6.HAFTA

YAPAY SİNİR AĞLARI

DR. ÖĞR. ÜYESİ ALPER TALHA KARADENİZ

SAMSUN ÜNİVERSİTESİ

TensorFlow :

TensorFlow, bir dizi görev arasında veri akışı ve türevlenebilir programlama için kullanılan ücretsiz ve açık kaynaklı bir yazılım kütüphanesidir. Sembolik bir matematik kütüphanesidir ve sinir ağları gibi makine öğrenimi uygulamaları için de kullanılır. TensorFlow, Google Brain ekibi tarafından dahili Google kullanımı için geliştirilmiştir.

Makine öğrenmesi ve derin öğrenme modelleri kurmaya yarayan açık kaynaklı bir kütüphanedir. Büyük ölçekli verilerle çalışmaya uygundur.

- Hızlı, büyük verilerle çalışabilir, GPU/TPU desteği vardır.

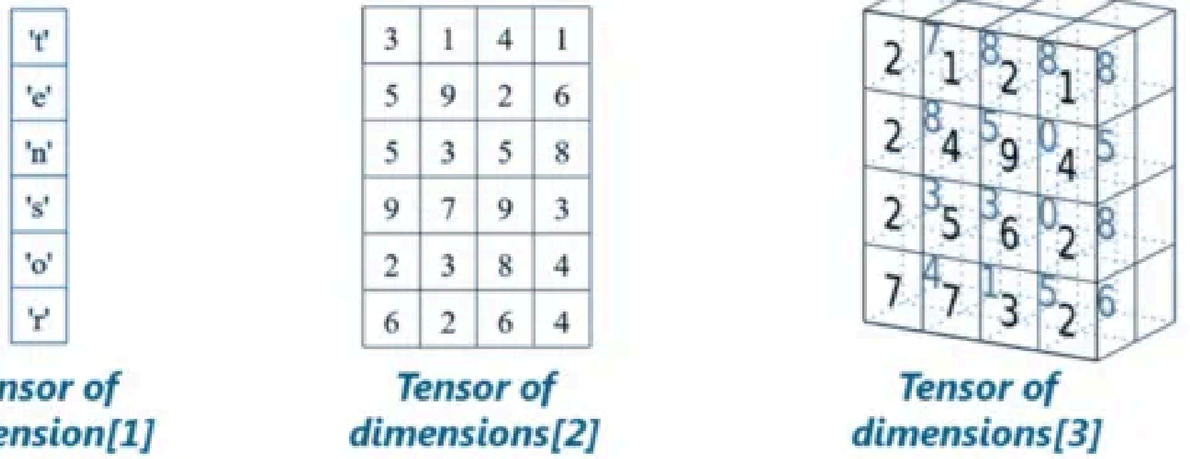
TensorFlow'un Temel Yapısı :

- **Tensör**: Çok boyutlu veri yapısıdır (vektör ve matrislerin genelleştirilmiş hali). TensorFlow ismini buradan alır.
- **Graph (hesap grafiği)**: İşlemler ve veriler düğümler ve kenarlar olarak gösterilir.
- **Eager Execution**: TensorFlow 2.0 ile birlikte, işlemler anında çalıştırılır (Python'daki normal hesaplamalar gibi).

Temel kullanım alanları :

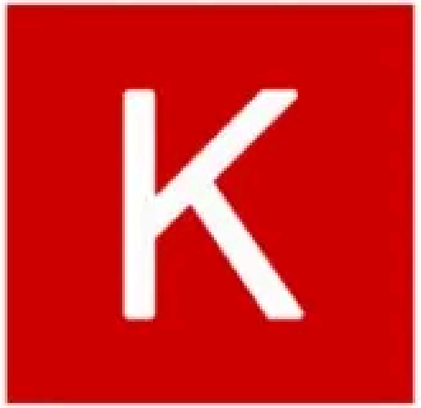
- Ses Tanıma/Algılama
- Duygu Analizi
- Kusur Tespiti
- Dil Algılama
- Metin Tabanlı Uygulamalar
- Metin Özetleme
- Görsel Tanıma
- Video Algılama
- Zaman Serileri

