; [0,0,0,0,0,0,1,0,0,0] burada ilk indeks 0 olarak alınmıştır.

```
y_cat_train = to_categorical(y_train, 10)
y_cat_test = to_categorical(y_test, 10)
```

Şekil 11

Yukarıdaki şekilde "to_categorical" fonksiyonu ile 10 sınıflı bir problem için etiketler sınıflandırılmıştır.

```
y_cat_test

array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.]])
```

Şekil 12

Model Oluşturma

CNN Model Mimarisi

Yukarıda CNN katmanlarından bahsedildi, bu katmanları kullanarak cifar10 veri seti için şu şekilde bir model oluşturuldu

Genel olarak bu model, CIFAR-10 veri setindeki görüntüleri sınıflandırmak için katmanlar halinde düzenlenmiş bir dizi evrişim, havuzlama ve tam bağlantılı katmanlardan oluşur. Her bir evrişim katmanı, görüntüdeki özellikleri çıkarmak için filtreler kullanır, havuzlama katmanları bu özellik haritalarını özetler ve boyutlarını küçültür ve tam bağlantılı katmanlar bu özellikleri kullanarak nihai sınıflandırma kararını verir. Model, nihai olarak 10 sınıfın her biri için bir olasılık değeri döndüren softmax katmanıyla sonlanır.

MODEL OLUŞTURMA

```
cnn = models.Sequential([


    #cnn adımları

    layers.Conv2D(filters=16,kernel_size = (3,3), activation = 'relu',input_shape = (32,32,3)),
    layers.MaxPooling2D((2,2)),

    layers.Conv2D(filters=32,kernel_size = (3,3), activation = 'relu'),
    layers.MaxPooling2D((2,2)),

    layers.Conv2D(filters=64,kernel_size = (3,3), activation = 'relu'),
    layers.MaxPooling2D((2,2)),

    #dense
    layers.Flatten(),
    layers.Dense(1000, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

Şekil 13

Katmanların Açıklaması

İlk olarak Cnn katmanlarından olan evrişim (convolution) ve havuzlama (pooling) katmanları kullanıldı. “Conv2D” metodu modelimize bir evrişim katmanı eklenmesini sağlıyor.

Parametrelerine bakacak olunursa;

- “filters” parametresi, kaç farklı filtre kullanılacağının sayısını belirtir
- “kernel_size” parametresi, kullanılacak filtrelerin boyutlarını belirtir
- “activation” parametresi, hangi aktivasyon fonksiyonunun kullanılacağını belirtir
- “input_shape” parametresi, giriş verisinin boyutunu belirtir

Bu katmanı eklerken 16 farklı filtre uygulanması istendi, bu filtrelerin boyutu 3x3 şeklinde oluşturuldu, aktivasyon fonksiyonu olarak “relu” fonksiyonu kullanıldı ve giriş boyutu olarak 32x32x3 boyutlu bir giriş değerinin olduğu belirtildi

Sonrasında eklenen katman havuzlama katmanı. Bu katmanı eklemek için MaxPooling2D metodu kullanıldı

Parametresine bakacak olunursa; kaç boyutlu bir havuzlama penceresi kullanılacağını belirtir. Bu model için 2x2 boyutunda bir havuzlama penceresi uygun görüldü. Havuzlama penceresi, her 2x2 penceredeki maksimum değeri alır.

Sonrasında ise evrişim ve havuzlama katmanları ikişer kez daha uygulandı. İlkinde 16 farklı filtre uygulanan evrişim katmanında ikincide 32 üçüncü ve son evrişimde 64 farklı filtre uygulandı.

Birden fazla evrişim ve havuzlama katmanının uygulanmasının bazı sebepleri şunlardır; İlk evrişim katmanları, kenarlar, köşeler ve dokular gibi düşük seviyeli özellikleri çıkarır. Daha derin evrişim katmanları, şekiller ve nesnelerin parçaları gibi daha karmaşık özellikleri öğrenir. En derin katmanlar, nesnelerin tamamını ve daha soyut özellikleri tanır.

Evrişim ve havuzlama katmanlarının ardından Flatten (Düzleştirme) katmanı uygulandı. Flatten (Düzleştirme) katmanı, önceki katmanlardan elde edilen çok boyutlu özellik haritalarını tek boyutlu bir vektöre dönüştürür.

Ardından tek boyutlu bir vektöre dönüştürülen veriler sinir ağlarından geçmeye hazır hale gelir.

- İlk eklenen katman, 1000 nörondan oluşur ve “ReLU” aktivasyon fonksiyonu ile aktivasyonlanır. Bu, gizli katmanlardan biridir ve modelin içsel temsillerini oluşturur.
- İkinci eklenen sinir ağı katmanı ise, 10 nörondan oluşur ve softmax aktivasyon fonksiyonu ile aktivasyonlanır. Bu, modelin çıkış katmanıdır ve modelin sınıflandırma sonuçlarını üretir.

10 farklı sınıflandırma çeşidi bulunduğu için çıkış katmanımız 10 adet nörondan oluşmaktadır ve Softmax aktivasyon fonksiyonu, her bir sınıf için olasılık değerleri sağlar.

Modelin eğitim sürecine geçilmeden önce derlenmesi gerekmektedir. Model derlenirken bazı parametreler alır. Bu model derlenirken optimizer, loss ve metrics parametresi kullanılmıştır.

