import numpy as np

from itertools import product

def show\_kernel(kernel, label=True, digits=None, text\_size=28):

*# Format kernel*

kernel = np.array(kernel)

if digits **is** **not** None:

kernel = kernel.round(digits)

*# Plot kernel*

cmap = plt.get\_cmap('Blues\_r')

plt.imshow(kernel, cmap=cmap)

rows, cols = kernel.shape

thresh = (kernel.max()+kernel.min())/2

*# Optionally, add value labels*

if label:

for i, j **in** product(range(rows), range(cols)):

val = kernel[i, j]

color = cmap(0) if val > thresh else cmap(255)

plt.text(j, i, val,

color=color, size=text\_size,

horizontalalignment='center', verticalalignment='center')

plt.xticks([])

plt.yticks([])

Introduction

Son derste, evrişimli bir sınıflandırıcının iki bölümden oluştuğunu görmüştük: evrişimsel bir **baz** ve yoğun katmanlardan oluşan bir **baş** kısmı. Baz kısmının görevinin, baş kısmının görüntüyü sınıflandırmak için kullanacağı görsel öznitelikleri görüntüden çıkarmak olduğunu öğrenmiştik.

Önümüzdeki birkaç derste, evrişimsel bir görüntü sınıflandırıcısının baz kısmında genellikle bulacağınız en önemli iki katman türünü öğreneceğiz. Bunlar, **ReLU aktivasyonlu evrişimsel katman** ve **maksimum havuzlama katmanıdır**. 5. Derste, bu katmanları öznitelik çıkarmayı gerçekleştiren bloklar halinde birleştirerek kendi evrişimli ağınızı (convnet) nasıl tasarlayacağınızı öğreneceksiniz.

Bu ders, ReLU aktivasyon fonksiyonuna sahip evrişimsel katman hakkındadır.

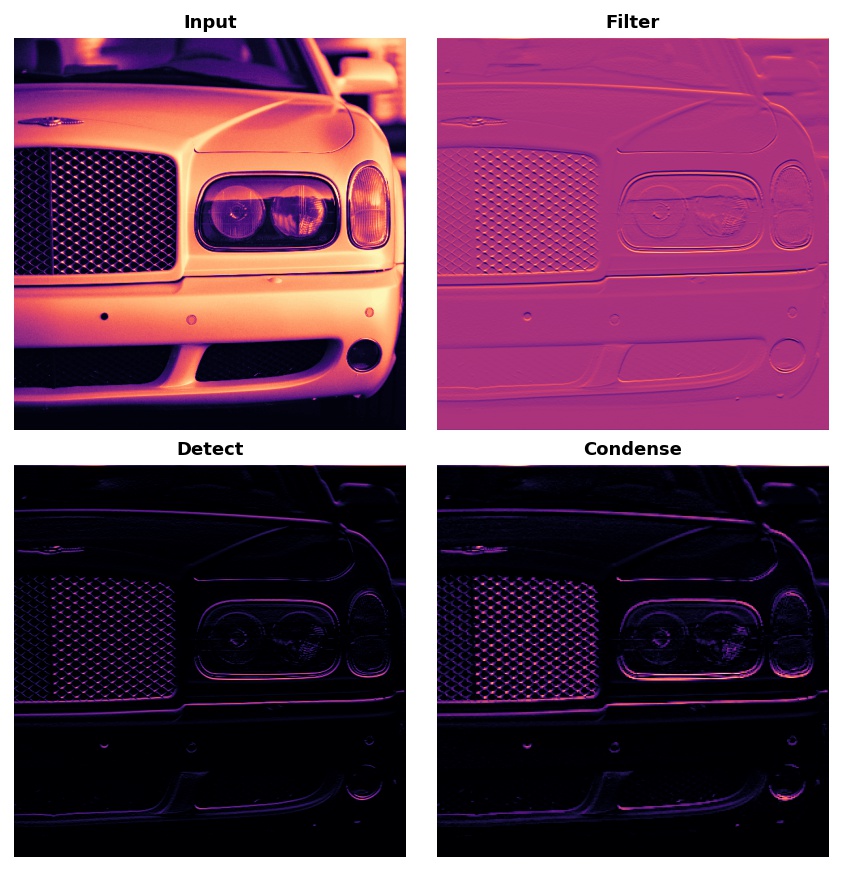
# Feature Extraction[¶](https://www.kaggle.com/code/ryanholbrook/convolution-and-relu" \l "Feature-Extraction" \t "_self)

Evrişimin (convolution) detaylarına girmeden önce, bu katmanların ağdaki amacını tartışalım. Bu üç işlemin (evrişim, ReLU ve maksimum havuzlama) öznitelik çıkarma sürecini uygulamak için nasıl kullanıldığını göreceğiz.

