DERİNLEMESİNE BİLGİSAYARLI GÖRÜ

Dr. Öğr. Üyesi Fatma AKALIN

Bilgisayarlı görme, derin öğrenmenin en eski ve en büyük başarı öyküsüdür. Her gün Google Fotoğraflar, Google görsel arama, YouTube, kamera uygulamalarındaki video filtreleri, OCR yazılımı ve çok daha fazlası aracılığıyla derin görüş modelleriyle etkileşime giriyorsunuz. Bu modeller aynı zamanda otonom sürüş, robot bilimi, yapay zeka destekli tıbbi teşhis, otonom perakende ödeme sistemleri ve hatta otonom tarım alanlarındaki en ileri araştırmaların da merkezinde yer alıyor. Bilgisayarlı görme, 2011 ile 2015 yılları arasında derin öğrenmenin ilk yükselişine yol açan sorun alanıdır. Evrişimsel sinir ağları adı verilen bir tür derin öğrenme modeli, görüntü sınıflandırma konusunda oldukça iyi sonuçlar almaya başladı.

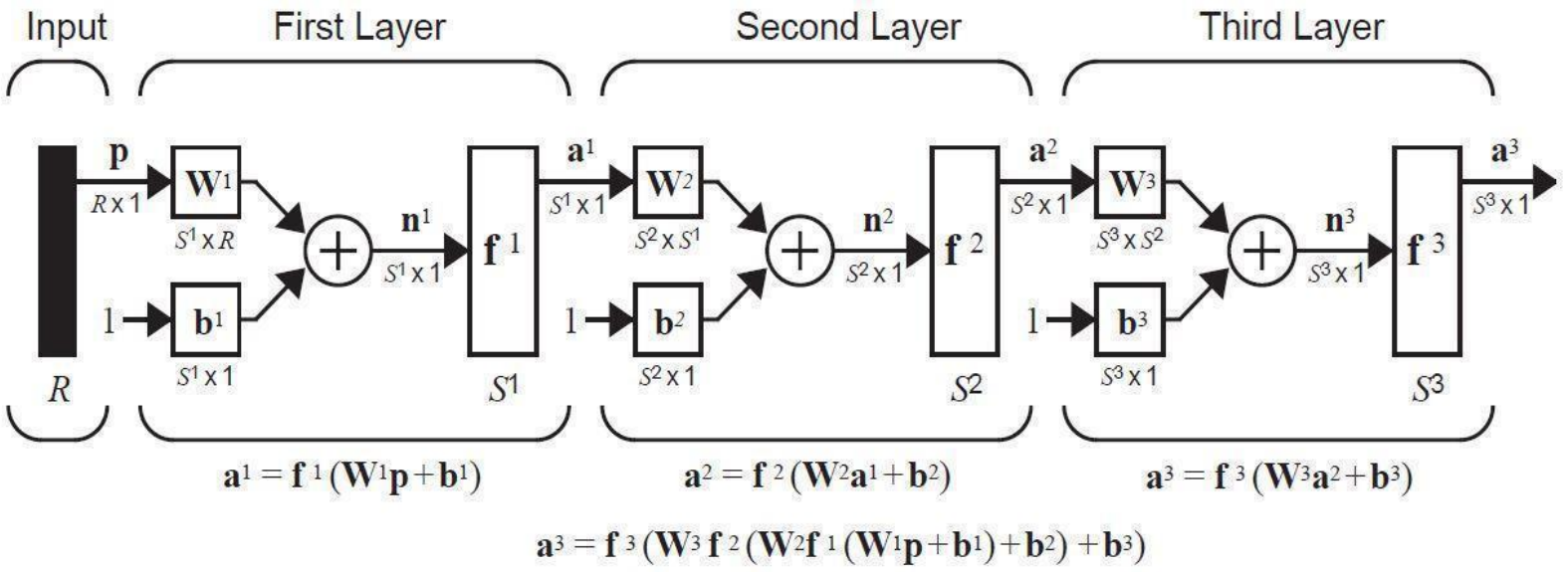
Şimdi evrensel olarak kullanılan derin öğrenme modeli türü olan convnet'ler (evrişimli sinir ağları) tanıtılacaktır.

Derin Sinir Ağları

Derin öğrenme, **karmaşık ilişkilerin** elde edilmesini mümkün kılar. **Öğrenme kalıpları ve bağlantılar oluşturarak güçlü bir bilgi çıkarma** süreci oluşturur. Bu yapıda her katmanın çıktısı bir sonraki katmana girdi olarak verilir ve her katmanın girişine doğrusal olmayan bir dönüşüm uygulanır. **Hiyerarşik tasarımın nedeni** temel özellikleri çıkarmaktır. Aynı zamanda **doğrusal olmayan dönüşümlerle daha derin çıkarımlar sağlanarak soyut ve karmaşık temsiller** elde edilir. Evrişimsel Sinir Ağları, çoğunlukla görüntüleri sınıflandırmak amacıyla kullanılan derin öğrenme algoritmaları içerisinde yer alır.

M. M. Najafabadi, et all., 'Deep learning applications and challenges in big data analytics', J. Big Data, 2015, v. 2, no. 1, pp. 1–21, p.4.

Evrişimsel Sinir Ağları, çoğunlukla görüntüleri sınıflandırmak amacıyla kullanılan derin öğrenme algoritmaları içerisinde yer alır. Aynı zamanda evrişimsel sinir ağları adı verilen bir tür derin öğrenme modeli, görüntü sınıflandırma konusunda oldukça iyi sonuçlar almaktadır.



M. M. Najafabadi, et all., 'Deep learning applications and challenges in big data analytics', J. Big Data, 2015, v. 2, no. 1, pp. 1–21, p.4.

EVRİŞİM İŞLEMİ

Evrişim işlemi, evrişimsel sinir ağlarının temel işlemidir. Aslında bu işlem ile işaretler ve sistemler gibi farklı mühendislik derslerinin içerisinde yaptığımız konvolüsyon işlemini gerçekleştiriyoruz. Fakat bu eylemi bilgisayarlı görü kapsamında iki boyutlu yapıyoruz. Çünkü görsellerimiz matris olarak ifade ediliyor ve matrisel işlemler yapabilmek için evrişimi iki boyutta tanımlamamız gerekiyor. Aslında derin öğrenmede GPU(Grafik İşlem Birimlerinin)'nun ön plana çıkmasının sebeplerinden biride budur. Çünkü Grafik İşlem Birimleri ekran kartıdır. Ekranlar matrisler şeklinde ifade edilir. Ekran kartının hızlı olması yada işlem gücünün yüksek olması matris işlemlerini o kadar hızlı yapabildiğiniz anlamına gelmektedir. Bu durum evrişimsel derin öğrenme modelimizin hızlı bir şekilde eğitilebilir hale geldiğini göstermektedir. Aynı zamanda evrişim işleminin (aynı anda toplama da var çarpma da var) yükü çok olduğu için grafik işlem ünitelerine ihtiyacımız olmaktadır.

Ayrıca her katmanda bir takım öznitelikler öğrenildiği için daha çok özelliğin öğrenilmesi hedef nesnenin diğer nesnelerden daha kolay ayrılacağı anlamına da gelmektedir.

SİMÜLASYON

https://cs231n.github.io/convolutional-networks/

UYARI : Renkli görüntüler, Kırmızı-Yeşil-Mavi (RGB) olarak 3 kanaldan meydana gelmektedir. Bu koşulda evrişim işlemi 3 kanal için yapılmaktadır.

https://www.kaggle.com/code/kanncaa1/convolutional-neural-network-cnn-tutorial

Bir örnek ile süreci somutlaştıralım…

https://github.com/fchollet/deep-learning-with-python-notebooks

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 128)         73856

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 10)                11530

=================================================================
Total params: 104,202
Trainable params: 104,202
Non-trainable params: 0
_________________________________________________________________
```

Her Conv2D ve MaxPooling2D katmanının çıktısının, şeklin (yükseklik, genişlik, kanallar) 3. derece tensörü olduğunu görebilirsiniz. Genişlik ve yükseklik boyutları genellikle modelin derinliklerine indikçe küçültülür.

https://github.com/fchollet/deep-learning-with-python-notebooks

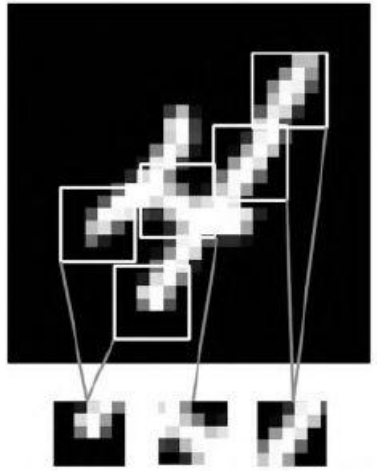
```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 128)         73856

