

COURSE4 – W1

Vid 1

Computer Vision


Computer Vision deep learning ile hızlıca gelişen alanlardan birisi. Autonomous driving için object detection problemlerinde veya telefonların face id özellikleri için ve daha bir çok yerde deep learning ile computer vision iş yapıyor. Hatta computer vision art generation için de kullanılıyor.

Doğrudan computer vision problemleri ile uğraşmayacak olsak bile bu alandaki gelişmeler diğer alanlara da aktarılabildiği için bunları bilmek değerli.

Aşağıda bazı computer vision problemleri yer alıyor.

Computer Vision Problems


Image Classification




64x64

→ Cat? (0/1)


Neural Style Transfer



↓



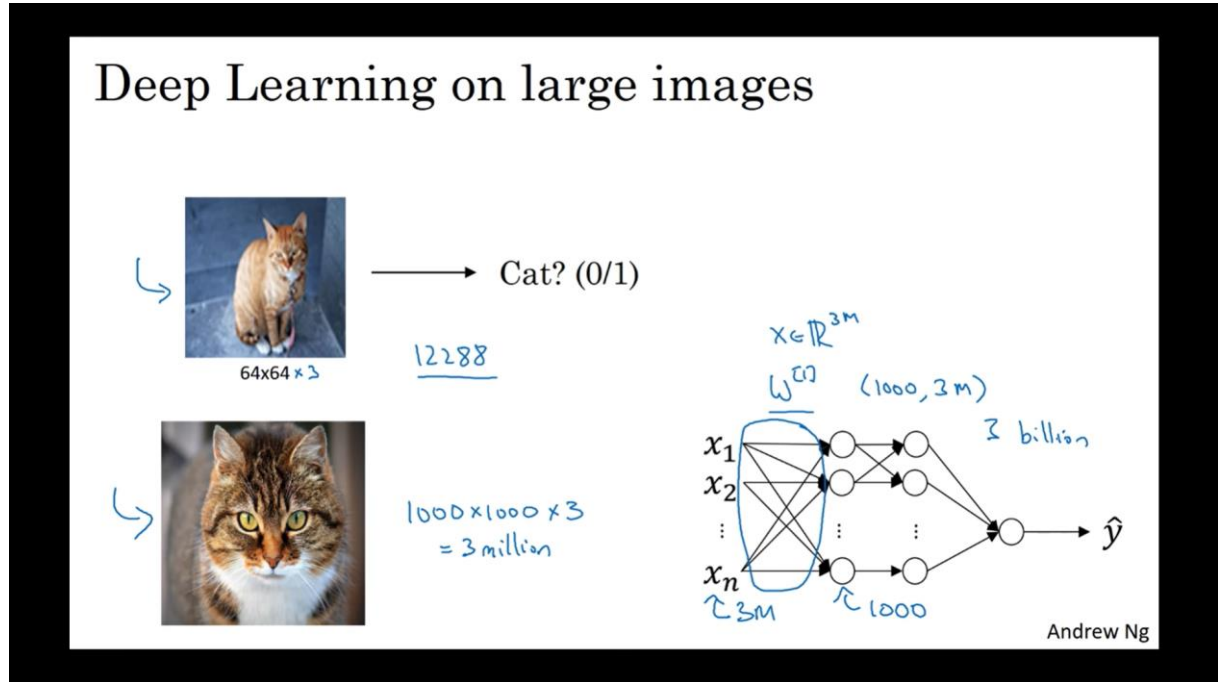
Object detection



Andrew Ng

Computer vision problemlerinin bir challenge'ı input sayısının çok büyük olması. Sonuçta bir image işlediği için, input sayısı çok fazladır.

Eğer 64x64 image'lar kullanılıyorsa ve RGB colour'u da katarsak 12288 input yapar. Bu rakam fena değil ama 64x64 image aslında çok küçük.



Eğer 1000x1000px images ile uğraşırsak input sayısı drastically artar. Colour'ı da kattığımız zaman 3 m feature oluşur. Bir NN kurduğumuzda 2. Layer da 1000 layer dahi olsa W1 matrisi 1000 x 3m'lik bir matris olur yani W1 için 3 billion parametre söz konusu. Bu çok fazla.

Bu denli fazla parametre söz konusuyken bu NN'i overfit etmekten kurtaracak kadar data toplamak çok zordur, ayrıca computational ve memory requirements is infeasible.

Ancak computer vision uygulamaları için sadece düşük çözünürlüklü image'lar kullanmak zorunda kalmak hoşumuza gitmez, işte bu sebeple yüksek çözünürlüklü image'ları kullanabileceğimiz bir yapıya ihtiyacımız var burada convolution yardımımıza koşar. Convolution CNN'in yapı taşlarından biridir.

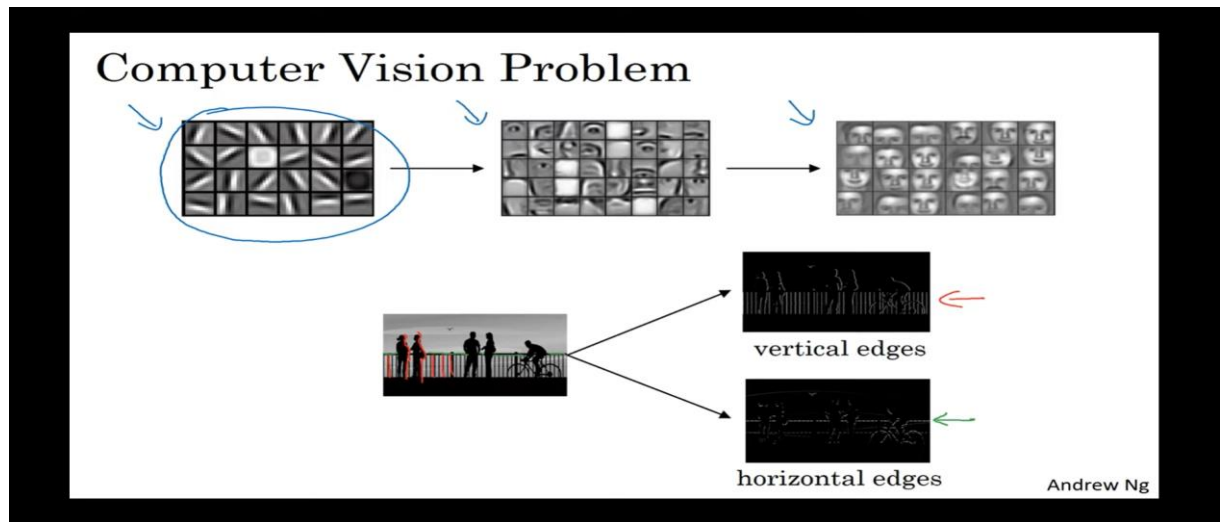
Vid 2

Edge Detection Examples

Convolution operasyonu CNN'in yapıtaşlarından biridir. Bu kısımda edge detection'ı örnek olarak kullanarak convolution'ın nasıl çalıştığı açıklanıyor. Zaten biliyorsunuz.

Daha önce bir NN'in early layerlarının nasıl edge detect edebileceğini daha sonraki layer'ların belki daha kompleks feature'lara dikkat ettiğini ve belki daha sonraki layerların ise doğrudan face detect edebildiğinden bahsetmiştik.

Bu kısımda bir imageda edge detection nasıl yapılabilir buna bakacağız. Aşağıdaki bisikletli resmi bilgisayarın anlamlandırabilmesi için horizontal ve vertical edge'leri ilk feature'lar olarak çıkarması mantıklı bir hamle olacaktır.



Peki bir image için edge detectionı nasıl yapacağız? Aşağıdaki örnekte en solda görülen matris 6x6 grayscale image. 3x3'lük ikinci matris ise vertical edge detection filter.

Vertical edge detection

$$3 \times 1 + 1 \times 1 + 2 \times 1 + 0 \times 0 + 5 \times 0 + 7 \times 0 + 1 \times -1 + 8 \times -1 + 2 \times -1 = -5$$

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6x6

convolution

1	0	-1
1	0	-1
1	0	-1

3x3 filter

*

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

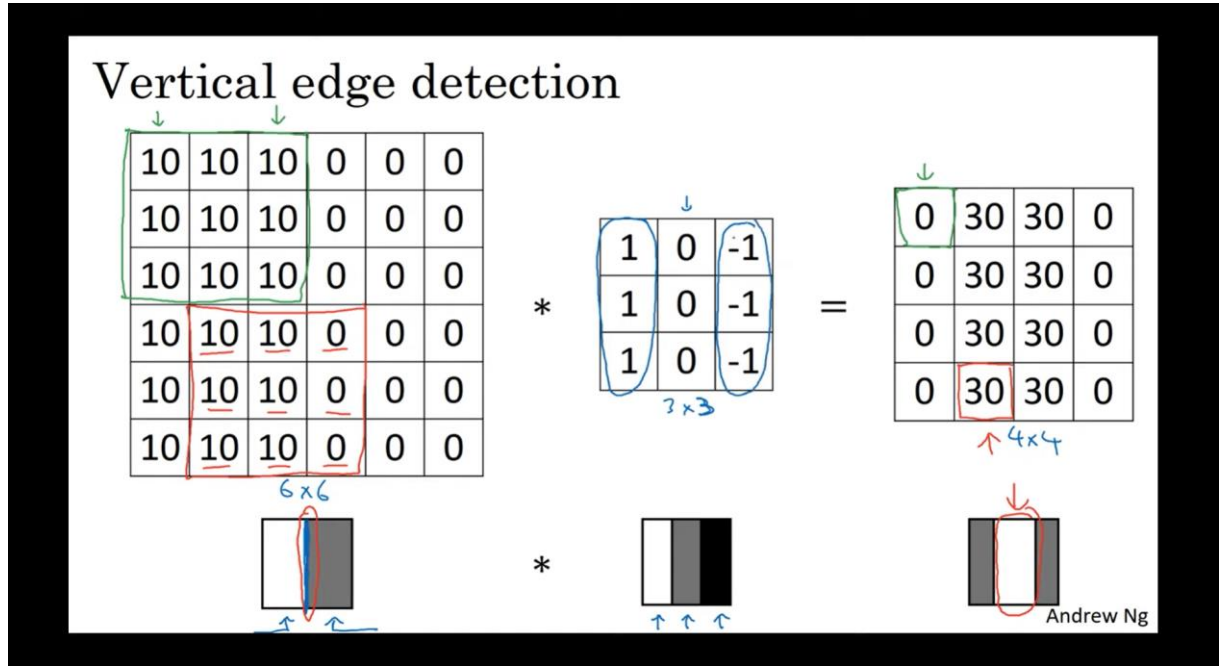
4x4

python: conv-forward
tensorflow: tf.nn.conv2d
keras: Conv2D

Andrew Ng

Yukarıdaki örnekte yapılan 6x6 image'ı 3x3'lük vertical edge detection filter ile convolve ederiz ve sonucunda elimizde 4x4'lük bir map kalır. Bu map temelde bize nerelerde vertical edge detect edildiğini gösterir. Nasıl convolve edildiğini biliyorsun. Filter'ı image üzerine yerleştirip elementwise multiplication ve sum yapıyoruz.

