PROJENİN AMACI

Projenin amacı, görsel veri setini kullanarak savaş ve normal fotoğrafları sınıflandırmak için bir derin öğrenme modeli oluşturmaktır. Model, kullanıcıların bu tür fotoğrafları otomatik olarak kategorize etmesine yardımcı olabilir ve bu, güvenlik, analiz veya diğer amaçlar için kullanılabilir. Modeli oluştururken Keras ve TensorFlow gibi popüler derin öğrenme kütüphanelerini kullanarak, güçlü ve etkili bir sınıflandırıcı geliştirmeyi hedefledim. Bu sınıflandırma işlemi, görüntü işleme ve derin öğrenme tekniklerini kullanarak yapılıyor.

Kullanılan Materyal ve Yöntemlerin Özellikleri

1. TensorFlow ve Keras

- **TensorFlow:** Google tarafından geliştirilen açık kaynaklı bir derin öğrenme kütüphanesi. Derin öğrenme modelleri oluşturmak, eğitmek ve dağıtmak için kullanılır.
- **Keras:** TensorFlow'un yüksek seviyeli bir API'sidir. Kullanıcıların derin öğrenme modellerini kolay ve hızlı bir şekilde oluşturmasını sağlar.

2. Veri Yükleme ve Ön İşleme

- **ImageDataGenerator:** Görselleri yüklemek ve veri artırma (data augmentation) işlemleri yapmak için kullanılır. Veri artırma, modelin genelleme yeteneğini artırmak için görseller üzerinde çeşitli dönüşümler uygular.

```
# TensorFlow Dataset oluşturucuları kullanarak veri yüklemesi
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

- **rescale:** Görsellerin piksel değerlerini 0-255 aralığından 0-1 aralığına normalize eder.
- **rotation_range, width_shift_range, height_shift_range, shear_range, zoom_range, horizontal_flip, fill_mode:** Görselleri çeşitli şekillerde dönüştürerek veri setini artırır ve modelin genelleme yeteneğini artırır.

3. Model Mimarisi

- **MobileNetV2:** Google tarafından geliştirilen, hafif ve verimli bir derin öğrenme modelidir. Görüntü sınıflandırma için yaygın olarak kullanılır.
- **GlobalAveragePooling2D:** Özellik haritalarını global ortalama havuzlama işlemiyle küçülterek, modelin fazla parametre kullanmasını önler.
- **Dense Layer:** Çıkış katmanı, sigmoid aktivasyon fonksiyonu ile ikili sınıflandırma yapar.

```
# Modeli oluşturma
IMG_SHAPE = (160, 160, 3)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                                include_top=False,
                                                weights='imagenet')
base_model.trainable = False  # Özellik çıkarımı için baz modelin katmanlarını
dondur


global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
prediction_layer = tf.keras.layers.Dense(1, activation='sigmoid')


model = tf.keras.Sequential([
  base_model,
  global_average_layer,
  prediction_layer
])
```

4. Modelin Eğitimi

- **optimizer:** Modelin parametrelerini güncellemek için RMSprop optimizasyon algoritması kullanılır.
- **loss:** Kayıp fonksiyonu olarak binary crossentropy kullanılır, bu ikili sınıflandırma problemleri için yaygın bir seçimdir.
- **metrics:** Modelin performansını değerlendirmek için doğruluk metriği kullanılır.

```
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=base_learning_rate),
              loss=tf.keras.losses.mean_squared_error(from_logits=True),
              metrics=['accuracy'])
```

5. Modelin Eğitilmesi ve Değerlendirilmesi

- **fit:** Modeli, eğitim veri seti üzerinde eğitir.
- **evaluate:** Modeli, test veri seti üzerinde değerlendirir ve doğruluk ve kayıp değerlerini raporlar.

```
# Model eğitimi
initial_epochs = 100
history = model.fit(train_generator,
                    epochs=initial_epochs,
                    validation_data=validation_generator)
test_loss, test_accuracy = model.evaluate(test_generator)
```

6. Eğitim ve Doğrulama Grafiklerinin Görselleştirilmesi

- **Matplotlib:** Eğitim ve doğrulama sürecindeki doğruluk ve kayıp değerlerini görselleştirmek için kullanılır.