```
cnn.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
```

Şekil 14

- Optimizer parametresi, bu parametre eğitim sürecinde ağırlıkları güncellemek için kullanılan bir optimizasyon algoritmasıdır. Bu model için oldukça popüler olan “Adam (Adaptive Moment Estimation)” algoritması kullanılmaktadır
- Loss (Kayıp Fonksiyonu), bu kayıp fonksiyonu, çok sınıflı sınıflandırma problemlerinde kullanılır ve doğru sınıfın olasılığını maksimize etmeyi hedefler.
- Metrics (İzlenecek Metrikler), burada model eğitilirken izlenecek metrikler yer alır. Accuracy (Doğruluk), modelin performansını değerlendirmek için kullanılan bir metriktir. Doğruluk, modelin doğru tahmin ettiği örneklerin toplam örneklere oranını ifade eder

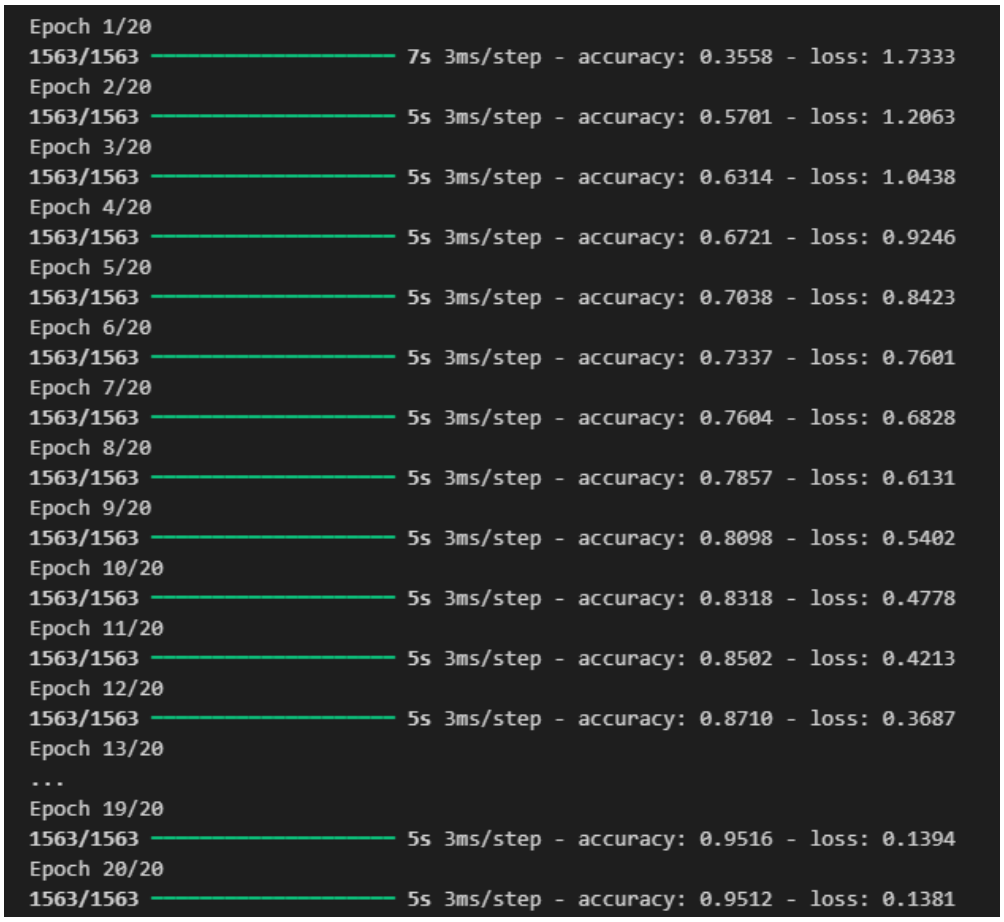
Modelin Eğitimi

Model eğitim için hazır duruma getirildi, sıradaki aşama modelin eğitimini gerçekleştirmek. Modelin eğitimini gerçekleştirmek için “fit” fonksiyonu kullanılır ve içerisine eğitim için kullanılan verileri, etiketleri ve kaç aşamalı eğitim olacağı belirtilir.