Bu convolution işleminin nasıl olup da vertical edge detection yapabildiğini daha iyi anlamaya çalışalım. Diyelim ki aşağıdaki gibi 6x6 bir image'ımız olsun görüldüğü gibi tam ortada bariz bir vertical edge var. Bizim vertical edge detector'ümüz bu edge'i nasıl detect ediyor bakalım.



Yukarıda gördüğün gibi en sol üste (yeşil kısma) filter uyguladığımız sonuç 0 çıkıyor, yani bir başka deyişle filterımızı burada bir edge detect etmedi. Buna karşın resimdeki kırmızı 3x3'lük alana filter uyguladığımızda sonuç 30 çıkıyor yani burada edge detect edildi.

Bu şekilde resmin her yerine filter uyguladığımızda, elde edilen map, resmin nerelerinde edge detect edildiğini gösteren bir matris özelliği taşımış oluyor. Bu örnek için resmin orta kısımlarında edge tespit edildiği için mapin orta kısımları 30 çıkıyor.

Bu örnekte resim çok küçük olduğu için sanki tespit edilen edge kalınmış gibi görünüyor ancak, gerçek boyutlu bir resimde map elbette bu kadar küçük olmayacaktır ve detect edilen edge de görece büyük değil küçük görünecektir.

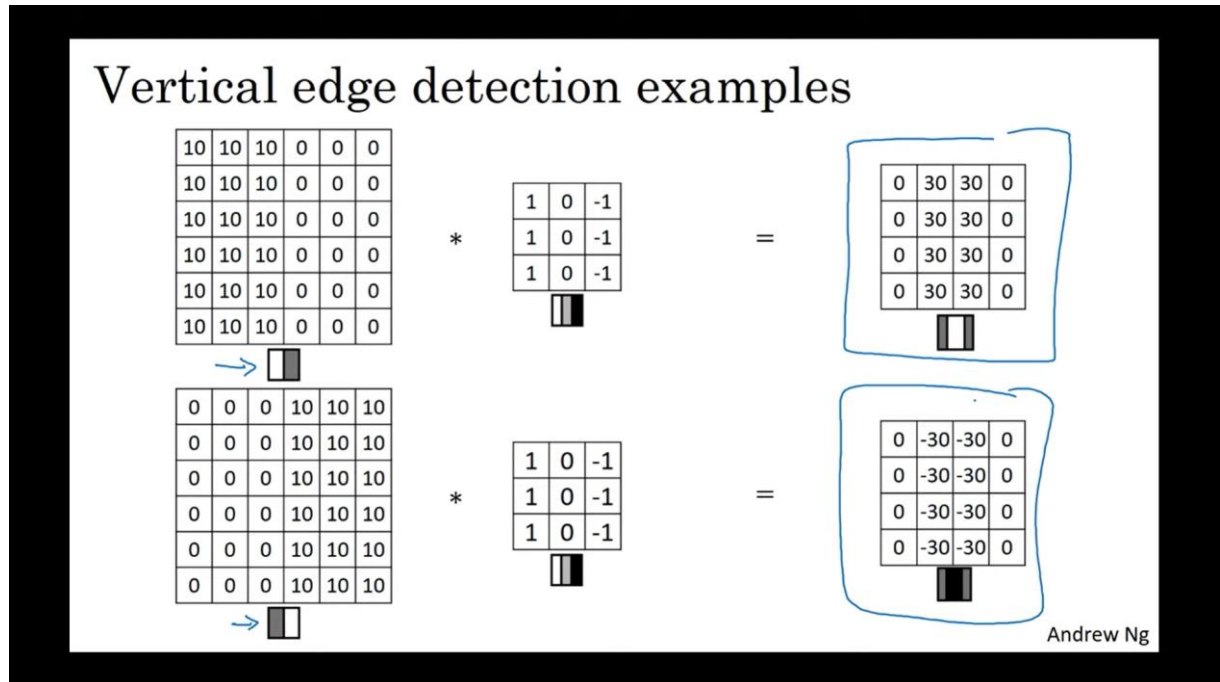
Bu filterin nasıl çalıştığına intuitive olarak bakalım, 3'e 3 lük alanlarda eğer sol taraf ile sağ taraf birbirinde farklı ise convolution sonucu da büyük veya küçük çıkacaktır, ancak sol ve sağ taraflar aynı ise convolution sonucunda bunlar birbirini sıfırlayacaktır.

Vid 3

More Edge Detection

Bu kısımda positive ve negative edges kavramlarını anlayacağız yani light to dark ile dark to light edge transitions'ın farkını anlayacağız. Ayrıca daha farklı edge detectorlara bakacağız. Son olarak da bir edge detector'ı elle kodlamak yerine bunun algoritma ile öğrenilebileceğinden bahsedeceğiz.

Aşağıda ilk görülen convolution bir önceki kısımdaki örneğin aynısı burada light to dark transition edge detect edilmişti, bu yüzden de mapte baktığımız zaman edge detect edilen konumlarda 30 sayısını görüyoruz.



Buna karşılık ikinci resme baktığımız zaman dark to light bir edge söz konusu bu yüzden de aynı filtre kullanıldığı takdirde detect edilen edge karşılığında mapte 30 yerine -30 görüyoruz.

Eğer bizim için edge'in transition'ı önemli değilse, simply we can take the absolute value of the map.

Şimdi edge detection ile ilgili daha fazla örnek inceleyelim. Vertical edge detector'ın nasıl çalıştığını anladığımıza göre, bir horizontal edge detector örneğine bakabiliriz. Aşağıdaki örnekte daha kompleks bir resim ve horizontal edge detector yer alıyor.

Vertical and Horizontal Edge Detection

1	0	-1
1	0	-1
1	0	-1

Vertical

1	1	1
0	0	0
-1	-1	-1

Horizontal

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

*

1	1	1
0	0	0
-1	-1	-1

=

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

6x6



Andrew Ng

Yukarıdaki örnek için yeşil ve mor işleme baktığımızda 30 ve -30'un nasıl çıktığını anlıyoruz, detector bu 3x3'lük image kısımlarında strong horizontal edges detect ediyor.

Buna karşın, sarı kısma baktığımızda sonuç 10 çıkmış, yani burada edge benzeri bir yapı var ama tam edge de değil, hiçbir transition yok da diyemiyoruz bu yüzden ne 30 ne 0 bunun yerine sonuç 10 yani almost a positive edge. Benzer şekilde -10 da almost a negative edge.

Özetle farklı filtreler ile vertical ve horizontal edge detect edebiliyoruz.

Ancak edge detect etmek için yukarıdaki örneklerde kullanılan filtre tek seçenek değil, aşağıdaki bazı diğer filtrelerle de edge detection yapılabilir.

Learning to detect edges

1	0	-1
1	0	-1
1	0	-1

1	0	-1
2	0	-2
1	0	-1

Sobel filter

3	0	-3
10	0	-10
3	0	-3

Scharr filter

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

3x3

45°
70°
73°

Andrew Ng

Deep learning ile gelişen bir düşünce ise şu, belki bu filtre'yi elimizle set etmek zorunda değildir, belki bu sayılar yerine parametrik değerler koyarak bu değerleri algoritmanın öğrenmesini bekleriz. Böylece algoritma bizim el ile set edeceğimizden daha robust low level featureları kendi kendine öğrenmiş olur, belki açılı edges detect eder belki başka bir şey.

Vid 4

Padding

Convolution'a yapılan bir modification padding'tir.

- Önceki örneklerde gördük ki 6x6 input ile 3x3'lik filtreyi convolve ettiğimizde output 4x4 oluyordu.
- Bunun sebebi filtrenin image üzerinde oturabileceği 4 kare olması (hem yatay hem dikeyde)
- Bu işlemi genelleştirsek, $[n \times n]$ bir input ile $[f \times f]$ bir filtreyi convolve ettiğimizde sonuç $[n-f+1 \times n-f+1]$ 'lik bir map olacaktır. Bu örnekte, $6-3+1=4$ olduğu için map 4x4 geliyor.

Padding

- shrinky output
- throw away info from edge

6×6
 $n \times n$

3×3
 $f \times f$

$n-f+1 \times n-f+1$
 $6-3+1=4$

4×4

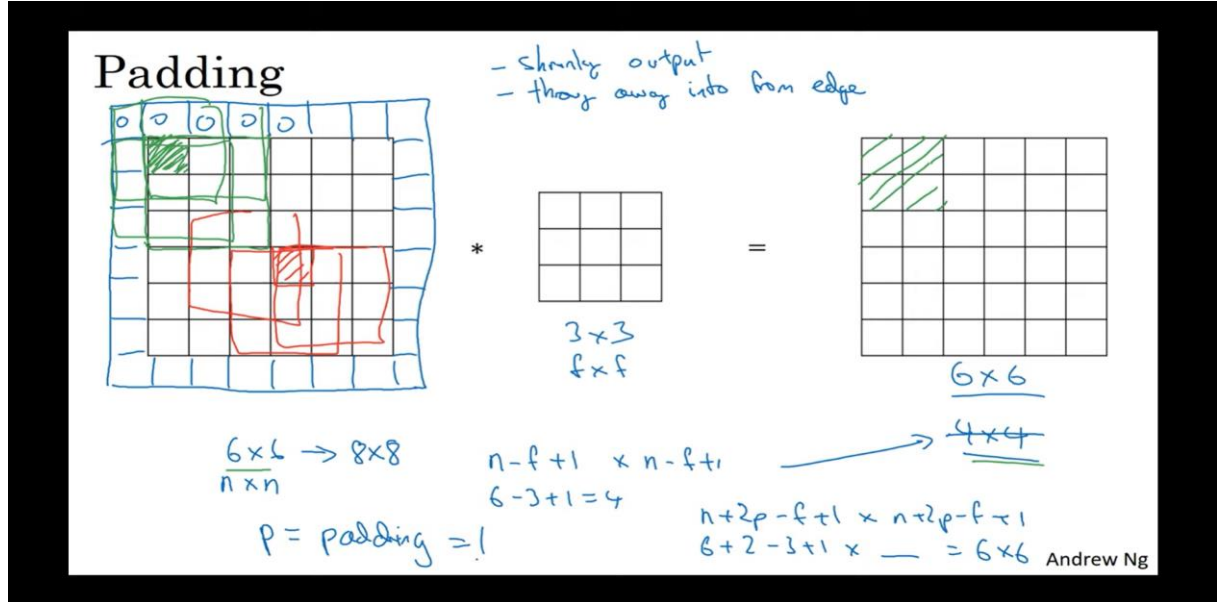
Andrew Ng

- Yani örneklerdeki convolution için 2 adet downside söz konusu:
 - Her convolution uyguladığımızda image'ımız shrinks. Yani ardarda convolution uyguladığımızda, sonunda elimizde çok küçük bir matris kalacaktır. Bu istenmeyen bir şey.
 - İkinci downside ise image'ın köşelerinde pixellerin convolution'a neredeyse hiç dahil olmaması, yukarıdaki örnekte, mavi padding olmasa köşede kalan yeşil pixel yalnızca 1 kez filtre ile convolve ediliyor, buna karşın köşede olmayan rastgele bir kırmızı pixel bir çok defa convolution'a katılıyor. Yani ikinci downside şu: throwing away a lot of information from the edges of the image.

