Cifar10 Görüntü Veri Seti Analizi

Nurseda DİKER

İçerik

- Proje amacı
- Veri Ön İşleme
- Model Oluşturma ve Eğitme
- Model Değerlendirme ve İyileştirme
- Sonuçlar

1. Proje Amacı

Bu proje, cifar-10 veri seti kullanılarak yapılan bir görüntü sınıflandırma çalışmasını içermektedir. Amaç, cifar-10 veri seti üzerinde çeşitli makine öğrenmesi ve derin öğrenme teknikleri kullanılarak veri setindeki görüntüleri doğru bir şekilde sınıflandırmaktır.

2. Veri Ön İşleme

CIFAR-10 veri seti, Keras kütüphanesinde hazır bir paket olarak bulunur ve kolayca yüklenebilir. Bu veri setini kullanarak modelimizi eğitmek için öncelikle veriyi yükledim ve ardından eğitim verisi ile etiketlerinin boyutlarını inceledim. CIFAR-10, her biri 32x32 piksel boyutunda ve RGB formatında olan 60.000 renkli görüntüden oluşur, bu görüntüler 10 sınıfa ayrılmıştır. Eğitim veri setinin boyutu 50.000, test veri setinin boyutu ise 10.000'dir. Eğitim veri boyutlarını ve etiket boyutlarını, ayrıca görüntü boyutları da inceledikten sonra son olarak, modelin daha iyi performans göstermesi için veri normalizasyonu işlemini gerçekleştirdim. Görüntü verilerinde her pikselin değeri 0 ile 255 arasında olduğu için, bu değerleri 0-1 aralığına dönüştürmek için her pikseli 255'e bölerek piksel değerlerini 0 ile 1 arasında ölçeklendirdim. Bu ön işlemler, modelin daha hızlı ve verimli bir şekilde öğrenmesini sağlar.

3. Model Oluşturma ve Eğitme

Modeli eğitmeye başlamadan önce veriyi düzleştirme işlemini yapmamız gerekir. Bu işlemi, görüntü verilerini modelimizin giriş formatına uygun hale getirmek için yapıyoruz. CIFAR-10 veri setindeki her bir görüntü 32x32 piksel boyutunda ve 3 renk kanalına (RGB) sahiptir. Bu nedenle, her bir görüntü aslında 32x32x3 boyutunda bir veri kümesidir.

Bazı makine öğrenmesi modelleri, özellikle de tam bağlantılı (fully connected) yapay sinir ağları, giriş verisinin tek boyutlu (flat) bir vektör halinde olmasını gerektirir. Bu nedenle, görüntü verilerini yeniden şekillendirmek (reshape) gerekir.

```
X_train = X_train.reshape(X_train.shape[0], -1)

X_test = X_test.reshape(X_test.shape[0], -1)
```

Bu kod satırları, her bir görüntüyü 32x32x3 boyutundan 3072 (32*32*3) elemanlı tek boyutlu bir vektöre dönüştürür. Bu dönüşüm, her bir görüntünün piksel değerlerini ardışık olarak bir vektör haline getirir. X_train.shape[0] ve X_test.shape[0] eğitim ve test veri setlerindeki görüntü sayısını belirtir, -1 ise bu sayıların korunarak kalan boyutların otomatik olarak hesaplanmasını sağlar.

3. Model Oluşturma ve Eğitme

Veri setini düzleştirip çeşitli makine öğrenmesi algoritmaları kullanarak modeller oluşturdum ve sonuçlarını değerlendirdim. KNN, Decision Tree, SVM, Logistic Regression ve Random Forest yöntemlerini denedim. SVM, diğer yöntemlere kıyasla en yüksek doğruluk (0.5436) ve F1 skoruna (0.5420) ulaşarak en iyi performansı gösterdi. KNN ve Logistic Regression orta düzeyde performans sergilerken, Decision Tree en düşük performansı (Accuracy: 0.2673) gösterdi. Random Forest ise karar ağaçlarından daha iyi performans sergilemesine rağmen SVM'nin altında kaldı.

Bu sonuçlar, karmaşık görüntü verileri üzerinde doğrusal ve karar ağaç tabanlı yöntemlerin sınırlı performansını göstermektedir. SVM, en iyi performansı sergileyen yöntem olsa da, CIFAR-10 gibi karmaşık veri setlerinde derin öğrenme tabanlı yaklaşımlar (örneğin, Convolutional Neural Networks - CNN'ler) genellikle çok daha yüksek doğruluk ve performans sağlar.

