Welcome to Computer Vision!

Bilgisayara görmeyi öğretmek istediniz mi hiç? Bu kursta tam olarak bunu yapacaksınız!

Bu kursta şunları yapacaksınız:

* Keras ile bir **görüntü sınıflandırıcı** oluşturmak için modern derin öğrenme ağlarını kullanacaksınız.
* Yeniden kullanılabilir bloklarla kendi **özel evrişimli ağınızı (convnet)** tasarlayacaksınız.
* Görsel **öznitelik çıkarmanın** ardındaki temel fikirleri öğreneceksiniz.
* Modellerinizin performansını artırmak için **transfer öğrenme** sanatında uzmanlaşacaksınız.
* Veri kümenizi genişletmek için **veri büyütmeyi** kullanacaksınız.

Eğer **Derin Öğrenmeye Giriş** kursunu aldıysanız, başarılı olmak için ihtiyacınız olan her şeyi biliyorsunuz demektir.

Şimdi başlayalım!

# Introduction[¶](https://www.kaggle.com/code/ryanholbrook/the-convolutional-classifier" \l "Introduction" \t "_self)

Bu kurs sizi bilgisayarlı görü alanının temel fikirleriyle tanıştıracak. Amacımız, bir sinir ağının, insan görsel sisteminin çözebildiği türden problemleri çözebilecek kadar doğal bir görüntüyü nasıl "anlayabildiğini" öğrenmektir.

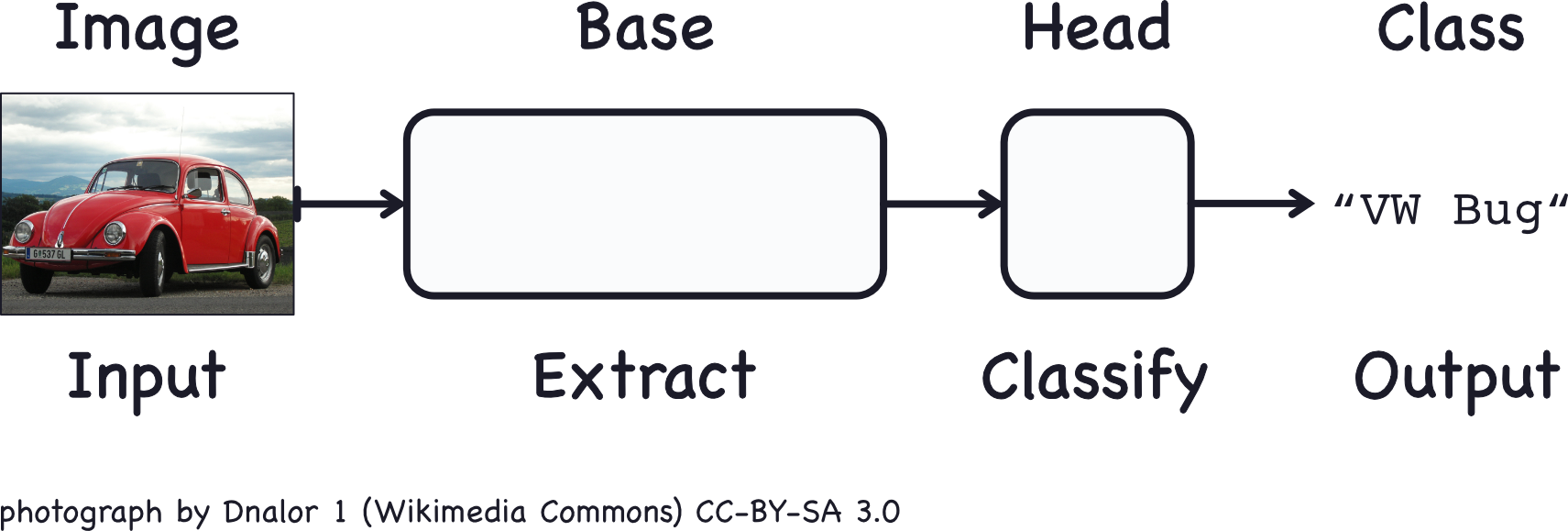
Bu görevde en iyi performansı gösteren sinir ağlarına **evrişimli sinir ağları** denir. (Bazen **convnet** veya **CNN** de diyebiliriz.) Evrişim, bir convnet'in katmanlarına özgün yapısını veren matematiksel bir işlemdir. Gelecek derslerde, bu yapının bilgisayarlı görü problemlerini çözmede neden bu kadar etkili olduğunu öğreneceksiniz.

