



TensorFlow Notları
(29/07/2021)
Fatih Es

Gözeceğiniz problem, "Celcius to Fahrenheit" olmaktadır. Bu örneğim, $f = c \cdot 1,8 + 32$ formülü ile kolaylıkla bulunabilir ancak buradaki hedef makine öğrenmesi modeli gerçekleştirmek

1) Kütüphanelerin içeri aktarılması burada tensorflow'u import ederken hataları gidermek istiyorsanız

```
import tensorflow as tf  
import numpy as np  
import logging  
logger = tf.get_logger()  
logger.setLevel(logging.ERROR)
```

2) Eğitim Verilerin Ayarlanması = Denetimli makine öğrenimi, bir dizi girdi ve çıktı verilen bir algoritma bulmakla ilgilidir. Bu yüzden 'celcius' ve 'fahrenheit' değerlerimizi veriyoruz.

```
celcius_g = np.array([-40, -10, 0, 8, 15, 22, 38], dtype=float)  
fahrenheit_a = np.array([-40, 14, 32, 46, 59, 72, 100], dtype=float)
```

3) Modelin Oluşturulması; sorun basit olduğunu için, bu çok tek bir nöron ile sadece bir katman yeter.

↳ Katman Oluştur = LO katmanını aşağıdaki konfig ile tf.keras.layers.Dense ile oluştur.

• input_shape=[1] ⇒ katmana girdinin tek bir değer olduğunu belirtir. Yani tek boyutlu bir listedir. Bu ilk (ve tek) katman olduğu için bu girdi teklikle tüm modelin girdi teklik.

• units=1 ⇒ katmandaki nöron sayısını belirtir. Nöron sayısı, katmanın problemi nasıl olacağının öğrenmek için kaç tane dahili değişkeni denemesi gerektiğini tamamlar. Bu son katman olduğu için aynı zamanda modelin çıktıının boyutudur.

```
LO = tf.keras.layers.Dense(units=1, input_shape=[1])
```

↳ Katmanları modelle birleştirir ⇒ Katmanlar tanımlandıktan sonra bir modelde monte edilmelidir.

```
model = tf.keras.Sequential([LO])
```

↳ penelde ⇒ "model = tf.keras.Sequential([tf.keras.layers.Dense(units=1, input_shape=[1])])" gibi kodlanır.

4) Modeli, kayıp (loss) ve optimize edici ifadeleriyle derleyin.

• Loss Fonk = Tahminlerin istenilen sonucdan ne kadar uzakta olduğunu ölçmenin yolu.

• Optimize Edici Fonk = Kaybi (loss değeri) azaltmak için dahili değerlerin ayarlanması.

```
model.compile(loss="mean_squared_error", optimizer=tf.keras.optimizers.Adam(0.01))
```

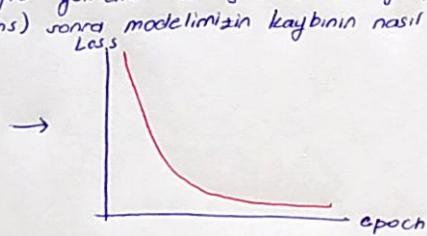
Burada kullanılan kayıp islevi (ort. kare hatası) ve optimize edici (Adam), bunun gibi basit modeller için standarttır. Optimize edicisinin önemli bir parçası "öğrenme oranıdır" (yukarıda "0.1") Bu modeldeki değerler ayarlanırken alınan adım boyutudur. Değer çok büyük olursa, modeli eğitmek için çok fazla iterasyon gereklidir. Çok büyük ise dovruluk değeri düşer. Genelde bu değer 0,1 ve 0,001 (varsayılan) aralığındadır.

5) Modeli Eğit ⇒ "fit" yöntemini kullanarak model eğitilir. Model, celcius değerlerini dır. Mevcut dahili değişkenleri (açıklıklar) kullanarak bir hesaplama yapar. Açıklıklar başlangıçta rastgele ayarlandığı için çıktı doğru değere yoton olmayacağındır. Gerekli çıktı ile olan çıktı arasındaki fark kayıp islevi ile hesaplanır ve optimize edici ile açıklıkların nasıl ayarlanması gereklidir. Bu "epochs" bağımsız değişkeni, bu doğrultunda kaç kez güncellirmesi gerektiğini belirtir. "verbose" değişkeni göntemin ne kadar çıktı ürettiğini kontrol eder.

```
history = model.fit(celcius_g, fahrenheit_a, epochs=500, verbose=False)
```

6) Epitim İstatistiklerini Görüntüle; "fit" yöntemi bir geçmiş 'history' nesnesi döndürür. Bu nesnesi, her epitim döneninden (epoch) sonra modelimizin kaybının nasıl düzleştiğini ^{Loss} görenetülemeğe iğin kullanabiliriz.

```
import matplotlib.pyplot as plt  
plt.xlabel('Epochs sayısı')  
plt.ylabel('Kazip boyutu')  
plt.plot(history.history['loss'])
```



7) Değerleri Tahmin Etmek İçin Modeli Kullanın

`print(model.predict([[100.0]])`) d^ü G^erek de^rer; $100 \times 1.8 + 32 = 212$ olmalıdır.
`>>> [[211.74]]` d^ü Dense katmanⁱ olan bir model oluşturduk.
d^ü 3500 örnekle eğitti^r. (7 çift de^rer * 500 tekrar)

8) Katman Ağırlıkları

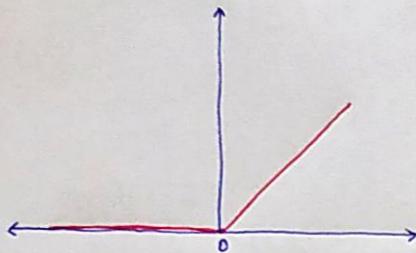
```
print ("Katman değerleri = {} ".format (l0.get_weights ()))  
''' : [array ([[ 1.79813 ]], dtype = float32), arr
```

İlk deşifreler 1.8'e oldukça yakın, ikinci de 82 deşifrene

BAZI TERIMLER:

- **Özellik (Feature)** = Modelimizin girdileri.
 - **Katman (Layer)** = Bir sınır apı içerisinde birbirine bağlı olan düğümler koleksiyonu
 - **Model** = Sınır apının temsil'i.
 - **Yogun ve Tambaglı (Dense and Fully Connected)** = Bir katmandaki her düğüm, önceki katmandaki her düğüme bağlanır.
 - **Ağırlıklar - Önyargılar (Weights - biases)** her düğüme bağlıdır.
↳ Modelin iç değişkenleri.
 - **Gradient Descent** = Kayıp fonk. kademeli olarak azaltmak için dahili değişkenleri her seferinde biraz değiştiren bir algoritma.
 - **Toplu (Batch)** = Sınır apının eğitimi sırasında kullanılan örnekler seti.
 - **Dönem (Epochs)** = Tüm eğitim veri kümesinin tam geçisi
 - **İleri Geçiş (Forward pass)** = Girişten çıktı değerlerinin hesaplanması.
 - **Geri Geçiş (Backward pass - back propagation)** = Dahili değişken ayarlamalarının optimizasyonuna göre, çıktı katmanından başlayarak ve her katmanda bir kez geri gidilerek hesaplanmasıdır.

The Rectified Linear Unit (ReLU)



ReLU, doğrultulmuş doğrusal birim anlamında gelir. Yandağı gibi görünen bir matematiksel iftedir.
Göründüğü gibi piriş negatif veya sıfır ise sıfır (0) çıkışı verir. Giriş pozitif ise piriş = giriş olur. $y = mx + b$ ile benzer. Ancak $gizlilik$ istenen gizli problem doğrusal olmaz. Bu durumlarda Dense katmanlarına ReLU eklemek sorunu gizmeye yardımcı eder.

ReLU, bir aktivasyon iftedir. Buna benzer iftedeler elbet vardır (Sigmoid, tanh, ELU) ancak en yaygınını ReLU'dur. Şimdi yeni terimlere göz atalım;

- Düzleştirme (Flattening) = 2D bir pürünthi \rightarrow boyutlu vektöre cevirmeye işlemi,
- ReLU = Bir modelin doğrusal olmayan problemleri gizlemesine iain veren aktivasyon fonk.
- Softmax = Olası her ikisi sınıfı için olasılıklar dağıtan bir ifted.
- Sınıflandırma (Classification) = İki veya daha fazla ikili kategorisini ayırt etmek için ML modeli

Tensorflow Veri Kümeleri : Tensorflow ile kullanma hazır veri kümesi sunar.
Veri kümeleri tipik olarak, sınırlı ağırlık eşitlik optimizasyon ve derişimlendirme deşimatlarında kullanılmak üzere farklı alt kümelerde ayrılır.

→ Eğitim (Training) Seti = Sınırlı ağırlık eğitmek için kullanılır.

→ Test Seti = Sınırlı ağırlık nihai performansını test etmek için kullanılır.

Test veri kümeleri, modelin daha önce hiç görmemişti veriler üzerinde kullanılır. Bu, modelin eğitim sırasında gördüğülerinin üzerinde nasıl penelketipini ve ebever yapıp yapmadığını görmemizi sağlar.

Aynı şekilde - doğrulama (Validation) setinin kullanılması da yaygındır. Bu veri kümesi eğitim için kullanılmaz. Bunun yerine, eğitim sırasında modeli test etmek için kullanılır. Bu, belirli sayıda eğitim setlerinden sonra yapılmış ve eğitimin nasıl ilerlediği hakkında bilgi verir. Örneğin eğitim sırasında kayıp azaltken doğrulama setindeki doğruluk bozulsrsa, bu modelin test setini etkilediğini gösterir.

GİYSİ GÖRÜNTÜLERİNİN SINIFLANDIRILMASI

Bu başlık altında, spor ayakkabı ve kadın giysi gibi görüntülerini sınıflandırmak için sınırlı API oluşturma ve eğitme işlenedir. Bu başlık altında, Tensorflow'da modeller oluşturmak ve eğitmek için ilk dizey bir API olan 'tf.keras' kullanıldı.

1) GEREKLİ KURULUM - İĞE AKTARMA -

! pip install - U tensorflow-datasets

...

import tensorflow as tf

Import tf datasets

import tensorflow_datasets as tfds

tfds.disable_progress_bar()

Helper - yardımcı kütüphane

import math

import numpy as np

import matplotlib.pyplot as plt

import logging

logger = tf.get_logger()

logger.setLevel(logging.ERROR)

2) "Fashion MNIST" VERİ SETİNİN İGE AKTARILMASI

Toplam 10 farklı kategoride 70.000 gri tonlamalı görüntü içeren veri seti bulunmaktadır. Düşük çözünürlükte 28×28 piksel görüntülerden oluşur.

İgi (modeli) eğitmek için 60K ve sınıflandırmayı ne kadar doğru yaptığıni görmek için 10K görüntü kullanacağız.

```
dataset, metadata = tfds.load('fashion-mnist') as_supervised=True, with_info=True  
train_dataset, test_dataset = dataset['train'], dataset['test']
```

Görseller 28×28 dizidir ve piksel değerleri $[0, 255]$ aralığındadır. Etiketler $[0, 9]$ aralığında bir tam sayı dizisidir. Görüntünün temsil ettiği sınıfı veri:

0 → Tişört, 1 → Pantolon, 2 → Kazak, 3 → Elbise, 4 → Çeşit, 5 → Sandalet

6 → Gömlek, 7 → Spor A., 8 → Ganta, 9 → Ayak bileği botu

```
class_names = metadata.features['label'].names
```

3) VERİLERİ KEŞFEDİN

```
num_train_examples = metadata.splits['train'].num_examples # OUT = 60000
```

```
num_test_examples = metadata.splits['test'].num_examples # OUT = 10000
```

4) VERİ ÖN İŞLEME

Görsel verilerindeki her piksel değeri 0-255 aralığında bir tam sayıdır. Modelin düzgün çalışması için bu değerin 0-1 aralığına normalleştirilmesi gereklidir. Normalleştirme işlevi (fonk.) oluşturuyoruz ve bunu test ve eğitimin setindeki her görüntüye uyguluyoruz.

```
def normalize(images, labels):
```

```
    images = tf.cast(images, tf.float32)
```

```
    images /= 255
```

```
    return images, labels
```

```
# .map() işlevi her bir veriye normalize işlevini uygular
```

```
train_dataset = train_dataset.map(normalize)
```

```
test_dataset = test_dataset.map(normalize)
```

Dataset'in ilk kullanımında görselleri diskten yükleyecektir.

Önbelleğe atıp orda tutmak eğitimi hızlandırır.

```
train_dataset = train_dataset.cache()
```

```
test_dataset = test_dataset.cache()
```

5) İŞLENEN VERİLERİ KEŞFEDİN

Nasıl görüntü hakkında bilgi sahibi olmak için görüntüün birini gözlemez.

Tek bir görüntü din ve yeniden ekillendirerek renk boyutunu kaldırır.

```
for image, label in test_dataset.take(1): # Eğitimin setindeki ilk 25 resmi görüntüle ve  
    break # her resmin altındaki sınıfını yazdır.
```

```
image = image.numpy().reshape((28, 28))
```

plot the image

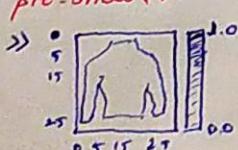
```
plt.figure()
```

```
plt.imshow(image, cmap=plt.cm.binary)
```

```
plt.colorbar()
```

```
plt.grid(False)
```

```
plt.show()
```



```
for i, (image, label) in enumerate(test_dataset):
```

```
    image = image.numpy().reshape((28, 28))  
    plt.subplot(5, 5, i+1)
```

```
    plt.xticks([])
```

```
    plt.yticks([])
```

```
    plt.grid(False)
```

```
    plt.imshow(image, cmap=plt.cm.binary)
```

```
    plt.xlabel(class_names[label])
```

```
plt.show()
```

6) MODELİ OLUSTURUN = Sınıf opinīn oluşturulması, modelin katmanlarının yapılandırılması ve ardından modelin derlenmesini içenir.

6. 3) Katmanların Yapılandırılması

→ Bir sınıf opinīn temel yapı taşı katmandır (layer). Bir katman, kendisine verilen verilerden bir temsil oluşturur.

→ Derin öğrenmenin çoğu, basit katmanları zincirlemekten oluşur. 'tf.keras.layers.Dense' gibi çoğunu katman, eğitim sırasında ayaktan ("öğrenilen") dahili parametrelerle sahiptir.

model = tf.keras.Sequential([

tf.keras.layers.Flatten(input_shape=(28, 28, 1)),
tf.keras.layers.Dense(128, activation=tf.nn.relu),
tf.keras.layers.Dense(10, activation=tf.nn.softmax)

])

! Bu opinīn 3 katmanı vardır:

→ Giriş = 'tf.keras.layers.Flatten' = Bu katman, görüntülerin 28×28 piksellik 2 boyutlu bir diziden 784 (28×28) piksellik bir boyutlu düzeye çevirir. Bu katmanı girdiğindeki piksel sıralarını折叠 ve bunları hizalamak olarak düşünebiliriz. Bu katman sadece yeniden biçimlendirmede kullanıldığı için öğrenilecek parametreye sahip değildir.

→ Gizli = 'tf.keras.layers.Dense' = 128 nörondan oluşan yoğun bir katman. Her nöron (düğüm), önceki katmandaki 784 düğümün hepsinden girdi alır, bu girdiğin eğitim sırasında öğrenilecek gizli parametrelerle şıraşılıklarındır ve bir sonraki katmana tek bir değer verir.

→ Çıktı = 'tf.keras.layers.Dense' = 128 nöronun ardından 10 düğümlü softmax katmanı. Her düğüm bir sınıfı temsil eder. Önceki katmanın banzer ettiği, son katman, önceki katmandan 128 düğümünden girdi alır ve girdiğinin o sınıfın alt olma olasığını temsil eden 0-1 aralığında değer üretir. 10 düğümün bu değer toplamı 1'dir.

! !

'softmax' aktivasyonu ve 'Sparse Categorical Crossentropy()' kullanımı sorunlar var ve bunları 'tf.keras' modeli düzeltti. Daha güvenli yaklaşım 'Sparse Categorical Crossentropy(from_logits=True)' ile doğrusal bir çıktı (etkinleştirme işlevi olmadan) kullanmaktadır.

6.2) MODELİN DERLENMESİ

Model eğitime hazır olmadan önce birkaç ayara daha ihtiyaci vardır. Buntar, derleme sırasında (compile) eklenir.

