به‌نام خداوند جان‌آفرین

پاسخ سوالات فاز اول پروژه درس هوش مصنوعی

اعضای گروه(به‌ترتیب حروف الفبا)

پوریا رحمانی ۴۰۲۱۱۱۴۱۸

نیما ملایی ۴۰۲۱۰۶۵۵۳

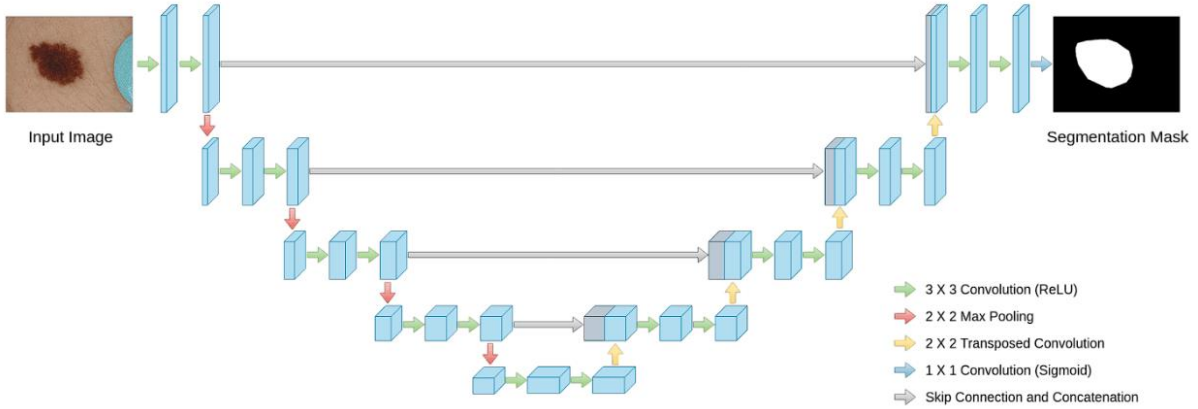
محمدرضا منعمیان ۴۰۲۱۰۶۶۰۴

پاسخ پرسش اول

ساختار کامل این معماری را برای مدلی که یک تصویر RGB ورودی می‌گیرد و یک تصویر تک کاناله باینری خروجی می‌دهد شرح می‌دهیم(این نوع ورودی و خروجی با آن‌چه در پروژه استفاده شد، مطابقت دارد، البته تغییر نوع ورودی و خروجی تاثیر زیادی در ساختار Unet نخواهد داشت)

به‌طور کلی این معماری سه بخش اصلی دارد. یک decode path ، یک encode path و یک bottleneck .

encode path و decode path ساختاری چندلایه دارند (تعداد لایه‌های decode و encode با هم برابرند) که در پروژه باتوجه به خواسته صورت پروژه، از چهار لایه استفاده کردیم. تصویر کلی ساختار Unet را در زیر می‌بینید:



هر لایه در encode path (سمت چپ ساختار نشان‌داده شده در تصویر) شامل یک convolution block و یک MaxPooling می‌شود. در هر convolution block دو عملیات convolution که سایز kernel هر دو ۳×۳ است داریم. بعد هر convolution یک BatchNoramlization و Relu داریم.

Convolution اول در هر convolution block ، تصویری با تعداد کانال in_channel می‌گیرد و تعداد کانال‌های خرورجی آن (که درواقع همان تعداد feature هاست) به‌اندازه out_channel است که در لایه اول in_channel برابر سه است (زیرا تصاویر RGB ، سه کانال دارند) و out_channel طبق تصویری که در مقاله اصلی وجود دارد ۶۴ است (که البته می‌تواند بسته به نیاز طراح مدل تفاوت کند) و در لایه‌های بعدی out_channel دوبرابر in_channel است (مطابق آنچه در مقاله اصلی هست و در پروژه هم از آن تقلید شده)

در convolution ها از stride برابر یک استفاده شده. در مقاله اصلی padding در convolution ها برابر صفر است و درنتیجه ابعاد هر کانال از تصویر پس از هر convolution کاهش می‌یابد، اما ما در پروژه به دلیلی که توضیح خواهیم داد، از padding = 1 استفاده کرده‌ایم که ابعاد هر کانال از تصویر پس از convolution ثابت بماند.

منظور از batchnormalization این است که پس از انجام convolution روی داده‌های batchای که داریم از آن استفاده می‌کنیم ، درایه‌های تصاویر خروجی convolution را نرمال می‌کنیم به گونه‌ای که میانگین و واریانس درایه‌های هر کانال در batch های متفاوت صفر و واریانس آن‌ها یک شود. یکی از فواید این کار این است که گرادیان‌ها پایدارتر بشوند و خیلی بزرگ یا کوچک نشوند و از مشکلاتی مثل gradiant vanishing جلوگیری شود.
پس از normalization تابع Relu روی تصاویر اعمال می‌شود. این تابع، پیکسل‌های منفی را به صفر تبدیل می‌کند.