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 10)                11530

=================================================================
Total params: 104,202
Trainable params: 104,202
Non-trainable params: 0
_________________________________________________________________
```

Evrişim İşlemi

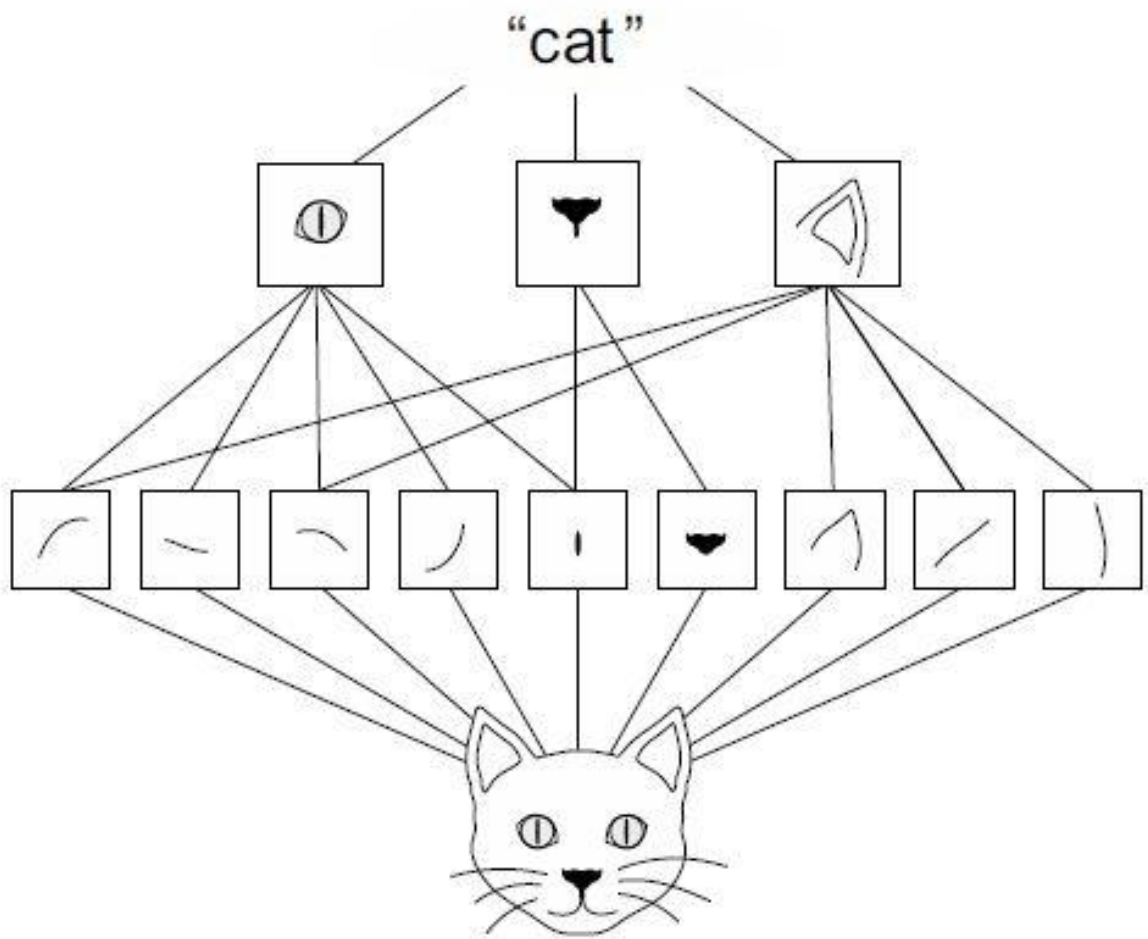
Yoğun bağlantılı katman ile evrişim katmanı arasındaki temel fark şudur: Yoğun katmanlar, giriş özellik uzayındaki küresel modelleri öğrenir (örneğin, bir MNIST rakamı için tüm pikselleri içeren modeller), evrişim katmanları ise yerel modelleri öğrenir



Convnet’lerin iki ilginç özelliği;

1-Bir resmin sağ alt köşesindeki belirli bir modeli öğrendikten sonra, bir convnet onu her yerde tanıyabilir: örneğin sol üst köşede.

2-Desenlerin mekansal hiyerarşilerini öğrenebilirler. Birinci evrişim katmanı, kenarlar gibi küçük yerel desenleri öğrenecek, ikinci evrişim katmanı, birinci katmanların özelliklerinden oluşan daha büyük desenleri öğrenecek ve bu şekilde devam edecektir(Bu durum bir sonraki sayfada tasvir edilmektedir). Bu, convnet'lerin giderek karmaşıklaşan ve soyutlaşan görsel kavramları verimli bir şekilde öğrenmesine olanak tanır, çünkü görsel dünya temelde mekansal olarak hiyerarşiktir.


“cat”

Evrişimler, iki uzamsal eksene (yükseklik ve genişlik) ve bir derinlik eksenine (kanal ekseni de denir) sahip, özellik haritaları adı verilen 3. derece tensörler üzerinde çalışır. Bir RGB görüntüsü için derinlik ekseninin boyutu 3’tür. Çünkü görüntünün üç renk kanalı vardır: kırmızı, yeşil ve mavi. MNIST rakamları gibi siyah beyaz bir resim için derinlik 1'dir (gri düzeyleri). Evrişim işlemi, giriş özellik haritasından yamaları çıkarır ve aynı dönüşümü bu yamaların tümüne uygulayarak bir çıktı özellik haritası üretir..

Filtreler, giriş verilerinin belirli yönlerini kodlar: yüksek düzeyde, tek bir filtre, örneğin "girişte bir yüzün varlığı" kavramını kodlayabilir.

MNIST örneğinde, ilk evrişim katmanı (28, 28, 1) boyutunda bir özellik haritası alır ve (26, 26, 32) boyutunda bir özellik haritası çıkarır: girişi üzerinden 32 filtre hesaplar. Bu 32 çıkış kanalının her biri, giriş üzerindeki filtrenin yanıt haritası olan ve girişteki farklı konumlardaki filtre deseninin yanıtını gösteren 26 × 26'lık bir değer ızgarası içerir

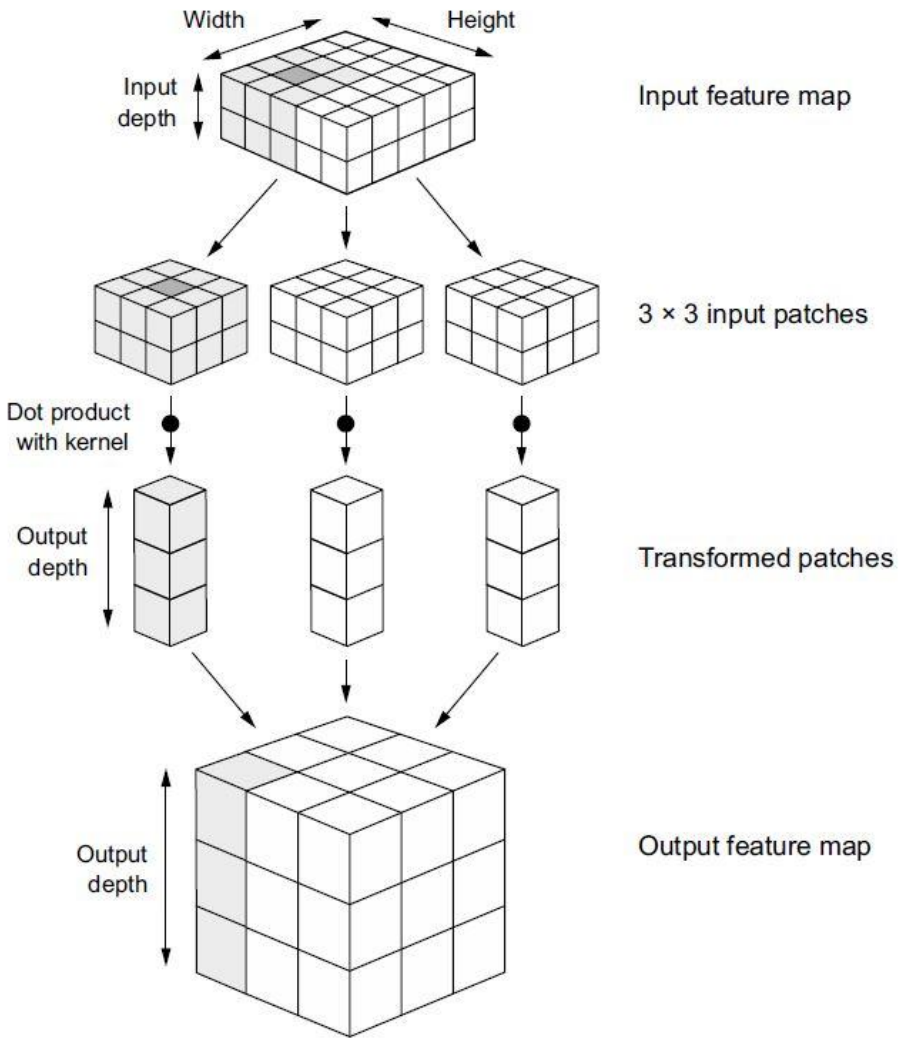


Figure 8.4 How convolution works

Çıkış genişliği ve yüksekliğinin iki nedenden dolayı giriş genişliğinden ve yüksekliğinden farklı olabileceğini unutmayın:

-Giriş özellik haritasının padding işlevselliğiyle doldurulması

-Stride kullanımı

Bu kavramlara daha derinlemesine bakalım.

UNDERSTANDING BORDER EFFECTS AND PADDING

5 × 5'lik bir özellik haritasını düşünün (toplam 25 döşeme). Etrafında 3 × 3'lük bir pencere oluşturup 3 × 3'lük bir ızgara oluşturabileceğiniz yalnızca 9 döşeme vardır. Dolayısıyla çıktı özellik haritası 3 × 3 olacaktır.

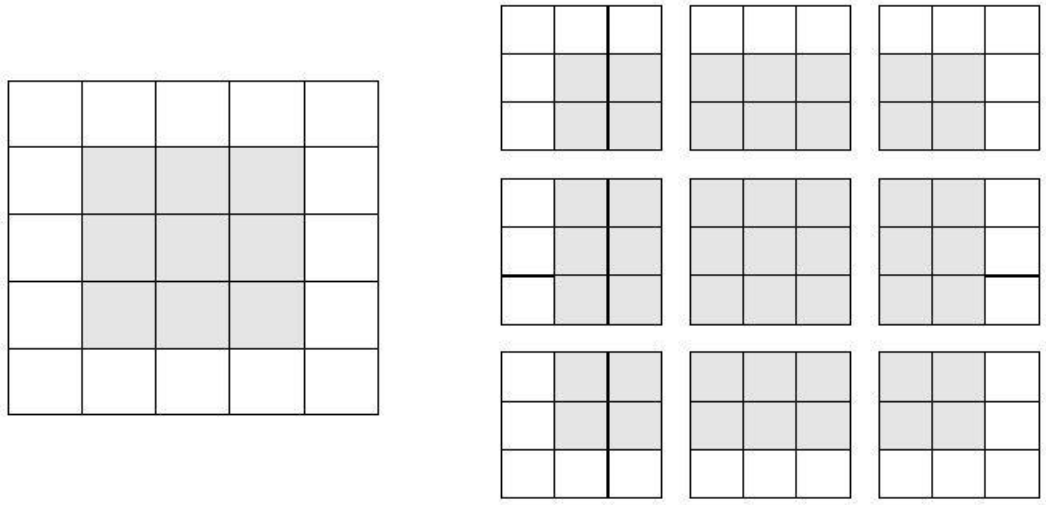


Figure 8.5 Valid locations of 3 × 3 patches in a 5 × 5 input feature map

