Introduction[¶](https://www.kaggle.com/code/ryanholbrook/custom-convnets" \l "Introduction" \t "_self)

Artık bir convnet'in özellikleri çıkarmak için kullandığı katmanları gördüğünüze göre, bunları bir araya getirip kendi ağınızı oluşturmanın zamanı geldi!

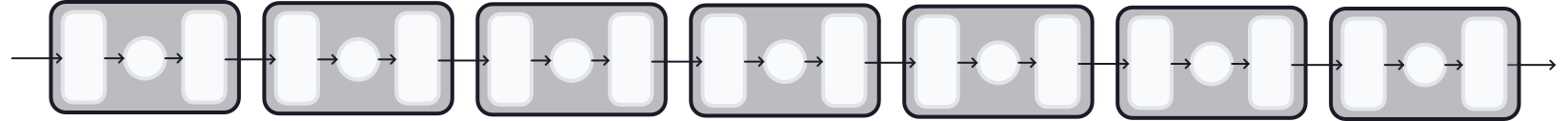
# Simple to Refined[¶](https://www.kaggle.com/code/ryanholbrook/custom-convnets" \l "Simple-to-Refined" \t "_self)

Son üç derste, evrişimli ağların **özellik çıkarmayı** üç işlemle nasıl gerçekleştirdiğini gördük: **filtreleme**, **tespit etme** ve **yoğunlaştırma**. Tek bir özellik çıkarma döngüsü, bir görüntüden yalnızca nispeten basit özellikleri (basit çizgiler veya zıtlıklar gibi) çıkarabilir. Bunlar, çoğu sınıflandırma problemini çözmek için çok basittir. Bunun yerine, evrişimli ağlar bu çıkarma işlemini tekrar tekrar yapar, böylece özellikler ağın derinliklerine doğru ilerledikçe daha karmaşık ve rafine hale gelir.

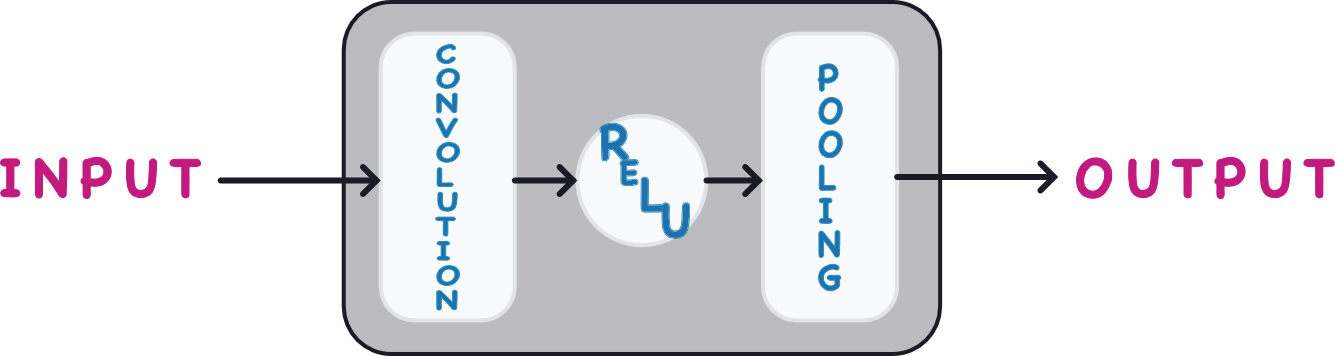


# Convolutional Blocks[¶](https://www.kaggle.com/code/ryanholbrook/custom-convnets" \l "Convolutional-Blocks" \t "_self)

Bunu, bu çıkarmayı gerçekleştiren uzun **evrişimsel blok** zincirlerinden geçirerek yapar.



Bu evrişimsel bloklar, rolünü son birkaç derste öğrendiğimiz **Conv2D** ve **MaxPool2D** katmanlarının yığınlarıdır (ardışık dizileridir).



Her blok bir öznitelik çıkarma döngüsünü temsil eder ve bu blokları bir araya getirerek evrişimli ağ, üretilen öznitelikleri birleştirip yeniden birleştirebilir, onları büyütebilir ve eldeki probleme daha iyi uyacak şekilde şekillendirebilir. Modern evrişimli ağların derin yapısı, bu sofistike öznitelik mühendisliğine olanak tanır ve üstün performanslarından büyük ölçüde sorumludur.

# Example - Design a Convnet[¶](https://www.kaggle.com/code/ryanholbrook/custom-convnets" \l "Example---Design-a-Convnet" \t "_self)

Karmaşık özellikleri tasarlayabilen derin bir evrişimli ağın nasıl tanımlanacağını görelim. Bu örnekte, bir Keras Dizisi modeli oluşturup ardından onu Arabalar veri setimiz üzerinde eğiteceğiz.