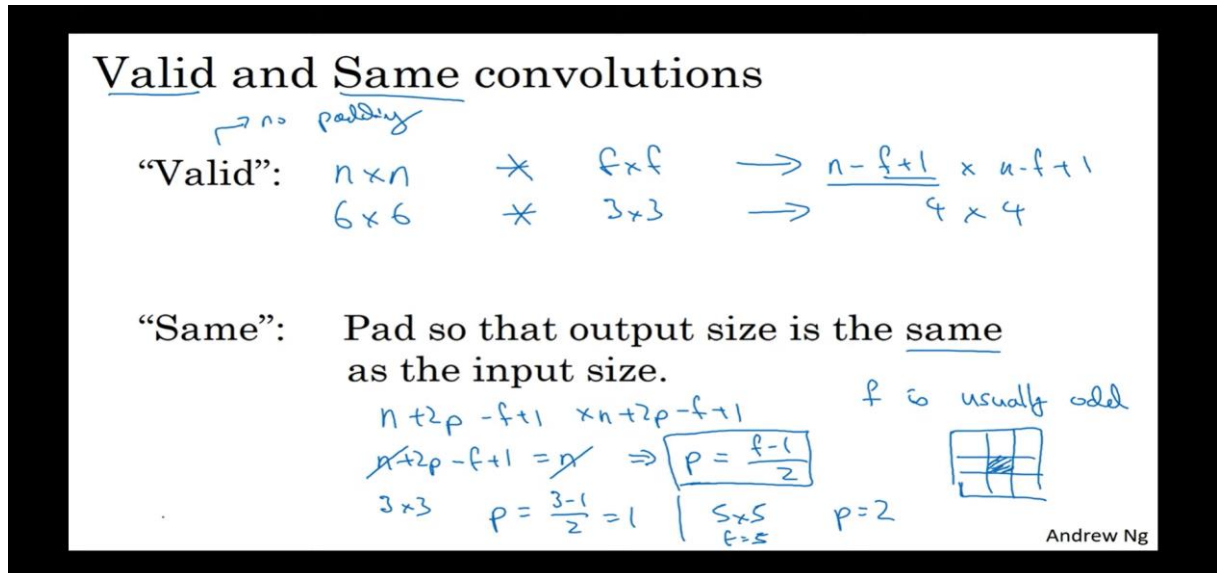
Sonuçta bu iki downside'dan kurtulmak için padding yardımımıza yetişiyor.

Padding=1 dersek, input image'ın dışına bir kat aşağıda görülen mavi katmandan çıkılıyor. Eğer 6x6 bir image'a bir kat padding çıkarsak yeni image 8x8 olur, yani $[n+2p \times n+2p]$.

Bu durumda convolution işlemi sonucunda output boyutları $[n+2p-f+1 \times n+2p-f+1]$ şeklinde bulunur, yani 6x6 olur ve orijinal image ile aynı boyutta olmuş olur.



Sonuçta padding'e göre Valid ve Same convolutions'dan bahsedilebilir.



Valid convolution demek padding yok demektir, bu durumda image – filter – output size formülü yukarıdaki gibi olur.

Same convolution ise padding var ve input size ile output size eşit demek. Bunu sağlamak için $p = (f-1)/2$ formülüne göre padding uygulanır.

Bu formül nereden çıkıyor? Bildiğimiz gibi padding varsa output size $[n+2p-f+1 \times n+2p-f+1]$ olarak bulunuyor, burada input ile output size'ın eşit olması demek $n+2p-f+1 = n$ demek buradan da $p = (f-1)/2$ gereksinimi bulunur.

Bu sebeple, f yani filterin boyutunu belirleyen parametre genelde tek seçilir ki same convolution uygulanabilsin. f 'in tek seçilmesinin bir diğer nedeni ise filtrenin bir merkezinin olmasının istenmesidir böylece filter position'dan bahsedilebilir.

Vid 5

Strided Convolutions

CNN için padding'e ek olarak kullanılan bir diğer building block da strided convolutions'dır.

Stride=2 dediğimizde filter'ı input üzerinde birer birer değil de ikişer ikişer kaydırıyoruz. En sol üstten başladığını varsayarsak, 2 sağa, 2 sağa daha kaydıktan sonra iki aşağı en sola dönüyor.

Strided convolution

$n \times n$ * $f \times f$
padding p stride s
 $s=2$

$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$
 $\left\lfloor \frac{7+0-3}{2} + 1 \right\rfloor = \frac{4}{2} + 1 = 3$

Andrew Ng

Bu çerçevede output size hesaplamak için yukarıda görülen formül kullanılıyor. Yani eski formülün $n+2p-f$ kısmı bu kez stride'a bölünüyor.

Burada dikkat edilmesi gereken bir nokta şu, yukarıdaki formülde köşeli paranteze benzer bir sembol görüyoruz bu **floor işlemi** yani içerideki sonuç tam çıkmazsa taban tam sayıya yuvarlayın diyor. Eğer input 8 x 8 ise padding yoksa filter 3x3 ve stride 2 ise sonuç 3.5'tan 3 olur.

Pratikte bu floor operasyonunu sağlayan şey, filter'ı image üzerinde kaydırırken image dışına çıkmamak. **Aşağıda formülün temize çekilmiş hali yer alıyor:**

Summary of convolutions

$n \times n$ image $f \times f$ filter
padding p stride s

Output size:

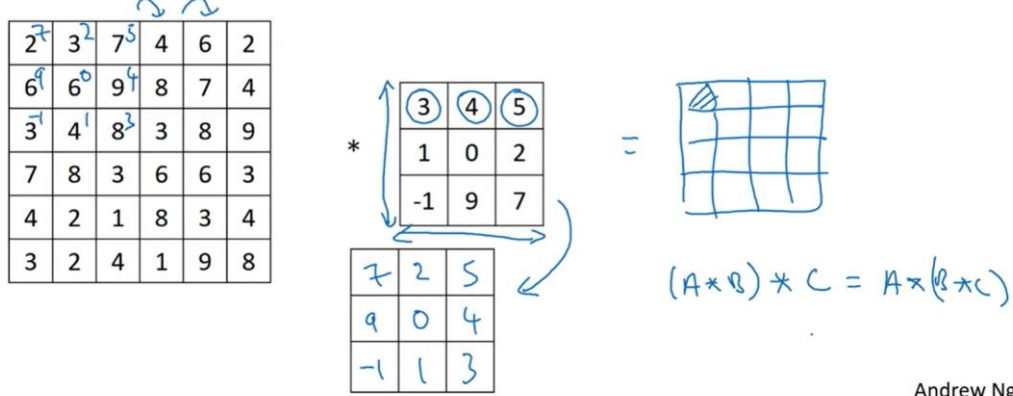
$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$

Andrew Ng

Son olarak teknik bir açıklama yapalım. CNN için yapılan convolution işlemi matematikte aslında cross-correlation olarak biliniyor, matematikçiler için convolution işlemi biraz daha farklı.

Technical note on cross-correlation vs. convolution

Convolution in math textbook:



Andrew Ng

Math problemi için yukarıda görülen input ile filtreyi convolve etmek bizim yaptığımızdan biraz farklı, önce filter hem horizontal hem de vertical eksenlerde flip edilir ve daha sonra bizim yaptığımız elementwise multiplication ile sum işlemi uygulanır.

Sonuçta deep learning uygulamalarında convention olarak flipping işlemini uygulamayız ve matematikteki cross-correlation işlemine convolution deriz. Çok önemli değil, sadece günün birinde görürsen kafan karışmasın.

Matematikte ve signal processingde convolution operasyonuna önce bu flipping'in uygulanmasının sebebi yukarıda görülen $(A*B)*C = A*(B*C)$ yani associativity özelliğini sağlamaktır, bu signal processing için useful olabilir ancak deep neural networks uygulamaları için pek önemli değil.

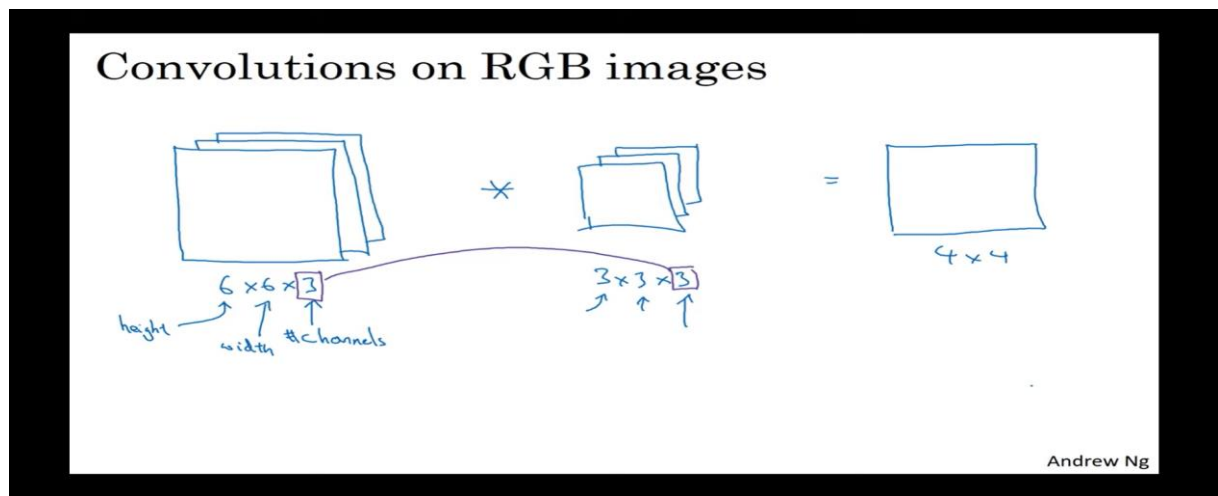
Vid 6

Convolutions over Volumes

2D images üzerinde convolution işleminin nasıl yapıldığını anladık, şimdi 3D volumes üzerinde convolution'ı nasıl yapabileceğimize bakalım.