Örneğimiz üzerinde bu işlevselliği irdelediğimizde; 28 × 28 giriş ile, ilk evrişim katmanından sonra 26 × 26'lık çıktı elde **ederiz (Dersin akışında her bir detay irdelenecektir).** Girişle aynı uzamsal boyutlara sahip bir çıkış özelliği haritası elde etmek istiyorsanız dolgu(padding) kullanmalısınız.

(n+2p-f/s)+1
formülü
yardımıyla çıkış
özelliği
haritasının
boyutunu elde
edebiliriz.

f=filtre boyutu
p=piksel doldurma
s=adım kaydırma
n=Filtre(Kanal) Sayısı

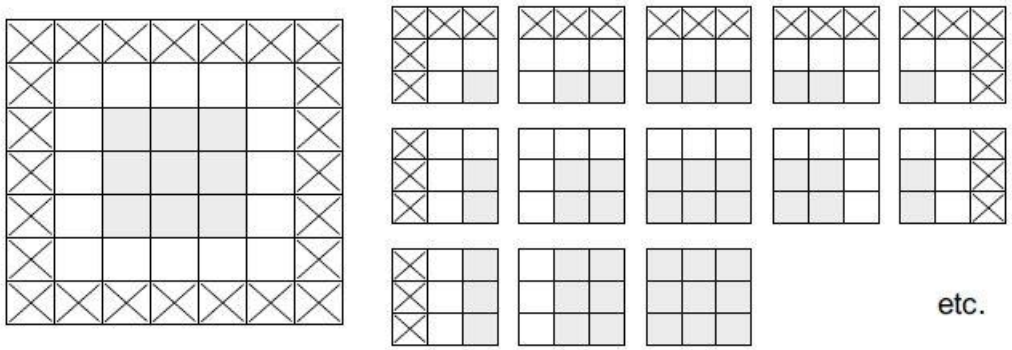


Figure 8.6 Padding a 5 × 5 input in order to be able to extract 25 3 × 3 patches

Dolgu, her giriş döşemesinin etrafına merkezi evrişim pencerelerinin sığdırılmasını mümkün kılmak için giriş özellik haritasının her iki tarafına uygun sayıda satır ve sütun eklenmesinden oluşur. 3×3 pencere için sağa bir sütun, sola bir sütun, üste bir satır ve alta bir satır eklersiniz. 5 × 5 pencere için iki satır eklersiniz

UYARI

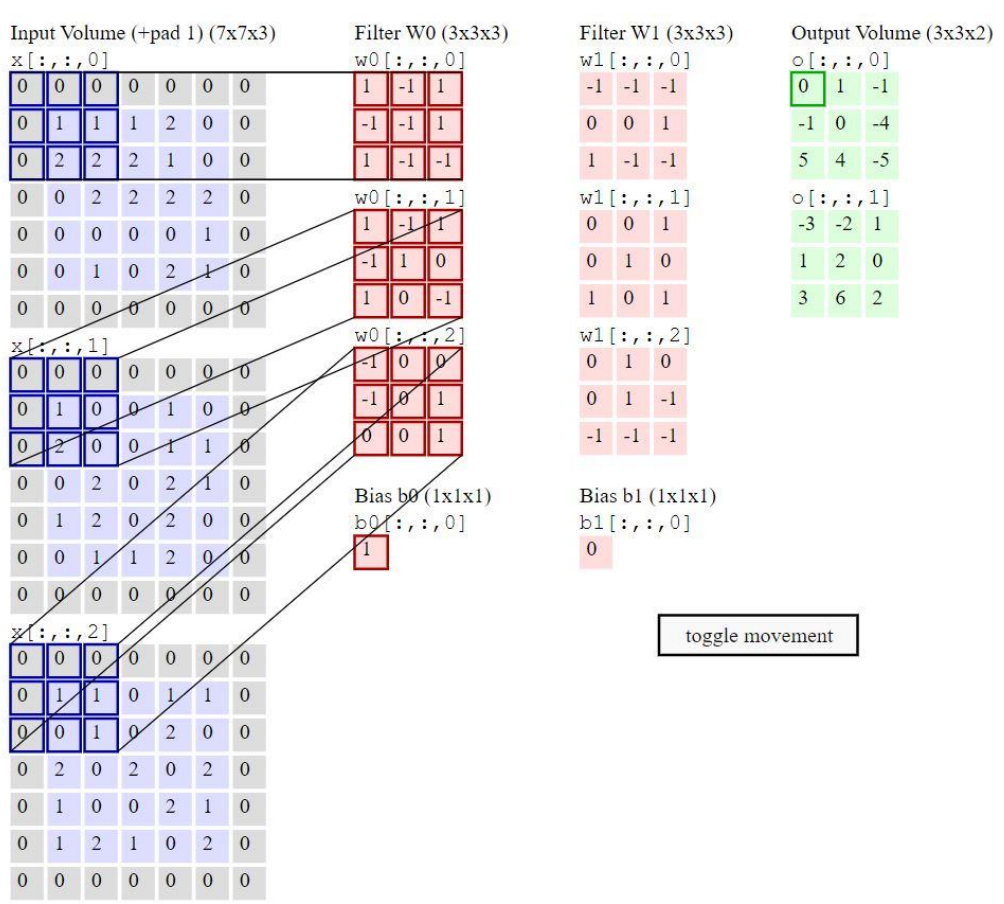
In Conv2D layers, padding is configurable via the padding argument, which takes two values: "valid", which means no padding (only valid window locations will be used), and "same", which means “pad in such a way as to have an output with the same width and height as the input.” The padding argument defaults to "valid".

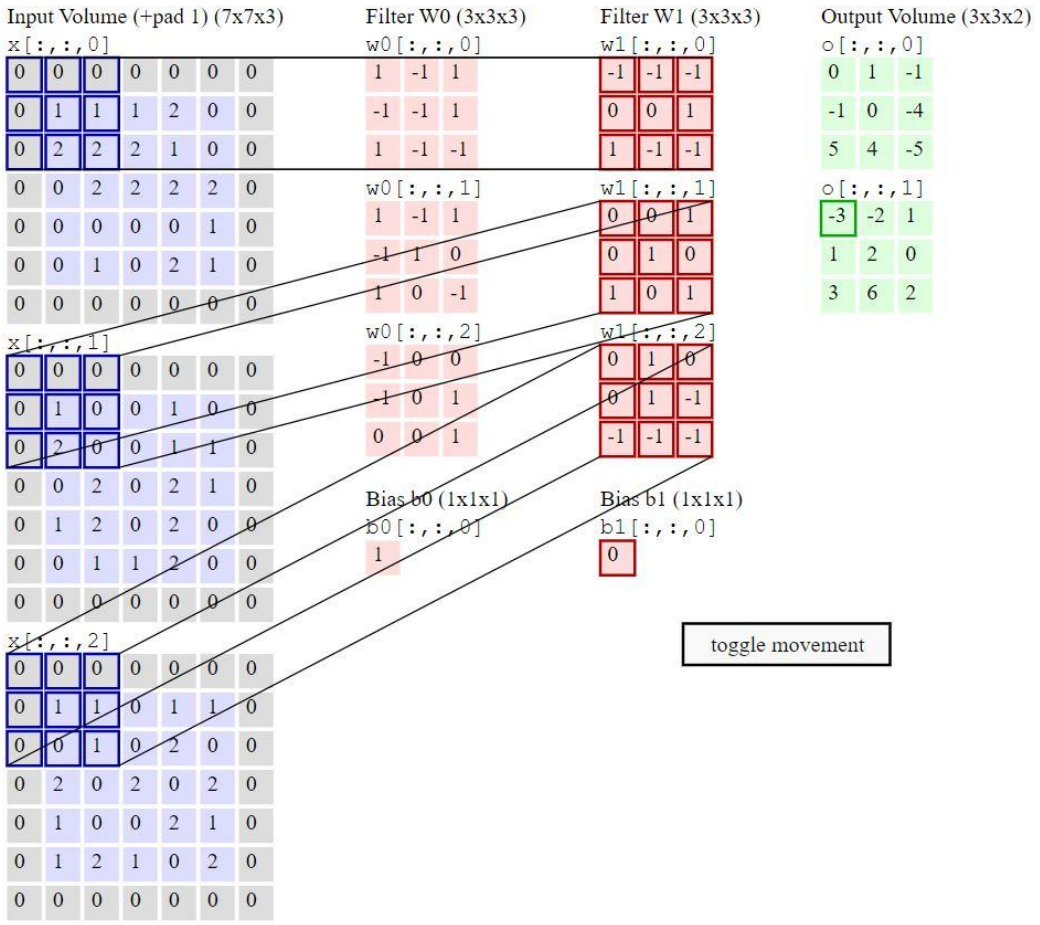
Uyarı kullanılan terimlerin ingilizce olarak öğrenilmesi gerekliliğinden dolayı dolayı türkçe olarak tercüme edilmemiştir.

UNDERSTANDING CONVOLUTION STRIDES

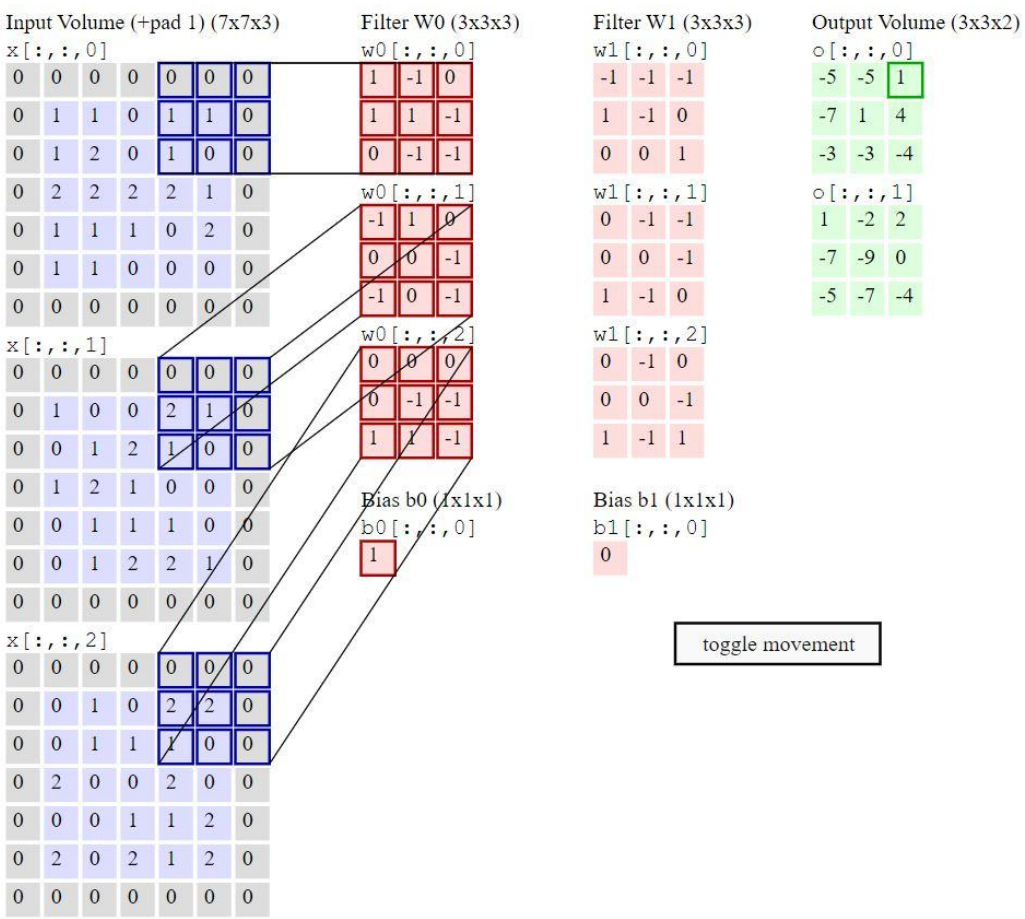
Çıktı boyutunu etkileyebilecek diğer faktör ise adım kavramıdır. Şu ana kadarki evrişim tanımımız, evrişim pencerelerinin merkez döşemelerinin hepsinin bitişik olduğunu varsaymıştır. Ancak birbirini takip eden iki pencere arasındaki mesafe, evrişimin bir parametresidir, adım adı verilir ve varsayılan değer 1'dir.

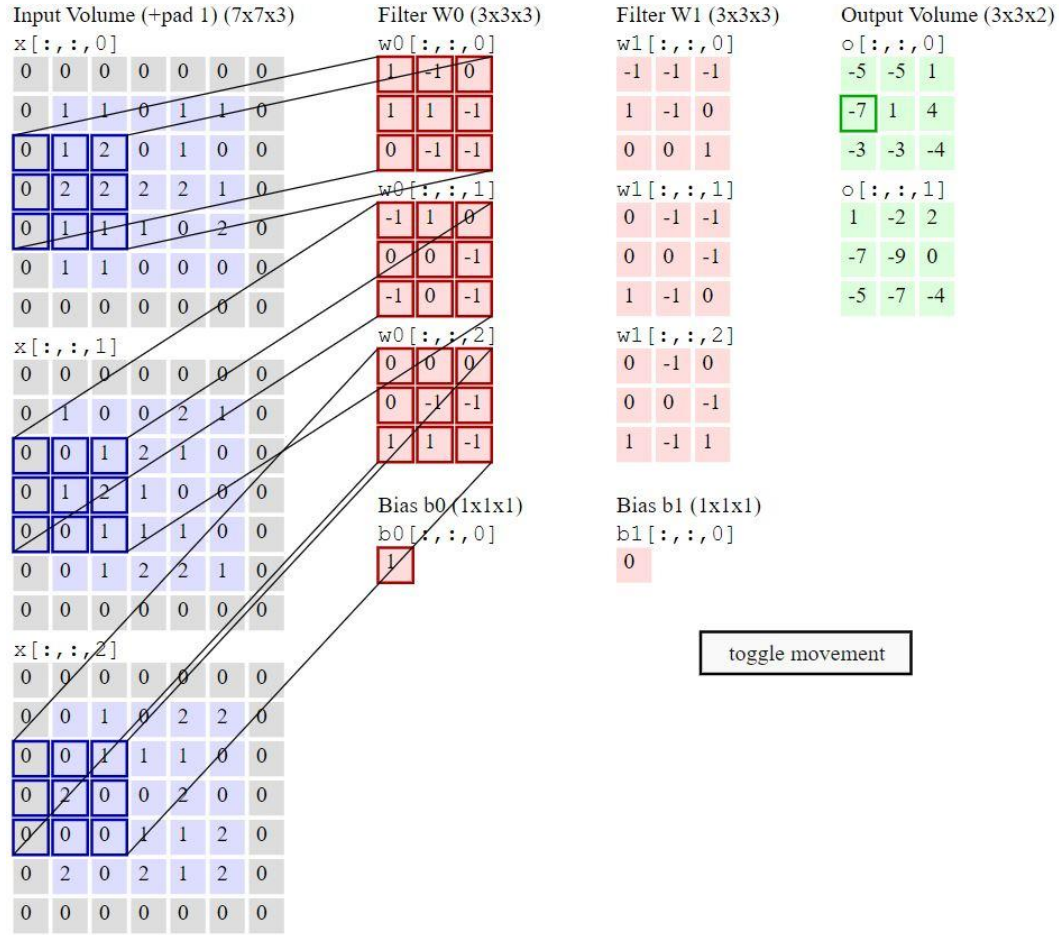
Farklı adıma sahip evrişimlerin olması mümkündür: Adım sayısının 2 olarak kullanılması, özellik haritasının genişliğinin ve yüksekliğinin 2 faktörüyle alt örneklenmesi anlamına gelir. Adımlı evrişimler sınıflandırma modellerinde nadiren kullanılsa da bazı model türleri için kullanışlı olurlar.





https://cs231n.github.io/convolutional-networks/





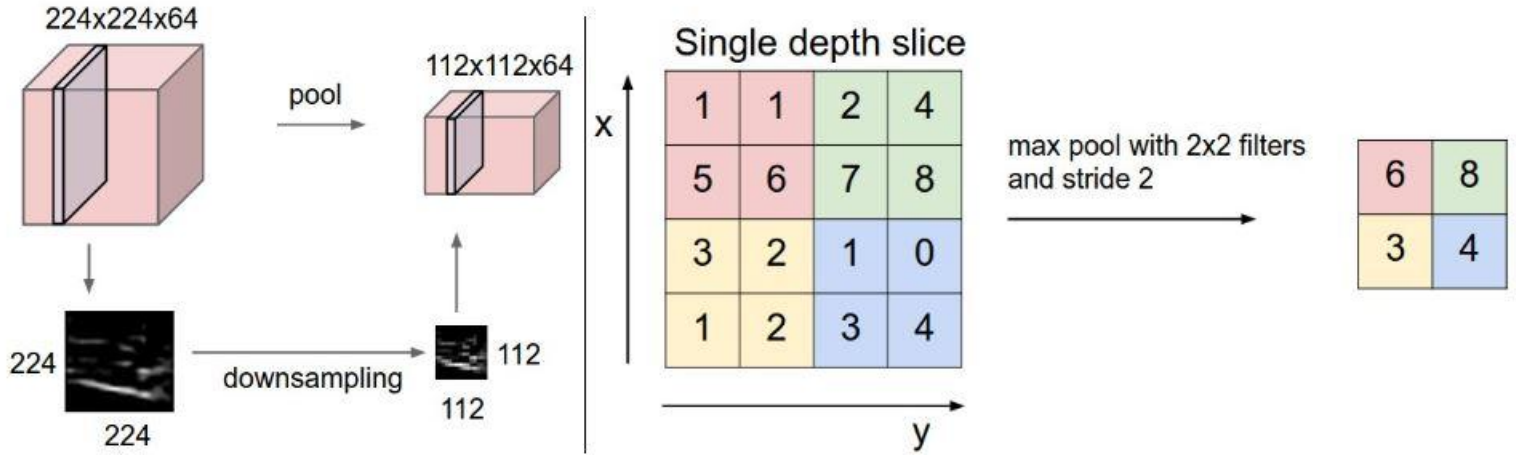
https://cs231n.github.io/convolutional-networks/

NOT

Sınıflandırma modellerinde, adımlar yerine, maksimum havuzlama işlemini kullanma eğilimi öne çıkmaktadır. Gelin buna daha derinlemesine bakalım.

The Max-Pooling Operation

Convnet örneğinde, özellik haritalarının boyutunun her MaxPooling2D katmanından sonra yarıya indiğini fark etmiş olabilirsiniz. Örneğin, ilk MaxPooling2D katmanlarından önce özellik haritası 26 × 26'dır, ancak maksimum havuzlama işlemi bunu yarıya indirerek 13 × 13'e indirir.



Dersin akışında farklı alt örnekleme türleri de sunulacaktır.

Özellik haritalarını neden bu şekilde (max-pooling) alt örnekleme yapmalısınız? Neden maksimum havuzlama katmanlarını kaldırıp oldukça büyük özellik haritalarını baştan sona tutmuyorsunuz?

Listing 8.5 An incorrectly structured convnet missing its max-pooling layers

