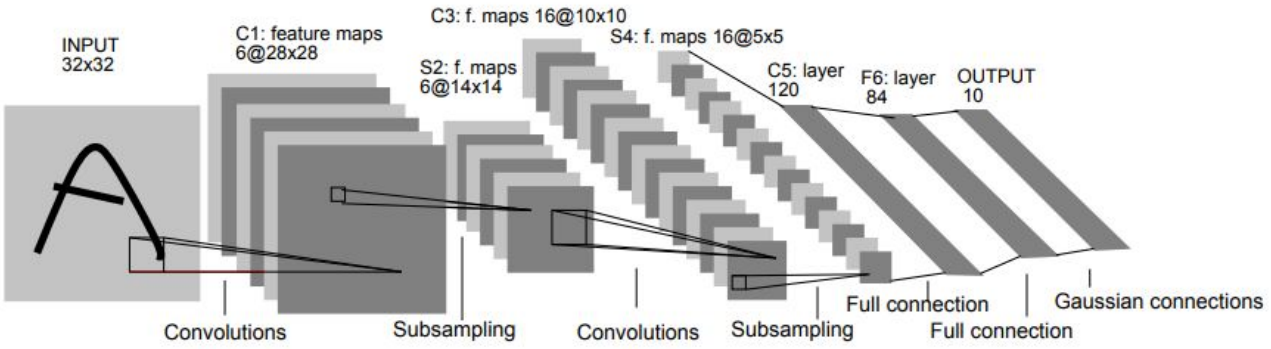
Հայտնի փաթույթային ցանցեր

Հայկ Կարապետյան

Հայտնի փաթույթային ցանցերը դիտարկելիս, փորձելու ենք հասկանալ գաղափարը, որով կառուցվել են ցանցերը: Շերտերի քանակը, շերտերում նեյրոնների քանակը և ակտիվացիոն ֆունկցիաները հիպերպարամետրեր են, որոնք որոշվել են validation տվյալների վրա:

1 LeNet (1998)

Առաջին փաթույթային ցանցը, որը ցուցաբերել է լավ արդյունք ուսուցանվել է 1998թ.-ին և դա եղել է LeNet ցանցը (Նկար 1): LeNet-ի հեղինակը Yann LeCun-ն է:



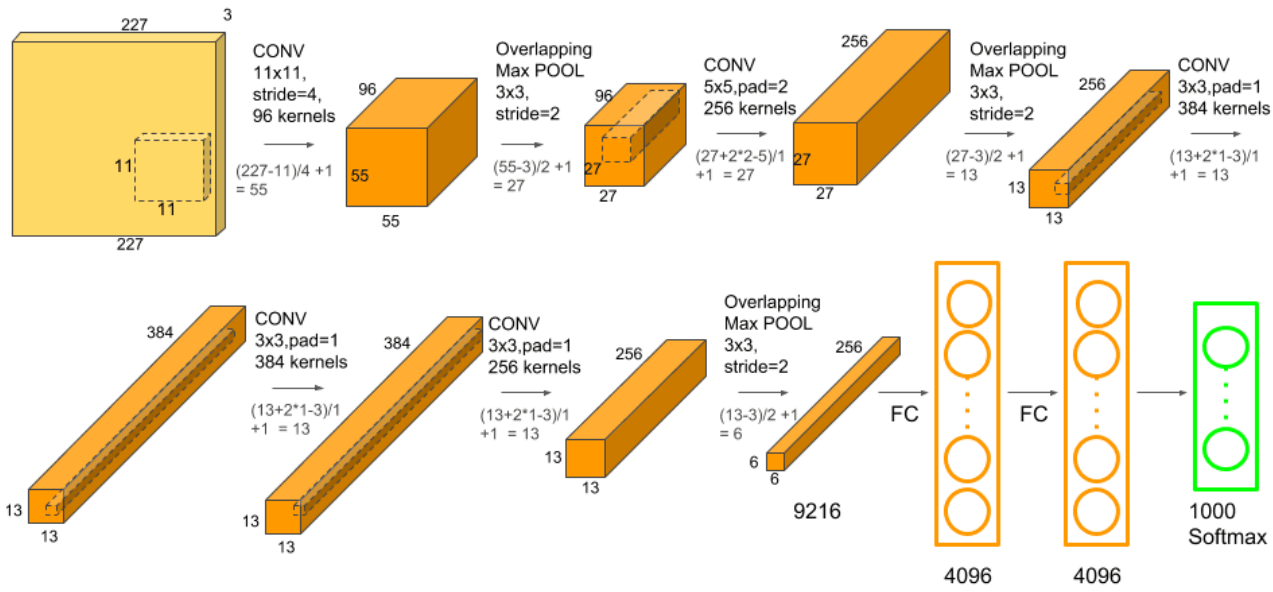
Նկար 1: LeNet փաթույթային ցանցի կառուցվածքը (architecture)

