ConvNet architectures

Magnus Jensen

ConvNet architectures

Lektion 7 og 10

Motivation

Træning af ConvNets

CNN er en kategori af neurale netværk der har vist sig at være effektive til billede genkendelse og klassifisering; her i blandt til ansigter, objekter, skilte og så videre, til f.eks brug i robotter og selv kørende biler.

Denne process udkommer fra at et convnet kan splittes beskrives som to dele, en encoder og en decoder.

Encoderen har til mening at udtrække features i et billede, alt generalisere hvad et billede består af ned til simple dele. Det er så decoderens job ud fra disse informationer at forstå hvad billedet repræsentere afgrænset af de mulige kategorier.

Layers

For at dette kan virke, består netværket af nogle layers som besidder nogle filtre og vægte, altså matricer af tal. Det er disse som træningen handler om, og har mans succesfuldt trænet et netværk, vil man kunne se at de forskellige layers lærer forskellige ting.

Til at starte med, vil lagene begynde at lære om kanter og streger. Går vi lidt dybere, ser vi at disse vil blive sammensat til simple former og tekstur. Længere nede ser vi da mere komplekse former, de kunne være ansigt, hjul, eller døre. Aller sidst i decoderen, i output laget, vil vi da se repræsentationer af hvad klasse, nærmest som en skabelon eller template. Og ud fra disse, vil de forskellige units så slå ud hvis deres input passer godt derpå.

Transfer learning

Men at træne sådan et netværk, til at kunne beside så komplekse vægte; kan

> **Træning af convnet**
>
> 1. Tage lang tid
> 2. Kræve et kæmpe dataset

Hvorfor det kan være godt at udnytte transfer-learning. Hvis man opbygger sit netværk omkring transfer learning, betyder det at man bruger en pre-trænet encoder, og så bruger sin egen decoder som man så træner.

Bruger man ImageNet som dataset, vil det normalt tage flere uger at træne sit netværk, men ved at bruge Transfer Learning kan det gøres omkring en times tid.

> Imagenet: 1 million billeder, 100 kategorier

ImageNet består af over 1 million billeder fordelt på 1000 kategorier, og bruger man en CNN der er trænet på ImageNet er der f.eks nogle forskellige måder at håndter det på.

Et Convolutional Netværk har jo typisk i sin decoder, flere fully connected lag, så lad os stille to forskellige scenario op:

Hvis man har et lille dataset selv, så erstat kun det sidste FC lag, da man ellers vil kunne komme til at overfitte.

> Lille dataset = sidste FC

Hvis man har et stort dataset selv, så erstat flere af de fully connected layers.

> Stort dataset = flere FC

Har men gjort det, så fryser man vægtene på alle de lag man ikke har erstattet og begynder at træne med sit eget dataset. Man kan derefter *fine-tune* sit netværk ved at unfreeze lagene

igen og træne lidt på hele netværket.

AlexNet

AlexNet var en revolution inden for machine learning, det var det første til at bruge ReLU, og er populært brugt som base for Transfer Learning grundet dets ligetil struktur. AlexNet er som så et simpelt netværk, der tager input af 277x277x3.

Det består af otte lag. De første fem er convolutionelle lag, nogle efterfulgt af maxpool. Og til sidste er der tre fully connected lag.

> - Conv > maxpool > norm
> - Conv > maxpool > norm
> - Conv > Conv > Conv
> - maxpool
> - FC > FC > FC

VGGNet

Men AlexNet bruger nogle store filter størrelser på sine convolution lag, helt op til 11x11.

Med VGGNet var målet at få langt færre parameter, som de opnår ved at bruge flere convolutions i træk med en filter størrelse på 3x3.

Og vi kan tage et eksempel på at det virker.

Hvis vi har et input billede på 5x5, og Alexnet bruger et filter på samme størrelse, med stride 1; så er det tydeligt at vi har et output på størrelsen 1; men for god ordens skyld kan vi bruge formlen:

> $outputsize = \frac{inputsize - filtersize}{stride} + 1$

Derved har vi for AlexNet:

> $\frac{5-5}{1} + 1 = 1$

Og med VGG og dobbelt conv får vi:

> - Conv1: $\frac{5-3}{1}+1=3$
> - Conv2: $\frac{3-3}{1}+1=1$

Men så kan vi så spørge os selv - og hvad er pointen så?
Jo med et filter på 5x5 er der 25 parameter, men med to filter af 3x3, er det 18 i alt; så vi når frem til det samme, men med et mindre antal af parametre.

GoogleNet

GoogleNet er et meget anderledes netværk, det har i midten nogle 1x1 convolutions, som den bruger til noget den kalder "inception", og så bruger GAP til sidst i stedet for FC.