```
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model_no_max_pool = keras.Model(inputs=inputs, outputs=outputs)
```

Here's a summary of the model:

```
>>> model_no_max_pool.summary()
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 22, 22, 128)       73856
_________________________________________________________________
flatten_1 (Flatten)          (None, 61952)             0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                619530
=================================================================
Total params: 712,202
Trainable params: 712,202
Non-trainable params: 0
_________________________________________________________________
```

Nihai özellik haritası örnek başına 22 × 22 × 128 = 61.952 toplam katsayıya sahiptir. Bu çok büyük. Üstüne 10 boyutunda bir yoğun katman yapıştırmak için onu düzleştirdiğinizde, bu katmanın yarım milyonun üzerinde parametresi olacaktır. Bu durum, bu kadar küçük bir model için çok büyüktür ve aşırı uyumla sonuçlanacaktır. Kısacası, alt örneklemeyi kullanmanın nedeni, işlenecek özellik haritası katsayılarının sayısını azaltmak ve aynı zamanda ardışık hale getirerek uzamsal filtre hiyerarşilerini teşvik etmektir.

Farz edelim ki 3×3 pencereler, yalnızca 7×7 pencerelerden gelen bilgileri içerecek bir model tasarımı olsun.

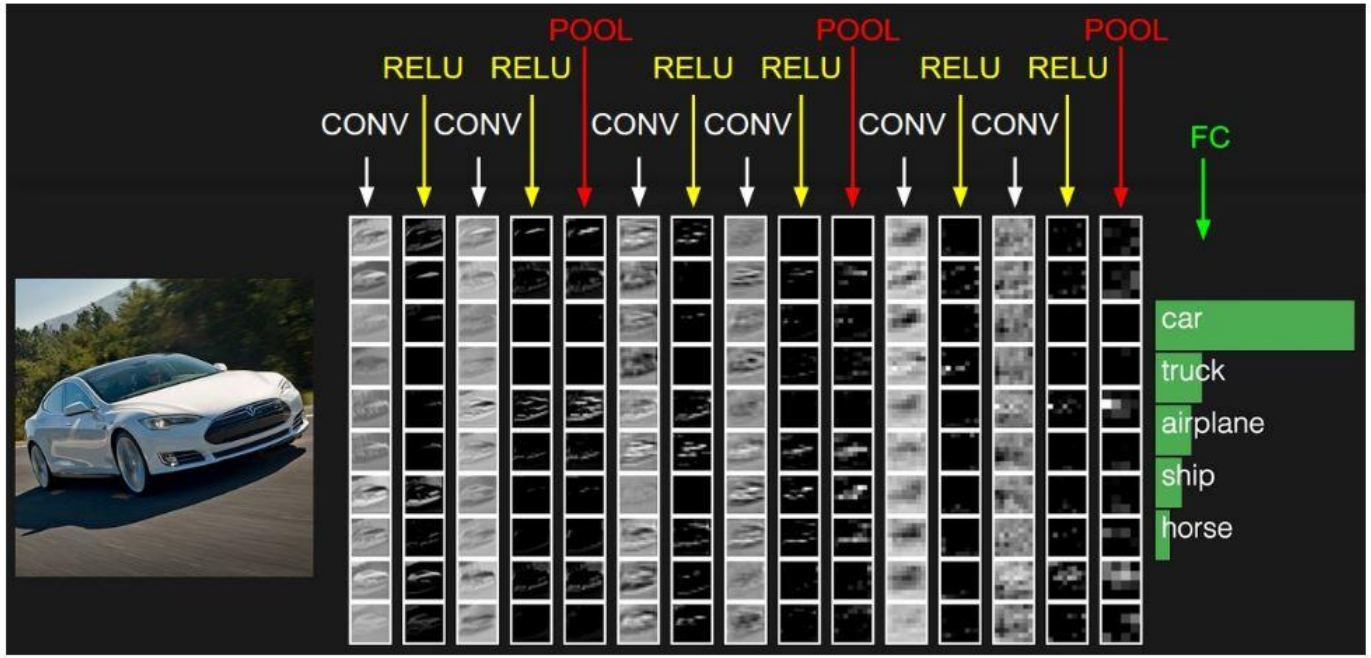
Böyle bir durumda, Convnet tarafından öğrenilen yüksek seviyeli desenler, ilk girişe göre hala çok küçük olacaktır. Bu da rakamları sınıflandırmayı öğrenmek için yeterli olmayabilir (bir rakamı yalnızca 7 × 7 piksellik pencerelerden bakarak tanımayı deneyin!).

Unutulmamalıdır ki girişin bütünlüğü hakkında bilgi içermek için son evrişim katmanındaki özelliklere ihtiyacımız var.

Ayrıca, maksimum havuzlama yerine **ortalama havuzlamayı** kullanabilirsiniz; burada her yerel giriş yaması, yama üzerinde maksimum yerine her kanalın **ortalama değeri** alınarak dönüştürülür. Ancak **maksimum havuzlama** bu alternatif çözümlerden **daha iyi sonuç verme eğilimindedir**. Bunun nedeni, özelliklerin, özellik haritasının farklı döşemeleri üzerindeki bazı model veya kavramların mekansal varlığını kodlama eğiliminde olmasıdır ve farklı özelliklerin **maksimum varlığına bakmak, onların ortalama varlığına bakmaktan daha bilgilendiricidir.**

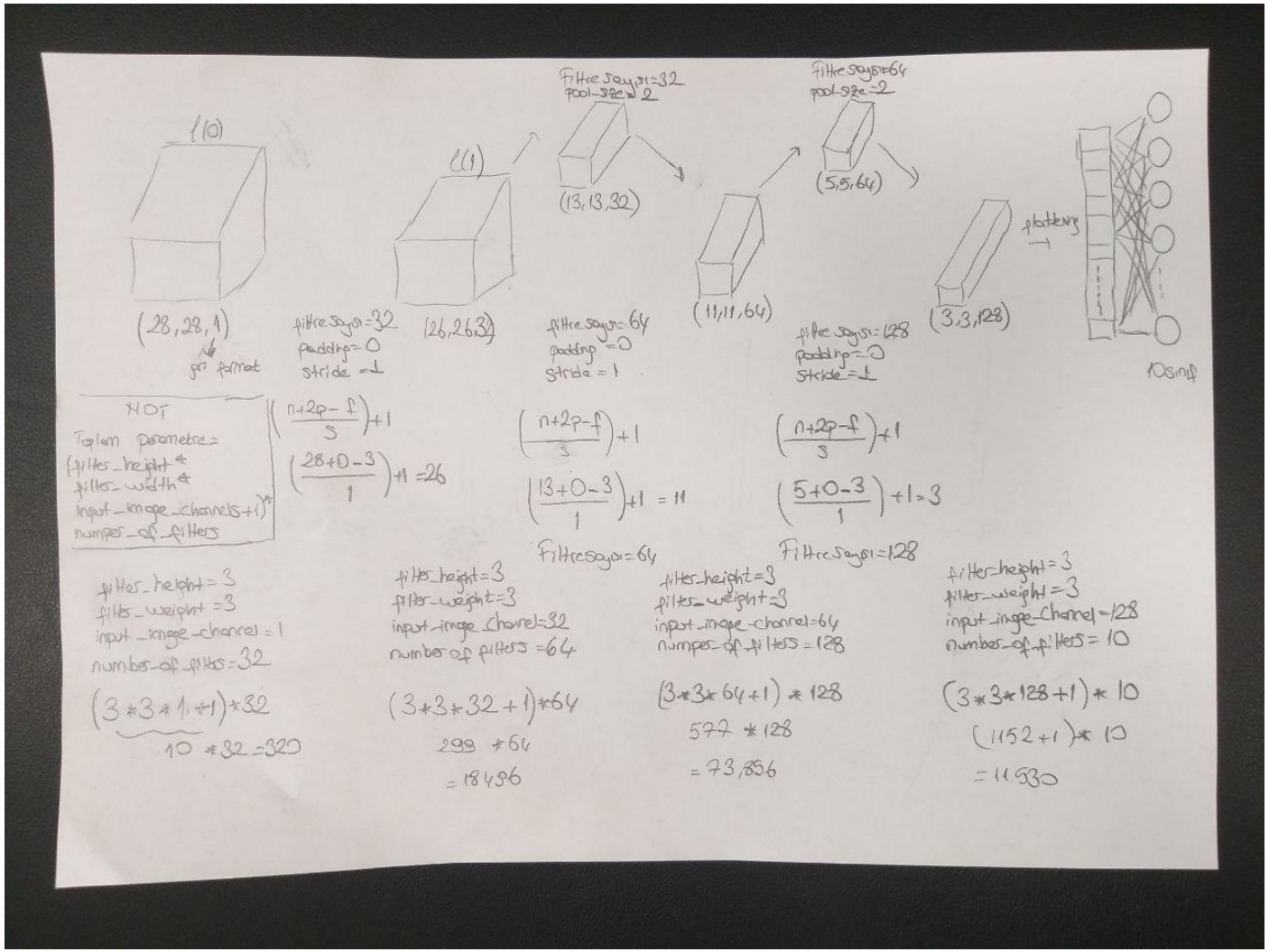
Özetle, en makul alt örnekleme stratejisi ilk önce yoğun özellik haritaları üretmek (**adımsız evrişimler aracılığıyla**) ve **ardından** girdilerin daha seyrek pencerelerine (adımlı evrişimler yoluyla) veya giriş yamalarının ortalamasını almak yerine, özelliklerin küçük yamalar üzerindeki **maksimum aktivasyonuna bakmaktır**.

Yukarıda açıkladığımız gibi, basit bir ConvNet bir dizi katmandan oluşur ve ConvNet'in **her katmanı, türevlenebilir bir fonksiyon aracılığıyla bir aktivasyon hacmini diğerine dönüştürür**. ConvNet mimarileri oluşturmak için üç ana katman türü kullanırız: *Evrişimsel Katman, Havuz Katmanı* ve *Tam Bağlantılı Katman* (tam olarak normal Sinir Ağlarında görüldüğü gibi). Tam bir ConvNet mimarisi oluşturmak için bu katmanları istifleyeceğiz.



https://cs231n.github.io/convolutional-networks/

İlk örneğimize geri dönelim…



https://github.com/fchollet/deep-learning-with-python-notebooks

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 128)         73856

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 10)                11530

=================================================================
Total params: 104,202
Trainable params: 104,202
Non-trainable params: 0
_________________________________________________________________
```