3. Model Oluşturma ve Eğitme

Model eğitme kısmında veri ön işleme aşamasının devamında, sınıf etiketlerini one hot encoding işlemi ile dönüştürdüm. One hot encoding, kategorik sınıfları ikili (binary) vektörlere dönüştürerek modelin sınıfları daha etkin bir şekilde öğrenmesini sağlar. Örneğin, CIFAR-10 veri setindeki 10 sınıf, her biri 10 uzunluğunda ve sadece bir elemanı '1', diğerleri '0' olan vektörler olarak temsil edilir.

One hot encoding işleminin ardından, derin öğrenme tabanlı bir Convolutional Neural Network (CNN) modeli oluşturdum. Modelde, dört tane convolutional katman ve bu katmanları takip eden pooling katmanları kullandım. Convolutional katmanlar, görüntüdeki özellikleri yakalamak için filtreler (kernels) uygular. Pooling katmanları ise, uzamsal boyutu azaltarak hesaplama yükünü ve overfitting riskini azaltır. Convolutional katmanların ardından, tam bağlantılı (fully connected) bir katman ekledim. Bu katman, convolutional katmanlardan gelen özellikleri sınıflandırma için kullanır. Modeli derledikten sonra, kayıp fonksiyonu olarak categorical crossentropy ve optimize edici olarak Adam optimizer kullandım. Son olarak, modeli `model.fit` ile eğitmeye başladım.

4. Model Değerlendirme ve İyileştirme

```
In [21]: # modeli eğitme
         history = model.fit(X_train, y_train, epochs=15, validation_data=(X_test, y_test))
```

```
Epoch 1/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 62s 36ms/step - accuracy: 0.3335 - loss: 1.7952 - val_accuracy: 0.5669 - val_loss: 1.2070
Epoch 2/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 57s 36ms/step - accuracy: 0.5896 - loss: 1.1696 - val_accuracy: 0.6667 - val_loss: 0.9504
Epoch 3/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 60s 39ms/step - accuracy: 0.6749 - loss: 0.9457 - val_accuracy: 0.6893 - val_loss: 0.8989
Epoch 4/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 60s 38ms/step - accuracy: 0.7251 - loss: 0.8041 - val_accuracy: 0.7218 - val_loss: 0.8068
Epoch 5/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 56s 36ms/step - accuracy: 0.7557 - loss: 0.7156 - val_accuracy: 0.7242 - val_loss: 0.8033
Epoch 6/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 57s 37ms/step - accuracy: 0.7865 - loss: 0.6170 - val_accuracy: 0.7199 - val_loss: 0.8612
Epoch 7/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 59s 38ms/step - accuracy: 0.8088 - loss: 0.5540 - val_accuracy: 0.7419 - val_loss: 0.7728
Epoch 8/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 58s 37ms/step - accuracy: 0.8282 - loss: 0.4908 - val_accuracy: 0.7327 - val_loss: 0.8246
Epoch 9/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 57s 37ms/step - accuracy: 0.8457 - loss: 0.4498 - val_accuracy: 0.7377 - val_loss: 0.8462
Epoch 10/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 58s 37ms/step - accuracy: 0.8644 - loss: 0.3958 - val_accuracy: 0.7443 - val_loss: 0.8557
Epoch 11/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 57s 36ms/step - accuracy: 0.8760 - loss: 0.3592 - val_accuracy: 0.7387 - val_loss: 0.9086
Epoch 12/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 57s 36ms/step - accuracy: 0.8910 - loss: 0.3198 - val_accuracy: 0.7426 - val_loss: 0.9733
Epoch 13/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 57s 37ms/step - accuracy: 0.8976 - loss: 0.2930 - val_accuracy: 0.7407 - val_loss: 1.0598
Epoch 14/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 58s 37ms/step - accuracy: 0.9071 - loss: 0.2632 - val_accuracy: 0.7344 - val_loss: 1.0195
Epoch 15/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 57s 36ms/step - accuracy: 0.9172 - loss: 0.2392 - val_accuracy: 0.7369 - val_loss: 1.1219
```

4. Model Değerlendirme ve İyileştirme

Bir önceki sayfadaki görsele baktığımızda accuracy değeri ne kadar yüksek olsa da loss, val_loss ve val_accuracy değerleri hep birlikte incelendiğinde aralarındaki fark overfitting olduğunu göstermektedir. Bu durum, modelin eğitim verisine çok iyi uyum sağlarken, test verisinde düşük performans göstermesi anlamına gelir. Overfitting'i önlemek ve modelin genelleme yeteneğini artırmak için Dropout, L1L2 Regularization ve Early Stopping yöntemlerini kullandım.