```
cnn.fit(X_train,y_train,epochs=20)
```

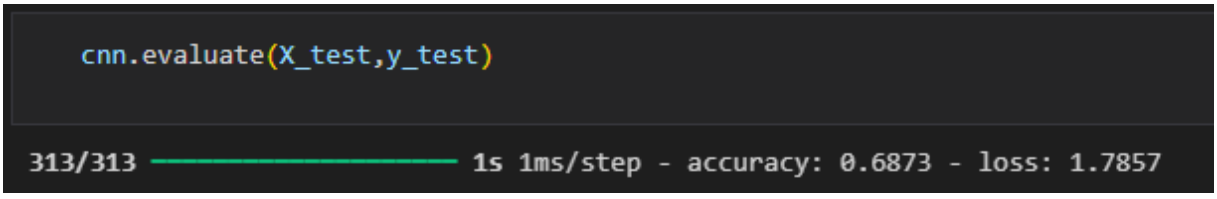
Şekil 15

Model 20 aşamalı bir eğitimden geçmektedir ve belirli çıktı sonuçları göstermektedir.



Şekil 16

Model eğitilirken doğruluk oranı 0.95 oranına kadar çıkmakta. Fakat eğitim seti içinde kullanılan metrikler bizi yanıltabilir. Nihai sonucu görmek için test veri setiyle doğrulama yapmamız gerekmektedir. Bu işlem "eveluate" fonksiyonu ile gerçekleşir ve içerisine test verileri, etiketleri gönderilir.



Şekil 17

Çıkan sonuçlara bakacak olursak test veri seti için doğruluk oranı 0.68 değerinde bir sonuca ulaşıldı. Test ve Eğitim setlerinin doğruluk oranları karşılaştırıldığı zaman modelin Overfit (Aşırı Öğrenme) olduğu anlaşılmakta. Modeli aşırı öğrenmeden kurtarmak için model karmaşıklığını azaltıp bazı parametrelerde değişiklik yapıp modelin yapısında oynamalar yapılabilir.

Modeli Aşırı Öğrenmeden Kurtarma

İlk amaç yeni bir model oluşturulup onu aşırı öğrenmeden kurtarmaktır. Modelin hangi işlemlerle aşırı öğrenmeden kurtulduğunu anlamak için oluşturulan model incelenebilir.

```
cnn3 = models.Sequential([

    #cnn adımları
    layers.Conv2D(filters=32,kernel_size = (3,3), activation = 'relu',input_shape = (32,32,3),padding='same'),
    BatchNormalization(),
    layers.Conv2D(filters=32,kernel_size = (3,3), activation = 'relu',input_shape = (32,32,3),padding='same'),
    BatchNormalization(),
    layers.MaxPooling2D((2,2)),
    layers.Dropout(0.25),

    layers.Conv2D(filters=64,kernel_size = (3,3), activation = 'relu',input_shape = (32,32,3),padding='same'),
    BatchNormalization(),
    layers.Conv2D(filters=64,kernel_size = (3,3), activation = 'relu',input_shape = (32,32,3),padding='same'),
    BatchNormalization(),
    layers.MaxPooling2D((2,2)),
    layers.Dropout(0.25),

    layers.Conv2D(filters=128,kernel_size = (3,3), activation = 'relu',input_shape = (32,32,3),padding='same'),
    BatchNormalization(),
    layers.Conv2D(filters=128,kernel_size = (3,3), activation = 'relu',input_shape = (32,32,3),padding='same'),
    BatchNormalization(),
    layers.MaxPooling2D((2,2)),
    layers.Dropout(0.25),
```

Şekil 18