- Kayıp (Loss) Fonksiyonu; model çıktılarının istenen çıktıdan ne kadar uzaktır ekl. olur.
- Optimize Edici (optimizer); Kaybu en azı indirmek için modelin iç parametrelerini ayarlamak için algoritma.
- Metrics; Eğitim ve test adımlarını izlemek için kullanılır. Bu direktte doğruluk (accuracy) kullanılır. Doğru sınıflandırılmış görüntülerin oranı.

model.compile(optimizer='adam',

loss='tf.keras.losses.SparseCategoricalCrossentropy()',
metrics=['accuracy'])

7) MODELİ EGİT

- İlk olarak `train` (eğitim) veri kümesini için yineleme davranışını tanımlar;
- 1) `'dataset.repeat()'` ögesini belirterek sonsuza kadar tekrarlayın. (açıkadaki `'epochs'` parametresi ne kadar sıklıkla eğitimi gerçekleştireceğini sınırlar)
 - 2) `'dataset.shuffle(60000)'` sırası rastgele ayarlar, böylece model örneklere sırasından ögrenmeye başlar.
 - 3) `'dataset.batch(32)'` ögesi `'model.fit'` ile modelin deej. güncellerken 32 görüntü - etiketten oluşan yığın kullanmasını belirtir.
- `'model.fit'` yöntemi çağrılarak eğitimin gerçekleştirileceğidir.
- 1) `'train_dataset'` ile kullanarak modeli besleyin.
 - 2) Modelin görüntü ve etiketleri ilişkilendirmeyi öğrenir.
 - 3) `'epochs=5'` parametresi, eğitimi eğitim veri kümesinin 5 tam yenilemesi ile sınırlar. Yani toplam $5 \times 60000 = 300\text{ K}$ örnek.

`BATCH_SIZE = 32`

`train_dataset = train_dataset.cache().repeat().shuffle(num_train_examples)`

`test_dataset = test_dataset.cache().batch(BATCH_SIZE)`

`model.fit(train_dataset, epochs=5, steps_per_epoch=math.ceil(num_train_examples/BATCH_SIZE))`

8) DOĞRULUĞU DEĞERLENDİRİN

Modelin test veri kümesinde nasıl performans gösterdiğine bakalım. Doğruluğu değerlendirmek için test veri setindeki tüm örnekleri kullanın.

`test_loss, test_accuracy = model.evaluate(test_dataset, steps=math.ceil(num_test_examples/32))`
=> 0.3702

Şu an test veri setindeki doğruluk, eğitim veri setindeki doğruluktan küçükter. Bu durum son derece normaldir.

9) TAHMİNLERDE BULUNUN VE KEFFEDİN

Eğitilen model ile bazı görüntüler hakkında tahminler yapmak için kullanabiliriz.

```
for test_images, test_labels in test_dataset.take(1):
    test_images = test_images.numpy()
    test_labels = test_labels.numpy()
    predictions = model.predict(test_images)
```

`predictions.shape #>>(32,10)`

Burada model, test setindeki her görüntü için etiket tahmin etmiştir. İlk tahmine bakalım.

`predictions[0] #>> array([—, —, —, —, —, 0.0, —])`

Tahmin 10 sayıdan oluşan bir dizidir. Buntar, görüntüdeki 10 farklı sınıfı parçasının her birine kareklik geldiğine dair modelin "gövenini" tanımlar. Hangi etiketin en yüksek güven değerine sahip olduğunu görebiliriz.

`np.argmax(predictions[0]) #>>> 4`

Dolayısıyla model, bu görüntünün bir `cat` veya `class_name[4]` olduğunu emin.

Bunun doğru olduğunu kontrol etmek için test etiketini kontrol edebiliriz.

`test_labels[0]`

`#>> OUT=4`

ÖZET

Bu bölümde giyim eşyalarının görüntülerini sınıflandırmak için bir sinir ağı eğitti - Bu yapılmak için 70.000 gri tonlamalı giyim eşyası görüntüyü içeren 'Fashion MNIST' veri bileseni kullanıldı. Bu görüntüye ağı eğitmek ve kalan 10类'yi modeli test etmek için kullanıldı. Bu görüntüler ile sinir ağını basitlemek için 28×28 görüntüsünden 784 elementli bir boyutlu diziler elde ettik. Ağımlar, 128 nöron ve 10 çıkış etiketine karşılık gelen 10 sınıfı çıktı katmanından oluşuyor. Bu 10 çıktı katmanı, her sınıf için olasılık temsil eder. Softmax aktivasyon fonksiyonu olasılık dağılımlarını hesapladı.

• Regresyon = Tek bir değer veren model, örneğin evin değerinin tahmini.

• Sınıflandırma = Geçitli kategorilerde olasılık dağılımını gösteren model.

» Chapter 4

Evrileşim, bir görüntüye filtre (kernel) uygulama işlemidir. Matrisimiz havuzlama ise, alt örneklere yoluyla görüntünün boyutunu kısıtlayıcı işlemidir.
Keras'ta 'Conv2D' katman türü kullanılarak sinir ağı modeline evrileşimi katmanları eklenebilir. Bu katman, "Dense" katmanına benzer ve doğru değerlere ayarlanması gereken ağırlıklara ve önyargılara sahiptir. 'Conv2D' katmanı ayrıca değerlerinin de ayarlanması gereken filtrelerle sahiptir. Bu nedenle filtre matrisinin içindeki değerler, doğru aktiflik üretmek için ayarlanır.

- CNN'ler = Evrileşimelik sinir ağı. Yani, en az bir evrileşim ağına sahip bir ağı. Tipik bir CNN havuz (pooling) katmanları ve yapın (Dense) katmanları da içerir.
- Evrileşim (Convolution) = Bir görüntüye bir kernel (filtre) uygulama süreci
- Kernel/Filtre = Girdiye parçalara dönüştürmek için kullanılan girdiden kategoriye matris
- Doldurma (Padding) = Giriş görüntüsünün etrafına genellikle 0 gibi bir değerde piksel ekleme.
- Havuzlama (Pooling) = Alt örneklere yoluyla bir görüntünün boyutunu kısıtlayıcı işlemi. Birbirlerinden havuzlama katmanı vardır. Örneğin, ortalamaya havuzlama, ortalamayı alarak birçok değeri tek bir değere düşürür. Ancak, 'max pooling' en yaygın olanıdır.
- Maxpooling = Birçok değerin, aralarından maksimum değer alınarak tek bir değere düşürtüldüğü bir havuzlama işlemidir.
- Adım (stride) = kernel'i (filtreyi) görüntü boyunca kaydıracağın piksel sayısı
- Alt örnekleme (Downsampling) = Bir görüntünün boyunu kısıtlayıcı işlemi.

EVRİSEL SINIR AĞLARI İLE GÖRÜNTÜ SINIFLANDIRMA

Bu başlık altında - Tensorflow'da modeller oluşturmak ve eğitmek için常用 API olan 'tf, keras' kullanılmaktadır.

1) YÜKLEMELER VE KÜTÜPHANELERİ İÇE AKTARMA:

- import tensorflow as tf
- import tensorflow_datasets as tfds
- tfds.disable_progress_bar()
- import math
- import numpy as np
- import matplotlib.pyplot as plt
- import logging
- Logger = tf.get_logger()
- Logger.setLevel(logging.ERROR)

2) 'Fashion MNIST' VERİ KÜMESİNİN İÇE ALTARI LMASI

10 kategoride 70K gri tonlamalı görüntü içeren Fashion MNIST kullanılır. Görüntüler = 28x28. Aşağı eğitmek için buk gürültü ve dğin sınıflandırması ne kadar doğru öğrenildiğini değerlendirmek için 10K görüntü kullanacağız.

```
dataset, metadata = tfds.load('fashion_mnist', as_supervised=True, with_info=True)
train_dataset, test_dataset = dataset['train'], dataset['test']
```

Veri kümesini yüklemek, bir eğitim veri kümesi ve test veri kümesinin yanı sıra meta verileri dndür. Görüntüler 28x28, 2D bir dizidir ve piksel değerleri 0-255 arasında. Etiketler 0-9 arasında tam sayılardır:

0 → Tişört 1 → Pantolon 2 → Sütyen 3 → Elbise 4 → Çeket
5 → Sandalet 6 → Gömlek 7 → Spor ayakkabı 8 → Şanta 9 → Ayak bileği botu

Her görüntü tek bir etikette eğlenceli. Jnif adları veri kümesine dahil edilmemiinden, daha sonra görüntülerini gizlenerek kullanmak da burda sağlanmalıdır.

```
class_name = ['T-shirt/Top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Skirt', 'Sneaker', 'Bag', 'Ankle boot']
```

2.1) Verileri Keşfedin

```
num_train_examples = metadata.splits['train'].num_examples # >> 60 000
num_test_examples = metadata.splits['test'].num_examples # >> 10 000
```

3) VERİLERİ DN İŞLEME

Görüntü verilerindeki her pikselin değeri 0-255 arasında. Modelin düzgün gidişini sağlamak için bu verilerin 0-1 aralığında normalleştirilmesi gereklidir. Bir normalleştirme işlemi oluşturuyoruz ve ardından bunu test ve eğitim veri kümelerindeki her görüntüye uygularız.

```
def normalize(images, labels):
```

```
    images = tf.cast(images, tf.float32)
    images /= 255
    return images, labels
```

```
# Harita işlevi, normalleştirme işlemini verisetinin tamamına uygular.
```

```
train_dataset = train_dataset.map(normalize)
```

```
test_dataset = test_dataset.map(normalize)
```

```
# Datasetinin ilk kullanıldığından diktten yüklenenecektir. Onbelki te tutmak eğitimi istandır.
```

```
train_dataset = train_dataset.cache()
```

```
test_dataset = test_dataset.cache()
```

3.1) İşlenen Verileri Keşfedin

```
for image, labels in test_dataset.take(10):
    break
image = image.numpy().reshape((28, 28))
plt.figure()
plt.imshow(image, cmap=plt.cm.binary)
plt.colorbar()
plt.grid(False)
plt.show()
```

```
plt.figure(figsize=(10, 10))
i = 0
for (image, label) in test_dataset.take(25):
    image = image.numpy().reshape((28, 28))
    plt.subplot(5, 5, i+1)
    plt.xticks([1])
    plt.yticks([1])
    plt.grid(False)
    plt.imshow(image, cmap=plt.cm.binary)
    plt.xlabel(class_names[label])
    i += 1
plt.show()
```

4) MODELİ OLUSTURUN

Sınıf ağıının oluşturulması, modelin katmanlarının yapılandırılması ve ardından modelin derlenmesi.

4.1) Katmanların Ayartılması

Bir sınıf ağıının temel yapı taşı katmanlardır (layer). Bir katman, kendisine verilen bir temsil etkiler.

Denim örenmenin olduğu, bağıt katmanları birbirine zincirlenerek olur. "tf.keras.Dense" gibi yoğun katman, eğitim sırasında ayarlanan (öğrenilen) dahili parametrelerle sahiptir.

model = tf.keras.Sequential ([

tf.keras.layers.Conv2D (32, (3,3), padding='same', activation=tf.nn.relu,

input_shape=(28,28,1)),

tf.keras.layers.MaxPooling2D ((2,2), strides=2),

tf.keras.layers.Conv2D (64, (3,3), padding='same', activation=tf.nn.relu),

tf.keras.layers.MaxPooling2D ((2,2), strides=2),

tf.keras.layers.Flatten (),

tf.keras.layers.Dense (128, activation=tf.nn.relu),

tf.keras.layers.Dense (10, activation=tf.nn.softmax)

])

Bu ağı katmanları sunlardır:

o "convolutions", tf.keras.layers.Conv2D and MaxPooling2D = Bu iki çift Conv/MaxPool ile birlikte katmanı girdi pürüntüsüne uygulanan, dolgu (padding) kullanarak orijinal pürüntü boyutunu koruyan ve 32 kırımlı (convoluted) çıktı pürüntü oluştururan bir Conv2Dfiltredir. (3,3). Böylece bu katman girdiyle aynı boyutta 32 kırımlı pürüntü oluşturur. Bundan sonra, bir MaxPooling2D ((2,2)) kullanılarak 32 çıktıının boyutu 2 adında kısıltılır. Bir sonraki Conv2D aynca bir (3,3) kerneli (çekirdek) sahiptir, 32 pürüntüyü girdi olarak alır ve 64 çıktı oluşturur. Maxpooling2D katmanı tarafından boyut olarak kısıltılır.

o output "tf.keras.layers.Dense" = 128 nöron, ardından 10 dğrmış softmax katmanı. Her dğrmüş bir sınıfın sınıfı temsil eder. Onceki katmandan olduğu gibi, son katman, onceki katmandan 128 dğrmenden girdi alır ve pürüntünün o sınıfa ait olma olasılığı temsil eden 0-1 aralığında bir değer verir. 10 dğrmüş değerinin toplamı bir (1)'dır.

Softmax aktivasyonu ve Sparse Categorical Crossentropy () kullanımda sorunlar vardır

ve bunlar tf.keras modeli tarafından yemalanmıştır. Genel olarak data pürüntü yaklaşımları, Sparse Categorical Crossentropy (from_logits=True) ile depreşusal bir aleti (etkin. elev. yok)

4.2) Modelin Derlenmesi

Model eğitime hazır olmadan önce birkaç ağıra daha ihtiyaci vardır. Bunlar modelin derlenmesi sırasında eklenir:

o Kafip Fonk. (Loss Function) = Model çıktılarının istenen çıktıdan ne kadar uzakta olduğunu ölçmek için bir algoritma. Eğitimin amacı, bu kaygıları elimektir.

o Optimizer Edici (Optimizer) = Kayrı en azı indirmek için modelin iç parametrelerini, ayarlamak için bir algoritma.

o Metrikler (metrics) = Eğitim ve test adımlarını izlemek için kullanılan. Bu örnek doğruluku (accuracy) kullanır, doğru sınıflandırılmış pürüntülerin oranı.

model.compile (optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(),

metrics=['accuracy'])

5) MODELİN EGİTİLMESİ

İlk olarak, ~~dataset~~ eğitim veri kümesi için yineleme davranışını tanımlarız:

- 1) 'dataset.repeat()' ifadesini belirterek sonsuza kadar tekrarlayınız (aşağıda dikkatler 'epochs' parametresi, ne kadar sıklıkla eğitim gerçekleştiripimizi sınırlar)
- 2) 'dataset.shuffle(60000)' sırasıyla rastgele ayırtır, böylece modelimiz öneklerin sırasından bir sey öğrenmeyecektir.
- 3) 'dataset.batch(32)', model.fit'e, model değıstırkenin güncellerken 32 boyutlu ve etiketten oluşan yapıntaları kullanmasını sağlar.

Model.fit yöntemi gördüğümüzde eğitimin gerçekleştirildiği.

1) train_dataset kullanarak eğitim verileriyle modeli besleyin.

2) model, görtütürleri ve etiketleri ilişkilendirmeyi öğrenir.

3) 'epochs=5' parametresi, eğitimi train verisetinin 5 tam yenilenmesi ile sınırlar.
Yani $5 * 60K = 300000$ örnek olur.

BATCH_SIZE = 32

train_dataset = train_dataset.cache().repeat().shuffle(num_train_examples).batch(BATCH_SIZE)
test_dataset = test_dataset.cache().batch(BATCH_SIZE)

model.fit(train_dataset, epochs=10, steps_per_epoch=math.ceil(num_train_examples/BATCH_SIZE))

Model iterasyonları, kayıp ve doğruluk metrikleri pöntülenir.

Bu model, eğitim verilerinde yaklaşık 0.97 (veya %97) doğruluğuna ulaşır.

6) DOĞRULUĞU DEĞERLENDİRME

Burdan, modelin test veri kümesinde nasıl performans gösterdiği - Doğrulugu değerlendirmek için test veri setindeki tüm önekleri kullanın.

test_loss, test_accuracy = model.evaluate(test_dataset, steps=math.ceil(num_test_examples/32))

Sonuç olarak, test veri kümesindeki doğruluk, eğitim veri kümesindeki doğruluktan daha düşük.
Model, 'train_dataset' üzerinde eğitildiği için bu tamamen normaldir.

