

AI VIET NAM – COURSE 2024

Object Detection Part I - Exercise

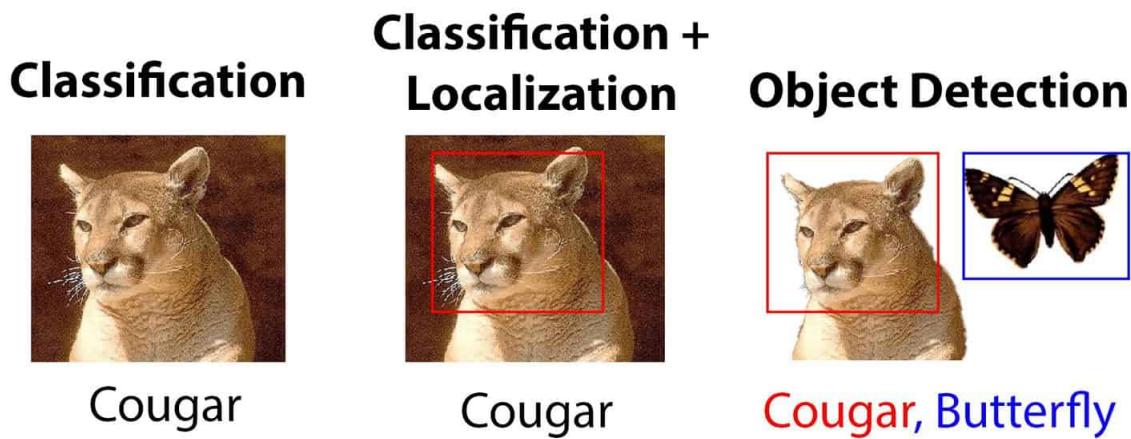
Bùi Minh Đức

Ngày 13 tháng 1 năm 2025

Phần I: Giới thiệu

Trong buổi hôm nay, chúng ta sẽ cùng nhau xây dựng model YOLO cho bài toán Object Detection. Tuy nhiên, thay vì đi thẳng vào YOLO, ta sẽ xây dựng từ bài toán Classification, sau đó thêm vào task Localization và cuối cùng là Object Detection (YOLO). Chúng ta sẽ đi qua 4 phần chính:

1. Classification (1 object trong ảnh)
2. Classification (1 object trong ảnh) + Localization (bounding box)
3. Classification (nhiều hơn 1 object trong ảnh) + Localization (bounding box)
4. YOLOv1



Hình 1: Hình ảnh minh họa cho 3 bài toán Classification, Localization và Object Detection.

Trong hình ảnh minh họa ở Figure 1, chúng ta có thể thấy sự khác biệt giữa ba bài toán chính: **Classification** chỉ đơn giản là phân loại một object trong ảnh, **Localization** không chỉ phân loại mà còn xác định vị trí của object đó bằng bounding box, và cuối cùng là **Object Detection** kết hợp cả hai yếu tố này để phát hiện và định vị nhiều object trong một hình ảnh.

Phần II: Nội dung chính

Bài toán 1: Classification (Phân loại một object trong hình ảnh)

Giới thiệu

Dầu tiên, chúng ta sẽ xây dựng một mô hình **Classification** đơn giản để làm nền tảng cho các bài toán phức tạp hơn sau này. Mục tiêu của bài toán này là phân loại hình ảnh chứa một object duy nhất thuộc về một trong hai class: **cat** (mèo) hoặc **dog** (chó).

Import và Tải Dữ liệu

Để bắt đầu, chúng ta cần import các thư viện cần thiết và tải xuống bộ dữ liệu. Bộ dữ liệu này bao gồm các hình ảnh của mèo và chó, được sử dụng rộng rãi trong các bài toán **Image Classification** và **Object Detection**. Việc sử dụng bộ dữ liệu từ kagglehub giúp chúng ta dễ dàng quản lý và truy cập dữ liệu một cách hiệu quả.

```
1 import kagglehub
2
3 # Download the latest dataset version
4 data_dir = kagglehub.dataset_download("andrewmd/dog-and-cat-detection")
5 print("Path to dataset files:", data_dir)
6
```

Import Các Thư Viện Cần Thiết

Tiếp theo, chúng ta sẽ import các thư viện cần thiết để xử lý dữ liệu và xây dựng mô hình.

```
1 import os
2 import torch
3 import numpy as np
4 import pandas as pd
5 import seaborn as sns
6 import torch.nn as nn
7 import torch.optim as optim
8 import matplotlib.pyplot as plt
9 import xml.etree.ElementTree as ET
10
11 from PIL import Image
12 from torchvision import transforms, models
13 from torch.utils.data import Dataset, DataLoader
14 from sklearn.metrics import confusion_matrix
15 from sklearn.model_selection import train_test_split
16 from torchvision.models.resnet import ResNet18_Weights
17
```