Tensörler yüksek boyutlu olan verileri temsil etmemize izin veren çok boyutlu dizilerdir. Aşağıda boyutların bir farklı gösterimi verilmiştir



Tensor of dimension[1] *Tensor of dimensions[2]* *Tensor of dimensions[3]*

TensorFlow; mobil uygulama, web uygulaması veya IoT cihazları üzerinde geliştirilecek projeler için uygun kütüphaneleri içerisinde barındırıyor.



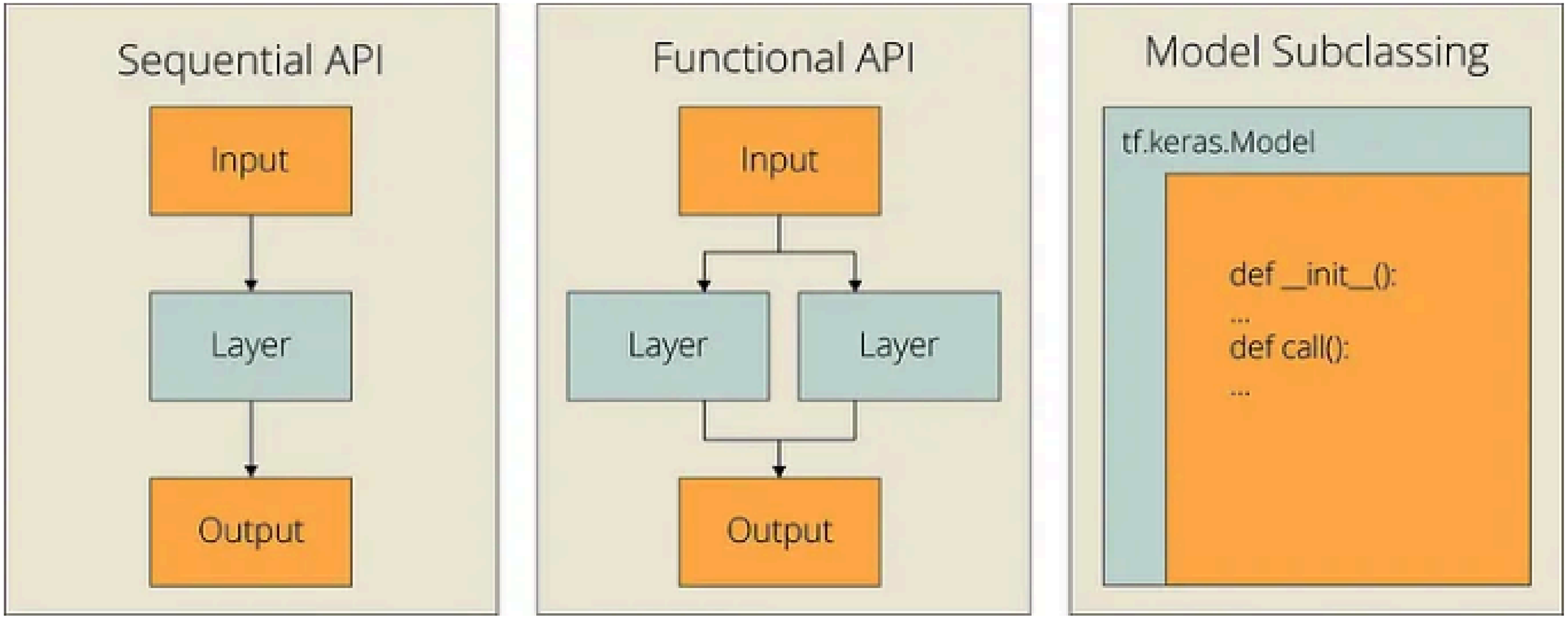
Keras

Simple. Flexible. Powerful.