7) TAHMİNLERDE BULUNMA VE KEEFETME

Eğitilen modeli bazı görüntüler hakkında tahmin yapmak için kullanabiliriz.

for test_images, test_labels in test_dataset.take(1):

test_images = test_images.numpy()

test_labels = test_labels.numpy()

predictions = model.predict(test_images)

predictions.shape # >> (32, 10)

Burada model, test setindeki her görüntü için her bir etiketin olasılığını tahmin etmiştir.
predictions[0] #>> array([_____, _____, _____, _____, dtype=float32])

np.argmax(predictions[0]) #>> 4

Bu nedenle model genellikle bu görüntünün bir ceket veya class_names[4] olduğunu biliyor.

test_labels[0] #>> 4

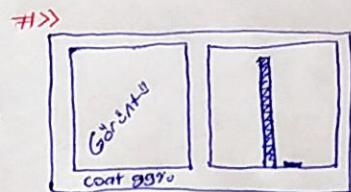
→ 10 sınıf tahminlerinin tamamına bakmak için bunun grafğini oluşturabiliriz.

```
def plot_image(i, predictions_array, true_labels, images):
    predictions_array, true_label, img = predictions_array[i], true_labels[i], images[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(img[0:280, 0], cmap=plt.cm.binary)
    predicted_label = np.argmax(predictions_array)
    if predicted_label == true_label:
        color = 'blue'
    else:
        color = 'red'
    plt.xlabel(f'{true_labels[i]} {100 * np.max(predictions_array)}% ({predicted_label})',
               class_names[predicted_label],
               color=color)

def plot_value_array(i, predictions_array, true_label):
    predictions_array, true_label = predictions_array[i], true_labels[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    thisplot = plt.bar(range(10), predictions_array, color="#777777")
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)
    thisplot[predicted_label].set_color('red')
    thisplot[true_label].set_color('blue')
```

→ 0. indeksteki görüntüye bakalım

```
i=0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions, test_labels)
```

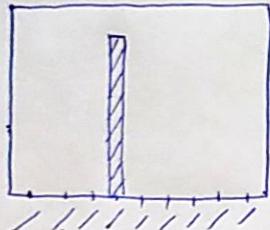


→ Tahminlerde birlikte birkaç resim gösterelim. Doğru tahminler mavi yanlış tahminler kırmızı.

Tahmin edilen etiket ikin yine deyi verir.

```
num_rows = 5
num_cols = 3
num_images = num_rows * num_cols
plt.figure(figsize=(2 * 2 * num_cols, 2 * num_rows))
for i in range(num_images):
    plt.subplot(num_rows, 2 * num_cols, 2 * i + 1)
    plot_image(i, predictions, test_labels, test_images)
    plt.subplot(num_rows, 2 * num_cols, 2 * i + 2)
    plot_value_array(i, predictions, test_labels)
```

- ~ San olarak, tek bir görüntü hakkında tahminde bulunmak için eğitilmiş modeli kullanın.
 $\text{img} = \text{test_images}[0]$
 $\text{print(img.shape)} \quad \# \gg \text{OUT} = (28, 28, 1)$
 - 'tf.keras' modelleri, bir seferde tek bir grup veya örneklere koaksiyonu üzerinde tahminler yapmadı için optimize edilmiştir. Yani tek bir resim kullanılarak olsak bile listeyle eklenmelii.
 $\text{img} = \text{np.array(img)}$
 $\text{print(img.shape)} \quad \# \gg \text{OUT} = (1, 28, 28, 1)$
 - Örnek resmi tahmin edelim;
 $\text{predictions-single} = \text{model.predict(img)}$
 $\text{print(predictions-single)} \quad \# \gg \text{OUT} = [[_, _, _, _, _, _]]$
- $\text{plot_value_array}(0, \text{predictions-single}, \text{test-labels})$
 $= \text{plt.xticks(range(10), class-names, rotation=45)}$



'model.predict', veri yığınındaki her görüntü için bir tane olmak üzere bir listesini döndürür. Grupta sadece resminiz için olan tahmini alalım;
 $\text{np.argmax}(\text{predictions-single}[0]) \quad \# \text{OUT} \rightarrow 4$

Daha önceki gibi model yani ceket etiketini tahmin etti.

Chapter : 5

KÖPEKLER ve KEDİLER GÖRÜNTÜ SINIFLANDIRILMASI (GÖRÜNTÜ BÜYÜTME YOK)

Bu başlık altında kedi ve köpek resimlerini nasıl sınıflandıracağımızı öğreneceğiz. 'tf.keras.Sequential' modelini kullanarak bir görüntü sınıflandırıcı oluşturacaz. 'tf.keras.preprocessing.image.ImageDataGenerator' kullanarak verileri yükleyeceğiz.

1) PAKETLERİN İÇE AKTARILMASI

Gerekli paketleri içe aktararak başlayalım.

↳ os = dosyaları ve dizin yapısını oluşturmak için

↳ numpy = tf. keras'daki bazı matris matematiği için

↳ matplotlib.pyplot = grafik çizmek ve eğitim doğrulama verilemizi görmek için.

```
import tensorflow as tf
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
import os
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
import logging
```

```
logger = tf.get_logger()
```

```
logger.setLevel(logging.ERROR)
```

2) VERİ YÜKLEME

Görüntü sınıflandırıcımızı oluşturmak için verisetini indiriyoruz. Bu veri kümlesi, Kaggle'in 'Dogs vs. Cats' veri kümnesinin filtrelenmemiş hali. (Microsoft Research tarafından sağlanıyor) Bu başlık altında Colab dizten veri okuyan ImageDataGenerator kullanır. Bu nedenle, bir URL'den Dogs vs Cats'i doğrudan indirmemiz ve onu Colab dosya sisteme yapamamız gerekiyor.

↳ URL = 'https://storage.googleapis.com/mledu-datasets/cats-and-dogs-filtered.zip'

↳ zip_dir = tf.keras.utils.get_file('cats-and-dogs-filtered.zip', origin=URL, extract=True)

İndirmiş olduğumuz veri setinin dizin yapısı:

cats_and_dogs_filtered

|_ train

 |--- cats: [cat0.jpg...]

 |--- dogs: [dog0.jpg...]

|_ validation

 |--- cats: [cat2000.jpg...]

 |--- dogs: [dog2000.jpg...]

Dizinleri aşağıdaki terminal komutuyla listeleyebiliriz.

zip_dir_base = os.path.dirname(zip_dir) } tüm dizinleri verir.

| find \$zip_dir_base -type d -print

Simdi de eğitim ve doğrulama için uygun dosya yolu ile deşşeklenleri atayalım.

base_dir = os.path.join(os.path.dirname(zip_dir), 'cats-and-dogs-filtered')

train_dir = os.path.join(base_dir, 'train')

validation_dir = os.path.join(base_dir, 'validation')

train_cats_dir = os.path.join(train_dir, 'cats')

train_dogs_dir = os.path.join(train_dir, 'dogs')

validation_cats_dir = os.path.join(validation_dir, 'cats')

validation_dogs_dir = os.path.join(validation_dir, 'dogs')

2.1) Verilerin Antlaşılması

Eğitim ve doğrulama döntümlerinde kategorik fotoğraflarımız var baktanız;

```
num_cats_tr = len(os.listdir('train-cats-dir')) # 1000  
num_dogs_tr = len(os.listdir('train-dogs-dir')) # 1000  
num_cats_val = len(os.listdir('validation-cats-dir')) # 500  
num_dogs_val = len(os.listdir('validation-dogs-dir')) # 500
```

```
total_train = num_cats_tr + num_dogs_tr # 2000
```

```
total_val = num_cats_val + num_dogs_val # 1000
```

3) MODEL PARAMETRELERİNİN AYRANMASI

Kolaylık sağlamak adına veri setimizi dn islemlerken ve eğitimi eğitirken kullanılır. degis_tanımla.

BATCH_SIZE=100 #model deş. güncellemeden önce izlenecek eğitim dökme sayısı

IMG_SHAPE=150 #Eğitim verimiz 150 piksel genişlik ve 150 piksel yükseklik. sahip görseller.

4) VERİ HAZIRLAMA

Görseller, oja beslenmeden önce uygun şekilde döndən iflenmiş kafan notta tensörlerine bölmeliydi. Bu görsellerin hazırlanmasında yer alan adımlar şöyledir;

1- Diskten görsellerin okunması.

2- Bu görsellerin içeriğini okun ve RGB içeriğinin doğru uygun 128x128 format. dönüştürün.

3- Buntarı kafan notta tensörlerine çevirin (dönüştürün)

4- Jelir apları tükete pırıf deşerlerle updatemayı tercih ettiğim için, tensörleri 0-255 arası farklı deşerlerden 0-1 arası deşereye yeniden öläklendirin.

Tüm bu görevler "tf.keras.preprocessing.image - Image Data Generator" sınıfı ile yapılabilir.

```
train_image_generator = ImageDataGenerator(rescale = 1./255)
```

```
validation_image_generator = ImageDataGenerator(rescale = 1./255)
```

Eğitim ve doğrulama görselleri için oluşturularımı tanımladıktan sonra, "flow_from_directory" yöntemi ile görselleri diskten yükler, yeniden öläklenirme uygular ve tek satır kod ile buntarı yeniden boyutlandırır.

```
train_data_gen = train_image_generator.flow_from_directory(batch_size = BATCH_SIZE,  
directory = 'train-dir',  
shuffle = True,
```

→ iki sınıfa ait 2K pörşel bulundu.

```
target_size = (IMG_SHAPE,IMG_SHAPE),  
class_mode = 'binary')
```

```
val_data_gen = validation_image_generator.flow_from_directory(batch_size = BATCH_SIZE,  
directory = 'validation-dir',
```

→ iki sınıfa ait 1K pörşel bulundu.

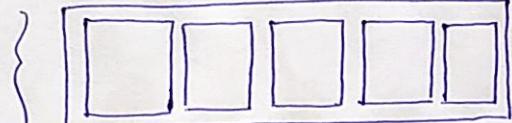
```
shuffle = False,  
target_size = (IMG_SHAPE,IMG_SHAPE),  
class_mode = 'binary')
```

4.3) Eğitimin Görüntülerini Görselleştirme

Eğitim görünümlerimizi eğitimin oluşturucudan bir numpy döngüsü içinde ve ardından 'matplotlib' kullanarak birkoçunu görselleştirebiliriz.

```
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20, 20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.show()

plotImages(sample_train_images[:5])
```



5) MODEL OLUSTURMA

5.1) Modelin Tanımlanması

Model, her birinde maksimum havuz katmanı bulunan dört evrişim (convolution) bloğundan oluşur. Ardından, yeniden etkileşime sahip 512 Üniteli tam bağıntılı bir katman (relu) yerir. Model, 'softmax' kullanarak iki sınıf (küpeler ve kediler) için olasılıkları alır.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(2)
])
```

5.2) Modelin Derlenmesi

Her zamanki gibi 'adam' optimizatörünü kullanacağız. Bir 'softmax' sınıflandırması, 'sparse_categorical_crossentropy' kullanacağız. Ayrıca optimizatör eğitikten her bir qapada (epochs) eğitimi ve doğrulama durumuna bilmek istiyoruz.

```
model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
               metrics=['accuracy'])
```

5.3) Model Özetİ

```
model.summary()
```

{ Her katmandanın durumun detayı verir.

5.4) Modelin Eğitilmesi

Gruplarınız bir generatorden (Image Data Generator) geldiği için, 'fit' yerine 'fit-generator'

EPOCHS = 100

history = model.fit_generator(train_data_gen,

steps_per_epoch = int(np.ceil(total_train / float(BATCH_SIZE))),

epochs = ~~100~~ EPOCHS,

validation_data = val_data_gen,

validation_steps = int(np.ceil(total_val / float(BATCH_SIZE))))

)

5.5) Eğitim Sonuçlarının Görüleştirilmesi

acc = history.history['accuracy']

val_acc = history.history['val_accuracy']

loss = history.history['loss']

val_loss = history.history['val_loss']

epochs_range = range(EPOCHS)

plt.figure(figsize=(8, 8))

plt.subplot(1, 2, 1)

plt.plot(epochs_range, acc, label='Train Acc')

plt.plot(epochs_range, val_acc, label='Val Acc')

plt.legend(loc='lower right')

plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)

plt.plot(epochs_range, loss, label='Train Loss')

plt.plot(epochs_range, val_loss, label='Val Loss')

plt.legend(loc='upper right')

plt.title('Train and Val. Loss')

plt.savefig('foo.png')

plt.show()

Oluşan grafiklere bakıldığında, eğitim ve doğrulama doğruluğu birek bir farklılıkta kapatıldır ve modelimiz doğrulama setinde yalnızca %70 doğruluk elde etmektedir.

Bu overfitting'in ağır bir göstergesidir. Eğitim ve doğrulama epelerde ayrılmaya başlandığında, model eğitim verilerini esterlemeye başlar ve doğrulama verilerinde işi performans göstermez.

Başlık 5.1'de modelin tanımlanması aşamasına tekrar bakarsak, son katmanımızda (sınıflandırıcı) katmandında, aşıgıda görüldüğü gibi 2 çıktı birimi ve bir 'softmax' aktivasyonu olan bir 'Dense' (yapın) katmandan oluştuğuna dikkat edin;

tf.keras.layers.Dense(2, activation='softmax')

İki sınıflandırma problemlerinde gallerken bir başka popüler yaklaşım, aşıgıda görüldüğü gibi, 1 çıktı birimi ve bir 'sigmoid' aktivasyonu fonk. ile 'Dense' katmandan oluşan bir sınıflandırıcı kullanmaktadır;

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

Bu iki seçenekten herhangi biri iki sınıflandırma probleminde işe yarayacaktır. Aşağıda, sınıflandırıcı olarak 'sigmoid' aktivasyonu ile kullanılmaya karar verirseniz, 'model.compile()' fonksiyonundaki kayip parametresini (loss) değiştirmeniz gerekeceğini unutmayın. Aşağıda görebileceğiniz gibi 'sparse_categorical_crossentropy' den 'binary_crossentropy' e;

model.compile(optimizer='adam',

loss='binary_crossentropy',

metrics=['accuracy'])

GÖRÜNTÜ BüYÜKTME DE KEDİLER VE KÖPEKLER SINIFLANDIRILMASI

Bu bölüm altındaki bölüm 2'de satırı önceki örnek ile aynı şekilde işlemektedir. Bu yüzden sadece farklılık olan kısımlara değinilecek olterkenmiştir. Model parametrelerini ayarlaması aşamasından sonra "Veri Büyütme" işlemine geçilir.

1) VERİ BüYÜKTME

"Overfitting" durumu genelde veri setinin at olduğu durumlarda kare kareye gelir. Bu sorunu çözmek bir yolu, veri setini yeterli sayıda ve çeşitli eğitim örneklerine sahip olacak şekilde genişletmektir. Veri büyütme, inanılır görünümler veren rastgele dönüşümler yoluyla örnek üretebilir, mevcut eğitim örneklerinden daha fazla eğitim verisi üretme yaklaşımını benimsenir. Amaç, eğitim sırasında modelin aynı resmi asta itki kazanmamasıdır. "tf-hubs" ta bunu daha önce kullanan ImageDataGenerator sınıfı kullanarak uygulanabilir. İstediğiniz farklı dönüşümleri bir argüman biçiminde olarak veri ülmenizde kullanabilirsiniz ve eğitim süreci sırasında veri ekmesine uygulanabilir.

Başlamadan önce, şereklereki deşifrelik türünü görebilmek için bir resmi görsüntüleyen fak. tanıyalayalım.

```
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20, 20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.show()
```

1.1- Resmi Yatay Olarak Çevirme

Veri setimize rastgele yatay çevirme uygulayarak ve dönüşüm sonrası birleşik görsüntülerin nasıl görüneceğini gözerek başlayabiliriz. Bunu "ImageDataGenerator" sınıfına argüman olarak "horizontal_flip=True" ileterek elde edilir.

```
image_gen = ImageDataGenerator(rescale=1./255, horizontal_flip=True)
train_data_gen = image_gen.flow_from_directory(batch_size=BATCH_SIZE,
                                                directory=train_dir,
                                                shuffle=True,
                                                target_size=(IMG_SHAPE, IMG_SHAPE))
```

Döndürmiş görünmek için eğitim setimizden bir örnek resim alıp kez kez tekrarlayalım. Arttırma, her tetrara rastgele uygulanacaktır (veya uygulanmayacaktır).