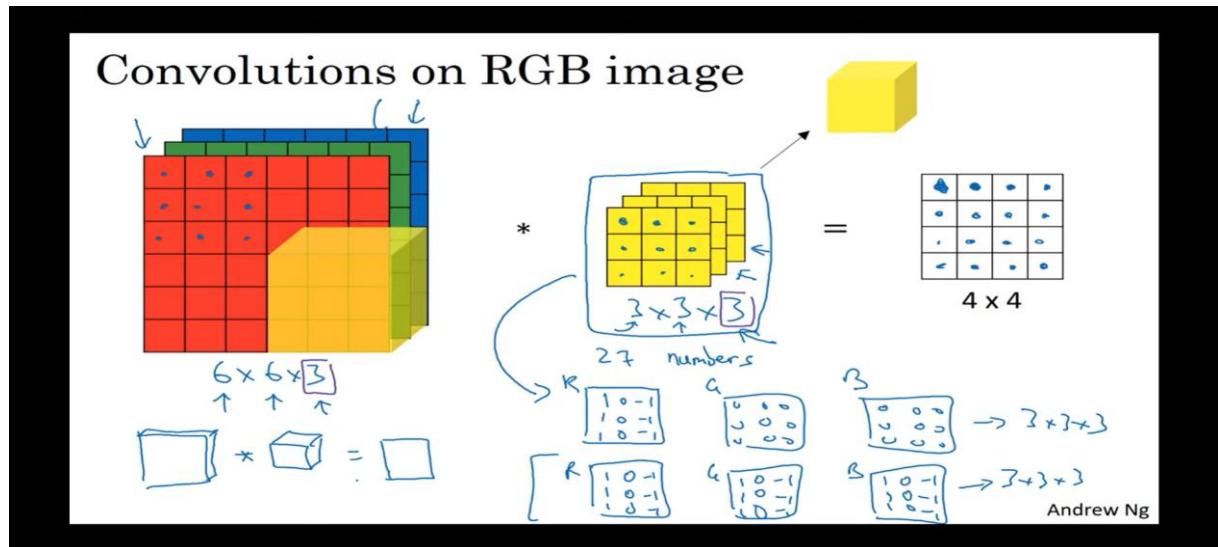
Diyelim ki bir RGB image üzerinde feature detect etmek istiyorum. 6x6 bir image için, RGB image'ın boyutu 6x6x3 olacaktır. Böyle bir input'a convolution uygulamak için artık 2D filtre yerine 3D filtre kullanıyoruz.

Inputun ve filtrenin z boyutu büyüklüklerinin eşit olması gerektiğine dikkat et. Bu z boyutu number of channels'ı temsil eder.



Böyle bir convolution'ın sonucu 4x4'lük 2D bir map olacaktır.

Bu işlemi daha iyi anlamak için, aşağıdaki daha temiz çizilmiş illustration'ı kullanalım.



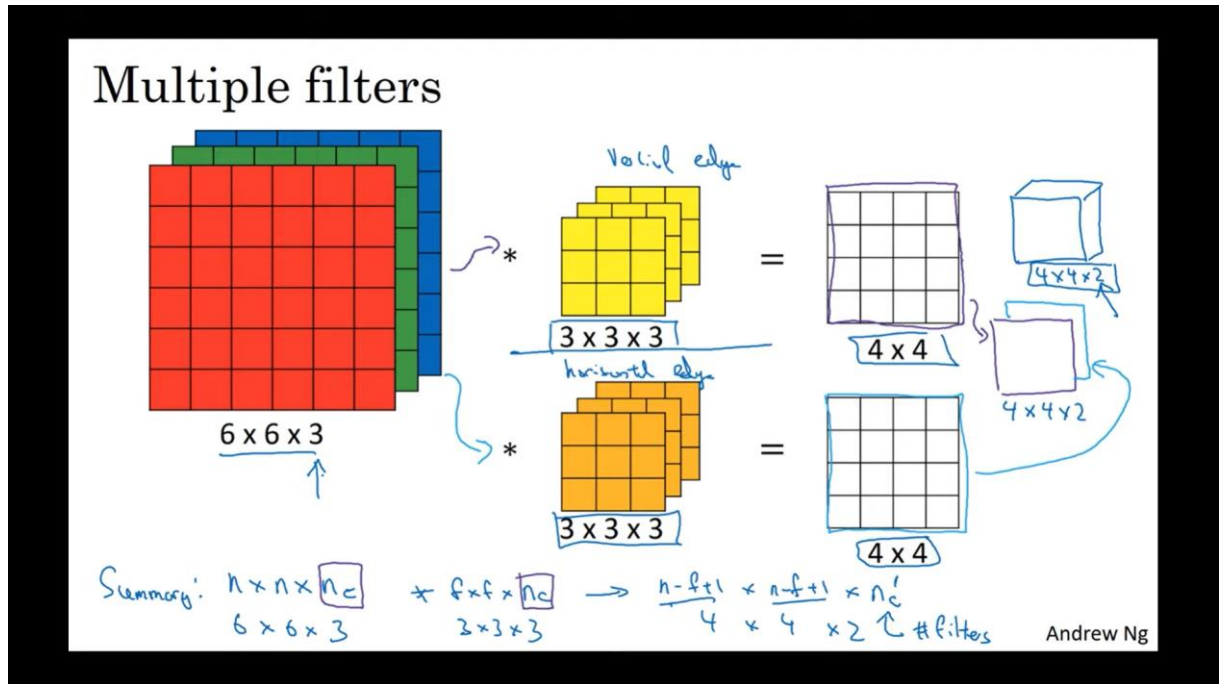
Sonuçta burada yapılan da 2D'de olana çok benzer, sadece 2D bir filtre yerine 3D bir filtre geziyor yine sol en üstten başlayıp sağa ve aşağıya doğru gittiğini düşün. Elementwise multiplication ve sum kullanılıyor.

Eğer sadece red channel'da vertical edge detection yapmak istersek yukarıdaki şeklin alt tarafında görülen ilk filtre yapısını kullanabiliriz, buna karşın onun hemen altındaki yapıda ise her channeldaki edge'ler detect edilmiş oluyor.

Convention olarak, input'un channel sayısı ile filter'ın channel sayısı aynı olur. Tabi teoride bunlar farklı olabilir yani filtre sadece red channel'a veya sadece green ve blue'ya bakabilir.

Artık volume'lar ile nasıl convolution yapılacağını anladık, bu kısımda son bir önemli noktaya değineceğiz, ya ben tek bir feature'u değil de birden fazla features detect etmek istiyorsam, yani mesela hem vertical edges hem horizontal edges hem 45 degree edges ve daha bir sürü başka low level features.

Bunu yapmak için doğal olarak birden fazla feature detector yani filter kullanmalıyız. Her bir filter input üzerinde farklı feature'ları arayacak ve sonucunda birbirinden bağımsız feature map'ler çıktı olarak elde edilecek.



Sonuçta iki feature detector için elde edilen 2 map'i birleştirerek $4 \times 4 \times 2$ boyutlarında tek bir map volume elde edebiliriz.

Output formülünü özetlersek, aslında eski formülle aynı. Sadece eğer input n_c tane channel'a sahipse filter da n_c tane channel'a sahip olacak fakat çıkan map 2D olacaktır, boyutunu $n-f+1$ ile bulabiliriz, elbette padding ve stride için içine girerse yine daha önceden belirttiğimiz formüller devreye girer.

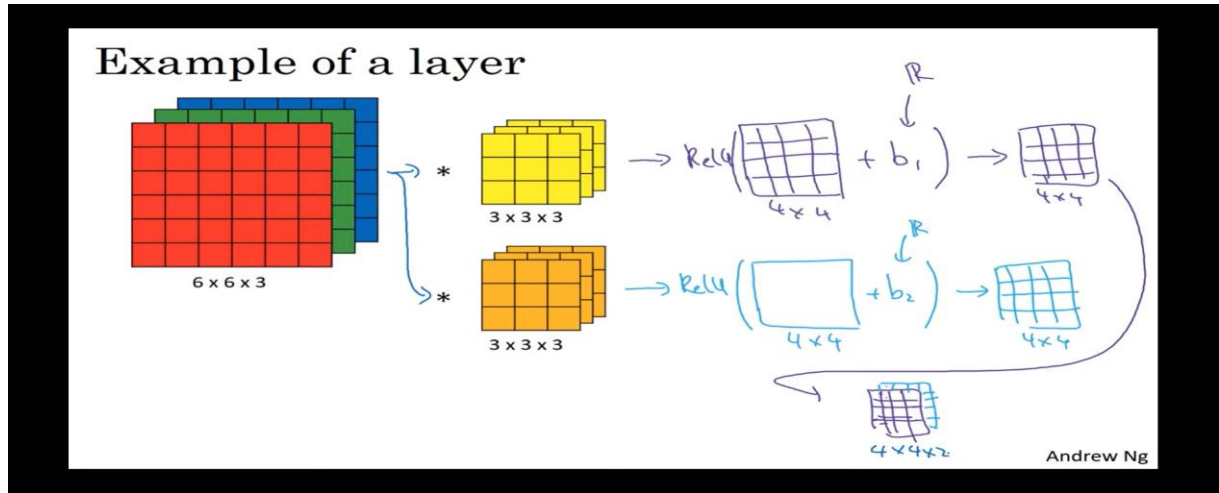
Eğer birden fazla feature detect ediyorsak, bu durumda 2D mappingleri birleştirebiliriz ve feature sayısı kadar bir 3. boyut eklenmiş olur.