```
# Eğitim ve doğrulama grafikleri
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Loss')
plt.ylim([0, 1])
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.show()
```

Veri Seti Bilgileri

Veri setini kendim toplayıp düzenledim ve içeriği şöyle;

Toplam görsel sayım 4000 ve bunların 2000 i normal 2000 i de rahatsız edici içerikten oluşuyor cinsellik içeren görselleri hariç tutarsak +18 fotoğraflar içermektedir. Bunları test ,train ve validation şeklinde 3 sınıfa ayırdım ve bunların oranları şu şekildedir.

- **Eğitim Seti (Training Set):** %70
- **Doğrulama Seti (Validation Set):** %15
- **Test Seti (Test Set):** %15

Bu oranlar, modelin performansını değerlendirmek ve genelleme yeteneğini artırmak için yaygın olarak kullanılıan oranlardır.

Veri Setinin Bölünmesi İçin Kodu

Aşağıdaki kod, veri setinizin doğru oranlarda nasıl ayrıldığını gösterir. Bu kod, Keras'ın ImageDataGenerator ve flow_from_directory fonksiyonlarını kullanarak veri setini yükler ve işler.

```
import tensorflow as tf

# Veri yolu tanımlamaları
train_dir = 'train_path'
validation_dir = 'valid_path'
test_dir = 'test_path'

# TensorFlow Dataset oluşturucuları kullanarak veri yüklemesi
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(160, 160),
```

```
    batch_size=32,
    class_mode='binary'
)

validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=(160, 160),
    batch_size=32,
    class_mode='binary'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(160, 160),
    batch_size=32,
    class_mode='binary'
)
```

Kullanılan Model ve Mimariler

1. MobileNetV2

- **MobilNetV2:** Google tarafından geliştirilen ve özellikle mobil ve gömülü cihazlar için optimize edilmiş hafif bir derin öğrenme mimarisidir. Görüntü sınıflandırma, nesne algılama ve diğer bilgisayarla görme görevleri için yaygın olarak kullanılır.

Avantajları:

- Hafif ve hızlıdır.
- Daha az hesaplama gücü ve bellek kullanımı gerektirir.
- Taşınabilir cihazlar ve gerçek zamanlı uygulamalar için uygundur.

Kullanımı:

- **include_top=False:** Modelin üst katmanlarını hariç tutarak, sadece özellik çıkarım katmanlarını kullanıyoruz.
- **weights='imagenet':** ImageNet veri seti üzerinde önceden eğitilmiş ağırlıkları kullanıyoruz, böylece modelin özellik çıkarım yeteneklerinden faydalanıyoruz.

```
IMG_SHAPE = (160, 160, 3)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                                include_top=False,
                                                weights='imagenet')
```

```
base_model.trainable = False  # Özellik çıkarımı için baz modelin katmanlarını dondur
```

2. Modelin Üstüne Eklenen Katmanlar

- **GlobalAveragePooling2D:** Bu katman, uzaysal boyutları ortalama alarak sıkıştırır ve tek bir vektör elde eder. Bu, özellik haritalarının boyutunu azaltır ve fazla parametre kullanmasını önler.
- **Dense Layer (Tam Bağlantılı Katman):** Tek bir nörona sahip olan bu katman, sigmoid aktivasyon fonksiyonunu kullanarak ikili sınıflandırma yapar. Çıktısı 0 ile 1 arasında bir olasılık değeri verir.

    - **Sigmoid Aktivasyon Fonksiyonu:** İkili sınıflandırma problemleri için uygundur. Çıktıyı 0 ve 1 arasında sıkıştırır, bu da sınıflandırma için gereken olasılık değerini sağlar.

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
prediction_layer = tf.keras.layers.Dense(1, activation='sigmoid')
model = tf.keras.Sequential([
  base_model,
  global_average_layer,
  prediction_layer
])
```

