ԲՈՎԱՆԴԱԿՈՒԹՅՈՒՆ

ԽՆԴՐԻ ԴՐՎԱԾՔ

Նախագծի նպատակն է մշակել համակարգ, որը կկատարի ձեռագիր թվերի սեգմենտացիա պատկերներում: Սեգմենտացիան ենթադրում է պատկերում առկա թվային սիմվոլների տեղերի ճշգրիտ նույնականացում և առանձնացում դիմակների (մասկաների) միջոցով: Սույն աշխատանքում մենք դիտարկում ենք երկուական (binary) սեգմենտացիայի խնդիր MNIST թվանշանների հավաքածուի վրա:

Հիմնական պահանջներ՝

1. Թվերի տեղայնացում:

   Համակարգը պետք է ավտոմատ կերպով հայտնաբերի պատկերում առկա բոլոր ձեռագիր թվերը:

2. Դիմակների (մասկաների) ստեղծում:

   Յուրաքանչյուր հայտնաբերված թվի համար համակարգը պետք է գեներացնի երկուական դիմակ, որտեղ թվին պատկանող պիքսելները նշված են մեկ արժեքով, իսկ մնացածը՝ զրո արժեքով:

ՆԵՐԱԾՈՒԹՅՈՒՆ

Ձեռագիր թվերի սեգմենտացիան շարունակում է մնալ արդիական խնդիր պատկերների մշակման և արհեստական բանականության ոլորտներում հետևյալ պատճառներով.

1. Փաստաթղթերի թվայնացում - Չնայած տեխնոլոգիական առաջընթացին, դեռևս գոյություն ունեն բազմաթիվ ձեռագիր փաստաթղթեր, որոնք պահանջում են ավտոմատացված մշակում:

2. Բարդ ֆոնի վրա ճանաչման խնդիրները - Իրական կիրառություններում ձեռագիր թվերը հաճախ գտնվում են բարդ ֆոնի վրա, ինչը պահանջում է ավելի խորը սեգմենտացիա, քան ավանդական OCR մեթոդները:

3. Անհստակ սահմաններ - Ձեռագիր թվերը կարող են լինել անհստակ, տարբեր չափերի և ձևերի, հաճախ միահյուսված, ինչը դժվարացնում է դրանց առանձնացումը:

4. U-Net-ի նորարարական կիրառում - Այս արխիտեկտուրան, որը սկզբնապես մշակվել է բժշկական պատկերների համար, առաջարկում է խոստումնալից մոտեցում ձեռագիր թվերի սեգմենտացիայի համար:

5. Ռեսուրսների օպտիմալացում - Ճշգրիտ սեգմենտացիան նվազեցնում է հետագա մշակման համար անհրաժեշտ հաշվողական ռեսուրսները:

U-Net-ը իր encoder-decoder կառուցվածքով և skip connection-ներով առանձնահատուկ հարմար է այս խնդրի համար, քանի որ այն պահպանում է տարածական տեղեկատվությունը և հստակ սահմանների որոշման հնարավորություն է տալիս:

Տվյալ պրոյեկտը նպատակ ունի հետազոտել U-Net-ի հնարավորությունները ձեռագիր թվերի սեգմենտացիայի խնդրում՝ առաջարկելով օպտիմալ մոդել իրական կիրառական խնդիրների համար:

ԳԼՈՒԽ 1 ՏԵՍԱԿԱՆ ԱՌԸՆՉՈՒԹՅՈՒՆՆԵՐ

1.1 ՍԵԳՄԵՆՏԱՑԻԱ ԵՎ ՍԵԳՄԵՆՏԱՑԻԱՅԻ ՏԵՍԱԿՆԵՐԸ

Պատկերի սեգմենտացիան (Image Segmentation) համակարգչային տեսության (Computer Vision) հիմնական խնդիրներից մեկն է, որի նպատակը պատկերը բաժանելն է առանձին հատվածների՝ հիմնվելով որոշակի հատկանիշների վրա: Այս գործընթացը թույլ է տալիս օբյեկտները կամ հետաքրքրող տարածքները տարբերակել մնացած պատկերից: Սեգմենտացիան կարելի է իրականացնել տարբեր մեթոդներով՝ կախված խնդրի բնույթից և տվյալների բնութագրերից:

Պատկերի սեգմենտացիայի տեսակներն են՝

1) Շեմային (Thresholding) սեգմենտացիա
    - Ամենապարզ մեթոդներից մեկն է:
    - Օգտագործում է պայծառության կամ գույնի շեմային արժեք՝ պիքսելները երկու կամ ավելի դասերի բաժանելու համար:
    - Կարող է լինել միաստիճան կամ բազմաստիճան, որտեղ շեմը փոխվում է՝ կախված տեղային պայմաններից:

Օրինակ՝ եթե ունենք սև-սպիտակ պատկեր, կարող ենք սահմանել մի շեմ (օրինակ՝ 128), և բոլոր պիքսելները, որոնց պայծառությունը մեծ է 128-ից, դարձնել սպիտակ, իսկ մնացածը՝ սև:

2) Եզրագծային (Edge-based) սեգմենտացիա
    - Հիմնվում է պիքսելների պայծառության կտրուկ փոփոխությունների վրա:
    - Օգտագործում է եզրագծերի հայտնաբերման ալգորիթմներ, օրինակ՝ Canny Edge Detector, Sobel Filter կամ Laplacian Filter:

Օրինակ՝ եթե ուզում ենք առանձնացնել օբյեկտի եզրագծերը, կարող ենք օգտագործել Canny Edge Detector-ը, որը թույլ կտա պարզ ձևով տեսնել ուրվագծերը:

3) Տարածքային (Region-based) սեգմենտացիա
    - Հիմնված է համասեռ հատկանիշներ ունեցող տարածքների նույնականացման վրա:

- Ընդհանուր մոտեցումներն են՝՝ Region Growing, Region Splitting and Merging:

Օրինակ՝ Region Growing ալգորիթմը սկսում է մեկ կամ մի միավորից (seed points) և աստիճանաբար ավելացնում հարևան պիքսելները, որոնք նման են նախնականին:

4) Կլաստերային (Clustering-based) սեգմենտացիա
   - Հիմնված է մաթեմատիկական խմբավորման տեխնիկաների վրա:
   - Օգտագործում է K-Means Clustering, Gaussian Mixture Models (GMM) կամ Mean-Shift Clustering ալգորիթմները:

Օրինակ՝ K-Means-ը բաժանում է պատկերը K խմբերի՝ հիմնվելով գույնի կամ պայծառության հատկությունների վրա:

5) Խոր ուսուցմամբ (Deep Learning-based) սեգմենտացիա
   - Օգտագործում է խոր նեյրոնային ցանցեր՝ ավելի ճշգրիտ արդյունքների համար:
   - Լայն տարածում ունեցող մոդելներն են՝ U-Net, Mask R-CNN, DeepLab: Համեմատությունը ներկայացված է աղ 1.1-ում:

Օրինակ՝ U-Net մոդելը լայնորեն կիրառվում է բժշկական պատկերների մշակման մեջ՝ օրգանների և ախտաբանական փոփոխությունների սեգմենտացիայի համար:

Աղյուսակ 1.1 U-Net, Mask R-CNN և DeepLab ճարտարապետությունների համեմատություն:

Հատկանիշ	**U-Net**	**Mask R-CNN**	**DeepLab**
Ճարտարապետության տեսակը	Կոդավորիչ-ապակոդավորիչ (encoder-decoder) skip կապերով	Երկաստիճան դետեկտոր օբյեկտների հատվածավորմամբ	CNN կոնվոլյուցիաներով և ASPP-ով
Ներկայացման տարեթիվը	2015	2017	2017
Հատվածավորման տեսակը	Իմաստային հատվածավորում	Օբյեկտային հատվածավորում	Իմաստային հատվածավորում