Dropout, eğitim sırasında belirli nöronları rastgele devre dışı bırakarak, modelin daha sağlam ve genelleyici olmasını sağlar. Bu yöntem, modelin belirli nöronlara veya yollarına bağımlılığını azaltarak overfitting'i engeller. **L1L2 Regularization** ise, modelin ağırlıklarını cezalandırarak daha küçük değerler almasını sağlar. L1 regularization, ağırlıkları sıfıra yaklaştırarak bazı ağırlıkların tamamen yok olmasına neden olurken, L2 regularization ağırlıkları küçültür fakat sıfırlamaz. Bu kombinasyon, modelin karmaşıklığını kontrol eder ve aşırı uyumu önler.

4. Model Değerlendirme ve İyileştirme

Early Stopping yöntemi ise, modelin doğrulama veri seti üzerindeki performansını izleyerek belirli bir epoch'tan sonra eğitim sürecini durdurur. Bu sayede, modelin doğrulama kaybı artmaya başladığında, eğitim durdurularak overfitting'in önüne geçilir.

Bu yöntemlerin kombinasyonu, modelin genelleme yeteneğini artırarak daha dengeli ve sağlam bir performans elde etmesini sağlar. Eğitim sırasında bu tekniklerin uygulanması, CIFAR-10 veri seti gibi karmaşık veri kümelerinde modelin doğruluğunu ve güvenilirliğini önemli ölçüde artırmaya yöneliktir.

5. Sonuçlar

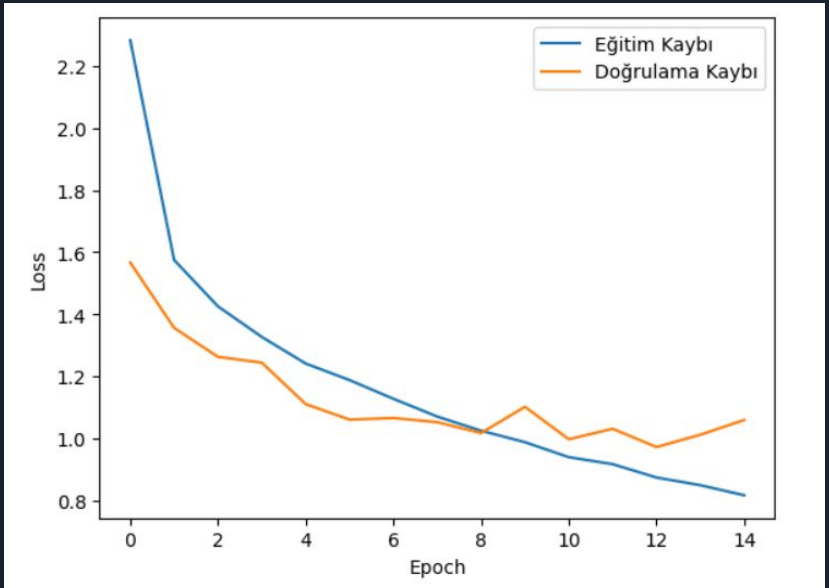
```
history = model.fit(X_train, y_train, epochs=15, validation_data=(X_test, y_test),callbacks=[early_stopping])
```