İlk örneğimiz olan deeplearning.ipynb isimli dosyayı detaylıca birlikte inceleyelim..

Başlangıçta...

Introduction to deep learning

https://github.com/fchollet/deep-learning-with-python-notebooks

In [1]:
```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

In [2]:
```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 128)         73856

 flatten (Flatten)           (None, 1152)              0
```

In [3]:
```
from tensorflow.keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype("float32") / 255
model.compile(optimizer="rmsprop",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

```
Epoch 1/5
938/938 [==============================] - 31s 31ms/step - loss: 0.1512 - accuracy: 0.9523
Epoch 2/5
938/938 [==============================] - 31s 34ms/step - loss: 0.0424 - accuracy: 0.9866
```

If your targets are **one-hot encoded**, use `categorical_crossentropy`.

- Examples of one-hot encodings:
  - `[1,0,0]`
  - `[0,1,0]`
  - `[0,0,1]`

But if your targets are **integers**, use `sparse_categorical_crossentropy`.

- Examples of integer encodings (*for the sake of completion*):
  - `1`
  - `2`
  - `3`

https://sanjivgautamofficial.medium.com/categorical-cross-entropy-vs-sparse-categorical-cross-entropy-b6a24de2b7f0

İkinci örneğimiz olan sequentialmodel.ipynb isimli dosyayı birlikte inceleyelim..

Model oluşturma ve daha birçok özellik burada öğrenilecektir.

Üçüncü örneğimiz olan functionalmodel.ipynb isimli dosyayı birlikte inceleyelim..

Early stopping ve daha birçok özellik burada öğrenilecektir.

Dördüncü örneğimiz olan dataaugmentation2.ipynb isimli dosyayı birlikte inceleyelim..

Veri arttırma ve daha birçok özellik burada öğrenilecektir.

Beşinci örneğimiz olan transferlearning.ipynb isimli dosyayı birlikte inceleyelim..

Transfer learning yaklaşımına ilişkin birçok özellik burada öğrenilecektir.

Table 2 Brief overview of CNN architectures

Model	Main finding	Depth	Dataset	Error rate	Input size	Year
AlexNet	Utilizes Dropout and ReLU	8	ImageNet	16.4	227 × 227 × 3	2012
NIN	New layer, called 'mlpconv', utilizes GAP	3	CIFAR-10, CIFAR-100, MNIST	10.41, 35.68, 0.45	32 × 32 × 3	2013
ZfNet	Visualization idea of middle layers	8	ImageNet	11.7	224 × 224 × 3	2014
VGG	Increased depth, small filter size	16, 19	ImageNet	7.3	224 × 224 × 3	2014
GoogLeNet	Increased depth,block concept, different filter size, concatenation concept	22	ImageNet	6.7	224 × 224 × 3	2015
Inception-V3	Utilizes small filtersize, better feature representation	48	ImageNet	3.5	229 × 229 × 3	2015
Highway	Presented the multipath concept	19, 32	CIFAR-10	7.76	32 × 32 × 3	2015
Inception-V4	Divided transform and integration concepts	70	ImageNet	3.08	229 × 229 × 3	2016
ResNet	Robust against overfitting due to symmetry mapping-based skip links	152	ImageNet	3.57	224 × 224 × 3	2016
ResNet	Robust against overfitting due to symmetry mapping-based skip links	152	ImageNet	3.57	224 × 224 × 3	2016
Inception-ResNet-v2	Introduced the concept of residual links	164	ImageNet	3.52	229 × 229 × 3	2016
FractalNet	Introduced the concept of Drop-Path as regularization	40,80	CIFAR-10 CIFAR-100	4.60 18.85	32 × 32 × 3	2016
WideResNet	Decreased the depth and increased the width	28	CIFAR-10 CIFAR-100	3.89 18.85	32 × 32 × 3	2016
Xception	A depthwise convolutionfollowed by a pointwise convolution	71	ImageNet	0.055	229 × 229 × 3	2017
Residual attention neural network	Presented the attention technique	452	CIFAR-10, CIFAR-100	3.90, 20.4	40 × 40 × 3	2017
Squeeze-and-excitation networks	Modeled interdependencies between channels	152	ImageNet	2.25	229 × 229 × 3 224 × 224 × 3 320 × 320 × 3	2017
DenseNet	Blocks of layers; layers connected to each other	201	CIFAR-10, CIFAR-100,ImageNet	3.46, 17.18, 5.54	224 × 224 × 3	2017
Competitive squeeze and excitation network	Both residual and identity mappings utilized to rescale the channel	152	CIFAR-10 CIFAR-100	3.58 18.47	32 × 32 × 3	2018

[6] L. Alzubaidi et al., Review of deep learning: concepts, CNN architectures, challenges, applications, future directions, vol. 8, no. 1. Springer International Publishing, 2021.

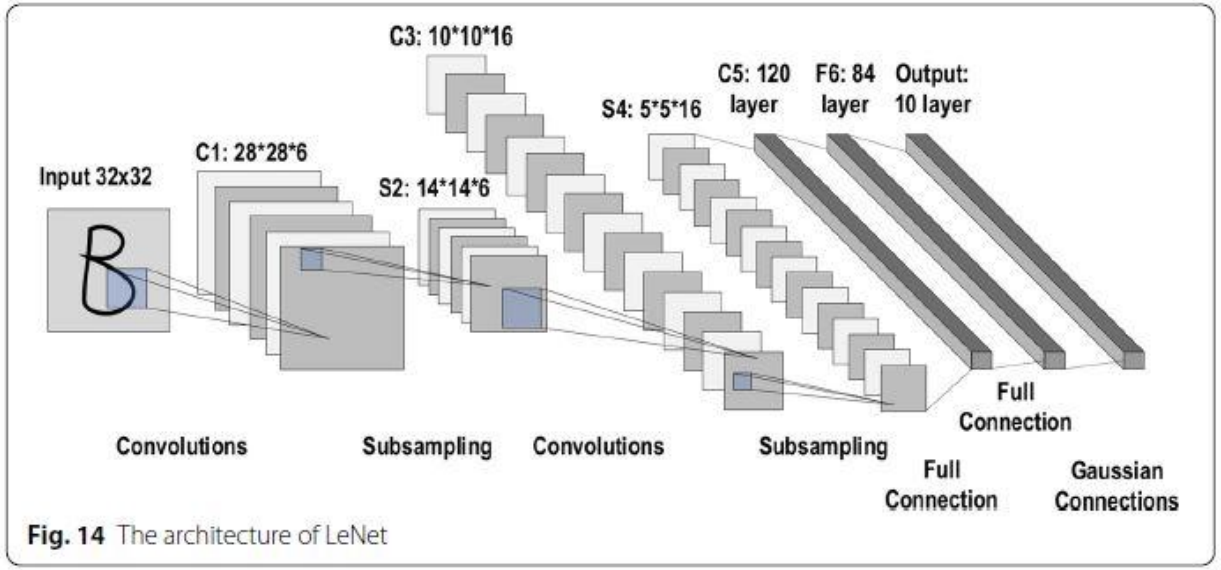


Fig. 14 The architecture of LeNet

Figure 7. The architecture of LeNet[6]

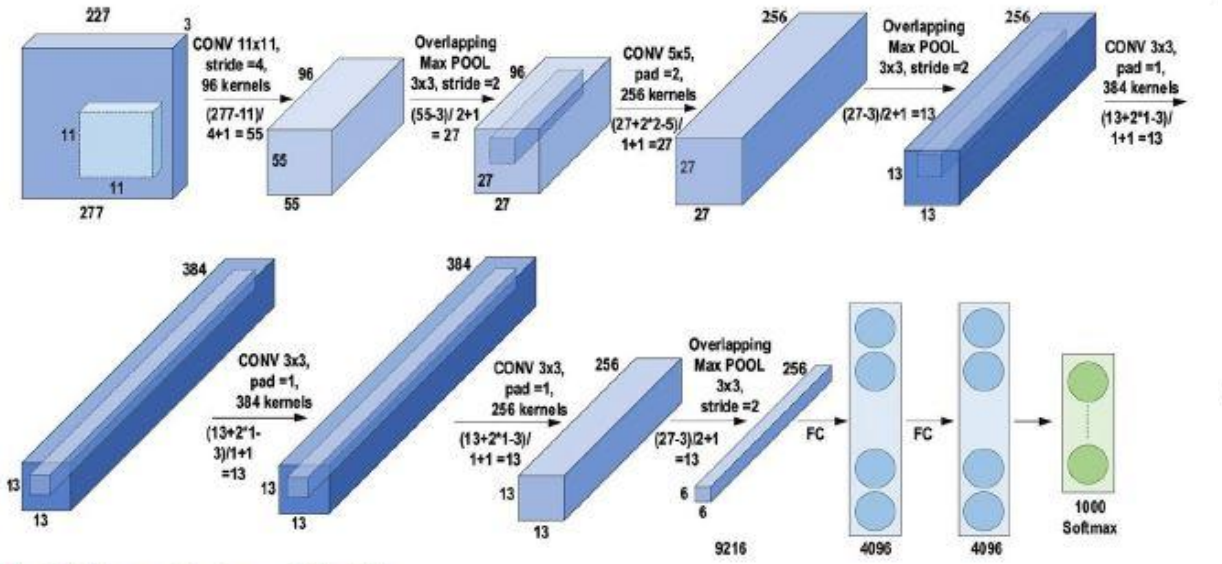


Fig. 15 The architecture of AlexNet

Figure 8. The architecture of AlexNet[6]

[6] L. Alzubaidi et al., Review of deep learning: concepts, CNN architectures, challenges, applications, future directions, vol. 8, no. 1. Springer International Publishing, 2021.