Baz tarafından gerçekleştirilen **öznitelik çıkarma** işlemi, üç temel operasyondan oluşur:

* Bir görüntüyü belirli bir öznitelik için **filtrelemek** (evrişim)
* Filtrelenmiş görüntü içindeki bu özniteliği **tespit etmek** (ReLU)
* Öznitelikleri daha belirgin hale getirmek için görüntüyü **yoğunlaştırmak** (maksimum havuzlama)

Aşağıdaki şekil bu süreci göstermektedir. Bu üç işlemin, orijinal görüntünün belirli bir özelliğini (bu örnekte yatay çizgiler) nasıl izole edebildiğini görebilirsiniz.



Özellik çıkarma işleminin üç adımı.

Ağ, genellikle tek bir görüntü üzerinde paralel olarak birkaç çıkarma işlemi gerçekleştirir. Modern ağlarda, tabandaki son katmanın 1000'den fazla benzersiz görsel özellik üretmesi alışılmadık bir durum değildir.

# Filter with Convolution[¶](https://www.kaggle.com/code/ryanholbrook/convolution-and-relu" \l "Filter-with-Convolution" \t "_self)

Bir evrişim katmanı filtreleme adımını gerçekleştirir. Keras modelinde bir evrişim katmanını şu şekilde tanımlayabilirsiniz:

from tensorflow import keras

from tensorflow.keras import layers

model = keras.Sequential([

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

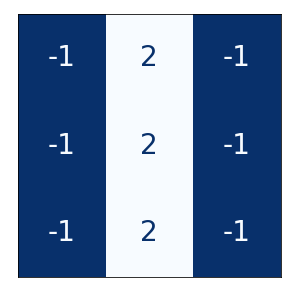
*# More layers follow*

])

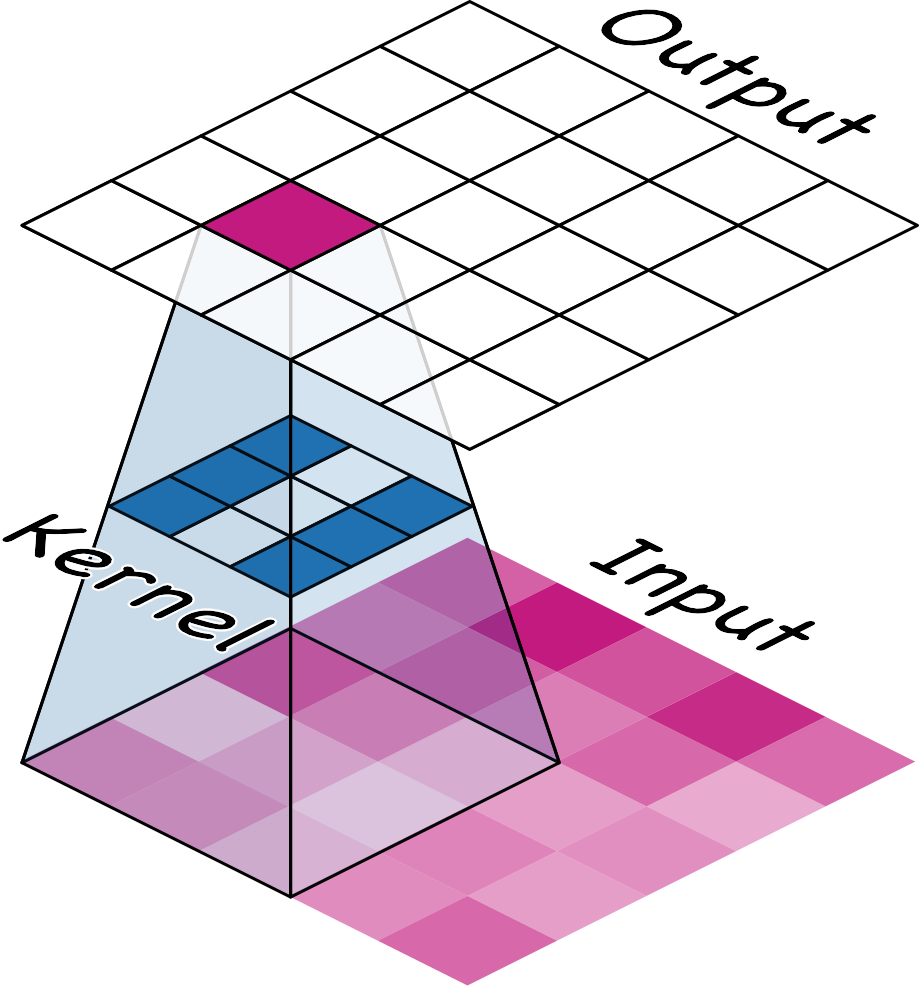
Bu parametreleri, katmanın ağırlıkları ve aktivasyonlarıyla olan ilişkilerine bakarak anlayabiliriz. Şimdi bunu yapalım.