```
augmented_images = [train_data_gen[0][0][0] for i in range(5)]
plotImages(augmented_images)
```

2.1- Görüntü Dönüştürme

Dönüştürerek, verisini büyütme göreviyle birlikte bir derecede rastgele döndürer - 45° olsun

```
image_gen = ImageDataGenerator(rescale=1./255, rotation_range=45)
train_data_gen = image_gen.flow_from_directory(batch_size=BATCH_SIZE,
                                                directory=train_dir,
                                                shuffle=True,
                                                target_size=(IMG_SHAPE, IMG_SHAPE))
```

Eğitim setimizde nasıl çalıştığını bakalım:

```
augmented_images = [train_data_gen[0][i][0] for i in range]
plotImages(augmented_images)
```

1.3) Yatınlastırma Uygulamaları

Ayrıca, görselleri rastgele %50'ye kadar yatınlastırarak veri türünün sayısını artıracaktır.

image_gen = ImageDataGenerator(rescale=1./255, zoom_range=0.5)

train_data_gen = image_gen.flow_from_directory(batch_size=BATCH_SIZE,

directory=train_dir,

shuffle=True,

target_size=(IMG_SHAPE, IMG_SHAPE))

Son ket yapılıcak denetimlikli görsel eğitimi;

augmented_images = [train_data_gen[i][0] for i in range(5)]

plotImages(augmented_images)

1.4) Hepsini Bir Araya Getirme

Tüm bu büyütümleri (veri), uygun değerlere sahip argümanlar olarak ilererek tek bir kod satırı ile uygulayabiliriz.

Buradı, eğitimin görsellerimizde yeniden ölçütlenmesi, 45° döndürme, dikdörtgen kaydırma, yatay kaydırma, yatay çevirme ve yatınlastırma ile veri setimizi oluşturuyoruz.

image_gen_train = 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')

train_data_gen = image_gen_train.flow_from_directory(batch_size=BATCH_SIZE,
directory=train_dir,
shuffle=True,
target_size=(IMG_SHAPE, IMG_SHAPE),
class_mode='binary')

Bu ayarlamaları rastgele veri setimizde aktardığımızda, tek bir görseli bir farklı zamanda

nasıl değiştireceğiz bakalım.

augmented_images = [train_data_gen[i][0] for i in range(5)]

plotImages(augmented_images)

2) DROPOUT Ekleme

Modeli tanımlama aşamasında katmanları tanımlarken Flatten() katmanından hemen önce ve son MaxPooling katmanından hemen sonra "Dropout()" katmanı ekliyoruz.

son yoğun(dense) katmanlarından önce 0.5'lik birakma olasılığına sahip 'Dropout' katmanı ekledik. Dropout katmanı gelen değerlerin %50'sinin sıfır'a ayarlanacağı anlamına gelir.

Bu durumda 'overfitting'i önlemeye yardımcı olur.

model = tf.keras.models.Sequential([

...
 tf.keras.layers.Dropout(0.5),

...
])

Yaptığınız bu değişikler sayesinde verisetimizi artırmış olduk böylelikle 'overfitting' yani aksı uygunun önne geçtiğ. Eğitilme ve Eğitim veri setimizin 'accuracy' (değrinlik) skorunda gözle püskür fark elde edilmiş oldu.



Bu ana kader osini uyuman dinde geometri icin bu teknigde bolktik:

- Early Stopping (Erken Durdurma) = Bu yontemde, egitim deamasi sirasinda dovrulama setindeki kaybi takip eder ve bunu, modelin dovrus oldugu ve fazla uymadigi surece egitimin devam etmesine imkan tanmasidir.
- Image Augmentation (Veriseti bilyetme) = Egitim setindeki nevut goruntuler'e rastgele goruntuler deneysimleri eylemlerden egitilecek egitim setinizdeki goruntuler sayisini yapay olarak artirmak.
- Dropout (Bırakma) = Egitim sirasinda bir sinir apindaki sabit sayida nöronun rastgele kaldırılması. Difer faktörlerini medium.com'dan bakabilirsiniz.

"tf.keras" KULLANARAK GORUNTÜ SINIFLANDIRMA

Bu baslik altında gerek resimlerini siniflandiracagiz. 'tf.keras.Sequential' modelini kullanarak bir goruntü siniflandirici olusturacak ve 'tf.keras.preprocessing.image.ImageDataGenerator' kullanarak veri getireceksiniz.

1) PAKETLERI IGE AKTARMA

Gerekli paketleri içe aktararak baslayalim. 'os' paketi, dosyaları ve dizin yapısını okumak için kullanılır. numpy, python listesini numpy listesini dönüştürmek ve gerekli matris ielmlerini gerçekleştirmek için kullanılır. 'matplotlib.pyplot', grafigi ölçmek ve egitim ve dovrulama verimizdeki goruntülerde bakmak için kullanılır.

```
import os
import numpy as np
import glob
import shutil
import matplotlib.pyplot as plt
```

1.1) CNN'ini olusturmak için kullandığımız Tensorflow ve Keras katmanlarını içe aktardımlım. Ayrıca veri setimizde bilyetme işlemini uygulamak için ImageDataGenerator', Keras'tan içe aktardım.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

2) VERILECI YÜKLEME

Goruntü siniflandiricimizi olusturmak için, gerekli veri setini indirerek baslayabiliriz. Once-likle veri setinin orjinal versiyonunu indirmemiz gereklidir ve indirme işleminden sonra 'tmp' dizinine kaydediyoruz. Veri setini indirdikten sonra onu okarmamız gereklidir.

```
-URL = "https://storage.googleapis.com/download.tensorflow.org/example-images/flower_photos.tgz"
zip_file = tf.keras.utils.get_file(origin = -URL,
                                    fname = "flower_photos.tgz",
                                    extract = True)
base_dir = os.path.join(os.path.dirname(zip_file), 'flower_photos')
```

Indirdigimiz veri seti 5 cift gerekten olusan gorunteller iceriyor. Gul(Rose), Papatyasi(Daisy), Karahindibasi(Dandelion), Ayçiçesi(Sunflowers), Lale(Tulips)

```
classes = ['roses', 'daisy', 'dandelion', 'sunflowers', 'tulips']
```

Ayrıca indirdiğiniz veri seti aşağıdaki dizin yapısına sahiptir.

flower_photos

|_ daisy

|_ dandelion

|_ roses

|_ sunflowers

|_ tulips

|_ train

|_ — daisy ~ [1,528]

:

:

|_ val

|_ — daisy: [507-JPGs, .jpg]

:

:

Görüldüğü gibi eğitim ve doğrulama verilerimizi içeren kod yok.

Bu nedenle kendimizini oluşturuyoruz. Bu yaparak kod lazım.

Aşağıdaki kod, her biri 5 klasör (her sınıf %5'ini bir tane) içeren bir train (eğitim) ve bir val (doğrulama) klasörü oluşturur. Daha sonra görüntülerin %80'i eğitim setine ve %20'si doğrulama setine gidecek şekilde, görüntülerin orijinali klasörden bu yeni klasöre taşır. Sonunda dizin esetilde olacaktır.

Original klasörleri silmediğiniz için hala 'flower_photos' dizininde olacaklar. Aşağıdaki kod her bir sınıf tane için sahip olduğunuz toplam sınıfı resmini gösterir.

for cl in classes:

 img_path = os.path.join(base_dir, cl)

 images = glob.glob(img_path + '/*.JPG')

 print(f'{cl}: {len(images)} Images' .format(cl, len(images)))

 num_train = int(round(len(images) * 0.8))

 train, val = images[:num_train], images[num_train:]

for t in train:

 if not os.path.exists(os.path.join(base_dir, 'train', cl)):

 os.makedirs(os.path.join(base_dir, 'train', cl))

 shutil.move(t, os.path.join(base_dir, 'train', cl))

for v in val:

 if not os.path.exists(os.path.join(base_dir, 'val', cl)):

 os.makedirs(os.path.join(base_dir, 'val', cl))

 shutil.move(v, os.path.join(base_dir, 'val', cl))

round(len(images) * 0.8) # 639

Kotayık olması için eğitim ve doğrulama kümelerinin yolunu oluşturuyoruz.

train_dir = os.path.join(base_dir, 'train')

val_dir = os.path.join(base_dir, 'val')

}
roses = 641 images
daisy = 633 images
dandelion = 893 images
sunflowers = 699 images
tulips = 490 images

3) VERİ SETİ BüYÜTME

3.1) Geçitli Veri Seti (Görüntü) Dönüştürmelerini Deneyin

Bu bölümde bazı temel görsel dönüşümleri yaparak pratik yapacağınız dönüştürmeleri yapmadan önce 'batch-size' ve görsel boyutunu tanımlayalım. CNN'imize girdinin aynı boyuttaki görüntüler olduğunu unutmayın. Bu nedenle veri kümemizdeki görüntülerin aynı boyutta yeniden boyutlandırılacak.

batch_size = 100

IMG_SHAPE = 130

3.2) Rastgele Yakalayıcı Geçirme Uygulama

```
image_gen = ImageDataGenerator(rescale=1./255, horizontal_flip=True)
train_data_gen = image_gen.flow_from_directory(batch_size=batch_size,
                                                directory=train_dir,
                                                shuffle=True,
                                                target_size=(IMG_SHAPE,IMG_SHAPE))

# 5 sınıfın toplam 2935 fotoğraf var.
```

Eğitim dönerlerinden 1 örnek görüntüsünü alıp 5 kez tekrarlayalım ki böylece aynı görüntüğe 5 kez uygulayabilem, böylelikle deşifrelerini eylem halinde görebilim.

```
def plotImages(images_arr):
    fig, axes = plt.subplots(1,5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.show()
```

```
augmented_images = [train_data_gen[0][0][0] for i in range(5)]
plotImages(augmented_images)
```

3.3) Rastgele Dönüşüm Uygulama

```
image_gen = ImageDataGenerator(rescale=1./255, rotation_range=45)
train_data_gen = image_gen.flow_from_directory(batch_size=batch_size,
                                                directory=train_dir,
                                                shuffle=True,
                                                target_size=(IMG_SHAPE,IMG_SHAPE))

# Görüntünün değişimlerini gözlemezsiniz!
```

```
augmented_images = [train_data_gen[0][0][0] for i in range(5)]
plotImages(augmented_images)
```

3.4) Rastgele Yakınlaştırma Uygulama

```
image_gen = ImageDataGenerator(rescale=1./255, zoom_range=0.5)
train_data_gen = image_gen.flow_from_directory(batch_size=batch_size,
                                                directory=train_dir,
                                                shuffle=True,
                                                target_size=(IMG_SHAPE,IMG_SHAPE))

augmented_images = [train_data_gen[0][0][0] for i in range(5)]
plotImages(augmented_images)
```

3.5) Hepsi Bir Araya Getirme

```
image_gen_train = ImageDataGenerator(rescale=1./255, rotation_range=45,
                                      width_shift_range=.15, height_shift_range=.15,
                                      horizontal_flip=True, zoom_range=0.5)

train_data_gen = image_gen_train.flow_from_directory(batch_size=batch_size,
                                                    directory=train_dir, shuffle=True,
                                                    target_size=(IMG_SHAPE,IMG_SHAPE),
                                                    class_mode='sparse')
```

Bu veri boyutlarını rastgele veriselimizde ortaklaşımına göre, tek bir görüntünün 5 farklı kez nasıl görüneceğine bakalım.

```
augmented_images = [train_data_gen[0][0][0] for i in range(5)]
plotImages(augmented_images)
```

3.6) Doğrulama Kümesi İçin Veri Oluşturma ve Oluşturma

Genel olarak, yalnızca eğitim örneklerimizde veri artırma yaparız. Bu nedenle aşağıdaki kod bloğunda göründüğü gibi 255 ile yeniden üretilen bir döngüm oluşturmak için 'Image DataGenerator' kullanırız.

```
image_gen_val = ImageDataGenerator(rescale=1./255)
```

```
val_data_gen = image_gen_val.flow_from_directory(batch_size=batch_size,
                                                 directory=val_dir,
                                                 target_size=(IMG_SHAPE,IMG_SHAPE),
                                                 class_mode='sparse')
```

4) CNN'in OLUSTURULMASI

Aşağıdaki kod satırlarında, 3 evrişim bloğundan oluşan bir evrişimsel sinir ağı oluşturulmuştur. Her evrişimsel blok, bir Conv2D katmanı ve ardından max. havuz katmanı içermiştir. İlk evrişim bloğu 16 filtreler ikincisi 32 ve üçüncü 64 filtreye sahiptir. Tüm evrişimsel filtreler 3×3 'dir. Tüm maksimum havuz katmanlarının pool-size $(2,2)$ 'dır.

3 evrişimli blok sonrasında, düzenlendirilen bir katmandan ve ardından 512 birimli tam bağlantılı bir katmana sahip olmaktadır. CNN, 'softmax' aktivasyon fonksiyonundan yapılan 5 sınıfı dğalı sınıf olasılıklarını vermektedir. Diğer tüm katmanlar bir yeniden etkinleştirme işlevi kullanmaktadır. Ayrıca uygun olduğunda $\%0.2$ olasılıkla bir katmana 'dropout' katmanı içermektedir.

```
model = Sequential()
```

```
[ model.add(Conv2D(16, 3, padding='same', activation='relu', input_shape=[2IMG_SHAPE, IMG_SHAPE, 3]))  
[ model.add(MaxPooling2D(pool_size=(2,2)))  
[ model.add(Conv2D(32, 3, padding='same', activation='relu'))  
[ model.add(MaxPooling2D(pool_size=(2,2)))  
[ model.add(Conv2D(64, 3, padding='same', activation='relu'))  
[ model.add(MaxPooling2D(pool_size=(2,2)))  
[ model.add(Flatten())  
[ model.add(Dropout(0.2))  
[ model.add(Dense(512, activation='relu'))  
[ model.add(Dropout(0.2))  
[ model.add(Dense(5))
```

5) MODELİN DERLENMESİ

Kayıp işlevi olarak segrek çapraz entropi işlevi olan 'ADAM' optimizörü kullanarak modeli derlefin. Aşağıda eğitirken her adıda (epoch) eğitim ve doğrulama durumuna bakmak istiyoruz. Bu nedenle metrics argümanı olarak 'accuracy' eklemiştik.

```
model.compile(optimizer='adam',
```

```
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
metrics=['accuracy'])
```

6) MODELİN EĞİTİMİ

```
epochs = 50
```

```
history = model.fit_generator(
```

```
train_data_gen,
```

```
steps_per_epoch=int(np.ceil(train_data_gen.n / float(batch_size))),
```

```
epochs=epochs,
```

```
validation_data=val_data_gen,
```

```
validation_steps=int(np.ceil(val_data_gen.n / float(batch_size))))
```

Bo aşağıda son 50 epocha) aşamalarına göre kayıp değerlerinin profili hem eğitim hem doğrulama seti için gözlebiliriz.



TENSORFLOW HUB AND TRANSFER LEARNING

Tensorflow Hub, kullanabileceğiniz önceden eğitilmiş tf modellerinin devrimiçi bir deposudur. Bu modeller ya doğrudan gibi kullanılrya da Transfer Öğrenimi için kullanılabilir.

Transfer Öğrenimi, mevcut bir eğitim modeli alıp ek iş yapmak için genişlettiğiniz bir yöntemdir. Bu farklı olası çıktılar elde etmek için son katmanları eklerken ve yeniden eğitirken modelin büyük kısmını değiştirmeden bırakmayı seçer.

A) IMPORT İŞLEMLERİ

Data Once ağına oldupnumuz bazı normal import işlemleri ve bununla beraber bu bağlılığı and konusu olan 'tensorflow-hub'u içe aktarıyoruz.

```
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_datasets as tfds
from tensorflow.keras import layers
```

```
import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
```

BÖLÜM 1 : Tahmin için Tensorflow Hub MobileNet Kullanımı

Bu bölümde, eğitilmiş bir model alınır ve kendi içe aktarılır ve denenmektedir.

A) Sınıflandırıcı İndirme

MobileNet modelini indirir ve sonrasında bir Keras modeli oluşturur. MobileNet, 3 renk kanalına (RGB) ve 224×224 piksellik görüntüler bekler.

```
CLASSIFIER_URL = 'https://tfhub.dev/google/tf2-preview/mobilenet_v2/classification/2'
IMAGE_RES = 224
```

```
model = tf.keras.Sequential([
    hub.KerasLayer(CLASSIFIER_URL, input_shape=(IMAGE_RES, IMAGE_RES))
```

B) TEK Bir Görüntü Üzerinde Batma

MobileNet, ImageNet veri seti üzerinde eğitilmiştir. ImageNet'in 1000 farklı objesi sınıfı vardır ve bu sınıflardan biri astarı bilinmekte. Modelin eğitiminde bulunmayan bir resim ile deneyelim.

```
import numpy as np
import PIL.Image as Image
grace_hopper = tf.keras.utils.get_file('image.jpg', 'https://storage.googleapis.com/tfhub-dev-public/testdata/grace_hopper/2.jpg')
grace_hopper = Image.open(grace_hopper).resize((IMAGE_RES, IMAGE_RES))
grace_hopper
```

```
grace_hopper = np.array(grace_hopper)/255.0
grace_hopper.shape # OUT: (224, 224, 3)
```

Unutmayın, modeller her zaman işlenecek bir dizi görüntü ister. Burada bir tane ifade ettiğimiz ve görüntüye tahmin için modele aktarıyoruz.

