Introduction

Artık evrişimsel sınıflandırıcıların temellerini öğrendiğinize göre, daha ileri düzey konulara geçmeye hazırsınız.

Bu derste, görüntü sınıflandırıcılarınıza ivme kazandırabilecek bir yöntem öğreneceksiniz: buna veri artırma denir.

# The Usefulness of Fake Data[¶](https://www.kaggle.com/code/ryanholbrook/data-augmentation#The-Usefulness-of-Fake-Data)

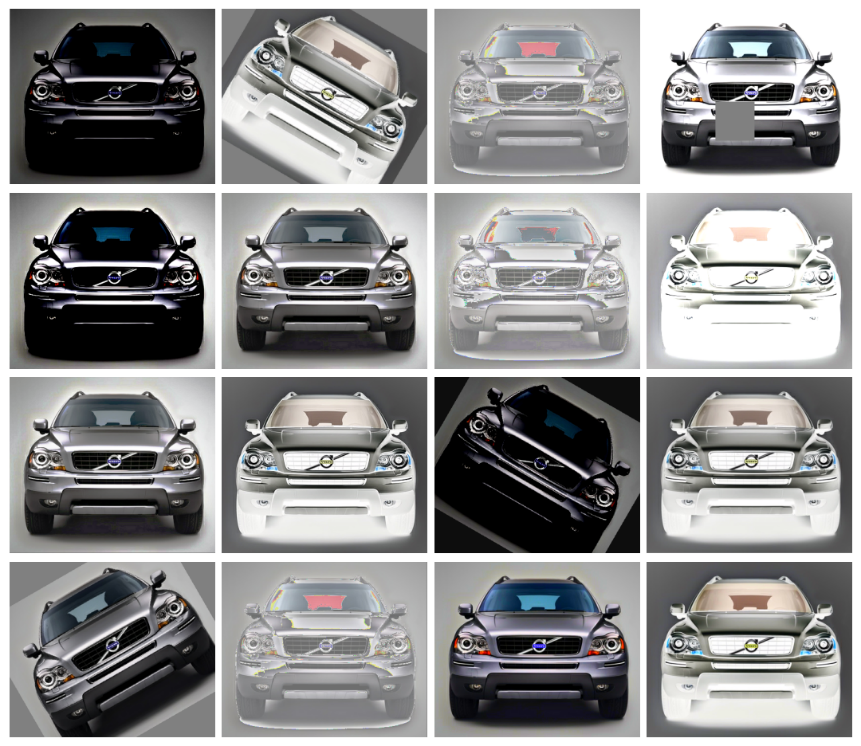
Bir makine öğrenmesi modelinin performansını artırmanın en iyi yolu, onu daha fazla veriyle eğitmektir. Modelin öğrenebileceği örnekler ne kadar fazlaysa, görüntülerdeki hangi farklılıkların önemli olduğunu ve hangilerinin olmadığını o kadar iyi anlayabilir. Daha fazla veri, modelin daha iyi **genellemesine** yardımcı olur.

Daha fazla veri elde etmenin kolay bir yolu, zaten sahip olduğunuz verileri kullanmaktır. Veri kümemizdeki görüntüleri, sınıfını koruyacak şekilde dönüştürebilirsek, sınıflandırıcımıza bu tür dönüşümleri göz ardı etmeyi öğretebiliriz. Örneğin, bir arabanın fotoğrafta sola mı yoksa sağa mı dönük olduğu, onun bir **Kamyon** değil de bir **Araba** olduğu gerçeğini değiştirmez. Bu nedenle, eğitim verilerimizi çevrilmiş görüntülerle **artırırsak**, sınıflandırıcımız "sol veya sağ" farkının göz ardı etmesi gereken bir fark olduğunu öğrenecektir.

