Introduction

## Evrişimsel Katmanlarda Özellik Çıkarma

1. derste, bir evrişimli ağın (convnet) baz kısmının **öznitelik çıkarmayı** nasıl gerçekleştirdiğine dair tartışmamıza başlamıştık. Bu süreçteki ilk iki operasyonun, **relu** aktivasyonuna sahip bir **Conv2D** katmanında gerçekleştiğini öğrenmiştik.

Bu derste, bu sıradaki üçüncü (ve son) operasyona bakacağız: **maksimum havuzlama** ile **yoğunlaştırma**. Keras'ta bu işlem bir **MaxPool2D** katmanı aracılığıyla yapılır.

# Condense with Maximum Pooling

Daha önce oluşturduğumuz modele yoğunlaştırma adımını eklediğimizde şu sonucu elde ederiz:

from tensorflow import keras

from tensorflow.keras import layers

model = keras.Sequential([

layers.Conv2D(filters=64, kernel\_size=3), *# activation is None*

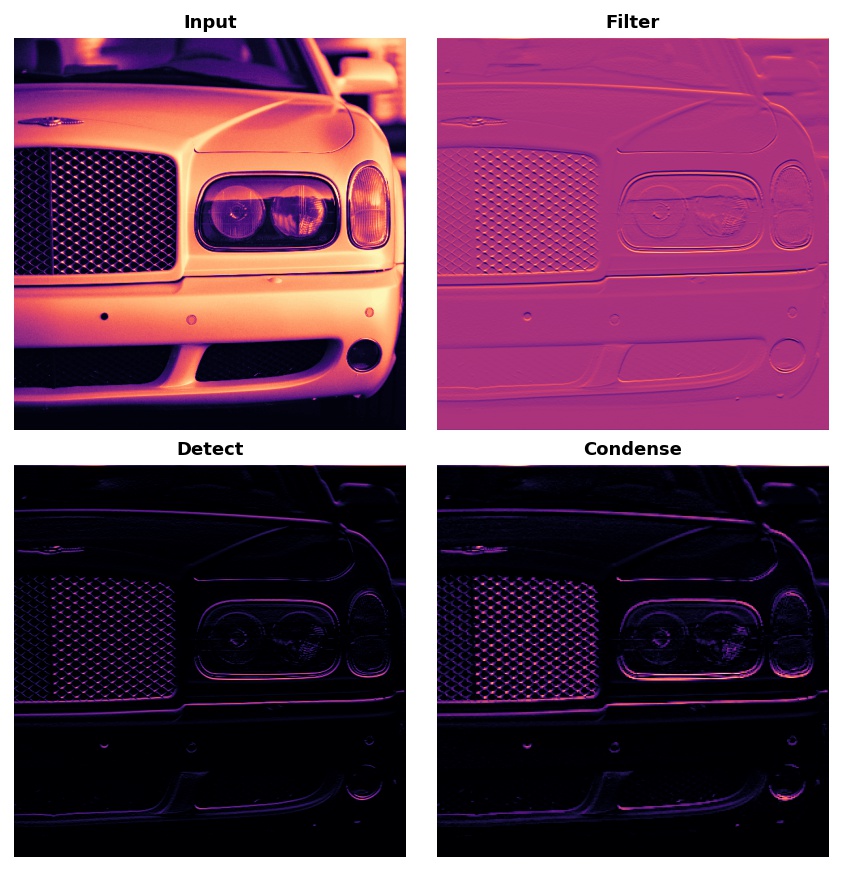
layers.MaxPool2D(pool\_size=2),

*# More layers follow*

])

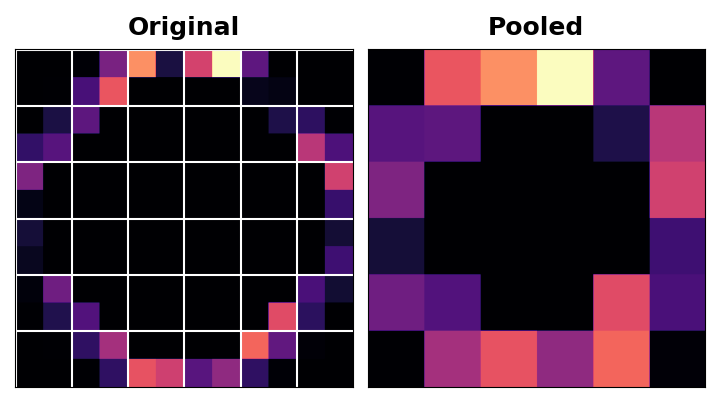
MaxPool2D katmanı, bir **çekirdek** yerine basit bir **maksimum fonksiyonu** kullanması dışında, bir Conv2D katmanına oldukça benzer. pool\_size parametresi kernel\_size parametresine benzer bir işlev görür. Ancak, bir MaxPool2D katmanının, evrişim katmanının çekirdeğinde olduğu gibi eğitilebilir ağırlıkları yoktur.