```
result = model.predict(np.expand_dims(grace_hopper, axis=0))
result.shape # OUT: (1, 1000)
```

Sonuç, görüntü için her sınıf olasılığını derecelendiren 1000 elementli bir vektördür.

```
predicted_class = np.argmax(result[0], axis=-1)
predicted_class # OUT = 653
```

ImageNet veri setinde tahmin edilen sınıfın ne olduğunu öğrenmek için ImageNet etiketlerini indirelim.

```
labels_path = tf.keras.utils.get_file('ImageNetLabels.txt', 'LABELS-TXT-FILE-URL')
```

```
imagenet_labels = np.array(open(labels_path).read().splitlines())
```

```
plt.imshow(grace_hopper)
```

```
plt.axis('off')
```

```
predicted_class_name = imagenet_labels[predicted_class]
```

```
_ = plt.title("Prediction: " + predicted_class_name.title())
```

Bölüm: 2 = Kediler ve Köpekler veri kümesi için Tensorflow Hub modeli Kullanımı
 Kedi ve köpek veri setini kullanmak için TF Datasets kullanabilir.
`(train-examples, validation-examples), info = tfds.load('cats-vs-dogs', with_info=True, as_supervised=True,`
`num-examples = info.splits['train'].num_examples`
`num-classes = info.features['label'].num_classes`

'Dogs vs Cats' veri setindeki resimlerin hepsi aynı boyutta değildir.

```
for i, example_image in enumerate(train-examples.take(3)):
    print("Image {} shape: {}".format(i+1, example_image[0].shape))
```

Bu yüzden tüm görselleri MobileNet (224, 224) tarafından beklenen格式e yeri-
 den bölmektedir memiz gerekiyor.

```
def format_image(image, label):
    image = tf.image.resize(image, (IMAGE_RES, IMAGE_RES)) / 255.0
    return image, label
```

BATCH_SIZE = 32

```
train_batches = train-examples.shuffle(num-examples // 4).map(format-image).batch(BATCH_SIZE).prefetch(1)
```

```
validation_batches = validation-examples.map(format-image).batch(BATCH_SIZE).prefetch(1)
```

A) Sınıflandırma Bir Grup Görüntü Üzerinde Hypotanması

'Model' nesnemiz hala ImageNet üzerinde eğitilmiş tam MobileNet modelidir. Bu nedenle
 1000 sınıfı耿ti sınıfı vardır. ImageNet'in içinde çok rágda köpek ve kedi var.

image_batch, label_batch = next(iter(train_batches.take(1)))

image_batch = image_batch.numpy()

label_batch = label_batch.numpy()

result_batch = model.predict(image_batch)

predicted_class_names = Inagnet_labels[np.argmax(result_batch, axis=1)]

predicted_class_names

Etketler, Köpekler ve Kediler ile eşleştiriyor. Şimdi 'dogs vs cats' veri setindeki gör-
 üntükleri sizelim ve ImageNet etiketlerinin yanlarına koymak.

```
plt.figure(figsize=(10,9))
for n in range(30):
    plt.subplot(6, 5, n+1)
    plt.subplots_adjust(hspace=0.3)
    plt.imshow(image_batch[n])
    plt.title(predicted_class_names[n])
    plt.axis('off')
    plt.subtitle("ImageNet predictions")
```

Bölüm: 3: TF Hub ile Basit Transfer Öğrenmesi

Sıradan aktarım (transfer) öğrenimi yapmak için TensorFlow Hub'ı kullanırsınız. Transfer öğrenimi ile önceki eğitilmiş bir modelin parçalarını yeniden kullanırız ve modelin son katmanını veya birkaç katmanını değiştirmiz ve ardından bu katmanları kendi veri kümemizde yeniden eğitiriz.

Tensorflow Hub, tamamlandırmış modellere elbette, modellenin son sınıflandırma katmanı olmadan da sunar. Bunlar transfer öğrenmeyi kolayca yapmak için kullanılabilir. MobileNet V2 ile devam edelim.

Ayrıca 'Dogs vs Cats' veri seti ile devam edeceğiz böylelikle bu modelin performansı önceki sınıflandırma ile karşılayabiliriz.

Tensorflow Hub'daki kümeli modeli (son sınıflandırma katmanı olmadan) bir 'feature-extractor' olarak adlandırıldığını unutmayalım. Bu terimin nedeni, girdiği bir dizi özellik içeren bir katmana kadar götürerek olmasıdır. Bu nedenle, nihai olasılık dağılımını oluşturmak gereklidir, bir görsüntün içeriğini belirkende işin büyük kısmını yaptı. Yani görsüntün özelliklerini öğrenmiştir.

URL = "URL-COME-HERE"

feature-extractor = hub.KerasLayer (URL, input_shape = (IMAGE_SIZE, IMAGE_SIZE))
Bunun üzerinde bir dizi görsüntü okutturulur ve son haline bakılır. 3)

32 = görsüntü sayısı ve 1280 = TF Hub'dan alınan kümeli modelin son katmanının sayısı.

feature-batch = feature-extractor (image-batch)

print (feature-batch.shape)

Özellik sırası katmandaki değişkenleri doldurulur, böylece eğitim yalnızca son sınıflandırıcı katman yapar.

feature-extractor.trainable = False

> Şimdi hub katmanını bir tf.keras.Sequential modeli ile saralırsınız ve yeni bir katman ekleyelim.
model = tf.keras.Sequential ([
 feature-extractor,
 layers.Dense (12)
])

model.summary()

Şimdi modeli derleme ve sonrasında "fit" ile eğitelim.

model.compile (optimizer = 'adam',

loss = tf.keras.losses.SparseCategoricalCrossentropy (from_logits = True),
metrics = ['accuracy'])

EPOCHS = 6

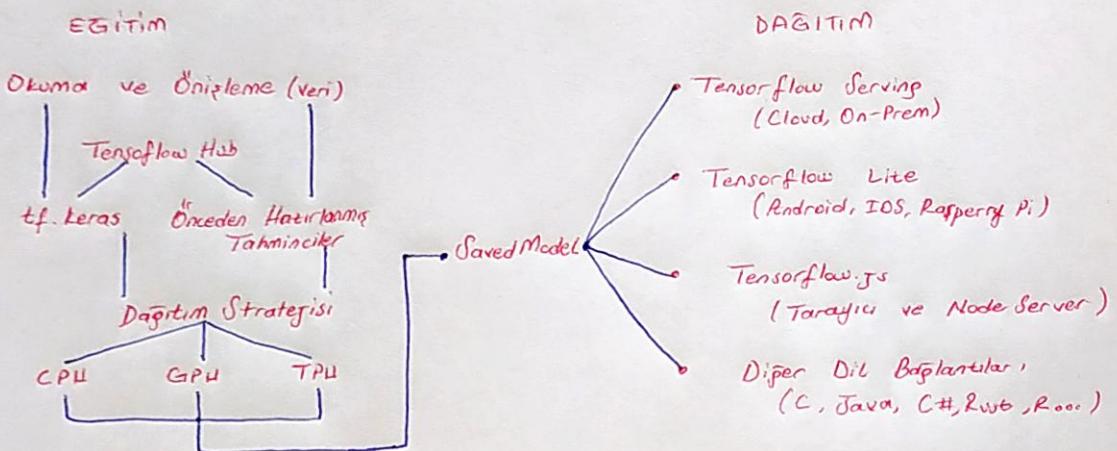
history = model.fit (train-batches,
epochs = EPOCHS,
validation_data = validation-batches)

Yaklaşık %97 doğruluk elde ettiğimizi görebiliyoruz. Dediğimiz gibi basitlerde olusturulan %83 doğruluk elde edilen modelde göre çok iyi bir seviye. Bu farklılık nedeni - MobileNet'in uzantıları tarafındanrawn bir süre boyunca örtelenen tasarılmış ve daha sonra büyük bir veri kümesi ~~taraftan~~ tara olaan (ImageNet) üzerinde eğitilmiş olmasıdır.

Bununla beraber doğrulama performansının yükseltme (eğitim) basından sonra kadar eğitim performansından iyi olmasıdır. Bunu bir nedeni, doğrulama performansının diğer (epoch) sonunda düşmesi, ancak eğitim performansının diğer boyunca ort. değerler olmazdır.

Anıktır, daha büyük neden köpekler ve kediler görsüntülerini üzerinde zaten eğitilmiş olan MobileNet'in büyük bir boyutlu yeniden kullanılmıştır. Eğitim yaparken, model hala eğitim görsüntülerinde veri kullanmayı gerçekleştirir, ancak doğrulama veri kümesinde yapmaz. Bu, eğitim görsüntülerini sınıflandırmasında daha zor olabileceğini anlamına gelir.

CHAPTER 8 & 7 : MODELLERİ KAYDETME ve YÜKLEME



2) MODELLERİ KAYDETME ve YÜKLEME

Bu bölüm altında eğitilmiş modelin kaydedilmesi, eğitime devam etmek için nasıl peri yükleneceğini resim yapmak için nasıl kullanıldığını öğreneceğiz. Önceki bölüm larda olduğu 'Dogs vs Cats' veri seti kullanılacak. Sonra eğitim verilerini kerasın kullandığı bir format olan HDF5 dosyası olarak kaydedeceğiz. Modeli peri yükleyip kullandıktan sonra TF SavedModel olarak kaydedeceğiz ve daha sonra diğer platformlara dağıtmaya uygun şekilde dize indirelim.

Bölümle başlamadan önce aktarma ve indirme işlemlerine bakalım.

```

! pip install -U tensorflow-hub
! pip install -U tensorflow-datasets
import time
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_datasets as tfds
tfds.disable_progress_bar()
from tensorflow.keras import layers
  
```

3) BÖLÜM=1 = Kediler ve Köpekler veri Kümesinin Yüklenmesi

```

(train_examples, validation_examples), info = tfds.load('cats-vs-dogs',
                                                       split=['train[:80%]', 'train[80%:]'],
                                                       with_info=True,
                                                       as_supervised=True,
  
```

Veri setindeki resimlerin hepsi aynı boyutta depildir. Bu nedenle tüm görsüntülerini mobilede tarafından beklenen gibi 224x224 (224,224) yeniden boyutlandırmamız gereklidir.

```

def format_image(image, label):
    image = tf.image.resize(image, (IMAGE_RES, IMAGE_RES)) / 255.0
    return image, label
num_examples = info.splits['train'].num_examples
BATCH_SIZE = 32
IMAGE_RES = 224
train_batches = train_examples.cache().shuffle(num_examples // 4).map(format_image).batch(BATCH_SIZE).prefetch(1)
validation_batches = validation_examples.cache().map(format_image).batch(BATCH_SIZE).prefetch(1)
  
```

BÖLÜM 2 = Tensorflow Hub ile Transferi Öğrenme
 URL = "TF-HUB-URL-COME-HERE"
 feature_extractor = hub.KerasLayers(URL, input_shape = (IMAGE_RES, IMAGE_RES, 3))

 # Seçili çıktı katmanındaki değişkenleri dondur
 feature_extractor.trainable = False
 Sırada hub katmanını bir 'tf.keras.Sequential' modeline saran ve yeni bir sınıflandırma kat.
 model = tf.keras.Sequential([feature_extractor, layers.Dense(2)])
 model.summary()
 Modeli önce derleyip sonrasında 'fit' ile eğitin.
 model.compile(optimizer = 'adam',
 loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True),
 metrics = ['accuracy'])
 EPOCHS = 3
 history = model.fit(train_batches,
 epochs = EPOCHS,
 validation_data = validation_batches)

 Sınıf adlarını alalım;
 class_names = np.array(info.features['label'].names) # ve yazdır - print
 Model üzerinde bir toplu gürültü işlemi gerçekleştirelim, dizinleri sınıf adlarına dengeleştirelim.
 image_batch, label_batch = next(iter(train_batches.take(1)))
 image_batch = image_batch.numpy()
 label_batch = label_batch.numpy()
 predicted_batch = model.predict(image_batch)
 predicted_batch = tf.squeeze(predicted_batch).numpy()
 predicted_ids = np.argmax(predicted_batch, axis=-1)
 predicted_class_names = class_names[predicted_ids] # ve 'print'

BÖLÜM 3 = Keras'ı .h5 modeli olarak kaydet
 Modeli eğittiğinden sonra onu Keras tarafından kullanılan bir format HDF5 dosyasında kaydedelim.
 t = time.time()
 export_path_keras = "./{0}.h5".format(int(t))
 print(export_path_keras)
 model.save(export_path_keras)

 Bu dosya, model mimarisi, modelin yapılık değerleri, varsa modelin eğitim yapılması, optimizasyonu ve durumu.

BÖLÜM 4 = Keras .h5 Modelinin Yüklentimesi
 Dosya yolunu ve custom_objects parametresini yapılamamıştır. Bu parametre, Keras'a
 aktarılmış ögrenmesi için kullanılmış 'feature_extractor'dan hub.KerasLayer'in nasıl yüklenmesi.
 reloaded = tf.keras.models.load_model(export_path_keras,
 custom_objects = {'KerasLayer': hub.KerasLayer})
 reloaded.summary()
 Yeniden yüklenen ve önceki modelin aynı sonucu verdigini kontrol edebiliriz.
 result_batch = model.predict(image_batch)
 reloaded_result_batch = reloaded.predict(image_batch)
 Fark sıfır (0) olmalıdır
 (abs(result_batch - reloaded_result_batch)).max() # OUT = 0
 Modeli eğitmeye devam etmekten 'fit' kullanalım.
 history = reloaded.fit(train_batches, epochs=3, validation_data=validation_batches)

Bölüm 5 = SavedModel Olarak Dosya Aktarılması

Oluşturdukları 'SavedModel' dosyaları, model ağırlıklarını içeren bir TF kontrol noktası, temeldeki TF grafiğini içeren bir 'SavedModel' protokolü. Tahmin, eğitim ve değerlendirme için ayrı grafikler kaydedilir, mimari yapılmamıştır. Original modelinizi kaydedelim. Kaydetme fonksiyonu bir varlıklar (assets) klasörü, bir değişkenler klasörü (variables) ve 'save-model.pb' dosyasını içeren bir klasör oluşturur.

`t = time.time()`

`export_path_sm = "-/{}"-format(int(t))`

`tf.saved_model.save(model, export_path_sm)`

Bölüm 6 = SavedModel'in Yüklenmesi

Şimdi savedModelimizi yükleyelim ve tahminler yapmak için kullanalım.

`reloaded_sm = tf.saved_model.load(export_path_sm)`

Şimdi de yüklenen modele tahminler yapalım ve sonra sonuçlar arasında fark var mı bakalım.

`reloaded_sm_result_batch = reloaded_sm(image_batch, training=False).numpy()`

`(db(np.result_batch - reloaded_sm_result_batch)).max() # OUT = 0.0`

Bölüm 7 = SavedModel'in Keras Modeli gibi Yüklenmesi

! 'tf.saved_model.load' tarafından oluşturulan nesne bir keras nesnesi değil (yani fit, predict gibi yöntem yok). Tf SavedModel'ini tam bir keras modeli olarak port etmek için birkaç bir yöntem kullanırız.

`reloaded_sm_keras = tf.keras.models.load_model(export_path_sm,`

`custom_objects = {'KerasLayer': hub.KerasLayer})`

`reloaded_sm_keras.summary()`

Original model ile farklına baktığımızda sonuc yine sıfır olacaktır.

Bölüm 8 = Modelin İndirilmesi

Bir .zip dosyası olarak oluşturulan modeli genel diskimize indirebilirsiniz. Tüm alt klasörleri sıkılaştırın ıgin -r (recursive) seçenekini kullanırı.

`!zip -r model.zip {export-path-sm}`

Dosya sıkıştırıldıktan sonra indirebilirsiniz.

`try:`

`from google.colab import files`

`files.download('model.zip')`

`except ImportError:`

`pass`

BÖLÜM 8: ZAMAN SERİLERİ

A) YAYGIN KALİPLER