Định Nghĩa Class Dataset

Để quản lý và xử lý dữ liệu hiệu quả, chúng ta sẽ định nghĩa một class **ImageDataset**. Class này sẽ chịu trách nhiệm tải hình ảnh, xử lý các label tương ứng và thực hiện các bước tiền xử lý dữ liệu cần thiết trước khi đưa vào model.

```
1 # Dataset Class
2 class ImageDataset(Dataset):
3     def __init__(self, annotations_dir, image_dir, transform=None):
4         self.annotations_dir = annotations_dir
5         self.image_dir = image_dir
6         self.transform = transform
7         self.image_files = self.filter_images_with_multiple_objects()
8
9     def filter_images_with_multiple_objects(self):
10        valid_image_files = []
11        for f in os.listdir(self.image_dir):
12            if os.path.isfile(os.path.join(self.image_dir, f)):
13                img_name = f
14                annotation_name = os.path.splitext(img_name)[0] + ".xml"
15                annotation_path = os.path.join(self.annotations_dir,
16                                               annotation_name)
17
18                # Keep images that have single object
19                if self.count_objects_in_annotation(annotation_path) <= 1:
20                    valid_image_files.append(img_name)
21                else:
22                    print(
23                        f"Image {img_name} has multiple objects and will be
24 excluded from the dataset"
25                    )
26        return valid_image_files
27
28    def count_objects_in_annotation(self, annotation_path):
29        try:
30            tree = ET.parse(annotation_path)
31            root = tree.getroot()
32            count = 0
33            for obj in root.findall("object"):
34                count += 1
35            return count
36        except FileNotFoundError:
37            return 0
38
39    def __len__(self):
40        return len(self.image_files)
41
42    def __getitem__(self, idx):
43        # Image path
44        img_name = self.image_files[idx]
45        img_path = os.path.join(self.image_dir, img_name)
46
47        # Load image
48        image = Image.open(img_path).convert("RGB")
49
50        # Annotation path
51        annotation_name = os.path.splitext(img_name)[0] + ".xml"
52        annotation_path = os.path.join(self.annotations_dir, annotation_name)
53
54        # Parse annotation
55        label = self.parse_annotation(annotation_path)
56
57        if self.transform:
58            image = self.transform(image)
```

```

57
58         return image, label
59
60     def parse_annotation(self, annotation_path):
61         tree = ET.parse(annotation_path)
62         root = tree.getroot()
63
64         label = None
65         for obj in root.findall("object"):
66             name = obj.find("name").text
67             if (
68                 label is None
69             ): # Take the first label for now. We are working with 1 label per
image
70             label = name
71
72         # Convert label to numerical representation (0 for cat, 1 for dog)
73         label_num = 0 if label == "cat" else 1 if label == "dog" else -1
74
75         return label_num
76

```

Trong class `ImageDataset`, chúng ta thực hiện các bước sau:

- **Khởi tạo:** Xác định thư mục chứa annotations và hình ảnh, cũng như các biến đổi dữ liệu nếu có.
- **Lọc dữ liệu:** Loại bỏ những hình ảnh chứa nhiều hơn một object để đảm bảo tính nhất quán của bài toán **Classification**.
- **Đếm object:** Đếm số lượng object trong mỗi hình ảnh dựa trên các annotations.
- **Lấy mẫu:** Định nghĩa method `__getitem__` để truy xuất hình ảnh và nhãn tương ứng.
- **Phân tích annotations:** Chuyển đổi nhãn văn bản thành số để dễ dàng xử lý trong mô hình.

Phân Tích và Chuẩn Bị Dữ Liệu

Sau khi định nghĩa class dataset, chúng ta tiến hành phân tích và chuẩn bị dữ liệu.

```

1 # Data directory
2 annotations_dir = os.path.join(data_dir, 'annotations')
3 image_dir = os.path.join(data_dir, 'images')
4
5 # Get list of image files and create a dummy dataframe to split the data
6 image_files = [f for f in os.listdir(image_dir) if os.path.isfile(os.path.join(
    image_dir, f))]
7 df = pd.DataFrame({'image_name': image_files})
8
9 # Split data
10 train_df, val_df = train_test_split(df, test_size=0.2, random_state=42)
11

