دانشکده مهندسی برق و کامپیوتر

گروه آموزشی مهندسی کامپیوتر

گزارش پروژه یادگیری ماشین

پیاده سازی شبکه عصبی VGG

استاد درس:

دکتر سلیمی

دانشجو:

مریم رضوانی

زمستان 1402

تقدیر و تشکر:

در ابتدا، لازم می‌دانم تا از زحمات استاد گران‌قدر جناب دکتر سلیمی بابت زحمات بی‌دریغشان تشکر نمایم و از خداوند متعال توفیق روزافزون برایشان خواستارم.

با سپاس از لطف شما

چکیده:

گزارشی که پیش روی شماست، گزارشی است که اینجانب برای گزارش پایانی پروژه یادگیری ماشین نوشته‌ام. در این گزارش ابتدا به معرفی شبکه عصبی عمیقVGG پرداخته شده و سپس بصورت جداگانه به معرفی هریک از لایه‌های این شبکه و معماری آن و بهینه سازی آن با استفاده از لایه‌کانولوشن ۳*۳ پرداخته شده و شیوه آموزش این شبکه نیز آورده شده است در پایان پیاده سازی آن و نتیجه پیاده‌سازی و تغییر دیتاست آن(تغییراتی که در این گزارش انجام داده‌ام)ارائه شده است.

فهرست مطالب

فصل اول: معرفی شبکه عصبی عمیقvgg

۱-۱ شبکه عصبی (VGG)

شبکه عصبی عمیق VGG یا به اصطلاح لاتین VGG Deep Neural Network یک شبکه عصبی کانولوشنی است که توسط محققانی بنام K. Simonyan و A. Zisserman از دانشگاه آکسفورد در مقاله "**Very Deep Convolutional Networks for Large-Scale Image Recognition**" پیشنهاد شد.

این شبکه جزو اولین شبکه‌هایی بود که درسال ۲۰۱۴ به دلیل دقت بالایی که در مسابقات ImageNet داشت انقلاب بزرگی به سهم خود در ظهور شبکه‌های عصبی عمیق بوجود آورد. این شبکه نسخه‌های مختلفی دارد که این نسخه‌ها از ۱۱ لایه تا ۱۹ لایه دارند این شبکه توجه اصلی روی تأثیر عمق شبکه روی دقت آن است.

۱-۱-۱ معماری شبکه عصبی VGG

در این شبکه برخلاف شبکه عصبی قبلی(AlexNet) از لایه‌های کانولوشنی ۳*۳ بسیاری استفاده شده که تاثیر بسیار زیادی در دقت شبکه بر روی دیتاست شامل تصاویر با ابعاد بزرگ(باتوجه به افزایش تعداد لایه‌های شبکه) دارد.

روشی که در این مقاله استفاده شده، استفاده از لایه‌های کانولوشنی ۳*۳ به جای استفاده از کانولوشن‌های ۵*۵ می‌باشد که محدوده دریافت کوچک‌تری را نسبت به شبکه های پیش از خود دارند اما می‌توان نشان داد که دو فیلتر 3*3 متوالی بدون لایه pooling میان آن‌ها به نحوی که ابعاد ورودی حفظ شود، محدوده‌ی دریافتی موثری برابر با یک فیلتر 5*5 خواهد داشت و در صورت استفاده از سه فیلتر 3*3 متوالی، محدوده دریافتی موثری معادل با فیلتر 7*7 حاصل می‌شود.

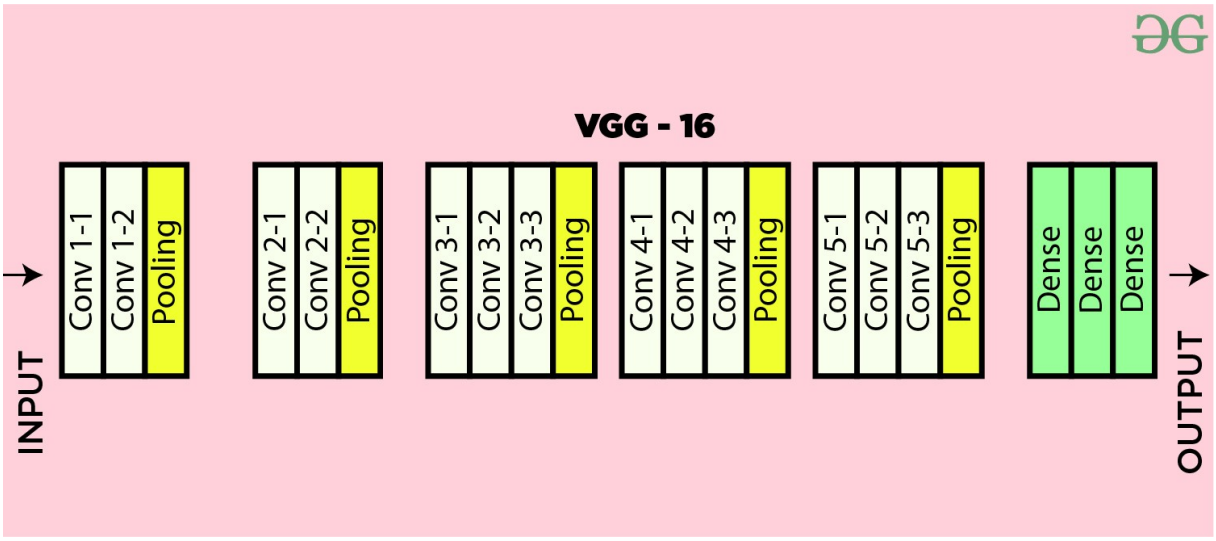
اینکار به نوبه‌ی خود نقاط مثبت خوبی برای دقت شبکه دارد که اولین مزیت آن :

- افزایش میزان غیرخطی بودن بدلیل استفاده از لایه های کانولوشنی متوالی است که باعث افزایش قدرت تصمیم تمایز شبکه خواهد شد.
- با توجه به افزایش لایه‌های کانولوشنی به منظور بهبود سرعت یادگیری شبکه، بهتر است که تعداد پارامترها افزایش چندانی نداشته باشد. استفاده از 3 لایه کانولوشنی 3*3 با تعداد C کانال، 27C^2 پارامتر را به شبکه اضافه خواهد کرد، در حالی که استفاده از یک فیلتر 7*7 با محدوده‌ی دریافت موثر معادل، 49C^2 پارامتر خواهد داشت.

مقدار stride و padding برای این لایه‌ها به گونه‌ای انتخاب شده‌است که ابعاد تصویر ورودی هر لایه، در خروجی آن حفظ شود. تابع فعال‌سازی استفاده شده در این لایه‌ها، ReLU می‌باشد که نسبت به بقیه توابع فعالسازی سرعت همگرایی بالاتری دارد و فاقد ناحیه اشباع به ازای ورودی مثبت است. در لایه آخر از تابع فعالسازی softmax استفاده شده است استفاده از این تابع فعال‌سازی این امکان را فراهم می‌سازد تا شبکه توزیعی احتمالی را برای کلاس‌های مختلف دیتاست ایجاد کند، به این صورت که کلاس شئ موجود در تصویر دارای بیشترین احتمال خواهد بود.

همچنین دارای ۵ لایه max pooling است که به دنبال بعضی از لایه‌های کانولوشنی قرار گرفته‌است، پنجره‌ی این لایه‌ها 2*2 و با stride برابر با 2 انتخاب شده‌است

در انتهای این شبکه از ۳ لایه fully connected استفاده شده که دو لایه اول دارای ۴۰۹۶ نورون و لایه آخر ۱۰۰۰ نورون(۱۰۰۰ کلاس).



شکل۱-۱-۱ ساختار شبکه عصبی vgg-16

فصل دوم: پیاده‌سازی شبکه عصبی vgg-16

2-1 پیاده‌سازی شبکه عصبی vggبا پایتون

عنوان: پیاده سازی مدل VGG16 برای طبقه بندی تصاویر در مجموعه داده CIFAR-100

موضوع: یادگیری عمیق، بینایی ماشین

شرح:

این کد با استفاده از کتابخانه پایتورچ در پایتون نوشته شده است.

۱. کتابخانه ها و تنظیمات اولیه:

- کتابخانه های مورد نیاز مانند numpy, torch و torchvision (از آن در transform استفاده شده) استفاده می شوند.
- ترنسفورم همان پیش پردازش داده می‌شود که قبل آموزش داده‌ها را آماده می‌کنیم.
- نوع پردازنده (CPU یا GPU) با بررسی دسترس بودن cuda انتخاب می شود.

۲. تابع data_loader:

این تابع برای این است که یک مجموعه داده حجیم را نمی‌توان مستقیم وارد حافظه کرد باید بصورت دسته ای این‌ها را وارد کند و لود شود که از تابع data_loaderاستفاده کردیم.

- این تابع برای بارگیری و پیش پردازش داده های CIFAR-100 استفاده می شود.
- پارامترهای ورودی شامل مسیر دایرکتوری داده ها (data_dir)، اندازه دسته (batch_size)، عدد تصادفی (random_seed)، درصد داده های اعتبارسنجی (valid_size)، پرچم تصادفی کردن(shuffle)برای جلوگیری از بیش برازش و پرچم تست (test) است.
- تابع بر اساس مقدار test خروجی متفاوتی دارد:
- در حالت تست، فقط داده های تست بارگیری و پیش پردازش می شوند.
- در حالت آموزش، داده ها به دو بخش تقسیم می شوند (80% برای آموزش و 20% برای اعتبارسنجی) و نمونه گیری تصادفی برای انتخاب تصاویر از هر بخش انجام می شود.
- تصاویر به اندازه 227x227 تغییر اندازه داده می‌شوند.
- تصاویر با استفاده از میانگین و انحراف استاندارد مشخص شده، نرمال می‌شوند.
- پیش پردازش تصاویر شامل تغییر اندازه، تبدیل به Tensor و نرمال سازی است.

۳. بارگیری دیتاست:

- تابع data_loader برای بارگیری و پیش پردازش مجموعه داده CIFAR-100 فراخوانی می شود.
- دیتاست آموزشی و اعتبارسنجی به ترتیب در train_loader و valid_loader ذخیره می شوند.
- یک test_loader جداگانه نیز برای مجموعه داده تست ایجاد می شود.

۴. شبکه VGG16:

- این کلاس معماری مدل VGG16 را با ۱۳ لایه کانولوشنال تعریف می‌کند.
- مدل از بلوک‌های متوالی از لایه‌های کانولوشنال، Batch Normalization و ReLU تشکیل شده است.
- همچنین از لایه‌های Max Pooling برای کاهش ابعاد ویژگی استفاده می‌شود.
- در انتهای مدل، از لایه‌های Dropout و لایه‌های خطی برای طبقه بندی استفاده می‌شود.

ساختن مدل از پایه:(from scratch)

در این کلاس، متد (nn.Module): این خط کلاسی به نام VGG16 را تعریف می‌کند که از nn.Module، یک بلوک اصلی برای شبکه های عصبی در PyTorch به ارث می‌رسد.

مقدار دهی اولیه:

def __init__(self, num_classes=10): این روش مقدار دهی اولیه کلاس را تعریف می‌کند. یک آرگومان اختیاری num_classes می‌گیرد که تعداد کلاس‌های خروجی را برای لایه نهایی مشخص می‌کند (پیش فرض 10 است).

super(VGG16, self).__init__(): این خط روش مقداردهی اولیه کلاس والد (nn.Module) را برای اطمینان از مقداردهی اولیه مناسب فراخوانی می کند.

کد چندین لایه را با استفاده از کانتینرهای متوالی (nn.Sequential) تعریف می کند.

شبکه در دو بخش اصلی ساخته شده است:

1. لایه های کانولوشنال: این لایه ها با استفاده از فیلترهای قابل یادگیری ویژگی‌ها را از تصویر ورودی(بصورت ضرب نقطه‌ای) استخراج می‌کنند. آن‌ها معمولاً شامل:

nn.Conv2d(in_channels، out_channels، kernel_size، stride، padding): این یک لایه کانولوشنال با تعداد مشخص کانال ورودی و خروجی، اندازه فیلتر، گام و padding را تعریف می‌کند.

2. nn.BatchNorm2d(num_features): این نرمال سازی دسته‌ای را اعمال می‌کند که به تثبیت روند آموزش کمک می‌کند.

nn.ReLU(): این تابع فعال سازی ReLU را اعمال می‌کند که غیرخطی بودن را به شبکه معرفی می‌کند.

MaxPooling Layer: این لایه‌ها عمق را دو برابر می‌کنند(باعث افزایش عمق شبکه می‌شوند.)
nn.MaxPool2d(kernel_size, stride)

کد زیر 13 لایه اول معماری VGG16 را تعریف می‌کند:

از self.layer1 به self.layer9: این لایه‌ها از یک الگوی دو لایه کانولوشن استفاده می‌کنند که به دنبال آن یک فعال‌سازی ReLU و نرمال‌سازی دسته‌ای، و به دنبال آن یک لایه ترکیبی MaxPooling استفاده می‌کنند. تعداد کانال‌های خروجی هر چه به عمق شبکه می‌رویم افزایش می‌یابد(64، 128، 256، 512).

```python
class VGG16(nn.Module):
    def __init__(self, num_classes=10) -> None:
        super(VGG16, self).__init__()
        self.layer1 : Any = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU())
        self.layer2 : Any = nn.Sequential(
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2))
        self.layer3 : Any = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU())
        self.layer4 : Any = nn.Sequential(
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2))
        self.layer5 : Any = nn.Sequential(
            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU())
        self.layer6 : Any = nn.Sequential(
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU())
        self.layer7 : Any = nn.Sequential(
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2))
        self.layer8 : Any = nn.Sequential(
            nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU())
        self.layer9 : Any = nn.Sequential(
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
d Mode  ⊗ 6 ⚠ 0  0
```

شکل ۲-۱-۱ پیاده‌سازی لایه اول تا نهم شبکه vgg

از self.layer10 به self.layer13: این لایه‌ها از یک الگوی مشابه پیروی می‌کنند، اما پس از هر دو لایه کانولوشن، یک لایه MaxPooling دارند.

```
43        self.layer10 : Any  = nn.Sequential(
44            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
45            nn.BatchNorm2d(512),
46            nn.ReLU(),
47            nn.MaxPool2d(kernel_size = 2, stride = 2))
48        self.layer11 : Any  = nn.Sequential(
49            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
50            nn.BatchNorm2d(512),
51            nn.ReLU())
52        self.layer12 : Any  = nn.Sequential(
53            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
54            nn.BatchNorm2d(512),
55            nn.ReLU())
56        self.layer13 : Any  = nn.Sequential(
57            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
58            nn.BatchNorm2d(512),
59            nn.ReLU(),
60            nn.MaxPool2d(kernel_size = 2, stride = 2))
```

شکل ۲-۱-۲ پیاده‌سازی از لایه ۱۰ تا ۱۳

متد forward:

این روش نحوه جریان داده های ورودی (x) را در شبکه تعریف می‌کند. در تمام لایه‌های تعریف شده (self.layer1 تا self.fc2) تکرار می‌شود و آن‌ها را به صورت متوالی در ورودی اعمال می‌کند.

- این متد نحوه عبور داده ها از شبکه را مشخص می‌کند.
- forward از تمام لایه ها به ترتیب عبور می‌کند.
- خروجی لایه های کانولوشن قبل از ورود به fully connected layers تسطیح می‌شود.

۵. فراخوانی مدل و پارامترهای آموزشی:

- Learning_rate = 0.005: این نرخ یادگیری را تنظیم می کند، که میزان به روز رسانی شبکه وزن های خود را بر اساس خطاها کنترل می کند (0.005 در این مورد).
- یک مدل VGG16 با خروجی 100 کلاس (تعداد کلاس های CIFAR-100) ایجاد و به دستگاه (CPU یا GPU) منتقل می شود.
- تابع زیان (Loss Function) به صورت CrossEntropyLoss تعریف می‌شود.

- criterion = nn.CrossEntropyLoss(): این تابع ضرر مورد استفاده برای اندازه‌گیری تفاوت بین برچسب‌های پیش بینی‌شده و واقعی را تعریف می‌کند. در اینجا nn.CrossEntropyLoss برای کارهای طبقه بندی چند کلاسه مناسب است.
- بهینه‌سازSGD:

torch.optim.SGD(model.parameters(),lr=learning_rate,weight_decay=0.005
:momentum=0.9

این خط بهینه ساز را تعریف می‌کند که وزن های مدل را بر اساس تلفات محاسبه شده به روز می‌کند. در اینجا از torch.optim.SGD استفاده می‌شود که مخفف Stochastic Gradient Descent، یک الگوریتم بهینه سازی رایج است. آرگومان های ارسال شده به بهینه ساز عبارتند از: model.parameters(): تمام پارامترهای قابل آموزش مدل را بازیابی می کند. lr=learning_rate: نرخ یادگیری را که قبلاً تعریف شده بود تنظیم می‌کند. weight_decay=0.005: با اعمال جریمه برای وزن‌های بزرگ در طول به‌روزرسانی، از بیش برازش جلوگیری می‌کند. ممنتم = 0.9: این سرعت همگرایی را با در نظر گرفتن گرادیان های تکرارهای قبلی بهبود می‌بخشد.

- بهینه‌ساز (Optimizer) به صورت SGD با پارامترهای نرخ یادگیری، وکاهش وزن و ممنتم مقداردهی اولیه می‌شود.

۶. آموزش مدل:

- حلقه اصلی آموزش برای تعداد مشخصی از epoch ها (دوره‌ها) اجرا می‌شود. که در اینجا تعداد epoch=20 در نظر گرفتیم و چون مدت زمان زیادی(حدود ۳ الی ۴ ساعت زمان برای اجرا شدن نیاز داشت آن را متوقف کردم.)
- در هر epoch، روی تمام دسته‌های موجود در دیتاست آموزشی (train_loader) تکرار می‌شود.
- در هر تکرار:
- تصاویر و برچسب‌های دسته به شبکه منتقل می‌شوند.
- خروجی مدل با عبور دادن تصاویر از مدل به دست می آید.
- تابع زیان بر اساس خروجی مدل و برچسب‌های واقعی محاسبه می‌شود.
- گرادیان‌ها با استفاده از بهینه‌ساز به عقب (backpropagation) محاسبه می‌شوند.
- پارامترهای مدل با به‌روزرسانی گرادیان‌ها بهینه می‌شوند.
- زیان مدل نمایش داده می‌شود.

انتقال داده: images = images.to(device): این خط دسته فعلی تصاویر را برای پردازش سریع‌تر در حین آموزش به دستگاه مشخص شده (CPU یا GPU) منتقل می‌کند. labels = labels.to(device): به همین ترتیب، برچسب ها نیز به همان دستگاه منتقل می‌شوند.

متد forward:این روش نحوه جریان داده های ورودی (x) را در شبکه تعریف می‌کند.

ضرر = معیار (خروجی ها، برچسب ها): این خط با استفاده از تابع ضرر تعریف شده (معیار) ضرر را محاسبه می کند. در اینجا، معیار احتمالاً روی nn.CrossEntropyLoss تنظیم می‌شود که تفاوت بین برچسب‌های پیش‌بینی شده (خروجی‌ها) و برچسب‌های واقعی (برچسب ها) را اندازه می‌گیرد.

Backward Pass و بهینه سازی: optimizer.zero_grad(): این خط گرادیان‌های جمع‌شده از تکرار قبلی را بازنشانی می‌کند و برای محاسبه گرادیان‌های جدید برای دسته فعلی آماده می‌شود. loss.backward(): این خط با محاسبه گرادیان تابع ضرر با توجه به پارامترهای مدل، گذر به عقب را انجام می‌دهد. optimizer.step(): این خط پارامترهای مدل را بر اساس گرادیان های محاسبه شده با استفاده از بهینه ساز تعریف شده به روز می‌کند.

7. اعتبارسنجی مدل:

- بعد از هر epoch، مدل روی مجموعه داده اعتبارسنجی (valid_loader) ارزیابی می‌شود.
- در حالت ارزیابی، گرادیان‌ها محاسبه نمی‌شوند (with torch.no_grad).
- دقت مدل بر روی مجموعه داده اعتبارسنجی محاسبه و نمایش داده می‌شود.

8. ارزیابی نهایی مدل:

- بعد از اتمام آموزش، مدل روی مجموعه داده تست (test_loader) ارزیابی می‌شود.
- مقداردهی اولیه متغیرها: correct = 0: این متغیر تعداد تصاویر طبقه بندی شده را به درستی ذخیره می‌کند. total = 0: این متغیر تعداد کل تصاویر را در مجموعه داده آزمایشی ذخیره می‌کند.
- یافتن پیش بینی های برتر: predicted = torch.max(outputs.data, 1): کلاس پیش بینی شده با بالاترین امتیاز را برای هر تصویر در دسته استخراج می کند. به معنی کنار گذاشتن حداکثر امتیازات است، زیرا ما فقط به کلاس های پیش بینی شده نیاز داریم.
- عملکرد مدل را ارزیابی می‌کند که در طول آموزش یا اعتبار سنجی استفاده نشده است. دقت نهایی مدل بر روی مجموعه داده تست محاسبه و چاپ می‌شود.

۲-۲ تغییر دیتاست مدل

توجه: این کد برای مجموعه داده CIFAR-100 نوشته شده است. برای استفاده از مجموعه داده‌های دیگر، باید نام و مسیر دیتاست را در تابع data_loader تغییر دهید.

```
    if test:
        dataset : Any  = datasets.CIFAR100(
          root=data_dir, train=False,
          download=True, transform=transform,
        )
                                  (function) utils: Unknown
        data_loader : Any  = torch.utils.data.DataLoader(
            dataset, batch_size=batch_size, shuffle=shuffle
        )

        return data_loader

    # load the dataset
    train_dataset : Any  = datasets.CIFAR100(
        root=data_dir, train=True,
        download=True, transform=transform,
    )

    valid_dataset : Any  = datasets.CIFAR10(
        root=data_dir, train=True,
        download=True, transform=transform,
    )
```

شکل ۲-۲-۱ قطعه ای ازکد مربوط به بخش بارگیری دیتا و انتخاب دیتاست

همانطور که می‌دانید چون در این کد از دیتاست کوچکتری Cifar-100به جای دیتاستImageNetاستفاده شده تعداد دسته‌ها در کد اصلی بصورت دسته‌های ۲۵۶تایی بودند اما در اینجا به دسته‌های batch_size = 16: تعداد نمونه‌هایی را که در هر بار تکرار با هم پردازش می‌شوند (16 نمونه در هر دسته) را مشخص می‌کند.

۳-۲نتیجه گیری

این یک پیاده سازی بصورت کاملا پایه از مدل vgg-16 با کتابخانه پایتورچ (pytorch) است و نیاز به زمان بسیار زیادی برای آموزش دارد تا به دقت خوبی دست پیدا کند همانطور که مشاهده کردید تعداد epoch=20در نظر گرفتیم و اگر این تعداد هر چه افزایش پیدا کند در دقت مدل تاثیر گذار است. و همچنین این مدل برای داده‌های تصاویر حجیم و در مقیاس بزرگ بکار برده می‌شود که برای تصاویر با وضوح بالا مناسب است و عملکرد خوبی را در زمان خودش داشته است.

منابع

https://paperswithcode.com/method/vgg