Այն ուսուցանվել է Mnist[1] dataset-ի վրա: Հիմա փորձենք հասկանալ կառուցման քայլերը: Սկզբից վերցնում ենք 32×32 չափանի մոխրագույն նկարը և կատարում $5\times5\times6$ չափանի միջուկով convolution (բոլոր շերտերում convolution-ի stride=1 և padding=0): Միջուկը երեք չափանի է, քանի որ սկզբնական նկարը մոխրագույն (grayscale) է: Արդյունքում ստանում ենք $28\times28\times6$ չափանի feature map: Դրանից հետո կիրառում ենք max pooling: Feature map-ի չափը երկու անգամ փոքրանում է և դառնում $14\times14\times6$: Հետո կիրառում ենք $5\times5\times6\times16$ չափանի միջուկով convolution: Այս դեպքում feature map-ը ուներ 6 խորություն, այդ պատճառով միջուկը 4 չափանի է: Արդյունքում ստանում ենք $10\times10\times16$ չափանի feature map: Հետո նորից կիրառում ենք max pooling: Ստանում ենք $5\times5\times16$ չափի feature map: Վերջին շերտը ունի 10 նեյրոն, քանի որ դասակարգում ենք 0-ից 9 թվանշանները: Այդ շերտը սովորական dense շերտ չէ, այլ gaussian connections է, որին ավելի մանրամասն կարող եք ծանոթանալ հետևյալ կայքում: Ցանցում ակտիվացիոն ֆունկցիաները կա՛մ sigmoid են, կա՛մ tanh, քանի որ 1998 թվականին relu-ն չէին օգտագործում որպես ակտիվացիոն ֆունկցիա: Գաղափարը, որը կարող ենք հաս-կանալ այս ցանցից հետևյալն է. ամեն քայլի մենք նկարի վրա կիրառում ենք փաթույթ՝ փոքրաց-նելով նկարի չափերը և շատացնելով խորությունը, ապա max pooling: Convolution շերտերից հետո ունենք երկու dense շերտ, որոնք չափերը գնալով փոքրանում են: Մեկ շերտի դեպքում արդյունքը ավելի վատ է եղել քան երկուսի դեպքում: Այս ցանցը ունի մոտավորապես 60000 պարամետր, որը այժմյան ցանցերի հետ համեմատած շատ քիչ է, բայց հաշվի առնելով 1998թ.-ին համակարգիչների հզորությունը, այն բավականին լավն էր:

1. Mnist - 32×32 չափանի ձեռագիր թվանշանների հավաքածու՝ մոտավորապես 70000 նկար:

2 AlexNet (2012)

LeNet ցանցից հետո լավ արդյունք ցուցաբերած ցանցը AlexNet-ն է (Նկար 2)։ AlexNet-ի հեղինակը Alex Krizhevsky-ն է։



Նկար 2: AlexNet փաթույթային ցանցի կառուցվածքը

Այս ցանցը ուսուցանվել է ImageNet[1] տվյալների հավաքածուի վրա։ Կարելի է ասել, որ սա LeNet-ի ավելի մեծ տարբերակն է։ Մուտքային տվյալները $227\times227\times3$ չափի նկարներ են։ Սկզբից կիրառում ենք $11\times11\times3\times96$ չափի միջուկ։ Միջուկի stride=4։ Այսքան մեծ միջուկը հնարավորություն է տալիս ավելի մեծ տեսողությամբ նայել նկարին և ավելի հեշտ գտնել մեզ հետաքրքրող օբյեկտները (շատ պոչ, շնաձկան ատամներ)։ Convolution-ը կիրառելուց հետո կստանանք $55\times55\times96$ չափի feature map։ Դրանից հետո կիրառում ենք max pooling։ Ստացված $27\times27\times96$ չափի feature map-ի վրա կիրառում ենք $5\times5\times96\times256$ չափի միջուկով, stride=2, padding=2 convolution։ Հետո նորից max pooling։ Ստացված $13\times13\times256$ feature map-ի վրա կիրառում ենք $3\times3\times256\times384$ չափի միջուկով, stride=1, padding=1 convolution։ Ստացվածի վրա նորից կիրառում ենք նույն convolution-ը։ Հետո կիրառում ենք $13\times13\times384\times256$ չափի միջուկով, stride=1, padding=1 convolution։ Հետո կիրառում ենք max pooling։ Ստացված feature map-ը հարթեցնում ենք և անցկացնում 2 dense շերտերի միջով, ամեն մեկը ունի 4096 նեյրոն։ Վերջնական ստացված 4096 արժեքը անցկացնում ենք 1000 նեյրոնից բաղկացած շերտի միջով, որի ակտիվացիոն ֆունկցիան softmax է։
Գաղափարը, որը կարող ենք հասկանալ այս ցանցից հետևյալն է. սկզբից նկարի վրա կիրառում ենք մեծ միջուկով convolution, ապա max pooling։ Գնալով convolution-ի միջուկի չափսը փոքրացնում ենք, իսկ feature map-ի խորությունը մեծացնում։ Convolution հետո max pooling զույգի կիրառությունը շատ նման է LeNet-ի կառուցվածքին։ Տարբերությունն այն է, որ այստեղ, երբեմն նկարի վրա կիրառում ենք padding և մեկ անգամ նկարի խորությունը չենք մեծացնում այլ թողնում ենք նույնը և հետո փոքրացնում ենք։ 120 նեյրոնի փոխարեն 4096-ի կիրառումը կապված է նրանով, որ output-ում ունենք 1000 class և LeNet-ի դեպքում պարամետրերի քանակը կշատանար, ցանցի աշխատանքը կդանդաղեր։

