

COURSE4 – W3

Vid 1

Object Localization

Şimdiye kadar ConvNets ile ilgili birçok şey öğrendik. Bu hafta da ConvNets'in bazı special applicationlarından bahsedeceğiz: "Face Recognition" ve "Neural Style Transfer" gibi.

Önce Face Recognition ile başlayacağız. RFID kart okuyan iş yeri vb. gibi turnikeli yerlerde, artık face recognition kullanılabilir. Üstelik liveless detection ile, system gördüğü yüzün gerçek mi yoksa sadece bir fotoğraf mı olduğunu ayırt edebiliyor. Bu liveless detection'ı supervised learning ile yapabiliyoruz, live human vs picture. Ancak bu konu üzerinde çok durmayacağız.

Öncelikle Face Recognition için kullanılan terminology'e bakalım.

- Face Verification:
 - Bu problem için, input image ve ID veriliyor, systemin görevi bu eşleşme doğru mu değil mi onu söylemek.
 - Yani bu image bu ID'ye mi ait onun cevabını arıyoruz.
 - Bu yüzden bazen bu probleme 1:1 problem.
- Face Recognition:
 - Bu ise 1:K problemidir.
 - Verification'dan çok daha zordur, burada input image'ı K person'lı bir database'den bir ID ile eşlememiz gerekir.

Verification için 99% accuracy iyi görünse de Recognition için K=100 olan bir database için 99% iyi bir oran değildir. Her bir insan için %1 hata şansımız var, yani 100 tane insan için bir tane hata gelmesini bekleyebiliriz.

Face verification vs. face recognition

→ Verification

- Input image, name/ID
- Output whether the input image is that of the claimed person

1:1 99.9%

→ Recognition

- Has a database of K persons
- Get an input image
- Output ID if the image is any of the K persons (or "not recognized")

1:K K=100 ←

Andrew Ng

Önce building block olarak Verification sistemi geliştireceğiz daha sonra bunu genelleştirerek, recognition sistemi geliştireceğiz.

Vid 2

One Shot Learning

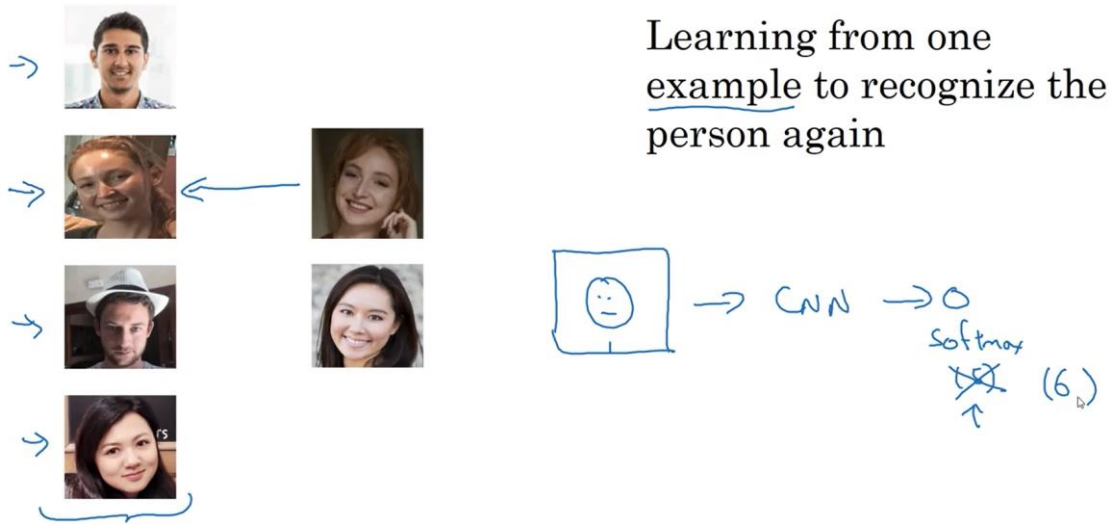
Face recognition'ın challenge'larından biri şu: One-shot learning problemini çözmemiz gerek. Yani bir insanın elimizde sadece tek bir fotoğrafı olacak, ve yalnızca bu fotoğrafı kullanarak kişiyi daha sonra tanıyabilmemiz gerekiyor.

İlk bakışta akla şu geliyor, elimizde tek bir örnek varsa modeli nasıl eğiteceğiz? Elbette tek bir örnekle bir deep learning modeli eğitemeyiz, bunun için başka yaklaşımlar kullanılacak.

Diyeelim ki iş yerinde 4 kişi çalışıyor, aşağıdaki gibi initial fotoğrafları hazır, modelimizin bu fotoğraflara bakarak, kişileri tanıması gerekiyor.

- Yani ofise bu 4 kişiden biri gelirse system kişiyi tanımalı ve kapıyı açmalı.
- Ancak bu database'de olmayan biri gelirse system kapıyı açmamalı.
- Bir insanı tanımak için onun yalnızca tek bir fotoğrafını kullanıyoruz, bu yüzden bu bir One-Shot Learning problem olarak adlandırılıyor.

One-shot learning



Andrew Ng

Böyle bir modeli build etmek için akla gelen ilk yaklaşım, fotoğrafı input eden ve daha sonra CNN ile işleyip, softmax ile 4 kişiden birini veya tanımadım'ı seçen bir model eğitmek. Ancak elimizde 4 tane fotoğraf var, bu kadar küçük bir dataset ile model eğitmek imkansız.

Onu geçtim, diyelim ki database'de olan ve olmayan kişilerin yüzbinlerce fotoğrafını elde ettik bir şekilde, sonra şirkete yeni biri katıldı, bu sefer modelin çıkışı değişecek ne yapacağız, yeni kişinin binlerce fotoğrafını bulup modeli tekrar mı eğiteceğiz?

Yani bu yaklaşımla modelin öğrenmeye çalıştığı şey, bir fotoğrafa bakıp, kişinin ağzını yüzünü güzelce inceleyip, doğrudan bu kişinin kayıtlı 4 kişiden biri olup olmadığını anlamak. Bu zor bir iş.

Bu yaklaşım, tırt, bunu sal.

E o zaman ne yapacağız?

Yapacağımız şey, modele bir similarity function öğretmek. Yani modelin yapacağı iş iki fotoğrafı input olarak almak ve bu iki fotoğrafın aynı kişi olup olmadığını söylemek olacak.

- Böylece modelin öğreneceği fonksiyon bizim çalışanlarımızın yüz şekli olmaktan çıkıyor, modelin öğrenmesi gereken tek şey iki ayrı fotoğrafa bakıp aynı kişi olup olmadığını söylemek.
- Böylece modelimizi eğitmek için kendi çalışanlarımızın fotoğraf dataset'i ile bağlanmayız, similarity function'ı öğrenmesi için modele her yerden data sağlayabiliriz, insan olsun yeter.
- Eğer model img1 ve img2'yi alıp difference olarak seçilen bir threshold'dan büyük bir değer döndürüyorsa bunlar farklı kişilerdir deriz, eğer düşük bir değer döndürüyorsa aynı kişidir deriz.

Learning a “similarity” function

→ $d(\text{img1}, \text{img2}) = \text{degree of difference between images}$

If $d(\text{img1}, \text{img2}) \leq \tau$
 $> \tau$

“same”
“different”

} Verification.



Andrew Ng

Bu yaklaşımla One-Shot Learning Problem'i de çözmüş oluyoruz.

Vid 3

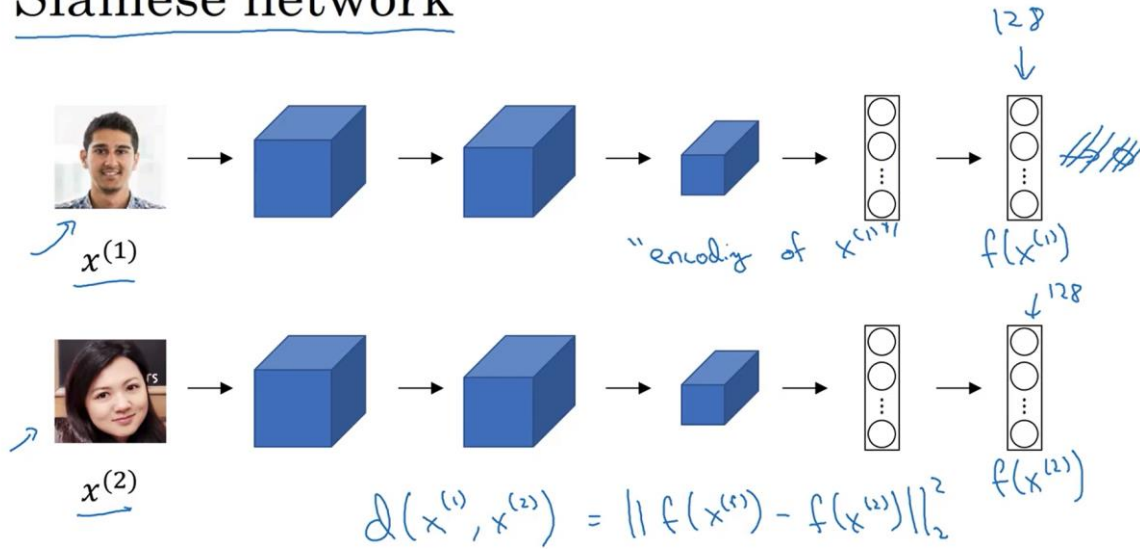
Siamese Network

Bir önceki kısımda gördüğümüz difference function'ın görevi, iki farklı face image'ı input olarak alıp bunların ne kadar benzer veya farklı olduğunu döndürmek.