Bu fikirleri **görüntü sınıflandırma** problemine uygulayacağız: Bize verilen bir resmin, neyin resmi olduğunu söylemesi için bir bilgisayarı eğitebilir miyiz? Bir bitki türünü fotoğraftan teşhis edebilen uygulamalar görmüş olabilirsiniz. İşte bu bir görüntü sınıflandırıcıdır! Bu kursta, profesyonel uygulamalarda kullanılanlar kadar güçlü görüntü sınıflandırıcıları nasıl oluşturacağınızı öğreneceksiniz.

Odak noktamız görüntü sınıflandırma olsa da, bu kursta öğrenecekleriniz her türlü bilgisayarlı görü problemiyle ilgilidir. Kurs sonunda, üretken çekişmeli ağlar ve görüntü bölümleme gibi daha gelişmiş uygulamalara geçmeye hazır olacaksınız.

# The Convolutional Classifier[¶](https://www.kaggle.com/code/ryanholbrook/the-convolutional-classifier" \l "The-Convolutional-Classifier" \t "_self)

Görüntü sınıflandırması için kullanılan bir convnet iki bölümden oluşur: evrişimsel taban ve yoğun kafa.(dense head)

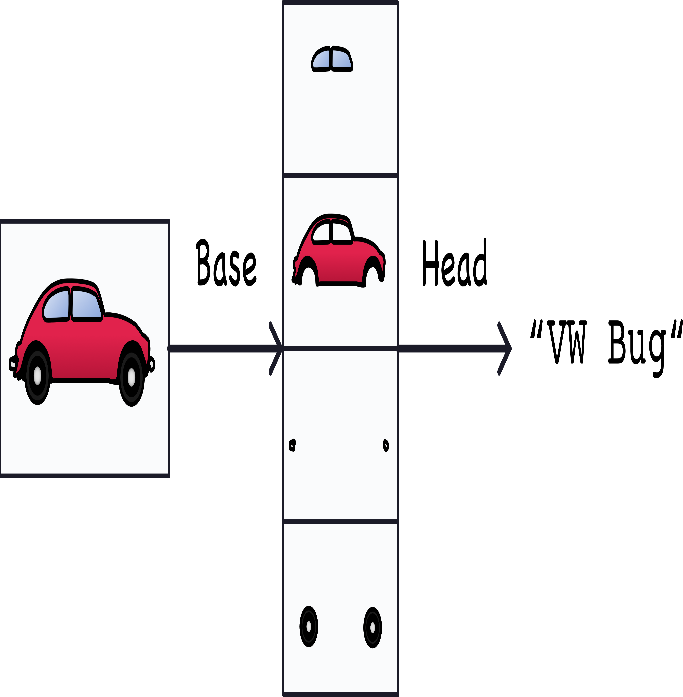


Baz kısmı, bir görüntüden **öznitelikleri çıkarmak** için kullanılır. Esas olarak evrişim işlemini gerçekleştiren katmanlardan oluşur, ancak genellikle başka türde katmanları da içerir. (Bunları bir sonraki derste öğreneceksiniz.)

Baş kısmı, görüntünün **sınıfını belirlemek** için kullanılır. Esas olarak yoğun katmanlardan oluşur, ancak dropout gibi başka katmanlar da içerebilir.

Görsel öznitelikten ne kastediyoruz? Bir öznitelik; bir çizgi, bir renk, bir doku, bir şekil, bir desen veya bunların karmaşık bir kombinasyonu olabilir.

Tüm süreç kabaca şöyle işler:



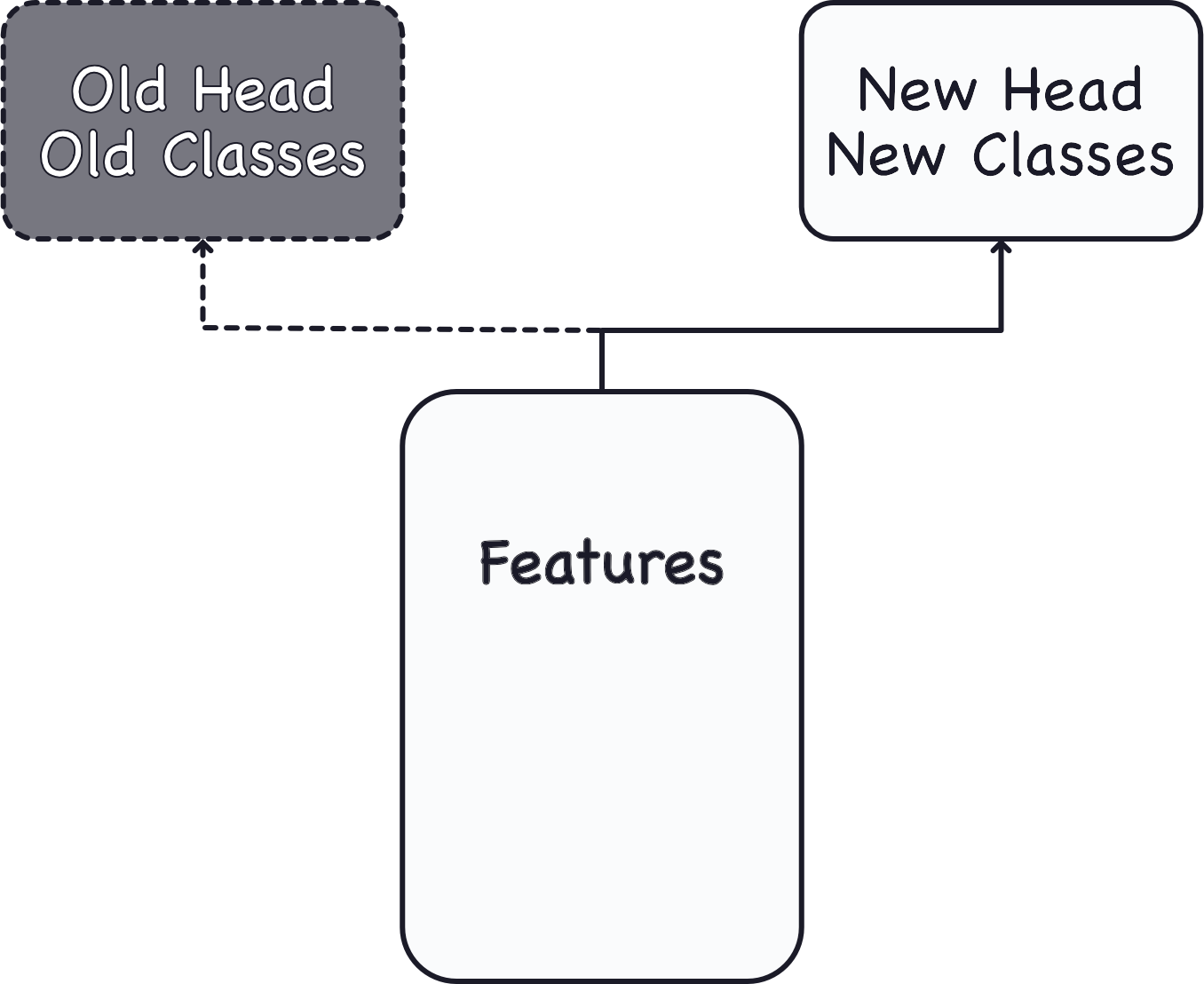
Çıkarılan özellikler aslında biraz farklı görünüyor ama fikir veriyor.

# Training the Classifier[¶](https://www.kaggle.com/code/ryanholbrook/the-convolutional-classifier" \l "Training-the-Classifier" \t "_self)

Ağın eğitim sırasındaki amacı iki şeyi öğrenmektir:

* Bir görüntüden hangi **öznitelikleri çıkaracağını** (**baz**),
* Hangi **sınıfın hangi özniteliklerle** eşleştiğini (**baş**).

Günümüzde evrişimli ağlar nadiren sıfırdan eğitilir. Çoğu zaman, **önceden eğitilmiş bir modelin baz kısmını yeniden kullanırız**. Önceden eğitilmiş baza daha sonra **eğitilmemiş bir baş kısmı** ekleriz. Diğer bir deyişle, ağın **1. Öznitelikleri çıkar** kısmını zaten öğrenmiş olan bölümünü yeniden kullanır ve ona **2. Sınıflandırmayı** öğrenmesi için yeni katmanlar ekleriz.



Baş kısmı genellikle yalnızca birkaç yoğun katmandan oluştuğu için, nispeten az miktarda veriyle çok doğru sınıflandırıcılar oluşturulabilir.

Önceden eğitilmiş bir modeli yeniden kullanma tekniği, **transfer öğrenme** olarak bilinir. Bu yöntem o kadar etkilidir ki, günümüzde neredeyse her görüntü sınıflandırıcı bu tekniği kullanır.

# Example - Train a Convnet Classifier[¶](https://www.kaggle.com/code/ryanholbrook/the-convolutional-classifier" \l "Example---Train-a-Convnet-Classifier" \t "_self)

Bu kurs boyunca, şu problemi çözmeye çalışan sınıflandırıcılar oluşturacağız: Bu bir Araba mı yoksa Kamyon mu? Veri setimiz, yaklaşık yarısı araba, yarısı kamyon olmak üzere çeşitli otomobillerin yaklaşık 10.000 resminden oluşuyor.