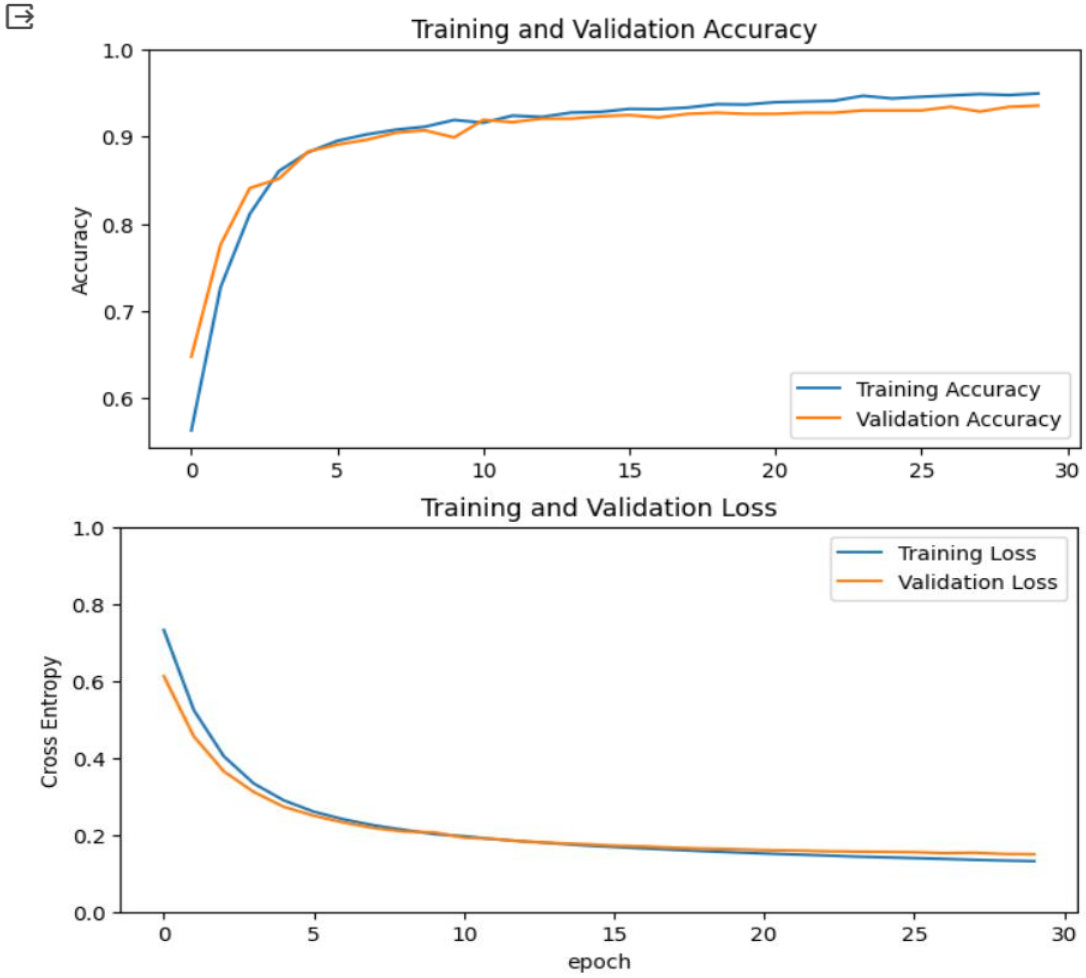
3. Modelin Derlenmesi

- **Optimizer (RMSprop):** RMSprop optimizasyon algoritması, özellikle derin öğrenme modellerinin eğitimi sırasında öğrenme hızını kontrol etmek ve kaybı azaltmak için kullanılır.
    - **Learning Rate (Öğrenme Oranı):** 0.0001 olarak belirlenmiş. Bu, modelin yavaş ama daha kararlı bir şekilde öğrenmesini sağlar.
- **Loss Function (Binary Crossentropy):** Binary crossentropy kayıp fonksiyonu, ikili sınıflandırma problemleri için kullanılır. Modelin doğru tahminler yapmasını sağlamak için kullanılır.
- **Metrics (Accuracy):** Modelin performansını izlemek için doğruluk metriği kullanılır. Bu, doğru sınıflandırılan örneklerin toplam örnek sayısına oranını verir.