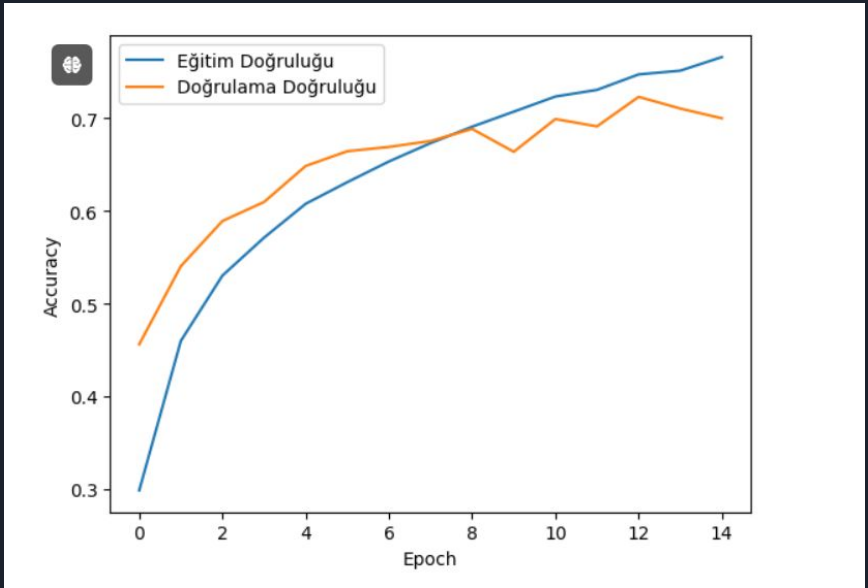
```
Epoch 1/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 70s 41ms/step - accuracy: 0.2174 - f1_score: 0.2079 - loss: 3.6393 - precision_1: 0.4289 - recal
l_1: 0.0243 - val_accuracy: 0.4562 - val_f1_score: 0.4320 - val_loss: 1.5666 - val_precision_1: 0.7075 - val_recall_1: 0.1737
Epoch 2/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 65s 42ms/step - accuracy: 0.4412 - f1_score: 0.4309 - loss: 1.6202 - precision_1: 0.6708 - recal
l_1: 0.1893 - val_accuracy: 0.5404 - val_f1_score: 0.5282 - val_loss: 1.3562 - val_precision_1: 0.7580 - val_recall_1: 0.3405
Epoch 3/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 63s 41ms/step - accuracy: 0.5140 - f1_score: 0.5083 - loss: 1.4559 - precision_1: 0.7288 - recal
l_1: 0.2890 - val_accuracy: 0.5891 - val_f1_score: 0.5770 - val_loss: 1.2626 - val_precision_1: 0.8032 - val_recall_1: 0.3648
Epoch 4/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 62s 40ms/step - accuracy: 0.5639 - f1_score: 0.5591 - loss: 1.3410 - precision_1: 0.7599 - recal
l_1: 0.3591 - val_accuracy: 0.6096 - val_f1_score: 0.6011 - val_loss: 1.2439 - val_precision_1: 0.7956 - val_recall_1: 0.4122
Epoch 5/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 63s 40ms/step - accuracy: 0.6018 - f1_score: 0.5971 - loss: 1.2537 - precision_1: 0.7808 - recal
l_1: 0.4094 - val_accuracy: 0.6484 - val_f1_score: 0.6426 - val_loss: 1.1103 - val_precision_1: 0.8120 - val_recall_1: 0.4781
Epoch 6/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 67s 43ms/step - accuracy: 0.6303 - f1_score: 0.6278 - loss: 1.1870 - precision_1: 0.7943 - recal
l_1: 0.4445 - val_accuracy: 0.6643 - val_f1_score: 0.6623 - val_loss: 1.0603 - val_precision_1: 0.7940 - val_recall_1: 0.5412
Epoch 7/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 77s 40ms/step - accuracy: 0.6472 - f1_score: 0.6474 - loss: 1.1456 - precision_1: 0.8041 - recal
l_1: 0.4767 - val_accuracy: 0.6689 - val_f1_score: 0.6667 - val_loss: 1.0654 - val_precision_1: 0.7878 - val_recall_1: 0.5490
Epoch 8/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 63s 40ms/step - accuracy: 0.6682 - f1_score: 0.6684 - loss: 1.0758 - precision_1: 0.8108 - recal
l_1: 0.5102 - val_accuracy: 0.6753 - val_f1_score: 0.6694 - val_loss: 1.0522 - val_precision_1: 0.8062 - val_recall_1: 0.5286
```