Vid 7

One Layer of a Convolutional Net

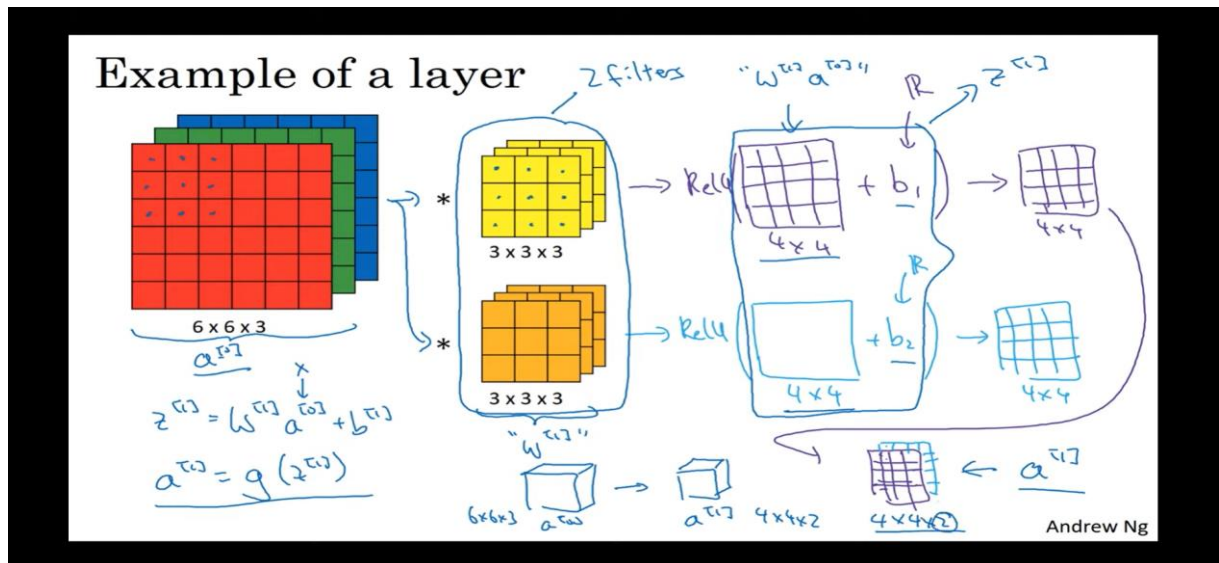
Bu kısma kadar convolution'ın nasıl yapıldığını anladık, artık bir convolution layer'ın nasıl implement edildiğine bakabiliriz.

Diyelim ki elimize 6x6x3 bir RGB resim var, bu resimde iki ayrı feature detect etmek istiyoruz. Bunun için her filtrenin çıkışı 4x4 bir map verecektir. Daha sonra bu maplere bir constant ekliyoruz (broadcasting ile her elamana ekleniyor) ve relu ile non-linearity ekliyoruz (başka bir non-linear function da olabilirdi.) Eklenen non-linearity'nin sebebini ben şöyle düşündüm: daha önce demiştik ki NN için activation non-linear olmazsa, ilerleyen layerlarda complex features öğrenilemiyor, sonuçta hep linear bir feature öğrenmiş oluyoruz burada da sebep bu olsa gerek.



Sonucunda yine 4x4'lük map'ler elde ettik, bu mapleri birleştirecek elimizde 4x4x2'lik bir resultant map olur. İşte tüm bu yukarıdaki adımlar ile bir convolution layer yaratmış olduk.

Yukarıdaki bu convolution layer şuanda standard bir NN layer'dan çok daha farklı görünüyor, fakat bu aslında benzer. Bu benzerliği anlamaya çalışalım.



Normal bir NN için $z1 = w1*a0 + b1$ ve $a1=g(z1)$ olarak tanımlandığını biliyoruz. Burada da input'u $a0$ olarak düşünebiliriz, feature detector'u ise $w1$ olarak düşünebiliriz. Sonuçta $a0*w1$ buluyoruz buna $b1$ ekliyoruz ve $z1$ 'i elde ettikten sonra $g(z1)$ ile maplerin son halini elde ediyoruz, bu mapleri birleştirince $a1$ 'e ulaşmış oluyoruz.

Sonuçta $a0$ 'dan $a1$ 'a bir convolutional layer ile böyle geçiyoruz, önce $a0$ 'a standard bir networkde olduğu gibi bir linear operation uygulanıyor (convolution) daha sonra da bias eklenip bir non-linear operation uygulanıyor ve resultant map $a1$ olarak elde ediliyor.

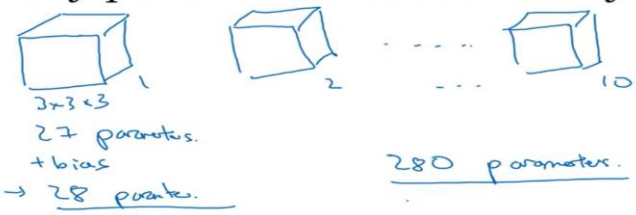
Boyutlara gelirse, $6x6x3$ 'ten $4x4x2$ elde edildi.

- Eğer input $6x6x3$ ise 3 channel var demektir, filterların da son dimension 3 olmalı.
- Daha sonra her bir filterin çıkışı 2D olur ve dimensions $n-f+1$ yani $6-3+1 = 4$ olarak hesaplanabilir.
- Son olarak kaç filtre varsa o layer çıkışının son dimension'ı ona göre belli olur, yukarıdaki durumda 2 filtre vardı, her filtrenin çıkışı $4x4$ olduğuna göre output $4x4x2$ oldu. Yani eğer 10 filtre kullanılsaydı, çıkış $4x4x10$ olacaktı.

Şimdi bir Convolutional Layer için parametre sayısını nasıl belirleriz ona bakalım, eğer yukarıdaki gibi $6x6x3$ bir RGB input için 10 feature detect etmek istiyorsak ve her filter $3x3$ boyutlarında olacaksa, ki input 3 channel olduğu için filter da $3x3x3$ olmak durumunda. Totalde bu layer da kaç parametre vardır?

Number of parameters in one layer

If you have 10 filters that are $3 \times 3 \times 3$ in one layer of a neural network, how many parameters does that layer have?



Andrew Ng

Bizim parametre dediğimiz şeyin w ve b olduğunu biliyoruz, bu durumda yukarıda yapılan anolojiden her bir filtre elemanlarının w olarak kabul edildiğini anlıyoruz bir de bunların yanında her biri için b var.

$3x3x3 + 1 = 28$ parameters for each filter. 10 tane filtre olduğuna göre bu layerda 280 parametre var.

Burada güzel bir nokta var: Input size ne olursa olsun convolution layer'ın parametre sayısı remains fixed. Böylece input çok büyük de olsa 280 tane parametre ile yani 10 filtre ile 10 farklı feature için detection yapabilirim. This is an important property of CNNs that makes them less prone to overfitting. Once we learn 10 feature detectors to work, we can apply this even very large images.

Şimdi videoyu sonlandırmak için bir CNN içindeki tek bir convolution layer'ı describe etmek için kullanılan notation'ı anlayalım.

Diyelim ki söz konusu layer l bir convolution layer.

- Önceden f ile tanımlanan filter size'ı artık $f[l]$ olarak tanımlıyoruz.
- Benzer şekilde $p[l]$, $s[l]$ ilgili convolution layer için padding ve stride bilgilerini verir.
- $nc[l]$ ise bu layer da kaç farklı feature'un detect edildiğini yani filter sayısını verir.
- Input'un height width ve channel bilgileri olacak bu bilgileri aşağıdaki gibi denote edebiliriz, daha önce olduğu gibi eğer l. Layerdan bahsediyorsak bu layer'ın inputunu denote ederken $[l-1]$ 'i kullanmak mantıklı.
- Aynen, 1. Layer'a a0'ın girip a1'in çıkması gibi output da aşağıdaki gibi $[l]$ ile denote edilecek, output'un da bir height ve width'i olacak ancak outputun son dimension'ı channel sayısı değil, filter sayısını onu unutma. Yani burada input ve output'un sonunda kullanılan nc'lerin ilki input'un channel sayısı iken 2.si layer da kaç tane filter olduğu.

Summary of notation

If layer l is a convolution layer:

$f[l]$ = filter size

$p[l]$ = padding

$s[l]$ = stride

$nc[l]$ = number of filters

→ Each filter is: $f^{[l-1]} \times f^{[l-1]} \times nc^{[l-1]}$

Activations: $a^{[l-1]} \rightarrow n_H^{[l-1]} \times n_W^{[l-1]} \times nc^{[l-1]}$

Weights: $f^{[l-1]} \times f^{[l-1]} \times nc^{[l-1]} \times nc^{[l]}$

bias: $nc^{[l]} = (1, 1, 1, nc^{[l]})$ ← #f: (fws in layer l.

Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times nc^{[l-1]}$

Output: $n_H^{[l]} \times n_W^{[l]} \times nc^{[l]}$

$$n_{HW}^{[l]} = \left\lfloor \frac{n_{HW}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

$$A^{[l]} \rightarrow m \times \underbrace{n_H^{[l-1]} \times n_W^{[l-1]} \times nc^{[l-1]} \times nc^{[l]}}_{nc^{[l]} \times n_H^{[l-1]} \times n_W^{[l-1]}}$$

Andrew Ng

- Output'un height ve width'ini aynı formülle hesaplayabiliyoruz, daha önce gördüğümüz gibi floor $((n+2p-f)/s + 1)$ yapısını kullanıyoruz hem H hem W için aynı formül geçerli.
- Filtre boyutu bahsettiğimiz gibi $f[l] \times f[l]$ olacak buna ek olarak input'un channel sayısı kadar bir depth olacak.
- Bu convolution layer'ın output'u ise $a[l]$ olarak gösterilecek ve boyutunu zaten belirttik, yukarıda activation olarak tekrar yazmış, eğer vectorized implementation istersek bunu $A[l]$ 'e çevireceğiz ve number of examples yani m için içine girecek.
- Weights'in sayısı da bildiğimiz gibi filtre boyutuyla, number of filters'ın çarpılmasıyla bulunabilir.
- Son olarak bias'a gelirse, her filter için bir bias olacak. Daha sonra göreceğiz ki kodda daha convenient olmas için $1, 1, 1, nc[l]$ 'lik bir matrix veya tensor olarak tanımlayacağız

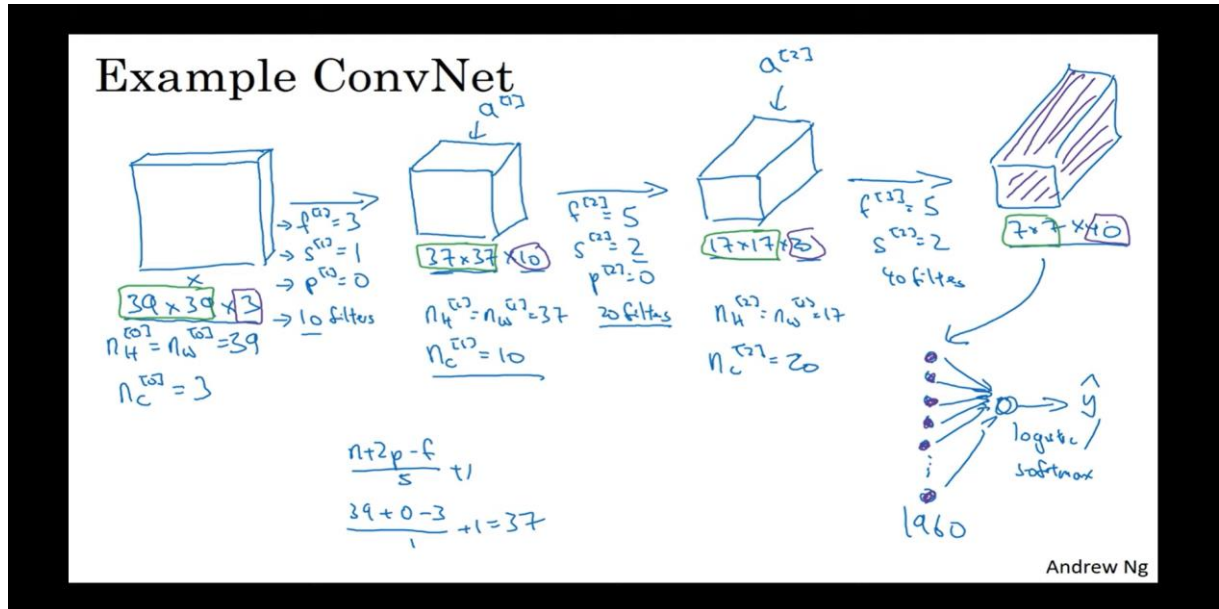
Vid 8

Simple Convolutional Network Example

Bir önceki örnekte single convolution layer'ın detaylarından bahsettik, şimdi bir örnek ile deep convolutional neural network'ü anlamaya çalışalım. Böylece geçen bölümde introduce edilen notasyonları da pratik yapmış oluruz.