Bu işlemi yapmak için Siamese Network kullanılabilir. Şimdi buna bakalım.

- Aşağıda görülen gibi ConvNet'lere zaten hakimiz. Diyelim ki x_1 input'u alınıyor, ve sonucunda bir ConvNet ile 128 feature elde ediliyor.
- Genelde bu 128 feature'ı bir softmax'e bağlayıp sonuç elde ederiz, ancak burada öyle yapmıyoruz.
- Burada bu 128 feature'a $f(x_1)$ diyelim yani bu 128 feature'ı input image1'in encode edilmiş hali olarak düşün.
- Benzer şekilde, eğer aynı network'e bir başka input image x_2 'ye beslersek bu da bize x_2 'yi encode edecek ve 128 tane feature verecektir.

Siamese network



[Taigman et. al., 2014, DeepFace closing the gap to human level performance]

Andrew Ng

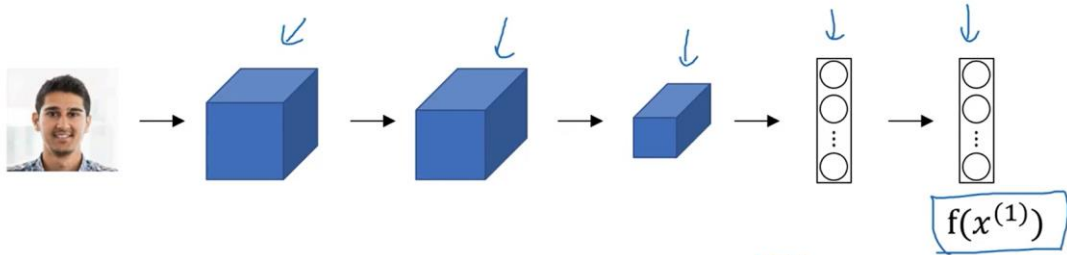
İşte bu network'un her input image için input image'ı başarıyla temsil eden 128 feature verdiğini düşünürsek, bu encoded feature'ların yukarıda görüldüğü gibi farkını alarak, verilen iki resmin aynı kişi olup olmadığını anlayabiliriz.

Yani özetle yukarıdaki gibi bir ConvNet kullanarak, difference function'ı yukarıda görüldüğü gibi tanımlayabiliriz.

Şimdi soru şu, böyle bir Siamese Network'u nasıl train ederim?

- Öyle bir NN train etmeliyiz ki, parameterler input image'ı 128 tane feature verecek şekilde encode etmeli
- Bununla da kalmıyor, öyle bir NN train etmeliyiz ki, NN bulduğu o parametreler eğer aynı kişinin iki fotoğrafını encode ederse sonuçlar benzer çıkmalı yani difference function düşük çıkmalı, ancak iki farklı kişinin fotoğrafını encode ederse difference büyük çıkmalı.

Goal of learning



Parameters of NN define an encoding $f(x^{(i)})$ 128

Learn parameters so that:

If $x^{(i)}, x^{(j)}$ are the same person, $\|f(x^{(i)}) - f(x^{(j)})\|^2$ is small.

If $x^{(i)}, x^{(j)}$ are different persons, $\|f(x^{(i)}) - f(x^{(j)})\|^2$ is large.

Andrew Ng

Kısacası model fotoğrafa bakarak, bir insanı diğer insandan ayırt eden 128 feature'ı verecek parametreleri bulacak şekilde eğitilmeli.

Bu parametreleri sağlamak için nasıl bir objective function tanımlamalıyız? Önümüzdeki başlıkta ona bakalım.

Vid 4

Triplet Loss

Ayırt edici özellikler ile encoding sağlayacak bir Siamese Network eğitebilmek için Triplet Loss function'ı kullanılabilir.

Bu kısımda triplet loss function'ı anlamaya çalışalım.

- Triplet loss'un tanımı gereği, tek seferde 3 image kullanılır. Bir Anchor bir Positive yani aynı kişinin bir başka fotoğrafı ve bir de Negative yani başka bir kişinin fotoğrafı.
- Sonuçta amacımız bir Siamese Network eğitmek ki, aynı kişinin farklı fotoğrafları için küçük bir difference verirken, farklı kişilerin fotoğrafları için büyük bir difference vermesi.

Learning Objective

Anchor A $d(A,P) = 0.5$

Positive P

Anchor A $d(A,N) = 0.7$

Negative N

Want: $\frac{\|f(A) - f(P)\|^2}{d(A,P)} + \alpha \leq$

$\frac{\|f(A) - f(N)\|^2}{d(A,N)}$

$\frac{\|f(A) - f(P)\|^2}{0} - \frac{\|f(A) - f(N)\|^2}{0} + \alpha \leq 0$ α margin $f(\text{img}) = \vec{o}$

[Schroff et al., 2015, FaceNet: A unified embedding for face recognition and clustering]

Andrew Ng

- Yani formally, ben Anchor ve Positive image için encoding yapıldığında oluşan difference d1'in, Anchor ve Negative image için encoding yapıldığında oluşan difference d2'den çok daha küçük olmasını istiyorum.
- Sadece d1<=d2 olması yetmiyor çünkü o zaman network, her input için d1 ve d2'yi 0 yapacak parametreleri seçebilir, veya d1=d2 yapacak parametreler seçilebilir.
- İşte bu yüzden işin içine bir alfa hyperparametresi giriyor, d1+alfa <= d2 olsun diyoruz veya d1 - d2 + alfa <=0 olsun diyoruz yani kısaca d2, d1'den zaten büyük olsun, ama yetmez d1+alfa'dan da büyük olsun kardeşim diyoruz.
- Biz istiyoruz ki aynı Anchor ve Positive için çok küçük bir d değeri bulunsun Anchor ve Negative için ise çok büyük bir d değeri bulunsun.
- Yani anchor ile P için difference d(A,P) = 0.5 çıktıysa, d(A,N)'nin 0.51 çıkması yeterli değil, çok daha büyük çıkması gerekiyor mesela 0.7 bu yüzden alfa kullanıyoruz.

Alfa hyperparameter'ı margin olarak da adlandırılır.

Triplet loss function'ın ne olduğunu anladık şimdi bunu formalize edelim:

- Given 3 images A, P, N. (Anchor, Positive, Negative)
- $\text{Loss}(A,P,N)$ aşağıdaki gibi tanımlanacak: eğer $d(A,P) - d(A,N) + \alpha \leq 0$ eşitsizliği sağlanıyorsa, ilgili A,P,N training örneği için loss 0 alınacak, çünkü zaten istediğimiz bu eşitsizliğin sağlanması.
- Yok eğer bu eşitsizlik sağlanmıyorsa, o zaman çıkan $d(A,P) - d(A,N) + \alpha$ değeri loss olarak alınacak.
- Tüm training set için cost function da bildiğimiz gibi her example'ı için hesaplanan loss'ların toplamı veya ortalaması şeklinde hesaplanabilir.

Loss function

Given 3 images A, P, N :

$$\mathcal{L}(A, P, N) = \max(\underbrace{\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha}_{\geq 0}, 0)$$
$$J = \sum_{i=1}^m \mathcal{L}(A^{(i)}, P^{(i)}, N^{(i)})$$

A, P

Training set: 10k pictures of 1k persons

[Schroff et al., 2015, FaceNet: A unified embedding for face recognition and clustering]

Andrew Ng

Eğer elimizde 1k insan için 10k picture varsa, yapmamız gereken bu 10k picture ile A P N şeklinde m tane triplet oluşturmamız daha sonra, learning algoritmamızı (Siamese Network olabilir) bu training set ile GDA veya bir başka optimizasyon algoritması kullanarak eğitmek.

Böyle eğitilen bir model en nihayetinde, aynı kişinin iki farklı fotoğrafı için düşük difference değerleri, buna karşın farklı kişilerin fotoğrafları için yüksek difference değerleri verecek feature'ları encode eden bir model olacaktır.

A ile P'nin aynı kişinin başka fotoğrafları olduğunu unutma, bu yüzden yukarıda 1k insan için 10k fotoğraf'lık bir training setten bahsediliyor.

Peki bu tripletleri nasıl seçeriz?

- A,P,N tripletlerini training set içerisinde random olarak seçersek, $d(A,P) + \alpha \leq d(A,N)$ constraint'i kolaysa satisfied olacaktır.
- Bunu şöyle düşün A ile P aynı insanın iki farklı fotoğrafı olsun N ise gökyüzü fotoğrafı olsun, bu durumda modelin iyi cost değerleri veren bir model eğitmesi çok kolay olacaktır, sonuçta model sadece kaş göz gibi şeylerin varlığını feature olarak output etse yetecektir.
- Ancak gerçek hayatta iki farklı insan'ı gösterdiğin zaman model afallayacak ve hata yapacaktır, çünkü ikisinde de göz ve burun var o zaman aynı kişi diyecektir.
- İşte bu yüzden, triplet'e Negative seçilirken, Anchor'a olabildiğince benzeyen bir negative seçmek daha iyi olacaktır, böylece modeli iki benze rinsanı dahi ayırt edebilecek detay feature'ları encode edecek şekilde eğitmiş oluyoruz.