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=base_learning_rate),
              loss=tf.keras.losses.mean_squared_error(from_logits=True),
              metrics=['accuracy'])
```

Bu modelleme sürecinde, önceden eğitilmiş MobileNetV2 modelinin tabanını kullanarak özellik çıkarımı yapıyoruz. Üzerine eklediğimiz global ortalama havuzlama katmanı ve tam bağlantılı katman ile ikili sınıflandırma görevini gerçekleştiriyoruz. Bu yapı, modeli hızlı ve verimli hale getirirken, sınıflandırma performansını da artırmayı hedefler. Kullanılan teknikler ve yöntemler, modelin genelleme yeteneğini artırarak farklı veri setleri üzerinde de iyi performans göstermesini sağlar.

Deney Sonuçları



Model sonuçlarım şu şekilde ; (50 Epoch ile)

Epoch 30/50

82/82 [==============================] - 121s 1s/step - loss: 0.0220 - accuracy: 0.9958 - val_loss: 0.1267 - val_accuracy: 0.9487

Epoch 31/50

82/82 [==============================] - 119s 1s/step - loss: 0.0239 - accuracy: 0.9942 - val_loss: 0.1322 - val_accuracy: 0.9447

Epoch 32/50

82/82 [==============================] - 118s 1s/step - loss: 0.0195 - accuracy: 0.9973 - val_loss: 0.1235 - val_accuracy: 0.9487

……

Epoch 48/50

82/82 [==============================] - 114s 1s/step - loss: 0.0043 - accuracy: 0.9996 - val_loss: 0.1208 - val_accuracy: 0.9622

Epoch 49/50

82/82 [==============================] - 117s 1s/step - loss: 0.0034 - accuracy: 1.0000 - val_loss: 0.1244 - val_accuracy: 0.9582

Epoch 50/50

82/82 [==============================] - 113s 1s/step - loss: 0.0020 - accuracy: 0.9996 - val_loss: 0.1266 - val_accuracy: 0.9582

Confusion Matrix

Modelimin confusion matrixini oluşturmak için Python'da sklearn kütüphanesini kullandım ve matplotlib kütüphanesi ile görselleştirdim. Ben test veri setimi kullanarak görselleştirme yaptım tabi bunu istediğimiz ver seti üzerinden gerçekleştirebiliriz.

Algotritmama başlamadan önce kütüphaneleri tanımladım. Daha sonra veri yolumu ekledim. Ve algoritamayı kurmaya başladım. Ben ikili sınıflandırma için Confusion Matrixi kullandım çünkü modelimin amacı buydu.

Burada etiket ve görüntü yollarını toplama işlemi yapılıyor

```
image_paths = []
labels = []

for label in os.listdir(image_dir):
    label_dir = os.path.join(image_dir, label)
    if os.path.isdir(label_dir):
        for image_name in os.listdir(label_dir):
            image_path = os.path.join(label_dir, image_name)
            image_paths.append(image_path)
            labels.append(label)
```

- image_paths: Bu liste, tüm görüntülerin dosya yollarını tutar.
- labels: Bu liste, her bir görüntünün sınıf etiketlerini tutar.
- `os.listdir(image_dir)`: `image_dir` dizinindeki tüm dosya ve klasörlerin isimlerini döndürür. Bu döngü, dizindeki her bir dosya veya klasör için tekrarlanır.
- os.path.join(image_dir, label): image_dir ve label'ı birleştirerek tam dosya yolunu oluşturur. Örneğin, image_dir "images" ve label "cat" ise, label_dir "images/cat" olacaktır.
- os.path.isdir(label_dir): label_dir'in bir dizin olup olmadığını kontrol eder. Eğer bir dizin ise, bu dizin içindeki görüntüler işlenecektir.
- `os.listdir(label_dir)`: `label_dir` dizinindeki tüm dosya isimlerini döndürür. Bu döngü, her bir görüntü dosyası için tekrarlanır.

- os.path.join(label_dir, image_name): label_dir ve image_name'i birleştirerek tam görüntü yolunu oluşturur. Örneğin, label_dir "images/cat" ve image_name "image1.jpg" ise, image_path "images/cat/image1.jpg" olacaktır.
- image_paths.append(image_path): Oluşturulan image_path'i image_paths listesine ekler.
- labels.append(label): Mevcut label'i labels listesine ekler

```
data = pd.DataFrame({
    'Image_Path': image_paths,
    'Label': labels
})
```

- `pd.DataFrame()`: Yeni bir DataFrame oluşturur. DataFrame, iki boyutlu, etiketli bir veri yapısıdır
- { 'Image_Path': image_paths, 'Label': labels }:Bir Python sözlüğüdür (dictionary). Anahtarlar (keys), DataFrame'in sütun adlarını temsil eder.
- 'Image_Path': Bu sütun adıdır ve karşılığı olarak image_paths listesi sağlanmıştır. Bu sütun, her bir görüntünün dosya yolunu içerir.
- 'Label': Bu da bir sütun adıdır ve karşılığı olarak labels listesi sağlanmıştır. Bu sütun, her bir görüntünün sınıf etiketini içerir.