Geçen dersteki çıkarma görseline tekrar bakalım. Unutmayın ki, MaxPool2D katmanı **Yoğunlaştırma** adımıdır.



Gördüğünüz gibi, ReLU fonksiyonunu (Algılama) uyguladıktan sonra, öznitelik haritasında çok fazla "ölü alan" oluşur; yani, yalnızca 0'lardan oluşan geniş alanlar (görüntüdeki siyah bölgeler). Bu 0 aktivasyonlarını ağın tamamı boyunca taşımak, çok fazla yararlı bilgi eklemeden modelin boyutunu artıracaktır. Bunun yerine, öznitelik haritasını yalnızca en yararlı kısmını - özniteliğin kendisini - koruyacak şekilde **yoğunlaştırmak** isteriz.

Aslında **maksimum havuzlama** tam olarak bunu yapar. Maksimum havuzlama, orijinal öznitelik haritasındaki bir aktivasyon yamasını alır ve bu yamadaki **maksimum aktivasyon** ile değiştirir.



ReLU aktivasyonundan sonra uygulandığında, maksimum havuzlama, öznitelikleri "yoğunlaştırma" etkisine sahiptir. Havuzlama adımı, etkin piksellerin sıfır piksellere oranını artırır.

# Example - Apply Maximum Pooling[¶](https://www.kaggle.com/code/ryanholbrook/maximum-pooling#Example---Apply-Maximum-Pooling)

"Lesson 2" örneğinde yaptığımız öznitelik çıkarma işlemine "yoğunlaştırma" adımını ekleyelim. Bu sonraki gizli hücre bizi kaldığımız yere geri götürecek.

import tensorflow as tf

import matplotlib.pyplot as plt

import warnings

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*

*# Read image*

image\_path = '../input/computer-vision-resources/car\_feature.jpg'

image = tf.io.read\_file(image\_path)

image = tf.io.decode\_jpeg(image)

*# Define kernel*

kernel = tf.constant([

[-1, -1, -1],

[-1, 8, -1],

[-1, -1, -1],

], dtype=tf.float32)

*# Reformat for batch compatibility.*

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

image = tf.expand\_dims(image, axis=0)

kernel = tf.reshape(kernel, [\*kernel.shape, 1, 1])

*# Filter step*

image\_filter = tf.nn.conv2d(

input=image,

filters=kernel,

*# we'll talk about these two in the next lesson!*

strides=1,

padding='SAME'

)

*# Detect step*

image\_detect = tf.nn.relu(image\_filter)

*# Show what we have so far*

plt.figure(figsize=(12, 6))

plt.subplot(131)

plt.imshow(tf.squeeze(image), cmap='gray')

plt.axis('off')

plt.title('Input')

plt.subplot(132)

plt.imshow(tf.squeeze(image\_filter))

plt.axis('off')

plt.title('Filter')

plt.subplot(133)

plt.imshow(tf.squeeze(image\_detect))

plt.axis('off')

plt.title('Detect')

plt.show();

Pooling adımını uygulamak için **tf.nn** modülündeki fonksiyonlardan bir diğerini, **tf.nn.pool**'u kullanacağız. Bu, model oluştururken kullandığınız **MaxPool2D** katmanıyla aynı işi yapan bir Python fonksiyonudur, ancak basit bir fonksiyon olduğu için doğrudan kullanımı daha kolaydır.

import tensorflow as tf

image\_condense = tf.nn.pool(

input=image\_detect, *# image in the Detect step above*

window\_shape=(2, 2),

pooling\_type='MAX',

*# we'll see what these do in the next lesson!*

strides=(2, 2),

padding='SAME',

)

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

plt.imshow(tf.squeeze(image\_condense))

plt.axis('off')

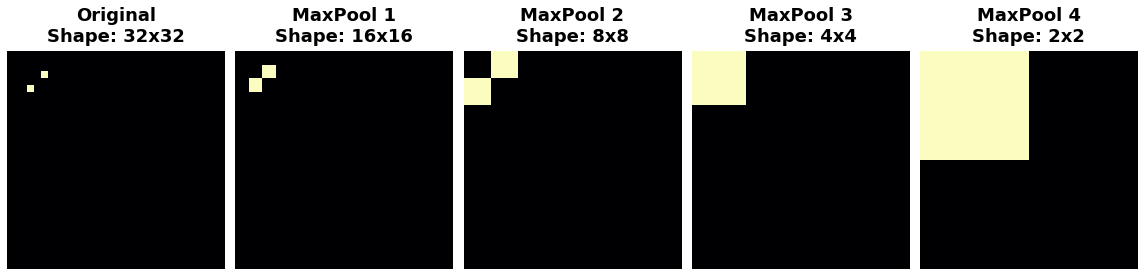
plt.show();

Çok güzel! Umarım, birleştirme adımının, görüntüyü en aktif pikseller etrafında yoğunlaştırarak özelliği nasıl yoğunlaştırabildiğini görebiliyorsunuzdur.