Derin sinir ağları ile hızlı deney yapabilmek için tasarlanmıştır. Kolay ve hızlı bir şekilde model oluşturulmasına olanak sağlarken deneme yanılma yaparak modelde neleri değiştirebileceğimize karar vermemizi kolaylaştırır. Bilgisayarlı görme modelleri için evrişimli sinir ağlarını (CNN), sürekli veriler içinse yinelemeli sinir ağlarını (RNN) destekler

Keras'ta model tanımlamanın iki yolu bulunmaktadır. Sequential ve Functional API. ***Sequential***, en sık kullanılan olmakla beraber katmanları yığmak için kullanılır. ***Functional API*** ise çoklu çıktılı modeller gibi karmaşık model mimarileri tasarlamak için kullanılır. Sequential için aşağıdaki kodu kullanabiliriz.

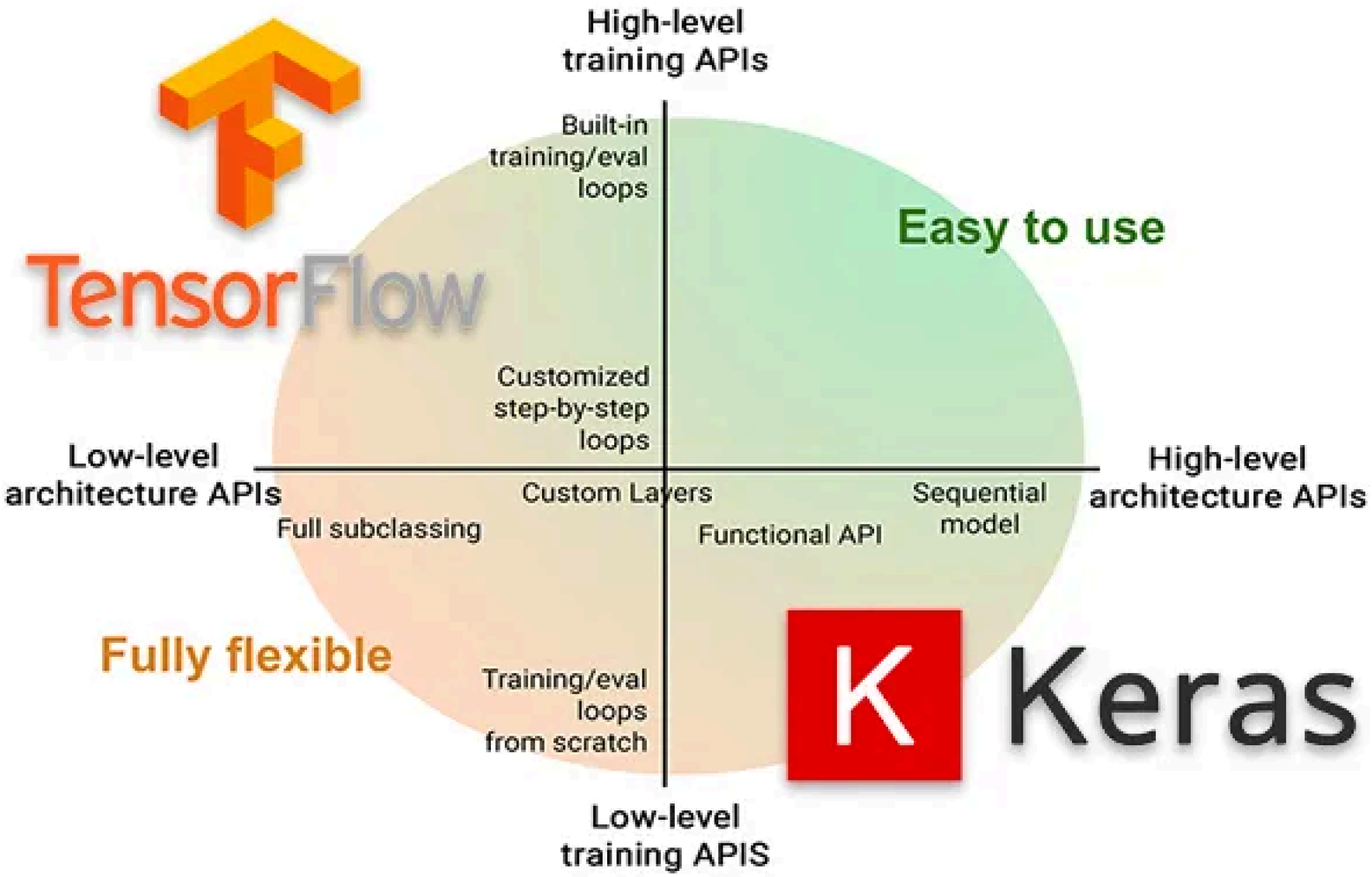
```
from keras.models import Sequential
```