Choosing the triplets A,P,N

During training, if A,P,N are chosen randomly,
 $d(A,P) + \alpha \leq d(A,N)$ is easily satisfied.

$$\|f(A) - f(P)\|^2 + \alpha \leq \|f(A) - f(N)\|^2$$

Choose triplets that're "hard" to train on.

$$\frac{\mathcal{L}(A,P) + \alpha}{\mathcal{L}(A,P)} \leq \frac{\mathcal{L}(A,N)}{\mathcal{L}(A,N)}$$

↓ ↑

Face Net
Deep Face

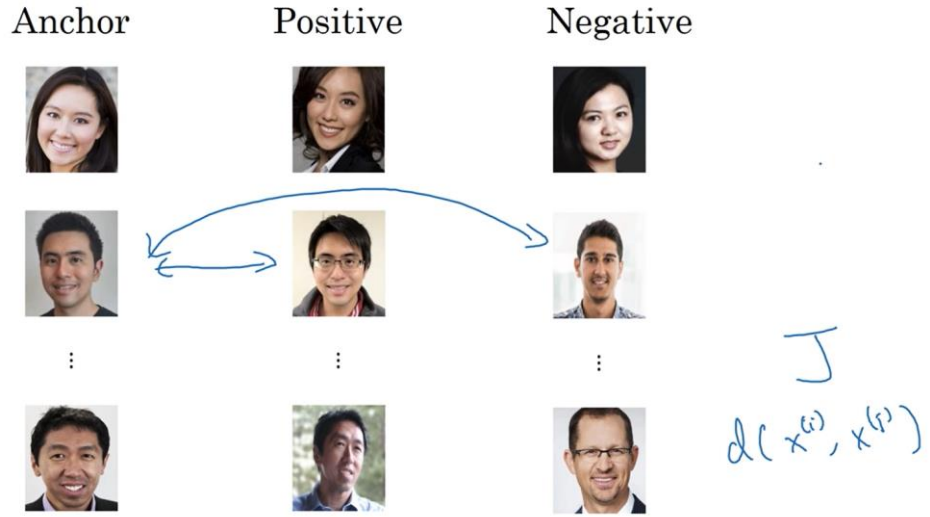
[Schroff et al., 2015, FaceNet: A unified embedding for face recognition and clustering]

Andrew Ng

Sonuçta toparlarsak:

- Triplet loss function ile bir eğitim yapmamız için, öncelikle training fotoğraflarını alıp A,P,N şeklinde tripletlere ayırmam lazım, her üçlü bir training example oluyor.
- Böyle tripletlerden oluşan bir training set oluşturduğum zaman, daha önce gördüğümüz Siamese Network gibi bir network'ü bu dataset ile eğitebilirim ve loss olarak triplet loss'ı kullanırım.
- Böylece eğitim sonucunda modelim detaylı feature'lar encode etmeyi öğrenmiş olacaktır ve aynı kişinin iki farklı fotoğrafı için düşük bir difference output ederken, farklı kişiler için yüksek bir difference output edecektir.

Training set using triplet loss



Andrew Ng

Vid 5

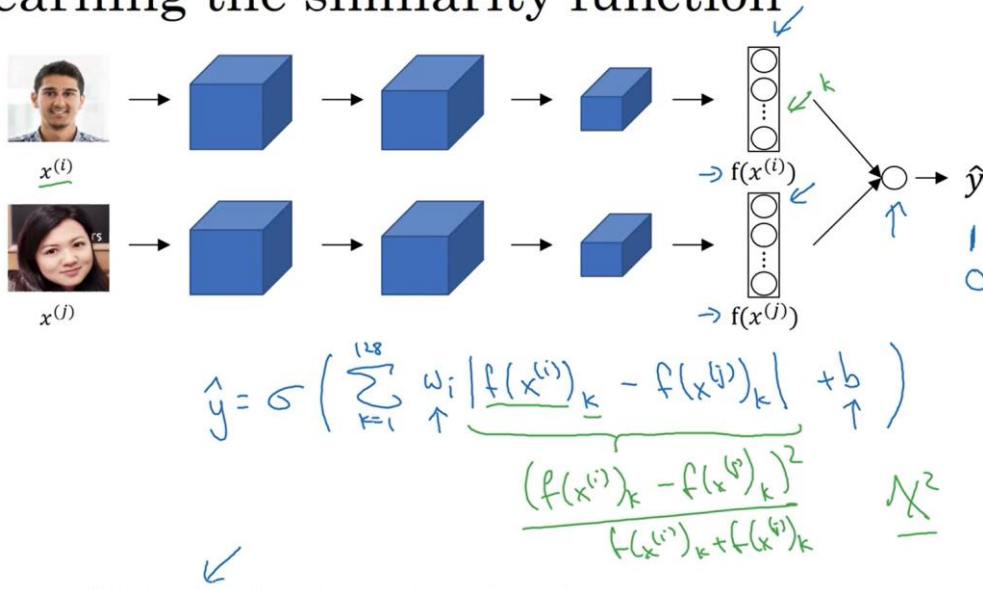
Face Verification

Önceki kısımda Triplet Loss function ile nasıl bir face recognition sistemi eğitebileceğimizi kavramıştık.

Bu kısımda başka bir yaklaşımı ele alacağız, face recognition problemine binary classification problemi gibi yaklaşacağız.

- Önceki kısımda yaptığımız şey, training set'i tripletlerden oluşturup, bir Siamese Network'u bu triplet data set ile minimum triplet cost'u verecek şekilde eğitmek idi.
- Yani Siamese network'e tek sefer 3 fotoğraf veriliyordu A,P,N sonucunda network her birini 128 feature ile encode ediyordu, ve eğer bu encoding sonucunda $d(A,P)$, $d(A,N)$ 'den küçükse sorun yok, eğer küçük değilse, network parametreleri ona göre eğitiliyordu.

Learning the similarity function



[Taigman et. al., 2014. DeepFace closing the gap to human level performance]

Andrew Ng

- Buna bir alternatif ise şu, training set iki fotoğraftan oluşur ve sonucunda tek bir label olur.
- Yani eğer bir örnekte aynı kişinin iki fotoğrafı varsa bunun label'ı 1 olsun, diğer türlü label 0.
- Daha sonra Siamese Network'ü biraz modifiye edip, 128 feature'ın farkları görüldüğü gibi bir logistic regression'a verip output olarak 0 veya 1 beklenebilir.
- Model yine, insan yüzünün detaylarını ayırt edebileceği iyi feature'ları encode etmeyi öğreniyor, bunun yanında bir de classification boundary öğreniyor.
- Sonuçta aynı iki insanın 128 feature'larının farkı uzayda benzer noktalara düşerken farklı insanlarınki farklı noktalara düşecek.



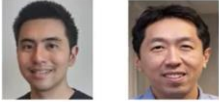

Logistic regression unit'e doğrudan 256 encoding'i beslemediğimize dikkat et, bunun yerine x_1 için elde edilen 128 feature – x_2 için elde edilen 128 feature diyerek yeni 128 feature elde ediyoruz, bu 128 yeni feature'ı difference'ın bir ölçüsü olarak düşünebilirsin. Daha sonra bu 128 feature'ı logistic regression'a besliyoruz.

Sonuçta model eğitildikten sonra, yapacağı iş şu olacak, turnikenin başındaki kameradan bir image alacak, daha sonra database'den bir image alacak ve ikisi için de 128 feature hesaplayacak daha sonra bunların farkını alacak ve logistic regression unit'e besleyerek bu iki kişinin aynı kişi olup olmadığı söylenecek.

- Bu noktada bir computational trick yapılabilir, database'de tutulan resimler için 128 feature precompute edilebilir, böylece turnike kamerasının başına biri geldiğinde sadece yeni gelen image için 128 feature hesaplanır, datasetteki herkes için tek tek 128 feature hesaplamaktan kurtulmuş oluruz.

Sonuçta toparlarsak, triplets of images yerine pair of images kullanarak, supervised learning ile de face verification sistemi geliştirebiliriz. Sonuçta problem binary classification'ı indirgeniyor ve bu yaklaşım da gayet başarılı bir şekilde işler.

Face verification supervised learning

x	y	
	1	"Same"
	0	"Different"
	0	
	1	

[Taigman et. al., 2014. DeepFace closing the gap to human level performance]

Andrew Ng

Vid 6

Neural Style Transfer

Şimdi de ConvNets'in eğlenceli ve exciting bir application'ından bahsedeceğiz. Neural Style Transfer.

Neural Style Transfer nedir?

- Aşağıda görüldüğü gibi, alınan bir content image'ın style image'ın stili ile recreate edilmesini sağlarız.
- Yani Generated image G, content image C'nin Style image S stili ile tekrar oluşturulmuş hali!