## **Step 1 - Load Data**[**¶**](https://www.kaggle.com/code/ryanholbrook/the-convolutional-classifier#Step-1---Load-Data)

Bir sonraki gizli hücre(hidden cell), bazı kütüphaneleri içe aktaracak ve veri hattımızı(data pipeline) kuracak. ds\_train adında bir eğitim bölümümüz ve ds\_valid adında bir doğrulama bölümümüz var.

*# 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(31415)

*# 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.

Eğitim setinden birkaç örneğe bakalım.

import matplotlib.pyplot as plt

## **Step 2 - Define Pretrained Base**[**¶**](https://www.kaggle.com/code/ryanholbrook/the-convolutional-classifier#Step-2---Define-Pretrained-Base)

Ön eğitim(pretraining) için en sık kullanılan veri kümesi, birçok doğal görüntü türünü içeren geniş bir veri kümesi olan **ImageNet**'tir. Keras, ImageNet üzerinde önceden eğitilmiş çeşitli modelleri **uygulamalar** modülünde içerir. Kullanacağımız önceden eğitilmiş modelin adı **VGG16**'dır.

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

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

)

pretrained\_base.trainable = False

## **Step 3 - Attach Head[¶](https://www.kaggle.com/code/ryanholbrook/the-convolutional-classifier" \l "Step-3---Attach-Head" \t "_self)**

Sıradaki adım, sınıflandırıcı başını eklemektir. Bu örnek için, bir gizli birim katmanı (**ilk Dense katmanı**) ve ardından çıktıları 1. sınıf olan **Kamyon** için bir olasılık puanına dönüştüren bir katman kullanacağız. **Flatten** katmanı ise, baz katmanın iki boyutlu çıktılarını, baş katmanın ihtiyaç duyduğu tek boyutlu girdilere dönüştürür.

from tensorflow import keras

from tensorflow.keras import layers

model = keras.Sequential([

pretrained\_base,

layers.Flatten(),

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

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

])

## **Step 4 - Train**[**¶**](https://www.kaggle.com/code/ryanholbrook/the-convolutional-classifier#Step-4---Train)

Son olarak, modeli eğitelim. Bu iki sınıflı bir problem olduğu için, **crossentropy** ve **accuracy**'nin ikili (binary) versiyonlarını kullanacağız. **Adam** optimize edicisi genellikle iyi performans gösterir, bu yüzden onu da seçeceğiz.

model.compile(

optimizer='adam',

loss='binary\_crossentropy',

metrics=['binary\_accuracy'],

)

history = model.fit(

ds\_train,

validation\_data=ds\_valid,

epochs=30,

verbose=0,

)

Bir sinir ağı eğitilirken, **kayıp** (loss) ve **metrik** grafiklerini incelemek her zaman iyi bir fikirdir. **history** nesnesi, bu bilgileri bir sözlük (history.history) içinde barındırır. Bu sözlüğü, Pandas kullanarak bir veri çerçevesine (dataframe) dönüştürebilir ve yerleşik bir metotla grafik haline getirebiliriz.

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();

# Conclusion

Bu derste, bir evrişimli ağ (convnet) sınıflandırıcısının yapısını öğrendik: **öznitelik çıkarmayı** gerçekleştiren bir **baz** kısmının üzerine, bir sınıflandırıcı görevi gören bir **baş** kısmı yerleştirilir.

Baş kısmı, esasen, giriş dersinde öğrendiğiniz gibi sıradan bir sınıflandırıcıdır. Öznitelikler için, baz kısmı tarafından çıkarılanları kullanır. Evrişimli sınıflandırıcıların temel fikri budur: Öznitelik mühendisliği yapan bir birimi, sınıflandırıcının kendisine ekleyebiliriz.

Bu, derin sinir ağlarının geleneksel makine öğrenimi modellerine göre sahip olduğu en büyük avantajlardan biridir: Doğru ağ yapısı verildiğinde, derin sinir ağı, problemini çözmek için ihtiyaç duyduğu öznitelikleri nasıl mühendis edeceğini öğrenebilir.

Gelecek birkaç ders boyunca, evrişimsel baz kısmının öznitelik çıkarmayı nasıl başardığına bakacağız. Ardından, bu fikirleri nasıl uygulayacağınızı ve kendi sınıflandırıcılarınızı nasıl tasarlayacağınızı öğreneceksiniz.