1v1 conv

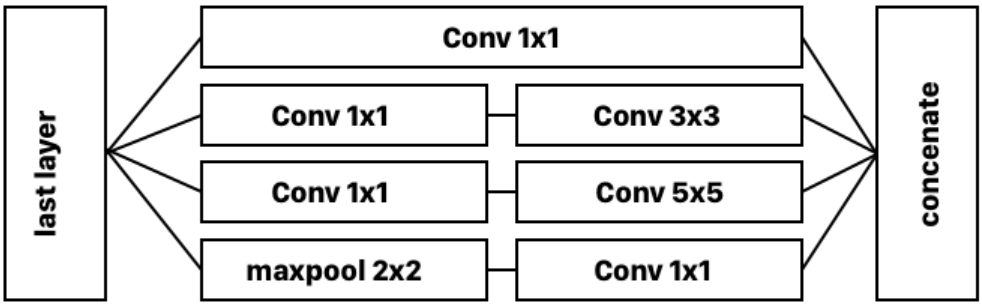
Ved at bruge 1x1 conv, reducere GoogleNet sin depth, der kan bruges til at spare mange udregninger.

Har man f.eks en shape af $14 \cdot 14 \cdot 480$ og ønsker en 5x5 conv til $14 \cdot 14 \cdot 48$, vil det tage over 100 millioner udregninger.

Starter man dog ud med en conv 1x1 til $14 \cdot 14 \cdot 16$ for så at conv 5x5 til $14 \cdot 14 \cdot 48$, tager det kun omkring 5 millioner udregninger.

Inception

Typisk bruger man conv layers af 5x5 eller 3x3 eller også bruger man et maxpool layer, og hver er gode at bruge og kan være svære at vælge imellem; så hvorfor ikke bare bruge dem alle sammen?



Inception modulet, som sidder i GoogleNet gør netop dette. Fra et lag, bliver der outputtet både til 1x1, 3x3, 5x5 og et maxpool, der så alle bliver concanated sammen. For at spare på udregningerne, bliver 3x3 og 5x5 conv med 1x1 først, og maxpool efter.

GAP

Skulle man bruge FC layers til decoder delen, ville der skulle bruges mange vægte. Ved at bruge Global Avarage Pooling, laves hver feature map om til en 1x1 hvilket kræver nul vægte og skulle gøre accuracyen 0.6 bedre.

Auxiliary Classifiers

GoogleNet besidder en sidste interresandt ting; i løbet af netværket er der nogle classifier outsputs.

Disse bruges under træning kun, og deres loss ligges til den samlede loss med en vægt på 0.3, der skulle modarbejde vanishing gradients.

Layer Activations

Så det var tre forskellige netværker, og det kan godt være lidt svært sådan at forstå hvordan og hvorfor de virker; da man har meget lille ansigt i træningen og de ting der sker.

Derfor kan det være meget smart at man kan visualisere de forskellige layers aktivationer og filtre. Det tillader os blandt andet at:

> - Observer stimuli for feature maps
> - Observer feature maps udvikling under træning
> - Lave bedre designs!

For at visualisere aktivationer, er den nemmeste måde er at visualisere de forskellige kanaler ved et forward pass. Når vi kommer længere ned i netværket, vil nogle kanaler virke tomme, grundet at det pattern de repræsentere ikke var tilstede i billedet.

~~I filtre vil vi gerne se at de er smooth og ikke fulde af noise~

Vil man visualizere hvordan et fully connected lag klarer sig, kan man træne med en masse billeder og gemme feature vektorerne fra det lag. Man kan så kører KNN på disse, og hvis billeder der repræsentere det samme (features) er det godt, men hvis det bare er billeder der ligner hinanden (f.eks to billeder af to slags dyr på en hvid baggrund) er det dårligt. Altså vil vi ikke have pixels der er ens.

Rekonstruktion

Hvad nu hvis vi *vendte* netværket om? Så det rent faktisk lavede billeder for os i stedet for?

Ideen er, at vi stadig giver netværket et billede, men dette billede bruger netværket til at beskrive hvilken content den skal lave et nyt billede af.

Vi vil nu ved at give et *content* billede x til netværket, finde det billede y der gør at outputtet ved laget l nærmest er det samme. Det betyder, at ud fra et content billede x vil vi genere et nyt billede, hvor at de ved laget l matcher hinandens feature maps.

Det virker ved at vi optimere på billedet i stedet for vægte. Så hvis vi initialisere vores billede y med random noise, og vi smider x igennem netværket, så vil optimeringen ske på y i stedet for de forskellige vægte.

Des dybere laget for outputtet er, des mere *abstract* ser resultatet ud.

Max filter visualizering

Vi kan også bede netværket om at outputtet et billede, som et lag vil slå maksimalt ud på.

Ved at sende et random noise billede ind i netværket, for det lag vi ønsker at visualiser filtre for, tager vi så for hver af dets filtre en mean, det bruger vi så til, igen, at optimere på vores input noise billede, ved at bruge *gradient Ascent*, hvilket gør at at det resulterende billede vil få det givne lag til at slå laks ud.

Klasse baseret

Vi kan også bede vores CNN, om at lave et billede, som den vil slå ud på for en given kategori.

Vi starter igen med et billede bestående af random noise, og beder nu netværket om stille at tweake billedet til hvad den mener er en banan.

Det kan hjælpe os med at forstå hvad et netværk tænker om en klasse, sæt nu den tror at en banen altid skal være i hånden på en abe? Så burde vi nu tjekke vores dataset, om der er nogle billede af isolerede bananer, eller sådan.