Sequential API vs Functional API vs Subclassing API

Keras'ta modelleri oluştururken aşağıdaki sırayı izleriz:

1. Veriyi tanımlama
2. Modeli derleme
3. Modeli yürütme
4. Tahminler yapma


TensorFlow
High-level training APIs
Built-in training/eval loops
Easy to use
Customized step-by-step loops
Low-level architecture APIs
High-level architecture APIs
Custom Layers
Sequential model
Full subclassing
Functional API
Fully flexible
Training/eval loops from scratch
Keras
Low-level training APIS

Sequential API ile Model Kurma

1. Sequential API Nedir?

- **Sequential API**, Keras'ta en basit model kurma yöntemidir.
- "Sequential" kelimesi sıralı demektir. Bu API'de katmanlar **üst üste** eklenir.
- Yani **giriş katmanı → gizli katman(lar) → çıkış katmanı** sıralı bir yapı oluşturur.
- Her katman bir sonrakine bağlanır, geri dönüş veya dallanma olmaz.

2. Sequential Model Kurma Adımları

a) Modeli Tanımlama

python

```python
from tensorflow.keras.models import Sequential

model = Sequential()
```

b) Katman Ekleme

- Katmanlar sırayla eklenir.
- Örnek katman türleri:
  - **Dense**: Tam bağlı katman (en çok kullanılan).
  - **Conv2D**: Görüntü işleme için evrişimsel katman.
  - **LSTM/GRU**: Zaman serileri veya metin için tekrarlayan katman.

```
from tensorflow.keras.layers import Dense

model.add(Dense(64, activation="relu", input_shape=(100,)))
model.add(Dense(32, activation="relu"))
model.add(Dense(10, activation="softmax"))
```

Burada:

- **input_shape=(100,)** → giriş verisinin 100 özellik (feature) içerdiğini söylüyoruz.
- **64, 32, 10** → her katmandaki nöron sayısı.
- **relu, softmax** → aktivasyon fonksiyonları.

3. Modeli Derleme (Compile)

- Model çalışmadan önce "nasıl öğreneceğini" tanımlamalıyız.
- 3 önemli parametre vardır:
    1. **optimizer**: Ağırlıkların nasıl güncelleneceğini belirler (örnek: `adam`, `sgd`).
    2. **loss**: Hata fonksiyonu, modelin yanlışını ölçer (örnek: `categorical_crossentropy`).
    3. **metrics**: Performans ölçütü (örnek: `accuracy`).

```
model.compile(optimizer="adam",
              loss="categorical_crossentropy",
              metrics=["accuracy"])
```

4. Modeli Eğitme (Fit)

- Veriyi kullanarak modelimizi eğitiriz.
- Parametreler:
  - **x_train, y_train** → eğitim verisi.
  - **epochs** → tüm verinin modele kaç kez gösterileceği.
  - **batch_size** → verinin kaçlı gruplar halinde işleneceği.

```
history = model.fit(x_train, y_train, epochs=10, batch_size=32)
```

5. Modeli Değerlendirme

- Test verisiyle modelin başarısını ölçeriz.

```python

loss, acc = model.evaluate(x_test, y_test)
print("Test doğruluğu:", acc)
```

Eğitim – Doğrulama – Test Döngüsü

1. Veri Neden 3'e Bölünür?

- **Eğitim (Training set):**

  Modelin ağırlıkları bu veriden öğrenilir. Amaç: eğitim hatasını (loss) düşürmek.