Diyelim ki elimizde bir image var ve amacımız image classification yapmak (cat / not a cat). Bu task için kullanabileceğimiz bir conv. Network oluşturalım.

- ➔ Diyelim ki input image boyutu 39x39x3.
- ➔ İlk katmanda feature detection için 3x3 filters kullanılıyor, yani $f[1] = 3$.
- ➔ Stride =1 ve padding yok yani valid convolution layer kullanıyoruz.
- ➔ Son olarak bu layerda 10 filtre kullanılıyor.



- ➔ Bu durumda, ilk layer'ın çıkışı yani a^1 'in boyutları 37x37x10 olacaktır.
- ➔ 10 olmasının sebebi 10 tane filtre kullanmamız.
- ➔ 37'de $n+2p-f/s + 1$ formülünden geliyor
- ➔ Burada $nc[0]=3$, $nc[1]=10$ olduğuna dikkat et, ilk layer için $nc[0]$ channel sayısıydı, $nc[1]$ ise filter sayısıydı, aynı notasyonla gösterilmesi bana anlamsız gelmişti ama şimdi olay anlaşıldı $nc[1]$ diğer convolutional layer'a channel gibi davranacak, bu yüzden notasyon aynı.
- ➔ Diyelim ki ikinci layer için 5x5 filters kullanıyoruz (filtrelerin depth'i 10 olmalı ki input ile uyum sağlasın), bu kez 20 tane filtre kullanacağız, stride 2 ve padding yine yok.
- ➔ Output dimensions yine aynı formülle hesaplanır. Depth zaten 20 olacak çünkü önceki layer da 20 tane filtre kullanıldı. 17 de $(37-5)/2+1$ formülüyle bulunur.
- ➔ Bu şekilde yeni conv. Layers eklenebilir, mesela diğer layer için 40 tane 5x5 filtre kullanılsın stride=2 padding yok. Bu kez output 7x7x40 boyutlarında gelecektir.

→ Bu noktadan sonra yapılabilecek bir şey şu: elde edilen 7x7x40'lık final output'u flatten ederiz, sonuçta 1960 tane inputumuz oluşmuş olur, ve bunu bir logistic regression unit'e veya softmax unit'e besleriz. Böylece, classification sonucu elde edilir.

→ Burda yapılan'ı intuitive anlamaya çalışırsak bence olan şu, 39x39 image üzerinde önce 10 adet low level feature'ın nerelerde görüldüğü detect ediliyor yani mesela edges detect ediliyor, daha sonra oluşan map üzerinde tekrar bazı low level features detect ediliyor, fakat detect edilen features her katmanda daha da complexleşiyor. En sonunda resim üzerinde direk kedi burnu kedi kulağı gibi features'a bakılmış oluyor ve bu özelliklere göre logistic regression kararını veriyor.

→ Convolutional layers dizaynı için yapılan işin çoğu filter size, padding, stride veya kaç filtre kullanılacağı gibi hyperparametrelerin seçilmesidir. Bu hafta ve önümüzdeki hafta bu hyperparametrelerin seçimleri ile ilgili guidelines'dan bahsedilecek.

→ Şimdilik, bu örnekten şunu çıkarabiliriz, tipik olarak input size her layer da giderek azalır burada 39x39 başladı ve en son 7x7'e kadar düştü. Buna karşın, number of channels ise giderek artar, burda 3 ile başladı ve 40'a kadar çıktı.

Tipik bir convolutional network'de 3 farklı tipte layers bulunur. İlki bizim gördüğümüz convolution layer, ikincisi pooling layer ve sonuncusu da fully connected layer.

Sadece convolution layers kullanarak iyi networkler kurmak mümkün olsa da bir çok CNN bu üç layerdan da faydalanır.

Types of layer in a convolutional network:

- Convolution (conv) ←
- Pooling (pool) ←
- Fully connected (FC) ←

Andrew Ng

Pooling layers ve fully connected layers are easier to define compared to convolution layers. Yani zor kısmı hallettik, önümüzdeki kısımlarda bu layerları da anlayalım.

Vid 9

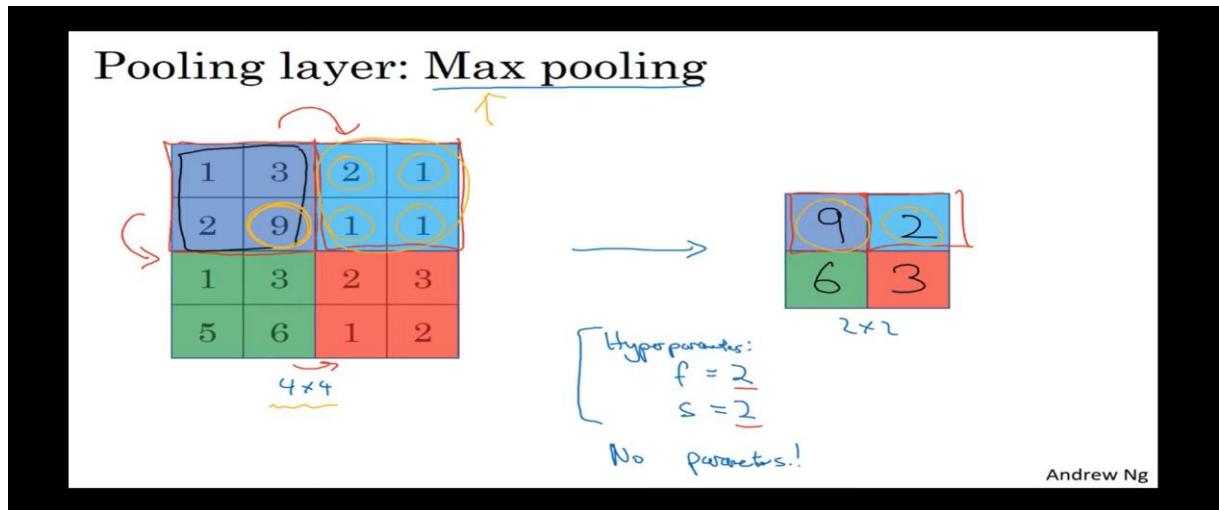
Pooling Layers

CNN için convolution layers'ın yanında genelde pooling layers da kullanılır, amaç:

- Reduce the size of the representation, increase the computation speed.
- Make feature detection more robust.

Önce pooling'i aşağıdaki örnekle anlayalım ve sonra bunu neden yapmak isteyebileceğimizden bahsedelim.

Diyelim ki aşağıdaki gibi 4x4 inputumuz var ve buna max pooling uygulamamız gerekiyor, bu durumda eğer max pooling hyperparameters $f=2$ ve $s=2$ kabul edilirse, aynı filter gibi 2x2 lik window input içinde stride=2 ile döner ve her window için max eleman alınır.



Dolayısıyla output size aynı filter uyguluyormuşuz gibi $(n+2p-f)/s + 1$ formülüyle bulunabilir, burada $(4-2)/2 + 1 = 2$ çıkar yani output 2x2.

Max pooling'in ne yaptığını dair intuition şu: Yukarıdaki 4x4'lük input'u bir feature map olarak düşün, yani her element aslında bir feature'un detect edilip edilmediği bilgisini taşıyor, element'in değeri büyükse aranan feature detect edilmiş demek.

Bu bağlamda mesela sol üstteki kısımda 9 var, demekki burada bir feature detect edilmiş bu belki low level bir feature (edge gibi) veya daha complex bir feature (an eye). Pooling sonucunda o 4 elemanlık kısımda eğer bu feature detect edilmişse bu bilgi tutuluyor, eğer feature detect edilmemişse bu bilgi de tutuluyor (mesela sağ üst 4 karede feature detect edilmemiş).

Böylece hem input size'ı küçültürken bizim için asıl önemli olan bilgiyi yani feature'ların detect edilip edilmediğini kaybetmiyoruz. Gereksiz information'ı atıyoruz böylece modelimizin bu information'a overfit etme ihtimalinden kurtuluyoruz, veya modelin bu information'a overfit etmemesi için tonlarca data toplama gereksinimimizden kurtuluyoruz.

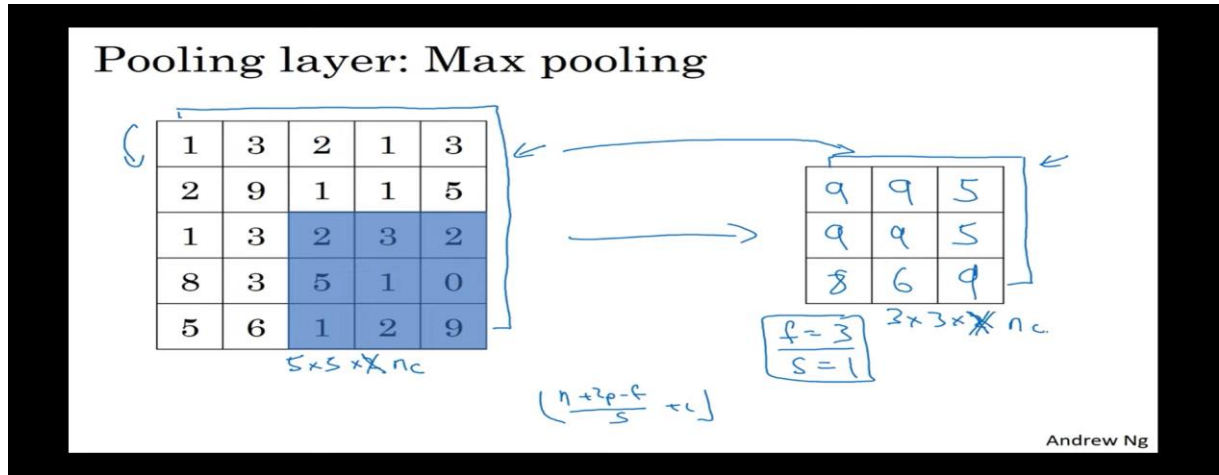
Ayrıca bu işlem bir robustluk sağlıyor, çünkü böylece feature'ın tam olarak spesifik bir konumda ortaya çıkmasını önemsememiş oluyoruz, eğer o bölgelerde bir feature varsa, bizim için

tamam. Bu nasıl robustluk sağlar? Böyle olmasa, model eğitilirken feature'ları spesifik bir noktada arayabilir, böyle olunca cat detect eden bir sistem kedi fotoğrafı 30 derece açılı olarak verilince afallayabilir.