Neural style transfer

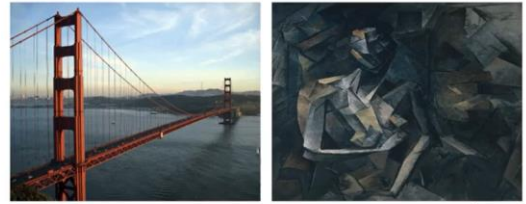


Content (C) Style (S)

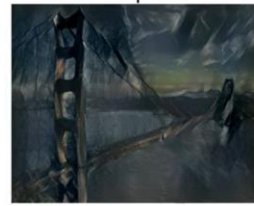


Generated image (G)

[Images generated by Justin Johnson]



Content (C) Style (S)



Generated image (G)

Andrew Ng

Yukarıdaki örnekler gerçekten etkileyici. İlerleyen kısımlarda da böyle image'ları nasıl generate edebileceğimizi anlayacağız.

In order to implement neural style transfer we need to look at the features extracted by convnet at various layers (shallow and deeper layers).

Neural style transfer'i iyice anlamadan önce, convnet'in layer'larının nasıl şeyler compute ettiğini intuitive olarak anlamaya çalışalım.

Vid 7

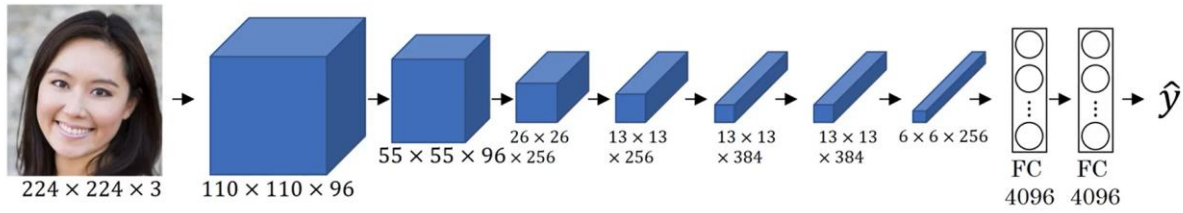
What are Deep CNNs learning?

Bu kısımda bir ConvNet'in nasıl feature'lar öğrendiğini visualize ederek kavramaya çalışacağız. Daha sonra bu intüition neural style transfer uygulaması için yararlı olacaktır.

Diyelim ki aşağıda görüldüğü gibi bir ConvNet train ettik, amacımız farklı layerlardaki hidden units'in nasıl feature'lar compute ettiğini görselleştirmeye ve anlamaya çalışmak.

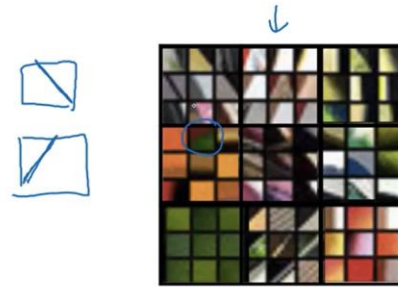
- Önce layer 1'de bir hidden unit seçelim.
- Diyelim ki tüm training set'i taradık ve hangi image kısımlarının bu unit'in aktivasyonunu maksimize ettiğini bulduk.
- İlk layerdaki bir unit'in aktivasyonu, bir filtrenin input image'ın belirli bir kısmına bakıp sonuç vermesi, yani input image'ın belirli bir kısmının non-linear bir fonksiyonu.
- İşte ne tip image kısımları için bu unit maksimize oluyor sorusunun cevabı aslında, bu unit nasıl bir feature detect etmeyi öğrenmiş sorusunun cevabı ile aynı. Bu sorunun cevabını görsel olarak arıyoruz.

Visualizing what a deep network is learning



Pick a unit in layer 1. Find the nine image patches that maximize the unit's activation.

Repeat for other units.



[Zeiler and Fergus., 2013, Visualizing and understanding convolutional networks]

Andrew Ng

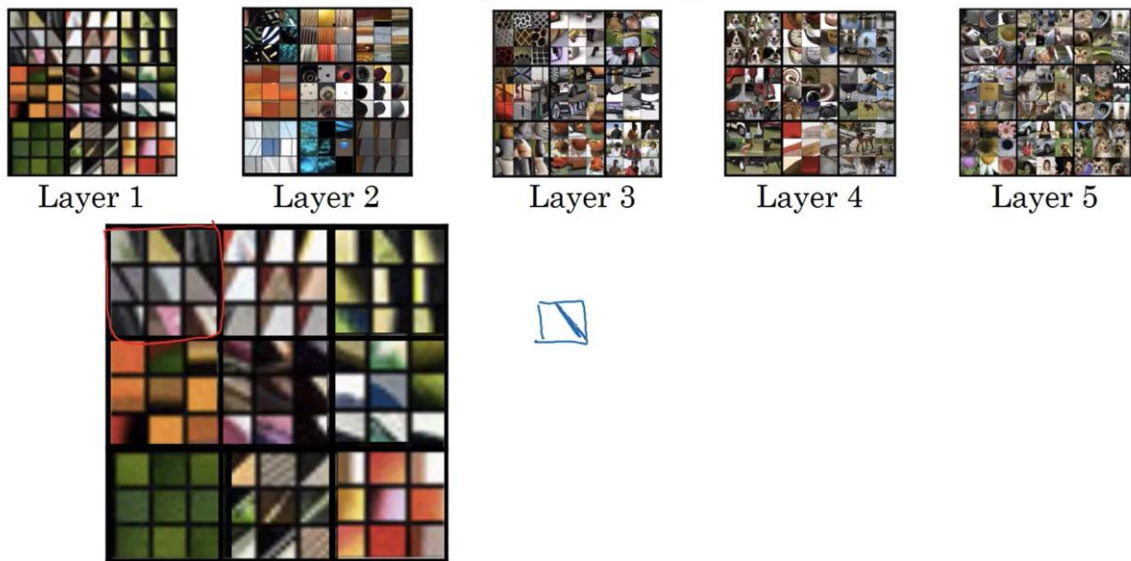
- Şunu unutma layer1'deki bir hidden unit, input image'ın çok küçük bir kısmını görecektir ve buna göre bir sonuç oluşturacaktır, yani diyelim ki 5x5 bir filtre kullandık, o zaman bu unit 5x5'lik bir image kısmında feature arar, bu kısımda pixellere tepki verir. Buna karşın bir sonraki layer, bu 5x5'lik kısımlarda feature detect edilmiş map üzerinde 5x5'lik kısımlarda feature detect ettiği için aslında original image üzerinde sanki 5x5x5'lik bir alana bakıyormuş gibi olacak, daha büyük alanlarda feature detect etmiş olacak.
- Yukarıda sağ altta görünün resmi şöyle yorumla, ilk 9 kutu şunu temsil ediyor: layer1'deki bir hidden unit'in aktivasyonunu maximize eden 9 adet input image patch'i.

- Buna karşın diğer 9 küçük kare de 1. Layer'daki bir başka unit'i maximize eden 0 input image patch'i gösteriyor.
- Bakınca görüyoruz ki, mesela ilk unit aslında 45 derecelik, edge'ler görünce maximize oluyor, yani 45 derecelik edge detect ediyor, buna karşın, bir başka unit yeşil parçalar görünce maximize oluyor bir başkası turuncular için vesaire.
- Ama genel kanı şu: ilk layerdaki unitler relatively simple features detect ediyor.

Aşağıda aynı işlemin farklı layerlar için uygulanması görselleştirilmiş, yani farklı layerlardaki farklı unitler alınmış ve ne tip input image kısımlarına yüksek aktivasyon verdiği, bir başka deyişle bu unit'lerin ne tip feature'ları detect edebilmek için eğitildiği görselleştirilmiş.

Az sonra her layer'ı detaylıca göreceğiz ancak genel kanı şu: Layer ilerledikçe unit'ler daha geniş bir alanda feature detect ediyor ve detect edilen feature'lar gitgide karmaşılaşıyor, bu mantıklı zaten fully connected layerda da olan bu. Tabi aşağıda hepsi aynı boyutta gösterilmiş, buna kanma, yani ilk layerdaki unitler 3x3'lük bir input image alanını baz alarak feature detect ederken, 2. Layerdaki unitler 3x3x3'lük bir input image alanını baz alıyor, ve bu her katmanda giderek genişliyor.