# Translation Invariance[¶](https://www.kaggle.com/code/ryanholbrook/maximum-pooling#Translation-Invariance)

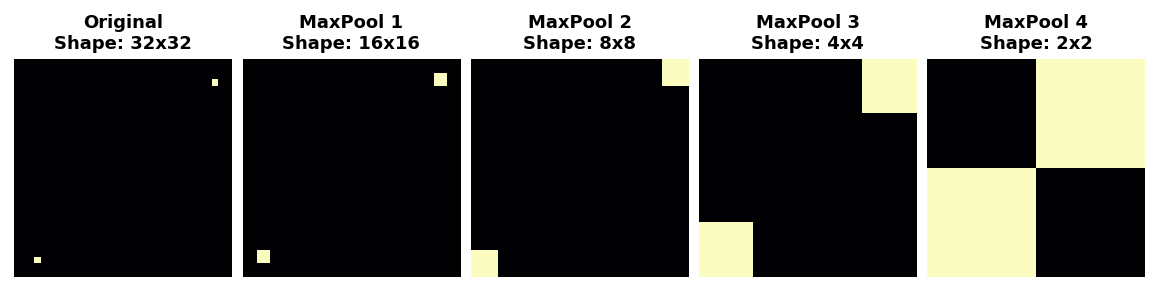
Sıfır piksellere "önemsiz" dedik. Bu, hiçbir bilgi taşımadıkları anlamına mı geliyor? Aslında, sıfır pikseller **konumsal bilgi** taşır. Boş alan, özniteliği görüntü içinde konumlandırmaya devam eder. **MaxPool2D** bu piksellerin bazılarını kaldırdığında, öznitelik haritasındaki konumsal bilginin bir kısmını da kaldırmış olur. Bu, bir evrişimli ağa **öteleme değişmezliği** adı verilen bir özellik kazandırır. Bu, maksimum havuzlama kullanan bir evrişimli ağın, öznitelikleri görüntüdeki **konumlarına** göre ayırt etme eğiliminde olmayacağı anlamına gelir. ("Öteleme", bir şeyin konumunu döndürmeden veya şeklini veya boyutunu değiştirmeden değiştirmenin matematiksel terimidir.)

Aşağıdaki öznitelik haritasına tekrar tekrar maksimum havuzlama uyguladığımızda ne olacağını izleyin.



Orijinal görüntüdeki iki nokta, tekrarlanan havuzlama işleminden sonra ayırt edilemez hale geldi. Başka bir deyişle, havuzlama, konumsal bilgilerinin bir kısmını yok etti. Ağ, öznitelik haritalarında artık bu noktaları ayırt edemediği için, orijinal görüntüde de ayırt edemez: konumdaki bu farklılığa karşı **değişmez** hale geldi.

Aslında, havuzlama bir ağda yalnızca **küçük mesafelerdeki** öteleme değişmezliğini yaratır, tıpkı görüntüdeki iki nokta gibi. Uzakta başlayan öznitelikler, havuzlamadan sonra da ayrı kalacaktır; konumsal bilginin **yalnızca bir kısmı** kaybolur, hepsi değil.



Özelliklerin konumlarındaki küçük farklılıklara karşı bu değişmezlik, bir görüntü sınıflandırıcısının sahip olması güzel bir özelliktir. Sırf bakış açısı veya kadraj farklılıkları nedeniyle, aynı türden bir özellik orijinal görüntünün çeşitli yerlerinde konumlanabilir, ancak biz yine de sınıflandırıcının bunların aynı olduğunu tanımasını isteriz. Bu değişmezlik ağın içine **yerleşik** olduğu için, eğitim için çok daha az veri kullanarak işi halledebiliriz: Artık ona bu farkı göz ardı etmeyi öğretmek zorunda kalmayız. Bu, evrişimsel ağlara, yalnızca yoğun katmanlara sahip bir ağa göre büyük bir verimlilik avantajı sağlar. (6. Derste, **Veri Büyütme** ile değişmezliği bedavadan elde etmenin başka bir yolunu göreceksiniz!)

# Conclusion[¶](https://www.kaggle.com/code/ryanholbrook/maximum-pooling#Conclusion)

Bu derste, özellik çıkarma işleminin son adımı olan MaxPool2D ile yoğunlaştırmayı öğrendik. 4. Ders'te, kayan pencerelerle evrişim ve havuzlama tartışmamızı tamamlayacağız.