## **Weights**

Bir evrişimli ağın (convnet) eğitim sırasında öğrendiği **ağırlıklar**, esas olarak evrişim katmanlarında bulunur. Bu ağırlıklara **çekirdek** (kernel) adını veriyoruz. Onları küçük diziler olarak temsil edebiliriz:



Bir çekirdek (kernel), bir görüntüyü tarayarak ve piksel değerlerinin **ağırlıklı bir toplamını** üreterek çalışır. Bu şekilde, bir çekirdek bir nevi polarize bir lens gibi davranır; belirli bilgi desenlerini vurgular veya önemini azaltır.

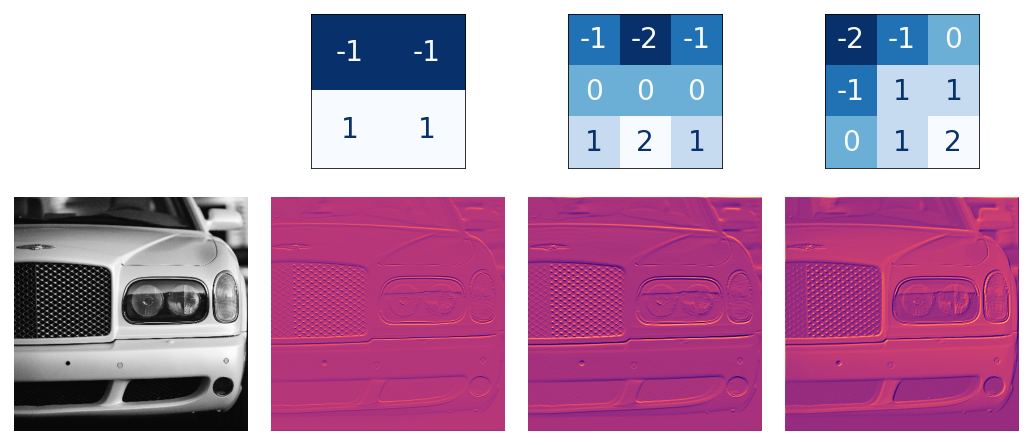


Çekirdekler (kernels), bir evrişim katmanının kendisinden sonra gelen katmana nasıl bağlandığını tanımlar. Yukarıdaki çekirdek, çıktıdaki her bir nöronu girişteki dokuz nörona bağlayacaktır. **kernel\_size** ile çekirdeklerin boyutlarını ayarlayarak, evrişimli ağa bu bağlantıları nasıl kuracağını söylersiniz. Çoğu zaman, bir çekirdeğin boyutları tek sayılardan oluşur (örneğin **kernel\_size=(3, 3)** veya **(5, 5)**) böylece merkezde tek bir piksel bulunur, ancak bu bir zorunluluk değildir.

Bir evrişim katmanındaki çekirdekler, oluşturduğu özniteliklerin türlerini belirler. Eğitim sırasında, evrişimli bir ağ, sınıflandırma problemini çözmek için hangi özniteliklere ihtiyacı olduğunu öğrenmeye çalışır. Bu da çekirdekleri için en iyi değerleri bulmak anlamına gelir.