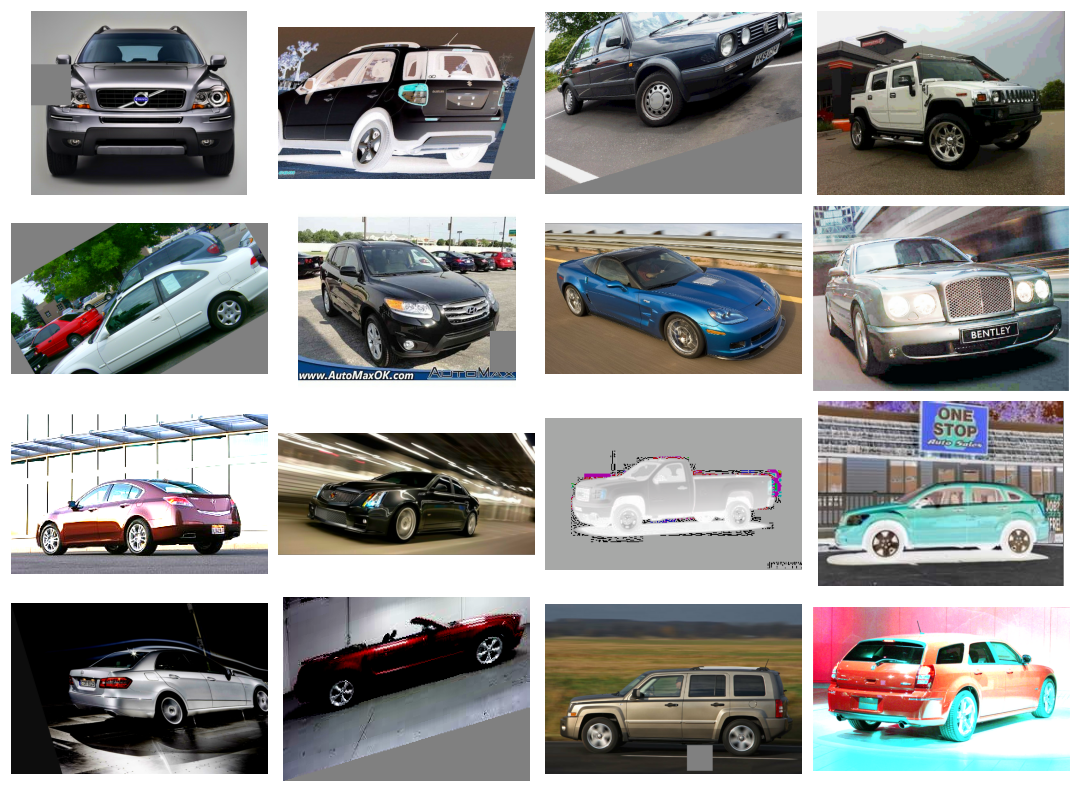
Veri büyütmenin (data augmentation) arkasındaki tüm fikir budur: Gerçek verilere makul derecede benzeyen ekstra sahte veriler ekleyin ve sınıflandırıcınız gelişecektir.

# Using Data Augmentation[¶](https://www.kaggle.com/code/ryanholbrook/data-augmentation" \l "Using-Data-Augmentation" \t "_self)

Genellikle, bir veri kümesini artırırken birçok farklı dönüşüm türü kullanılır. Bunlar, görüntüyü döndürmeyi, rengi veya kontrastı ayarlamayı, görüntüyü bozmayı (warping) veya genellikle bir arada uygulanan diğer birçok şeyi içerebilir. İşte tek bir görüntünün dönüştürülebileceği farklı yollara bir örnek.



Veri büyütme (data augmentation) genellikle **çevrimiçi** (online) olarak, yani görüntüler eğitim için ağa beslenirken yapılır. Eğitimin genellikle veri mini-toplu işleri (mini-batches) üzerinde yapıldığını hatırlayın. Veri büyütme kullanıldığında, 16 görüntüden oluşan bir toplu iş (batch) şöyle görünebilir:



Eğitim sırasında bir görüntü her kullanıldığında, yeni ve rastgele bir dönüşüm uygulanır. Bu sayede, model her zaman daha önce gördüğünden biraz farklı bir şey görür. Eğitim verisindeki bu ekstra çeşitlilik, modelin yeni veriler üzerinde daha iyi performans göstermesine yardımcı olur.

Ancak, her dönüşümün belirli bir problem için faydalı olmayacağını unutmak önemlidir. En önemlisi, kullandığınız dönüşümler sınıfları karıştırmamalıdır. Örneğin, bir **rakam tanıyıcı** eğitiyor olsaydınız, görüntüleri döndürmek '9'ları ve '6'ları karıştırırdı. Sonuç olarak, iyi veri büyütme tekniklerini bulmanın en iyi yolu, çoğu makine öğrenimi probleminde olduğu gibi şudur: deneyin ve görün!

# Example - Training with Data Augmentation[¶](https://www.kaggle.com/code/ryanholbrook/data-augmentation" \l "Example---Training-with-Data-Augmentation" \t "_self)

Keras, verilerinizi iki şekilde zenginleştirmenize olanak tanır. Birincisi, **ImageDataGenerator** gibi bir fonksiyonla veri işlem hattına dahil etmektir. İkincisi ise, Keras'ın **ön işleme katmanlarını** kullanarak model tanımına dahil etmektir. Biz bu yaklaşımı benimseyeceğiz. Bunun bizim için başlıca avantajı, görüntü dönüşümlerinin CPU yerine **GPU** üzerinde hesaplanmasıdır, bu da eğitimi potansiyel olarak hızlandırır.