Visualizing deep layers: Layer 1



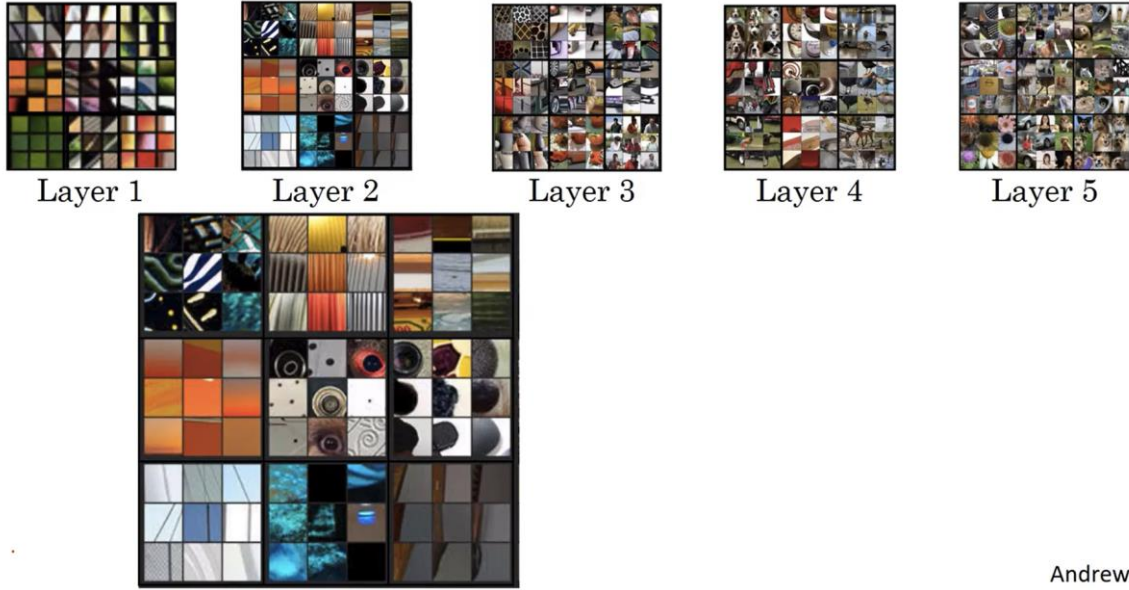
Andrew Ng

Yukarıda ilk layer'daki unitlerin edges, or specific colors gibi relatively simple feaure'lar detect ettiğini tekrar görüyoruz.

Aşağıda 2. Layer için seçilen 9 hidden unit'in nasıl feature'lar detect ettiği görülüyor. Netleştirmek için şunu belirteyim, her bir 9 küçük kare bir hidden unit'in tepki verdiği input image region'ları gösteriyor, büyük 9 karenin her biri ilgili layerdaki bir başka unit'e denk geliyor.

Dediğimiz gibi, 2. Layerda detect edilen feature'lar daha complex ve bunun yanında aslında daha büyük bir input image region'ı gözeterek detect ediliyor.

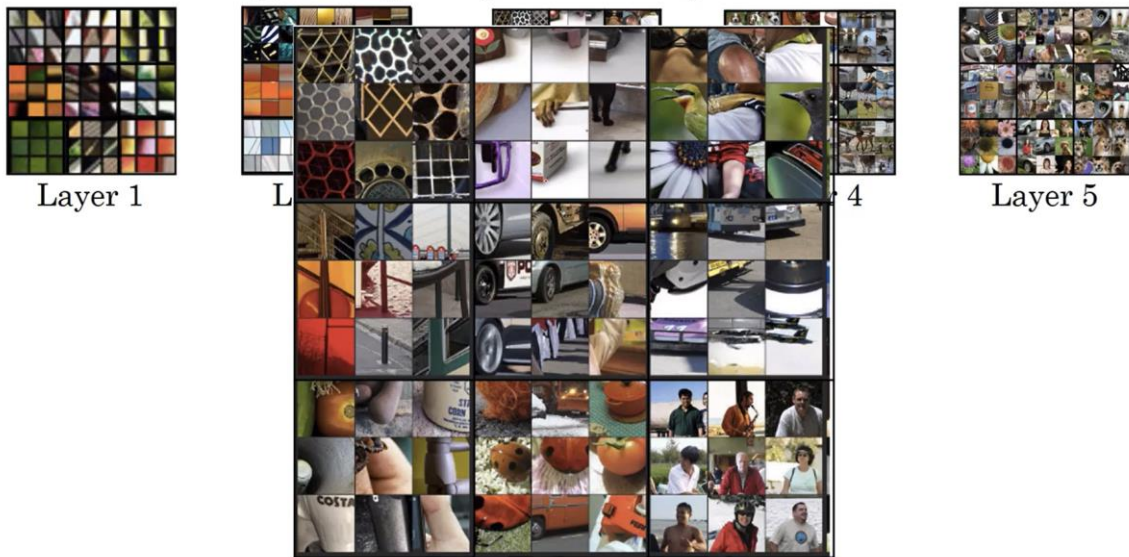
Visualizing deep layers: Layer 2



Andrew Ng

Benzer şekilde 3. layerdaki units için de detect edilen feature'ların bariz şekilde kompleksleştiğini aşağıda görebiliriz.

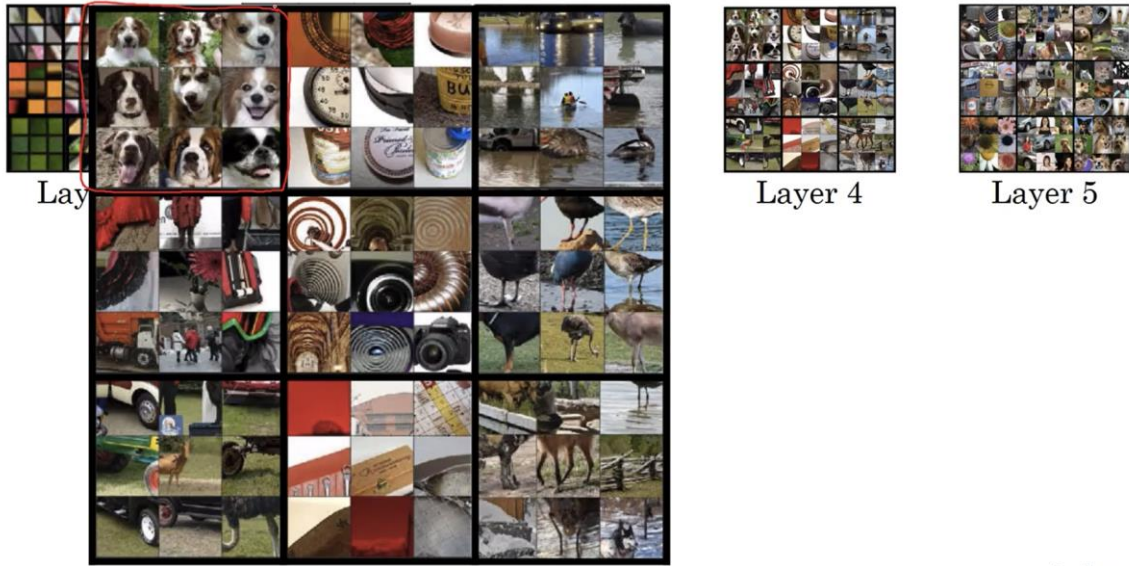
Visualizing deep layers: Layer 3



Andrew Ng

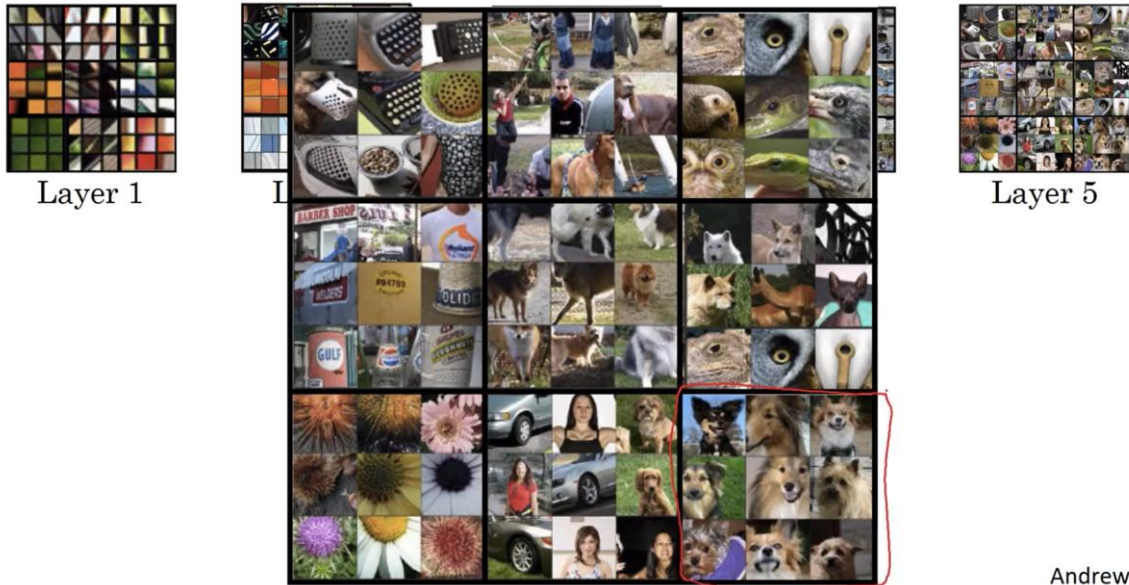
Aynı şekilde layer 4 unitleri de daha complex feature'ları detect ediyor, burada tekrar intuitive bir açıklama yapayım, nasıl oluyorda layer 4 unitleri aşağıda görüldüğü gibi köpek yüzünü doğrudan detect edebiliyor? Şöyle oluyor, aslında layer 4 unitleri layer 1 unitleri ile aynı işlemi yapıyor, gidiyor 3x3'lük filtre uyguluyor şu o filtrenin aradığı feature'lara bakıyor, ancak 1. Layer'da aranan feature'lar edges, 2. Layerda aranan feature'lar edges'in bir fonksiyonu oluyor, yani şekiller aranmaya başlıyor, 3. Layerda ise aranan şey bu şekillerin bir fonksiyonu, 4. Layerda artık o kadar complex bir fonksiyon ortaya çıkıyor ki 3x3'lük bir alanda bir feature detect edildiğinde onun karşılığı input image'da neye denk geliyor diye bir bakıyorsun köpek yüzüne denk geliyor, yani her layer giderek daha complex şeyler öğreniyor aynı fully connected layerda olduğu gibi.