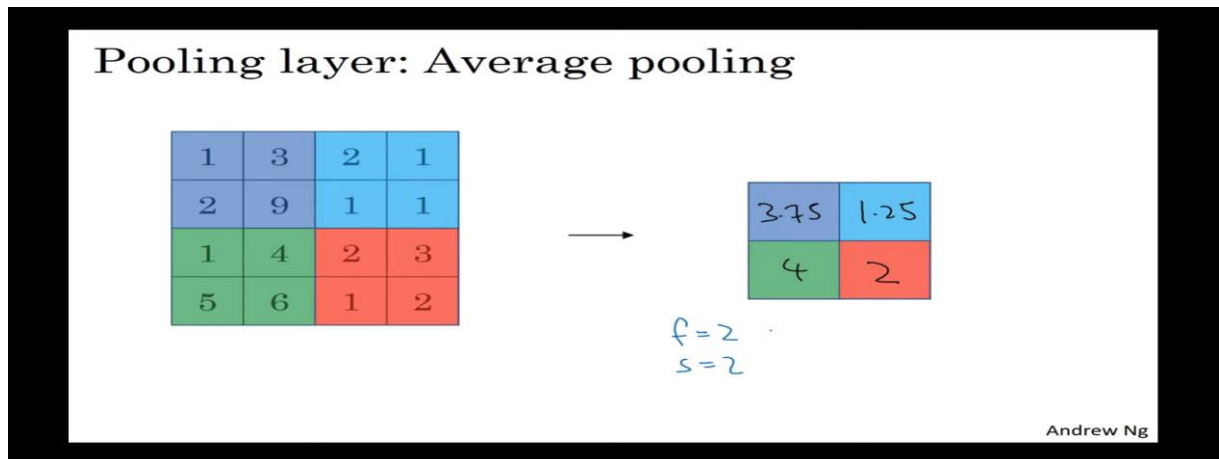
Pooling layer için herhangi bir parametre olmadığına dikkat et, yani bu layer için gradient descent ile öğrenilecek bir weight vb. söz konusu değil; f,s ve p gibi hyperparameters tune edildikten sonra gradient descent doesn't change anything.

Aşağıda f=3 ve s=1 için bir başka max pooling örneği görünüyor, 5x5'lik bir input için çıktı aşağıdaki formülle (convolution layer için kullanılanın aynısı) hesaplanabilir.

Burada bir başka nokta şu, eğer input için channel size = nc ise output da nc channel'lı olacaktır. Yani inputun her channel'ı için aynı parametrelerle bağımsız bir şekilde max pooling yapmış oluyoruz. Filters için durum farklıydı, feature detect edildikten sonra (3D filtre ile yapıyordu) output yani map 2D yine çıkıyordu. Burda pooling window 2D kalmaya devam ediyor output 3D oluyor.



Bir başka pooling yöntemi average pooling'dir, mantık aynı sadece max yerine average alınıyor. Çok nadiren derin layerlarda kullanılır mesela 7x7x1000'lik input alır 1x1x1000'lik average outputu verir.



Özetle, pooling için öğrenilen bir parametre yoktur. f veya s gibi hyperparameters tune edilir genelde $f=2, s=2$ seçilir, böylece inputun height and width'i output da yarıya düşürülür, bunu aynı convolution gibi formülle hesaplayabiliriz.

Padding pratikte pooling haftaya göreceğimiz bir exception dışında neredeyse hiç kullanılmaz.

Summary of pooling

Hyperparameters:

- f : filter size $f=2, s=2$
 $f=3, s=2$
- s : stride
- Max or average pooling

~~p : padding~~

No parameters to learn!

$$n_H \times n_W \times n_C$$
$$\downarrow$$
$$\left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_W - f}{s} + 1 \right\rfloor$$
$$\times n_C$$

Andrew Ng

Yukarıda sağda verilen input size için ve output size formülü yer alıyor.

CONV – RELU – POOLING sıralaması kullanılıyor, önce convolve ederiz sonra non-linearity ekleriz, en son pooling yaparız.

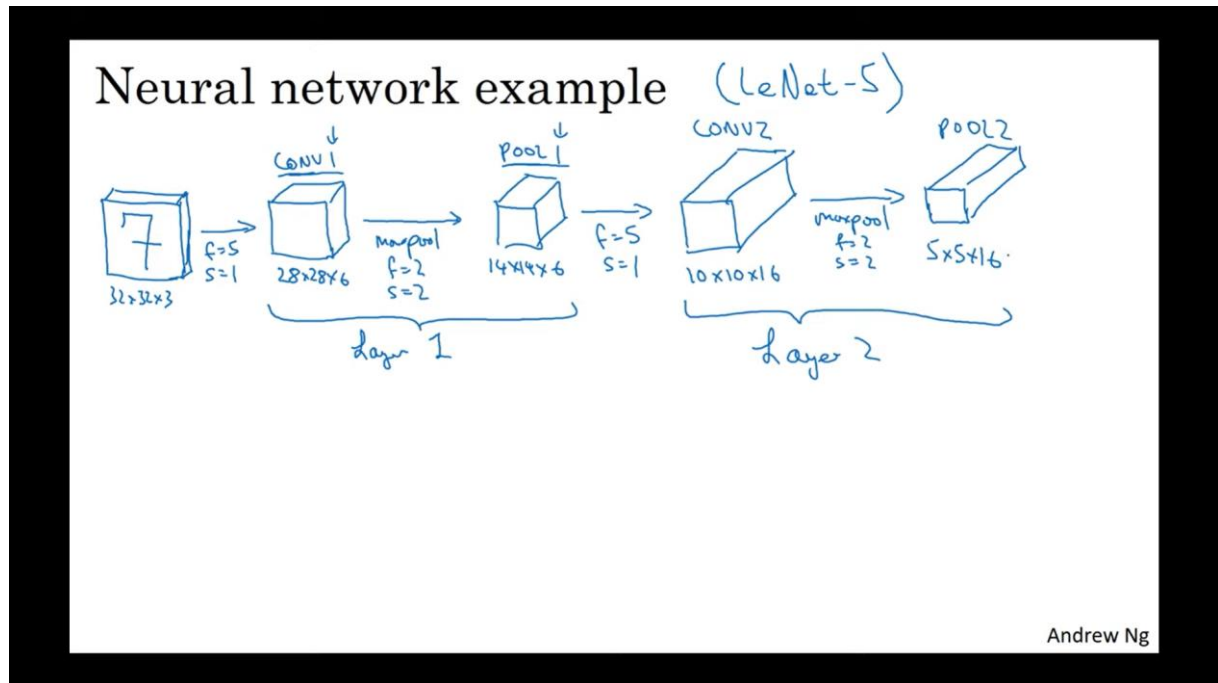
Vid 10

CNN Example

Önceki kısımlarda CNN için tüm building blocks'u öğrendik, bu kısımda bir CNN örneğine bakalım.

Diyelim ki hand written digit recognition yapmaya çalışıyoruz, 32x32x3 lük RGB image input olarak alınıyor.

- İlk layer bir conv. layer ve 6 farklı filtre kullanılıyor. $F=5$ yani filter size 5x5x3 olmalı.
- Sonuçta her filter 28x28 output verecek 6 tane filtre olduğu için 28x28x6'lık bir output bekleriz.
- Daha sonra bir pooling layer kullanacağız, bunun için $f=2$, $s=2$ ve max pooling kullanalım. Channel size korunacak, çünkü her channel için bağımsız olarak pooling yapılıyor, bundan bağımsız olarak height ve width yarıya düşecek yani 14x14x6'lık bir output elde etmiş olacağız.
- Burada önemli bir convention şu: genelde Conv+Pooling layer toplamına tek bir layer denir, çünkü pooling layer'ın herhangi bir parametresi yoktur, yani bu kısma kadar tek bir layer'ı tamamlamış olduk.



Buna bir layer daha ekleyelim, benzer şekilde önce $f=5$ $s=1$ 'lik 16 tane filtre (5x5x6 size'ında olacak) kullanalım sonucunda 10x10x16'lık bir output elde ederiz buna da yine $f=2$, $s=2$ 'lik bir pooling işlemi uygularsak 5x5x16'lık bir output elde etmiş oluruz.

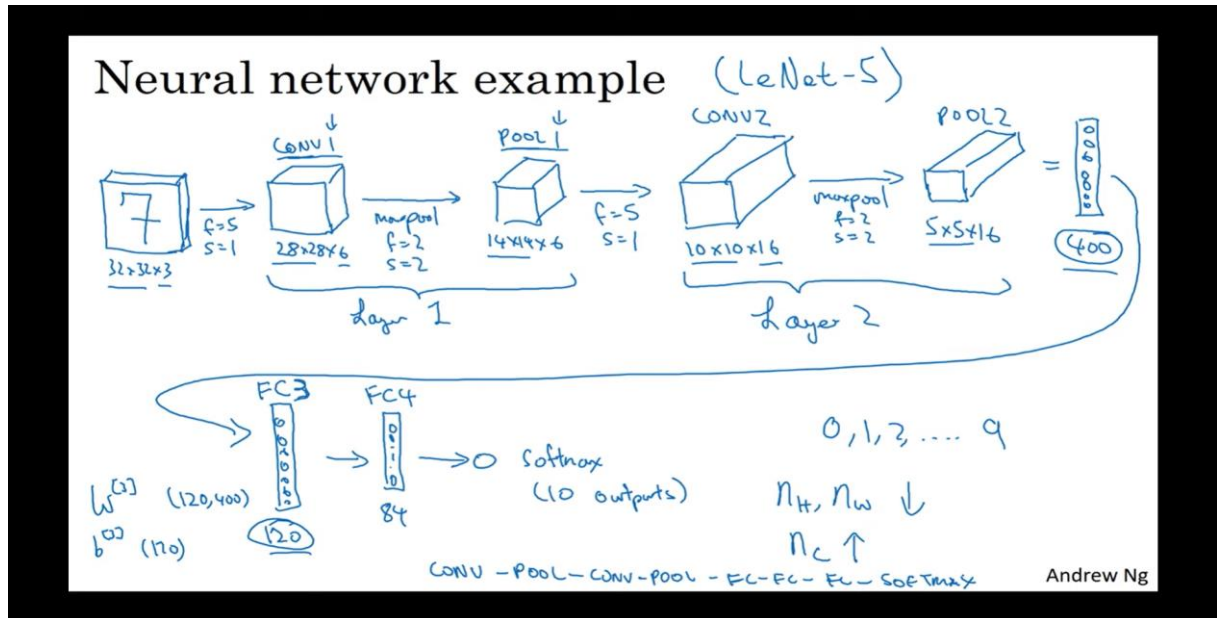
İkinci layer da böylece tanımlandı.