Bu alıştırmada, 1. Dersteki sınıflandırıcıyı veri büyütme yoluyla nasıl geliştireceğimizi öğreneceğiz. Aşağıdaki gizli hücre, veri işlem hattını kurar.

*# Imports*

import os, warnings

import matplotlib.pyplot as plt

from matplotlib import gridspec

import numpy as np

import tensorflow as tf

from tensorflow.keras.preprocessing import image\_dataset\_from\_directory

*# Reproducability*

def set\_seed(seed=31415):

np.random.seed(seed)

tf.random.set\_seed(seed)

os.environ['PYTHONHASHSEED'] = str(seed)

*#os.environ['TF\_DETERMINISTIC\_OPS'] = '1'*

set\_seed()

*# Set Matplotlib defaults*

plt.rc('figure', autolayout=True)

plt.rc('axes', labelweight='bold', labelsize='large',

titleweight='bold', titlesize=18, titlepad=10)

plt.rc('image', cmap='magma')

warnings.filterwarnings("ignore") *# to clean up output cells*

*# Load training and validation sets*

ds\_train\_ = image\_dataset\_from\_directory(

'../input/car-or-truck/train',

labels='inferred',

label\_mode='binary',

image\_size=[128, 128],

interpolation='nearest',

batch\_size=64,

shuffle=True,

)

ds\_valid\_ = image\_dataset\_from\_directory(

'../input/car-or-truck/valid',

labels='inferred',

label\_mode='binary',

image\_size=[128, 128],

interpolation='nearest',

batch\_size=64,

shuffle=False,

)

*# Data Pipeline*

def convert\_to\_float(image, label):

image = tf.image.convert\_image\_dtype(image, dtype=tf.float32)

return image, label

AUTOTUNE = tf.data.experimental.AUTOTUNE

ds\_train = (

ds\_train\_

.map(convert\_to\_float)

.cache()

.prefetch(buffer\_size=AUTOTUNE)

)

ds\_valid = (

ds\_valid\_

.map(convert\_to\_float)

.cache()

.prefetch(buffer\_size=AUTOTUNE)

)

Found 5117 files belonging to 2 classes.

Found 5051 files belonging to 2 classes.

## **Step 2 - Define Model**[**¶**](https://www.kaggle.com/code/ryanholbrook/data-augmentation#Step-2---Define-Model)

Artırmanın etkisini göstermek için, Eğitim 1'deki modele birkaç basit dönüşüm ekleyeceğiz.

from tensorflow import keras

from tensorflow.keras import layers

*# these are a new feature in TF 2.2*

from tensorflow.keras.layers.experimental import preprocessing

pretrained\_base = tf.keras.models.load\_model(

'../input/cv-course-models/cv-course-models/vgg16-pretrained-base',

)

pretrained\_base.trainable = False

model = keras.Sequential([

*# Preprocessing*

preprocessing.RandomFlip('horizontal'), *# flip left-to-right*

preprocessing.RandomContrast(0.5), *# contrast change by up to 50%*

*# Base*

pretrained\_base,

*# Head*

layers.Flatten(),

layers.Dense(6, activation='relu'),

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

])

## **Step 3 - Train and Evaluate[¶](https://www.kaggle.com/code/ryanholbrook/data-augmentation" \l "Step-3---Train-and-Evaluate" \t "_self)**

Ve şimdi eğitime başlıyoruz!

model.compile(

optimizer='adam',

loss='binary\_crossentropy',

metrics=['binary\_accuracy'],

)

history = model.fit(

ds\_train,

validation\_data=ds\_valid,

epochs=30,

verbose=0,

)

import pandas as pd

history\_frame = pd.DataFrame(history.history)

history\_frame.loc[:, ['loss', 'val\_loss']].plot()

history\_frame.loc[:, ['binary\_accuracy', 'val\_binary\_accuracy']].plot();

Tutorial 1'deki modelin eğitim ve doğrulama eğrileri oldukça hızlı bir şekilde birbirinden ayrılmıştı, bu da bir miktar düzenleme (regularization) yapılması gerektiğini gösteriyordu. Bu modelin öğrenme eğrileri birbirine daha yakın kalabildi ve doğrulama kaybı (validation loss) ile doğruluğunda (accuracy) bazı mütevazı iyileşmeler elde ettik. Bu durum, veri kümesinin büyütmeden gerçekten faydalandığını gösteriyor.