Հատկանիշ	U-Net	Mask R-CNN	DeepLab
Հիմնական նորարարությունը	Skip կապեր կոդավորիչի և ապակոդավորիչի միջև	Faster R-CNN-ի ընդլայնում՝ դիմակի ճյուղով	Տարածական Բուրգային Փուլինգ (ASPP)
Հիմնական ցանց	Հատուկ (սկզբնապես)	ResNet և այլն	ResNet, Xception, MobileNet
Մուտքի չափը	Ճկուն, հաճախ 572×572	Ճկուն	Ճկուն
Արագությունը	Արագ	Դանդաղ (երկաստիճան հայտնաբերում)	Միջին
Ուժեղ կողմերը	Արդյունավետ է սահմանափակ տվյալներով ցանցում Լավ է բժշկական պատկերների համար Պահպանում է տարածական տեղեկատվությունը	Կարող է տարբերակել օբյեկտների օրինակները Լավ է բարդ տեսարանների համար Բազմախնդիր ուսուցում (հայտնաբերում + հատվածավորում)	Բարձր ճշգրտություն Աշխատում է տարբեր մասշտաբներով Լավ է մանրամասն սահմանագծերի համար
Սահմանափակումները	Չի տարբերակում օբյեկտների օրինակները Արդյունավետ չէ բնական տեսարանների համար	Հաշվողական տեսանկյունից թանկ Բարդ ուսուցման գործընթաց	Հաշվողական տեսանկյունից ինտենսիվ Ավելի բարդ է, քան U-Net-ը
Հիշողության պահանջները	Ցածրից միջին	Բարձր	Միջինից բարձր

1.2 ԴԱՍԱԿԱՐԳՄԱՆ ԵՎ ՍԵԳՄԵՆՏԱՑԻԱՅԻ ՏԱՐԲԵՐՈՒԹՅՈՒՆԸ

Պատկերի դասակարգումը որոշում է, թե պատկերը որ դասին է պատկանում: Այսինքն՝ համակարգը ամբողջ պատկերը դասակարգում է նախապես սահմանված դասերի (classes) մեջ:

Օրինակ՝ ունենք կատուների և շների նկարներ, դասակարգման մոդելը պատկերը կդասակարգի որպես Կատու կամ Շուն, բայց չի նշի, թե որ հատվածում է կենդանին: Հիմնական մեթոդներ՝

- Դասական մեքենայական ուսուցում՝ SVM, Random Forest, KNN
- Խոր ուսուցման մոդելներ՝ CNN (Convolutional Neural Networks)

Պատկերի Սեգմենտացիա (Image Segmentation) բաժանում է պատկերը հատվածների՝ հիմնվելով գույնի, հյուսվածքի կամ ձևի վրա, կարող է սահմանազատել օբյեկտները:

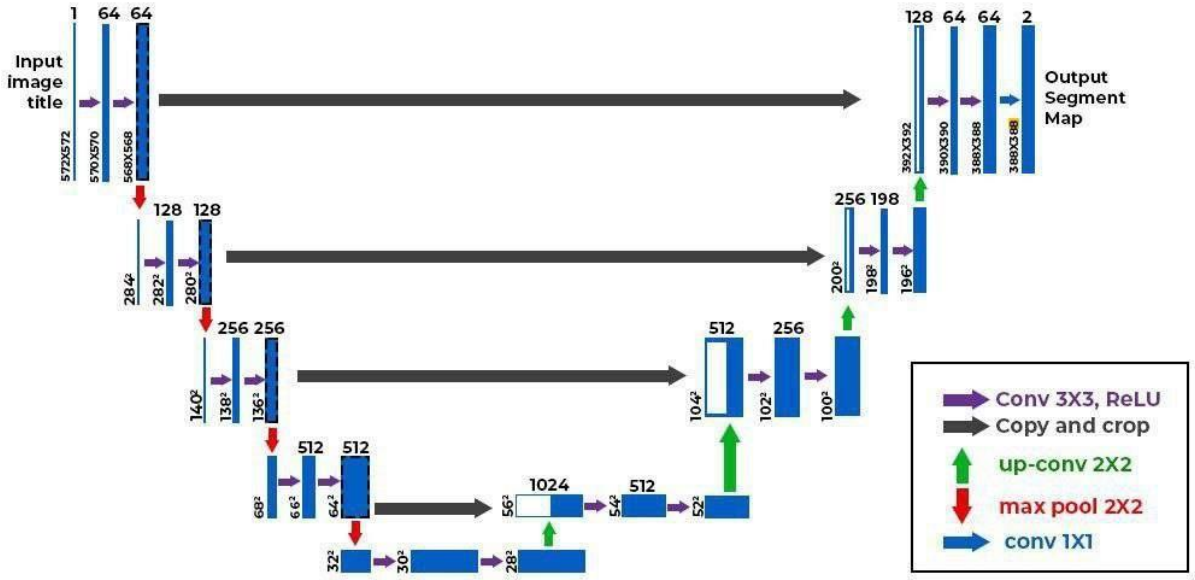
Օրինակ՝ ունենք կատուներ և շներ, սեգմենտացիան կարող է սահմանազատել նրանց ուրվագծերը պատկերում:

Աղ 1.1 Դասակարգման և սեգմենտացիայի համեմատություն

	Դասակարգում (Classification)	Սեգմենտացիա (Segmentation)
Հիմնական Նպատակ	Պատկերի դասակարգում որոշակի խմբերի մեջ	Պատկերի հատվածների բաժանում
Արդյունք	Մեկ կամ մի քանի դասերի պատկանելիություն	Մասշտաբային քարտեզ (mask), որտեղ տարբեր գույներով նշված են տարբեր հատվածները
Տեսակներ	Single-label classification Multi-label classification	Semantic segmentation Instance segmentation
Օգտագործվող մոդելներ	CNN, ResNet, EfficientNet, VGG	U-Net, Mask R-CNN, DeepLabV3

1.3 U-NET ՃԱՐՏԱՐԱՊԵՏՈՒԹՅԱՆ ԿԱՌՈՒՑՎԱԾՔԸ

U-Net-ը խորը ուսուցման ճարտարապետություն է, որը մշակվել է 2015 թվականին Օլաֆ Ռոննեբերգի և գործընկերների կողմից՝ բժշկական պատկերների սեգմենտացիայի համար: Այն իր անունը ստացել է U-աձև կառուցվածքից, որը կազմված է կոդավորիչից (encoder), "կամրջից" (bottleneck) և ապակոդավորիչից (decoder)(նկ.1.1):



Նկ.1.1 U-NET ճարտարապետության կառուցվածքը:

Կոդավորիչը U-Net-ի ձախ մասն է, որը կատարում է հատկանիշների արտահանում և տարածական չափերի նվազեցում: Այն բաղկացած է հաջորդական կոնվոլյուցիոն բլոկներից, որոնցից յուրաքանչյուրի կազմում ներառված են.

- Երկու 3×3 կոնվոլյուցիոն շերտ, յուրաքանչյուրին հաջորդում է ReLU ակտիվացում
- 2×2 MaxPooling գործողություն, որը կրճատում է տարածական չափերը երկու անգամ

Հատկանիշների քարտեզների (feature maps) քանակը սովորաբար կրկնապատկվում է ամեն MaxPooling գործողությունից հետո, ինչը թույլ է տալիս ցանցին սովորել ավելի բարդ և վերացական հատկանիշներ:

Կամուրջը կապակցող օղակ է կոդավորիչի և ապակոդավորիչի միջև: Այս մասում պատկերի տարածական չափերը նվազագույնն են, սակայն հատկանիշների քարտեզների քանակը՝ առավելագույնը: Այստեղ կատարվում են հատկանիշների արտահանման վերջին գործողությունները երկու կոնվոլյուցիոն շերտերի միջոցով:

Ապակոդավորիչը U-Net-ի աջ մասն է, նպատակն է վերականգնել տարածական չափերը և ստեղծել սեգմենտացիայի քարտեզ: Այն բաղկացած է հաջորդական ապակոդավորման բլոկներից, որոնցից յուրաքանչյուրի կազմում ներառված են. • UpSampling (2×2) գործողություն, որը մեծացնում է տարածական չափերը երկու անգամ

- Համապատասխան կոդավորիչի հատկանիշների քարտեզների հետ միավորում (concatenation) skip connection-ների միջոցով
- Երկու 3×3 կոնվոլյուցիոն շերտ, յուրաքանչյուրին հաջորդում է ReLU ակտիվացում

Վերջին շերտը 1×1 կոնվոլյուցիոն շերտ է սիգմոիդ ակտիվացիայով, որն հաշվում է հավանականությունը յուրաքանչյուր պիքսելի համար:

U-Net-ի ամենակարևոր նորարարություններից մեկը skip connection-ների կիրառումն է, որոնք ուղղակիորեն կապում են կոդավորիչի համապատասխան շերտերը ապակոդավորիչի շերտերի հետ: Այս միացումները մի քանի կարևոր առավելություններ ունեն.