- **Doğrulama (Validation set):**

  Eğitim sırasında her epoch sonunda bu veriyle modelin durumu ölçülür.

  → Hiperparametre seçimi, overfitting tespiti burada yapılır.

- **Test (Testing set):**

  Eğitim bittiğinde en son kullanılır.

  → Modelin **gerçek hayattaki performansını** ölçer.

Aşağıda en basit **ikili sınıflandırma** örneği var. Adım adım inceleyelim:

python

```python
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
```

- **numpy:** Rastgele veri oluşturmak için (sayısal işlemler).
- **tensorflow/keras:** Modeli kurmak için.
- **layers:** Sinir ağı katmanlarını tanımlamak için (`Dense`, `Dropout`, `Normalization` gibi).
- **train_test_split:** Veriyi eğitim/doğrulama/test setlerine bölmek için (scikit-learn fonksiyonu).

Veri Hazırlama

```python
# Sahte veri: 5000 örnek, 20 özellik
X = np.random.randn(5000, 20).astype("float32")

# Etiketler: basit kural -> ilk 3 özelliğe göre 0 veya 1
y = (X[:, 0] + 0.5*X[:, 1] - 0.2*X[:, 2] > 0).astype("int32")
```

- `X` : Giriş verisi (özellikler/features).
  - 5000 örnek, her biri 20 boyutlu.
  - `np.random.randn` : Normal dağılıma göre rastgele sayı üretir.
  - `.astype("float32")` : TensorFlow 32-bit float ister, uyumlu hale getiriyoruz.
- `y` : Çıkış etiketleri (sınıf = 0 veya 1).
  - `X[:, 0]` : ilk özellik.
  - `X[:, 1]` : ikinci özellik.
  - Küçük bir matematiksel kombinasyonla sınıf belirliyoruz.

Veriyi Bölme

python

```python
# %60 eğitim, %20 doğrulama, %20 test
X_train, X_tmp, y_train, y_tmp = train_test_split(
    X, y, test_size=0.4, stratify=y, random_state=42
)
X_val, X_test, y_val, y_test = train_test_split(
    X_tmp, y_tmp, test_size=0.5, stratify=y_tmp, random_state=42
)
```

- **train_test_split**: Veriyi böler.
- `test_size=0.4` : Verinin %40'ı train dışına çıkarılıyor (yani %60 train kalıyor).
- `stratify=y` : Sınıf oranlarını korur (ör. %50–%50).
- `random_state=42` : Rastgelelik için sabit seed → Tekrarlanabilir sonuç.
- İkinci bölmede: kalan %40'ın yarısı val, yarısı test → %20 val, %20 test.

Normalizasyon

```python
norm = layers.Normalization()
norm.adapt(X_train)  # SADECE train verisine adapte edilir
```

- **Normalization layer:** Özellikleri belli bir ölçeğe çeker.
- `adapt` : Eğitim verisinin ortalaması ve standart sapması hesaplanır.
- Sonra val ve test aynı değerlerle normalize edilir (leakage engellenir).

