TRƯỜNG ĐẠI HỌC VĂN LANG

**KHOA CÔNG NGHỆ THÔNG TIN**

A picture containing text, clipart

Description automatically generated

**BÁO CÁO ĐỒ ÁN MÔN HỌC**

MÔN: THỊ GIÁC MÁY TÍNH

***Đề tài:***

***Semantic Segmentation***

**SVTH: Trần Minh Tâm**

**MSSV: 2174802010640**

**NHÓM : 98**

**T.S. Đỗ Hữu Quân**

TP. Hồ Chí Minh – năm 2023

# LỜI CẢM ƠN

Em xin bày tỏ lòng biết ơn sâu sắc và lòng tôn trọng chân thành đối với sự hỗ trợ và sự dẫn dắt tận tâm của Thầy trong quá trình thực hiện đồ án thuyết trình của mình.

Thầy không chỉ là người hướng dẫn xuất sắc mà còn là nguồn động viên và nguồn đầu tư quý báu. Những lời góp ý, phản hồi chân thành và sự kiên nhẫn của Thầy đã giúp tôi vượt qua những thách thức, phát triển kỹ năng và hiểu sâu hơn về chủ đề của dự án.

Cảm ơn thầy đã cho em cơ hội để làm bài sau 1 số vấn đề cá nhân không thể nộp bài sớm.

Lời cảm ơn này không thể nào diễn đạt hết lòng biết ơn của em. Hy vọng rằng dự án của mình có thể đáp lại một phần nho nhỏ của những đóng góp quý báu này.

Nếu có thiếu xót mong thầy nhận xét để em được rút kinh nghiệm.

Trân trọng,

# NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

....................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................

Mục lục

[LỜI CẢM ƠN 2](#_Toc153629422)

[NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN 2](#_Toc153629423)

[**Cơ sở lý thuyết** 4](#_Toc153629424)

[**Sử dụng các mô hình** 17](#_Toc153629425)

[1. Deep Learning 17](#_Toc153629426)

[**2.** Mô hình **U-Net** 19](#_Toc153629427)

[Dưới đây là một số ưu điểm chính của Semantic Segmentation: 20](#_Toc153629428)

[1. **Phân biệt đối tượng chi tiết:** 20](#_Toc153629429)

[2. **Thông tin ngữ cảnh đầy đủ:** 20](#_Toc153629430)

[3. **Ứng dụng trong thị giác máy và trí tuệ nhân tạo:** 20](#_Toc153629431)

[4. **Định vị đối tượng:** 20](#_Toc153629432)

[5. **Ứng dụng trong y học:** 20](#_Toc153629433)

[6. **Tích hợp với hệ thống tự động:** 20](#_Toc153629434)

[7. **Hỗ trợ trong dự báo và quyết định:** 20](#_Toc153629435)

[Dưới đây là một số hạn chế phổ biến của phương pháp Semantic Segmentation: 20](#_Toc153629436)

[1. **Khả năng xử lý đối tượng chồng lấn (object occlusion):** 20](#_Toc153629437)

[2. **Khả năng xử lý biên giữa các đối tượng (object boundaries):** 21](#_Toc153629438)

[3. **Yêu cầu lượng dữ liệu lớn cho đào tạo:** 21](#_Toc153629439)

[4. **Khó khăn với các lớp đối tượng hiếm gặp:** 21](#_Toc153629440)

[5. **Tính chính xác phụ thuộc vào chất lượng dữ liệu đào tạo:** 21](#_Toc153629441)

[6. **Tính toán nặng nề:** 21](#_Toc153629442)

[7. **Khả năng tổng quát hóa:** 21](#_Toc153629443)

[8. **Thiếu khả năng hiểu ngữ cảnh:** 21](#_Toc153629444)

[Tài liệu tham khảo 21](#_Toc153629445)

# **Cơ sở lý thuyết**

mask\_path = "/content/annotations/trimaps/Abyssinian\_1.png"

mask = cv2.imread(mask\_path, cv2.IMREAD\_GRAYSCALE)

plt.imshow(mask)

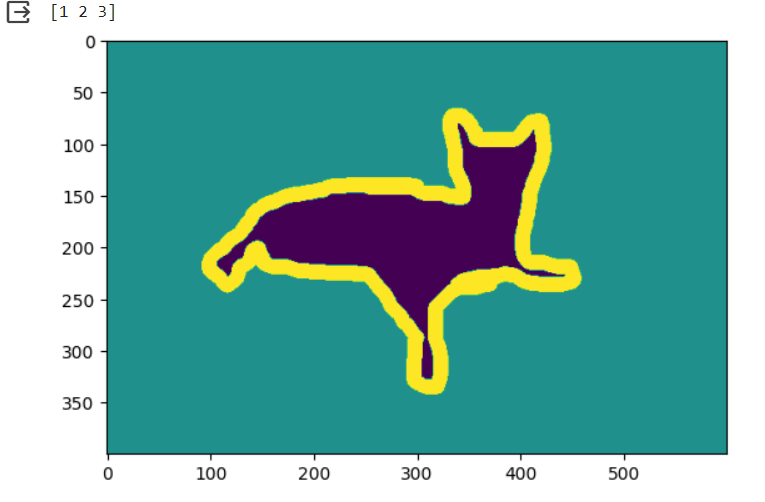
print(np.unique(mask))

Thuật toán sử dụng để đọc hình ảnh là **cv2.imread** với cờ **cv2.IMREAD\_GRAYSCALE**. Cờ này cho biết rằng bạn muốn đọc ảnh ở chế độ màu xám.