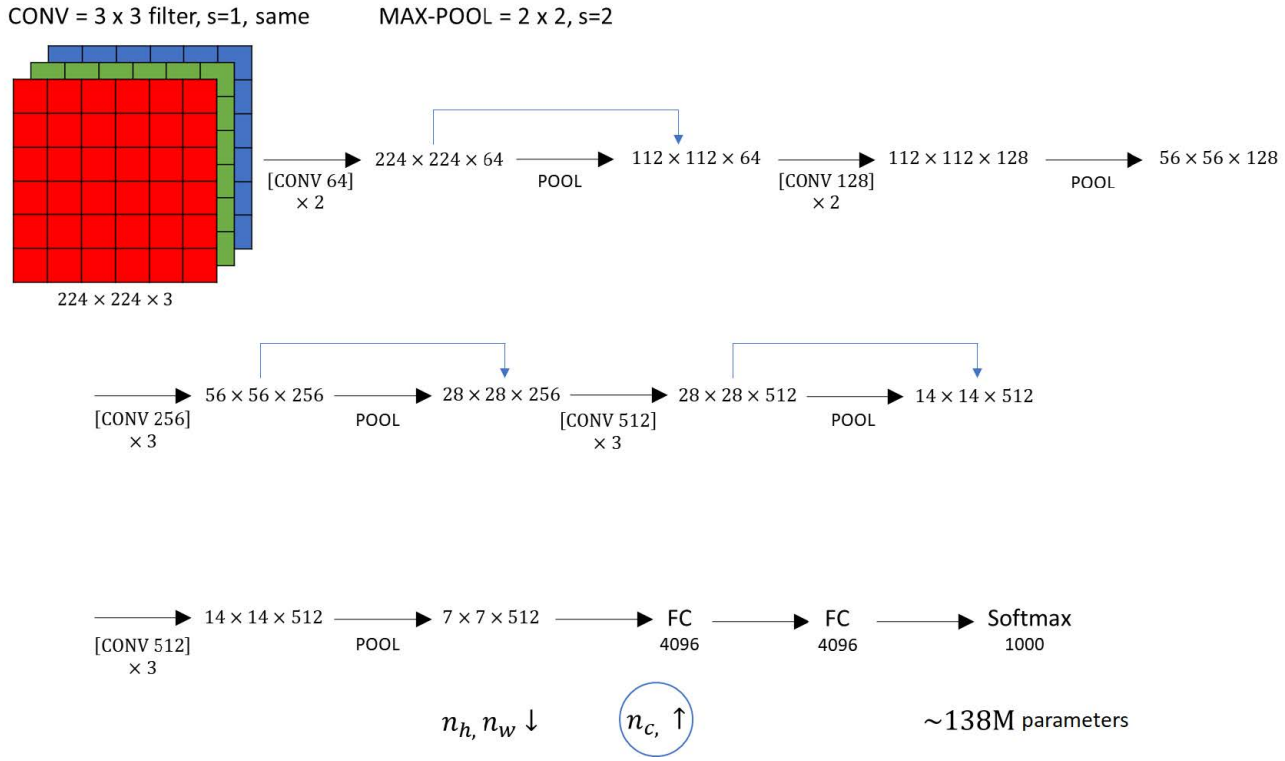
---

1. ImageNet - 1000 class ունեցող գունավոր նկարների հավաքածու։ Նկարների մեջ առկա են տարբեր տեսակի շներ (doberman, poodle), շնաձկներ (blue shark, tiger shark), այդ պատճառով էլ class-երի քանակը շատ է։

Այս ցանցի բոլոր շերտերում, բացի վերջինից, ակտիվացիոն ֆունկցիաները relu են: Շերտերում նաև կիրառված է local response normalization, որը նորմավորման եղանակ է, որը օգտագործվել է մինչ batch normalization-ի հիայտ գալը: Այս ցանցը ունի 60 միլիոն պարամետր և ImageNet-ի վրա ցուցաբերել է հետևյալ արդյունքները՝ Top1[1]=63.3%, Top5[2]=84.6%: Top5 ճշգրտությունը 20%-ով տարբերվում է Top1-ից, քանի որ ցանցը կարող է հասկանալ, որ նկարում շուն է պատկերված, բայց սխալվի շան տեսակն ասելիս:

3 VGG-16 (2014)

VGG ցանցի հեղինակներն են՝ Կարեն Սիմոնյանը և Andrew Zisserman-ը: Ցանցի կառուցվածքը պատկերված է նկար 3-ում:



Նկար 3: VGG-16 փաթույթային ցանցի կառուցվածքը

Ցանցի առանձնահատկություններից է միջուկի չափսը փիքսելը: Բոլոր տեղերում, որտեղ կիրառում ենք convolution, միջուկի չափսը 3×3 է, stride=1, padding=same (նկարի ինչ չափի մուտքում եկավ նույն չափի էլ դուրս է գալիս): Բոլոր տեղերում, որտեղ կիրառում ենք max pooling միջուկի չափսը 2×2, stride=2, նկարի չափը երկու անգամ փոքրանում է: Այս քայլերը անելով մեզ մնում է ընտրել մի քանի հիպերպարամետրեր՝ քանի անգամ կիրառել նկարի վրա convolution և ինչ խորությամբ (filter): Մուտքային նկարները 224×224×3 չափի են: Սկզբից երկու անգամ կիրառում ենք 64 filter ունեցող convolution, ապա max pooling: Հետո նորից երկու անգամ կիրառում ենք, արդեն 128 խորությամբ convolution ու նորից max-pooling:

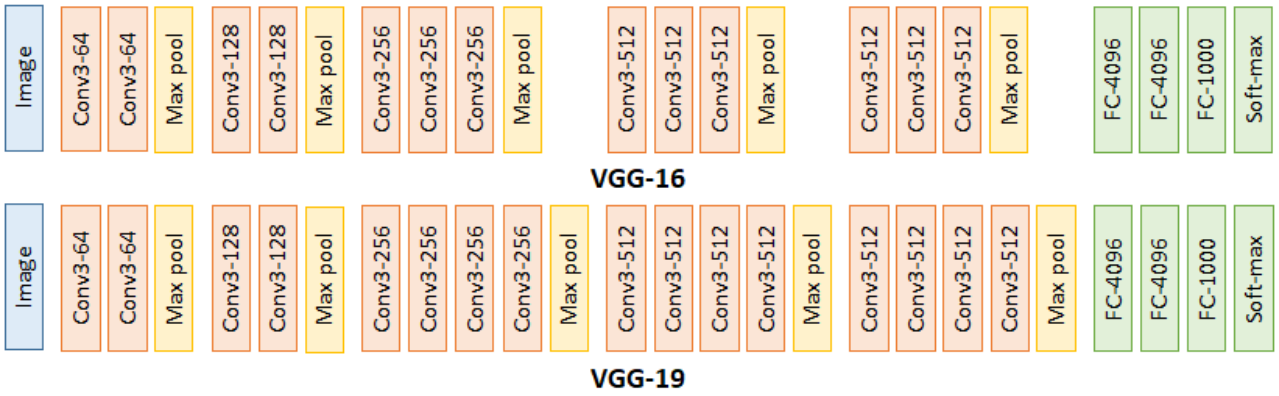
---

1. Top1 - Վերցնում ենք մոդելի վերադարձրած ամենամեծ հավանականությամբ class-ը և համեմատում իրական class-ի հետ: Եթե համընկնում են, ապա ճիշտ գուշակածների քանակը ավելացնում ենք մեկով և վերջում ճշգրտությունը հաշվում ենք հետևյալ բանաձևով. $\frac{\text{Ճիշտ գուշակածներ}}{\text{Բոլոր գուշակածներ}}$
2. Top5 - Վերցնում ենք մոդելի վերադարձրած ամենամեծ 5 հավանականությունները: Եթե իրական class-ը առկա է դրանց մեջ, ապա ճիշտ գուշակածների քանակը ավելացնում ենք մեկով:

Հետո 3 անգամ կիրառում ենք 256 խորությամբ convolution և max pooling: Բոլոր max pooling-ները կատարվում են մեկ անգամ: Հետո 3 անգամ կիրառում ենք 512 filter convolution և max pooling: Վերջում նորից 3 անգամ կիրառում ենք 512 filter convolution և max pooling: Ստացված $7\times7\times512$ feature map-ը հարթեցնում ենք և հերթով փոխանցում dense layer-ներին:
Այս ցանցում բոլոր շերտերում, բացի վերջինից, ակտիվացիոն ֆունկցիաները relu են:
Գաղափարը, որը կարող ենք հասկանալ այս ցանցից հետևյալն է. կարող ենք convolution միջուկների չափերը բոլոր տեղերում ֆիքսել՝ քչացնելով հիպերպարամետրերի քանակը: Ամեն քայլի feature map-ի խորությունը շատացնում ենք, իսկ չափը փոքրացնում, այսինքն նկարի մասին ինֆորմացիան տարածում ենք feature map-ի խորությունների մեջ:
Այս ցանցը ունի մոտավորապես 138 միլիոն պարամետր (երկու անգամ AlexNet-ից շատ) և 16 շերտ: ImageNet-ի վրա ցուցաբերել է հետևյալ արդյունքները՝ Top1=74.4%, Top5=91.9%:

4 VGG-19 (2014)

VGG-19 ցանցը VGG-16 ցանցն է, ավելացված 3 շերտ: Որտեղ 3 անգամ էինք կիրառում convolution, այստեղ կիրառում ենք 4 անգամ (Նկար 4):



Նկար 4: VGG-16 և VGG-19 փաթույթային ցանցերի կառուցվածքների համեմատություն

Այս ցանցն ունի 144 միլիոն պարամետր, 6 միլիոն պարամետր ավելի քան VGG-16-ը: ImageNet-ի վրա ցուցաբերել է հետևյալ արդյունքները՝ Top1=74.5%, Top5=92%: Ընդամենը 0.1% բարելավում: Այսինքն ավելի շատ պարամետրեր ունենալը չի նշանակում, որ ճշգրտությունը կլավանա:

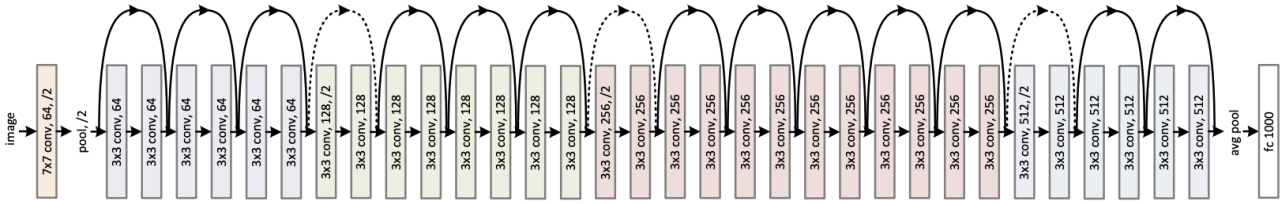
5 Resnet-50 (2015)

VGG-19-ի դեպքում տեսանք, որ ցանցը ավելի խորացնելիս ցանցի ճշգրտությունը չնչին չափով է լավանում: Խորը ցանցերում կարող են առաջանալ երկու տեսակի խնդիրներ կորչող գրադիենտ (vanishing gradient) և մեծացող գրադիենտ (exploding gradient): Դիտարկենք, որ ունենք 100 շերտից բաղկացած նեյրոնային ցանց և այդ ցանցում ակտիվացիոն ֆունկցիան relu է: Ամեն շերտից դուրս է գալիս հետևյալը՝ $relu(wx+b)$: x-ը այս դեպքում հավասար է նախորդ շերտից դուրս եկած output-ին: Դիտարկենք առաջին 3 շերտերից հետո output-ը:

$$x->layer1=relu(w_1x+b_1)$$
$$layer1->layer2=relu(w_2relu(w_1x+b_1)+b_2)$$
$$layer2->layer3=relu(w_3relu(w_2relu(w_1x+b_1)+b_2)+b_3)$$

relu ակտիվացիոն ֆունկցիան բացասական արժեքները դարձնում է 0, իսկ դրականները թողնում է նույնը: Այսինքն w_1x+b_1-ի արժեքը հնարավոր է մնա նույնը, կամ մի մասը մնա նույնը մյուս մասը դառնա 0: Հաջորդ շերտ փոխանցելիս՝ ստացված արդյունքը բազմապատկվում