عملیات MaxPooling ای که استفاده می‌کنیم، ۲×۲ است. در این عملیات، تعداد کانال‌ها ثابت می‌ماند ولی ابعاد هر کانال نصف می‌شود. این عملیات کمک می‌کند با convolution در لایه‌های بعدی، ویژگی‌های global تری استخراج کنیم. شیوه این عملیات به این صورت است که ورودی را به مربع‌های ۲×۲ تقسیم می‌کنیم و از هر مربع ماکزیمم مقادیر آن را به عنوان یک پیکسل انتخاب می‌کنیم تا تصویری با نصف ابعاد تصویر اولیه بدست آوریم.

Bottleneck هم درواقع یک convolution block با توضیحات مشابه است که تعداد کانال‌ها را دوبرابر می‌کند.

در هر لایه از decoder path ، روی خروجی لایه decoder قبلی(یا برای پایین ترین لایه، خروجی bottlenck) یک upsampling انجام می‌دهیم و سپس تصویر حاصل را با خروجی لایه encoder متناظر concat می‌کنیم و سپس تصویر را از یک convolution block رد می‌کنیم.

در عملیات upsampling که انجام می‌دهیم ، تعداد کانال‌ها نصف ولی ابعاد هر کانال، دوبرابر می‌شود. شیوه عملیات(با استفاده از تابع convTranspose2D که در torch.nn وجود دارد) به این‌گونه‌است:

ما از out_channel ای برابر نصف in_channel استفاده می‌کنیم ، stride ای برابر دو ، یک kernel دو در دو، padding و output padding برابر صفر. هر درایه از ورودی را در kernel ضرب می‌کنیم تا یک ماتریس ۲×۲ پدید آید(باید روی کانال‌های مختلف ورودی جمع ببندیم) و این ماتریس‌های دو در دو را در کنار هم می‌چینیم.(به‌صورت دقیق‌تر در فیلم به صورت تصویری نشان می‌دهیم)

تنها نکته‌ای که در concat کردن خروجی لایه encoder متناظر به خروجی upsampling باید مورد توجه قرار بگیرد این است که اگر مانند مقاله از padding = 0 در convolution ها استفاده کنیم، ابعاد هر تک کانال پس از convolution ها کاهش می‌یابد، درنتیجه ابعاد هر تک‌کانال از خروجی encoder از تصویری که در لایه decoder وجود دارد بیشتر است و برای رفع این مشکل، تصویر لایه encoder از وسط crop شده‌است. اما در پروژه ما از padding = 1 استفاده کردیم تا ابعاد هر کانال، قبل و بعد از convolution ثابت بماند و دچار این مشکل نشویم.

ساختار convolution block ها در decoder path کاملا مشابه encoder path است، با این تفاوت که در decoder path ، تعداد کانال‌های خروجی به‌جای دوبرابر، نصف تعداد کانال‌های ورودی است.

در نهایت نیز از آنجا که تصویر تک کاناله است، از یک convolution با kernel به سایز ۳×۳ و با stide و padding برابر یک(برای تغییر نکردن ابعاد هر تک کانال) و تعداد کانال خروجی یک استفاده می‌کنیم و سپس برای map کردن پیکسل ها

به بازه‌ی [0,1] روی پیکسل‌های تصویر تابع sigmoid اعمال می‌کنیم. درنهایت برای بدست آوردن پیش‌بینی، می‌توانیم پیکسل‌هایی از خروجی که مقدارشان بیشتر از 0.5 است را به 1 و بقیه را به صفر تبدیل کنیم.

پاسخ پرسش دوم

Loss function استفاده شده در پروژه برابر iou_loss + dice_loss + bce_loss است.

iou_loss برابر یک منهای iou_score است و اگر تصویر prediction باینری بود، iou_score برابر تعداد پیکسل‌های یک مشترک در تصویر prediction و تصویر پاسخ واقعی تقسیم بر تعداد پیکسل‌های یک در اجتماع دو تصویر (یعنی تعداد پیکسل‌هایی که در حداقل یکی از دو تصویر برابر یک هستند) می‌شد. طبعا هرچه این مقدار که کمتر از یک است، به یک نزدیک‌تر باشد، دقت ما بیشتر است، زیرا اگر پیش‌بینی کاملا درست باشد، باید تعداد پیکسل‌های یک در اجتماع و اشتراک دو تصویر یکسان باشد. بنابراین، هر چه یک منهای این عدد کمتر باشد بهتر است. برای اینکه از تقسیم بر صفر جلوگیری شود، به صورت و مخرج تقسیم، یک عدد بسیار کوچک مثبت که در کد با نام smooth مشخص است، اضافه شده‌است.(درواقع iou score برابر $\frac{TP}{TP+FN+FP}$ می‌شد)

dice loss برابر یک منهای dice score است و اگر تصویر prediction باینری بود، dice score برابر دوبرابر تعداد پیکسل‌هایی که در هر دوی prediction و پاسخ واقعی یک هستند تقسیم بر حاصل جمع تعداد پیکسل‌های یک در دو تصویر می‌شد. قاعدتا این عدد هم کمتر از یک است و هرچه بزرگتر باشد بهتر است و اگر پیش‌بینی دقیق باشد، این عدد دقیقا برابر یک خواهد بود. بنابراین یک منهای این عدد، هرچه کمتر باشد بهتر است و می‌تواند به عنوان loss استفاده شود.(درواقع dice score برابر $\frac{2\times TP}{2\times TP+FN+FP}$ می‌شد)

حالا که تصویرهای prediction ما پیکسل‌هایی بین صفر و یک دارند، اشتراک با ضرب داخلی دو تصویر محاسبه می‌شود و اجتماع از جمع درایه‌های دو تصویر منهای اشتراک محاسبه شده.

BCE loss هم، برحسب همان cross entropy است.

پاسخ پرسش سوم

قاعدتا این پروژه به‌خودی خود برای پاسخ به این سوال کمکی نمی‌کند و از web search با LLM ها به پاسخی نسبی می‌رسیم!

از Unet در تسک های segmentation دیگری مثل تشخیص تومورها یا ارگان‌های مختلف در تصاویر پزشکی مثل MRI ، تشخیص بیماری گیاهان، تشخیص ترک‌ها و عیب‌ها در محصولات صنعتی در کارخانه‌ها، تشخیص دیواره‌های سلول و ... در تصاویر میکروسکوپی دارد.

Dice loss برای وقتی که کلاس هدف در segmentation کم باشد (مثلا در تشخیص جاده‌ها، تعداد 1 های هر تصویر نسبت‌به صفرهای آن بسیار کمند) مناسب است (به‌خاطر همان ضریب دو در 2*TP که به TP ارج بیشتری می‌نهد!!) و

بنابراین در کارهایی مثل segmentation تصاویر پزشکی بسیار مناسب است. مثلا مقاله‌ای وجود دارد که از cross entropy + Dice loss استفاده کرده است.

از focal loss هم برای زمانی که برخی از کلاس‌ها ساده‌هستند و برخی سخت‌تر استفاده می‌شود(به عبارتی اگر مدل زمانی که یک پیکسل یک عدد خاص است خیلی confident است و در مورد چند مقدار دیگر ضعف دارد، با افزایش مقدار loss برای آن کلاس‌ها که در آن‌ها ضعف دارد، روی بهبود در آن‌ها تمرکز می‌کند)

اگر به‌جای آن convolution نهایی در Unet، یک fully connected layer جایگزین کنیم، می‌توانیم از Unet در classification هم استفاده کنیم و در آن صورت از loss هایی مثل CCE استفاده می‌کنیم(CCE شبیه cross entropy عادی است و برابر جمع لگاریتم احتمال‌های کلاس‌های درست در جفت‌های داده-برچسب است) و اگر activation function نهایی را حذف کنیم می‌توانیم از آن در task های regression هم استفاده کنیم که در آن صورت از loss هایی مثل MSE استفاده خواهیم کرد.

پاسخ پرسش چهارم

البته در فایل ReadMe گیت‌هاب نیز نقش skip connection را توضیح داده‌ایم. در فرایند encoding و در max pooling ها، ابعاد کانال‌ها را کوچک کرده‌ایم تا ویژگی‌های global را استخراج کنیم و مثلا بفهمیم به‌طور کلی جاده‌ها در چه جاهایی ظاهر می‌شوند و ... اما با بدست آوردن ویژگی‌های کلی، ساختارهای کوچک، لبه‌ها و ... که ویژگی‌های local تری هستند را از دست داده‌ایم و در فرایند decoding که دوباره این تصاویر را بزرگ می‌کنیم هم صرفا با استفاده از upsampling نمی‌توانیم این ویژگی‌ها را دوباره بدست آوریم. اما این ویژگی‌ها در ماتریس‌های خروجی لایه‌های encoder هستند. بنابراین می‌توانیم با concat کردن این خروجی به ماتریس پس از upsampling در لایه decoder ، این ویژگی‌ها را دوباره بدست بیاوریم.