```
import numpy as np  
import matplotlib.pyplot as plt  
Şimdi zaman serilerini göstermek için bir işlev tanımlayalım.  
def plot_series(time, series, format="-", start=0, end=None, label=None):  
    plt.plot(time[start:end], series[start:end], format, label=label)  
    plt.xlabel("Time")  
    plt.ylabel("Value")  
    if label:  
        plt.legend(fontsize=14)  
    plt.grid(True)
```

A.1) Trend ve Mevsimsellik

Jadece yukarı doğru giden bir zaman serisi oluşturalım:

time = np.arange(4 * 365 + 1)

baseline = 10

series = baseline + trend(time, 0.1)

plt.figure(figsize=(10, 6))

plot_series(time, series)

plt.show()

Enesind trend işlevi tanımlanmalı,
def trend(time, slope=0):
 return slope * time

Böylelikle 0. zaman değeri 10 olan ve her bir
dönüm zamanda 0.1 artan trend zaman serisi elde
etmiş oluruz.

Bu işlevlere benzer şekilde mevsimsel gürültü ve bantların bir arada (ayrınlarda)
grafiklenmesi colab-8'de bulunuyor.

NAIVE FORECASTING (Saf Tahmin)

Bu basit altında basit bir tahmini ekleyelim. Bunu yaparken bir önceki colab dosyasındaki
serileri gösterme, trend, mevsimsellik ve periyodik fonksiyonlarını bir arada kullanarak bir
zaman serisi elde edip sonrasında serininin bir kısmını eğitim ve kalan kısmını doğrulama
olarak ayıriz. Bakalım! İlk olarak bir zaman serisi oluşturalım:

slope = 0.05

baseline = 10

amplitude = 40

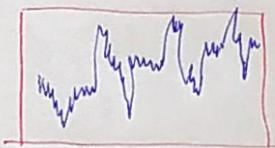
series = baseline + trend(time, slope) + seasonality(time, period=365, amplitude=amplitude)

noise_level = 5

noise = white_noise(time, noise_level, seed=42)

series += noise

plt.figure(figsize=(10, 6))
plot_series(time, series) + plt.show()



Bu grafifi (zaman serisini) iki döneme ayıralım eğitim ve doğrulama - Bu bölünme zamanı = 1000

split_time = 1000

time_train = time[:split_time]

x_train = series[:split_time]

time_valid = time[split_time:]

x_valid = series[split_time:]

naive_forecast = series[split_time-1:-1]

plt.figure(figsize=(10, 6))

plot_series(time_valid, x_valid, label='Series')

plot_series(time_valid, naive_forecast, label='Forecast')

Naif tahmin zaman serisinin bir adım
periyodunda kalıyor. Hata
oranını bakalım.

errors = naive_forecast - x_valid

abs_errors = np.abs(errors)

mae = abs_errors.mean()

MAE = 5, 93...



HAREKETLİ ORTALAMA

Bu bölüm altında önceki bölüm altında saf tahmin yaptığından zaman serisini kullandık. Bu yüzden 'plot-series', 'trend', 'seasonalpattern', 'seasonality' ve 'white-noise' işlevlerini tekrar kullanmak. Buna ek olarak bir tf işlevini kullanmadan perekcepti için import ediyoruz.

```
import tensorflow as tf
```

```
keras = tf.keras
```

Trend, mevsimsellik ve periyodik içeren zaman serisini tekrar oluşturuyor. Naive forecast sonucu mae oranının (hatamı) ≈ 5.93 çıktı. Batalım daha iyi bir hale gelecek mi?

```
def moving-average-forecast(series, window-size):
```

```
forecast = []
```

```
for time in range(len(series) - window-size):
```

```
    forecast.append(series[time:time + window-size].mean())
```

```
return np.array(forecast)
```

Yukarıda tanımlı fonk. son birkaç değerin ortalaması tahmin eder. window-size != 1 olursa durumda ise saf değer eğit olur. Aşağıda tanımlı olan fonk. aynı işlevi yerine getirir ama çok daha hızlı bir şekilde.

```
def moving-average-forecast(series, window-size):
```

```
mov = np.cumsum(series)
```

```
mov[window-size:] = mov[window-size:] - mov[: -window-size]
```

```
return mov[window-size - 1:-1] / window-size
```

Şimdi tahmin için son 30 güne bakarak olan bir tahmin oluşturuyor (yukarıdaki fonk. ile)

```
moving-avg = moving-average-forecast(series, 30)[split-time - 30:]
```

İşte çizelim

```
plt.figure(figsize=(10, 6))
```

```
plot-series(time-valid, x-valid, label="series")
```

```
plot-series(time-valid, moving-avg, label="Moving average")
```

Skor

```
keras.metrics.mean_absolute_error(x-valid, moving-avg).numpy() # OUT = 7.14
```

Bu değer saf tahminden kötü durumda ve bir pacitme olduğunu söyleyebilir. Hareketli ortalama, trendi ve mevsimselliliği düşüremez. Bunu farkları kullanarak ortadan kaldırıralım. Mevsimsellik döneni 365 gün olduğu için, t zamanındaki değerden t-365 döndeki değerini çıkar.

```
diff-series = (series[365:]) - series[:-365]
```

```
diff-time = time[365:]
```

Bu işlemlerden sonra doğrulama dönenime odaklanırız;

```
plt.figure(figsize=(10, 6))
```

```
plot-series(time-valid - diff-series[split-time - 365:], label="Series(t) - Series(t-365)")
```

```
plt.show()
```

Oluşan grafikte baktığımızda trend ve mevsimselliliğin yok olduğunu görebiliyoruz. Şimdi hareketli ortalamayı kullanıralım.

```
diff-moving-avg = moving-average-forecast(diff-series, 50)[split-time - 365 - 50:]
```

'diff-moving-avg' grafik haline getirdiğimizde ortalamaya yakın değerler gösterdiğiğini söyleyebiliriz.

Şimdi t-365 arasındaki geçmiş değerleri ekleyip trend ve mevsimselliliği peri getirelim.

```
diff-moving-avg-plus-past = series[split-time - 365:-365] + diff-moving-avg
```

```
keras.metrics.mean_absolute_error(x-valid, diff-moving-avg-plus-past).numpy() # OUT = 5.83
```

Saf tahminden biraz daha iyi. Ama tek tahliller biraz rastgele görünüyor, çünkü sadece genellikle olan değerleri okuyor. Gerçekliğin bir kısmını ortadan kaldırırmak için geçmiş değerler üzerinde hareketli ortalamamı kullanıralım.

```
diff-moving-avg-plus-smooth-past = moving-average-forecast(series[split-time - 370:-358], 11) + diff-moving-avg
```

```
keras.metrics.mean_absolute_error(x-valid, diff-moving-avg-plus-smooth-past).numpy() # OUT = 4.56
```



ZAMAN PENCERELERİ (TIME WINDOWS)

Bu başlık altında bir zaman serisini, ml modellerini eğitmek için kullanabileceğiniz bir bigimle nasıl döngü ettiğinizi öğrenmek isteyebilirsiniz. Bu bölümde TensorFlow'dan faydalanaçapı.

İlk olarak sıfırdan önceki kadar tam sayılar içeren bir veri kümlesi oluşturalım.

```
import tensorflow as tf
dataset = tf.data.Dataset.range(10) } Toplam 10 tane tensor oluşturur ve anumpy()
for val in dataset: } işlevi ile bu tensorlerin sadece değini
    print(val.numpy()) alırız.
```

Sırada window metodunu çağırınca var window boyutu olarak 5 döndürür, kaydırma=1 shift (kaydırma) her pencere (window), önceki pencereye kıyasta bir zaman dilimi kaydırılır. end = " " oynesinde bir pencerenin tüm elementleri yazılmadan sonraki satırda geçmez.

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1)
for window in dataset:
    for val in window:
        print(val.numpy(), end=" ")
    print()
```

} Kod satırı içinde son pencerein diğer pencelerden kısa olduğunu fark ederit sebebi = zaman serisinin sonuna gelmemiştir. ml modelleri genelde sabit sayıda girdi bekleyeceğii için bu durumu dikkatliyim (drop_remainder met parametresi)

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
for window in dataset:
    for val in window:
        print(val.numpy(), end=" ")
    print()
```

Şuan da kadar oldukça iyi ama sahip olmak istediğiniz gey dikenli tensorler bigiminde veri yığınları içeren tek bir veri kümlesi 'flat_map' metodunu bize yardımcı olacaktır. Bir önceki kod bloğundan hemen önce aşağıdaki satırı ekleyelim.

```
dataset = dataset.flat_map(lambda window: window.batch(5)) } Eski for döngülerini silmemi unutmayalım!
for window in dataset:
    print(window.numpy())
```

ML modelleri genellikle girdi özniteliklerine ve etiketlerine sahip olmak isteyen. Bunu yapmak için flat_map içeren satırından sonrası sıkalım ve eu satırları ekleyelim.

```
dataset = dataset.map(lambda window: (window[:-1], window[-1:])) } [0,1,2,3][4] şeklinde her pencere parçalanmış oldu
for x,y in dataset: } deneceklerin karıştırılması da oldukça önemlidir. ml sırası dönenmez.
    print(x.numpy(), y.numpy())
```

dataset = dataset.shuffle(buffer_size=10)

Gruplama yaparak TF mevcut veri kümlesi ile gelleğinden sonra veri grubunu yenileyebilir.

Tüm işlevleri bir araya getirelim.

```
def window_dataset(series, window_size, batch_size=32, shuffle_buffer=1000):
    dataset = tf.data.Dataset.from_tensor_slices(series)
    dataset = dataset.window(window_size+1, shift=1, drop_remainder=True)
    dataset = dataset.flat_map(lambda window: window.batch(window_size+1))
    dataset = dataset.shuffle(shuffle_buffer)
    dataset = dataset.map(lambda window: window[:-1], [window[-1]]).batch(batch_size).prefetch(1)
    return dataset
```

3) MAKİNE ÖĞRENİMİ İLE TAHMİN

Bu bölümde altında zaman serilerini tahmin etmek için makine öğrenmesi modeli geliştireceğiz. İlk olarak basit bir doğrusal regresyon modeli oluşturacağız sonrasında ise iki katmanlı sinir döşer oluşturacağız. Daha önceden kullandığımız ve zaman serisi oluşturduğumuz ifadeleri kullanacağız. Kerasphare ile aktarmalar ile başlayalım:

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

Ve yine bir önceki basit altında oluşturulan 'window-dataset' fonksiyonunu kullanacağız. Zaman serisi eğitim ve doğrulama veri seti olarak iki parçaya ayıralım:

```
split_time = 1000  
time_train = time[:split_time]  
x_train = series[:split_time]  
time_valid = time[split_time:]  
x_valid = series[split_time:]
```

Sırada linear modelimizi oluşturma var;

[keras.backend.clear_session() → keras arkası oturumunu temizledik]

[tf.random.set_seed(42) → Kodun tekrarlanabilir olmasını sağlar.]

[np.random.seed(42)]

[window_size = 30]

[train_set = window_dataset(x_train, window_size)]

[valid_set = window_dataset(x_valid, window_size)]

[model = keras.models.Sequential([

[keras.layers.Dense(1, input_shape=[window_size])]
])]

[optimizer = keras.optimizers.SGD(Lr=1e-5, momentum=0.9)]

[model.compile(loss=keras.losses.Huber(), optimizer=optimizer, metrics=['mae'])]

[model.fit(train_set, epochs=100, validation_data=valid_set)]

Düzen eğitme oranının nasıl seçilipne bakalım; eğitimi defalarca genişletir ve en iyi sonucu veren değer bulmaya çalışır. Yukarıdaki kod bir 'eğitme oranı planlayacısı' oluşturuyor. Modelin tanımlanması ile optimizator arasına bir kod satırı ekleyelim 'fit' metodunda motive edilmesi gerekliliğini unutmayalım.

Lr_schedule = keras.callbacks.LearningRateScheduler(

[Lambda(epoch=1e-6 * 10 ** (epoch/30))])

history = model.fit(train_set, epochs=100, callbacks=[Lr_schedule])

Eğitme kaybının hızla düşer ve bir süre yavaş hızda bir düzeye yarar ve sonra bir patlama noktası ile tekrar en yüksek değerleri alır. Grafiğini çizelim

plt.semilogx(history.history['Lr'], history.history['loss'])

plt.axis([1e-6, 1e-3, 0, 20])

Grafiğe baktığımızda lr değeri için en uygun değerin 10^{-5} olabileceği anlaşıldı. Bir diğer piste nokta ise erken durdurma kullanmaktadır. Bunu az önceki kodda 'Lr-schedule' yi yaradığınızda aralığa yazabilirisiniz. fit metodu yine modifiye bekleyecektir.

early_stopping = keras.callbacks.EarlyStopping(patience=10)

model.fit(train_set, epochs=500, validation_data=valid_set, callbacks=[early_stopping])

'patience = 10' olması 10 epochs için doğrulama oranı durma noktasına yaklaşırsa eğitimi durdurur böylelikle fazla uygun (overfitting) olmamıştır.

Simdi tahminler yapınca igin modelimizi kullanabiliriz. Bu nedenle, zaman serisinin bir kısmını ve pencere boyutunu parametre olarak alan 'model-forecast' adında bir işlev oluşturuyoruz.

```
def model_forecast(model, series, window_size):
    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size + shift, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size))
    ds = ds.batch(32).prefetch(1)
    forecast = model.predict(ds)
    return forecast
```

Simdi tahmin yapabiliyoruz, zaman serisi olarak test setinden bir kısım vereceğiz.
lin_forecast = model_forecast(model, series[split_time - window_size:-1], window_size)[:, 0]

OUT = (46,) # lin_forecast.shape

ikinci boyut
tan kurtulur.

Sırada buna gizemek var;

```
plt.figure(figsize=(10, 6))
plot_series(time_valid, x_valid)
plot_series(time_valid, lin_forecast)
```

Modelimizin performansına bakmak isterseniz:

```
keras.metrics.mean_absolute_error(x_valid, lin_forecast).numpy() # OUT = 5.34
```

Simdi de iki katmanlı bir sinir ağı oluşturuyoruz. Bu aşamada önceki modelde bantlar eğriler yapacağımız 'Lr-schedule' ile en iyi parametre değerini bulup sonrasında 'early-stopping' işlevi ile modeli eğitime tabi tutacağız.

```
[ keras.backend.clear_session()
  tf.random.set_seed(42)
  np.random.seed(42)
```

```
[ window_size = 20
  train_set = window_dataset(x_train, window_size)
  model = keras.models.Sequential([
    keras.layers.Dense(10, activation="relu", input_shape=[window_size]),
    keras.layers.Dense(10, activation="relu"),
    keras.layers.Dense(1)])
```

```
( Lr_schedule = keras.callbacks.LearningRateScheduler(
    lambda epoch: 1e-7 * 10 ** (epoch / 20))
  optimizer = keras.optimizers.SGD(lr=1e-7, momentum=0.9))
  model.compile(loss=keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
  history = model.fit(train_set, epochs=100, callbacks=[Lr_schedule])
```

'Lr-schedule' nin grafiğini gözden geçirmekte optimizator igin en iyi parametrenin ' 10^{-5} ' olacağının karar verip kodu güncelliyoruz. Sıradıkla mavi işaretli alana erken durdurma kodu ekleyip fit metodunu modifiye etmemiz kalmış. Tüm bu işlemlerden sonra modelimizin göreceli ve tahmin yeteneğine bakalım.

```
dense_forecast = model_forecast(model, series[split_time - window_size:-1],
                                  window_size)[:, 0]
```

Grafiğini gösterelim ve 'mae' değerine bakalım.