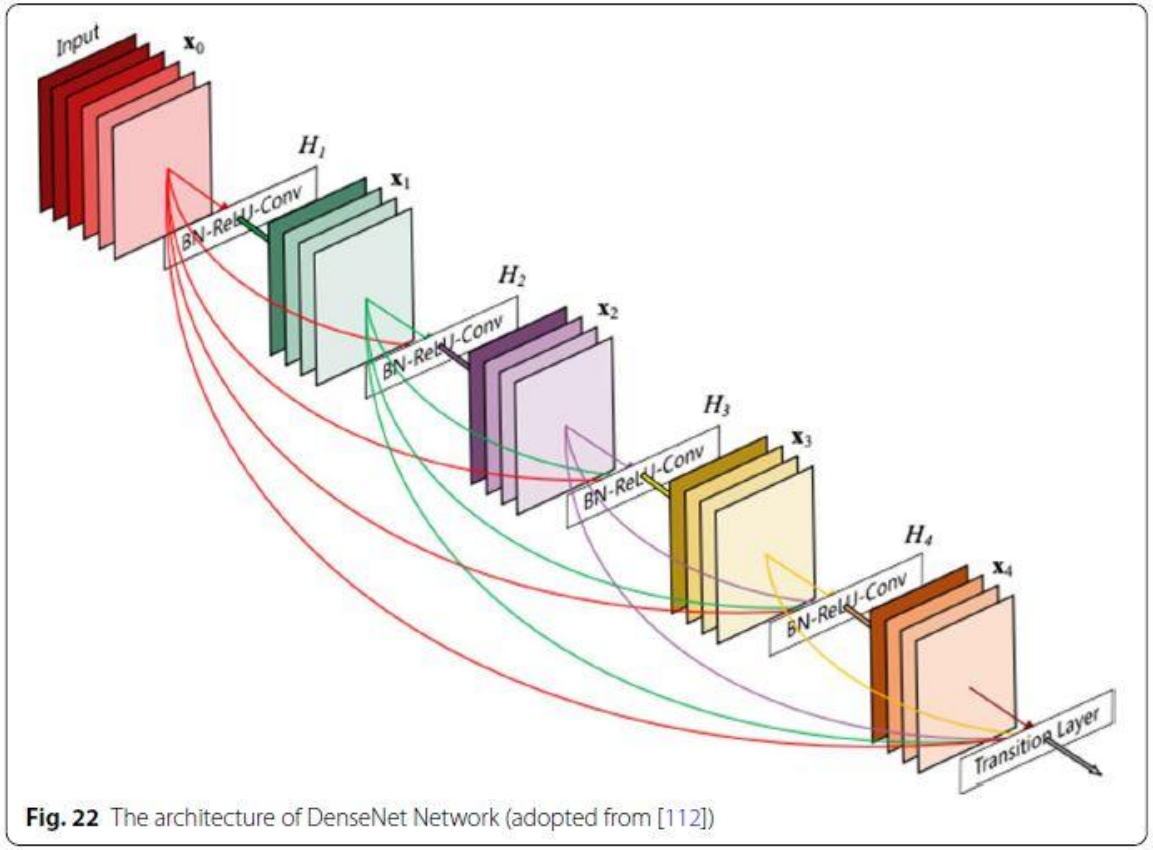


Fig. 22 The architecture of DenseNet Network (adopted from [112])

The architecture of DenseNet Network[6]

Evrişimli sinir ağları **computer vision**, speech processing, face recognition gibi farklı alanlarda uygulanan yapılardır.

BİLGGİSAYARLI GÖRÜ / COMPUTER VISION

Görüntülerden bilgi elde eden yapay sistemlerin arkasındaki teoriyle ilişkili olan bilgisayarlı görme teknolojisindeki gelişmeler hayatın birçok alanına yansıyor. Bilgisayarla görmede nesne tespiti ve bölümleme birçok alanda yaygın olarak kullanılan bir işlevselliktir. Mevcut süreçte nesnelerin net bir şekilde algılanması, çıktıya ilişkin kararı etkileyeceğinden, algılama ve bölütlemenin doğruluğu önem kazanmaktadır. Bu nedenle segmentasyonun doğru bir şekilde gerçekleştirilmesinin ardından sinir ağlarının doğruluğunu artırmak amacıyla çeşitli yapılar geliştirmek ve bunların kombinasyonlarını oluşturmak için farklı yaklaşımlar mevcuttur. [1][8][9].

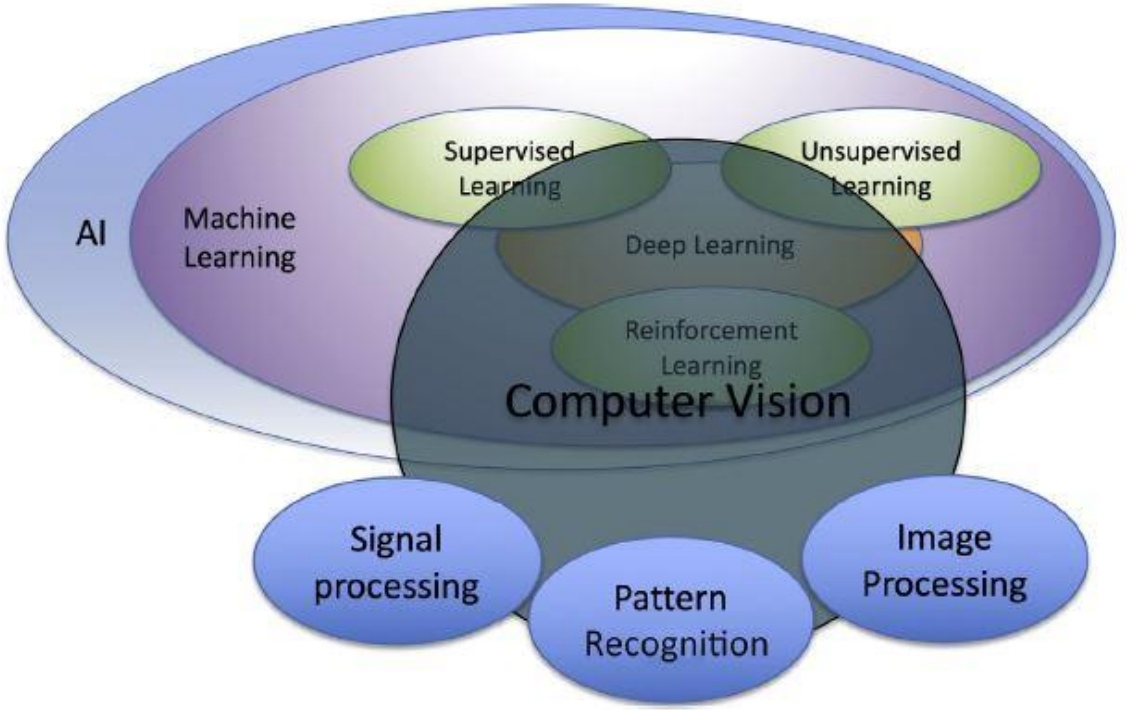


Figure 10. The architecture of DenseNet Network[1]

[1] D.A. Hashimato et.all., “The Role of Artificial Intelligence in Surgery”, Advances in Surgery, 2020, pp. 89-101, p. 90 and 94
[8] Q. Song, et all., ‘Object detection method for grasping robot based on improved yolov5’, Micromachines, 2021, v. 12, no. 11, pp.1-18, p. 1.
[9] A. Bochkovskiy, et all. “YOLOv4: Optimal Speed and Accuracy of Object Detection”, 2020, pp.1-17, p.5-9
[10] https://tr.wikipedia.org/wiki/Bilgisayarl%C4%B1_g%C3%B6r%C3%BC

Segmentasyon

Segmentasyon işlemi, modelin eğitiminden önce gerçekleştirilmesi gereken kritik bir aşamadır. Bu aşamada hedef veriler üzerinde etiketlenecek nesnelerin ana hatlarının doğru bir şekilde belirlenmesi doğru bir sonucun üretilmesi açısından önemlidir. Derin öğrenme teknikleri, (i) semantic segmentation (labels are class-aware); and (ii) instance segmentation (labels are instance aware) şeklinde 2 ayrı segmentasyon tipine sahiptir.

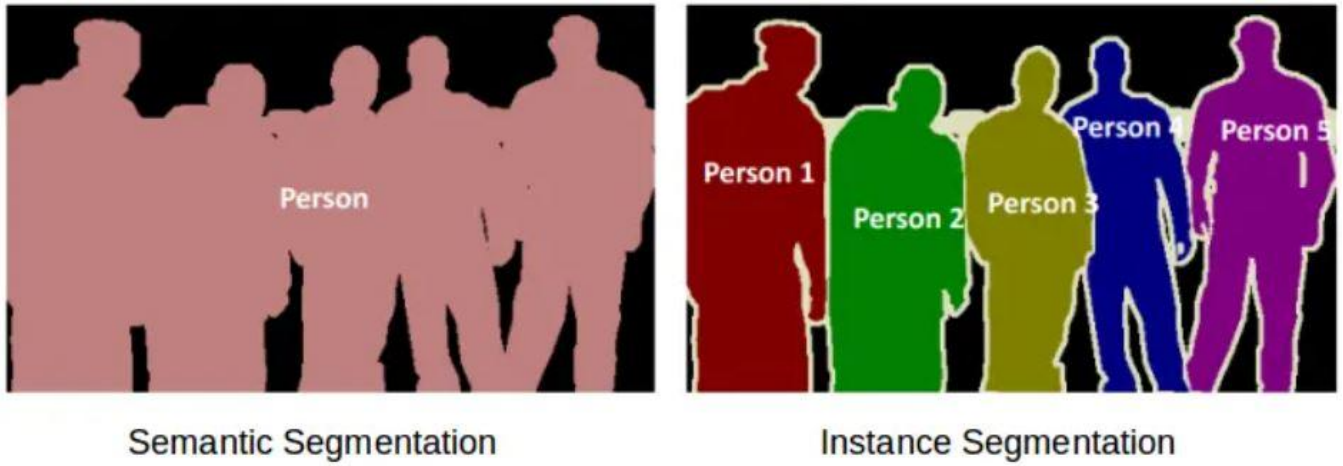


Figure 10. Semantic and Instance segmentation sample[11]

[11] https://teknoloji.org/image-segmentation-goruntu-bolutleme-nedir/

Object Detection

Nesne tanıma, veri kümesindeki görüntüler üzerinde hedef nesneleri bulmak, dikdörtgen sınırlayıcı kutularla göstermek, güven değerini belirtmek ve sınıflandırmak için kullanılan bir yaklaşımdır. Nesne algılama çerçeveleri iki temel yaklaşıma sahiptir. İlk yaklaşımda ilk olrak bölge önerileri oluşturulur ve ardından her bir teklif farklı nesne kategorilerine göre sınıflandırılır. Region proposed temelli yöntemler, başlıca R-CNN , SPP-net , Fast R-CNN, Faster R-CNN, R-FCN, FPN ve Mask R-CNN'dir. İkinci yaklaşımda nesne algılama bir regresyon ya da sınıflandırma problemi olarak görülmektedir. The regression/classification temelli yöntemler başlıca MultiBox, AttentionNet , G-CNN, YOLO, SSD , YOLOv2 , DSSD ve DSOD yöntemleridir.