Visualizing deep layers: Layer 4



Andrew Ng

Visualizing deep layers: Layer 5



Andrew Ng

5. layer unitleri de beklendiği gibi son derece complex feature'lar söz konusu.

Vid 8

Cost Function



Bir Neural Style Transfer sistemi kurmak istiyoruz bu amaçla önce generated image için bir cost function tanımlayalım.

Bu cost function'ı minimize ettiğimiz zaman, istediğimiz tarzda image'lar üretebiliyoruz demektir.


Neural style transfer'i uygulamak için yapacağımız şey bir $J(G)$ cost function'ı tanımlamak, sonuçta bunun ölçmesi gereken Generated image G 'nin ne kadar iyi olduğu. Yani $J(G)$ 'yi minimize ettiğimizde generated image'ımız tam istediğimiz gibi olacak. Content C 'nin style S ile regenerated hali olacak.

Böyle bir cost function'ı iki farklı componentten oluşacak, ilk kısmı, C ile G 'nin ne kadar birbirine benzediğini ölçerken, ikinci kısım ise S ile G 'nin ne kadar birbirine benzediğini ölçüyor. Ayrıca alfa ve beta hyperparametreleri ile bu costlara weight atıyacağız, nihayetinde bu hyperparameters, generated image'ın C 'ye mi S 'yemi daha fazla benzeyeceğini belirlemiş oluyor.

Neural style transfer cost function



$$J(G) = \alpha J_{\text{content}}(C, G) + \beta J_{\text{style}}(S, G)$$



Generated image G



[Gatys et al., 2015. A neural algorithm of artistic style. Images on slide generated by Justin Johnson] Andrew Ng


Sonuçta algoritma şöyle çalışacak:

- Cost function J yukarıdaki gibi tanımlandı diyelim.
- Önce G image'ını random olarak initialize edelim.
- Daha sonra GD gibi bir optimization algorithm ile $J(G)$ 'yi minimize etmeye çalışırız.
- En nihayetinde $J(G)$ 'nin minimize edilmiş olması, tam da istediğimiz gibi bir G 'nin üretilmiş olmasına denk gelir.

Find the generated image G

1. Initiate G randomly
 $G: 100 \times 100 \times 3$
 \uparrow
RGB
2. Use gradient descent to minimize $J(G)$
$$G := G - \frac{d}{dG} J(G)$$





[Gatys et al., 2015. A neural algorithm of artistic style]

Andrew Ng

Randomly initialized G yukarıdaki gibi bir White noise picture'a denk gelir daha sonra $J(G)$ minimize edildikçe giderek yukarıda görüldüğü gibi istediğimiz image ortaya çıkmaya başlar.

Şimdiye kadar herhangi bir ConvNet kullanılmadığına dikkat et, ayrıca burada weights'in direkt G 'yi temsil eden pixeller olduğuna da dikkat et. Ama $J(G)$ 'yi hesaplamak için kullanılacak, $J_{content}$ ve J_{style} henüz tanımlanmadı. Muhtemelen bunları tanımlamak için bir ConvNet feature extraction yapması için kullanılacak.

İlerleyen kısımlarda $J_{content}$ ve J_{style} 'in nasıl tanımlandığını anlayalım.

Vid 9

Content Cost Function

Yukarıda gördüğümüz üzere Neural Style Transfer algorithm'ın cost function'ı iki farklı componentten oluşuyordu: $J_{content}(C,G)$ ve $J_{style}(S,G)$. Bunların nasıl tanımlandığını görmemiştik.

Bu kısımda content cost compoenent'e odaklanacağız. Overall cost function $J(G)$ aşağıda görüldüğü gibi tanımlanmıştı.

- Diyelim ki content cost'u hesaplamak için hidden layer l 'i kullanıyoruz.
- Eğer l çok küçükse mesela $l=1$ ise bu durumda, generated image'ımız content image'a çok benzeyecektir.
- Eğer daha derin layerlardan birini kullanırsak, generated image content image'dan etkilenecektir ama birebir aynı olmayacaktır.
- Sonuçta layer l 'in seçimi ne çok sığ olmalı ne de çok derin. Programming excercise da bu hyperparameter ile oynayarak değişimi kendimiz görebileceğiz.

Content cost function

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

- Say you use hidden layer l to compute content cost.
- Use pre-trained ConvNet. (E.g., VGG network)
- Let $a^{[l](C)}$ and $a^{[l](G)}$ be the activation of layer l on the images
- If $a^{[l](C)}$ and $a^{[l](G)}$ are similar, both images have similar content

$$J_{content}(C, G) = \frac{1}{2} \|a^{[l](C)} - a^{[l](G)}\|^2$$

[Gatys et al., 2015. A neural algorithm of artistic style]

Andrew Ng

- Tahmin ettiğimiz gibi $J_{content}$ 'i tanımlamak için bir pretrained ConvNet kullanılıyor.
- Seçilen bir l layer'ı için, Content image ve Generated image bu network'e veriliyor, network bir nevi feature extraction yapıyor ve bu iki aktivasyonun farkı bize aslında bu iki image'ın ne kadar farklı olduğunun bir ölçüsünü veriyor.
- Sonda görülen $J_{content}(C,G)$ tanımı, C ve G image'ının ne kadar farklı olduğunu veriyor.
- Son görülen tanım: element-wise sum of squarees of differences between the activations in layer l between the images C and G.

Eğer bu fark ilk layerda hesaplanırsa ve bu cost minimize edilirse, ne olacak C ile G image'ı neredeyse birebir aynı olacak, çünkü beklentimiz tüm low level feature'ların tam olarak aynı konumlarda görülmesi, bunun tek yolu aynı image'ı üretmek.

Ancak ilerleyen layerlardan biri için bu fark hesaplanırsa, iki image tam olarak aynı olmak zorunda değil, complex feature'ları aynı olmak zorunda, yani C de biryerlerde bir köpek varsa G'de de olacak ama illaha tam aynı noktada olmak zorunda değil gibi düşün.

En nihayetinde overall cost function GDA gibi bir algoritma ile minimize edilirken Jcontent de minimize edilecek ve bunun da etkisi, I. Layerda content image ile aynı complex feature'ları taşıyan bir Generated image üretmek olacak.

Vid 10

Style Cost Function

Bu kısımda odaklanacağımız şey style cost function. Geçtiğimiz başlıkta gördük ki content cost function dediğimiz şey şu: content image convnet'e verildiğinde l. Layer'da oluşan aktivasyon ile generated image convnet'e verildiğinde l. Layer'da oluşan aktivasyonun farkı.

Bir başka deyişle eğer iki image benzer content feature'lar içeriyorsa, content cost function küçük çıkacaktır.

Böylece content cost fonksiyonu minimize ederek, generated image'in content image'in sahip olduğu complex feature'lara sahip olmasını sağlarız.

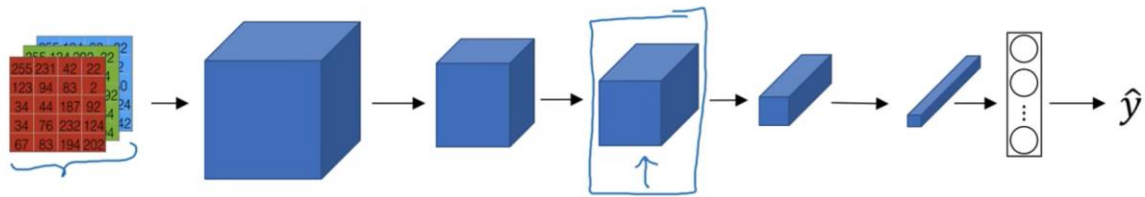
Benzer şekilde, generated image'in style image'in stiline sahip olmasını sağlayacak şey de style cost function'ın minimize edilmesi.

İki image'in content olarak benzemesini bir convnet'e verildiklerinde l. Layer'da elde edilen aktivasyonların benzer olmasından anlayabiliyoruz, peki ama iki image'in style olarak benzemesini nasıl anlarız nasıl tanımlarız?

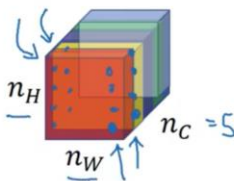
İşte bu tanımlı yapabilirsek o zaman style cost function'ı tanımlayabiliriz.

Toparlıyorum: Bir resmin content'i dediğimizde şunu anlayabiliriz, image bir ConvNet'e verildiğinde l. layerda oluşan aktivasyonlar yani detect edilen complex featurelar. Peki bir resmin style'ı dediğimizde ne anlayacağız?

Meaning of the “style” of an image



Say you are using layer l 's activation to measure “style.”
Define style as correlation between activations across channels.



How correlated are the activations
across different channels?

Diyelim ki image'ımızı yukarıdaki gibi bir ConvNet'e verdik ve style'ı define etmek için yukarıda işaretlenen I. layer'ın aktivasyonunu kullanacağım.