```
plt.figure(figsize=(10, 6))
plot_series(time_valid, x_valid)
plot_series(time_valid, dense_forecast)
```

```
keras.metrics.mean_absolute_error(x_valid, dense_forecast).numpy() # OUT = 5.29
```

TEKRARLAYAN SINIR AĞLARI - RECURRENT NEURAL NETWORK - RNN

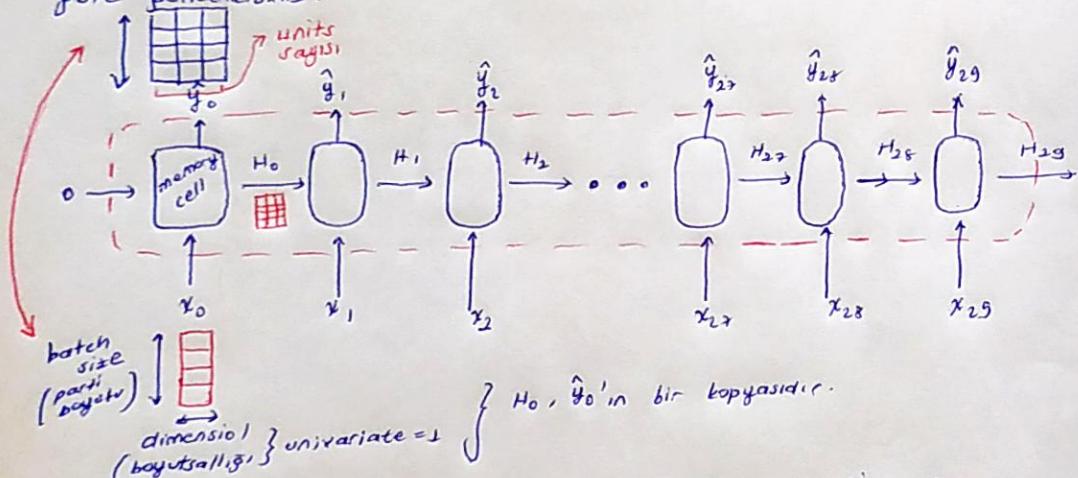
Bir dizi girdigi sırayla işlegebilen bir katmandır. RNN'ler tüm dizileri girdi olarak okuyabilir. Tam girdi ekteli ve boyutludur.

Şekil = [parti boyutu, zaman adımları, zaman adımı girdilerinin boyutsallığı]
 Shape = [batch size, #time steps, # dims]

Recurrent Layer (tekrarlayan katman), gitti hesaplamak için tekrar tekrar kullanılmış tek bir bellek hücreinden olur. Bu bellek hücresi temelde tek bir sinir apıdır.

Yineleme bir katmanın yalnızca tek bir hücre içerdipini anlamak önemlidir. Bellek hücresinin bir zaman adımdındaki etkisinin bir kismı, bir sonraki zaman adımdında kendisini geri bester.

Bu mimariyi söyle övmeye başzetebilirsiniz, okurken her seferinde tek bir belimeye odaklanırız ancak oynar zamanda her tekmenin bağlamını da bilirsiniz ve bu bağlamı her zaman buna göre güncellersiniz.



RNN ve zaman verileri birlikte galterken gereklisini ki eğitim sırasında kayıp deperi beklenmedik şekilde yukarı ve aşağı zıplar. Bu nedenle, erken durdurma teknikini kullanırsanız eğitim çok kişiye kesilebilir. Böyle bir durumda 'patience' parametresini yeteri kadar büyük bir deperi atadığınızdan emin olun. Ancak bu nihai modelin eğitim sırasında göreven en iyi modelden çok sonra olacağının ve daha kötü olabileceğini anlamına gelir.

Bu nedenle, model her iyileştirildiğinde bir kontrol noktasına kaydetmek ve eğitim sonunda en iyi modele geri dönmek oldukça iyi bir fikirdir. Bunun için callback olarak 'model_checkpoint' tanımlanmalıdır.

RNN'ler özellikle yüksek frekanslı bir zaman serisi, çok fazla eğitim verisi olduğundan ve singel-predictor oranı yüksek olduğunda etkili olabilir, ancak her zaman iyi galermaz.

RNN ile Tahmin

Bu başlık altında daha önceki zaman serisi bağıtları altında oluşturduğumuz zaman serisinin aynısını oluşturacağız. Mevsimsellik, gürültü gibi tanımlı fonk. yanı sıra 'window-dataset' ve 'model-forecast' işlevlerini tekrar kullanacağız. Son olarak 1461 birimlik zaman serisimizi 1000 noktadan (split-time) ayırp eğitim ve doğrulama periyodu oluşturduk.

Simdi RNN ile tahmin modeli oluşturalım;

```
[ keras.backend.clear_session()
  tf.random.set_seed(42)
  np.random.seed(42)
  window_size = 30
  train_set = window_dataset(x_train, window_size, batch_size=128)
  model = keras.models.Sequential([
    keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1), input_shape=[None]),
    keras.layers.SimpleRNN(100, return_sequences=True),
    keras.layers.SimpleRNN(100),
    keras.layers.Dense(1),
    keras.layers.Lambda(lambda x: x * 200.0)
  ])
```

lr_schedule = keras.callbacks.LearningRateScheduler (Lambda epoch: 1e-7 * 10^(epoch/10))
optimizer = keras.optimizers.SGD (lr = 1e-7, momentum=0.9)

model.compile (loss=keras.losses.Huber(), optimizer=optimizer, metrics=['mae'])
history = model.fit (train_set, epochs=100, callbacks=[lr_schedule])

'callback' olarak tanımlı olan 'lr_schedule' yi önceki basılıktaki gibi gizerek 'lr' parametresi igin idel değerin 1.5e-6 olarak bulunur. Bu değeri optimizörde yerine yazdım ve sonrasında 'lr_schedule'ının bulunduğu kod satırını silip 'compile' işlevinden hemen sonra etken durum ve ığilaştırdığımız modeli kaydetmek igin kontrol nok. peri aramasi (callback) kullandım.

'fit' işlevini modifiye etmemi unutmayalım.

early_stopping = keras.callbacks.EarlyStopping (patience=50)

model_checkpoint = keras.callbacks.ModelCheckpoint ("my-checkpoint.h5", save_best_only=True)

model.fit (train_set, epochs=500, validation_data=valid_set, callbacks=[early_stopping, model_checkpoint])

Bu aşamadan sonra en iyi modeli checkpoint olarak kaydedelim. checkpoint])

model = keras.models.load_model ("my-checkpoint.h5")

Modelimizi yükledikten sonra doğrulama veri seti üzerinde tahminler yapalım ve mae.

rnn_forecast = model_forecast (model, series [split_time - window_size :-1], window_size)[0]

plt.figure (figsize=(10,6))

plot_series (time_valid, x_valid)

plot_series (time_valid, rnn_forecast)

keras.metrics.mean_absolute_error (x_valid, rnn_forecast).numpy() #OUT=5.41

RNN ile yaptığımızı aynı 'Sıradan Sıraya' (Sequence-to-sequence) ile yapalım. Bu durumda farklı bir veri kümelerine ihtiyaçımız olacak çünkü etiketlerin tensörler yerine sıralı olması gereklidir. Veri seti oluşturmak için işlev tanımlayalım.

```
def seq2seq_window_dataset(series, window_size, batch_size=32, shuffle_buffer=1000):
    series = tf.expand_dims(series, axis=-1)
    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size+1, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size+1))
    ds = ds.shuffle(shuffle_buffer)
    ds = ds.map(lambda w: (w[:-1], w[1:]))
    return ds.batch(batch_size).prefetch()
```

Normalde kullandığımız 'window-dataset' işlevine oldukça benzer bir işlev tanımladık. return anahtar kelimesinden dınceki map fonk. farklılığı var. Giriş öznitelikleri gine aynı olursa etiket son dəfər olmaz yerine penceredeki ilk dəfərler haric, təm dəfərlər olur. Gelin 10 dəfər içeren bir veri küməsinə baxalım;

```
for X_batch, Y_batch in seq2seq_window_dataset(tf.range(10), 3, batch_size=1):
    print("X:", X_batch.numpy())
    print("Y:", Y_batch.numpy())
```

Tanımlayacağımız yeni model RNN modeline oldukça benzer olacaktır. İlk olarak RNN modelimizin ilk katmanı olan Lambda katmanını siliyoruz. İhtiyaçımız yok gənclər girdi boyutlarını zaten olmasa gerektiği gibi dəğər.

Burunla beraber ikinci SimpleRNN katmanımızda da iki gibi 'return_sequences=True' parametresini ekliyoruz. Bu aşamadan sonra gine ilk adım olarak,

Lr (Learning rate) parametremiz iğin en uygun dəfəri bulmak iğin callback işlevini kullanıyoruz. Lr iğin en ideal dəfər = 1e-6 olaraq kəşimizə əlikif.

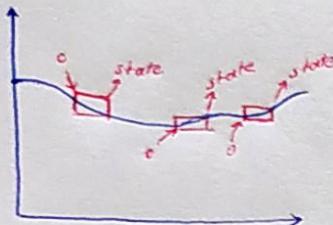
Bu aşamadan sonra callback işlevi olaraq erken durdurma kullanıyoruz. Kifavt işlevində kontrol noktası iğin bir callback tanımlanmadı bu zamanın bu parametresi istəsək elde edə biliriz. Bu noktada eklemeden devam edelim. Və simdi sırasıyla tahmin, tablo qızılı ve 'mae' dəfərimizi bulma zamanı.

```
rnn_forecast = model_forecast(model, series[:, np.newaxis], window_size)
rnn_forecast = rnn_forecast[split_time - window_size:-1, -1, 0]
```

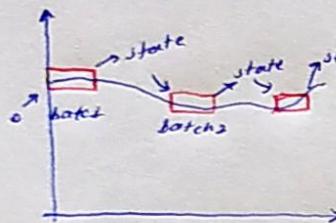
```
plt.figure(figsize=(10, 6))
plot_series(time_valid, rvalid)
plot_series(time_valid, rnn_forecast)
```

```
keras.metrics.mean_absolute_error(x_valid, rnn_forecast).numpy() # out: 5.13
```

DURUMLU VE DURUMSUZ RNN (Stateless and Stateful)



Her pencere iain RNN oluşturacak ve tahminler yapacak, ancak bunu yapmak için sıfır eşit bir başlangıç durumu kullanacaktır. Dahili olarak, tahminlerini yapana kadar her zaman adminda durumu güncellefetcektir ve eğitimin sırasında bir geri yayılım türü olacaktır. Ancak bundan sonra RNN son durumunu kesecektir(futmat). Bu nedenle bu tür RNN'ler durumsuz (stateless) RNN olur. Bu RNN'leri kullanması kolaydır ancak pencerenin uzunluğundan daha uzun kalıpları öğrenmez.



İlk parti 0'dan başlar ancak bir sonraki parti, bir öncekinden hemen sonra bulunan tek bir pencereden oluşur, "0" durum vektörü ile başlamaktır. Yerine, öncekçe eğitimin yineleneşmesinin son durum vektörü ile başlar. RNN, her zaman adminda durum faktörünü güncellerek tahminler yapar. Bu tüm partiler iain aynı olarak işler ve sonan serisinin sonuna ulaşınca bir son durum vektörüdür olsor. Bu nedenle son durumu sıfır.

İşte, zaman serisinin başlangıcından baştan başlarız.

Durum bilgisi olan RNN iain veri wmesi oluşturmak iain yeni bir fonk. ihtiyaç doğar. Bu işlev öncelikle oldukça benzerdir. window yöntemi geçtiğinden "shift=1" yerine "shift=window-size" olacaktır. Ayrıca her pencere bir önceki pencereyi takip edecektir.

İkinci degritlik, artık pencereleri karıştırıyoruz. (shuffle). RNN bu durumda ardışık olmalarını bekler. Jon olarat, tek bir pencere içeren grupları (.batch(1)) kullanırız.

G) DURUM BİLGİSİ OLAN RNN'LER İLE TAHMİN

Bu bölüm altında yine serileri öreme, trend, gürültü gibi fonksiyonları kullanacağız. Yukarıda belirttiğim gibi sadece window-dataset ifadesini güncellememiz gerekiyor.

`def sequential_window_dataset(series, window_size):`

```
series = tf.expand_dims(series, axis=-1)
ds = tf.data.Dataset.from_tensor_slices(series)
ds = ds.window(window_size+1, shift=window_size, drop_remainder=True)
ds = ds.flat_map(lambda window: tf.stack([window[:-1], window[1:]]))
ds = ds.map(lambda window: (window[:-1], window[1:]))
return ds.batch(1).prefetch(1)
```

Nasıl gürültüyü düşmek için basit bir zaman serisi verelim;

`for X_batch, y_batch in sequential_window_dataset(tf.range(10), 3):`
`print(X_batch.numpy(), y_batch.numpy())`

Sırada her yinelemenin (epoch) başında modelin durumunu sıfırlamamız gerekiyor. Bunun iain kendi sıfırlama durumunu geri geçiriyoruz. (call back)

`class ResetStatesCallback(keras.callbacks.Callback):`

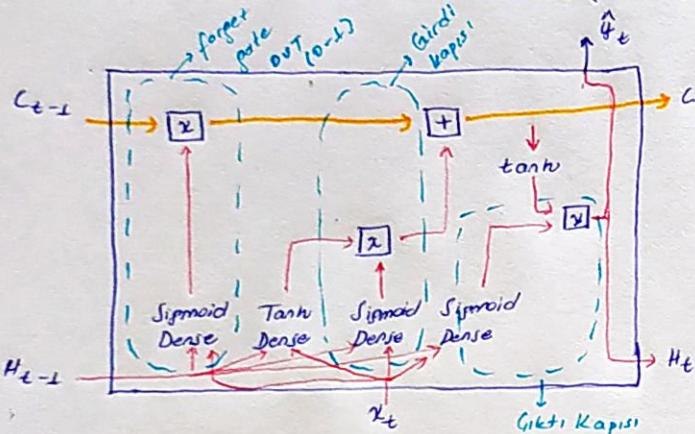
```
def on_epoch_begin(self, epoch, logs):
    self.model.reset_states()
```

Bu aşamadan sonra ise sırasıyla,

modelimiz iain en iy learning rate değerini buluyoruz. Bu değer ile optimizatörümüz güncelliyoruz. ve call back olarat early stopping (erken durdurma) ve kontrol noktası ekliyoruz. 'reset-state' iki durumda da callback olarat kullanılır. Sequential model iaint keras.layers.SimpleRNN(100, return_sequences=True, stateful=True, batch_input_shape=[1, None, 1]),

keras.layers.SimpleRNN(100, return_sequences=True, stateful=True), model.compile ifadesi 'reset_states=ResetStatesCallback()' ile oluşturup fit metodundaki 'callbacks' listesine eklenmemi unutmayı. Diğer kısımlar önceliği başlıklar ile aynı şekilde iterler.

LSTM - LONG-SHORT TERM memory (Uzun-Kısa Süreli Belirleme Aşları)



Unutma kapisi gelen değerin 0 veya $t-1$ 'e yakın olma durumuna göre bu iki değerden biri ile çarpılır. 0 ile çarpılması unutulması yani silinmesi anlamına gelir.

Girdi kapisi ise uzun süreli bellekte bilgiyi ne zaman saklaması gerektiğini öğrenir.

Keras kullanarak bir LSTM oğru uygulamak için tek yapılması gereken durum bilgisi olan RNN modellimizdeki SimpleRNN katmanlarını LSTM ile değiştirmektir. Alternatif olarak genel bir RNN katmanı oluşturarak (keras.layers.RNN) parametre olarak istedigimiz h_t ’e erişebilirsiniz (keras.layers.RNN / keras.layers.LSTM).

Bu basitçe altında durum bilgisi RNN de oldin işlevlerin (prensip, trend vb.) aynısını kullanacağız. Yapmanız gereken tek değişiklik Sequential mode tanımlanırken kullanılan SimpleRNN yerine aşağıdaki katmanları kullanmak olacaktır;

`keras.layers.LSTM(100, return_sequences=True, stateful=True,`
`batch_input_shape=[1, None, 1]),`

`keras.layers.LSTM(100, return_sequences=True, stateful=True),`

CNNs

$$\text{ReLU}(\omega_0 x_0 + \omega_1 x_1 + \omega_2 x_2 + b) \rightarrow \text{3 boyutlu filtr}$$

Padding

Ayarlanması gereken diğer bir parametre kollanmak istediğiniz dolgu (padding) türüdür.
 = 'same' olursa çıktı dizisi ile girdi dizisi aynı uzunlukta olur. girdinin sağ-soluna 0 ekler
 = 'causal' olursa 'same' gibiidir. ancak sağda soluna sıfır boyar.

Stride

Diğer bir parametre ise stride (adım)'dır. Varsayılan olarak 1'e esittir.

Kernels

İon olarak filtrin sayısını tanımlamamız gereklidir. Düşük olursa model performansı düşer
 yüksek olursa ise overfitting durumuna girebilir.

3) CNN ile Tahmin

Bu bölüm altında, dn isteme için 1 boyutlu evrişim katmanlarını kullanarak tekrarlayan (RNN) sinir ağımız nasıl eğitilebileceğimizi ve ayrıca zaman serisi tahmini yapmak için tamamen evrişimli bir sinir ağı oluşturacağız.

Bu bölüm altında daha önceki tanımlı olan gürültü, trend gibi iplerleri yeniden kullanacağız. Bununla beraber daha önceki oluşturduğumuz 'seq2seq-window-dataset' ve 'model-forecast' scriptlerinde de ihtiyacımız olacak.

Sıra ana kodda oluşturduğumuz her veri setindeki gibi gürültü ve trendleri eklediğimiz bir zaman serisini eğitim ve doğrulama periyodu olarak ayıryoruz. Bu aşamadan sonra 1 boyutlu bir evrişimli katman ile LSTM modelimizi eğitebilmek gerekmeyecektir:

Bunu yapmak için LSTM (ilk) katmanından önce bir adet 1-D Convolution katman ekliyoruz. Bu yapıyımda ilk LSTM katmanındaki 'input-shape' parametresini sil.
model = keras.models.Sequential ([