## **Activations[¶](https://www.kaggle.com/code/ryanholbrook/convolution-and-relu" \l "Activations" \t "_self)**

Ağdaki **aktivasyonlara** **öznitelik haritaları** (feature maps) diyoruz. Bunlar, bir filtreyi bir görüntüye uyguladığımızda elde ettiğimiz sonuçlardır; çekirdeğin çıkardığı görsel öznitelikleri içerirler. Aşağıda, ürettikleri öznitelik haritaları ile birlikte resmedilmiş birkaç çekirdek görülmektedir.



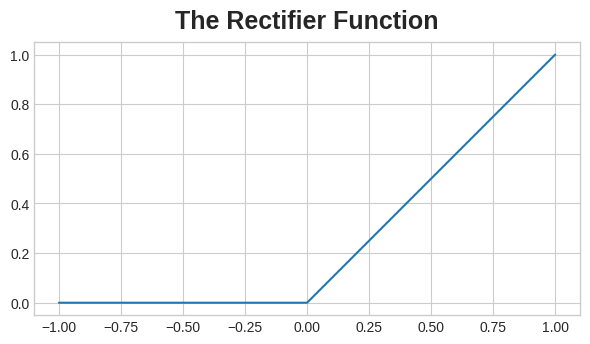
Kernels and features.

Çekirdekteki (kernel) sayıların deseninden, hangi tür öznitelik haritalarını oluşturduğunu anlayabilirsiniz. Genel olarak, bir evrişimin girdilerinde vurguladığı şey, çekirdekteki **pozitif** sayıların şekliyle eşleşir. Yukarıdaki soldaki ve ortadaki çekirdekler, her ikisi de yatay şekilleri filtreleyecektir.

**filters** parametresi ile, evrişim katmanına çıktı olarak kaç tane öznitelik haritası oluşturmasını istediğinizi söylersiniz.

# Detect with ReLU[¶](https://www.kaggle.com/code/ryanholbrook/convolution-and-relu" \l "Detect-with-ReLU" \t "_self)

Filtreleme işleminden sonra, öznitelik haritaları aktivasyon fonksiyonundan geçer. **Doğrultucu fonksiyonunun** (rectifier function) grafiği şöyledir:



Doğrultucu fonksiyonunun grafiği, negatif kısmı 0'a "düzeltilmiş" bir çizgiye benzer.

Doğrultucu (rectifier) eklenmiş bir nörona **doğrultulmuş doğrusal birim** (rectified linear unit) denir. Bu nedenle, doğrultucu fonksiyona **ReLU aktivasyonu** veya sadece ReLU fonksiyonu da diyebiliriz.

ReLU aktivasyonu, kendi **Activation** katmanında tanımlanabilir, ancak çoğunlukla **Conv2D**'nin aktivasyon fonksiyonu olarak dahil edilir.

model = keras.Sequential([

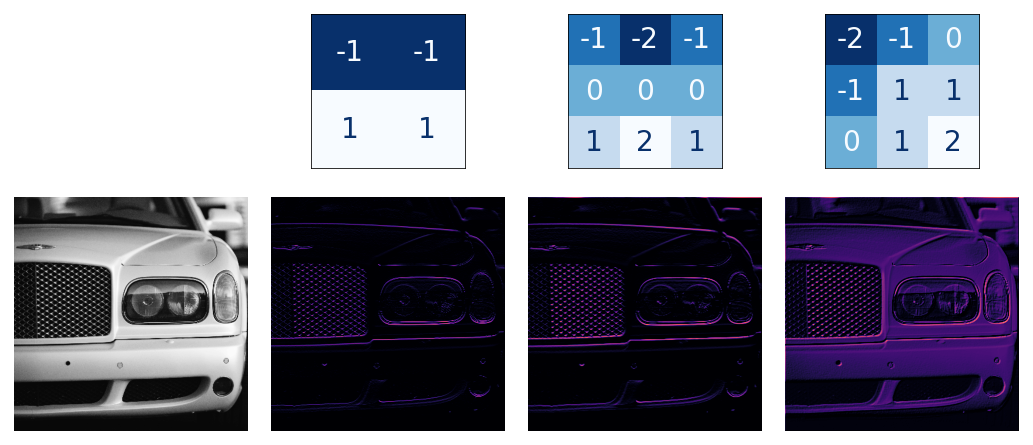
layers.Conv2D(filters=64, kernel\_size=3, activation='relu')

*# More layers follow*

])

Aktivasyon fonksiyonunu, piksel değerlerini belirli bir önem ölçüsüne göre puanlayan bir unsur olarak düşünebilirsiniz. ReLU aktivasyonu, negatif değerlerin önemli olmadığını söyler ve bu nedenle onları 0'a ayarlar. ("Önemsiz olan her şey eşit derecede önemsizdir.")