Model Kurma

python

```python
def build_model():
    inputs = keras.Input(shape=(X.shape[1],))  # 20 özelliklik giriş
    x = norm(inputs)                           # Normalizasyon
    x = layers.Dense(64, activation="relu")(x) # 1. gizli katman
    x = layers.Dropout(0.3)(x)                 # Overfitting azaltmak için
    x = layers.Dense(32, activation="relu")(x) # 2. gizli katman
    outputs = layers.Dense(1, activation="sigmoid")(x) # Çıkış katmanı
    model = keras.Model(inputs, outputs)

    model.compile(
        optimizer=keras.optimizers.Adam(1e-3),
        loss="binary_crossentropy",
        metrics=["accuracy"]
    )
    return model
```

- **keras.Input:** Giriş boyutunu tanımlar (20 özellik).
- **Dense(64, relu):** 64 nöron, ReLU aktivasyonlu tam bağlı katman.
- **Dropout(0.3):** Rastgele %30 nöronu devre dışı bırakır → ezberlemeyi azaltır.
- **Dense(1, sigmoid):** Çıkış katmanı → 0–1 arası olasılık döndürür (ikili sınıflandırma).
- **compile:**
  - `optimizer="adam"` : Öğrenme algoritması (gradyan inişi).
  - `loss="binary_crossentropy"` : İkili sınıflandırma için uygun kayıp fonksiyonu.
  - `metrics=["accuracy"]` : Başarı ölçütü.

Callback'ler

python

```python
early = keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=5, restore_best_weights=True
)
plateau = keras.callbacks.ReduceLROnPlateau(
    monitor="val_loss", factor=0.5, patience=2
)
ckpt = keras.callbacks.ModelCheckpoint(
    filepath="best_model.keras", monitor="val_loss", save_best_only=True
)
```

- **EarlyStopping:**
  - `monitor="val_loss"` → doğrulama kaybını izler.
  - `patience=5` → 5 epoch boyunca iyileşme olmazsa durdurur.
  - `restore_best_weights=True` → en iyi epoch'taki ağırlıkları geri yükler.
- **ReduceLROnPlateau:**
  - `factor=0.5` → öğrenme oranını yarıya düşürür.
  - `patience=2` → 2 epoch iyileşme yoksa çalışır.
- **ModelCheckpoint:**
  - En iyi modeli (en düşük `val_loss`) diske kaydeder.

Modeli Eğitme

```python
history = model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=64,
    validation_data=(X_val, y_val),
    callbacks=[early, plateau, ckpt],
    verbose=0
)
```

- **epochs=100**: Model 100 kez tüm eğitim setini görebilir (ama early stopping durdurabilir).
- **batch_size=64**: Eğitim verisi 64'lük gruplara bölünür.
- **validation_data**: Doğrulama seti. Her epoch sonunda loss/accuracy burada hesaplanır.
- **callbacks**: Eğitim sırasında otomatik kontroller yapılır.
- **verbose=0**: Çıktıyı bastırmaz (sessiz mod).

Test

python

```
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
```

- Eğitim bittikten sonra **tek seferlik** çalıştırılır.
- `evaluate` : Loss ve accuracy değerini hesaplar.

Tahmin ve Rapor

```python
y_prob = model.predict(X_test, verbose=0).ravel()
y_pred = (y_prob >= 0.5).astype("int32")
```