```
    keras.layers.Conv1D (filters=32, kernel_size=5, strides=1, padding="causal",
                        activation="relu", input_shape=[None, 1]),
    ...
```

Bu aşamadan sonra ailek olduğumuz gibi ilk olarak ideal 'Lr' (Learning-rate) değerini optimize ediciye parametre olarak verip ikinci kez 'fit' işlemi yaparız. Bu sefer callbacks olarak erken durdurma ve kayıt noktası veririz. Bu aşamadan sonrası ailek olduğumuz taminin grafiğinin ve 'mae' değerinin bulunmasıdır. İlk 'fit' işleminde 'validation set' yokken ikinci (son) 'fit' işleminde olduğunda dikkat edin!

4) CNN MODELİ

```
[ keras.backend.clear_session()
  tf.random.set_seed(42)
  np.random.seed(42)
  window_size=64
  train_set=seq2seq.WindowDataset(x_train, window_size, batch_size=128)
  model=keras.models.Sequential()
  model.add(keras.layers.InputLayer(input_shape=[None,1]))
  for dilation_rate in [1, 2, 2, 4, 16, 32]:
    model.add(keras.layers.Conv1D(filters=32, kernel_size=2, strides=1,
                                dilation_rate=dilation_rate, padding='causal', activation='relu'))
  model.add(keras.layers.Conv1D(filters=1, kernel_size=1))
  Lr_schedule=keras.callbacks.LearningRateScheduler(lambda epoch: 1e-4 * 10 ** (epoch // 10))
  optimizer=keras.optimizer.Adam(lr=1e-4)
  model.compile(loss=keras.losses.Huber(), optimizer=optimizer, metrics=['mae'])
  history=model.fit(train_set, epochs=100, callbacks=[Lr_schedule])]
```

Bu aşamadan sonra en iyi 'Lr' değerini bulmuş oluyoruz. En iyi 'Lr' değerini bulduktan sonra optimizör metodumuzdaki değeri güncelleştirdik 'Lr-schedule' kod satırını siliyoruz. Bundan sonra ise 'train_set' değerinin hemen altına valid_set (doğrulama seti) tanımlıyoruz. Her zamanki gibi erken durdurma ve kayıt noktası geri aramalarını oluşturup aşağıdaki 'fit' metodu ile modeli eğitiyoruz.

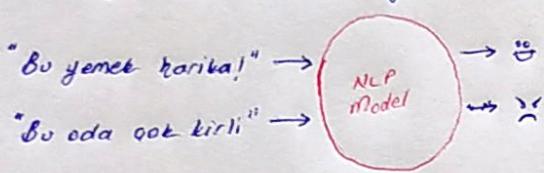
```
history=model.fit(train_set, epochs=500, validation_data=valid_set,
                    callbacks=[early_stopping, model_checkpoint])
```

En son kaydedilen (en iyi) checkpoint noktasını model olarak tanımlayıp, taminin değerlerini gitiyoruz. En son aşamada olarakta 'mae' değerini buluyoruz.

NLP: Tokenizasyon ve Gömme (Tokenization and Embeddings)

NLP büyük ölçüde metin ve konusmadaki anlamı analiz etmeye odaklanır. Bu işlem genellikle belirli bir metinde veya konuşmada hangi kelimenin bulunduğuunun anlaşılmasıyla başlar. Daha sonra bu kelimelerin döngülerine takılır. Yeni metinler oluşturmak için de kullanılır (script veya film senaryosu yazmak).

NLP birçok uygulamada kullanılır. Örneğin: duygusal analizi, çeviri, sesli asistan, akıllı hoparlör ve diğer akıllı ev aletleri gibi.



NLP with TensorFlow

1. BÖLÜM

- * Tokenizasyon
→ harflerin veya kelimelerin birbirinden ayrılmış hali.
- * Gömme
→ işlevselliğe, duyguya vb.

2. BÖLÜM

- * RNN
- * Metin Oluşturucu

4) METİNİN TOKENLETTİRİLMESİ

Sınırlı sayıları, girdiler olarak sayıları kullanır, bu nedenle girdi metnimizi sayılarla denetirmeliyiz. Belirtekleme (tokenizasyon), girdilerimizde sayı olmayan içeriğimiz, ancak bunu yapmanın birden fazla yolu vardır. Her harfin kendisi sayısal simgesi, her bir sözcüğün ve sözleşke ögeinin sayısal simgesi vb.

Mevcut sınırlı sayıları harflere dayalı tokenleştirme her zaman iyi çalışmaz. Örneğin anahtarlar aynı harflerden oluşabilir ancak çok farklı anlamları vardır. Bu nedenle, her bir kelimeyi tek tek belirterek başlayalım.

→ Belirteç: TensorFlow ile işlem, 'tf.keras.preprocessing.text' içinde bulunan bir 'Tokenizer' ile kolayca yapılabilir.

tokenizer = Tokenizer(num_words=10)

→ Fit on Texts - Ardından, belirteç girdilerinize uydurmak için (anapıdatı durumda bu bir dizi listesi)
→ 'fit_on_texts()' kullanır.
tokenizer.fit_on_texts(sentences)

→ Text to Sequences

Buradan, cümeleri belirteğimize diziye dönüştürmek için kullanır.
tokenizer.texts_to_sequences(sentences)

→ Sözlük Dizi Sözcükler
Yeni cümleler, belirteci uyadığı yeni kelimeye sahip olabilir. Varsayılan olarak, belirteç bu sözcükleri görmeden gelir ve bunları belirteğinse diziye dahil etmeyecektir. Bu kelimeleri temsil etmek için 'out of vocabulary' veya OOV belirteci kullanabilirsiniz. Bu, ilk başlangıçta tokenizer nesnesi oluştururken belirtilemelidir.

tokenizer = Tokenizer(num_words=10, oov_token='OOV')

→ Kelime Dizinin Gösterilmesi

Son olarak, belirteç oluşturucusunun sayıları sözcüklerle nasıl eşlediği iin, tokenizer.word_index'ı kullanabilirsiniz.

6) METİNDEKİ DİZİLERE (Text to Sequences)

Cümleleri sağda deşertere dönüştürdükten sonra hala sınırlı olmamızı eğit mevcutta girdiler

Eritme konusunda sorunlar var, her cümlenin aynı mevcutta olmayacağından.

Bunu başarmanın iki yolu vardır: data kesişmeleri sıfırlarla doldurmak ve data'ın dizileri data kesişmeleriyle birlikte kullanmak faydalı olacaktır.

TensorFlow'daki 'tf.keras.preprocessing.sequence' sınıfındaki 'pad_sequences' işlevi bu iki görev için de kullanılabilir durumdadır.

Bir dizi listesi verildiğinde, bir 'maxlen' (bu deşerden daha büyük değerler kesilir) belirttiğinizde ve bunan yanı sıra doldurma (padding) ve kesme (truncating) argümanları içeren 'pre' veya son ('post') ayarlarına bağlı olarak eklemeye - kırmaının hangi yöntemiyle yapılacağını da belirtebiliriz.

Varsayılan olarak, doldurma ve kesme işlevleri dizinin başlangıcından itibaren olur. Bu nedenle dizinin sonuna olmasını istiyorsanız bunu 'post' ile belirtmelisiniz.

Örneğin, baştan doldurmak ve ksaltmak için aşağıdaki kod kullanılabilir.

$\text{padded} = \text{pad_sequences}(\text{sequences}, \text{ maxlen}=10)$

10) Kelime Gömmeleri (Word Embeddings)

Gömmeler, çok boyutlu bir uzayda vektör küpleridir. Burada her vektör bu boyuttarda belirli bir ~~kelimeyi~~ kelimeyi temsil eder. İnsanlar iain birçok boyutta duygular olsada TF projektör bu küpleri 3D projeksiyonda gözletememiz sağlar.

Bu, her bir kelimeyle ilişkili daha olumlu veya daha olumsuz duyguya etrafında kümeler görmemiş beklediğimiz duygularını iain faydalı olabilir.

11) Basit Bir Duygu Modeli Oluşturma

Gömmelerimi oluşturmak için once 'tf.keras.layers.Embedding' adlı bir gömme katmanı kullanacağız. Bu katman 3 boyutlu deşerken olur.

d tokenize edilen sözcük deşerinin boyutu

d kullanılabilecek gömme boyutlarının sayısı

d girdi etkinliği.

Bu katmanın gittikten tam bağıntılı katmanlar ile çalışması için yeniden şekillendirilmeli. Bunu 'Flatten' katmanı ile yapabilir veya birebir daha iyi sonuç veren 'GlobalAveragePooling1D' kullanabilir.

14) Modelin Ayarlanması

Halkatırda oluşturduğumuz duyguların geliştirmenin birkaç yolu vardır.

o Veri ve öne işlemeye dayalı yaklaşımlar:

d Data fazla veri

d Kelime boyutunu ayarlama

d Sira sıralığını ayarlama (az veya daha çok döngü ve kesme işlemi)

d Döngü ve kesmenin önce veya sonra olarak tanımlanması

o Model Tabanlı Yaklaşımlar

1 Gömme boyutlarının (embedding dimensions) sayısının ayarlanması

d Flatten veya GlobalAveragePooling1D kullanmanın değiştirilmesi

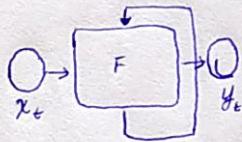
d Dropout gibi katmanları getirdiğinde bulandırmak.

d Aradaki tam bağıntılı katmanlardaki düşüm sayılarının ayarlanması

TEKRARLAYAN SINIR AĞLARI GIRİF

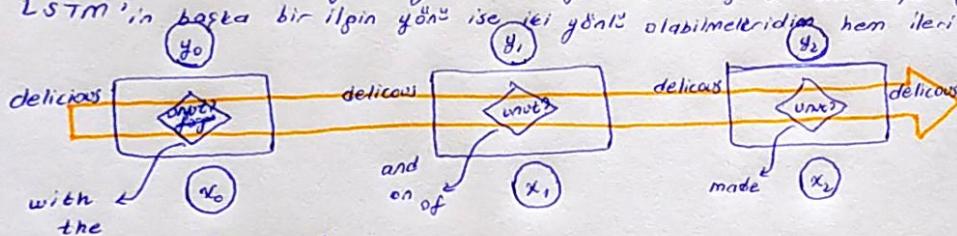
Bu basit örnekte RNN'lerin ve ayrıca yeni metin üretmek için "Text Generation" konularına.

- Tekrarlayan Sinir Ağları (RNN) x girdilerinin bir kismini alır ve bir miktar y çıktı üretir. Ancak aynı zamanda aynı çıktıının bir kisminde kendi içini geri besler. Bu tekrar tekrar yapılırlar. Böylece metin parçası ile aynı bir dizi olarak çok daha önce gelen kelimelerin bir hafızası olur.



- Çömlü Bağılamlı ve LSTM: Metin verikleri de girdiğinden basit RNN yeterli değildir. Basit RNN yapsı deneği girdilerde ilgili bilgileri oldukça hızlı bir şekilde kaybetmiş olabilir, igin bir paragrafta sözlerin.

Uzun-Kısa Süreli Belirleme Modelleri veya LSTM'ler, zaman içinde "hücre durumu" (cell state) koruyarak bu sorunu geçmeye yardımcı olurlar. Bu nedenle hücrenin bir dizi ile igerde taşınabilecek belirli kelimeleri saklamayı veya unutmamayı sebebileceği bir "unut kapısı" (forget gate) vardır. LSTM'in başta bir ilginin yönü ise igin yönü olabilmektedir, hem igeri hem geri.



- Kod ile LSTM Oluşturma:

Bir LSTM katmanın kodu, kullanılabilecek LSTM hücrelerinin sayısı ile birlikte 'tf.keras.layers' sınıfından bir 'LSTM' katmanıdır. Ancak bu yukarıdaki belirtilen hem igeri hem geri bilgi aktarımını kullanmak igin tipik bir çift yönü ('Bidirectional') bir katmana sarılır.

» `tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64))`

Güçlü yönü ('Bidirectional') katman kullanıldığında dikkat edilmesi gereken bir nokta, model igerine bakıldığında bu LSTM hücresi (düzeyi) boyasanz, aslında $(128 = 64 \times 2)$ düzelmeli katman olur. LSTM katmanı ile birlikte 'Flatten' veya 'GlobalAveragePooling1D' katmanına gerek yoktur. LSTM bir gümme ('embedding') katmanı gibi olabilir ve kendi çıktııyla doğrudan tam bağıntılı, yoğun ('Dense') katmanına bağlayabilir.

Ayrıca bir LSTM katmanı ile başka bir LSTM katmanını da bastırabiliriz. Bunu yapmak igin daha önceki LSTM katmanına argüman olarak ('return_sequences=True') vermemiz gereklidir. Verilmesi igeri tam bağlı katmanlar için hazır olursa LSTM katmanının istediği gibi değil.

`tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64), return_sequences=True)`

» LSTM vs. CNN vs. GRU's

o Metin igeri evrimsel katman = Metin girişleri igin evrimsel bir katman kullanmak igin, gümme ('embedding') katmanından sonra bir 'Conv1D' katmanı ekleyebiliriz.

`tf.keras.layers.Conv1D(128, 5, activation='relu')`, buradan tam olarak bağlı kat-

manlara bağlanmak igin bu katmanın çıktılarından 'Flatten' veya 'GlobalAveragePooling1D' kulan-

o Kapılıcı Tekrarlayan Birimler (~~Cells~~ Cells) = Punktelleme ve sıfırlama katılarına sahiptir. Bu katılar neyin tutulacağına veya neyin atılacağına karar verir. LSTM'de olduğu gibi hücre durumu 'cell state', kod LSTM'e çok benzer. 'Bidirectional' katman tarafından sarılan GRU katmanı 'cell state'.

`tf.keras.layers.Bidirectional(tf.keras.layers.GRU(32))`

METİN OLUSTURMA

- Metin oluşturma, bir giriş sırasında verilen ve bir sonraki en olası kelimeyi basitçe tahmin ederek yapılabılır. Bu, orijinal giriş sırasını ve yeni tahmin edilen bitiş kelimesini modele bir sonraki giriş sırasında olarak besleyerek bunu defalarca yapabılır. Bu nedenle, çok kısa bir orijinal girdiden çok edilen tam çıktı, istediğiniz kadar uzun bir süre etkili olarak devam eder.
- Bu aşagidakı deşifrelik, çıktı katmanının artık her olası yeni kelime başında birden fazla eşdeğer olacaktır. Bu nedenle, derlemede 1000 olası kelime olması demek 1000 uzunluğunda bir çıktı diziniz olur. Ayrıca, kayıp fonksiyonunu itili çapraz entropiden kategorik çapraz entropiye gevirmemiz gerektir. Daha önce说得im 0-1 iken, bir kez olası çıktı 'sınıf' vardır.
- Bu işlemde aşamasında kullanılan N-Gramlar şöylededir.

$$I = I \text{ went to the beach} \rightsquigarrow I = \text{went to the beach} \rightsquigarrow I = \text{to the beach with my dog}$$
$$O = \text{with} \quad O = \text{my} \quad O = \text{dog}$$

"I went to the beach with my dog" cümlemiz olsaydı ve max. 5 kelimelik girdi uzunluğumuz olsaydı. Yukarıdaki işlemlerini elde ederdi.

• Metin Üretme Modelinin Optimizasyonu

Modelimizi geliştirmenin birkaç yolu vardır:

▫ Daha fazla veri kullanmak mı?

• Beltek ve çıktı boyutlarını göz önünde bulundurmanız gerekecek.

• Ayrıca sadece en yaygın kullanılan sözcükleri de kullanabilirsiniz.

▫ Verilerinizi Bilin

• Sözlüklerde bir tweetten çok daha fazla kelime var!

▫ Model ayarlamaya devam edin:

• Katman boyutları veya gömme (embedding) boyutlarına eklemeye/ekleme

▫ Tahmin edilen çıktılarında daha fazla varyans ile ilişkisi için olasılıklarla birlikte 'np.random.choice' kullanın.

TensorFlow Lite

- TensorFlow Lite temelde cihazda ML çıkarımını geliştirmek için bir araç takımıdır. Böylece bir ML modeli oluşturabilir, gömülü veya mobil cihazlarda gerekten iyi çalışacak şekilde bir forma dönüştürmek için TF Lite dönüştürücü kullanılır.