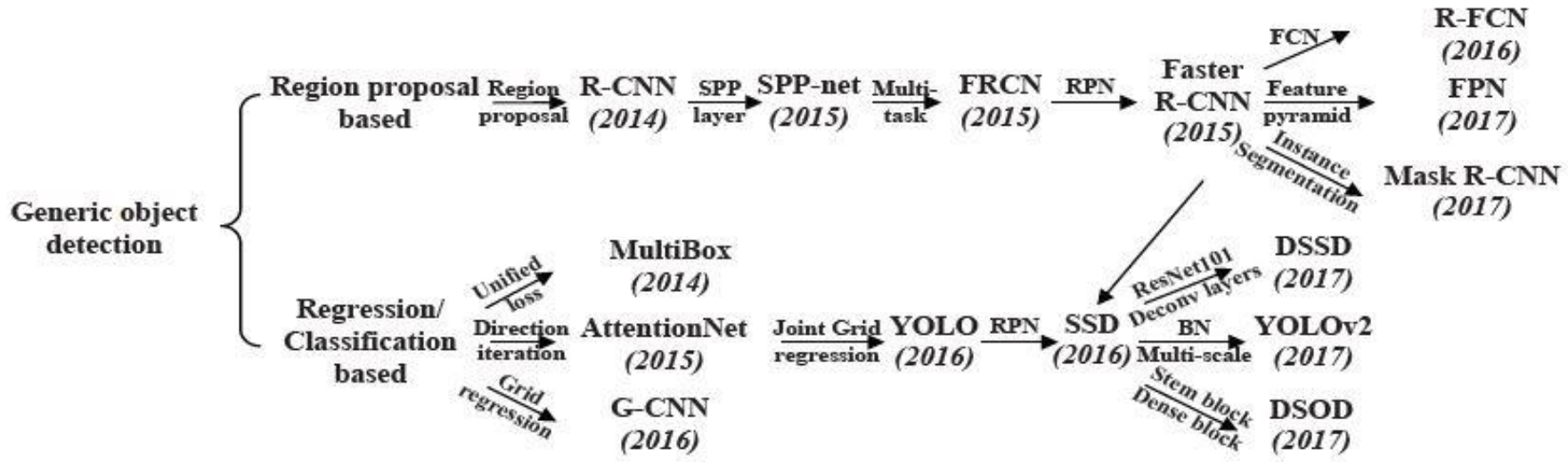


Figure 11. Generic object detection types[12]

[12] Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu, 'Object Detection with Deep Learning: A Review', IEEE Trans. Neural Networks Learn. Syst., vol. 30, no. 11, pp. 3212–3232, 2019, doi: 10.1109/TNNLS.2018.2876865.

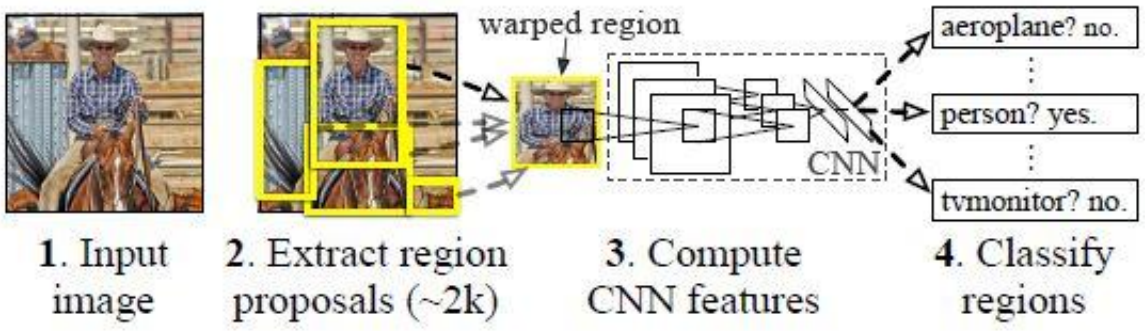


Figure 12. The flowchart of R-CNN[12]

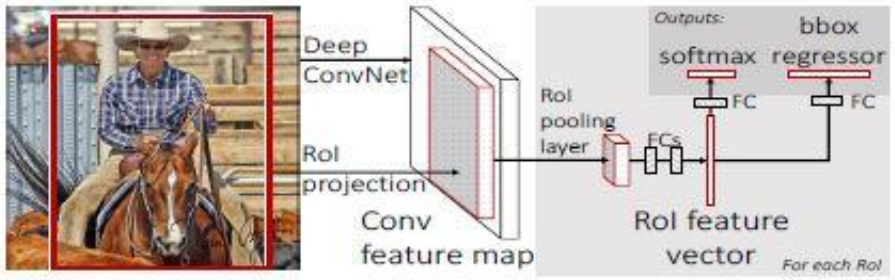


Figure 13. The architecture of Fast R-CNN[12]

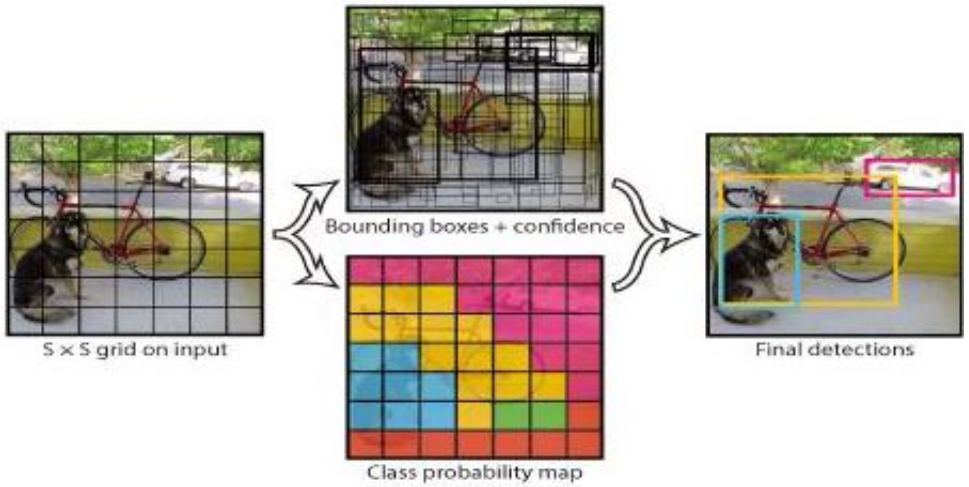


Figure 14. Main idea of YOLO[12]

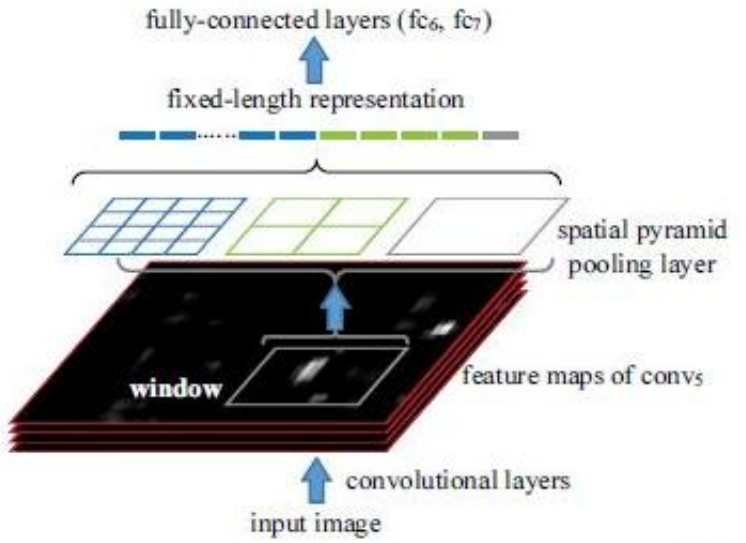


Figure 15. The architecture of SPP-net for object detection[12]

[12] Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu, 'Object Detection with Deep Learning: A Review', IEEE Trans. Neural Networks Learn. Syst., vol. 30, no. 11, pp. 3212–3232, 2019, doi: 10.1109/TNNLS.2018.2876865.

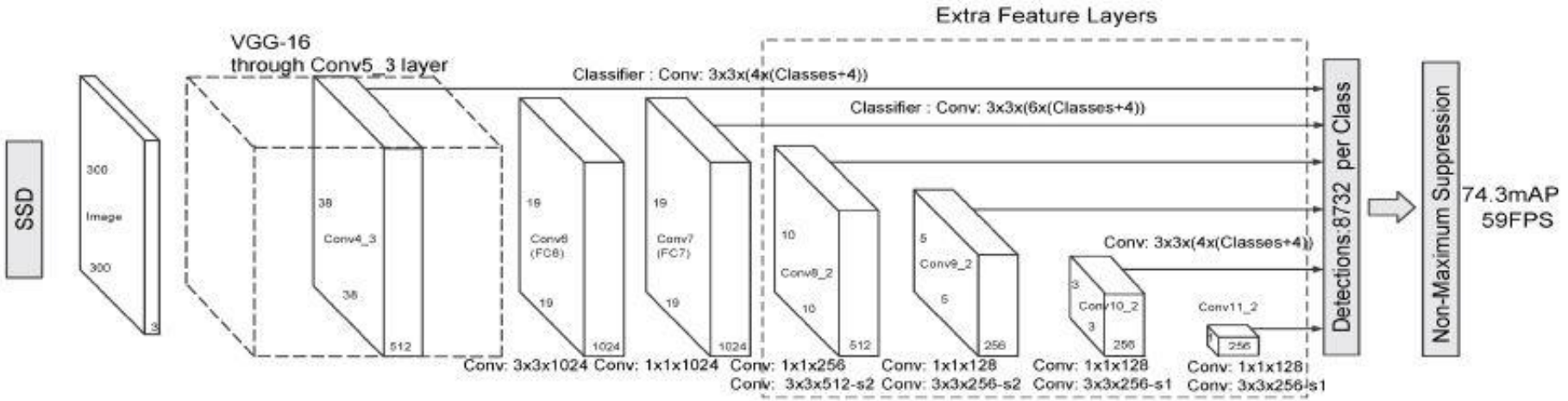


Figure 16. The architecture of SSD 300[12]

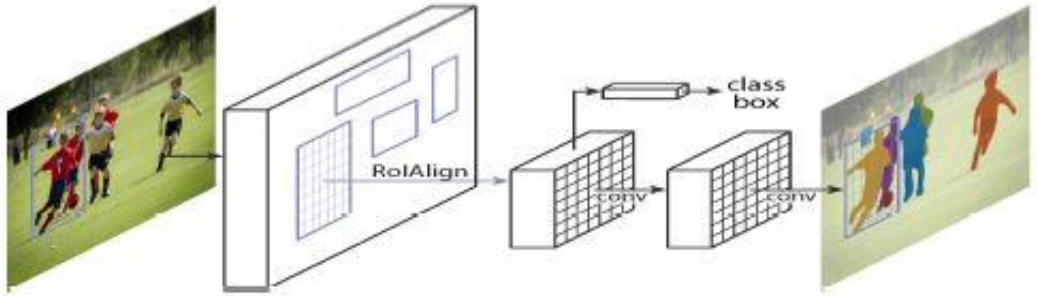


Figure 17. The Mask R-CNN framework for instance segmentation[12]

[12] Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu, 'Object Detection with Deep Learning: A Review', IEEE Trans. Neural Networks Learn. Syst., vol. 30, no. 11, pp. 3212–3232, 2019, doi: 10.1109/TNNLS.2018.2876865.