```
data.to_csv("image_labels.csv", index=False)
```

Bu kod, oluşturulan pandas DataFrame'i bir CSV dosyasına kaydetmek için kullanılır.

```
# Gerçek etiketler
actual = data['Label']

# Tahmin edilen etiketler (şu anda gerçek etiketlerle aynı)
predicted = []
from keras.models import load_model
model_path = 'MobileNetV2/mobileNetV2.h5'
model = load_model(model_path)
```

Burada MobileNetV2 ile oluşturduğum modeli yüklüyor (matrixi bu modele göre çıkarıyor.)

```
for image_path in data['Image_Path']:

    image = tf.keras.preprocessing.image.load_img(image_path, target_size=(160,
160))
    image = np.array(image)
    x = preprocess_input(image)

    x = x.reshape((1,) + x.shape)
    y_prob = model.predict(x)[0]
    predicted_class = 'War' if y_prob[0] > 0.5 else
    predicted.append(predicted_class)


cm = confusion_matrix(actual, predicted)
```

- `tf.keras.preprocessing.image.load_img` fonksiyonu, belirtilen dosya yolundan bir görüntü yükler ve belirtilen boyuta (`target_size=(160, 160)`) yeniden boyutlandırır.
- image = np.array(image): : Yüklenen görüntü, daha sonra işlenmesi için bir NumPy dizisine dönüştürülür.
- Ön İşleme: `preprocess_input` fonksiyonu, genellikle modelin beklediği formatta görüntüyü ölçeklendirir ve normalleştirir. Bu fonksiyon, kullanılan modele bağlı olarak değişebilir.
- x = x.reshape((1,) + x.shape):Modelin girdi olarak tek bir örnek beklediği formata dönüştürülür. `(1,) + x.shape` ifadesi, örneğin `(1, 160, 160, 3)` boyutlarında bir dizi oluşturur.
- model.predict(x) fonksiyonu, modelin görüntü için sınıf olasılıklarını tahmin etmesini sağlar. y_prob[0] ifadesi, tahmin edilen olasılığı alır.
- predicted_class = 'War' if y_prob[0] > 0.5 0.5 olasılık eşiği kullanılarak sınıf belirlenir. Eğer olasılık 0.5'ten büyükse, görüntü 'War' sınıfına, aksi halde 'Normal' sınıfına atanır.
- Tahmin Edilen Sınıfları Saklama: predicted listesine tahmin edilen sınıflar eklenir.

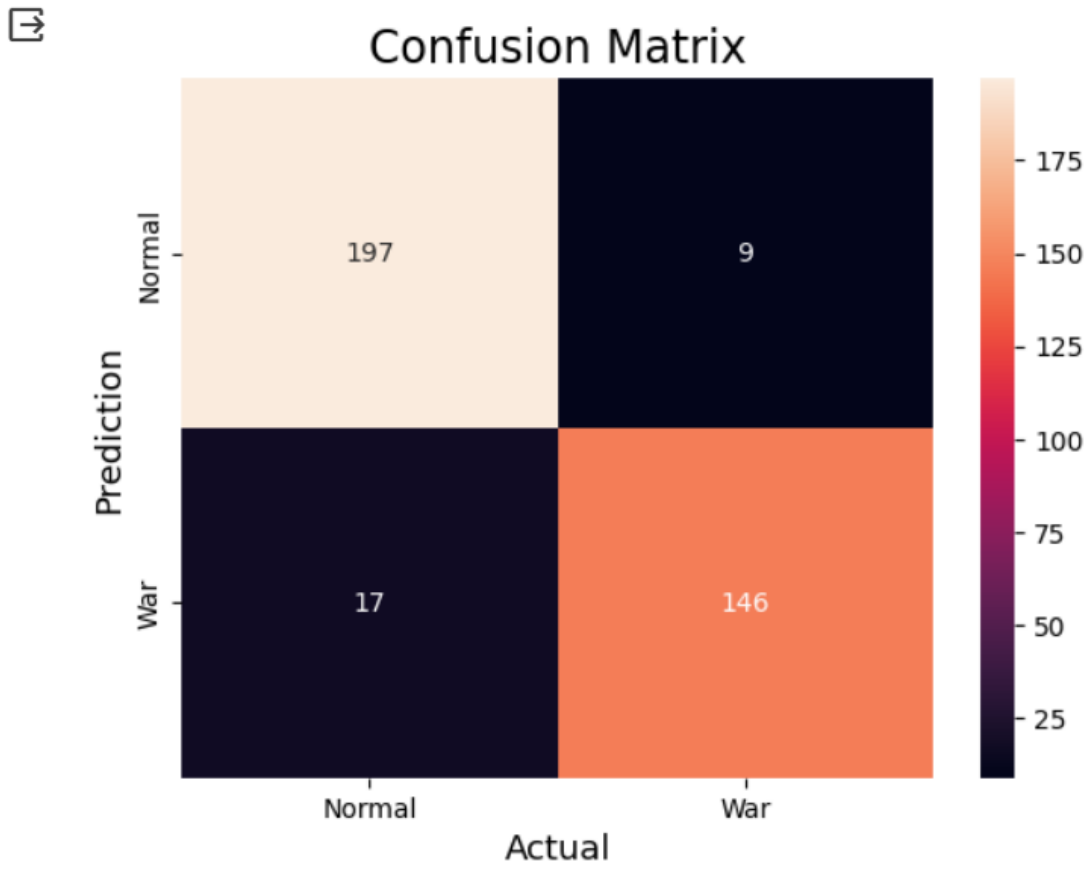
- confusion_matrix fonksiyonu, gerçek etiketlerle tahmin edilen etiketleri karşılaştırarak bir karışıklık matrisi oluşturur.
- actual: Gerçek sınıf etiketlerinin bulunduğu liste.
- predicted: Tahmin edilen sınıf etiketlerinin bulunduğu liste.