1. Տարածական տեղեկատվության պահպանում

   Կոդավորիչում կատարվող տարածական կրճատումների ընթացքում կորցվում է դիրքային տեղեկատվություն, որն անհրաժեշտ է ճշգրիտ սեգմենտացիայի համար: Skip connection -ները թույլ են տալիս պահպանել այս տեղեկատվությունը:

2. Գրադիենտների ավելի լավ տարածում

   Խորը ցանցերում գրադիենտների անհետացման խնդիրը մեղմելու համար skip connection-ները ապահովում են այլընտրանքային ուղիներ ուսուցման ընթացքում:

3. Բազմամասշտաբային վերլուծություն

   Տարբեր մակարդակների հատկանիշների համակցումը թույլ է տալիս մոդելին օգտագործել ինչպես բարձր մակարդակի, այնպես էլ մանր դետալների վերաբերյալ տեղեկատվություն:

U-Net-ի կարևոր առանձնահատկություններից է, որ այն կարող է արդյունավետ աշխատել համեմատաբար փոքր տվյալների հավաքածուների հետ: Սակայն, ավելի լավ արդյունքներ ստանալու համար, սովորաբար օգտագործվում են տվյալների հավելման տեխնիկաներ՝

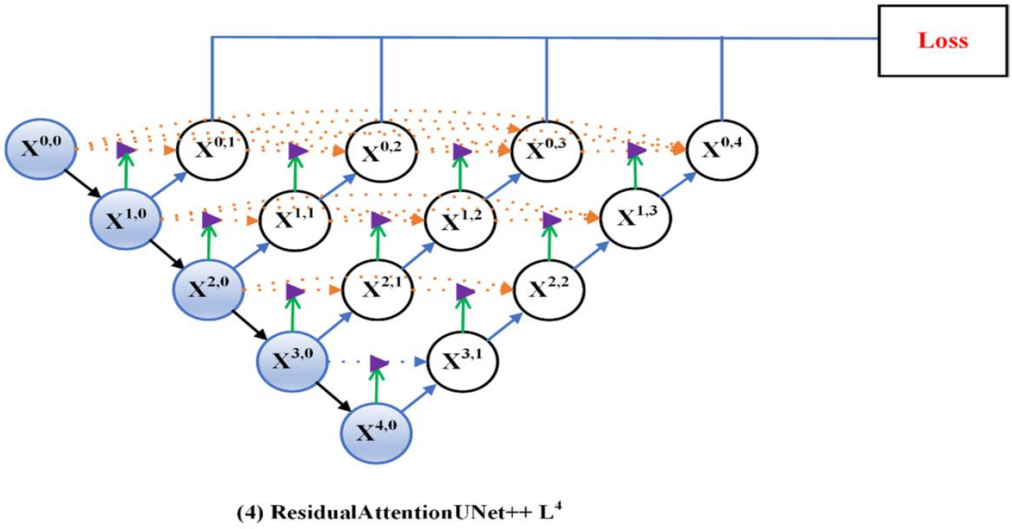
- Պատահական պտույտներ
- Մասշտաբավորում
- Հորիզոնական և ուղղահայաց շրջում
- Պայծառության և կոնտրաստի փոփոխություններ
- Կտրվածքներ (cropping)
- Էլաստիկ դեֆորմացիաներ

U-Net-ի ուսուցման համար հաճախ օգտագործվում են ժամանակակից օպտիմիզացիոն ալգորիթմներ, ինչպիսիք են՝

- Adam
- RMSProp
- SGD (ուժեղացված մոմենտումով)

Սովորաբար օգտագործվում է նվազող ուսուցման արագություն (learning rate decay), որը նպաստում է մոդելի կոնվերգենցիային:
Սկզբնական U-Net ճարտարապետության հիման վրա ստեղծվել են բազմաթիվ բարելավված տարբերակներ՝

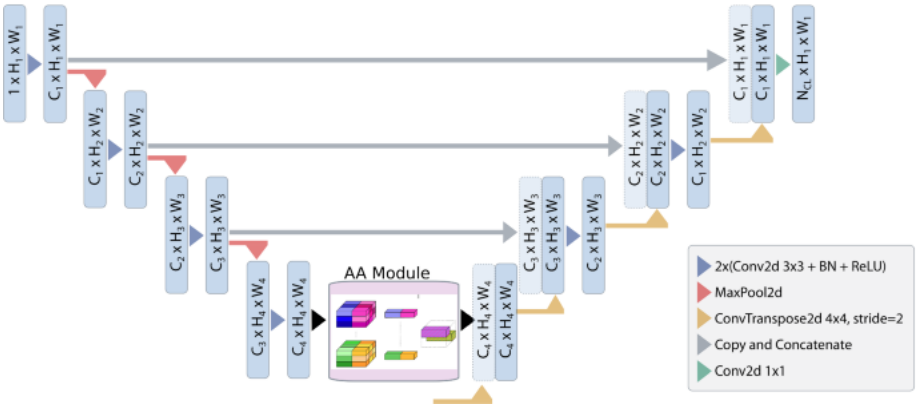
U-Net++-ը (նկ. 1.2) ընդլայնում է սկզբնական ճարտարապետությունը՝ ավելացնելով խիտ (dense) skip connection-ներ կոդավորիչի և ապակոդավորիչի բլոկների միջև: Այս փոփոխությունը նվազեցնում է սեմանտիկ խզումը (semantic gap) տարբեր մակարդակների հատկանիշների քարտեզների միջև: U-Net++-ը նաև ներառում է ճյուղավորման և միաձուլման մեխանիզմներ (deep supervision), որոնք թույլ են տալիս օգտագործել տարբեր խորության ենթացանցեր:



Նկ. 1.2 U-Net+-ի ճարտարապետության կառուցվածքը:

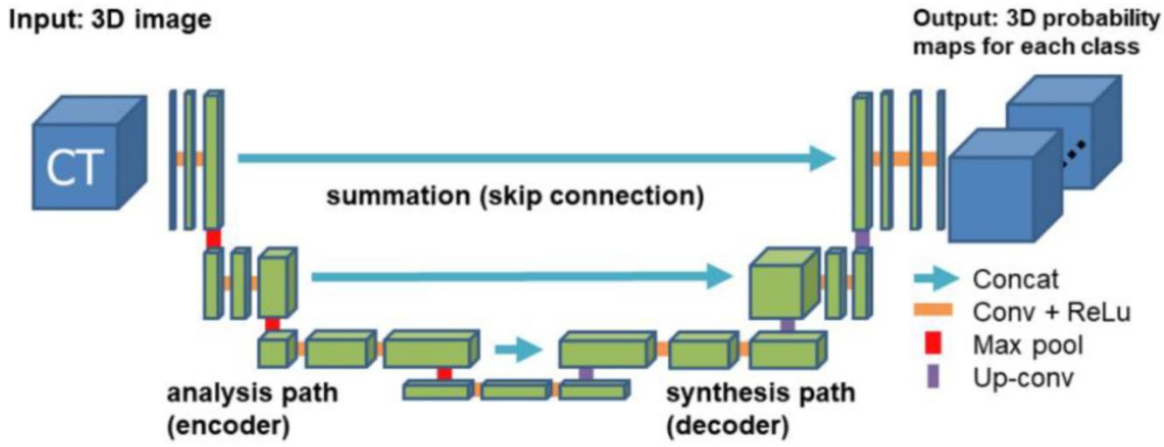
Attention U-Net-ը (նկ. 1.3) ներդնում է ուշադրության մեխանիզմ (attention mechanism) կոնվոլյուցիոն գործողությունների մեջ, որը թույլ է տալիս մոդելին ավելի լավ կենտրոնանալ կարևոր հատվածների վրա: Ուշադրության մեխանիզմը կիրառվում է յուրաքանչյուր skip connection-ում, որպեսզի ապակոդավորիչը կարողանա ընտրել համապատասխան հատկանիշներ կոդավորիչից:

Ուշադրության մեխանիզմը մաթեմատիկորեն կարելի է ներկայացնել հետևյալ կերպ՝ $a_i = \sigma(W_a(U_i) + W_b(F_i))$ $F_i^{'} = a_i \odot F_i$, որտեղ a_i ուշադրության գործակիցն է, σ ակտիվացիայի ֆունկցիան, իսկ $\odot$ տարրի-առ-տարր բազմապատկումը:



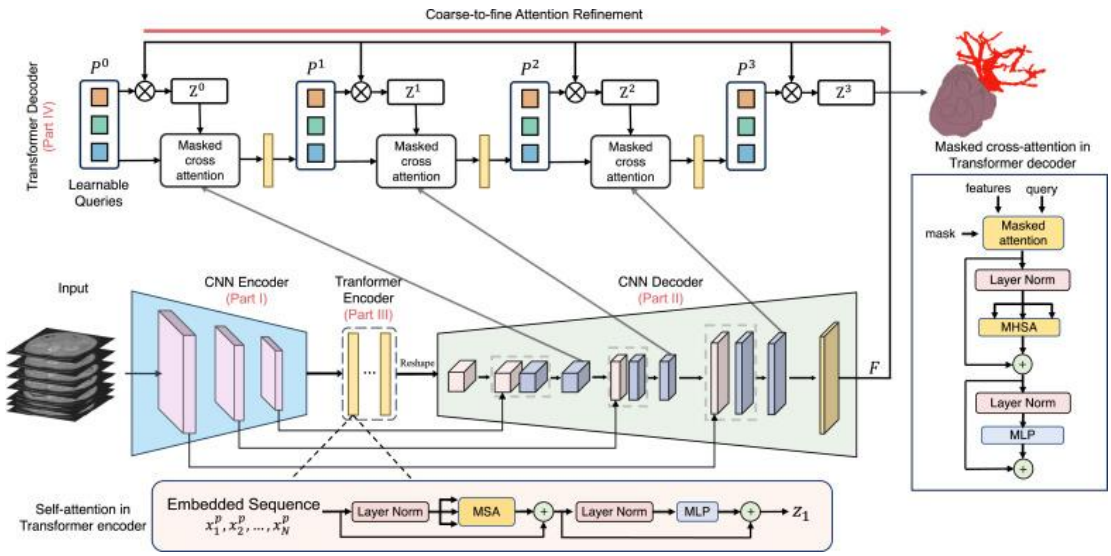
Նկ. 1.3 Attention U-Net-ի ճարտարապետության կառուցվածքը:

3D U-Net-ը (նկ. 1.4) մշակվել է եռաչափ պատկերների սեգմենտացիայի համար, ինչպիսիք են MRI և CT սկանները: Այն կիրառում է 3D կոնվոլյուցիոն և պուլինգ գործողություններ 2D գործողությունների փոխարեն, թույլ տալով մոդելին սովորել ծավալային հատկանիշներ:



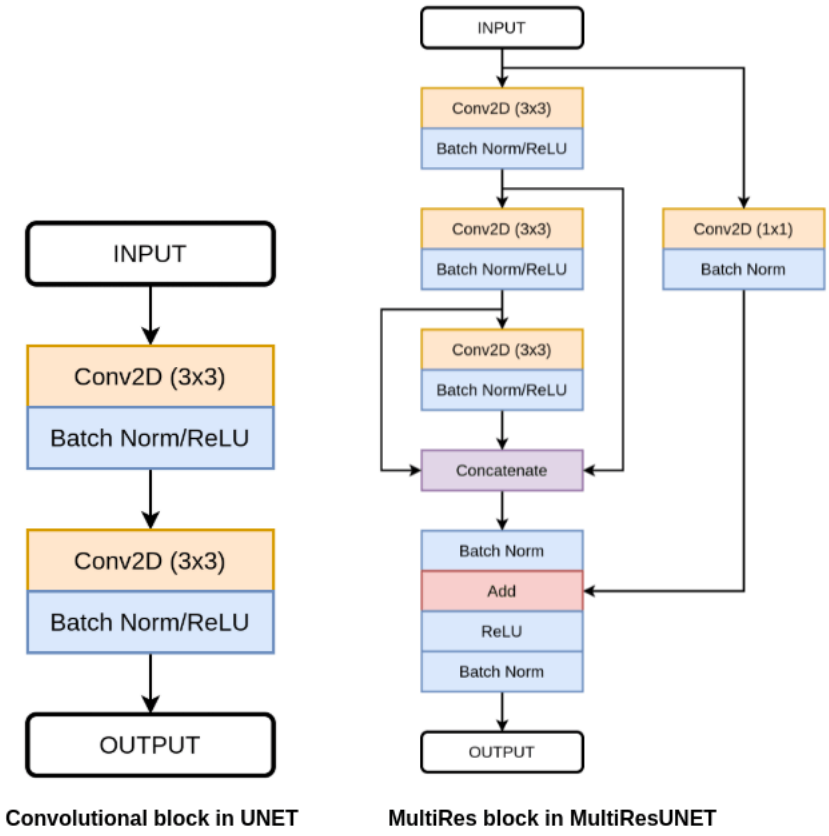
Նկ. 1.4 3D U-Net-ի ճարտարապետության կառուցվածքը:

TransUNet-ը (նկ. 1.5) համակցում է տրանսֆորմերների ճարտարապետությունը U-Net-ի հետ: Այն օգտագործում է Vision Transformer (ViT) որպես կոդավորիչ՝ երկարատև կախվածությունները սովորելու համար, և ապակոդավորիչ U-Net-ի ստանդարտ ապակոդավորիչի նման:



Նկ. 1.5 TransUNet-ի ճարտարապետության կառուցվածքը:

MultiResUNet-ը (նկ. 1.6) ներմուծում է MultiRes բլոկներ, որոնք ավելի արդյունավետ կերպով են մշակում տարբեր մասշտաբների հատկանիշները: Այն նաև օգտագործում է ցածր-ռեզոլյուցիայի մնացորդային ուղիներ (residual paths), որոնք բարելավում են գրադիենտների տարածումը ցանցում:



Նկ. 1.6 MultiResUNet -ի ճարտարապետության կառուցվածքը:

U-Net-ը սկզբնապես մշակվել է բջիջների սեգմենտացիայի համար: Այսօր այն լայնորեն օգտագործվում է տարբեր բժշկական պատկերների սեգմենտացիայի համար.

- Օրգանների սեգմենտացիա CT և MRI պատկերներում
- Ուռուցքների հայտնաբերում
- Անոթների սեգմենտացիա
- Ռետինայի պատկերների վերլուծություն
- Ոսկրերի և հոդերի տարանջատում ռենտգեն պատկերներում

U-Net-ը օգտագործվում է հեռազննման (remote sensing) և GIS համակարգերում.

- Հողօգտագործման դասակարգում
- Ճանապարհների և շենքերի հայտնաբերում
- Անտառների և բնական տարածքների քարտեզագրում
- Աղետների հետևանքների գնահատում

Արդյունաբերական կիրառություններ՝

- Արտադրական գործընթացներում որակի վերահսկում
- Նյութերի ատոմային կառուցվածքի վերլուծություն
- Քիմիական միացությունների կառուցվածքի սեգմենտացիա

Պատկերների վերականգնում և գեներացիա՝

U-Net-ի տարբերակներն օգտագործվում են.