Bu ders kapsamında;
Nesne tanıma algoritmaları kapsamında yer alan **regresyon/sınıflandırma** temelli yaklaşım çerçevesinde **YOLO** yöntemi çeşitli versiyonları üzerinden anlatılacaktır.

TOPARLARSAK; Convnet'ler, bilgisayarla görme görevleri için en iyi makine öğrenimi modeli türüdür ve iyi sonuçlarla sıfırdan bir eğitim vermek mümkündür. Convnet'ler, görsel dünyayı temsil etmek için modüler kalıplar ve kavramlardan oluşan bir hiyerarşiyi öğrenerek çalışır. Küçük bir veri setinde aşırı uyum ana sorun olacaktır. Veri artırma, görüntü verileriyle çalışırken aşırı uyumla mücadele etmenin güçlü bir yoludur. Özellik çıkarma yoluyla mevcut bir convnet'i yeni bir veri kümesinde yeniden kullanmak kolaydır. Bu, küçük görüntü veri kümeleriyle çalışmak için değerli bir tekniktir. Özellik çıkarımının bir tamamlayıcısı olarak, mevcut bir problem tarafından önceden öğrenilen temsillerden bazılarını yeni bir probleme uyarlayan ince ayarı kullanabilirsiniz. Bu performansı biraz daha ileriye taşımaktadır.

DERİN ÖĞRENME MODELLERİ GELİŞTİRME REHBERİ

Makine öğrenmesinde ya da derin öğrenmede birtakım farklılıklar vardır. Makine öğrenmesinde öznitelik çıkartmak(problemi anlata özellikler) ve seçmek ciddi bir iştir. Ardından verilerden çıkarılan bilgilerin (feature extraction) anlamlı olup olmaması eğitim ve sınıflandırma sonucunda görülür. Önemsiz olanlar işlem yükünü azaltmak için çıkarılır ve önemli olanlar (feature selection) kullanılır. Bu kritik ve önemli bir iş olduğu için öznitelik mühendisliği adı altında bir kavram çıkmıştır.

Fakat derin öğrenmede katmanlar boyunca öznitelikler kendiliğince çıkarılır ve seçilir(bazı özniteliklerin ağılıkları daha fazla bazılarının daha az).Kendi kendine çıkarma ve güncelleme işlemi yapar. Burada bize düşen görev mimariyi düzgün tasarlamaktır. Böylece derin öğrenmede öznitelik mühendisliğini model mimarisi almış olur. Fakat burada da katman sayısı, nöron sayısı hangi optimizasyon algoritmalarının kullanıldığı, öğrenme oranının ne seçildiği gibi parametrelere başarılı bir şekilde karar verilmeli. Derin öğrenmede tecrübeniz çok fazla bile olsa ilk aşamada hiper parametreleri efektif seçmek zordur. Çünkü veriler ve kullanılan donanıma bağlı olarak düşünülmesi gereken bir şeydir. Her model her veri kümesine ya da donanıma uygun değildir ama her modele uygun parametreleri doğru seçebilmek için bir takım yollar vardır.

UYARI 1

Çıkış kanal sayımı, filtre sayım belirler. Bir matris birden çok kanaldan oluşuyorsa buna tensör diyoruz.

Veriden öğrenen makine öğrenmesi ya da derin öğrenme modelleri tasarlarken ne olması gerektiği çeşitli koşullar göz önüne alınarak modeli tasarlayan kişiye bırakılmıştır. Probleme, veri setine ve donanıma da bağlı olarak değişkenlik gösteren parametrelere hiper parametre denilmektedir. Yani hiçbir zaman net bir şey söyleyemeyeceğimiz konudur hiperparametre. Seçmek te önemlidir. Şimdi başarılı bir eğitim için hedef fikirlere değinelim.

Eğitim veri kümesinde elde edilen başarının test veri setinde elde edilen başarı ile birlikte sağlanması için fikirler şunlardır;

1-Daha çok veri toplamak

2-Veri kümesini çeşitlendirme

3-Gradyan iniş algoritması ile eğitimi uzun tutmak ya da gradyan iniş dışında farklı optimizasyon yöntemleri denemek

4-Daha büyük / daha küçük bir ağ denemek

5-Seyreltme (dropout) uygulamak

6-Düzenleme yöntemleri uygulamak

7-Ağ mimarisindeki hiperparametreleri(aktivasyon fonksiyonu, gizli katman sayısı vb.) fine tune etmek

8-Early stopping / erken durdurma sağlamak

9-Transfer Öğrenme yaklaşımı kullanmak

10-Ağa gürültü eklemek

YAPAY ÖĞRENME MODELİ GELİŞTİRİLİRKEN KARŞILAŞILAN PROBLEMLER

Dengesiz veri kümesi problemleri (kategoriler arası dengesiz sınıflandırma)

Bias vs. variance (yani eğitim kümesinin başarı oranı artarken test kümesinin azalmıs olması ya da loss için tam tersi)

Aşırı uydurma -az uydurma

Bazen karmaşık bir model yerine basit bir model seçmek daha mantıklı olabilir.

Basit bir model üretirsek

Hesaplama karmaşıklığı düşük olacaktır

Eğitim kolay olacaktır

Açıklanması kolay olacaktır

Genelleştirilebilir olacaktır (Yani iki sınıfı birbirinden mükemmel olmasa da genel çerçevede ayırabilmeli)

Özetle modeli karmaşık tuttuğumda çok iyi bir iş yapıyorum anlamına gelmemeli. İyi bir işte yapabilirim fakat önemli olan işlevselliktir. Basit ve küçük bir ağ ile daha başarılı ve efektif çıktılar üretebilirim.

Dengesiz Veri Kümesi Problemi

Modelimiz için çeşitlilik barındıran bir veri kümesi olmalıdır. Fakat veri setindeki örnekler arttıkça başarımda sonsuza değin artmayacaktır. Verinin kapladığı alan da önemlidir Fakat veri kümemiz küçükse illa derin öğrenme kullanmak zorunda değiliz. Geleneksel yöntemler kullanabiliriz.

İllaki az boyutlu veri sayısı problemimizi çözmek istersek; Ekstra veri toplayabiliriz. Fakat iş maliyeti ve zaman alır. Diğer yol ise (sentetik veri üretimi) görüntüyü kaydırma, rotasyon, çevirme, renk değişimi vb. yaklaşımlarile arttırmaktır.

Dünya çapında görsel veri işleme, ses ve konuşma işleme ya da sosyal ağ analizi gibi çeşitli alanlarda kullanılan farklı Deep Learning uygulamaları vardır. Bu uygulamaların her ne kadar kendi hedefleri var olsada temel olarak Şekil 18'de gösterilen pipeline'ı kullanırlar.

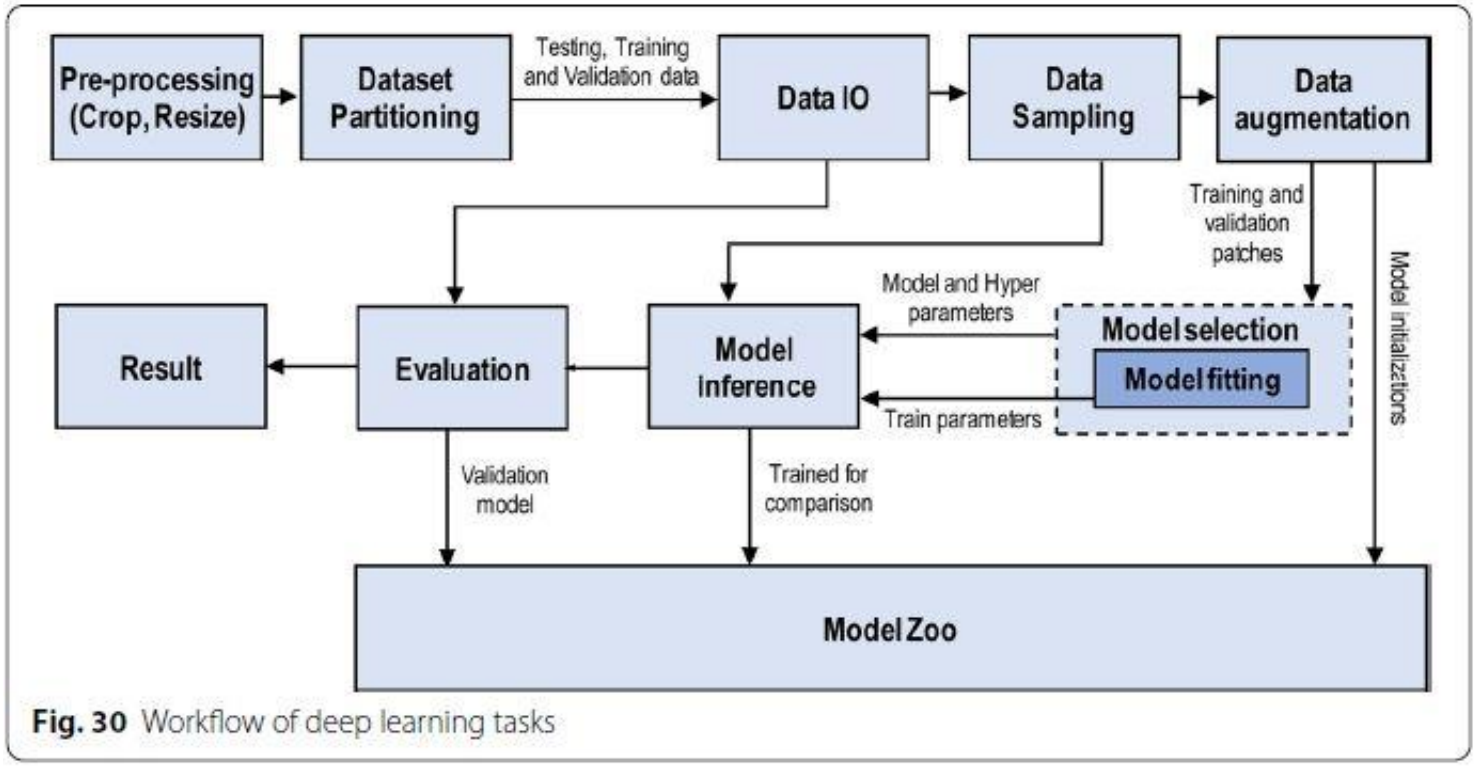


Figure 18. The Mask R-CNN framework for instance segmentation[6]

[6] L. Alzubaidi et al., Review of deep learning: concepts, CNN architectures, challenges, applications, future directions, vol. 8, no. 1. Springer International Publishing, 2021.

Kaynaklar

Prof. Dr. Ercan Öztemel, Yapay Sinir Ağları, Papatya Yayıncılık, 5. Basım Kasın 2020

Dr. Öğr. Üyesi Atınç Yılmaz, Öğr. Gör. Umut Kaya,Derin Öğrenme, Kodlab Yayınları, 3 Bası Ocak 2021

Deep Learning with Python second edition françois chollet

https://medium.com/@tuncerergin/convolutional-neural-network-convnet-yada-cnn-nedir-nasil-calisir-97a0f5d34cad (filtre mantığı)

Çeviri Editörü: Cemil Öz ‘ün kitab yazıalcak

Yapay zeka ve bilgisayarlı görü kitabu yazılacak

https://www.kaggle.com/kanncaa1/code