```
#dense
layers.Flatten(),

layers.Dense(1024, activation='relu'),
layers.Dropout(0.25),
layers.Dense(10, activation='softmax')
```

Şekil 19

Bu modelde yapılan değişikliklerden başlıcaları şunlardır;

- Evrişim katmanının sayısı arttırıldı
- Evrişim katmanı parametrelerinde oynamalar yapıldı
- Normalizasyon işlemi uygulandı
- Rastgele nöronlar kapatılarak model karmaşıklığı azaltıldı

İlk fark evrişim katmanı sayısıdır. Önceki modelde 1 evrişim katmanı uygulayıp 1 pooling işlemi gerçekleştirilmişti. Bu modelde ise 2 evrişim katmanı uygulanıp ardınan 1 pooling işlemi gerçekleştirildi. Bu modelin resimlerin özelliklerini daha iyi şekilde algılayıp daha derin bir model oluşturulmasına olanak tanır. Ayrıca her evrişim katmanının ardından bir normalizasyon işlemi gerçekleşmektedir.

Batch Normalization (BN), derin öğrenme modellerinin eğitim sürecini iyileştirmek ve genel performansını artırmak için yaygın olarak kullanılan bir tekniktir. Bu işlemin temel amaçları şunlardır; Batch Normalization, her mini-batch'teki aktivasyonların ortalamasını ve varyansını hesaplayarak bu değerleri kullanarak aktivasyonları normalleştirir. Bu normalizasyon işlemi, gradyanların daha dengeli ve kararlı olmasını sağlar, bu da eğitim sürecinin hızlanmasına ve daha hızlı yakınsamasına yol açar. Sonuç olarak, model daha kısa sürede daha iyi performans gösterir ve aşırı öğrenmeyi azaltır.

İkinci değişiklik ise daha derin bir model oluşturmak için evrişim katmanındaki filtre sayıları 2 katına çıkarıldı ve padding parametresi eklendi.

Padding parametresi, giriş verisinin etrafına sıfır (veya başka bir sabit değer) ekleyerek veri boyutlarını kontrol etme işlemidir. Parametreye “same” değeri atandığında evrişim katmanının giriş ve çıkış değerleri sabit tutulur. Yani çıkış değeri giriş değerinden küçükse, çıkış değerinin etrafına 0 dizerek girişle aynı boyuta denk getirir. Bu işlem de kararlı ve daha derin bir model oluşturmamızda yardımcı olur.

Modeli aşırı öğrenmeden kurtaran en önemli etkenlerden bir tanesi de Dropout katmanını eklemektir.

Modele çok fazla parametre ve katman girildiğinde modelin karmaşıklığı artabilir ve bu da overfitting olasılığını arttıran bir özelliktir. Dropout, her eğitim adımında (epoch) modelin belirli nöronlarını rastgele seçerek devre dışı bırakır. Bu nöronlar, o adımda ileri besleme (forward pass) ve geri yayılım (backpropagation) hesaplamalarına katılmaz. Devre dışı bırakılan nöronlar

rastgele seçildiği için, model her eğitim adımında farklı bir yapı gibi davranır ve bu da modelin daha genelleştirilebilir ve esnek olmasını sağlar.

Bu model oluşturulurken belirli kısımlarında 0.25 oranında dropout kullanılmıştır.

Yeni modelin derlenme aşamasına geçinildiği zaman daha detaylı inceleme yapmak istediğimiz için izlemek istediğimiz metrikler değiştirildi.