- Պատկերների ապակոնվոլյուցիայի (deconvolution) համար
- Պատկերների աղմուկազերծման համար
- Քերծվածքների և փսասվածքների վերականգնման համար
- Գեներատիվ մոդելներում, ինչպես օրինակ Pix2Pix-ում

Առավելություններ՝

1. Բարձր ճշգրտություն

   U-Net-ը ապահովում է բարձր ճշգրտություն նույնիսկ քիչ քանակությամբ ուսուցման տվյալների դեպքում:

2. Պատկերների տարբեր մասշտաբների հատկանիշների համակցում

   Skip connection-ների շնորհիվ ցանցը կարողանում է արդյունավետ կերպով համադրել տարբեր մակարդակների հատկանիշներ:

3. Բազմակողմանիություն

   U-Net-ը կարող է հարմարեցվել տարբեր ոլորտների խնդիրներին:

4. Փոքր տվյալների հավաքածուների հետ աշխատանք:

   Տվյալների հավելման տեխնիկաների օգտագործմամբ U-Net-ը կարող է արդյունավետ աշխատել համեմատաբար փոքր ուսուցման հավաքածուների հետ:

Թերություններ

1. Հաշվարկային ռեսուրսների պահանջներ:

   Խորը U-Net մոդելները կարող են պահանջել զգալի հաշվարկային հզորություն:

2. Հիպերպարամետրերի կարգավորում:

   Լավագույն արդյունքների հասնելու համար հաճախ անհրաժեշտ է մանրակրկիտ կարգավորել հիպերպարամետրերը:

3. Տարածական ճշգրտության սահմանափակումներ:

   Չնայած skip connection-ներին, կարող են լինել մանր դետալների կորուստներ:

Կատարողականության բարելավման մեթոդներ՝

1. BatchNormalization շերտերի ավելացում:

   Կոնվոլյուցիոն շերտերից հետո BatchNormalization-ի ավելացումը կարող է արագացնել ուսուցումը և բարելավել կոնվերգենցիան:

2. Dropout-ի կիրառում:

   Ուսուցման ընթացքում 0.3-0.5 հավանականությամբ Dropout-ի կիրառումը կարող է նվազեցնել overfitting-ը:

3. Փոխված ակտիվացիա:

   ReLU-ի փոխարեն կարելի է կիրառել LeakyReLU կամ ELU ակտիվացիաներ, որոնք կարող են բարելավել ուսուցումը:

4. Տվյալների նորմալիզացիա:

   Մուտքային պատկերների նորմալիզացիան [0,1] կամ [-1,1] միջակայքերում կարող է նպաստել ուսուցման կայունությանը:

1.4 ՕԳՏԱԳՈՐԾՎԱԾ ԱԿՏԻՎԱՑՄԱՆ ՖՈՒՆԿՑԻԱՆԵՐ, ԿՈՐՍՏԻ ՖՈՒՆԿՑԻԱ ԵՎ ՕՊՏԻՄԻԶԱՑԻԱ

ReLU ակտիվացման ֆունկցիան սահմանվում է հետևյալ կերպ՝

$$f(x) = \max(0, x)$$

Այն ամենատարածված ակտիվացման ֆունկցիան է խորը ուսուցման մեջ հետևյալ առավելությունների շնորհիվ՝

- Պարզ հաշվարկ և արագ կատարում
- Գրադիենտների անհետացման խնդրի մեղմացում
- Ոչ գծային հատկություններ, որոնք թույլ են տալիս մոդելին սովորել բարդ օրինաչափություններ

Մեր U-Net մոդելում ReLU-ն օգտագործվում է բոլոր թաքնված շերտերում: Սիգմոիդ ակտիվացման ֆունկցիան սահմանվում է հետևյալ կերպ՝

$$\sigma(x) = 1 / (1 + e^{(-x)})$$

Այն արտապատկերում է մուտքային արժեքները [0, 1] միջակայքում, ինչը հատկապես հարմար է երկուական դասակարգման խնդիրների համար: Մեր մոդելում սիգմոիդը կիրառվում է վերջին շերտում՝ յուրաքանչյուր պիքսելի համար սեգմենտացիայի հավանականություն ստանալու համար:

Երկուական սեգմենտացիայի խնդիրների համար հաճախ օգտագործվում է երկուական խաչաձև էնտրոպիայի (binary cross-entropy) կորստի ֆունկցիան, որը սահմանվում է հետևյալ կերպ՝

$$-\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

y_i-ն իրական պիտակն է, իսկ p_i-ն՝ մոդելի կանխատեսած հավանականությունը:

Օպտիմիզացիայի համար հիմնականում օգտագործվում է Ադամ (Adam) ալգորիթմը, որը համակցում է մոմենտումի և ադապտիվ ուսուցման տեմպի առավելությունները՝ արագ և կայուն զուգամիտման համար:

ԳԼՈՒԽ 2 ԾՐԱԳՐԻ ԻՐԱԿԱՆԱՑՈՒՄ

2.1 ՕԳՏԱԳՈՐԾՎԱԾ ՏԵԽՆՈԼՈԳԻԱՆԵՐ ԵՎ ԳՐԱԴԱՐԱՆՆԵՐԸ

Ծրագրի համար օգտագործվել են հետևյալ գրադարանները և տեխնոլոգիաները՝

TensorFlow-ն և Keras-ը Google-ի կողմից մշակված խորը ուսուցման համակարգեր են, որոնք լայնորեն օգտագործվում են մեքենայական ուսուցման հետազոտություններում և արդյունաբերական կիրառություններում: Մեր ծրագրում.

- TensorFlow-ն օգտագործվում է ցածր մակարդակի հաշվարկների համար
- Keras-ը հանդիսանում է բարձր մակարդակի API, որը թույլ է տալիս արագ կառուցել և ուսուցանել մոդելներ

Մասնավորապես, մենք օգտագործել ենք Keras-ի layers մոդուլը՝ ցանցի շերտերը կառուցելու համար, և Model դասը՝ ամբողջական մոդելը սահմանելու համար:

NumPy-ն գրադարան է գիտական հաշվարկների համար, առաջարկում է միջոցներ բազմաչափ զանգվածների հետ աշխատելու համար: Ծրագրում օգտագործվում է՝

- Տվյալների նախապատրաստման համար
- Զանգվածների ձևափոխման համար
- Թվային գործողությունների իրականացման համար

Matplotlib-ը Python-ի թվային տվյալների վիզուալիզացիայի գրադարան է: Մեր ծրագրում այն օգտագործվում է.

- Բնօրինակ պատկերների ցուցադրման համար
- Ground Truth դիմակների ցուցադրման համար
- Մոդելի կանխատեսումների վիզուալիզացիայի համար
- Արդյունքների համեմատական վերլուծության համար

OpenCV-ն պատկերի մշակման բաց կոդով գրադարան է: Մեր ծրագրում այն օգտագործվում է՝

- Պատկերների բեռնման համար
- Պատկերների չափերի փոփոխման համար
- Սև-սպիտակ փոխակերպումների համար
- Թեստային պատկերների նախապատրաստման համար

2.2 ՏՎՅԱԼՆԵՐԻ ՆԱԽԱՊԱՏՐԱՍՏՈՒՄ, ՆՈՐՄԱԼԻԶԱՑԻԱ, ՁԵՎԱՎՈՐՈՒՄ, ՍԵԳՄԵՆՏԱՑԻԱ, ԴԻՄԱԿՆԵՐԻ ՍՏԵՂԾՈՒՄ

MNIST տվյալների հավաքածուն ներառված է TensorFlow-ի տվյալների հավաքածուների մեջ և հեշտությամբ հասանելի է mnist մոդուլի միջոցով:

Բեռնված տվյալները ներառում են.

- x_train - 60,000 ուսուցողական պատկեր (28×28 պիքսել)
- y_train - 60,000 ուսուցողական պիտակ (0-9 թվանշաններ)
- x_test - 10,000 թեստային պատկեր (28×28 պիքսել)
- y_test - 10,000 թեստային պիտակ (0-9 թվանշաններ)

Նախքան նեյրոնային ցանցի ուսուցումը, անհրաժեշտ է նորմալիզացնել պատկերները, որպեսզի պիքսելների արժեքները գտնվեն [0, 1] միջակայքում՝ բաժանելով դրանք 255-ի:

- x_train = x_train / 255.0
- x_test = x_test / 255.0

Այնուհետև, պատկերները վերաձևավորվում են U-Net մոդելի համար պահանջվող ձևաչափին: Քանի որ մոդելը պահանջում է 4D մուտքային տվյալներ, ավելացվում է լրացուցիչ չափ՝ ներկայացնելու խորությունը (channels):

- x_train = np.expand_dims(x_train, axis=-1)
- x_test = np.expand_dims(x_test, axis=-1)

MNIST-ը չի պարունակում պատրաստի սեգմենտացիայի դիմակներ, այդ պատճառով մենք ստեղծում ենք դրանք՝ հիմնվելով պիքսելների ինտենսիվության վրա: Դիմակները ստեղծվում են՝ օգտագործելով 0.5 շեմային արժեք (նորմալիզացված պատկերների համար):

- y_train = (x_train > 0.5).astype(np.uint8)
- y_test = (x_test > 0.5).astype(np.uint8)

Այս գործընթացում՝

- Եթե պիքսելի արժեքը մեծ է 0.5-ից, այն համարվում է թվանշանի մաս (1)
- Եթե պիքսելի արժեքը փոքր է կամ հավասար 0.5-ի, այն համարվում է ֆոն (0)