Sau khi đọc xong, bạn sử dụng **plt.imshow** để hiển thị hình ảnh.

**np.unique(mask)** được sử dụng để in ra tất cả các giá trị duy nhất có trong mảng **mask**, giúp bạn kiểm tra các giá trị pixel khác nhau trong ảnh grayscale. Điều này có thể hữu ích để hiểu rõ về nội dung của ảnh, đặc biệt là khi đối mặt với các bài toán như semantic segmentation, nơi mỗi giá trị pixel thường tương ứng với một lớp hoặc một đặc trưng cụ thể.

Để chạy đoạn mã này, bạn cần đảm bảo rằng thư viện OpenCV (**cv2**) và Matplotlib (**plt**) đã được import và cài đặt trước đó.



image\_path = "/content/images/Abyssinian\_10.jpg"

image = cv2.imread(image\_path)

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

plt.imshow(image)

print(image.shape)

1. **cv2.imread(image\_path)**: Dòng này sử dụng hàm **imread** từ OpenCV để đọc hình ảnh từ đường dẫn được chỉ định bởi biến **image\_path**. Mặc định, **imread** đọc hình ảnh ở chế độ màu BGR (Blue-Green-Red).
2. **cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)**: Sau khi đọc hình ảnh, bạn sử dụng hàm **cvtColor** để chuyển đổi không gian màu từ BGR sang RGB. Thư viện Matplotlib hiển thị hình ảnh dưới dạng RGB, vì vậy bước này là cần thiết để đảm bảo hiển thị đúng màu.
3. **plt.imshow(image)**: Dòng này sử dụng **imshow** từ Matplotlib để hiển thị hình ảnh đã đọc và chuyển đổi. Nếu không có bước chuyển đổi màu từ BGR sang RGB, hình ảnh có thể hiển thị với màu sắc không chính xác.
4. **print(image.shape)**: Dòng này in ra kích thước của hình ảnh, bao gồm chiều cao, chiều rộng và số kênh màu. Điều này giúp bạn kiểm tra kích thước của ảnh và số lượng kênh màu (thường là 3 cho ảnh màu RGB).

A cat with its paw on a stuffed animal

Description automatically generatedĐể chạy đoạn mã này, bạn cần đảm bảo rằng thư viện OpenCV (**cv2**) và Matplotlib (**plt**) đã được import và cài đặt trước đó.

#kiểm tra 1 cặp ảnh đầu vào và ảnh kết quả phân vùng trước khi đưa vào mô hình training

train\_dataset = DogCatDataset("/content", "/content/annotations/trainval.txt", train\_transform)

test\_dataset = DogCatDataset("/content", "/content/annotations/test.txt", test\_trainsform)

image, mask = train\_dataset.\_\_getitem\_\_(10)

plt.subplot(1, 2, 1)

plt.imshow(unorm(image).permute(1, 2, 0))

plt.subplot(1, 2, 2)

plt.imshow(mask)

plt.show()

1. **DogCatDataset("/content", "/content/annotations/trainval.txt", train\_transform)**: Đây có thể là một lớp hoặc hàm tạo của **DogCatDataset** được xây dựng để quản lý dữ liệu đào tạo. Bạn cung cấp đường dẫn của ảnh (**"/content"**) và một tệp tin chứa đường dẫn của ảnh và mask tương ứng (**"/content/annotations/trainval.txt"**) cùng với các biến đổi (**train\_transform**).
2. **image, mask = train\_dataset.\_\_getitem\_\_(10)**: Dòng này lấy một cặp ảnh và mask từ tập dữ liệu đào tạo. Điều này thường được sử dụng để kiểm tra xem dữ liệu đã được định dạng đúng và có những biến đổi mong muốn hay không.
3. **plt.imshow(unorm(image).permute(1, 2, 0))**: Dòng này hiển thị ảnh từ tập dữ liệu. Bạn có thể đang sử dụng một hàm **unorm** để chuyển đổi giá trị pixel trở lại khoảng [0, 1], và **permute(1, 2, 0)** để chuyển đổi chiều của ảnh từ (C, H, W) sang (H, W, C).
4. **plt.imshow(mask)**: Dòng này hiển thị mask tương ứng.

A cat lying down and a yellow object

Description automatically generatedCả hai đều được hiển thị bằng thư viện Matplotlib.