է w_2-ով: Նորից մի մասը մնում է նույնը, մյուս մասը դառնում է 0: Հետո 3-րդ շերտում բազմապատկվում է w_3-ով: Արդյունքում կունենանք հետևյալ՝ $w_1w_2w_3$ արտադրյալը: 100 շերտով անցնելուց հետո՝ $w_1w_2 \dots w_{100}$: w-ների սկզբնարժեքավորումը հիմնականում լինում է 0-ի միջակայքում և այս բազմապատկումից հետո իրենց արժեքը շատ մոտ կլինի 0-ին: w_1-ը թարմացնելիս կունենանք հետևյալ $w_2w_3 \dots w_{100}$ արտադրյալը և w_1-ի արժեքը չի թարմացվի: Կառաջանա vanishing gradient-ի խնդիր: Կամ w-ների 1-ից մեծ արժեքի դեպքում exploding gradient-ի խնդիր: Այս խնդիրներից խուսափելու համար առաջացել է Resnet-50 ցանցը (Նկար 5):



Նկար 5: Resnet-50 փաթույթային ցանցի կառուցվածք

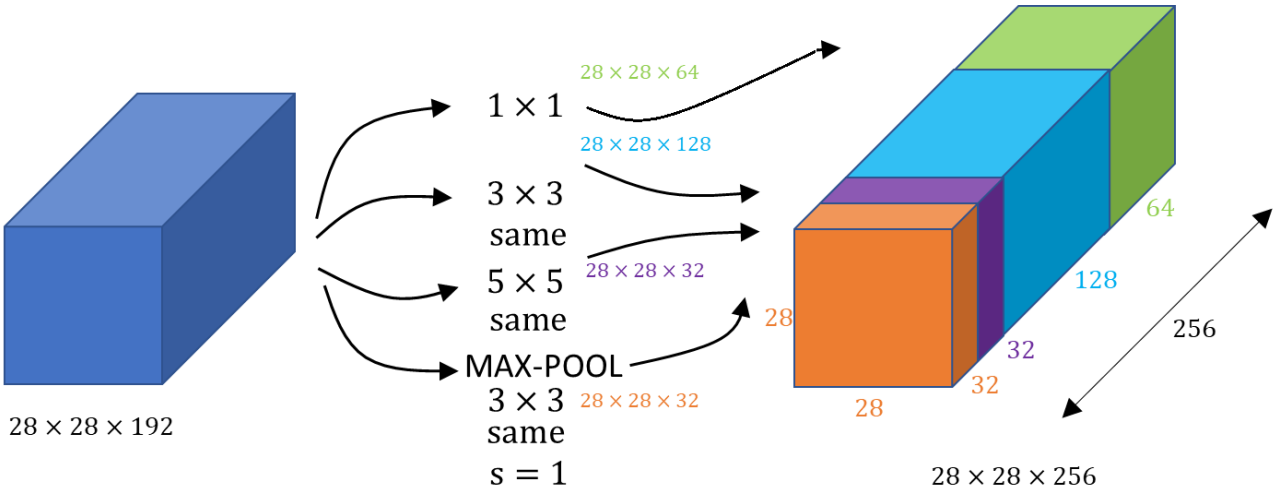
Վերցնենք մեր շերտից դուրս եկած output-ը և գումարենք իրենից երկու շերտ հետո դուրս եկած output-ին: Հետո ստացվածը գումարենք երկու շերտ հետո դուրս եկածին և այդպես մինչև ցանցի ավարտը: Օրինակ՝ առաջին շերտից դուրս է եկել $relu(w_1x + b_1)$, այս արժեքը հարկավոր է գումարել երկու շերտ հետո դուրս եկած output-ին: Երկու շերտ հետո դուրս եկած $output_3$-ը՝ $relu(w_3relu(w_2relu(w_1x + b_1) + b_2) + b_3)$: Մնում է հասկանանք $output_3$-ի վրա կիրառելու ենք ակտիվացիոն ֆունկցիա հետո գումարենք, թե գումարենք հետո կիրառենք ակտիվացիոն ֆունկցիա: Նեյրոնային ցանցերում սկզբնական շերտերը սովորում են ավելի ընդհանրական հատկանիշներ (low level features), իսկ վերջին շերտերը ավելի խնդրին հատուկ (high level features): Օրինակ՝ փաթույթային ցանցերի սկզբնական շերտերը կարող են սովորել նկարից առանձնացնել եզրերը (հորիզոնական կամ ուղղահայաց), վառ գույները և այլ բոլոր նկարներին հատուկ հատկանիշներ: Իսկ վերջնական շերտերը այնքան խնդրին հատուկ են, որ վերջին շերտում մենք մեկ գծային ձևափոխում՝ $wx + b$ կատարելուց հետո կարողանում ենք ասել, թե նկարը որ class-ին է պատկանում: Եթե $output_3$-ի վրա կիրառենք ակտիվացիոն ֆունկցիա, հետո գումարենք նախորդ շերտի output-ը, այդ դեպքում սկզբնական շերտի output-ը կհասնի մինչև վերջին շերտ: Նախավերջին շերտին կգումարենք $relu(w_1x+b_1)$ արժեքը և նոր կփոխանցենք վերջին շերտին: Ստացվում է, որ low level feature-ը գումարում ենք high level feature-ին: Իսկ դա կարող է շփոթեցնել ցանցին և այն չկարողանա լավ արդյունք ցուցաբերել: Այդ պատճառով, սկզբից $output_3$-ին գումարում ենք երկու շերտ առաջ դուրս եկած output-ը, ապա կիրառում ակտիվացիոն ֆունկցիան: Եվ այս դեպքում վերջին շերտում չենք ունենա $relu(w_1x+b_1)$ գումարելին:

Նկար 5-ում գծերով ցույց է տալիս, որ շերտի output-ը որին ենք գումարում: Որտեղ, որ մուգ գծերի փոխարեն պատկերված են կետագծեր, նշանակում է, որ output-ի խորությունը փոփոխում ենք նոր ենք գումարում: Առաջին կետագծերի դեպքում շերտից դուրս եկած output-ը ունի 128 խորություն (filter), իսկ երկու շերտ առաջ output-ը 64 խորություն: Դրա համար մեզ անհրաժեշտ է 64 խորությունը դարձնել 128: Կարող ենք ըստ խորության կրկնօրինակել 64 խորությամբ feature map-ը և ստանալ 128 խորություն, բայց դա կաշխատի եթե 2 անգամ է խորությունների միջև տարբերությունը: Դրա համար կիրառում ենք 1×1 convolution և այս դեպքում կարող ենք փոխել feature map-ի խորությունը ինչքան ցանկանանք:

Գաղափարը, որը կարող ենք հասկանալ այս ցանցից հետևյալն է. խորը ցանցերի դեպքում մեր սկզբնական սիգնալը կորչում է w-ների հետ բազմապատկման արդյունքում և առաջանում vanishing gradient խնդիր: Դրա համար ամեն երկրոր շերտին գումարում ենք երկու շերտ առաջվա output-ը, որպեսզի սիգնալը չթուլանա, այսինքն ըստ w-ի ածանցելիս այդ գումարելիի ածանցյալը արդեն շատ փոքր չի լինի: Feature map-ի խորությունը փոփոխելու համար օգտագործում ենք 1×1 convolution: Այս ցանցը ունի 50 շերտ և 25.6 միլիոն պարամետր: ImageNet-ի վրա ցուցաբերել է հետևյալ ճշգրտությունները՝ Top1 = 77.15%, Top5 = 93.29%:

6 GoogLeNet/Inception v1 (2014)

VGG ցանցի դեպքում մենք ֆիքսում էինք միջուկի չափը ամբողջ ցանցի ընթացքում և միայն որոշում էինք, որ շերտից հետո, որը դնենք: Այս ցանցի դեպքում մենք մի շերտում միաժամանակ կիրառում ենք մի քանի փաթույթներ, ինչպես նաև max pooling (Նկար 6) Ինչպես երևում է

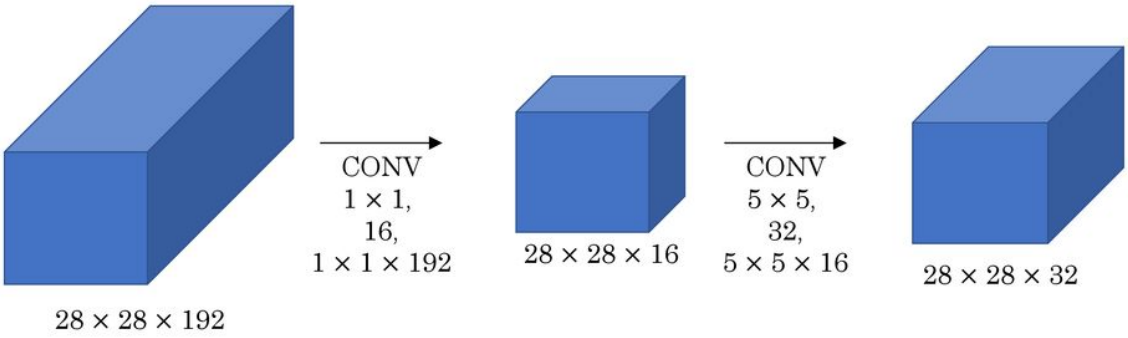


Նկար 6: GoogLeNet փաթույթային ցանցի մեկ շերտի կառուցվածք

նկար 6-ում, մուտքային նկարի վրա կիրառում ենք 1×1 convolution, փոփոխելով խորությունը և դարձնելով 64, ապա սկզբնական նկարի վրա կիրառում ենք 3×3 convolution, padding=same և խորությունը դարձնում 128, հետո սկզբնական նկարի վրա կիրառում ենք 5×5 convolution, padding=same և խորությունը 32 նաև կիրառում ենք max pooling 3×3, padding=same, խորությունը 32: Բոլոր գործողությունների ժամանակ stride=1: Վերջում ստացված feature map-երը ըստ խորության միացնում ենք իրար և արդյունքում ստանում 28×28×256 feature map: Max pooling-ը գիտենք, որ խորությունը չի փոփոխում և 32 խորություն դարձնելու համար max pooling-ից հետո կիրառում ենք 1×1 convolution: Հիմա եկեք հաշվենք գործողությունների քանակը, երբ 28×28×192 feature map-ի վրա կիրառում ենք 5×5 միջուկով, padding=same և 32 խորությամբ convolution: Բազմապատկումների քանակը հավասար կլինի.

$$28 \cdot 28 \cdot 32 \cdot 5 \cdot 5 \cdot 192 \approx 120M$$

Այստեղ կարող ենք տեսնել, որ բազմապատկումների քանակը այսքան շատացնում է խորությունը (192): Այդ պատճառով կարող ենք սկզբից խորությունը փոքրացնել՝ կիրառելով 1×1 convolution, ապա կիրառել մեր ուզած convolution-ը (Նկար 7):