Filtering uygulandığında output depth'inin filter sayısına bağlı olduğunu buna karşın pooling uygulandığında output depth'in input depth ile aynı kaldığına dikkat et.

Şimdi elde edilen 5x5x16'lık output'u flatten edelim ve elimizde 400 tane input olmuş oluyor, bu input'u artık fully connected bir NN'in inputu olarak kullanabiliriz.

Fully connected network'ün ilk layer'ı olarak 120 unitlik aşağıda görülen FC3 layer'ını tanımlayabiliriz. Bu layer için bir W3 söz konusu ve boyutu (120,400) olacak.

Bunun ardından bir başka fully connected layer FC4 84 unit ile tanımlanabilir. Ve son olarak belki 10 unitli bir softmax unit kullanabiliriz, böylece CNN ile 10 farklı class (0,1,...9) detect çalışırız.



Burada bir sürü hyperparameter söz konusu, bunların nasıl seçilebileceğinden daha sonra bahsedeceğiz ancak bir öneri şu olabilir, kafana göre hyperparameter set etmek yerine, literatürde kullanılan değerlere bakmak yararlı olabilir.

Şimdilik şunu söyleyebiliriz, as we go deeper in the network, nh and nw decreases, yani input height ve width 32x32'den 5x5'e kadar düşer, buna karşın number of channels giderek artar, 3'ten 16'ya kadar çıktı.

Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072 $a^{(0)}$	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208
POOL1	(14,14,8)	1,568	0
CONV2 (f=5, s=1)	(10,10,16)	1,600	416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,001
FC4	(84,1)	84	10,081
Softmax	(10,1)	10	841

Andrew Ng

Son olarak yukarıda farklı layer'lar için shape, size ve # of parameters bilgisi yer alıyor. Burada bazı noktalar şunlar: → Pooling layers has no parameters. → Conv layers has relatively low number of parameters. → Activation size gradually decreases as we go deep in the NN.

Vid 11

Why Convolutions?

Bu kısımda neden convolution'ın işlevsel olduğunu anlayacağız ve her şeyi birleştirip bir CNN'i nasıl eğitebileceğimizden bahsedeceğiz.

Convolutional layers'ın fully connected layers'a göre iki avantajından söz edebiliriz:

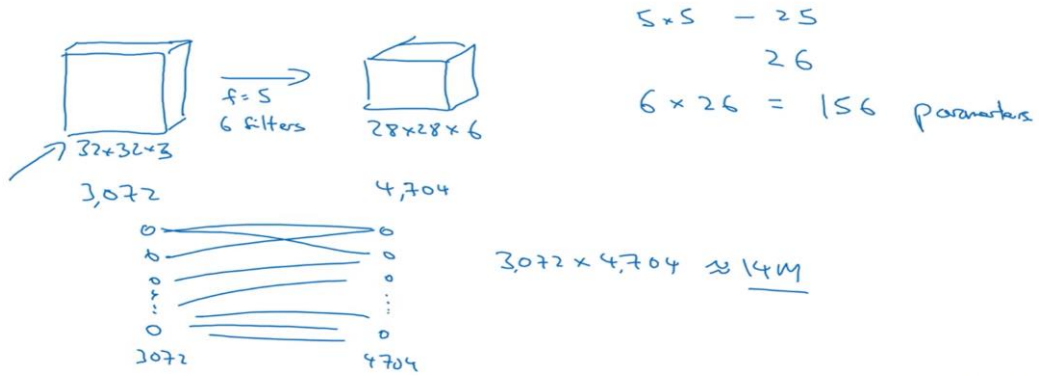
- Parameter sharing
- Sparsity of connections

Anladığım kadarıyla bu iki özellik, daha az parametre ile feature detect edebilmemize olanak sağlıyor, yani asıl olay fully connected layers kullanmış olsaydık çok çok büyük networkler yani parametreler kullanmamızı gerektirecek durumlarda (ki bu durumda overfitting ihtimali çok fazla, çünkü bu kadar büyük inputlar için overfittingten kurtulmak için çok çok daha büyük data'ya ihtiyacımız olacak) convNet yardımımıza yetişiyor ve çok küçük sayıda parametre ile büyük inputlardan (high res. images) feature detect edebiliyoruz.

Bir örnekle durumu daha iyi anlamaya çalışalım, diyelim ki 32x32x3'lük bir RGB image'im input, buna 5x5x3'lük 6 tane filtre uyguluyoruz ve convolutional layer sonucu output 28x28x6 oluyor.

İlk resmi flatten etseydik, bu 3072 tane input anlamına geliyor, benzer şekilde output da 4704 tane value tutuyor. Fully connected layer ile 3072 tane inputu alıp 4704 tane output'u çıkarmak istersek sadece bunun için 14M parametre ortaya çıkıyor. Bu belki inanılmaz derecede fazla değil ama 32x32 px bir image çok çok küçük sayılır, böyle küçük bir image için bile fully connected layers kullanmak istersek, işler çok zorlaşır, bir de 1000x1000 px bir image kullandığımızı düşün.

Why convolutions



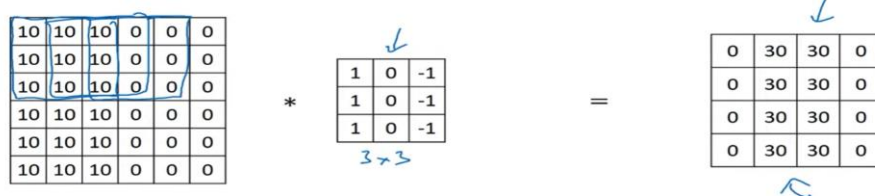
Ancak bu işlem için convolution layer kullandığımızda filtreler 5x5 olduğu için 25 + constant = 26 parametre bir filtre için (sınırım depth'in 3 olması önemli değil çünkü her üç channel için de aynı parametreler kullanılacak.) Total 6 filtre olduğuna göre 156 parametre ile bu işlemi yapmış olduk.

156 nerede 14M nerede!

ConvNet'in relatively small number of parameters'a sahip olmasının da iki nedeni var.

- İlki parameter sharing.
 - Bu şu demek, fully connected layer kullansaydık, her input pixel için ayrı bir weight (her input için output sayısı kadar ayrı weight) kullanacaktık, ancak convolution 3x3'lük bir filtreyi resmin farklı kısımlarında feature detect etmesi için kullanmış oluyoruz. Sonuçta totalde $3 \times 3 + 1 = 10$ parametre ile bir feature'u resmin her yerinde detect etmiş oluyoruz.

Why convolutions



Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

Andrew Ng

- İkincisi ise sparsity of connections

- Bu şu demek, tek bir output value hesaplanırken tüm inputlar işin içine girmiyor, fully connected layers da olan bu, her output'un hesaplanması için bütün input'lar işin içine giriyor, bu yüzden tonlarca parametre kullanılıyor.
- ConvNet için ise bir output hesabı için sadece 3x3'lük bir input area kullanılıyor.

Why convolutions

translation invariance

Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

→ **Sparsity of connections:** In each layer, each output value depends only on a small number of inputs.

Andrew Ng

Bu açıklamalardan sonra, CNN'i daha iyi anladım, aslında NN ile benzer bir şey yapıyor, ancak çok daha az parametre kullanıyor, input resmi flatten ettiğimizi düşün ve CNN'in yaptığı işlemleri öyle anlamaya çalış, 3x3 bir filtre olsun bunu 9 weight gibi düşün.

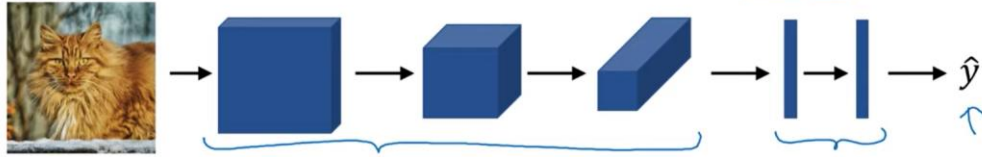
Aynı NN'de olduğu gibi 9 weight input'un ilk 9 unit'ine bağlanmış sonucunda tek bir output hesaplanıyor, sonra aynı weight'ı bir aşağıya (stride=1) kaydırıyorsun ve orada aynı 9 weight ile bir başka output hesaplıyorsun. NN için ise her input-output pair için bir weight var bu yüzden çok fazla parameter var. CNN aynı weightleri network boyunca kaydırıyor ve tek bir output'u hesaplamak için tüm inputları değil sadece çok küçük bir kısmını kullanıyor.

İşte bu iki sebepten dolayı, CNN FCNN'e göre çok daha az parameter ile çalışıyor böylece, image gibi çok sayıda input'u olan koşullarda network'ü overfitting'den kurtarıyor. Diğer türlü overfitting'ten kurtulmak için inanılmaz büyük sayıda veri toplamamız gerekirdi. Ayrıca CNN'in translation invariance'a karşı son derece robust olduğunu da söylerler, yani input image saga sola kaydırılmış olarak alınsa da network gayet iyi çalışır.

Sonunda bir CNN'in eğitimi ile FCNN'in eğitimi arasında belirgin bir fark yok, yine network boyunca w ve b parametreleri var, yine bir outptumuz var, cost function'ımız tanımlı. Bu cost'u minimize etmek için gradient descent kullanıyor.

Putting it together

Training set $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$.



$$\text{Cost } J = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Use gradient descent to optimize parameters to reduce J

Andrew Ng

Cost'u hesaplamak için forward propagation kullanılırken, gradient descent için gerekli gradients hesaplamaları için cost function'dan başlayarak backpropagation kullanılır.