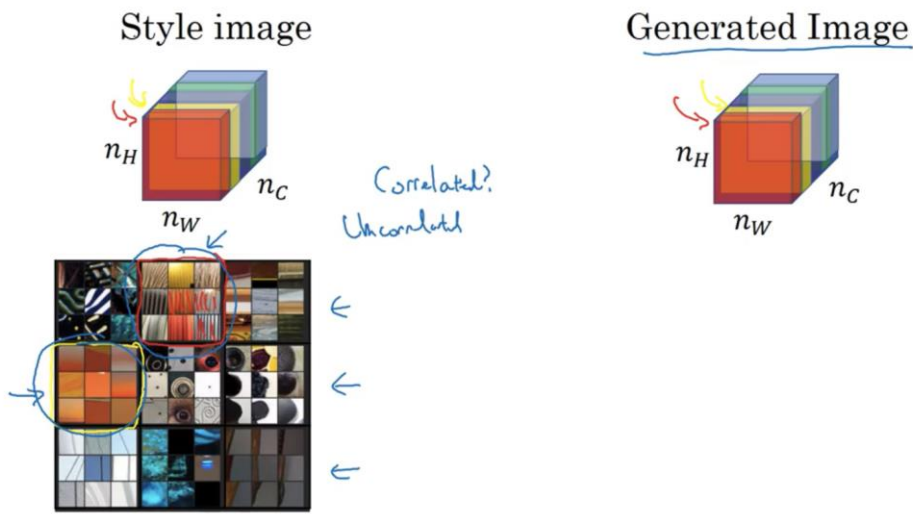
→Image'ın stilini, kanallar arası aktivasyonların korelasyonu olarak tanımlayabiliriz!!!

- Bunu açalım:
 - İlgili layer'ı yukarıda renkler ile gösteriyoruz.
 - Sorumuz şu: farklı kanalların aktivasyonu birbirili ile ne kadar ilintili.
 - Bunu anlamak için yaptığımız şey şu: Mesela kırmızı kanalın ilk elemanı ile sarı kanalın ilk elemanı bir pair oluşturuyor, benzer şekilde karşılıklı pairlar yukarıda noktalar ile gösterilmiş, işte bu pairların birbiri ile ne kadar correlated olduğunu bulursak, kırmızı ve sarı kanalın birbiri ile ne kadar correlated olduğunu bulmuş oluruz.
 - Kırmızı kanalın temsil ettiği şey şu: input üzerinde kırmızı filtrenin uygulanması ile elde edilen map yani input üzerinde hangi noktalarda kırmızı filtrenin temsil ettiği feature bulundu onu gösteriyor.
 - Benzer şekilde sarı kanal da input üzerinde sarı filtrenin temsil ettiği feature nerelerde detect edildi onu gösteriyor.
 - Eğer iki pair correlated ise bu şu demek, input'un ilgili region'ında kırmızı feature ile sarı feature birarada bulunuyor bu yüzden biri detect edilince diğeri de detect ediliyor, biri detect edilemezse diğeri de detect edilemiyor.

İyi de nasıl oluyor da bu kanalların korelasyonu image'ın style'ını temsil ediyor? Bir örneğe bakalım:

- Diyelim ki kırmızı channel'ın ilk unit'i şekilde görüldüğü gibi dikey line'ları detect eden bir unit iken sarı channel'ın ilk unit'i ise orange color detect eden bir unit.
- Eğer bu iki unit correlated ise bu şu demek, input image'ın ilgili unit'in gözlemlediği region'ında ne zaman çizgiler belirse burada orange color da oluyor.
- Benzer şekilde eğer uncorrelated ise, input image da bir feature yer alırken diğer yer almıyor demek.

Intuition about style of an image



[Gatys et al., 2015. A neural algorithm of artistic style]

Andrew Ng

Yani channellar arası korelasyonun bize: hangi l. layer feature'larının ilgili image'da birarada bulunduğunu ve hangilerinin birarada bulunmadığını söyler.

Image style dediğimiz şeyi: hangi high level feature'ların birarada yer alıp hangilerinin birarada yer almadığı olarak tanımlıyoruz.

Channellar arası korelasyonun derecesini, measure of style olarak kullanırsak bu bize şunu sağlayacak, style image'ın style'ı ile generated image'ın style'ını birbirine yaklaştırmak. Yani eğer style image için vertical lines ile orange texture hep birarada kullanılıyorsa, biz aynı işlemin generated image için de yapılmasını isteriz. Yani style image'ın measure of style'ı ile generated image'ın measure of style'ı birbirine yakın olsun isteriz ki iki image'ın stilleri benzer olsun.

Yani content cost function'ı minimize ederek yapmak istediğimiz şey: öyle bir generated image yarat ki ConvNet'in l. layer'ında content image ile benzer complex feature'lar detect edilsin yani birisinde kapı varsa diğerinde de kapı olsun vesaire.

Style cost function'ı minimize ederek yapmak istediğimiz şey ise: öyle bir generated image yarat ki ConvNet'in l. layer'ında style image için birarada bulunan feature'lar generated image için de birarada bulunsun, yani eğer style image da bir region da sarı çizgiler ile yapraklar hep biradada bulunuyorsa, bu stil generated image da da olsun.

Mantığı anladık, şimdi bu mantığı formalize edelim yani bir image'ın style'ını nasıl temsil edeceğimizi görelim:

- Hem style image için hem de generated image için bir style matrix tanımlayacağız.
- Aşağıda görüldüğü gibi ConvNet'in i, j, k koordinatlarındaki unitin aktivasyonu belirtildiği gibi temsil edilsin.
- i ve j sırasıyla height ve width'i temsil ederken k ise channel'ı temsil ediyor.
- Style image S ve ConvNet'in l . layer'ı için bir style matrix G tanımlayacağız. Bu matrisin boyutu doğrudan l . layer'daki channel sayısı ile belirlenecek $n_c \times n_c$ boyutunda olacak.
- l . layer için aynı style matrix G 'yi hem style image hem de Generated image için aşağıdaki gibi tanımlayabiliriz:
 - $G_{kk'}$ bir value return eder k . Ve k' . Layerların korelasyonunu temsil ediyor.
 - Bu şekilde eğer l . layerda 3 channel varsa: $G_{12} G_{13} G_{23} G_{21} G_{32} G_{31}$ hesaplanacak ve G matrisi böyle hesaplanacak ki zaten bunların ilk yarısı diğer yarısıyla aynı olacak.

Style matrix

Let $a_{i,j,k}^{[l]}$ = activation at (i, j, k) . $G^{[l]}$ is $n_c^{[l]} \times n_c^{[l]}$

$$\rightarrow G_{kk'}^{[l](S)} = \sum_{i=1}^{n_H} \sum_{j=1}^{n_W} a_{ijk}^{[l](S)} a_{ijk'}^{[l](S)}$$

$$\rightarrow G_{kk'}^{[l](G)} = \sum_{i=1}^{n_H} \sum_{j=1}^{n_W} a_{ijk}^{[l](G)} a_{ijk}^{[l](G)}$$

"Gram matrix"

$$J_{style}^{[l]}(S, G) = \frac{1}{(n_H n_W n_c)^2} \sum_k \sum_{k'} (G_{kk'}^{[l](S)} - G_{kk'}^{[l](G)})^2$$

[Gatys et al., 2015. A neural algorithm of artistic style]

Andrew Ng

Yukarıda l . layer daki k ve k' . channel'ların korelasyonunu hesaplamak için yapılan işlemin basitçe bu layerların aktivasyonlarını elementwise çarpmak ve hepsini toplamaktan fazlası olmadığını görebiliriz.

Bu işlemi hem style hem de generated image'ın l . layer'ındaki her bir channel için yapabiliriz ve böylece l . layer için style matrixler hesaplanabilir.

Tüm bunlar anlaşıldıysa l . layer için style cost function da yukarıda görüldüğü gibi bu iki style matrixin farklarının toplamı şeklinde hesaplanabilir. Bu fonksiyon minimize edilince iki image'ın l . layerdaki stillerinin benzerliği sağlanmış olur.

Daha da iyi bir sonuç için style cost function birden fazla layer için hesaplanabilir. Aşağıda görüldüğü gibi overall style cost function hesaplanırken birden fazla layer için style cost'u hesaplıyoruz ve weighting uyguluyoruz.

Style cost function

$$\|G^{[L](S)} - G^{[L](G)}\|_F^2$$

$$J_{style}^{[l]}(S, G) = \frac{1}{(2n_H^{[l]}n_W^{[l]}n_C^{[l]})^2} \sum_k \sum_{k'} (G_{kk'}^{[l](S)} - G_{kk'}^{[l](G)})^2$$

$$J_{style}(S, G) = \sum_l \lambda_l J_{style}^{[l]}(S, G)$$

$$\underbrace{J(G)}_G = \alpha J_{content}(G) + \beta J_{style}(S, G)$$

[Gatys et al., 2015. A neural algorithm of artistic style]

Andrew Ng

En nihayetinde overall neural style transfer cost function ise yukarıdaki gibi tanımlanabilir. Böylece bu cost function minimize edildiğinde, generate imageımız content image'ın l. layerda içerdiği feature'ları içerecek yani content'i benzer olacak, bunun yanında da generated image'ın stili yani hangi feature'ların birarada kullanıldığı bilgisi de style image ile benzerlik gösterecek. Stil için sadece l. layer değil birden fazla layer hesaba katılıyor.