Top of Form

#Định nghĩa Dataset

class DogCatDataset(Dataset):

    def \_\_init\_\_(self, root\_dir, txt\_file, transform=None): #transform: augmentation + norm + np.array -> torch.tensor

        super().\_\_init\_\_()

        self.root\_dir = root\_dir

        self.txt\_file = txt\_file

        self.transform = transform

        self.img\_path\_lst = []

        with open(self.txt\_file) as file\_in:

            for line in file\_in:

                self.img\_path\_lst.append(line.split(" ")[0])

    def \_\_len\_\_(self):

        return len(self.img\_path\_lst)

    def \_\_getitem\_\_(self, idx):

        image\_path = os.path.join(self.root\_dir, "images", "{}.jpg".format(self.img\_path\_lst[idx]))

        mask\_path = os.path.join(self.root\_dir, "annotations", "trimaps", "{}.png".format(self.img\_path\_lst[idx]))

        image = cv2.imread(image\_path)

        image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

        mask = cv2.imread(mask\_path, cv2.IMREAD\_GRAYSCALE)

        # foreground -> 1

        # background 2 -> 0

        # 3 -> 1

        mask[mask == 2] = 0

        mask[mask == 3] = 1

        # image (RGB), mask (2D matrix)

        if self.transform is not None:

            transformed = self.transform(image=image, mask=mask)

            transformed\_image = transformed['image']

            transformed\_mask = transformed['mask']

        return transformed\_image, transformed\_mask

Mô hình phân đoạn thường được xây dựng dựa trên các kiến trúc mạng nơ-ron sâu như U-Net, FCN, SegNet, Deeplab, vv.

Các biến đổi dữ liệu (**transform**) cũng được áp dụng để tăng cường dữ liệu và chuẩn hóa dữ liệu trước khi đưa vào mô hình. Bạn sử dụng OpenCV để đọc ảnh và thực hiện một số biến đổi trên ảnh và mask, chẳng hạn như chuyển đổi màu từ BGR sang RGB, và điều chỉnh mask để chỉ có hai giá trị (0 và 1).

#Định nghĩa các phép augmentation trên ảnh

#Sử dụng thư viện Albumentations

trainsize = 384

train\_transform = A.Compose([

    A.Resize(width=trainsize, height=trainsize),

    A.HorizontalFlip(),

    A.RandomBrightnessContrast(),

    A.Blur(),

    A.Sharpen(),

    A.RGBShift(),

    A.Cutout(num\_holes=5, max\_h\_size=25, max\_w\_size=25, fill\_value=0),

    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225), max\_pixel\_value=255.0),

    ToTensorV2(),

])

test\_trainsform = A.Compose([

    A.Resize(width=trainsize, height=trainsize),

    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225), max\_pixel\_value=255.0),

    ToTensorV2(), # numpy.array -> torch.tensor (B, 3, H, W)

])

1. **A.Resize(width=trainsize, height=trainsize)**: Thực hiện việc resize ảnh về kích thước mong muốn (**trainsize x trainsize**).
2. **A.HorizontalFlip()**: Lật ảnh theo chiều ngang với xác suất 0.5, tăng cường dữ liệu bằng cách đảo ngược ảnh.
3. **A.RandomBrightnessContrast()**: Ngẫu nhiên thay đổi độ sáng và độ tương phản của ảnh.
4. **A.Blur()**: Áp dụng một lớp làm mờ để làm mịn ảnh.
5. **A.Sharpen()**: Sắc nét hóa ảnh.
6. **A.RGBShift()**: Ngẫu nhiên thay đổi giá trị pixel của các kênh màu RGB.
7. **A.Cutout(num\_holes=5, max\_h\_size=25, max\_w\_size=25, fill\_value=0)**: Thực hiện phép "cutout" bằng cách cắt ổ trống từ ảnh với số lượng lỗ và kích thước tùy chọn.
8. **A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225), max\_pixel\_value=255.0)**: Chuẩn hóa giá trị pixel của ảnh theo giá trị trung bình và độ lệch chuẩn đã cho.
9. **ToTensorV2()**: Chuyển đổi ảnh từ định dạng numpy.array sang định dạng torch.tensor, phù hợp với đầu vào của mô hình PyTorch.

Cả hai **train\_transform** và **test\_transform** đều được sử dụng để chuẩn bị dữ liệu đầu vào cho quá trình đào tạo và kiểm thử mô hình, với **train\_transform** được áp dụng thêm các phép tăng cường dữ liệu. Các phép tăng cường này giúp mô hình học được các đặc trưng phức tạp và tăng cường khả năng tổng quát hóa của mô hình.

Top of Form

#Đoạn code dùng để convert ảnh sau khi đã chuẩn hoá thành ảnh ban đầu

class UnNormalize(object):

    def \_\_init\_\_(self, mean, std):

        self.mean = mean

        self.std = std

    def \_\_call\_\_(self, tensor):

        """

        Args:

            tensor (Tensor): Tensor image of size (C, H, W) to be normalized.