```
METRICS = [
      'accuracy',
      tf.keras.metrics.Precision(name='precision'),
      tf.keras.metrics.Recall(name='recall')
]
cnn3.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=METRICS)
```

Şekil 20

Modelin özeti ise bu şekildedir

conv2d_3 (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
conv2d_4 (Conv2D)	(None, 32, 32, 32)	9,248
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout (Dropout)	(None, 16, 16, 32)	0
conv2d_5 (Conv2D)	(None, 16, 16, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 64)	256
conv2d_6 (Conv2D)	(None, 16, 16, 64)	36,928
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 64)	256
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
conv2d_7 (Conv2D)	(None, 8, 8, 128)	73,856
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 128)	512
conv2d_8 (Conv2D)	(None, 8, 8, 128)	147,584
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_5 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten_1 (Flatten)	(None, 2048)	0
dense_2 (Dense)	(None, 1024)	2,098,176
dropout_3 (Dropout)	(None, 1024)	0
dense_3 (Dense)	(None, 10)	10,250

Şekil 21

En sol satır katmanı, orta satır çıkış boyutunu ve en sol satırda parametre sayısını belirtmektedir

```
Total params: 2,397,226 (9.14 MB)

Trainable params: 2,396,330 (9.14 MB)

Non-trainable params: 896 (3.50 KB)
```

Şekil 22

Modelin eğitimi için 32 adımlı bir eğitim uygulanmakta ve validation verileri için eğitim verilerinin yanında test verileri de verilmekte.

```
r = cnn3.fit(X_train,y_cat_train,epochs=32,validation_data=(X_test,y_cat_test))
```

Şekil 23

Eğitim sonuçlarımız ise şu şekildedir