5. Sonuçlar

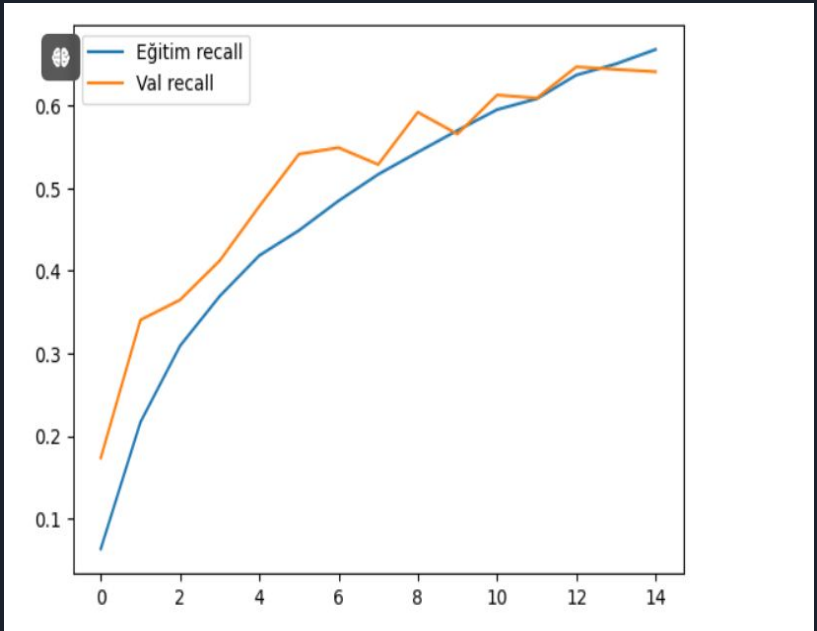
```
Epoch 9/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 63s 40ms/step - accuracy: 0.6933 - f1_score: 0.6928 - loss: 1.0171 - precision_1: 0.8287 - recal
l_1: 0.5462 - val_accuracy: 0.6883 - val_f1_score: 0.6864 - val_loss: 1.0170 - val_precision_1: 0.7934 - val_recall_1: 0.5919
Epoch 10/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 66s 42ms/step - accuracy: 0.7070 - f1_score: 0.7080 - loss: 0.9838 - precision_1: 0.8278 - recal
l_1: 0.5684 - val_accuracy: 0.6637 - val_f1_score: 0.6653 - val_loss: 1.1015 - val_precision_1: 0.7721 - val_recall_1: 0.5655
Epoch 11/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 64s 41ms/step - accuracy: 0.7221 - f1_score: 0.7237 - loss: 0.9443 - precision_1: 0.8366 - recal
l_1: 0.5890 - val_accuracy: 0.6990 - val_f1_score: 0.6987 - val_loss: 0.9970 - val_precision_1: 0.7880 - val_recall_1: 0.6127
Epoch 12/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 64s 41ms/step - accuracy: 0.7325 - f1_score: 0.7323 - loss: 0.9112 - precision_1: 0.8375 - recal
l_1: 0.6101 - val_accuracy: 0.6911 - val_f1_score: 0.6919 - val_loss: 1.0308 - val_precision_1: 0.7833 - val_recall_1: 0.6090
Epoch 13/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 62s 40ms/step - accuracy: 0.7493 - f1_score: 0.7502 - loss: 0.8604 - precision_1: 0.8477 - recal
l_1: 0.6402 - val_accuracy: 0.7229 - val_f1_score: 0.7228 - val_loss: 0.9718 - val_precision_1: 0.7985 - val_recall_1: 0.6468
Epoch 14/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 61s 39ms/step - accuracy: 0.7532 - f1_score: 0.7536 - loss: 0.8450 - precision_1: 0.8495 - recal
l_1: 0.6516 - val_accuracy: 0.7103 - val_f1_score: 0.7072 - val_loss: 1.0115 - val_precision_1: 0.7849 - val_recall_1: 0.6437
Epoch 15/15
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 59s 38ms/step - accuracy: 0.7697 - f1_score: 0.7707 - loss: 0.8044 - precision_1: 0.8580 - recal
l_1: 0.6706 - val_accuracy: 0.6998 - val_f1_score: 0.7011 - val_loss: 1.0591 - val_precision_1: 0.7745 - val_recall_1: 0.6409
```