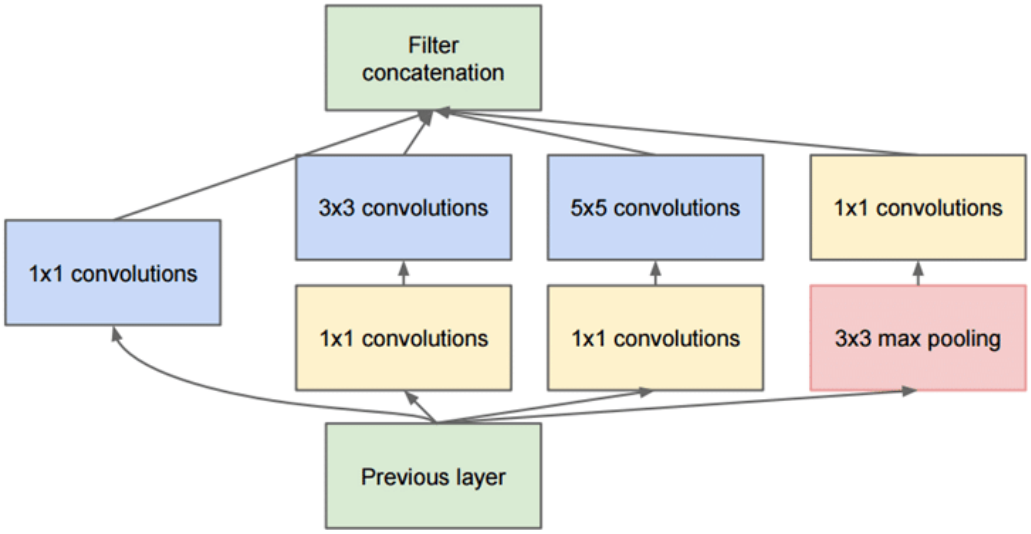


Նկար 7: Սկզբից կիրառենք 1×1×192×16 convolution, ապա 5×5×16×32

Այս դեպքում բազմապատկումների քանակը հավասար կլինի.

$$28 \cdot 28 \cdot 16 \cdot 1 \cdot 1 \cdot 1 \cdot 192 + 28 \cdot 28 \cdot 32 \cdot 5 \cdot 5 \cdot 16 \approx 12.4M$$

Այսինքն օգտագործելով սկզբից 1×1 convolution, ապա 3×3 կամ 5×5 convolution կարող ենք քչացնել բազմապատկումների և կշիռների քանակը: Այս ցանցում նաև կիրառվել է մի քանի