2.3 ՄՈԴԵԼԻ ԿԱՌՈՒՑՎԱԾՔԸ

Մոդելը հիմված է U-Net ճարտարապետության վրա, հետևյալ բաղադրիչներից՝

Կոդավորիչը բաղկացած է երկու հիմնական բլոկից, յուրաքանչյուրը ներառում է երկու կոնվոլյուցիոն շերտ, որին հաջորդում է MaxPooling:

Առաջին կոդավորման բլոկ՝

- c1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
- c1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(c1)
- p1 = layers.MaxPooling2D((2, 2))(c1)

Երկրորդ կոդավորման բլոկ՝

- c2 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(p1)
- c2 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(c2)
- p2 = layers.MaxPooling2D((2, 2))(c2)

Կամուրջը կազմված է երկու շերտից՝ առավելագույն հատկանիշների քարտեզներով:

- c3 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(p2)
- c3 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(c3)

Ապակոդավորիչը կազմված է երկու բլոկից, յուրաքանչյուրը ներառում է UpSampling, կոդավորիչի հատկանիշների քարտեզների հետ միավորում և երկու կոնվոլյուցիոն շերտ:

Առաջին ապակոդավորման բլոկ՝

- u1 = layers.UpSampling2D((2, 2))(c3)
- u1 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(u1)
- merge1 = layers.concatenate([c2, u1]) # Skip Connection
- c4 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(merge1)
- c4 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(c4)

Երկրորդ ապակոդավորման բլոկ՝

- u2 = layers.UpSampling2D((2, 2))(c4)
- u2 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(u2)
- merge2 = layers.concatenate([c1, u2]) # Skip Connection
- c5 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(merge2)
- c5 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(c5)

2.4 ՀԻՊԵՐՊԱՐԱՄԵՏՐԵՐ

Հիպերպարամետրերը հանդիսանում են մոդելի կառուցվածքի և ուսուցման գործընթացի կարևոր կարգավորիչներ, որոնք՝

- Որոշում են մոդելի ճարտարապետությունը
- Կառավարում են ուսուցման գործընթացի արագությունը և որակը
- Կանխարգելում են օվերֆիթինգը և անդերֆիթինգը
- Ազդում են մոդելի ճշգրտության վրա

Ծրագրում օգտագործվող հիպերպարամետրերը ներկայացված են աղ. 2.1-ում:

Աղ. 2.1 հիպերպարամետրերի համեմատություն:

Category	Hyperparameter	Value in Code
Model Architecture	Filters	`64, 128, 256`
	Kernel Size	`(3,3)`
	Activation Function	`'relu'`
	Dropout Rate	❌ (not used)
Training	Learning Rate	(default in `Adam()`, likely `0.001`)
	Batch Size	`32`
	Epochs	`5`
	Loss Function	`'binary_crossentropy'`
	Optimizer	`Adam()`
Data Processing	Image Size	`(28,28)`
	Normalization	`image / 255.0`
	Segmentation Threshold	`> 0.5`

2.5 ԾՐԱԳՐԻ ՄԱՆՐԱՄԱՍՆ ՆԿԱՐԱԳՐՈՒԹՅՈՒՆ

```
# Import libraries
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, Model
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
```

Ներմուծում է մեքենայական ուսուցման համար անհրաժեշտ գրադարաններ: Ահա ներմուծված գրադարանները՝

1. NumPy (np-ի տեսքով) - գիտական հաշվարկների համար հիմնական փաթեթ
2. TensorFlow (tf-ի տեսքով) - բաց կոդով մեքենայական ուսուցման հարթակ
3. TensorFlow Keras բաղադրիչներ (layers, Model) - նեյրոնային ցանցերի բարձր մակարդակի API
4. MNIST տվյալների հավաքածու TensorFlow Keras-ից - ձեռագիր թվանշանների հանրահայտ հավաքածու
5. Matplotlib-ի pyplot (plt-ի տեսքով) - գրաֆիկների պատկերման գրադարան:

```
[ ] # Load MNIST dataset
    (x_train, y_train), (x_test, y_test) = mnist.load_data()

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

Այս հատվածը ներբեռնում է MNIST տվյալների հավաքածուն:

1. mnist.load_data() ֆունկցիան կանչվում է, որպեսզի ներբեռնի MNIST տվյալների հավաքածուն
2. Տվյալները բաժանվում են չորս մասի՝
    - x_train - ուսուցման համար նախատեսված պատկերներ
    - y_train - ուսուցման պատկերների համապատասխան պատկերներ
    - x_test - ստուգման համար նախատեսված պատկերներ
    - y_test - ստուգման պատկերների համապատասխան պիտակներ

Ներբեռնման հաղորդագրությունը ցույց է տալիս, որ տվյալները ներբեռնվում են Google APIs-ից, և ներբեռնման գործընթացը ավարտվել է 1 վայրկյանում:

```
# Normalize and reshape
x_train = x_train / 255.0
x_test = x_test / 255.0
x_train = np.expand_dims(x_train, axis=-1) # (num_samples, height, width) to (num_samples, height, width, 1) (channel)
x_test = np.expand_dims(x_test, axis=-1) # (num_samples, height, width) to (num_samples, height, width, 1) (channel)
```

Նորմալիզացիա (Normalization)

- x_train և x_test տվյալները բաժանվում են 255.0-ի, որպեսզի արժեքները լինեն 0-ից 1 միջակայքում
- Սա կարևոր է, քանի որ MNIST-ի պատկերները սովորաբար պահվում են 0-255 արժեքներով (8-բիթանոց grayscale)

Տվյալների ձևափոխում (Reshaping):

- np.expand_dims() ֆունկցիայի օգնությամբ պատկերներին ավելացվում է լրացուցիչ չափողականություն
- Փոխարկվում է (num_samples, height, width) ձևաչափից (num_samples, height, width, 1) ձևաչափի
- Վերջին չափողականությունը (1) ներկայացնում է պատկերի ալիքների քանակը (grayscale պատկերների համար՝ 1):

```
# Create segmentation masks (if > 0.5 => white/digit, else background)
y_train = (x_train > 0.5).astype(np.uint8)
y_test = (x_test > 0.5).astype(np.uint8)
```

Տրված կոդը ստեղծում է սեգմենտացիոն դիմակներ (segmentation masks)՝ հիմնված x_train և x_testտվյալների վրա։

Ստեղծում է սեգմենտացիոն դիմակներ (եթե > 0.5 -> սպիտակ/թվանշան, այլապես ֆոն)

1. x_train > 0.5 - ստուգում է, թե արդյոք x_train-ի արժեքները մեծ են 0.5-ից։
    - Եթե այո (ճիշտ է), ստացվում է True։
    - Եթե ոչ (սխալ է), ստացվում է False։
2. .astype(np.uint8) - True-ն փոխակերպվում է 1, իսկ False-ը 0 uint8 (8-բիթանոց աննշանակ թիվ) տիպի տվյալով։
3. Նույնը կատարվում է x_test-ի համար՝ y_test-ում։

- Ստեղծվում է դիմակ, որտեղ
    - 1 (սպիտակ) նշանակելու է թվանշանը/օբյեկտը։

- 0 (սև) նշանակելու է ֆոնը։

```python
# Create model
def unet_model(input_shape=(28, 28, 1)):
    inputs = layers.Input(input_shape)

    # Encoder
    c1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
    c1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(c1)
    p1 = layers.MaxPooling2D((2, 2))(c1)

    c2 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(p1)
    c2 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(c2)
    p2 = layers.MaxPooling2D((2, 2))(c2)

    # Bottleneck
    c3 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(p2)
    c3 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(c3)