- **predict:** Olasılık çıktıları üretir.
- `.ravel()` : Çıktıyı tek boyutlu hale getirir.
- `>= 0.5` : Olasılık 0.5'ten büyükse sınıf = 1, küçükse sınıf = 0.

```python
from sklearn.metrics import classification_report, confusion_matrix

print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

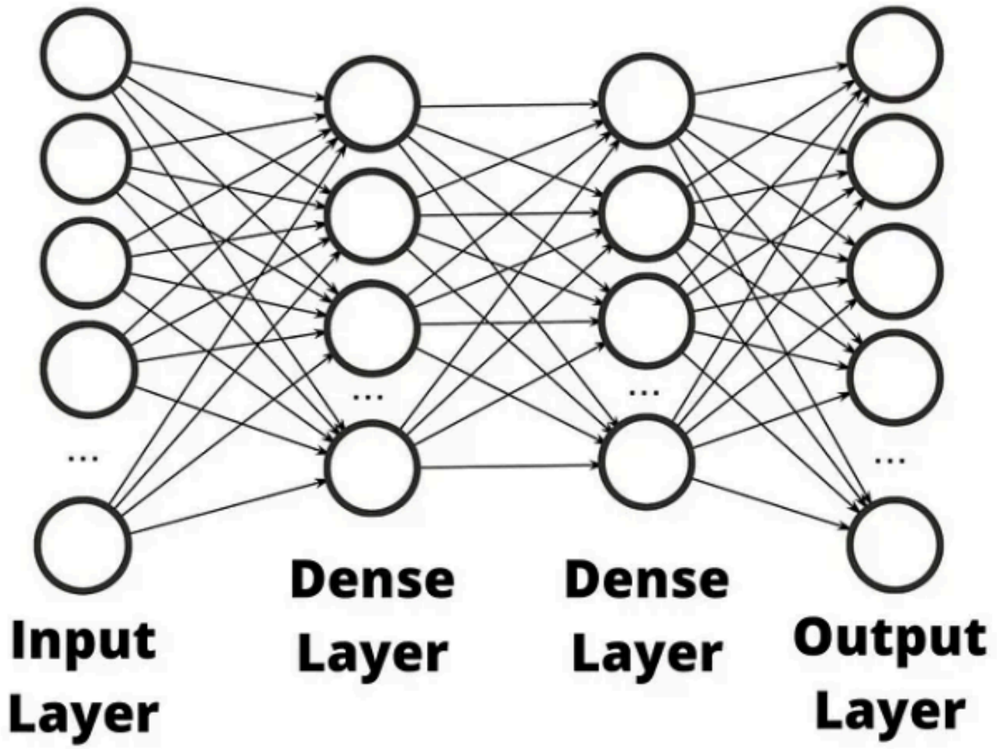
- **classification_report:** Precision, recall, F1 gibi detaylı metrikler.
- **confusion_matrix:** Doğru ve yanlış sınıflandırmaların tablosu.

Dense (tam bağlı) katmanlı sinir ağı:

- Bu tür bir katmanda, her nöron bir önceki katmandaki tüm nöronlara bağlantılıdır.
- Her nöronun çıktısı, giriş değerlerinin ağırlıklarla çarpımı ile bias teriminin toplamına bir aktivasyon fonksiyonu uygulanarak elde edilir:

$$a_j = f\left(\sum_i w_{ij} x_i + b_j\right)$$

- Dense katmanlar, verinin tüm bileşenlerini dikkate alarak bilgiyi yoğun bir şekilde işler.
- Dezavantajı, giriş boyutu büyük olduğunda ağırlık sayısının çok artması ve bunun hesaplama maliyetini yükseltmesidir.
- Genellikle veri işleme ve sınıflandırma görevlerinde, özellikle ağın sonunda sınıflandırma katmanı olarak kullanılır.



MNIST El Yazısı Rakam Tanıma: Tam Bağlantılı (Dense) Model Uygulaması

Adım 1: Gerekli Kütüphaneleri Yükleme

İlk olarak, modelimizi oluşturmak ve veri üzerinde işlem yapmak için ihtiyacımız olan Python kütüphanelerini içe aktarırız. Bu kütüphaneler arasında derin öğrenme için temel olan **TensorFlow** ve **Keras** bulunur.

Python

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
```

Adım 2: Veri Kümesini Yükleme ve Hazırlama

Bir yapay sinir ağının doğru sonuçlar üretebilmesi için verinin önceden işlenmesi gerekir. Bu aşamada, veri kümesini yükler ve modelin kolayca işleyebileceği bir formata dönüştürürüz.