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

Bu gizli hücre verileri yükler.

*# 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)

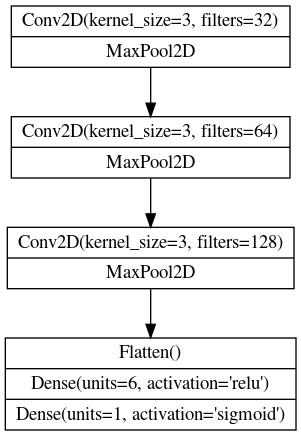
)

Found 5117 files belonging to 2 classes.

Found 5051 files belonging to 2 classes.

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

Kullanacağımız modelin diyagramı şöyle:



Şimdi modeli tanımlayacağız. Modelimizin, üç blok **Conv2D** ve **MaxPool2D** katmanından (baz kısmı) ve ardından **Dense** katmanlarından oluşan bir baş kısmından oluştuğunu görebilirsiniz. Bu diyagramı, uygun parametreleri girerek neredeyse doğrudan bir Keras **Sequential** modeline dönüştürebiliriz.

from tensorflow import keras

from tensorflow.keras import layers

model = keras.Sequential([

*# First Convolutional Block*

layers.Conv2D(filters=32, kernel\_size=5, activation="relu", padding='same',

*# give the input dimensions in the first layer*

*# [height, width, color channels(RGB)]*

input\_shape=[128, 128, 3]),

layers.MaxPool2D(),

*# Second Convolutional Block*

layers.Conv2D(filters=64, kernel\_size=3, activation="relu", padding='same'),

layers.MaxPool2D(),

*# Third Convolutional Block*

layers.Conv2D(filters=128, kernel\_size=3, activation="relu", padding='same'),

layers.MaxPool2D(),

*# Classifier Head*

layers.Flatten(),

layers.Dense(units=6, activation="relu"),

layers.Dense(units=1, activation="sigmoid"),

])

model.summary()

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 128, 128, 32) 2432

max\_pooling2d (MaxPooling2D (None, 64, 64, 32) 0

)

conv2d\_1 (Conv2D) (None, 64, 64, 64) 18496

max\_pooling2d\_1 (MaxPooling (None, 32, 32, 64) 0

2D)

conv2d\_2 (Conv2D) (None, 32, 32, 128) 73856

max\_pooling2d\_2 (MaxPooling (None, 16, 16, 128) 0

2D)

flatten (Flatten) (None, 32768) 0

dense (Dense) (None, 6) 196614

dense\_1 (Dense) (None, 1) 7

=================================================================

Total params: 291,405

Trainable params: 291,405

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Bu tanımda, filtre sayısının blok blok nasıl ikiye katlandığına dikkat edin: 32, 64, 128. Bu yaygın bir kalıptır. MaxPool2D katmanı öznitelik haritalarının **boyutunu** azalttığı için, oluşturduğumuz **miktarı** artırabiliriz.

## **Step 3 - Train**[**¶**](https://www.kaggle.com/code/ryanholbrook/custom-convnets#Step-3---Train)

Bu modeli tıpkı 1. Ders'teki model gibi eğitebiliriz: İkili sınıflandırmaya uygun bir kayıp ve metrik ile birlikte bir optimize edici ile derleyebiliriz.

model.compile(

optimizer=tf.keras.optimizers.Adam(epsilon=0.01),

loss='binary\_crossentropy',

metrics=['binary\_accuracy']

)

history = model.fit(

ds\_train,

validation\_data=ds\_valid,

epochs=40,

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();

Bu model, 1. Derste bahsettiğimiz VGG16 modelinden çok daha küçüktür; VGG16'nın 16 evrişim katmanına karşılık sadece 3 evrişim katmanına sahiptir. Buna rağmen, bu veri kümesine oldukça iyi uyum sağlayabildi. Daha fazla evrişim katmanı ekleyerek bu basit modeli hala geliştirebilir, böylece veri kümesine daha iyi uyum sağlayan öznitelikler oluşturmayı umabiliriz. Alıştırmalarda deneyeceğimiz şey budur.

# Conclusion[¶](https://www.kaggle.com/code/ryanholbrook/custom-convnets" \l "Conclusion" \t "_self)

Bu eğitimde, birçok **evrişimsel bloktan** oluşan ve karmaşık öznitelik mühendisliği yapabilen özel bir evrişimsel ağın nasıl oluşturulacağını gördünüz.