```

Chuẩn Bị và Split Dữ Liệu

Chúng ta sẽ áp dụng các biến đổi (transforms) cho dữ liệu hình ảnh và chia tách dữ liệu thành tập training (train set) và tập testing (test set).

```
1 # Transforms
2 transform = transforms.Compose([
3     transforms.Resize((224, 224)),
4     transforms.ToTensor(),
5     transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
6 ])
7
8 # Datasets
9 train_dataset = ImageDataset(annotations_dir, image_dir, transform=transform)
10 val_dataset = ImageDataset(annotations_dir, image_dir, transform=transform)
11
12 # Filter datasets based on train_df and val_df
13 train_dataset.image_files = [f for f in train_dataset.image_files if f in
14     train_df['image_name'].values]
14 val_dataset.image_files = [f for f in val_dataset.image_files if f in val_df['
    image_name'].values]
15
16 # Dataloaders
17 train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
18 val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
19
```

Xây Dựng Model

Chúng ta sẽ sử dụng model ResNet18 và finetune để phù hợp với bài toán classification của chúng ta.

```
1 # Model
2 model = models.resnet18(weights=ResNet18_Weights.DEFAULT)
3 num_ftrs = model.fc.in_features
4 model.fc = nn.Linear(num_ftrs, 2) # 2 classes: cat and dog
5
6 # Device
7 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
8 model.to(device)
9
10 # Loss and Optimizer
11 criterion = nn.CrossEntropyLoss()
12 optimizer = optim.Adam(model.parameters(), lr=0.001)
13
14 # Show model summary
15 print(model)
16
```

Training loop

Cuối cùng, chúng ta sẽ training model trên train set và evaluate trên test set.

```

1 # Training Loop
2 num_epochs = 10
3 for epoch in range(num_epochs):
4     model.train()
5     for batch_idx, (data, targets) in enumerate(train_loader):
6         data = data.to(device)
7         targets = targets.to(device)
8
9         scores = model(data)
10        loss = criterion(scores, targets)
11
12        optimizer.zero_grad()
13        loss.backward()
14        optimizer.step()
15
16    # Validation
17    model.eval()
18    with torch.no_grad():
19        correct = 0
20        total = 0
21        for data, targets in val_loader:
22            data = data.to(device)
23            targets = targets.to(device)
24            scores = model(data)
25            _, predictions = scores.max(1)
26            correct += (predictions == targets).sum()
27            total += targets.size(0)
28
29            print(f'Epoch {epoch+1}/{num_epochs}, Validation Accuracy: {float(correct)
30 )/float(total)*100:.2f}%)'

```

Sau khi huấn luyện xong, kết quả đạt được của mô hình như sau:



Hình 2: Kết quả của mô hình sau khi huấn luyện 10 epoch.

Bài toán 2: Classification + Bounding Box Regression

Trong bài toán thứ hai, chúng ta không chỉ thực hiện **classification** object trong hình ảnh mà còn định vị vị trí của object đó bằng cách dự đoán **bounding box**.

Import và Tải Dữ liệu

Giống như bài toán trước, chúng ta bắt đầu bằng việc import các thư viện cần thiết và tải dữ liệu.

```
1 import kagglehub
2 # Download latest version
3 data_dir = kagglehub.dataset_download("andrewmd/dog-and-cat-detection")
4 print("Path to dataset files:", data_dir)
5
```

Import Các Thư Viện Cần Thiết

```
1 import os
2 import torch
3 import numpy as np
4 import pandas as pd
5 import seaborn as sns
6 import torch.nn as nn
7 import torch.optim as optim
8 import matplotlib.pyplot as plt
9 import matplotlib.patches as patches
10 import xml.etree.ElementTree as ET
11
12 from PIL import Image
13 from torchvision import transforms, models
14 from torch.utils.data import Dataset, DataLoader
15 from sklearn.metrics import confusion_matrix
16 from sklearn.model_selection import train_test_split
17 from torchvision.models.resnet import ResNet18_Weights
18
```