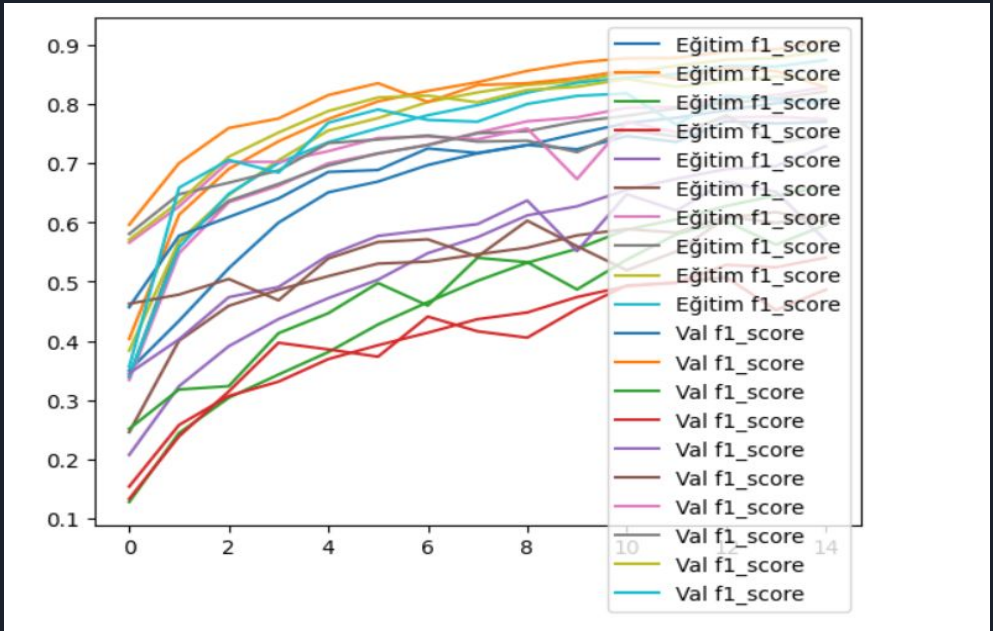
5. Sonuçlar


Eğitim Kaybı
Doğrulama Kaybı
Loss
Epoch
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0
2
4
6
8
10
12
14


Eğitim Doğruluğu
Doğrulama Doğruluğu
Accuracy
Epoch
0.7
0.6
0.5
0.4
0.3
0
2
4
6
8
10
12
14

5. Sonuçlar


Eğitim recall
Val recall
0.1
0.2
0.3
0.4
0.5
0.6
0
2
4
6
8
10
12
14


Eğitim f1_score
Eğitim f1_score
Eğitim f1_score
Eğitim f1_score
Eğitim f1_score
Eğitim f1_score
Eğitim f1_score
Eğitim f1_score
Eğitim f1_score
Eğitim f1_score
Val f1_score
Val f1_score
Val f1_score
Val f1_score
Val f1_score
Val f1_score
Val f1_score
Val f1_score
Val f1_score
Val f1_score
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
2
4
6
8
10
14

5. Sonuçlar

Model eğitiminin sonuçlarını gösteren bu görseller, eğitim sürecinin her bir epoch'u için eğitim doğruluğu, eğitim kaybı, doğrulama doğruluğu ve doğrulama kaybını sunmaktadır. Modelin eğitiminin başlangıcında, eğitim doğruluğu %21.74 ve eğitim kaybı 3.6393 olarak düşük bir performans sergiledi. Eğitimin ilerleyen aşamalarında, eğitim doğruluğu hızla artarak son epoch'ta %76.97'e ulaşırken, eğitim kaybı 0.8044'e düştü. Doğrulama doğruluğu da genellikle artarak son epoch'ta %69,98 e çıktı ve doğrulama kaybı 1.0591 ' e düştü, ancak bazı epoch'larda küçük dalgalanmalar görüldü. Bu sonuçlar, modelin hem eğitim hem de doğrulama verisi üzerinde performansının istikrarlı bir şekilde iyileştiğini göstermektedir.

Eğitim doğruluğu doğrulama doğruluğundan sürekli daha yüksek olsa da, aradaki fark çok büyük değil. Bu, modelde overfitting'in bir miktar kontrol altında tutulduğunu gösteriyor. Ayrıca, doğrulama kaybı başlangıçtan itibaren sürekli düşüyor, bu da modelin genelleme yeteneğinin iyi olduğunu gösteriyor.

5. Sonuçlar

Regularization (L1L2) ve Dropout gibi tekniklerin etkisi burada görülebilir. Bu yöntemler, modelin aşırı uyum yapmasını engelleyerek doğrulama setinde daha iyi performans göstermesini sağlamıştır. Early stopping kullanıldığında, model doğrulama doğruluğu belirli bir noktadan sonra iyileşmezse veya doğrulama kaybı artmaya başlarsa eğitim durdurulur. Bu örnekte, model 15 epoch boyunca eğitim gördü ve doğrulama doğruluğu ve kaybında istikrarlı bir gelişme gösterdiği için early stopping devreye girmedi.

Bu sonuçlar, modelin eğitim sürecinde hem eğitim verisi hem de doğrulama verisi üzerinde performansını sürekli iyileştirdiğini göstermektedir. Eğitim ve doğrulama doğruluğu arasındaki fark, kullanılan regularization ve dropout tekniklerinin overfitting'i önlemede etkili olduğunu işaret ediyor. Modelin doğrulama doğruluğu ve kaybı, eğitim doğruluğu ve kaybına paralel olarak iyileşmekte, bu da modelin genelleme yeteneğinin iyi olduğunu gösterir.