    # Decoder
    u1 = layers.UpSampling2D((2, 2))(c3)
    u1 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(u1)
    merge1 = layers.concatenate([c2, u1]) # Skip Connection

    c4 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(merge1)
    c4 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(c4)

    u2 = layers.UpSampling2D((2, 2))(c4)
    u2 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(u2)
    merge2 = layers.concatenate([c1, u2]) # Skip Connection

    c5 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(merge2)
    c5 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(c5)

    outputs = layers.Conv2D(1, (1, 1), activation='sigmoid')(c5)

    model = Model(inputs, outputs)
    return model
```

Ֆունկցիան unet_model() ստանում է մուտքային պատկերի չափը (input_shape=(28, 28, 1)) և վերադարձնում է U-Net մոդելը։

1. Մուտք (Input Layer)

inputs = layers.Input(input_shape)

Ստեղծում է մուտքային շերտ՝ (28, 28, 1) չափի։

2. Կոդեր (Encoder - Վերացարկում)

c1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)

c1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(c1)

p1 = layers.MaxPooling2D((2, 2))(c1)

- 2× 3x3 Convolution՝ ReLU ակտիվացմամբ, որից հետո MaxPooling՝ չափերը կիսելու համար։

- Նույնը կատարվում է հաջորդ շերտերում՝ ավելի շատ ֆիլտրերով (128, հետո 256):

c2 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(p1)

c2 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(c2)

p2 = layers.MaxPooling2D((2, 2))(c2)

3. Bottleneck (Խորքային շերտ)

c3 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(p2)

c3 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(c3)

- Ամենախոր մակարդակում ունենք 256 ֆիլտրով կոնվոլյուցիոն շերտեր:

4. Դեկոդեր (Decoder - Վերականգնում)

u1 = layers.UpSampling2D((2, 2))(c3)

u1 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(u1)

merge1 = layers.concatenate([c2, u1]) # Skip Connection

- UpSampling2D՝ չափերը կրկնապատկելու համար (upsampling):
- Skip Connection՝ merge1 = layers.concatenate([c2, u1])
    - Սա միացնում է վերականգնվող տվյալները կոդերից ստացված տվյալների հետ, որպեսզի մանրամասները պահպանվեն:

Նույնը կրկնվում է c1, u2 համար:

u2 = layers.UpSampling2D((2, 2))(c4)

u2 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(u2)

merge2 = layers.concatenate([c1, u2]) # Skip Connection

5. Ելք (Output Layer)

outputs = layers.Conv2D(1, (1, 1), activation='sigmoid')(c5)

- Վերջնական 1x1 կոնվոլյուցիոն շերտ sigmoid ակտիվացմամբ՝ գորշ (grayscale) դիմակ ստանալու համար:

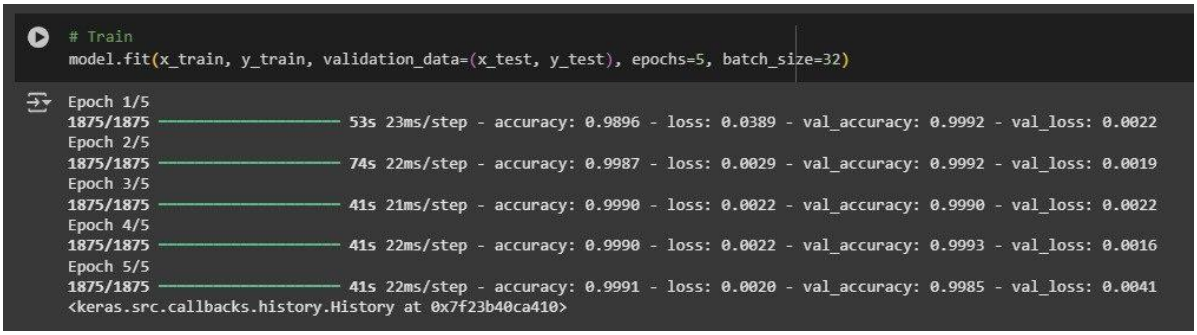
```
[ ]  # Compile
     model = unet_model()
     model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Մոդելի ստեղծում՝

- unet_model() ֆունկցիան կանչվում է՝ U-Net մոդելը ստեղծելու համար:

Մոդելի կոմպիլյացիա (compile)՝

- optimizer= adam → Օգտագործվում է Adam օպտիմալիզատորը, որը արագ և կայուն համակցում է Momentum և RMSprop:
- loss='binary_crossentropy → Պատճառն այն է, որ սեգմենտացիայի խնդիրներում հաճախ ամեն պիքսել երկարժեք (0 կամ 1) դասակարգում ունի (background vs object):
- metrics=['accuracy'] → Մոդելը գնահատվում է ճշգրտությամբ (accuracy):



Նկարում երևում է նեյրոնային ցանցի ուսուցման գործընթացի արդյունքները:

Ներկայացված է մոդելի ուսուցման կոդի հատված և դրա արդյունքները: Կոդը ցույց է տալիս, որ մոդելը ուսուցանվում է x_train և y_train տվյալների վրա, վալիդացիոն տվյալներն են x_test և y_test, ուսուցման էպոխների քանակը 5 է, իսկ batch size-ը 32:

Արդյունքներում երևում է յուրաքանչյուր էպոխի համար հետևյալ տեղեկատվությունը՝

- Էպոխի համարը (1/5, 2/5 և այլն)
- Կատարված քայլերի քանակը (1875/1875)
- Յուրաքանչյուր էպոխի կատարման ժամանակը (41-74 վայրկյան)
- Մեկ քայլի կատարման միջին ժամանակը (21-23մվ/քայլ)
- Ճշգրտությունը ուսուցման տվյալների վրա (accuracy՝ 0.9896-0.9991)
- Կորուստի ֆունկցիայի արժեքը ուսուցման տվյալների վրա (loss՝ 0.0020-0.0389)
- Ճշգրտությունը վալիդացիոն տվյալների վրա (val_accuracy՝ 0.9985-0.9993)
- Կորուստի ֆունկցիայի արժեքը վալիդացիոն տվյալների վրա (val_loss՝ 0.0016-0.0041)

Մոդելի ուսուցումը շատ հաջող է ընթացել՝ վերջնական արդյունքներում ցույց տալով 99.91% ճշգրտություն ուսուցման տվյալների վրա և 99.85% ճշգրտություն վալիդացիոն տվյալների վրա:

```
# Create segmentation masks (if > 0.5 => white/digit, else background)
y_train = (x_train > 0.5).astype(np.uint8)
y_test = (x_test > 0.5).astype(np.uint8)
```

Նկարում ցուցադրված է Google Colab-ում Google Drive-ը միացնելու գործընթացը:

Առաջին մասում երևում է կոդի հատված, որտեղ՝

- from google.colab import drive տողը ներմուծում է Colab-ի drive մոդուլը
- drive.mount('/content/drive') տողը կատարում է Google Drive-ի միացումը Colab-ի ֆայլային համակարգին '/content/drive' հասցեով

Երկրորդ մասում երևում է համակարգի պատասխանը՝ «Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).»

Սա նշանակում է, որ Google Drive-ն արդեն միացված է '/content/drive' հասցեով: Եթե ցանկանում եք կրկին միացնել, պետք է օգտագործեք force_remount=True պարամետրը, օրինակ՝ drive.mount("/content/drive", force_remount=True):

```
[ ] # Save
    model_path = "/content/drive/My Drive/GAN_project_model.h5"
    model.save(model_path)
    print(f"Model saved at {model_path}")

    WARNING:absl:You are saving your model as an HDF5 file via `mode
    Model saved at /content/drive/My Drive/GAN_project_model.h5
```

Նկարում ցուցադրված է Colab-ում մոդելի պահպանման կոդը և դրա արդյունքը:

Կոդի հատվածում՝

- model_path = "/content/drive/My Drive/GAN_project_model.h5" տողը սահմանում է ճանապարհը, որտեղ մոդելը կպահպանվի Google Drive-ում
- model.save(model_path) տողն օգտագործում է Keras-ի save մեթոդը մոդելը պահպանելու համար նշված հասցեում
- print(f"Model saved at {model_path}") տողը ցուցադրում է հաղորդագրություն պահպանման գործողության մասին

Արդյունքում երևում են երկու տող՝

1. Զգուշացում (WARNING), որը նշում է, որ մոդելը պահպանվում է որպես HDF5 ֆայլ՝ օգտագործելով model.save մեթոդը
2. Հաղորդագրություն, որ մոդելը հաջողությամբ պահպանվել է "/content/drive/My Drive/GAN_project_model.h5" հասցեում

HDF5 ֆորմատը (Hierarchical Data Format) հատուկ ֆայլային ֆորմատ է, որն օգտագործվում է մեծ տվյալների հավաքածուներ և օբյեկտներ պահելու համար, և հաճախ օգտագործվում է մեքենայական ուսուցման մոդելների պահպանման համար:

```
# Loading a new image

# Import libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load
image_path = "/content/1_3.png"
# image_path = '/content/drive/MyDrive/IMG_20240806_132109.jpg'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
print(image_path)
if image is None:
    print("Failed to load image. Please check the image path.")
else:
    print("Image loaded successfully.")

# Resize (model size 28x28)
image_resized = cv2.resize(image, (28, 28))

# Normalize and reshape
image_input = image_resized / 255.0  # Normalize to [0,1]
image_input = np.expand_dims(image_input, axis=-1)  # (28, 28) to (28, 28, 1) (channel)
image_input = np.expand_dims(image_input, axis=0)  # (28, 28, 1) to (1, 28, 28, 1) (batch)
```

```
/content/1_3.png
Image loaded successfully.
```

Նկարում երևում է պատկերի բեռնման, նախապատրաստման և մշակման պրոցեսը Python-ով:

Կոդի հատվածում՝

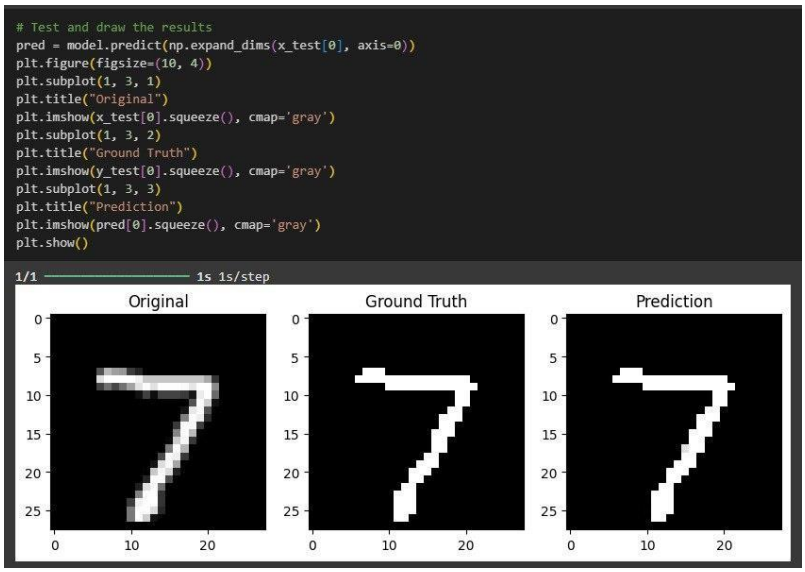
- Սկզբում ներմուծվում են անհրաժեշտ գրադարանները՝ cv2 (OpenCV պատկերների մշակման համար), numpy (թվային գործողությունների համար), և matplotlib.pyplot (պատկերների ցուցադրման համար)
- Որպես պատկերի հասցե սահմանվում է "/content/1_3.png"
- Մեկնաբանված է այլընտրանքային հասցե Google Drive-ում

- Պատկերը բեռնվում է որպես մոնոխրոմ (grayscale) կերպար cv2.imread ֆունկցիայի միջոցով
- Կոդը ստուգում է բեռնման հաջողությունը և տպում համապատասխան հաղորդագրություն
- Պատկերը փոքրացվում է 28×28 չափերի՝ համապատասխանեցնելով մոդելի պահանջներին
- Պատկերը նորմալացվում է՝ բաժանելով 255.0-ի, որպեսզի պիքսելների արժեքները դառնան 0-ից 1 միջակայքում
- np.expand_dims ֆունկցիայով ավելացվում են լրացուցիչ չափերը՝ ստանալով (28, 28, 1), այնուհետև (1, 28, 28, 1) ձևաչափը, որպեսզի համապատասխանի նեյրոնային ցանցի մուտքային շերտի պահանջներին

Արդյունքում երևում է, որ պատկերի հասցեն է "/content/1_3.png", և այն հաջողությամբ բեռնվել է ("Image loaded successfully."):

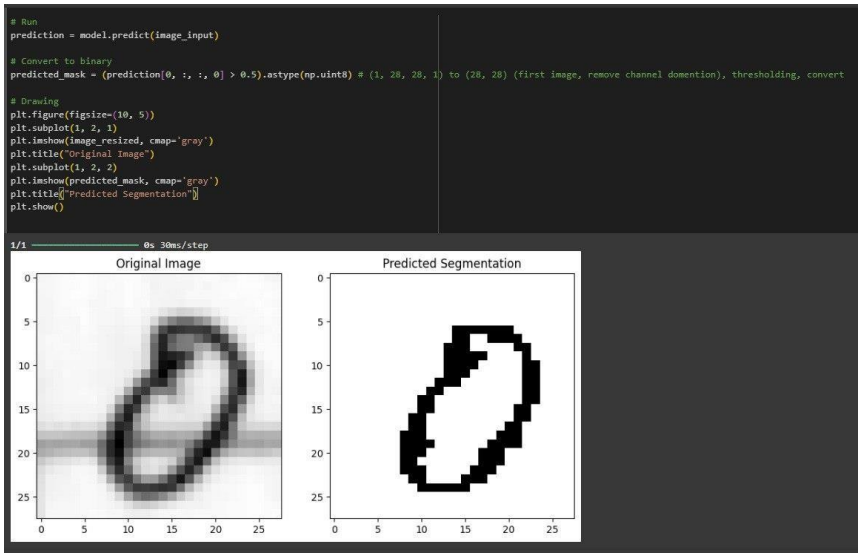
ԱՐԴՅՈՒՆՔՆԵՐ

Ուսուցման արդյունքները վիզուալ գնահատելու համար, նկ.2.3 -ում ներկայացված է օրինակ: Նկարում պատկերված է տվյալների բազայում առկա պատկերներից մեկը, մոդելին տրված նկարը որպես ճիշտ սեգմենտացիա և մոդելի կանխատեսումը:



Նկ 2.3 Մոդելի աշխատանքի արդյունք:

Նկ.2.4 -ում մոդելի արդյունքն է իրական տվյալի հիման վրա: Տրվել է «0» թվանշանը: Արտապատկերված են իրական նկարը սև-սպիտակ սպեկտորում, չափերի փոփոխությունից հետո և մոդելի կանխատեսումը:



Նկ 2.4 Պատկերի սեգմենտացիայի գործընթացը մեքենայական ուսուցման մոդելի միջոցով:

ԵԶՐԱԿԱՑՈՒԹՅՈՒՆ

Ավարտական աշխատանքի շրջանակներում մշակված U-Net մոդելը ցուցաբերել է բացառիկ արդյունավետություն MNIST տվյալների հավաքածուի ձեռագիր թվանշանների սեգմենտավորման խնդրում:

Ուսուցման գործընթացի ընթացքում, որը տևել է 5 epoch, մոդելը հասել է 99.91% ճշգրտության ուսուցման տվյալների և 99.85% ճշգրտության վալիդացիայի տվյալների վրա:

Կորուստի ցածր ցուցանիշները (ուսուցման համար ՝ 0.0020 և վալիդացիայի համար ՝ 0.0041) վկայում են մոդելի կայուն աշխատանքի մասին: Ուսուցման և վալիդացիայի ցուցանիշների միջև նվազագույն տարբերությունը ցույց է տալիս, որ մոդելը զերծ է գերհարմարման խնդրից, ինչը կարևոր է նոր տվյալների վրա այն կիրառելու համար:

Ստացված արդյունքները հաստատում են, որ ընտրված U-Net ճարտարապետությունը և MNIST տվյալների հավաքածուն օպտիմալ համադրություն են ձեռագիր թվանշանների սեգմենտավորման խնդրի լուծման համար: Մոդելի բարձր ճշգրտությունը թույլ է տալիս այն հաջողությամբ կիրառել գործնական խնդիրներում, ինչպիսիք են թվայնացված փաստաթղթերի թվանշանների ճանաչումը և մշակումը:

Մշակված մոդելը կարող է ծառայել որպես՝

- Ձեռագիր թվերի ճանաչման հիմնական գործիք
- OCR համակարգերի զարգացման մոդել
- Պատկերների սեգմենտացիայի էտալոն լուծման համար

ԳՐԱԿԱՆՈՒԹՅԱՆ ՑԱՆԿ

1) https://www.researchgate.net/publication/322303286_Handwritten_Digit_Segmentation_Is_it_still_necessary
2) https://paperswithcode.com/paper/continuous-offline-handwriting-recognition
3) https://paperswithcode.com/method/u-net
4) https://www.deeplearning.ai/courses/generative-adversarial-networks-gans-specialization/