```
Epoch 21/32
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 56s 36ms/step - accuracy: 0.9399 - loss: 0.1737 - precision: 0.9492 - recall:
0.9311 - val_accuracy: 0.8470 - val_loss: 0.5666 - val_precision: 0.8661 - val_recall: 0.8337
Epoch 22/32
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 57s 36ms/step - accuracy: 0.9452 - loss: 0.1608 - precision: 0.9533 - recall:
0.9367 - val_accuracy: 0.8377 - val_loss: 0.6357 - val_precision: 0.8557 - val_recall: 0.8236
Epoch 23/32
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 58s 37ms/step - accuracy: 0.9474 - loss: 0.1541 - precision: 0.9556 - recall:
0.9401 - val_accuracy: 0.8381 - val_loss: 0.5989 - val_precision: 0.8557 - val_recall: 0.8285
Epoch 24/32
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 57s 37ms/step - accuracy: 0.9505 - loss: 0.1494 - precision: 0.9582 - recall:
0.9439 - val_accuracy: 0.8417 - val_loss: 0.5689 - val_precision: 0.8650 - val_recall: 0.8269
Epoch 25/32
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 57s 37ms/step - accuracy: 0.9497 - loss: 0.1441 - precision: 0.9573 - recall:
0.9435 - val_accuracy: 0.8383 - val_loss: 0.6026 - val_precision: 0.8594 - val_recall: 0.8265
Epoch 26/32
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 58s 37ms/step - accuracy: 0.9536 - loss: 0.1351 - precision: 0.9601 - recall:
0.9481 - val_accuracy: 0.8372 - val_loss: 0.6036 - val_precision: 0.8577 - val_recall: 0.8238
Epoch 27/32
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 59s 37ms/step - accuracy: 0.9536 - loss: 0.1358 - precision: 0.9597 - recall:
0.9482 - val_accuracy: 0.8490 - val_loss: 0.6139 - val_precision: 0.8643 - val_recall: 0.8372
Epoch 28/32
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 58s 37ms/step - accuracy: 0.9588 - loss: 0.1246 - precision: 0.9649 - recall:
0.9539 - val_accuracy: 0.8392 - val_loss: 0.6162 - val_precision: 0.8569 - val_recall: 0.8276
Epoch 29/32
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 58s 37ms/step - accuracy: 0.9591 - loss: 0.1201 - precision: 0.9648 - recall:
0.9546 - val_accuracy: 0.8452 - val_loss: 0.6063 - val_precision: 0.8626 - val_recall: 0.8359
Epoch 30/32
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 59s 38ms/step - accuracy: 0.9606 - loss: 0.1181 - precision: 0.9656 - recall:
0.9562 - val_accuracy: 0.8395 - val_loss: 0.6457 - val_precision: 0.8543 - val_recall: 0.8296
Epoch 31/32
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 58s 37ms/step - accuracy: 0.9619 - loss: 0.1113 - precision: 0.9666 - recall:
0.9575 - val_accuracy: 0.8559 - val_loss: 0.5873 - val_precision: 0.8704 - val_recall: 0.8456
Epoch 32/32
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 58s 37ms/step - accuracy: 0.9613 - loss: 0.1133 - precision: 0.9659 - recall:
0.9569 - val_accuracy: 0.8477 - val_loss: 0.6078 - val_precision: 0.8659 - val_recall: 0.8356
```

Şekil 24

32.epoch'un ardından eğitim veri seti için doğruluğumuz 0.96 çıkarken test veri seti için doğruluğumuz 0.84 oranına çıkmış oldu. Yapılan işlemler sonucu test doğruluk oranında %28'lik bir artış mevcut.

Sonuç

Modelin Performans Metrikleri

Modelin son durumdaki performans metrikleri şu şekildedir;

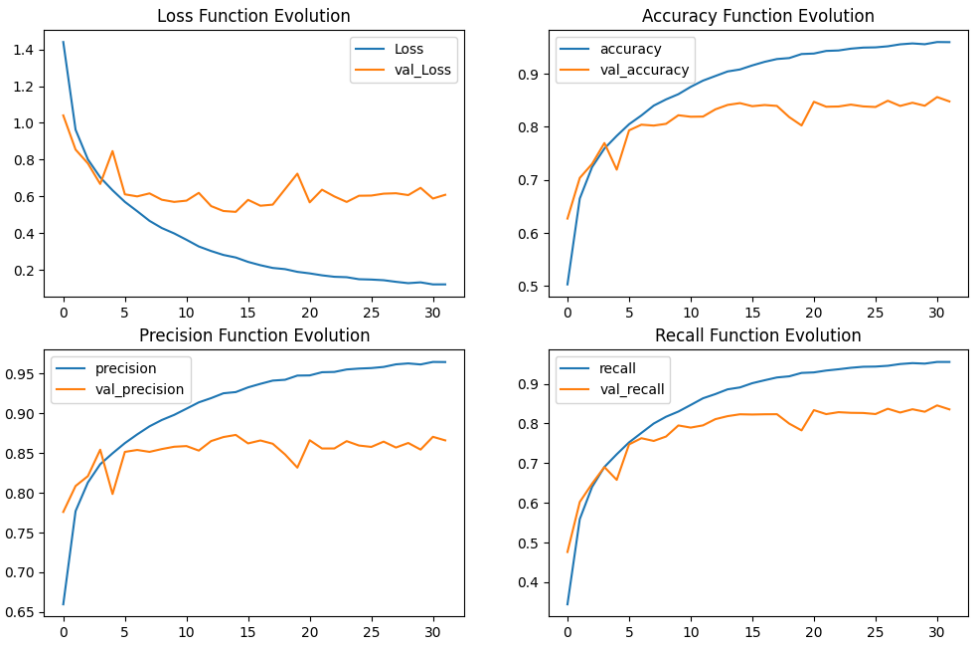
Eğitim Veri Seti İçin;

- Accuracy (Doğruluk): 0.9613
- Loss (Kayıp): 0.1133
- Precision (Hassasiyet): 0.9659
- Recall (Duyarlılık): 0.9569

Test Veri Seti İçin;

- Accuracy (Doğruluk): 0.8477
- Loss (Kayıp): 0.6078
- Precision (Hassasiyet): 0.8659
- Recall (Duyarlılık): 0.8356

Sonuçların Görselleştirilmesi

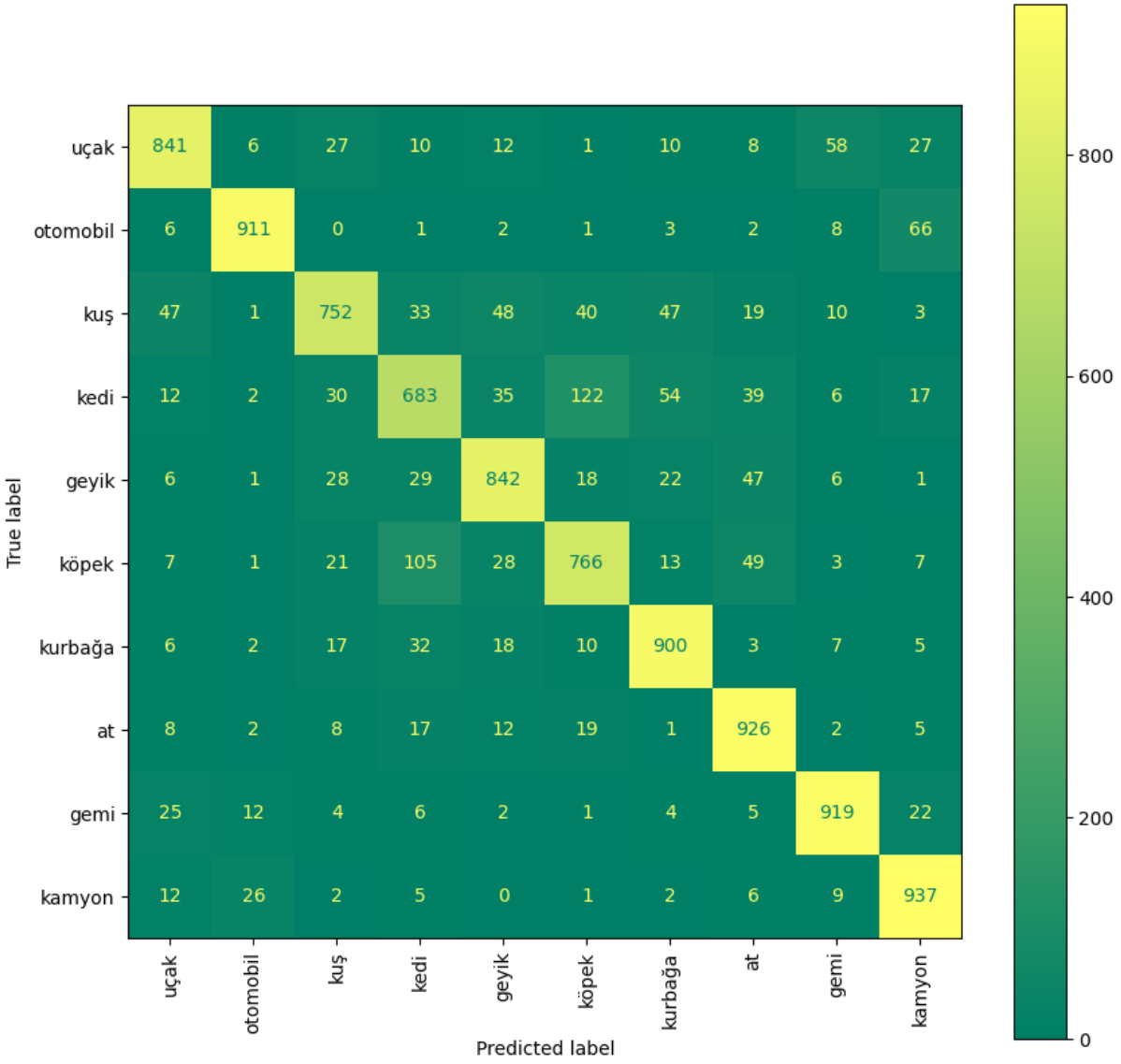


Şekil 25

Yukarıdaki şekilde Eğitim ve test setinin; Accuracy, Loss, Precision ve Recall metriklerinin karşılaştırmalı bir şekilde görselleştirildiğini görebilirsiniz.

Eğitim veri seti düzgün bir artış gösterirken bu durum test veri setinde biraz daha dalgalıdır.

Aşağıda tahmin edilen değerlerin ve aslında gerçek değerlerin karşılaştırmasını gösteren bir Confusion Matrix yer almakta



Şekil 26

Sonuçlar incelendiği zaman; kedi ve köpek, kamyon ve otomobil gibi benzer cisimleri ayırt ederken daha zorlanıldığı görülebilir.

Aşağıda ise etiketlere göre değerlendirilme metrikleri gösterilmektedir