1. **Veri Yükleme:** MNIST veri kümesi, Keras kütüphanesi ile eğitim ve test setleri olarak otomatik bir şekilde yüklenir.
2. **Normalizasyon:** Görüntü piksellerinin değerleri (0-255 aralığında), **0 ile 1 arasına ölçeklenerek** modelin eğitiminin daha hızlı ve kararlı olması sağlanır.
3. **Düzleştirme (Flattening): Dense** (tam bağlantılı) katmanlar 2 boyutlu görüntüleri doğrudan işleyemez. Bu nedenle, 28x28 piksellik her bir görüntüyü 784 elemanlı tek bir vektöre dönüştürürüz.

```
# Veri kümesini yükle
(train_images, train_labels), (test_images, test_labels) = keras.datasets.mnist.load_data()

# Normalizasyon
train_images = train_images / 255.0
test_images = test_images / 255.0

print("Eğitim verisi boyutu:", train_images.shape)
print("Test verisi boyutu:", test_images.shape)
```

Adım 3: Model Mimarisi Oluşturma

Bu aşamada, modelimizin mimarisini, yani hangi katmanlardan oluşacağını tanımlarız.

- `Flatten` **Katmanı:** Görüntüleri tek bir vektöre dönüştürür.
- **Gizli Katman (Dense):** 128 nöronlu bu katman, görüntüdeki özelliklerin öğrenildiği yerdir. **ReLU** (Rectified Linear Unit) aktivasyon fonksiyonu kullanır.
- **Çıkış Katmanı (Dense):** 10 nöron (her rakam için bir tane) içerir. **Softmax** aktivasyon fonksiyonu, her bir nöron için bir olasılık değeri üreterek modelin nihai tahminini belirler.

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

model.summary()
```

Adım 4: Modeli Derleme (Compilation)

Modeli eğitmeye başlamadan önce, hangi optimizasyon algoritmasını, hangi hata fonksiyonunu ve hangi metrikleri kullanacağını belirtmeliyiz.

- `optimizer='adam'`: En etkili optimizasyon algoritmalarından biridir.
- `loss='sparse_categorical_crossentropy'`: Çok sınıflı sınıflandırma problemleri için yaygın olarak kullanılan hata fonksiyonudur.
- `metrics=['accuracy']`: Modelin performansını doğruluk oranıyla ölçer.

Python

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

Adım 5: Modeli Eğitme

Model, eğitim verisi (`train_images`) ve etiketleri (`train_labels`) kullanılarak eğitilir. Bu aşamada, modelin binlerce örnekten öğrenmesi sağlanır.

- `epochs=10` : Modelin tüm eğitim verisini 10 kez göreceğini belirtir.
- `validation_split=0.1` : Eğitim verisinin %10'unu ayırarak, modelin eğitim sırasında görmediği verilere karşı performansını takip etmemizi sağlar.

Python

```
model.fit(train_images, train_labels, epochs=10, validation_split=0.1)
```

Adım 6: Modeli Değerlendirme

Eğitilen modelin gerçek performansı, eğitimde hiç kullanılmamış olan test verisi üzerinde değerlendirilir.

Python

```
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f"\nTest doğruluğu: {test_acc:.4f}")
```

1. Veri Hazırlığı Çıktısı:

```
Eğitim verisi boyutu: (60000, 28, 28)
Test verisi boyutu: (10000, 28, 28)
```

2. Model Özeti Çıktısı:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0
 dense (Dense)               (None, 128)               100480
 dense_1 (Dense)             (None, 10)                1290
=================================================================
Total params: 101770
Trainable params: 101770
Non-trainable params: 0
_________________________________________________________________
```

3. Eğitim Süreci Çıktıları:

```
Epoch 1/10
1688/1688 [==============================] - 5s 3ms/step - loss: 0.2828 - accuracy: 0.926
Epoch 2/10
1688/1688 [==============================] - 4s 2ms/step - loss: 0.1287 - accuracy: 0.962
...
Epoch 10/10
1688/1688 [==============================] - 5s 3ms/step - loss: 0.0245 - accuracy: 0.992
```

4. Model Değerlendirme Çıktısı:

```
313/313 - 1s - loss: 0.0792 - accuracy: 0.9779
Test doğruluğu: 0.9779
```