İşte yukarıdaki özellik haritalarına uygulanan ReLU. Özellikleri nasıl başarıyla izole ettiğine dikkat edin.



Diğer aktivasyon fonksiyonları gibi, ReLU fonksiyonu da **doğrusal olmayan** bir fonksiyondur. Esasen bu, ağdaki tüm katmanların toplam etkisinin, yalnızca etkileri bir araya ekleyerek elde edeceğinizden farklı hale geldiği anlamına gelir. (Bu, tek bir katmanla elde edebileceğiniz sonuçla aynı olurdu.) Doğrusallık dışı (nonlinearity), özniteliklerin ağın derinliklerine doğru ilerledikçe ilginç şekillerde birleşmesini sağlar. (Bu "öznitelik birleşmesini" 5. Derste daha ayrıntılı olarak inceleyeceğiz.)

# Example - Apply Convolution and ReLU[¶](https://www.kaggle.com/code/ryanholbrook/convolution-and-relu" \l "Example---Apply-Convolution-and-ReLU" \t "_self)

Bu örnekte, evrişimli ağların "perde arkasında" neler yaptığını daha iyi anlamak için çıkarma işlemini kendimiz yapacağız.

Bu örnek için kullanacağımız görsel şu şekilde:

import tensorflow as tf

import matplotlib.pyplot as plt

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

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

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

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

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

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

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

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

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

plt.axis('off')

plt.show();

Filtreleme adımı için bir **çekirdek** (kernel) tanımlayacak ve ardından onu **evrişimle** uygulayacağız. Bu durumdaki çekirdek, bir "kenar algılama" çekirdeğidir. Tıpkı Numpy'da np.array ile bir dizi tanımlar gibi, tf.constant ile de tanımlayabilirsiniz. Bu, TensorFlow'un kullandığı türden bir **tensör** oluşturur.

import tensorflow as tf

kernel = tf.constant([

[-1, -1, -1],

[-1, 8, -1],

[-1, -1, -1],

])

plt.figure(figsize=(3, 3))

show\_kernel(kernel)

TensorFlow, **tf.nn** modülünde sinir ağları tarafından gerçekleştirilen birçok yaygın işlemi içerir. Kullanacağımız iki tanesi **conv2d** ve **relu**. Bunlar, Keras katmanlarının fonksiyon versiyonlarıdır.

Aşağıdaki gizli hücre, bir şeyleri TensorFlow ile uyumlu hale getirmek için bazı yeniden biçimlendirmeler yapıyor. Ayrıntılar bu örnek için önemli değildir.

*# 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])

kernel = tf.cast(kernel, dtype=tf.float32)

Şimdi çekirdeğimizi uygulayalım ve ne olacağını görelim.

image\_filter = tf.nn.conv2d(

input=image,

filters=kernel,

*# we'll talk about these two in lesson 4!*

strides=1,

padding='SAME',

)

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

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

plt.axis('off')

plt.show();

Sıradaki adım, ReLU fonksiyonu ile gerçekleştirilen **algılama** adımıdır. Bu fonksiyon, ayarlanacak herhangi bir parametresi olmadığı için evrişimden çok daha basittir.

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

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

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

plt.axis('off')

plt.show();

Ve böylece bir **öznitelik haritası** oluşturduk! Baş kısmı, sınıflandırma problemini çözmek için bunun gibi görüntüleri kullanır. Belirli özniteliklerin **Otomobillere**, diğerlerinin ise **Kamyonlara** daha karakteristik olduğunu hayal edebiliriz. Bir evrişimli ağın (convnet) eğitim sırasındaki görevi, bu öznitelikleri bulabilecek çekirdekler oluşturmaktır.