```
['0 : uçak', '1 : otomobil', '2 : kuş', '3 : kedi', '4 : geyik', '5 : köpek', '6 : kurbağa',
'7 : at', '8 : gemi', '9 : kamyon']
              precision    recall  f1-score   support

           0       0.87      0.84      0.85      1000
           1       0.95      0.91      0.93      1000
           2       0.85      0.75      0.80      1000
           3       0.74      0.68      0.71      1000
           4       0.84      0.84      0.84      1000
           5       0.78      0.77      0.77      1000
           6       0.85      0.90      0.88      1000
           7       0.84      0.93      0.88      1000
           8       0.89      0.92      0.91      1000
           9       0.86      0.94      0.90      1000

    accuracy                           0.85     10000
   macro avg       0.85      0.85      0.85     10000
weighted avg       0.85      0.85      0.85     10000
```

Şekil 27

Değerlendirme ve Karşılaştırma

İnceleyeceğimiz Makale: An Analysis Of Convolutional Neural Networks For Image Classification - Neha Sharma,Vibhor Jain, Anju Mishra

Bu makalede önceden eğitilmiş 3 adet Cnn modeli kullanılmaktadır. Bu modeller; AlexNet, GoogleNet, ResNet50. Makalede yer alan doğruluk oranları sınıflara göre belirtilmiştir. Bu sebepten dolayı karşılaştırmalar genel yerine sınıflar özelinde olacaktır.

Makaledeki sınıflara göre doğruluk oranlarını inceleyelim

CIFAR-10		AlexNet	GoogLeNet	ResNet50
Image Category	Airplane	41.80%	51.10%	90.80%
	Automobile	21.80%	62.10%	69.10%
	Bird	00.02%	56.70%	72.60%
	Cat	00.03%	78.80%	61.90%
	Deer	87.60%	49.50%	75.40%
	Dog	23.00%	57.50%	82.10%
	Frog	24.20%	90.20%	76.60%
	Horse	34.70%	78.20%	84.70%
	Ship	31.70%	95.50%	83.20%
	Truck	95.90%	97.10%	84.60%

Göründüğü üzere her modelin her sınıf özelinde doğruluk oranı değişmektedir. Kimi model bazı sınıflarda daha iyi bir tahmin sonucuna giderken kimi model başka bir sınıfta daha iyi bir tahmin sonucu üretmektedir. Bu raporda üretilen model için sınıflar özelindeki doğruluk oranları aşağıdaki gibidir

SINIFLAR	BU RAPORDA HAZIRLANAN MODELİN DOĞRULUĞU
UÇAK	84.1%
OTOMOBİL	91.1%
KUŞ	75,2%
KEDİ	68,3%
GEYİK	84,2%

KÖPEK	76,6%
KURBAĞA	90%
AT	92,6%
GEMİ	91,9%
KAMYONET	93,7%

Sonuçları karşılaştıracak olursak şu tablo ortaya çıkmakta

Sınıflar	**En İyi Sonuç**
Uçak	ResNet50
Otomobil	Bu Raporda Hazırlanan Model
Kuş	Bu Raporda Hazırlanan Model
Kedi	GoogLeNet
Geyik	AlexNet
Köpek	ResNet50
Kurbağa	GoogLeNet
At	Bu Raporda Hazırlanan Model
Gemi	GoogLeNet
Kamyonet	GoogLeNet

Karşılaştırma sonuçları incelendiğinde Otomobil,Kuş ve At sınıflarında modelimiz makaledeki modellere göre daha olumlu sonuç göstermektedir. En çok doğru sonuç gösteren model ise GoogLeNet olmuştur.

Karşılaştırmanın sonucunda da oldukça başarılı bir model oluşturduğumuz söylenebilir.

KAYNAKÇA

I. **Makale**: Neha Sharma,Vibhor Jain, Anju Mishra (2018), An Analysis Of Convolutional Neural Networks For Image Classification https://www.sciencedirect.com/science/article/pii/S1877050918309335

II. Roblex Nana (2024) cifar10 with CNN for beginer, Kaggle https://www.kaggle.com/code/roblexnana/cifar10-with-cnn-for-beginer

III. Tuncer Ergin (2018) Convolutional Neural Network (ConvNet yada CNN) nedir, nasıl çalışır?, Medium https://medium.com/@tuncerergin/convolutional-neural-network-convnet-yada-cnn-nedir-nasil-calisir-97a0f5d34cad

IV. Fares Sayah (2023) Cifar-10 Images Classification using CNNs (88%), Kaggle https://www.kaggle.com/code/faressayah/cifar-10-images-classification-using-cnns-88