Burada l. layer darken kastettiğimiz şey, eğitilmiş bir convnet'in l. layer'ı. Bu ConvNet'e content ve generated image veriliyor ve content cost hesaplanıyor, benzer şekilde style ve generated image veriliyor ve style cost hesaplanıyor, bu cost'u hesapladıktan sonra da generated image'ın weightleriz update edilerek, her adımda isteğimize daha yakın bir image yaratılıyor.

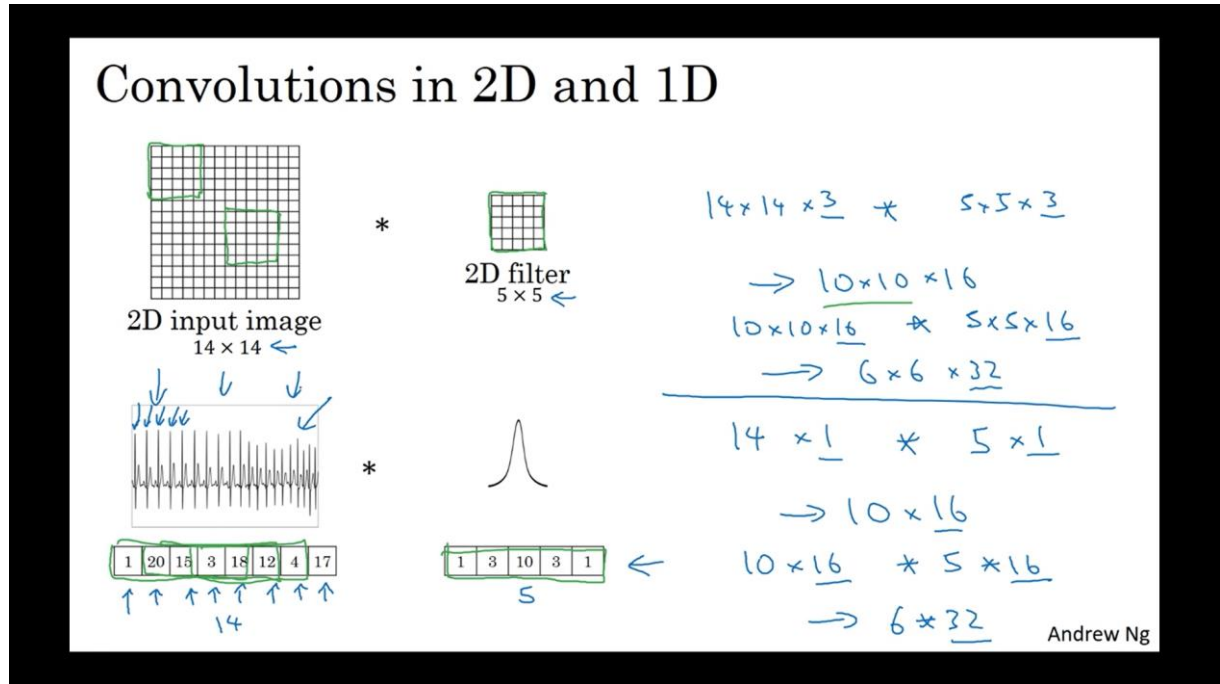
Vid 11

1D and 3D Generalizations

Şimdiye kadar, ConvNet'i iyice anladık ancak hep 2D inputlar için kullandık.

ConvNet fikri 1D veya 3D verilere de genellenebilir. Bu kısım tamamen bu intuitor'u sağlamayı hedefliyor.

İlk haftalarda 2D convolution'ı görmüştük, 14x14'lük 2D bir input image'a 5x5'lik 16 tane filtre uygularsak sonucun 10x10x16 olacağını biliyoruz.



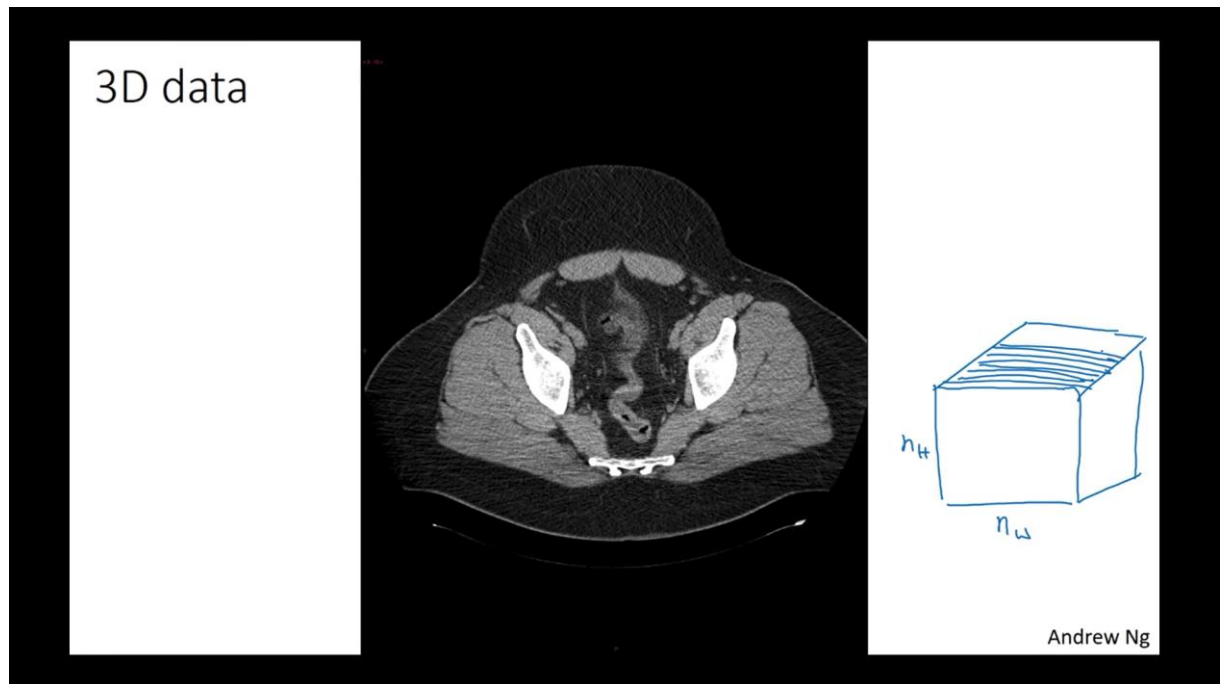
Benzer şekilde yukarıdaki 1D EKG datasına convolution uygulayarak, medical diagnosis için kullanabiliriz. Görüldüğü gibi EKG verisi 14x14 yerine yalnızca 14 boyutunda yani, time series data.

Tam da bu sebeple convolution uygulanacak filtre de 1D olacak, yani 5x5 yerine yalnızca 5 boyutunda bir filtre kullanabiliriz, convolution işlemi birebir aynı olacak, filtre input üzerinde gezecek ve convolution uygulayacak.

Sonuçta da 10 filtre uygulanırsa 10x16'lık bir output elde ederiz.

Aslında 1D datanın birçoğu için recurrent neural networks kullanılır ama bazıları convolutional neural network de kullanmayı deneyebilir. Gelecek kursta Sequence models'i göreceğiz ve recurrent neural networks ile LSTM'den bahsedip kıyaslama yapacağız.

Benzer şekilde convolution işlemi 3D data ile de kullanılabilir, mesela aşağıda görülen fotoğraflar, beynin kesitlerini gösteriyor. Yani her fotoğraf 3 boyutlu bir fotoğrafın dilimlenmiş hali gibi. En nihayetinde elimizde 3 boyutlu bir image var diyebiliriz.

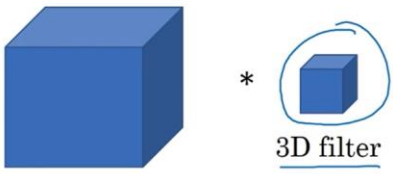


Eğer bu 3D scan üzerinde feature detect etmek için convnet kullanacaksak, 3D convolution kullanılabilir.

Diyelim ki elimizde 14x14x14x1 boyutunda gray scale 3 boyutlu bir image var. Buna gidip 5x5x5x1'lik 16 filtre uygulayabiliriz.

Sonucunda 10x10x10x16'lık bir çıkış elde ederiz. Bu sadece tek bir layer, istersek daha fazla layer ekleyebiliriz.

3D convolution



The diagram shows a large blue cube labeled "3D volume" and a smaller blue cube labeled "3D filter" with a circular arrow around it. They are separated by a multiplication symbol (*).

Handwritten calculations:

$$\begin{aligned} & \downarrow \downarrow \downarrow \downarrow n_c \\ & \underline{14 \times 14 \times 14 \times 1} \\ & * \underline{5 \times 5 \times 5 \times 1} \quad 16 \text{ filtre.} \\ & \rightarrow 10 \times 10 \times 10 \times \underline{16} \\ & * \underline{5 \times 5 \times 5 \times 16} \quad 32 \text{ filtre.} \\ & \rightarrow 6 \times 6 \times 6 \times 32 \end{aligned}$$

Andrew Ng

Bir başka 3D data örneği de movie data'dır. Böylece motion detection vb. gibi şeyler yapılabilir.