```
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['Normal', 'War'],
            yticklabels=['Normal', 'War'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('MobileNetV2',fontsize=17)
plt.show()
```

- sns.heatmap
- cm: Görselleştirmek istediğiniz karışıklık matrisi. cm daha önce confusion_matrix(actual, predicted) fonksiyonuyla oluşturulmuştu.
- annot=True: Bu parametre, her hücredeki değerin görüntülenmesini sağlar. Yani, matrisin her hücresindeki sayısal değerler ısı haritası üzerinde gösterilir.
- fmt='g': Bu parametre, hücrelerde gösterilecek değerlerin formatını belirtir. 'g' genel sayı formatıdır, yani sayıları olduğu gibi gösterir.
- xticklabels=['Normal', 'War'] **ve** yticklabels=['Normal', 'War']: Bu parametreler, x ve y eksenlerinde gösterilecek etiketlerdir. Bu durumda, sınıflar 'Normal' ve 'War' olarak etiketlenir.
- plt.ylabel
- 'Prediction': Y ekseni etiketi olarak 'Prediction' (Tahmin) metnini ayarlar.
- fontsize=13: Y ekseni etiketinin yazı boyutunu 13 olarak ayarlar.
- plt.xlabel
- 'Actual': X ekseni etiketi olarak 'Actual' (Gerçek) metnini ayarlar.
- fontsize=13: X ekseni etiketinin yazı boyutunu 13 olarak ayarlar.
- plt.title
- 'MobileNetV2': Grafiğin başlığı olarak 'MobileNetV2' metnini ayarlar.

- fontsize=17: Başlığın yazı boyutunu 17 olarak ayarlar.
- plt.show
- plt.show(): Grafiği ekranda görüntüler. Bu fonksiyon olmadan, grafik çizilmez ve gösterilmez.

Bu da confusion matrixin çıktısı;



TARTIŞMA

1. Eğitim ve Doğrulama Doğruluğu (Accuracy) Grafiği

- **İyi Yönler:**
  - Eğitim doğruluğu ve doğrulama doğruluğu benzer eğilimler gösteriyor, bu da modelin aşırı öğrenme (overfitting) yapmadığını işaret ediyor.
  - Eğitim ve doğrulama doğruluğu yüksek seviyelerde, bu da modelin iyi performans gösterdiğini gösterir.
- **Eksiklikler:**
  - İlk birkaç epoch'da doğrulama doğruluğu hızlı bir şekilde artmış ancak daha sonra yavaşlamış. Bu, modelin daha fazla epoch ile daha iyi hale gelmeyeceği anlamına gelebilir.
  - Doğrulama doğruluğu eğitim doğruluğundan biraz düşük, bu da modelin doğrulama verisinde biraz zorlandığını gösterebilir.

2. Eğitim ve Doğrulama Kayıp (Loss) Grafiği

- **İyi Yönler:**
  - Eğitim kaybı ve doğrulama kaybı benzer eğilimler gösteriyor, bu da modelin aşırı öğrenme yapmadığını işaret eder.
  - Eğitim ve doğrulama kayıpları zamanla azalmış, bu da modelin zamanla daha iyi öğrendiğini gösterir.
- **Eksiklikler:**