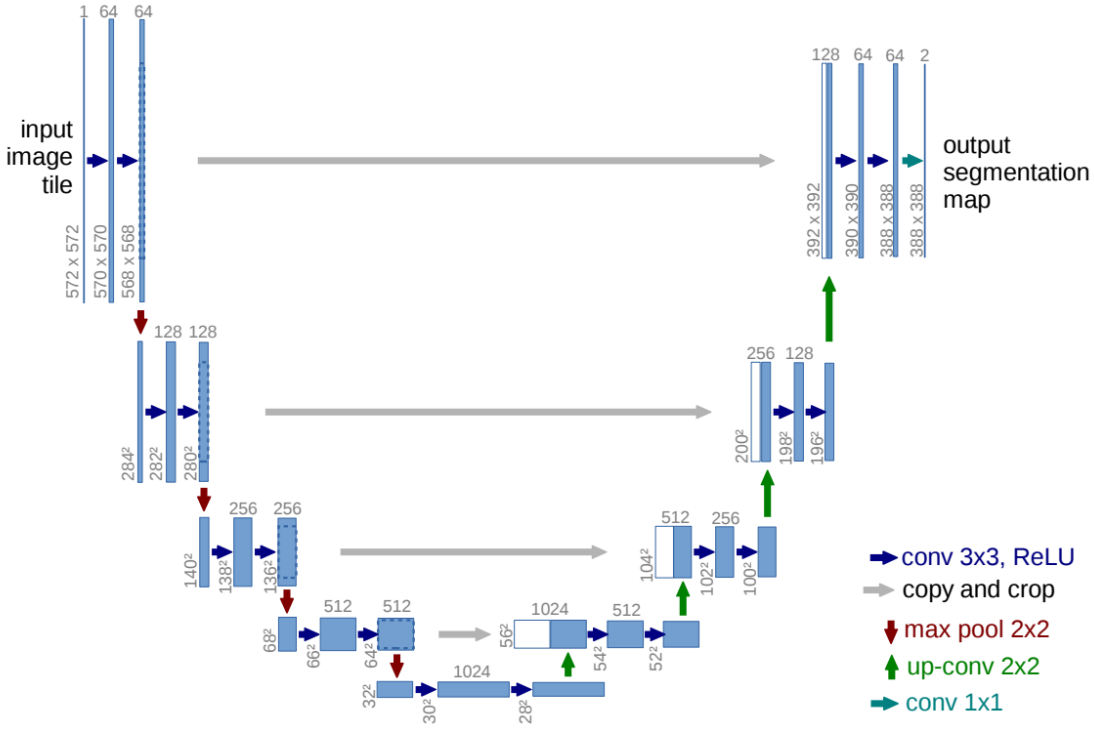


Նկար 8: 1x1 convolution հետո 3×3 կամ 5×5

կորուստ ունենալու տեխնիկան: Պատկերացնենք ցանցի 1/3 մասից, 2/3 մասից և ամենավերջից դնում ենք նույն կորստի ֆունկցիաները: Ամբողջ ցանցը ունի 3 կորստի ֆունկցիա: Առաջին կորուստը միայն վերաբերում է առաջին 1/3 շերտերին (նշանակենք L1), երկրորդը առաջին 2/3 շերտերին (նշանակենք L2), իսկ վերջին կորուստը բոլոր շերտերին (նշանակենք L3): Դրա շնորհիվ սկզբի շերտերի կշիռները թարմացնելիս, ըստ L3-ի ածանցյալը կարող է շատ փոքր լինել ցանցի խորության պատճառով, L2-ի ածանցյալը նույնպես փոքր կլինի, իսկ L1-ի ածանցյալը ըստ սկզբնական շերտի կշիռների ավելի մեծ կլինի: Նույն կորուստը կիրառելը նշանակում է, որ այդ կետում, որտեղ կիսել ենք (1/3) դնում ենք 1000 class-ից բաղկացած softmax և output-ը համեմտում իրական class-ի հետ: Սա vanishing gradient-ից խուսափելու եղանակ է:
Գաղափարը, որը կարող ենք հասկանալ այս ցանցից հետևյալն է. Մի շերտում կարող ենք կիրառել միանգամից մի քանի convolution և max pooling: Ցանցում կարող ենք ունենալ մի քանի output ցանցի տարբեր մասերից և դա կօգնի խուսափել vanishing gradient խնդրից: 1×1 convolution-ը կարող ենք օգտագործել խորություն փոփոխելու համար:
Այս ցանցը ունի 5 միլիոն պարամետր: ImageNet-ի վրա ցուցաբերել է հետևյալ ճշգրտությունները. Top1 = 69.8%, Top5 = 89.9%: Այս ցանցը նաև անվանում են Inception v1: Inception v2-ը տարբերվում է նրանով, որ բոլոր նորմավորումները փոխարինված են batch normalization-ով: Դրա արդյունքում հնարավոր է եղել ավելի խորացնել ցանցը և այն ունի 11.2 միլիոն պարամետր: ImageNet-ի վրա ցուցաբերել է հետևյալ արդյունքները. Top1 = 74.8%, Top5 = 92.2%: Receptive field-ի ժամանակ ծանոթացանք, որ 5×5 convolution-ները երկու հատ 3×3 convolution-ներով փոխարինելիս կշիռների քանակը քչանում է, իսկ receptive field-ը մնում է նույնը: Նույն կերպ n×n convolution-ի փոխարեն n×1 և 1×n կիրառելիս նորից կշիռների քանակը քչանում է, receptive field-ը մնում է նույնը: Այս երկու տեխնիկաները կիրառվել են Inception v3-ում: Այն ունի 23.8 միլիոն պարամետր և ImageNet-ի վրա ցուցաբերել է հետևյալ արդյունքները. Top1 = 78.8%, Top5 = 94.4%: Inception v4-ը ավելի է խորացրել ցանցը օգտագործելով ResNet ցանցի տեխնիկան և դրա հաշվին ունի 55.8 միլիոն պարամետր և ImageNet-ի վրա ցուցաբերել է հետևյալ արդյունքները. Top1 = 80.1%, Top5 = 95.1%:
Այս պահին ImageNet-ի վրա լավագույն արդյունք ցուցաբերած ցանցը ունի 2440 միլիոն պարամետր, Top1 = 80.1%, Top5 = 95.1%

7 UNet

Դիտարկենք հետևյալ խնդիրը: Ունենք նկար, նկարի մեջ կենտրոնում պատկերված է մարդը և պետք է նկարում պահել միայն մարդուն, իսկ ետնամասը (background) ջնջել: Դրա համար մեզ անհրաժեշտ է մի ցանց, որը կկարողանա նկարի ամեն պիքսել տարբերակել, արդյոք դա մարդու մի մասն է, թե ոչ (segmentation): Այս խնդիրը կարելի է լուծել UNet ցանցի օգնությամբ (Նկար 9):



Նկար 9: UNet փաթույթային ցանցի կառուցվածքը

Սկզբից նկարի վրա երկու անգամ կիրառում ենք 3×3 միջուկով convolution և $572\times572\times1$ նկարից ստանում ենք $568\times568\times64$ չափի feature map: Հետո կիրառում ենք սովորական max pooling, այսինքն նկար չափերը երկու անգամ փոքրացնում ենք: Հետո նորից երկու անգամ convolution, ապա max pooling: Այս գործողությունները կատարում ենք այնքան մինչև ստանում ենք $28\times28\times1024$ չափի feature map (Նկար 9-ի ներքևի հատվածը): Դրանից հետո մեզ անհրաժեշտ է մեծացնել նկարը: Նկարը մեծացնելու համար կիրառում ենք transposed convolution կամ որ նույնն է up convolution: Ամեն անգամ up convolution կատարելիս, համապատասխան max pooling-ի feature map-ը crop ենք անում և միացնում ենք մեր այս պահի feature map-ին ըստ խորության: Օրինակ՝ վերջին max pooling-ի feature map-ը $64\times64\times512$ չափի է: Մեր այս պահի feature map-ը՝ $56\times56\times512$: Max pooling-ի feature map-ը crop ենք անում մեջտեղից և ըստ խորության միացնում մեր եղած feature map-ին: Քանի որ գիտենք, որ նկարում մարդը գտնվում է մեջտեղում, այդ պատճառով crop ենք անում մեջտեղից: Ըստ խորության միացումները նկար 9-ում նշված են մոխրագույն սլաքներով: Հետո նորի ստացված $104\times104\times1024$ feature map-ի վրա երկու անգամ կիրառում ենք convolution, հետո up convolution: Այսինքն max pooling-ի շերտերը փոխարինում ենք up convolution-ով: Այս գործողությունները շարունակում ենք այնքան ժամանակ մինչև ստանում ենք $388\times388\times64$ չափի feature map (Նկար 9-ի նախավերջին շերտը): Ստացվածի վրա կիրառում ենք 1×1 convolution և խորությունը փոփոխում ենք դարձնելով 2, feature map-ի չափը կդառնա $388\times388\times2$:
Իսկ ինչի՞ համար են այդ երկու output-ները: Առաջին feature map-ի առաջին պիքսելը ասելու է, թե ինչքան հավանականությամբ է այդ պիքսելում մարդ պատկերված, իսկ երկրորդ feature map-ի առաջին պիքսելը ասելու է, թե ինչքան հավանականությամբ այդտեղ մարդ պատկերված չէ: Այդ հավանականությունների գումարը հավասար է մեկի: Այսինքն վերջին շերտում ակտիվացիոն ֆունկցիան softmax է: Երկու class-ի դեպքում, ինչպես dense շերտերում, այնպես էլ այստեղ

կարող էինք ունենալ մի output և կիրառել sigmoid ակտիվացիոն ֆունկցիա, բայց եթե class-երի քանակը շատ լինի պետք է օգտագործենք softmax և ունենանք class-երի քանակով output: