بسمه تعالی

Islamic Azad University,
Science and Research Branch

**عنوان ارائه:**

# U-Net: Convolutional Networks for Biomedical Image Segmentation

**درس مربوطه:**

## یادگیری ماشینی در زیست پزشکی


**تهیه کنندگان:**

## شمیم نجفی و هلیا حاجی

**استاد درس:**

## سرکار خانم دکتر مهسا اخباری

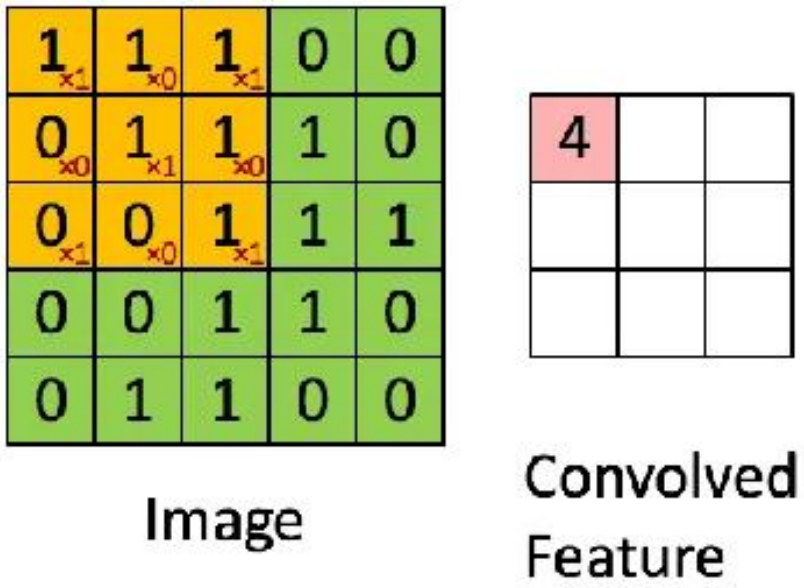

**آبان 1403**

## مروری بر مفهوم شبکه‌های عصبی مصنوعی (Convolution Neural Network)

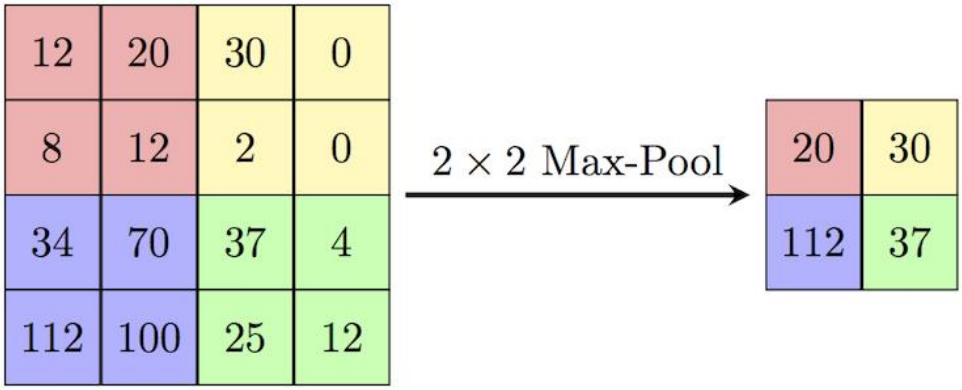
Convolution Neural Network یا به اختصار CNN که به آن‌ها شبکه‌های عصبی پیچشی هم گفته می‌شود، یکی از مدل‌های یادگیری عمیق هستند که اغلب برای بینایی کامپوتر و برای کارهایی مانند تقسیم بندی تصاویر، تشخیص اشیا و تقسیم بندی تصویر (Image segmentation) استفاده می‌شوند. CNN ها عمدتا برای یادگیری و استخراج اطلاعات از تصاویر استفاده می‌شوند و تا کنون در تجزیه و تحلیل داده‌های بصری عملکرد خیلی خوبی داشته اند.

## اجزای CNN

- **Convolutional Layers یا لایه های کانولوشنال**: CNN ها شامل مجموعه ای از فیلترهای قابل یادگیری (Kernel) هستند که با تصویر ورودی یا نقشه های ویژگی (feature maps) در هم آمیخته می‌شوند. هر یک از کرنل‌ها از ضرب و جمع عناصر استفاده می‌کنند و یک نقشه ویژگی درست می‌کنند. در حقیقت کرنل‌ها ویژگی‌ها یا الگوهای خاص را در تصویر برجسته می‌کنند. به طور مثال کرنل‌ها می‌توانند عناصر بصری مانند لبه‌ها، گوشه‌ها یا بافت را به تصویر بکشند.



- **Pooling Layers یا لایه‌های ادغام**: لایه‌های پولینگ یا ادغام لایه های کانولوشنال نمونه برداری شده نزولی(downsample) را با هم ادغام کرده و نقشه‌های ویژگی را ایجاد می‌کنند. ادغام کردن باعث می‌شود که ابعاد فضایی نقشه‌های ویژگی کمتر شود اما اطلاعات مهم آن حفظ شود. به این ترتیب پیچیدگی‌های محاسباتی در لایه‌های بعدی کم می‌شود و مدل در برابر نوسانات ورودی مقاومت بیشتری پیدا می‌کند. یکی از مهم‌ترین عملیاتی که برای ادغام استفاده می‌شود، max pooling است. در max pooling، بیشترین مقدار داده (معمولا intensity پیکسل)، در منطقه خاص تعیین شده برداشته می‌شود.




- **Activation Functions یا توابع فعال سازی**: به کمک توابع فعال سازی، مفهوم غیر خطی بودن را در مدل‌های CNN اعمال می‌کنیم. این توابع بر خروجی لایه های کانولوشنال یا لایه‌های پولینگ اعمال می‌شوند و به شبکه‌های عصبی مصنوعی اجازه می‌دهند که ارتباطات پیچیده را درک کند و تصمیمات غیر خطی بگیرد. یکی از توابع فعالسازی مهم و پرکاربرد Rectified Linear Unit یا ReLU است که سادگی و کارایی خوبی در رفع مشکل vanishing gradient یا گرادیان ناپدید شدن دارد.

- **Fully Connected Layers یا لایه‌های تمام متصل:** لایه‌های تمام متصل که به آن‌ها لایه‌های متراکم نیز گفته می‌شود، از ویژگی‌های بازیابی شده برای تکمیل عملیات تقسیم بندی یا رگرسیون نهایی استفاده می کنند. در لایه‌های تمام متصل، هر نورون از یک لایه، به تمامی نورون‌های لایه بعدی متصل

است و به شبکه اجازه می‌دهد که بازنمایی کلی را بیاموزد و با استفاده از ورودی بدست آمده از لایه‌های قبلی تصمیم گیری کند.

یک شبکه عصبی مصنوعی با مجموعه‌ای از لایه‌های کانولوشن شروع می‌شود که ویژگی‌های سطح پایین یا اولیه را استخراج می‌کنند. به دنبال لایه‌های کانولوشن، لایه‌های ادغام قرار دارند. هر چقدر که لایه‌های کانولوشن در شبکه عصبی عمیق‌تر شوند، ویژگی‌های مهم‌تری را استخراج می‌کنند. در نهایت هم در شبکه‌های عصبی بیچشی از یک یا چند لایه کامل (full layers) برای طبقه بندی یا رگرسیون نهایی استفاده می‌شود.

## معایب و کاستی‌های CNN

CNN های سنتی عموما برای تقسیم بندی تصاویری استفاده می‌شوند که یک برچسب واحد به کل تصویر ورودی داده می‌شود. همچنین این شبکه‌ها برای کارهای دقیق‌تری مانند تقسیم بندی های معنایی که در آن هر پیکسل از تصویر به کلاس‌ها یا مناطق مختلف تقسیم بندی می‌شود، مشکلاتی دارد.

## محدودیت‌های معماری سنتی CNN در تقسیم بندی تصویر:

**از بین رفتن اطلاعات مکانی:** CNN های سنتی از لایه‌های Pooling برای کاهش تدریجی ابعاد نقشه‌های ویژگی استفاده می‌کنند. اگرچه این کار باعث ثبت ویژگی‌های سطح بالا و مهم می‌شود، اطلاعات مکانی و تشخیص دقیق یا تقسیم بندی اشیا را در سطح پیکسل دشوار می‌کند.

**اندازه ثابت برای ورودی:** ساختار های CNN اغلب برای پردازش تصویرهایی با ابعاد خاص طراحی شده اند. این در حالی است که تقسیم بندی تصاویر یا segmentation باعث می‌شود تصاویر ابعاد مختلفی داشته باشند. در نتیجه مدیریت کردن و ساماندهی تصاویر ورودی در CNN های سنتی چالش برانگیز است.

**دقت محدود در مکان‌یابی:** شبکه‌های عصبی پیچشی سنتی (CNN) اغلب از لایه‌های کاملاً متصل در انتها استفاده می‌کنند تا یک بردار خروجی با اندازه ثابت برای تقسیم بندی ارائه دهند. از آنجا که این لایه‌ها اطلاعات مکانی را حفظ نمی‌کنند، نمی‌توانند به طور دقیق اجسام یا نواحی داخل تصویر را مکان‌یابی کنند.

برای رفع این مشکلات Fully Convolutional Networks (FCNs) معرفی شدند.

## شبکه‌های عصبی تمام پیچشی(Fully Convolutional Networks)

شبکه‌های عصبی تمام پیچشی که به اختصار به آن‌ها FCNs می‌گوییم، به عنوان یک راه حل برای تقسیم بندی معنادار و حفظ اطلاعات مکانی در سراسر شبکه عصبی ایجاد شده اند که منحصرا روی لایه‌های کانولوشنال کار می‌کنند. FCN ها محدودیت‌های ساختار کلاسیک CNN را در تقسیم بندی تصویرها برطرف می‌کند. FCN ها پیش بینی را پیکسل به پیکسل انجام می‌دهد. این نوع شبکه عصبی به هر پیکسل از تصویر ورودی یک برچسب یا یک کلاس اختصاص می‌دهد. FCN ها یک نقشه تقسیم بندی متراکم می‌سازند که پیش بینی را در سطح پیکسل انجام می‌دهد و نقشه ویژگی را می‌سازد.

Transposed convolutions که به deconvolutions یا upsampling layers (لایه‌های نمونه برداری یا نمونه افزایی) هم معروف هستند، برای جایگزینی لایه‌های تمام متصل بعد از CNN استفاده می شوند. به این ترتیب وضوح فضایی(یا رزولوشن) نقشه‌های ویژگی با Transposed convolutions افزایش پیدا می‌کند.

در طول نمونه افزایی، FCN ها معمولاً از اتصالات پرشی استفاده می‌کنند، لایه‌های خاصی را دور می‌زنند و مستقیماً نقشه‌های ویژگی سطح پایین را با لایه‌های بالاتر مرتبط می‌کنند. این روابط پرشی به حفظ جزئیات دقیق و اطلاعات متنی کمک می کند و دقت محلی سازی (localization) مناطق تقسیم شده را افزایش می دهد. FCN ها در تقسیم بندی‌های مختلف از جمله تقسیم بندی تصاویر پزشکی، تجزیه صحنه و تقسیم بندی نمونه بسیار کارآمد هستند. به این ترتیب شبکه عصبی هم برای تصاویر ورودی با اندازه‌های مختلف کارامد است و هم اطلاعات مکانی را در سراسر حفظ می‌کند.

## تقسیم بندی تصویر(Image Segmentation)

تقسیم‌بندی تصویر فرآیندی اساسی در بینایی کامپیوتری است که در آن یک تصویر به بخش‌های معنادار و جداگانه تقسیم می‌شود. بر خلاف طبقه‌بندی تصویر، که یک برچسب واحد را برای یک تصویر کامل ارائه می‌کند، تقسیم‌بندی برچسب‌هایی را به هر پیکسل یا گروهی از پیکسل‌ها اضافه می‌دهد و اساساً تصویر را به بخش‌های معنایی مهم تقسیم می‌کند.

تقسیم‌بندی تصویر از این نظر اهمیت دارد که درک دقیق‌تری از محتوای یک تصویر برای ما فراهم می‌کند. در نتیجه با تقسیم بندی تصویر به بخش‌های مختلف، ما می‌توانیم اطلاعات قابل توجهی را در مورد مرز اشیاء، شکل،

اندازه و روابط فضایی اجسام موجود در تصویر بدست بیاوریم. این مدل تجزیه و تحلیل دقیق تصاویر در کارهای مختلفی که با بینایی ماشین انجام می‌شود اهمیت بالایی دارد و باعث می‌شود پردازش‌ها در سطح بالاتر و پیشرفته‌تر انجام شود.


## آشنایی با معماری U-Net

کارهایی مانند حاشیه‌نویسی دستی و طبقه‌بندی پیکسلی، دارای معایب مختلفی هستند که آنها را برای کارهای تقسیم‌بندی دقیق و مؤثر ناکارآمد می‌کند. برای رفع این محدودیت ها، راه حل های پیشرفته تری مانند معماری U-Net توسعه یافته است. ابتدا معایب روش های قبلی و اینکه چرا U-Net برای غلبه بر این مسائل ایجاد شده است را بررسی می‌کنیم.

**حاشیه نویسی دستی(Manual Annotation)**: در حاشیه نویسی دستی ما باید مرزهای تصویر یا مناطق مد نظر را به صورت دستی ترسیم و علامت گذاری کنیم. این کار زمان بر، سخت و مستعد اشتباهات انسانی است. همچنین این کار برای داده‌های بزرگ مقیاس پذیر نیست و ایجاد یک توافق ثابت بین حاشیه نویس‌ها به خصوص برای تقسیم بندی‌های پیچیده دشوار است.

**طبقه‌بندی پیکسلی (Pixel-wise Classification)**: یکی دیگر از روش‌های رایج، طبقه‌بندی پیکسلی است که در آن هر پیکسل در یک تصویر به طور مستقل طبقه‌بندی می‌شود و معمولاً از الگوریتم‌هایی مانند

درخت‌های تصمیم، ماشین‌های بردار پشتیبان (SVM) یا جنگل‌های تصادفی(Random Forests) استفاده می‌کند. از سوی دیگر، طبقه‌بندی پیکسلی، برای به تصویر کشیدن تصویر کلی و وابستگی‌های بین پیکسل‌های اطراف ناتوان است و منجر به تقسیم‌بندی بیش از حد یا کم می‌شود. این روش تقسیم بندی نمی تواند روابط فضایی را در نظر بگیرد و اغلب در ارائه مرزهای دقیق اشیاء ناتوان است.

## غلبه بر چالش‌ها

معماری U-Net برای رسیدگی به این محدودیت‌ها و غلبه بر چالش‌های رویکردهای سنتی تقسیم‌بندی تصویر توسعه داده شد. رروش‌های برخورد U-Net با این مشکلات عبارت اند از:

**یادگیری سراسری (End-to-End Learning): U-Net** از یک تکنیک یادگیری سرتاسر استفاده می کند، به این معنی که یاد می گیرد تصاویر را مستقیماً از جفت های ورودی-خروجی و بدون حاشیه نویسی کاربر تقسیم بندی کند. U-Net می‌تواند به‌طور خودکار ویژگی‌های کلیدی را استخراج کرده و با آموزش بر روی یک مجموعه داده بزرگ برچسب‌گذاری شده، تقسیم‌بندی دقیق را اجرا کند، و نیاز به حاشیه‌نویسی دستی پر زحمت را از بین ببرد.

**معماری تمام کانولوشنال:U-Net** مبتنی بر یک معماری تمام کانولوشنال است. یعنی کاملاً از لایه های کانولوشن تشکیل شده است و هیچ لایه کاملاً متصلی را شامل نمی شود. این ویژگی، معماری U-Net را قادر می سازد تا روی تصاویر ورودی با هر اندازه ای کار کند و انعطاف پذیری و سازگاری آن را برای تقسیم بندهای مختلف و ورودی‌های متغیر افزایش دهد.

**معماری U شکل با اتصالات پرشی:** معماری این شبکه شامل یک مسیر رمزگذاری شده (مسیر فشرده سازی یا Encoding) و یک مسیر رمزگشایی شده (مسیرگسترش یا Decoding) است که به آن امکان می دهد اطلاعات جزئی و اطلاعات کلی (اصطلاحا جهانی) را جمع آوری کند. اتصالات پرشی، فاصله بین مسیرهای رمزگذاری و رمزگشایی را پر می کند، اطلاعات حیاتی را از لایه های قبلی حفظ می کند و امکان تقسیم بندی دقیق تر را فراهم می کند.

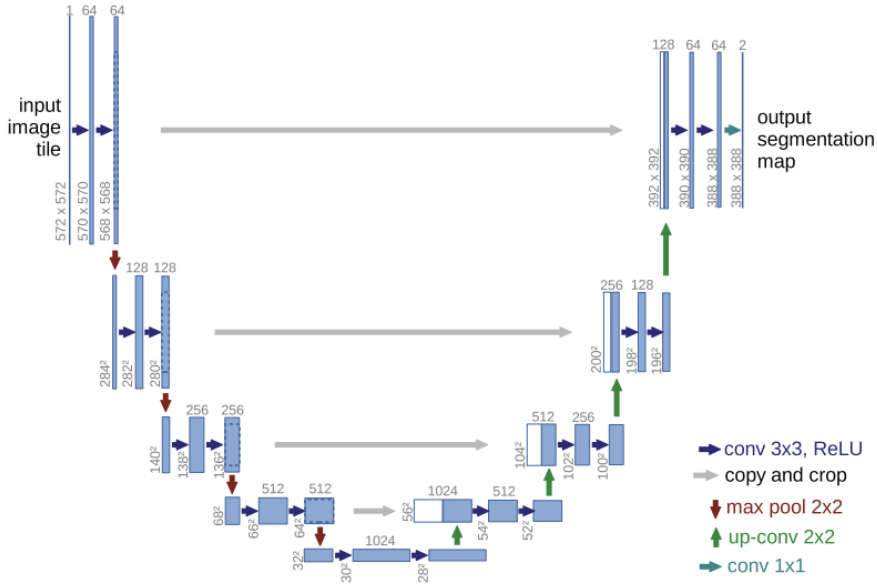
**اطلاعات متنی و محلی‌سازی:** اتصالات پرش در معماری U-Net نقشه‌های ویژگی را با چندمقیاس مختلف و از لایه‌های مختلف جمع آوری می‌کند و به شبکه اجازه می‌دهد اطلاعات متنی را جذب و جزئیات را در سطوح مختلف ثبت کند. این ادغام اطلاعات، دقت محلی‌سازی یا localization را بهبود می‌بخشد، و اجازه می‌دهد تا مرزهای اشیاء  به صورت دقیق مشخص شود و نتایج تقسیم‌بندی دقیق باشد.

**افزایش و منظم سازی داده ها (Data Augmentation and Regularization):** U-Net از تکنیک های تقویت و منظم سازی داده ها برای بهبود انعطاف پذیری و توانایی تعمیم خود در طول آموزش استفاده می کند. برای افزایش تنوع داده‌های آموزشی، از ایجاد تغییرات متعدد در تصاویر آموزشی مانند چرخش، برش، مقیاس‌بندی و تغییر شکل استفاده می‌شود. تکنیک‌های مختلف منظم‌سازی مانند حذف و نرمال‌سازی دسته‌ای از تطبیق بیش از حد داده‌ها جلوگیری می‌کند و عملکرد مدل را در داده‌های ناشناخته بهبود می‌بخشد.

## بررسی کلی ساختار U-Net

U-Net یک ساختار FCN هست که در تقسیم بندی تصاویر کاربرد دارد و اولین بار توسط Olaf Philipp Fischer, Ronneberger و Thomas Broxدر سال 2015 ارائه شد. U-Net معمولا بخاطر دقت یا Accuracy اش در تقسیم بندی تصاویر بکار می رود و تبدیل به گزینه ی پرطرفداری برای کاربرد های مختلف تصویر برداری پزشکی شده است. U-Net یک مسیر انکودر یا مسیر فشرده سازی را با یک مسیر دیکدر یا مسیر گسترش ترکیب می کند. این ساختار، وقتی آن را در یک دیاگرام به تصویر میکشیم، بخاطر شبیه بودن به یک ساختار U-شکل بدین نام نام گذاری شده است. بخاطر این

ساختار U-شکل، شبکه میتواند هم ویژگی های جزئی و هم اطلاعات کلی را ثبت کند که باعث نتایج دقیق در تقسیم بندی می شود.




## اجزای اساسی ساختار U-Net

- **مسیر فشرده سازی (Encoding Path)**: مسیر فشرده سازی U-Net شامل مسیر های کانولوشنال هست که به دنبال آن عملیات Max Pooling می باشد. این روش مشخصه های دارای رزولوشن بالا و سطح پایین را می گیرد و به تدریج ابعاد فضایی تصویر ورودی را کم می کند.
- **مسیر گسترش (Decoding Path)**: دیکانولوشن ها یا لایه های Upsampling، برای نمونه افزایی نقشه ی ویژگی مسیر انکودینگ در مسیر گسترش U-Net، استفاده می شوند. رزولوشن فضایی نقشه ی ویژگی در طی فاز نمونه افزایی ، افزایش می یابد و باعث می شود شبکه یک نقشه تقسیم بندی متراکم بازسازی کند.
- **اتصالات پرشی(Skip Connections)**: این قسمت برای اتصال لایه های متناظر از مسیر های کدگزاری به کدگشایی استفاده می شوند. اتصالات پرشی به شبکه این امکان را می دهد تا هردو داده های جزئی و کلی را جمع آوری کند. شبکه اطلاعات فضایی ضروری را نگه میدارد و دقت طبقه بندی را با ادغام نقشه ی ویژگی لایه های پیشین با آنهایی که در مسیر کدگشایی هستند، بهبود می دهد.

- **اتصال (Concatenation)**: اتصال معمولا برای تکمیل Skip Connection ها در U-Net بکار می رود. نقشه های ویژگی مسیر کدگذاری به نقشه های Upsample شده در حین فرآیند نمونه افزایی، متصل می شوند. این اتصال به شبکه این امکان را می دهد تا اطلاعات چند-مقیاسی را برای طبقه بندی مناسب، با استخراج زمینه های سطح بالا و ویژگی های سطح پایین، ترکیب کند.
- **لایه های کاملا کانولوشنال (Fully Convolutional Layers)**: U-Net شامل لایه های کانولوشنال فاقد لایه های تماما متصل می باشد. این ساختار پیچیده باعث می شود U-Net بتواند تصاویر با اندازه های نامحدود را در حالی که اطلاعات فضایی در سر تا سر شبکه را حفظ میکند، بکار ببرد، که باعث می شود شبکه با کار های طبقه بندی متعدد سازگار شود.

مسیر انکودینگ یا فشرده سازی، جزئی مهم از ساختار U-Net می باشد، چون مسئول استخراج ویژگی های سطح بالا از تصویر ورودی و در عین حال کوچک کردن ابعاد فضایی به صورت تدریجی، می باشد.

## لایه های کانولوشنال (Convolutional Layers)

فرآیند انکودینگ با مجموعه ای از لایه های کانولوشن شروع می شود. این لایه ها با بکارگیری فیلتر های قابل آموزش بر روی تصویر ورودی، اطلاعات را در مقیاس های متعددی استخراج می کند. این فیلتر ها در زمینه گیرنده های جزئی عمل می کنند و به شبکه اجازه می دهد الگو های فضایی و ویژگی های جزئی را پیدا کند. با هر لایه ی کانولوشن، عمق نقشه های ویژگی افزایش می یابد و باعث می شود شبکه طرح های پیچیده تری را بیاموزد.

## تابع فعال سازی (Activation Function)

به دنبال هر لایه ی کانولوشن، یک تابع فعال سازی مانند Rectified Linear Unit (ReLU) المان به المان اعمال می شود تا غیر خطی بودن را در شبکه ایجاد کند. تابع فعال سازی به شبکه در یادگیری همستگی های غیر خطی بین تصاویر ورودی و ویژگی های بازیابی شده، کمک می کند.

## لایه های ادغام (Pooling Layers)

لایه های ادغام بعد از لایه های کانولوشن برای کاهش ابعاد فضایی نقشه های ویژگی استفاده می شود. عملیاتی مانند max pooling، نقشه های ویژگی را به نواحی ای تقسیم می کند که همپوشانی نداشته باشند و فقط ماکزیمم مقدار را در داخل هر ناحیه نگه میدارد. این لایه رزولوشن فضایی را با نقشه های ویژگی downsampling کاهش میدهد و به شبکه اجازه می دهد تا داده های کلی تر و سطح بالاتر را بگیرد.

وظیفه ی مسیر انکودینگ گرفتن ویژگی ها در مقیاس ها و سطوح مختلف از ساده سازی به صورت سلسله مراتبی می باشد. فرآیند انکودینگ بر روی استخراج زمینه کلی و اطلاعات سطح بالا در حین کاهش ابعاد فضایی، تمرکز دارد.

## اتصالات پرشی (Skip Connections)

اتصالات پرشی در طراحی U-Net ضروری هستند، زیرا امکان انتقال اطلاعات بین مسیر های فشرده سازی (انکودینگ) و گسترش (دیکدینگ) را فراهم می کنند. این اتصالات برای حفظ اطلاعات فضایی و بهبود دقت طبقه بندی بسیار مهم هستند.

وجود اتصالات پرشی که سطوح مناسب مسیر انکودینگ را به مسیر دیکدینگ متصل می کنند، یکی از ویژگی های بارز ساختار U-Net است. این اتصالات در حفظ داده های کلیدی در طول فرآیند انکودینگ نقش حیاتی دارند.

نقشه های ویژگی از لایه های قبلی در مسیر انکودینگ، اطلاعات جزئی و دقیق را جمع آوری می کنند. این نقشه های ویژگی با نقشه های ویژگی upsample شده در مسیر دیکدینگ، با استفاده از اتصالات پرشی ، ادغام می شوند. این کار به شبکه اجازه میدهد تا داده های چند مقیاسی، ویژگی های سطح پایین و اطلاعات سطح بالا را در فرآیند تقسیم بندی وارد کند.

با حفظ اطلاعات فضایی از لایه های قبلی، U-Net قادر است اشیاء را به طور دقیق مکان یابی کرده و جزئیات ریز را در نتایج طبقه بندی نگه دارد. اتصالات پرشی U-Net به حل مشکل از دست رفتن اطلاعات ناشی از کاهش ابعاد کمک می کنند. این اتصالات امکان ادغام بهتر اطلاعات جزئی و کلی را فراهم کرده و عملکرد طبقه بندی را به طور کلی بهبود می بخشد.

به طور خلاصه، روش انکودینگ U-Net برای گرفتن ویژگی های سطح بالا و کاهش ابعاد فضایی تصویر ورودی مهم است. مسیر انکودینگ با استفاده از لایه های کانولوشن، توابع فعال سازی و لایه های ادغام تصاویر کلی به طور تدریجی استخراج می کند. با ادغام ویژگی های جزئی و کلی، معرفی اتصالات پرشی به حفظ اطلاعات فضایی مهم کمک می کند و نتایج تقسیم بندی مطمئنی را آسان تر می کند.

## مسیر دیکدینگ یا رمزگشایی در U-Net

یک بخش مهم از ساختار U-Net مسیر دیکدینگ است که به آن مسیر گسترش نیز گفته می شود. این بخش مسئول بالا بردن ابعاد نقشه های ویژگی مسیر انکودینگ و ساختن طرح نهایی طبقه بندی است.

## لایه های کانولوشن معکوس (Upsampling Layers)

برای افزایش رزولوشن فضایی نقشه های ویژگی، روش دیکدینگ U-Net شامل لایه های upsampling است که معمولا با استفاده از کانولوشن های معکوس یا دیکانولوشن ها انجام می شود. کانولوشن های معکوس ابعاد فضایی را به جای کاهش دادن، افزایش می دهند، و باعث نمونه افزایی می شود. با ساخت یک kernel پراکنده و اعمال بر روی نقشه ی ویژگی ورودی، کانولوشن های معکوس یاد می گیرند که نقشه های ویژگی را نمونه افزایی کنند. در این فرآیند، شبکه یاد می گیرد که فاصله های بین مکان های فضایی فعلی را پرکند و به این ترتیب رزولوشن نقشه های ویژگی را افزایش می دهد.

## اتصال (Concatenation)

نقشه های ویژگی از لایه های قبلی در مرحله دیکدینگ به نقشه های ویژگی نمونه افزایی متصل می شوند. این اتصال به شبکه امکان می دهد تا اطلاعات چند مقیاسی را برای تقسیم بندی صحیح تجمیع کند و از اطلاعات سطح بالا و ویژگی های سطح پایین بهره برداری کند. علاوه بر نمونه افزایی، مسیر دیکدینگ U-Net شامل اتصالات پرشی از سطوح مشابه در مسیر دیکدینگ نیز می باشد.

شبکه می تواند با اتصال نقشه های ویژگی از طریق اتصالات پرشی ، ویژگی های دقیق و جزئی که در مرحله انکودینگ از دست رفته اند را بازیابی و یکپارچه کند. این کار امکان مکان یابی و تعریف مرز دقیق تر اشیاء در طرح تقسیم بندی را فراهم می سازد.

فرآیند دیکدینگ در U-Net با نمونه افزایی تدریجی نقشه های ویژگی و اضافه کردن اتصالات پرشی، یک نقشه تقسیم بندی متراکم بازسازی می کند که با رزولوشن فضایی تصویر ورودی مطابقت دارد.

وظیفه ی مسیر دیکدینگ این است که اطلاعات فضایی از دست رفته در مسیر انکودینگ را بازیابی کرده و نتایج تقسیم بندی را بهبود ببخشد. این مسیر با ترکیب جزئیات سطح پایین حاصل از انکودینگ و زمینه سطح بالا که از لایه های نمونه افزایی به دست آمده است، یک طرح تقسیم بندی دقیق و کامل فراهم می کند.

U-Net می تواند با استفاده از کانولوشن های معکوس در فرآیند دیکدینگ، رزولوشن فضایی نقشه های ویژگی را افزایش دهد و آنها را به اندازه ی تصویر اصلی نمونه افزایی کند. کانولوشن های معکوس به شبکه کمک می کنند تا طرح تقسیم بندی متراکم و دقیق ایجاد کند، زیرا این لایه ها قادرند با یادگرفتن پر کردن فواصل و گسترش ابعاد فضایی، جزئیات را بازسازی کنند.

به طور خلاصه، فرآیند دیکدینگ در U-Net با افزایش رزولوشن فضایی نقشه های ویژگی از طریق لایه های نمونه افزایی و اتصالات پرشی، طرح تقسیم بندی را بازسازی می کند. کانولوشن های معکوس در این مرحله بسیارر مهم هستند، زیرا به شبکه امکان می دهند نقشه های ویژگی را نمونه افزایی کرده و یک طرح طبقه بندی دقیق ایجاد کنند که با تصویر ورودی اصلی مطابقت داشته باشد.

## مسیر های فشرده سازی و گسترش در U-Net ( Contracting and Expanding Paths)

ساختار U-Net از ساختار " انکودر-دیکدر" پیروی می کند، به طوری که مسیر فشرده سازی نشان دهنده ی مسیر انکودر و مسیر گسترش نشان دهنده دیکدر است. این طراحی شبیه به رمزگذاری اطلاعات به صورت فشرده و سپس رمزگشایی آن برای بازسازی داده ی اصلی است.

### مسیر فشره سازی (Encoder)

رمزگذار یا انکودر در U-Net مسیر فشرده سازی است که با کاهش تدریجی ابعاد فضایی، بافت و زمینه ی تصویر ورودی را استخراج و فشرده می کند. این روش شامل لایه های کانولوشنی است که به دنبال آن عملیات نمونه برداری نزولی (Downsampling)، مانند ماکس پولینگ، برای کاهش اندازه نقشه های ویژگی انجام می شود. مسیر فشرده سازی مسئول استخراج ویژگی های سطح بالا، یادگیری اطلاعات کلی و کاهش رزولوشن فضایی

است. این مسیر بر فشرده سازی و ساده سازی ورودی تمرکز دارد و اطلاعات مرتبط برای طبقه بندی را بطور کارآمد ثبت می کند.

### مسیر گسترش (Decoder)

رمزگشا یا دیکدر در U-Net مسیر گسترش است. با نمونه افزایی نقشه های ویژگی از مسیر فشرده سازی، اطلاعات فضایی را بازیابی کرده و نقشه ی نهایی طبقه بندی را تولید می کند. مسیر گسترش شامل لایه های نمونه افزایی که اغلب با کانولوشن های معکوس یا دکانولوشن ها برای افزایش رزولوشن فضایی نقشه های ویژگی انجام می شود. مسیر گسترش با استفاده از اتصالات پرشی، ابعاد فضایی اصلی را بازسازی می کند و نقشه های ویژگی نمونه افزایی شده را با نقشه های معادل از مسیر فشرده سازی ترکیب می کند. این روش به شبکه امکان می دهد ویژگی های دقیق را بازیابی کرده و اشیاء را به درستی مکان یابی کند.

طراحی U-Net با ترکیب مسیرهای فشرده سازی و گسترش، هم اطلاعات کلی و هم اطلاعات جزئی را ثبت می کند. مسیر فشرده سازی تصویر ورودی را به یک نمایش فشرده تبدیل می کند، که در نهایت توسط مسیر گسترش به یک نقشه ی تقسیم بندی دقیق تبدیل می شود. مسیر گسترش به رمزگشایی این نمایش فشرده شده به یک نقشه ی تقسیم بندی متراکم و دقیق می پردازد. این مسیر اطلاعات فضایی از دست رفته را بازسازی کرده و نتایج طبقه بندی را بهبود می بخشد. این ساختار انکودر-دیکدر امکان طبقه بندی دقیق را با استفاده از زمینه ی سطح بالا و اطلاعات فضایی جزئی فراهم می کند.

به طور خلاصه، مسیر های فشرده سازی و گسترش در U-Net به ساختار "انکودر-دیکدر" شباهت دارند. مسیر گسترش به عنوان رمزگشا عمل کرده و اطلاعات فضایی را بازیابی و نقشه ی نهایی طبقه بندی را تولید می کند. در مقابل، مسیر فشرده سازی به عنوان رمزگذار عمل کرده و بافت و زمینه را ثبت و تصویر ورودی را فشرده می کند. این ساختار به U-Net امکان می دهد تا اطلاعات را به طور موثری رمزگذاری و رمزگشایی کند و طبقه بندی دقیق و کامل تصویر را فراهم آورد.

## حفظ اطلاعات فضایی (Preserving Spatial Information)

برخی از اطلاعات فضایی ممکن است در مسیر انکودینگ از دست بروند، زیرا نقشه های ویژگی در فرآیندهای نمونه برداری نزولی ماکس پولینگ تحت تاثیر قرار می گیرند. این از دست رفتن اطلاعات می تواند به کاهش دقت مکان یابی و از بین رفتن جزئیات دقیق در طرح طبقه بندی منجر شود.

با ایجاد اتصالات مستقیم بین لایه های متناظر در فرآیند های انکودینگ و دیکدینگ، اتصالات پرشی به حل این مسئله کمک می کنند. اتصالات پرشی اطلاعات فضایی مهمی را که در فرآیند نمونه برداری نزولی ممکن است از دست برود، حفظ می کنند و اطلاعات مسیر رمزگذاری اجازه می دهند تا بدون نمونه برداری نزولی مستقیما به مسیر دیکدینگ منتقل شود.

## ترکیب اطلاعات چند مقیاسی (Multi-scale Information Fusion)

اتصالات پرشی امکان ترکیب اطلاعات چند مقیاسی از لایه های مختلف شبکه را فراهم می کنند. لایه های پایانی در فرآیند انکودینگ، بافت و اطلاعات معنایی سطح بالا را استخراج می کنند، در حالی که لایه های ابتدایی جزئیات دقیق و اطلاعات جزئی را ثبت می کنند.  با اتصال این نقشه های ویژگی از مسیر انکودینگ به لایه های متناظر در مسیر دیکدینگ، U-Net می تواند به طور موثر اطلاعات جزئی و کلی را ترکیب کند. این ترکیب اطلاعات چندمقیاسی بهبود دقت طبقه بندی را به همراه دارد. شبکه می تواند از داده های سطح پایین مسیر انکودینگ برای بهبود نتایج طبقه بندی در مسیر دیکدینگ استفاده کند و مکان یابی دقیق تر و تعریف مرز بهتر اشیاء را ممکن سازد.

## ترکیب اطلاعات سطح بالا و جزئیات سطح پایین (Combining High-Level Context and Low-Level Details)

اتصالات پرشی به مسیر دیکدینگ اجازه می دهند تا اطلاعات سطح بالا و جزئیات سطح پایین را ترکیب کند. نقشه های ویژگی ادغام شده از اتصالات پرشی شامل نقشه های ویژگی نمونه افزایی از مسیر دیکدینگ و نقشه های ویژگی مسیر انکودینگ هستند.

این ترکیب به شبکه این امکان را می دهد تا از اطلاعات سطح بالا که در مسیر دیکدینگ ثبت شده و ویژگی های دقیق که در مسیر انکودینگ گرفته شده، بهره برداری کند. شبکه می تواند اطلاعات را در اندازه های مختلف ادغام کند که این موضوع امکان طبقه بندی دقیق تر و با جزئیات بیشتر را فراهم می کند.

U-Net می تواند با استفاده از اطلاعات چند مقیاسی، حفظ جزئیات فضایی، و ترکیب اطلاعات سطح بالا با جزئیات سطح پایین از طریق افزودن اتصالات پرشی، عملکرد بهتری داشته باشد. در نتیجه، دقت طبقه بندی افزایش می یابد، مکان یابی اشیاء بهبود می یابد و اطلاعات دقیق در طرح تقسیم بندی حفظ می شود.

در نتیجه، اتصالات پرشی در U-Net برای حفظ اطلاعات فضایی، ادغام اطلاعات چندمقیاسی و افزایش دقت طبقه بندی حیاتی هستند. این اتصالات جریان مستقیم اطلاعات را بین مسیرهای انکودینگ و دیکدینگ فراهم می کنند و به شبکه امکان جمع آوری اطلاعات جزئی و کلی را می دهند که منجر به طبقه بندی تصویر دقیق تر و با جزئیات بیشتر شود.

## تابع خطا (Loss Function in U-Net)

تابع خطا، عددی را ارائه می دهد که نشان دهنده میزان خطا ناشی از پیش بینی های مدل است .به عبارت دیگر، این تابع مقادیر پیش بینی شده توسط مدل را با مقادیر واقعی مقایسه می کند. مدل تلاش می کند با استفاده از الگوریتم های بهینه سازی، پارامتر های خود را طوری تنظیم کند که مقادیر این تابع به حداقل برسد.

انتخاب یک تابع خطای مناسب هنگام آموزش U-Net و بهینه سازی پارامتر های آن برای وظایف تقسیم بندی تصویر بسیار مهم است. U-Net اغلب از توابع خطا مناسب برای طبقه بندی مانند ضریب دایس یا کراس انتروپی استفاده می کند.

- **ضریب خطا دایس (Dice Coefficient Loss)**

ضریب دایس یک معیار شباهت است که میزان هم پوشانی بین طرح های طبقه بندی پیش بینی شده و واقعی را محاسبه می کند. تابع خطا ضریب دایس، یا خطای دایس نرم (soft Dice loss)، با کم کردن یک از ضریب دایس محاسبه می شود. وقتی طرح پیش بینی شده و واقعی به خوبی با هم تطابق دارند، خطا کاهش می یابد و منجر به ضریب دایس بالاتری می شود.

تابع خطا ضریب دایس به ویژه برای مجموعه داده های نامتوازن که کلاس پس زمینه دارای پیکسل های زیادی است، موثر است. با جریمه کردن (penalizing) مثبت های کاذب و منفی های کاذب، این تابع شبکه را تشویق می کند تا نواحی پیش زمینه و پس زمینه را به طور دقیق طبقه بندی کند.

- **خطا کراس انتروپی (Cross-Entropy Loss)**

از تابع خطا کراس انتروپی در کار های طبقه بندی تصاویر استفاده می شود. این تابع میزان تفاوت بین احتمالات کلاس پیش بینی شده و برچسب های واقعی را اندازه گیری می کند. در تقسیم بندی تصویر، هر پیکسل به عنوان یک مسئله طبقه بندی مستقل در نظر گرفته می شود و تابع خطا کراس انتروپی به صورت پیکسل به پیکسل محاسبه می شود.

تابع خطا کراس انتروپی شبکه تشویق می کند تا احتمالات بالایی به برچسب های کلاس صحیح برای هر پیکسل اختصاص دهد. این تابع انحرافات از برچسب های واقعی را جریمه کرده و نتایج تقسیم بندی دقیق تر را افزایش می کند. این تابع خطا زمانی موثر است که کلاس های پیش زمینه و پس زمینه متوازن باشند یا زمانی که چندین کلاس در وظیفه تقسیم بندی دخیل باشند.

## مزایا U-Net

برخی از مزایا معماری U-Net عبارت اند از:

**کارآمد بودن:** U-Net برای تولید نقشه‌های تقسیم بندی دقیق به ویژه هنگام کار با تصاویری که وضوح یا کلاس‌های زیادی دارند بسیار کارآمد است.

**مدیریت خوب وظایف چند کلاسه**: این الگوریتم برای تقسیم بندی تصویر چند کلاسه مناسب است زیرا می‌تواند تعداد زیادی از کلاس‌ها را مدیریت کند و برای هر کلاس یک نقشه طبقه بندی در سطح پیگسل درست کند.

**استفاده کارآمد از داده های آموزشی**: U-Net از اتصالات پرش استفاده می‌کند که به مدل اجازه می‌دهد ویژگی‌های سطح بالا و سطح پایین را از تصاویر ورودی استخراج و ترکیب کند. این کار باعث می‌شود Unet در استفاده از داده‌های آموزشی کارآمئتر باشد و عملکرد بهتری داشته باشد.

## معایب U-Net

معایب این معماری عبارت اند از:

**زیاد بودن تعداد پارامترها**: Unet از تعداد زیادی لایه و اتصالات پرش تشکیل می‌شود که پارامترهای زیادی دارند. این مسئله باعث می‌شود Unet بیشتر در معرض overfitting باشد. احتمال این اتفاق به خصوص در هنگام کار با داده‌های کوچک بیشتر است.

**هزینه محاسباتی بالا**: Unet به دلیل داشتن اتصالات پرش به محاسبات زیادی نیاز دارد به همین دلیل هزینه‌های محاسباتی آن از معماری‌های دیگر بیشتر است.

**حساس به مقداردهی اولیه**: U-Net می تواند به مقداردهی اولیه پارامترهای مدل حساس باشد، زیرا اتصالات پرش می تواند هر گونه خطا را در وزن های اولیه تقویت کند. این می تواند آموزش U-Net را در مقایسه با معماری های دیگر دشوارتر کند.

## مثالی از کاربرد U-Net برای پردازش تصاویر پزشکی

اصلی‌ترین کاربرد الگوریتم Unet در حوزه پزشکی و برای تقسیم بندی تصاویر پزشکی است. زیرا یکی از مهم‌ترین آزمایش‌های تشخیصی که برای تشخیص بیماری‌هایی مانند سرطان، بیماری‌های قلبی، بیماری‌های ریوی، آلزایمر و... استفاده می‌شود، تصویر برداری پزشکی است. برای درک بهتر این موضوع ما از دیتا ست **Data Science Bowl 2018** به عنوان نمونه استفاده کردیم. این دیتاست در سال 2018 و در وبسایت Kaggle معرفی شد. این دیتاست حاوی تعداد زیادی تصویر از هسته سلول است. اهمیت بررسی هسته سلول‌ها این است که شناسایی هسته سلول و بررسی آن پایه بسیاری از تجزیه و تحلیل‌های پزشکی است. با بررسی هسته سلول‌ها و زیر نظر گرفتن واکنش آن‌ها به درمان‌های مختلف، محققین می‌توانند بهترین درمان را در سریع‌ترین زمان برای عارضه‌ها پیدا کنند و حتی داروهای جدید و موثرتری را برای درمان بیماری‌ها ارائه دهند.

2018 Data Science Bowl حاوی تعداد زیادی تصویر از هسته سلول تقسیم شده است که تحت شرایط مختلف تهیه شده است. این تصاویر در نوع سلول، بزرگنمایی و در روش تصویر برداری با هم متفاوت هستند. هدف از طراحی این دیتا ست به چالش کشیدن الگوریتم‌های یادگیری ماشین و الگوریتم‌های پردازش تصویر بوده است.

در این دیتا ست هر تصویر یک آیدی مشخص دارد و در پوشه ای با نام آیدی همان تصویر قرار گرفته است. همچنین این دیتا ست حاوی تعداد زیادی mask است. این ماسک‌ها تصاویری باینری از هسته‌های تقسیم شده سلول هستند و هیچ گونه همپوشانی با هم ندارند. ماسک‌ها در این دیتاست برای آموزش مدل‌های تقسیم‌بندی تصویر مانند U-Net استفاده می‌شوند. ماسک‌ها به مدل کمک می‌کنند تا یاد بگیرد که سلول‌ها را به‌طور دقیق از پس‌زمینه و از یکدیگر جدا کند.

## کد استفاده از U-Net برای آموزش ماشین با دیتاست Data Science Bowl 2018

```
import tensorflow as tf
import os
import numpy as np
from tqdm import tqdm
from skimage.io import imread, imshow
from skimage.transform import resize
import matplotlib.pyplot as plt
import random
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

IMG_WIDTH = 128
IMG_HEIGHT = 128
IMG_CHANNELS = 3

seed = 42
np.random.seed(seed)
```

در قدم اول ما کتابخانه‌های مورد نیاز مانند کتابخانه‌های tensorflow، numply، os و ... را فراخوانی کردیم. سپس مقدار واحدی را برای ابعاد تصاویر و تعداد کانال‌های تصاویر تعریف کردیم تا در پیش پردازش تصاویر از آن‌ها استفاده کنیم.

```
TRAIN_PATH = r"E:\\New folder\\stage1_train (1)"
TEST_PATH = r"E:\\New folder\\stage1_test (1)"

train_ids = next(os.walk(TRAIN_PATH))[1]
test_ids = next(os.walk(TEST_PATH))[1]
```

سپس مسیر ذخیره شده داده‌های آموزش و تست را به سیستم معرفی کردیم.

```
X_train = np.zeros((len(train_ids), IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS),
dtype=np.uint8)
Y_train = np.zeros((len(train_ids), IMG_HEIGHT, IMG_WIDTH, 1), dtype=bool)
```

در گام بعدی ما دو ارایه تعریف کردیم تا داده‌های مربوط به تصاویر آموزشی و ماسک‌ها در آن‌ها ذخیره کنیم.

```
for n, id_ in tqdm(enumerate(train_ids), total=len(train_ids)):
    path = os.path.join(TRAIN_PATH, id_)
```

```
    img_path = os.path.join(path, 'images', f'{id_}.png')
    mask_path = os.path.join(path, 'masks')
```

سپس از یک حلقه برای تعریف مسیر مربوط به تصاویر و ماسک‌ها استفاده کردیم.

```
    # Load and resize the image
    img = imread(img_path)[:,:,:IMG_CHANNELS]
    img = resize(img, (IMG_HEIGHT, IMG_WIDTH), mode='constant',
preserve_range=True)
    X_train[n] = img

    # Initialize an empty mask
    mask = np.zeros((IMG_HEIGHT, IMG_WIDTH, 1), dtype=bool)

    # Check if the mask directory exists and is not empty
    if os.path.exists(mask_path) and len(os.listdir(mask_path)) > 0:
        # Load and combine masks
        for mask_file in os.listdir(mask_path):
            mask_ = imread(os.path.join(mask_path, mask_file))
            mask_ = np.expand_dims(resize(mask_, (IMG_HEIGHT, IMG_WIDTH),
mode='constant', preserve_range=True), axis=-1)
            mask = np.maximum(mask, mask_)
    else:
        print(f"No mask files found in path: {mask_path}")
```

سپس تصاویر و ماسک‌ها را فراخوانی کردیم و ابعاد آن‌ها را اصلاح کردیم.

```
s = tf.keras.layers.Lambda(lambda x: x / 255)(inputs)
```

برای پیش پردازش داده‌ها نیز آن‌ها را نرمالیزه کردیم.

```
#Contraction path
c1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(s)
c1 = tf.keras.layers.Dropout(0.1)(c1)
c1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',
 kernel_initializer='he_normal', padding='same')(c1)
p1 = tf.keras.layers.MaxPooling2D((2, 2))(c1)

c2 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
```

```
kernel_initializer='he_normal', padding='same')(p1)
c2 = tf.keras.layers.Dropout(0.1)(c2)
c2 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c2)
p2 = tf.keras.layers.MaxPooling2D((2, 2))(c2)

c3 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p2)
c3 = tf.keras.layers.Dropout(0.2)(c3)
c3 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
 kernel_initializer='he_normal', padding='same')(c3)
p3 = tf.keras.layers.MaxPooling2D((2, 2))(c3)

c4 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p3)
c4 = tf.keras.layers.Dropout(0.2)(c4)
c4 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c4)
p4 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(c4)

c5 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p4)
c5 = tf.keras.layers.Dropout(0.3)(c5)
c5 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c5)
```

در قدم بعدی ما از چندین لایه کانولوشن و Pooling برای فشرده سازی داده‌ها و کاهش ابعاد فضایی تصویر استفاده کردیم تا ویژگی‌های مهم را استخراج کنیم.

```
u6 = tf.keras.layers.Conv2DTranspose(128, (2, 2), strides=(2, 2),
padding='same')(c5)
u6 = tf.keras.layers.concatenate([u6, c4])
c6 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal',
padding='same')(u6)
c6 = tf.keras.layers.Dropout(0.2)(c6)
c6 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal',
padding='same')(c6)

u7 = tf.keras.layers.Conv2DTranspose(64, (2, 2), strides=(2, 2),
padding='same')(c6)
u7 = tf.keras.layers.concatenate([u7, c3])
```

```
c7 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal',
padding='same')(u7)
c7 = tf.keras.layers.Dropout(0.2)(c7)
c7 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal',
padding='same')(c7)

u8 = tf.keras.layers.Conv2DTranspose(32, (2, 2), strides=(2, 2),
padding='same')(c7)
u8 = tf.keras.layers.concatenate([u8, c2])
c8 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_normal',
padding='same')(u8)
c8 = tf.keras.layers.Dropout(0.1)(c8)
c8 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_normal',
padding='same')(c8)

u9 = tf.keras.layers.Conv2DTranspose(16, (2, 2), strides=(2, 2),
padding='same')(c8)
u9 = tf.keras.layers.concatenate([u9, c1], axis=3)
c9 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',
kernel_initializer='he_normal',
padding='same')(u9)
c9 = tf.keras.layers.Dropout(0.1)(c9)
c9 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',
kernel_initializer='he_normal',
padding='same')(c9)
```

در گام بعدی که اصطلاحا به آن بخش گسترش یا Expansion Path می‌گوییم، ابعاد فضایی تصویر را بازسازی کردیم و اطلاعات مهم را حفظ کردیم.

```
outputs = tf.keras.layers.Conv2D(1, (1, 1), activation='sigmoid')(c9)
```

سپس از یک ماسک باینری در انتها استفاده کردیم که احتمال حضور یا عدم حضور سلول را در هر پیکسل از تصویر نشان می‌دهد.

```
model = tf.keras.Model(inputs=[inputs], outputs=[outputs])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

در قدم بعدی ما مدلمان را ساختیم. فرمول کلی مدل به شرح زیر است و پیش بینی می‌کند که مقدار پیش بینی شده چقدر به مقدار واقعی نزدیک است.

```
# Create a ModelCheckpoint callback to save the best model during training
checkpointer = tf.keras.callbacks.ModelCheckpoint('model_for_nuclei.keras',
                                                  verbose=1, save_best_only=True)

# Define other callbacks
callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=2, monitor='val_loss'),
    tf.keras.callbacks.TensorBoard(log_dir='logs')
]

# Fit the model
results = model.fit(X_train, Y_train, validation_split=0.1, batch_size=16,
epochs=25,
                    callbacks=[checkpointer] + callbacks)
```

سپس مدل را با epochs=25 آموزش دادیم و بهترین مدل را ذخیره کردیم.

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,
TensorBoard

# Define the log directory for TensorBoard
log_dir = r"E:\\New folder\\logs"
if not os.path.exists(log_dir):
    os.makedirs(log_dir)

# Create a ModelCheckpoint callback to save the best model during training
checkpointer = ModelCheckpoint('model_for_nuclei.keras',
                               verbose=1, save_best_only=True)

# Define other callbacks
callbacks = [
    EarlyStopping(patience=2, monitor='val_loss'),
    TensorBoard(log_dir=log_dir)
]

# Fit the model
results = model.fit(X_train, Y_train, validation_split=0.1, batch_size=16,
epochs=25,
                    callbacks=[checkpointer] + callbacks)
```

در نهایت مدل را روی داده‌های تست امتحان کردیم.. برای این کار ابتدا تصاویر داده‌های تست را آماده کردیم و ابعاد آن‌ها را مانند داده‌های آموزش اصلاح کردیم.

```
preds_train = model.predict(X_train[:int(X_train.shape[0]*0.9)], verbose=1)
preds_val = model.predict(X_train[int(X_train.shape[0]*0.9):], verbose=1)
preds_test = model.predict(X_test, verbose=1)
preds_train_t = (preds_train > 0.5).astype(np.uint8)
preds_val_t = (preds_val > 0.5).astype(np.uint8)
preds_test_t = (preds_test > 0.5).astype(np.uint8)
```

در مرحله بعد کار پیش بینی را روی داده‌های تست انجام دادیم و نتیجه پیش بینی‌ها را باینری کردیم.

```
test_loss, test_accuracy = model.evaluate(X_test, Y_test, verbose=1)

print(f'Test Loss: {test_loss}')
print(f'Test Accuracy: {test_accuracy}')
```

در آخر نیز به کمک دستور evaluate مقدار Accuracy و Loss را روی داده‌ها سنجیدیم.

## نتیجه گیری

تقسیم‌بندی تصویر در بینایی ماشین بسیار حیاتی است و این امکان را فراهم می‌کند که تصاویر به نواحی یا اشیای معنادار تقسیم شوند. روش‌های سنتی مانند حاشیه‌نویسی دستی و تقسیم‌بندی پیکسل‌به‌پیکسل معمولاً از نظر کارایی و دقت محدودیت‌هایی دارند. برای رفع این محدودیت‌ها، معماری U-Net به عنوان یک شبکه عصبی کاملاً کانولوشنال (FCN) توسعه یافته است که مسیر کدگذاری را برای استخراج ویژگی‌های سطح بالا با مسیر رمزگشایی برای تولید طرح های تقسیم‌بندی دقیق ترکیب می‌کند. U-Net از اتصالات پرشی استفاده می‌کند تا اطلاعات فضایی را حفظ کرده و انتشار ویژگی‌ها را تقویت کند که منجر به دقت بهتر در تقسیم‌بندی می‌شود. این معماری در زمینه‌های مختلفی مانند تصویربرداری پزشکی، تحلیل تصاویر ماهواره‌ای و کنترل کیفیت صنعتی موفقیت‌های چشمگیری کسب کرده و در مسابقات مختلف به رسمیت شناخته شده است.

## منابع


1. https://www.analyticsvidhya.com/blog/2023/08/unet-architecture-mastering-image-segmentation/#:~:text=UNET%20is%20frequently%20utilized%20for,path%20called%20the%20expanding%20path
2. https://data.broadinstitute.org/bbbc/BBBC038/GIMP%20strategy.pdf
3. https://github.com/kamalkraj/DATA-SCIENCE-BOWL-2018/tree/master/data
4. https://www.kaggle.com/competitions/data-science-bowl-2018/data
5. https://www.analyticsvidhya.com/blog/2023/01/top-8-interview-questions-on-unet-architecture/