  - Doğrulama kaybı eğitim kaybına göre biraz daha yüksek. Bu, modelin doğrulama verisi üzerinde biraz daha zorlandığını ve belki de modelin genelleme yeteneğini artırmak için daha fazla regularization uygulanabileceğini gösterebilir.

3. Karışıklık Matrisi (Confusion Matrix)

- **İyi Yönler:**
  - Normal ve savaş fotoğraflarını büyük oranda doğru sınıflandırmış, bu da modelin iyi performans gösterdiğini gösterir.
  - Yanlış sınıflandırmalar az sayıda, bu da genel doğruluğun yüksek olduğunu işaret eder.
- **Eksiklikler:**
  - Model, 9 normal fotoğrafı savaş olarak, 17 savaş fotoğrafını ise normal olarak sınıflandırmış. Bu yanlış sınıflandırmalar, modelin bazı durumlarda zorluk çektiğini gösterebilir.
  - Belirli sınıflandırma hatalarının nedeni incelenmeli; belki de model bazı belirli özelliklere fazla odaklanıyor olabilir ve bu nedenle bu hatalar oluşabilir. Bu durumu çözmek için daha fazla veri veya veri artırma teknikleri kullanılabilir.

Genel Değerlendirme

İyi Yönler:

- Model yüksek doğruluk oranlarına ulaşmış ve genel olarak iyi performans göstermektedir.
- Eğitim ve doğrulama metrikleri benzer eğilimler gösteriyor, bu da modelin aşırı öğrenme yapmadığını işaret eder.
- Karışıklık matrisi, modelin büyük ölçüde doğru tahminler yaptığını gösterir.

Eksiklikler:

- Doğrulama doğruluğu ve kaybı, eğitim doğruluğu ve kaybına göre biraz daha kötü durumda. Bu, modelin doğrulama verisi üzerinde biraz daha zorlandığını gösterebilir.
- Yanlış sınıflandırma sayıları düşük olsa da, hataların nedenleri üzerine gidilerek modelin daha da iyileştirilmesi mümkün olabilir.

Öneriler:

- Modelin genelleme yeteneğini artırmak için veri artırma (data augmentation) yöntemleri daha geniş ölçekte kullanılabilir.
- Modelin performansını artırmak için dropout gibi regularization teknikleri eklenebilir.
- Yanlış sınıflandırmaların nedenlerini anlamak için hataların üzerine analiz yapılabilir ve veri setine benzer daha fazla örnek eklenebilir.

Kaynakça:

- **https://pythontutorials.eu/deep-learning/transfer-learning/**
- **https://github.com/amineHorseman/mobilenet-v2-custom-dataset**
- **https://deeplizard.com/learn/video/Zrt76AIbeh4**
- **https://blog.roboflow.com/how-to-train-mobilenetv2-on-a-custom-dataset/**
- **https://keras.io/api/applications/**
- **https://builtin.com/data-science/confusion-matrix-python**
- **https://medium.com/@gubrani.sanya2/understanding-confusion-matrix-with-python-76bf1d074408**