        Returns:

            Tensor: Normalized image.

        """

        for t, m, s in zip(tensor, self.mean, self.std):

            t.mul\_(s).add\_(m)

            # The normalize code -> t.sub\_(m).div\_(s)

        return tensor

unorm = UnNormalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))

định nghĩa một lớp **UnNormalize** và tạo một đối tượng **unorm** từ lớp đó. Mục đích chính của lớp này là chuyển đổi lại giá trị pixel từ dạng đã chuẩn hóa về dạng ban đầu.

Phương thức **\_\_call\_\_** của lớp **UnNormalize** thực hiện việc áp dụng phép chuyển đổi ngược lại từ dạng đã chuẩn hóa về dạng ban đầu. Cụ thể, nó sử dụng các giá trị trung bình (**mean**) và độ lệch chuẩn (**std**) đã được cung cấp để thực hiện phép ngược của phép chuẩn hóa.

Phương thức này được gọi khi bạn truyền một tensor (ảnh đã chuẩn hóa) vào đối tượng **unorm**. Nó sẽ cộng trở lại giá trị trung bình và nhân độ lệch chuẩn cho từng kênh màu của ảnh.

Tóm lại, đoạn code này sử dụng thuật toán đơn giản của PyTorch để thực hiện phép ngược của phép chuẩn hóa trên tensor ảnh đã được chuẩn hóa. Phép chuẩn hóa giữa giá trị pixel của ảnh vào khoảng [0, 1] để đảm bảo rằng mô hình có thể học được mối quan hệ tốt hơn trong quá trình đào tạo.

#Lập trình mô hình UNet cơ bản

#model UNet

def unet\_block(in\_channels, out\_channels):

    return nn.Sequential(

        nn.Conv2d(in\_channels, out\_channels, 3, 1, 1),

        nn.ReLU(),

        nn.Conv2d(out\_channels, out\_channels, 3, 1, 1),

        nn.ReLU()

    )

class UNet(nn.Module):

    def \_\_init\_\_(self, n\_classes):

        super().\_\_init\_\_()

        self.n\_classes = n\_classes

        self.downsample = nn.MaxPool2d(2)

        self.upsample = nn.Upsample(scale\_factor=2, mode="bilinear")

        self.block\_down1 = unet\_block(3, 64)

        self.block\_down2 = unet\_block(64, 128)

        self.block\_down3 = unet\_block(128, 256)

        self.block\_down4 = unet\_block(256, 512)

        self.block\_neck = unet\_block(512, 1024)

        self.block\_up1 = unet\_block(1024+512, 512)

        self.block\_up2 = unet\_block(256+512, 256)

        self.block\_up3 = unet\_block(128+256, 128)

        self.block\_up4 = unet\_block(128+64, 64)

        self.conv\_cls = nn.Conv2d(64, self.n\_classes, 1) # -> (B, n\_class, H, W)

    def forward(self, x):

        # (B, C, H, W)

        x1 = self.block\_down1(x)

        x = self.downsample(x1)

        x2 = self.block\_down2(x)

        x = self.downsample(x2)

        x3 = self.block\_down3(x)

        x = self.downsample(x3)

        x4 = self.block\_down4(x)

        x = self.downsample(x4)

        x = self.block\_neck(x)

        x = torch.cat([x4, self.upsample(x)], dim=1)

        x = self.block\_up1(x)

        x = torch.cat([x3, self.upsample(x)], dim=1)

        x = self.block\_up2(x)

        x = torch.cat([x2, self.upsample(x)], dim=1)

        x = self.block\_up3(x)

        x = torch.cat([x1, self.upsample(x)], dim=1)

        x = self.block\_up4(x)

        x = self.conv\_cls(x)

        return x

model = UNet(1)

x = torch.rand(4, 3, trainsize, trainsize)

print("Input shape =", x.shape)

y = model(x).squeeze()

print("Output shape = ", y.shape)

#y true (4, 384, 384)

#y hat.squeeze() (4, 1, 384, 384) -> (4, 384, 384)

Top of Form

Triển khai một mô hình U-Net cơ bản trong PyTorch. Mô hình U-Net là một kiến trúc được sử dụng chủ yếu trong các bài toán segmentation, trong đó mục tiêu là dự đoán các đối tượng hoặc ranh giới đối tượng trong hình ảnh.

Thuật toán được sử dụng trong đoạn mã của bạn bao gồm các phần chính:

1. **Convolutional Layers (nn.Conv2d):** Được sử dụng để trích xuất đặc trưng từ ảnh đầu vào.
2. **Activation Function (nn.ReLU):** Hàm kích hoạt Rectified Linear Unit (ReLU) được áp dụng sau mỗi lớp convolution để giới thiệu tính phi tuyến vào mô hình.
3. **Max Pooling (nn.MaxPool2d):** Được sử dụng để giảm kích thước của đầu ra bằng cách chọn giá trị lớn nhất từ các vùng nhỏ của đầu vào.
4. **Upsampling (nn.Upsample):** Được sử dụng để tăng kích thước của đầu ra trước khi kết hợp nó với đầu vào từ các lớp trước đó.
5. **Concatenation (torch.cat):** Sử dụng để kết hợp đầu ra của các lớp giảm kích thước (downsampling) với đầu ra của các lớp tăng kích thước (upsampling).
6. **Final Convolution (nn.Conv2d):** Cuối cùng, một lớp convolution được sử dụng để giảm số lượng kênh đối với số lớp đầu ra mong muốn (trong trường hợp của bạn, là 1 kênh cho bài toán segmentation nhị phân).

Thuật toán này được thiết kế để tạo ra đầu ra có cùng kích thước với đầu vào và chứa thông tin về các vùng đối tượng trong hình ảnh.

#Tạo AverageMeter

class AverageMeter(object):

    def \_\_init\_\_(self):

        self.reset()

    def reset(self):

        self.val = 0

        self.avg = 0

        self.sum = 0

        self.count = 0

    def update(self, val, n=1):

        self.val = val

        self.sum += val \* n

        self.count += n

        self.avg = self.sum / self.count

**AverageMeter** là một lớp được thiết kế để theo dõi giá trị trung bình của một biến trong quá trình lặp (iteration), thường được sử dụng trong quá trình huấn luyện mô hình học máy để theo dõi giá trị mất mát, độ chính xác, hoặc bất kỳ số liệu nào khác mà bạn muốn theo dõi.

Thuật toán chính được sử dụng trong **AverageMeter** là cơ bản và bao gồm các phương thức sau:

1. **reset:** Đặt lại tất cả các giá trị về giá trị khởi tạo (thường là 0). Thường được gọi ở đầu của mỗi chu kỳ lặp hoặc epoch.
2. **update:** Cập nhật giá trị trung bình với giá trị mới và số lượng ảnh (n) tương ứng. Phương thức này được gọi ở mỗi lần lặp để cập nhật giá trị hiện tại, tổng, số lượng và giá trị trung bình.

Cụ thể:

* **val**: Giá trị hiện tại (có thể là mất mát, độ chính xác, hoặc bất kỳ số liệu nào cần theo dõi).
* **sum**: Tổng của các giá trị hiện tại (val) nhân với số lượng (n) tính đến thời điểm hiện tại.
* **count**: Tổng số lượng ảnh (n) tính đến thời điểm hiện tại.
* **avg**: Giá trị trung bình tính được từ tổng và số lượng.

#Lập trình hàm tính toán độ chính xác

#accuracy fn

def accuracy\_function(preds, targets):

    preds\_flat = preds.flatten()

    targets\_flat = targets.flatten()

    acc = torch.sum(preds\_flat == targets\_flat)

    return acc/targets\_flat.shape[0]

* So sánh từng phần tử trong **preds\_flat** và **targets\_flat**.
* Đếm số lượng phần tử khớp.
* Chia cho tổng số lượng phần tử để có độ chính xác.
* #device
* device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")
* #load data
* batch\_size = 8
* n\_workers = os.cpu\_count()
* print("num\_workers =", n\_workers)
* trainloader = torch.utils.data.DataLoader(train\_dataset, batch\_size=batch\_size,
* shuffle=True, num\_workers=n\_workers)
* testloader = torch.utils.data.DataLoader(test\_dataset, batch\_size=batch\_size,
* shuffle=False, num\_workers=n\_workers)
* #model
* model = UNet(1).to(device)
* #loss
* criterion = nn.BCEWithLogitsLoss()
* #optimizer
* optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
* n\_eps = 30
* #metrics
* dice\_fn = torchmetrics.Dice(num\_classes=2, average="macro").to(device)
* iou\_fn = torchmetrics.JaccardIndex(num\_classes=2, task="binary", average="macro").to(device)
* #meter
* acc\_meter = AverageMeter()
* train\_loss\_meter = AverageMeter()
* dice\_meter = AverageMeter()
* iou\_meter = AverageMeter()
* **Model:** Mô hình được sử dụng là một U-Net (UNet) được xác định trong đoạn mã. U-Net thường được sử dụng cho các bài toán segmentation.
* **Loss Function (Hàm Mất Mát):** Hàm mất mát được sử dụng là **nn.BCEWithLogitsLoss()**. Đây là hàm mất mát nhị phân được sử dụng thường xuyên trong các bài toán segmentation.
* **Optimizer:** Trình tối ưu hóa là Adam optimizer (**torch.optim.Adam**) với một learning rate (tốc độ học) là 1e-4.
* **Metrics (Đánh Giá):** Đối với các metrics, bạn sử dụng hai metrics là Dice (torchmetrics.Dice) và Jaccard Index (torchmetrics.JaccardIndex). Cả hai đều là các metrics thường được sử dụng trong bài toán segmentation để đo lường độ tương đồng giữa các dự đoán và nhãn thực tế.
* **Meters:** Bạn đã tạo một số AverageMeter để theo dõi và lưu trữ các thông số như độ chính xác (**acc\_meter**), mất mát (**train\_loss\_meter**), Dice (**dice\_meter**), và Jaccard Index (**iou\_meter**). Các AverageMeter này giúp theo dõi giá trị trung bình qua các lần lặp.
* **Device:** Mô hình và dữ liệu được đưa lên thiết bị GPU (cuda) nếu CUDA khả dụng; nếu không, nó sẽ chuyển về CPU.
* for ep in range(1, 1+n\_eps):
* acc\_meter.reset()
* train\_loss\_meter.reset()
* dice\_meter.reset()
* iou\_meter.reset()
* model.train()
* for batch\_id, (x, y) in enumerate(tqdm(trainloader), start=1):
* optimizer.zero\_grad()
* n = x.shape[0]
* x = x.to(device).float()
* y = y.to(device).float()
* y\_hat = model(x)
* y\_hat = y\_hat.squeeze() # -> logit (-vc, +vc)
* loss = criterion(y\_hat, y)
* loss.backward()
* optimizer.step()
* with torch.no\_grad():
* y\_hat\_mask = y\_hat.sigmoid().round().long() # -> mask (0, 1)
* dice\_score = dice\_fn(y\_hat\_mask, y.long())
* iou\_score = iou\_fn(y\_hat\_mask, y.long())
* accuracy = accuracy\_function(y\_hat\_mask, y.long())
* train\_loss\_meter.update(loss.item(), n)
* iou\_meter.update(iou\_score.item(), n)
* dice\_meter.update(dice\_score.item(), n)
* acc\_meter.update(accuracy.item(), n)
* print("EP {}, train loss = {}, accuracy = {}, IoU = {}, dice = {}".format(
* ep, train\_loss\_meter.avg, acc\_meter.avg, iou\_meter.avg, dice\_meter.avg
* ))
* if ep >= 25:
* torch.save(model.state\_dict(), "/content/model\_ep\_{}.pth".format(ep))
* **Vòng lặp ngoại (Epoch Loop):** Vòng lặp chính chạy qua các epoch từ 1 đến **n\_eps** (số epoch được đặt là 30).
* **Reset Meters:** Trước mỗi epoch mới, tất cả các AverageMeter (acc\_meter, train\_loss\_meter, dice\_meter, iou\_meter) đều được đặt lại để tính toán giá trị trung bình cho epoch mới.
* **Đặt Mô hình vào Chế Độ Huấn Luyện:** **model.train()** đặt mô hình vào chế độ huấn luyện để đảm bảo rằng các layer như dropout hoặc batch normalization hoạt động đúng cách trong quá trình huấn luyện.
* **Vòng lặp nội (Batch Loop):** Vòng lặp này chạy qua từng batch trong **trainloader**.
* **Quá Trình Lan Truyền Ngược (Backward Pass):** Dùng optimizer (**optimizer**) để thực hiện lan truyền ngược và cập nhật trọng số của mô hình.
* **Đánh Giá và Cập Nhật Meters:** Dùng các metrics (Dice, IoU, accuracy) và các meters để đánh giá hiệu suất của mô hình trên batch hiện tại và cập nhật giá trị trung bình của các meters.
* **In thông tin đào tạo:** In ra màn hình giá trị trung bình của mất mát, độ chính xác, IoU, và Dice trên toàn bộ tập dữ liệu đào tạo tại thời điểm hiện tại.
* **Lưu Trọng số Mô Hình (Optional):** Nếu số epoch là lớn hơn hoặc bằng 25, thì mô hình được lưu lại dưới dạng checkpoint (**model.state\_dict()**).
* model.eval()
* test\_iou\_meter = AverageMeter()
* test\_dice\_meter = AverageMeter()
* with torch.no\_grad():
* for batch\_id, (x, y) in enumerate(tqdm(testloader), start=1):
* n = x.shape[0]
* x = x.to(device).float()
* y = y.to(device).float()
* y\_hat = model(x)
* y\_hat = y\_hat.squeeze()
* y\_hat\_mask = y\_hat.sigmoid().round().long()
* dice\_score = dice\_fn(y\_hat\_mask, y.long())
* iou\_score = iou\_fn(y\_hat\_mask, y.long())
* test\_dice\_meter.update(dice\_score.item(), n)
* test\_iou\_meter.update(iou\_score.item(), n)
* print("TEST: IoU = {}, dice = {}".format(test\_iou\_meter.avg, test\_dice\_meter.avg))
* **Dice Score và IoU Score:** Cả hai metrics này được tính toán trong quá trình đánh giá mô hình trên tập dữ liệu kiểm thử (**testloader**).
* **Lưu Ý:**
  + Mô hình được đặt vào chế độ đánh giá (**model.eval()**) để đảm bảo các layer như dropout hoặc batch normalization hoạt động đúng cách trong quá trình đánh giá.
  + Các giá trị Dice và IoU được tính toán cho mỗi batch trong tập kiểm thử và được cập nhật vào **test\_dice\_meter** và **test\_iou\_meter**.
* **Thông Báo Kết Quả:**

Kết quả cuối cùng được in ra màn hình sau khi kiểm thử xong.

import random

model.eval()

idx = random.randint(0, 100)

with torch.no\_grad():

    x, y = test\_dataset[idx]

    # print(x.shape, y.shape) (C, H, W) -> (1, C, H, W) -> model

    x = x.to(device).float().unsqueeze(0)

    y\_hat = model(x).squeeze() #(1, 1, H, W) -> (H, W)

    y\_hat\_mask = y\_hat.sigmoid().round().long()

    # x, y, y\_hat\_mask

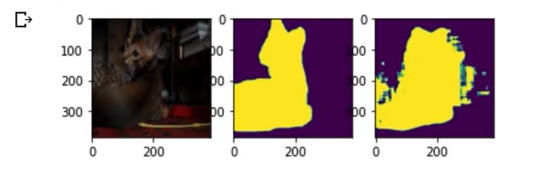
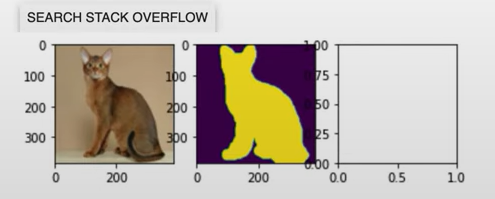
    plt.subplot(1, 3, 1)

    plt.imshow(unorm(x.squeeze().cpu()).permute(1, 2, 0)) # x (GPU) -> x(CPU)

    plt.subplot(1, 3, 2)

    plt.imshow(y)

    plt.subplot(1, 3, 3)

    plt.imshow(y\_hat\_mask.cpu())

sử dụng mô hình đã được huấn luyện để thực hiện dự đoán trên một hình ảnh kiểm thử cụ thể từ tập dữ liệu kiểm thử (**test\_dataset**). Dưới đây là các bước cụ thể:

1. **Chế Độ Đánh Giá của Mô Hình:**

Mô hình được đặt vào chế độ đánh giá để đảm bảo rằng các layer như dropout hoặc batch normalization hoạt động đúng cách trong quá trình đánh giá.

1. **Chọn Ngẫu Nhiên Hình Ảnh từ Tập Kiểm Thử:**

Một hình ảnh và nhãn tương ứng được chọn ngẫu nhiên từ tập dữ liệu kiểm thử.

1. **Chuẩn Bị Dữ Liệu và Dự Đoán:**

Hình ảnh được đưa lên thiết bị và dự đoán được thực hiện bằng cách truyền hình ảnh qua mô hình.

1. **Chuyển Đổi Dự Đoán và Hiển Thị:**

Dự đoán được chuyển đổi thành một mask bằng cách sử dụng hàm sigmoid và làm tròn.

# **Sử dụng các mô hình**

1. A screenshot of a computer

   Description automatically generatedDeep Learning

A close-up of a chart

Description automatically generated

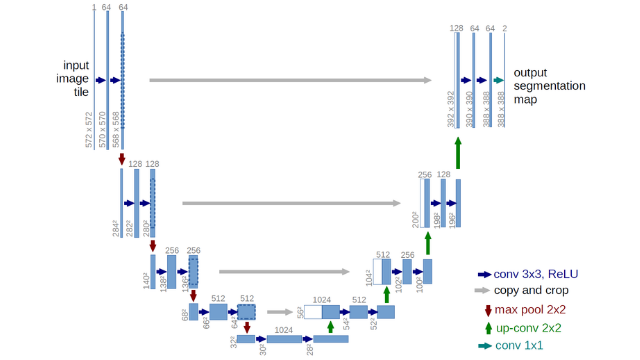
A screenshot of a computer

Description automatically generatedA diagram of a complex structure

Description automatically generated with medium confidenceA diagram of a road

Description automatically generated

1. Mô hình **U-Net**

* U-Net là một mạng nơ-ron tích chập ban đầu được phát triển để phân đoạn các hình ảnh y sinh. Kiến trúc của nó trông giống như chữ U và do đó có tên là U-Net. Kiến trúc của nó được tạo thành từ hai phần, phần bên trái – trích xuất đặc trưng (encoder) và phần bên phải – giải mã đặc trưng (decoder). Mục đích của encoder là cô đọng thông tin ngữ cảnh trong khi vai trò của decoder là giải mã thông tin, xác định chính xác các đối tượng.

# Dưới đây là một số ưu điểm chính của Semantic Segmentation:

1. **Phân biệt đối tượng chi tiết:**

Một trong những ưu điểm quan trọng nhất của Semantic Segmentation là khả năng phân biệt đối tượng ở mức pixel. Thay vì chỉ nhận diện đối tượng trong hình ảnh, nó cung cấp thông tin chi tiết về vị trí chính xác của từng pixel và đặc trưng đối tượng.

1. **Thông tin ngữ cảnh đầy đủ:**

Semantic Segmentation giúp cung cấp thông tin ngữ cảnh đầy đủ cho mỗi phần của hình ảnh. Điều này quan trọng trong việc hiểu rõ môi trường xung quanh và hỗ trợ quyết định tự động, như trong trường hợp của xe tự lái hoặc robot tự hành.

1. **Ứng dụng trong thị giác máy và trí tuệ nhân tạo:**

Semantic Segmentation là một công cụ quan trọng trong nhiều ứng dụng, bao gồm nhận diện đối tượng, theo dõi đối tượng, tăng cường thực tế, và nhiều ứng dụng khác trong lĩnh vực thị giác máy và trí tuệ nhân tạo.

1. **Định vị đối tượng:**

Khả năng xác định vị trí chính xác của đối tượng trên hình ảnh là quan trọng trong nhiều ứng dụng, từ theo dõi đối tượng đến nhận diện vị trí trong không gian 3D.

1. **Ứng dụng trong y học:**

Semantic Segmentation có thể được áp dụng trong lĩnh vực y học để phân định các cấu trúc trong hình ảnh y tế, như xác định biên của các cơ quan nội tạng, khối u, hay các vùng khác trong hình ảnh y khoa.

1. **Tích hợp với hệ thống tự động:**

Semantic Segmentation có thể tích hợp chặt chẽ với các hệ thống tự động, đặc biệt là trong các lĩnh vực như xe tự lái, robot tự hành, giám sát an ninh, và hệ thống quản lý giao thông.

1. **Hỗ trợ trong dự báo và quyết định:**

Thông tin chi tiết từ Semantic Segmentation có thể hỗ trợ trong việc dự báo và quyết định tự động, giúp hệ thống hiểu rõ hơn về môi trường và tăng cường khả năng ra quyết định.

# Dưới đây là một số hạn chế phổ biến của phương pháp Semantic Segmentation:

1. **Khả năng xử lý đối tượng chồng lấn (object occlusion):**

Khi có nhiều đối tượng chồng lấn lên nhau trong hình ảnh, mô hình Semantic Segmentation có thể gặp khó khăn trong việc phân biệt giữa chúng, đặc biệt là nếu các đối tượng này thuộc vào cùng một lớp.

1. **Khả năng xử lý biên giữa các đối tượng (object boundaries):**

Các mô hình Semantic Segmentation thường khó khăn khi xác định rõ ràng ranh giới giữa các đối tượng, làm cho kết quả segment không luôn chính xác ở biên giữa các đối tượng.

1. **Yêu cầu lượng dữ liệu lớn cho đào tạo:**

Để huấn luyện một mô hình Semantic Segmentation có hiệu suất cao, đôi khi cần một lượng lớn dữ liệu được gán nhãn, điều này có thể là một thách thức đối với các ứng dụng nơi dữ liệu nhãn hiệu khó có được.

1. **Khó khăn với các lớp đối tượng hiếm gặp:**

Nếu có lớp đối tượng ít xuất hiện trong tập dữ liệu đào tạo, mô hình có thể gặp khó khăn trong việc nhận diện và phân loại chúng.

1. **Tính chính xác phụ thuộc vào chất lượng dữ liệu đào tạo:**

Nếu dữ liệu đào tạo không đại diện cho mọi trường và điều kiện trong thực tế, mô hình có thể không chính xác khi triển khai trên dữ liệu mới.

1. **Tính toán nặng nề:**

Các mô hình Semantic Segmentation thường đòi hỏi nhiều tài nguyên tính toán, đặc biệt là khi áp dụng cho ảnh có độ phân giải cao hoặc trong thời gian thực.

1. **Khả năng tổng quát hóa:**

Mô hình huấn luyện trên một tập dữ liệu cụ thể có thể không tổng quát hoá tốt cho các tập dữ liệu khác, đặc biệt là khi có sự khác biệt lớn về điều kiện ánh sáng, góc chụp, hoặc đối tượng.

1. **Thiếu khả năng hiểu ngữ cảnh:**

Một số mô hình có thể thiếu khả năng hiểu ngữ cảnh xung quanh, dẫn đến việc phân loại đối tượng không chính xác trong ngữ cảnh tổng thể.

# Tài liệu tham khảo

<https://vinbigdata.com/camera-ai/phan-vung-anh-cac-thuat-toan-va-co-so-du-lieu-ma-nguon-mo-huu-ich.html>

<https://github.com/tungbt-2992/semantic-segmentation-with-pytorch-course>

<https://arxiv.org/abs/1903.11816>