Định Nghĩa Class Dataset với Bounding Box

Chúng ta sẽ mở rộng lớp `ImageDataset` để bao gồm cả thông tin về **bounding box**.

```
1 # Dataset Class
2 class ImageDataset(Dataset):
3     def __init__(self, annotations_dir, image_dir, transform=None):
4         self.annotations_dir = annotations_dir
5         self.image_dir = image_dir
6         self.transform = transform
7         self.image_files = self.filter_images_with_multiple_objects()
8
9     def filter_images_with_multiple_objects(self):
10         valid_image_files = []
11         for f in os.listdir(self.image_dir):
12             if os.path.isfile(os.path.join(self.image_dir, f)):
```

```
13     img_name = f
14     annotation_name = os.path.splitext(img_name)[0] + ".xml"
15     annotation_path = os.path.join(self.annotations_dir,
16                                     annotation_name)
17
18         if self.count_objects_in_annotation(annotation_path) == 1:
19             valid_image_files.append(img_name)
20
21     return valid_image_files
22
23 def count_objects_in_annotation(self, annotation_path):
24     try:
25         tree = ET.parse(annotation_path)
26         root = tree.getroot()
27         count = 0
28         for obj in root.findall('object'):
29             count += 1
30
31     return count
32 except FileNotFoundError:
33     return 0
34
35 def __len__(self):
36     return len(self.image_files)
37
38 def __getitem__(self, idx):
39     # Image path
40     img_name = self.image_files[idx]
41     img_path = os.path.join(self.image_dir, img_name)
42
43     # Load image
44     image = Image.open(img_path).convert("RGB")
45
46     # Annotation path
47     annotation_name = os.path.splitext(img_name)[0] + ".xml"
48     annotation_path = os.path.join(self.annotations_dir, annotation_name)
49
50     # Parse annotation
51     label, bbox = self.parse_annotation(annotation_path) # Get both label
52     and bbox
53
54     if self.transform:
55         image = self.transform(image)
56
57     return image, label, bbox
58
59 def parse_annotation(self, annotation_path):
60     tree = ET.parse(annotation_path)
61     root = tree.getroot()
62
63     # Get image size for normalization
64     image_width = int(root.find('size/width').text)
65     image_height = int(root.find('size/height').text)
66
67     label = None
68     bbox = None
69     for obj in root.findall('object'):
70         name = obj.find('name').text
71         if label is None: # Take the first label
72             label = name
73
74         # Get bounding box coordinates
```

```

70         xmin = int(obj.find('bndbox/xmin').text)
71         ymin = int(obj.find('bndbox/ymin').text)
72         xmax = int(obj.find('bndbox/xmax').text)
73         ymax = int(obj.find('bndbox/ymax').text)
74
75         # Normalize bbox coordinates to [0, 1]
76         bbox = [
77             xmin / image_width,
78             ymin / image_height,
79             xmax / image_width,
80             ymax / image_height,
81         ]
82
83         # Convert label to numerical representation (0 for cat, 1 for dog)
84         label_num = 0 if label == 'cat' else 1 if label == 'dog' else -1
85
86     return label_num, torch.tensor(bbox, dtype=torch.float32)
87

```

Phân Tích và Chuẩn Bị Dữ Liệu

Tương tự như bài toán trước, chúng ta sẽ chuẩn bị dữ liệu cho bài toán này.

```

1 # Data directory
2 annotations_dir = os.path.join(data_dir, 'annotations')
3 image_dir = os.path.join(data_dir, 'images')
4
5 # Get list of image files and create a dummy dataframe to split the data
6 image_files = [f for f in os.listdir(image_dir) if os.path.isfile(os.path.join(
    image_dir, f))]
7 df = pd.DataFrame({'image_name': image_files})
8
9 # Split data
10 train_df, val_df = train_test_split(df, test_size=0.2, random_state=42)
11

```

Chuẩn Bị và Split Dữ Liệu

Chúng ta áp dụng các biến đổi (`transforms`) cho dữ liệu và chia tách dữ liệu thành các `train set` và `test set`.

```

1 # Transforms
2 transform = transforms.Compose([
3     transforms.Resize((224, 224)),
4     transforms.ToTensor(),
5     transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
6 ])
7
8 # Datasets
9 train_dataset = ImageDataset(annotations_dir, image_dir, transform=transform)
10 val_dataset = ImageDataset(annotations_dir, image_dir, transform=transform)
11
12 # Filter datasets based on train_df and val_df

```

```

13 train_dataset.image_files = [f for f in train_dataset.image_files if f in
14     train_df['image_name'].values]
15 val_dataset.image_files = [f for f in val_dataset.image_files if f in val_df['
16     image_name'].values]
17
18 # Dataloaders
19 train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
20 val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
21

```

Xây Dựng Model với 2 head

Chúng ta sẽ xây dựng một mô hình có hai head: một để **classification** và một để dự đoán **bounding box**.

```

1 # Model with Two Heads
2 class TwoHeadedModel(nn.Module):
3     def __init__(self, num_classes=2):
4         super(TwoHeadedModel, self).__init__()
5         self.base_model = models.resnet18(weights=ResNet18_Weights.DEFAULT)
6         self.num_ftrs = self.base_model.fc.in_features
7
8         # Remove the original fully connected layer
9         self.base_model.fc = nn.Identity()
10
11        # Classification head
12        self.classifier = nn.Linear(self.num_ftrs, num_classes)
13
14        # Bounding box regression head
15        self.regressor = nn.Linear(self.num_ftrs, 4)
16
17    def forward(self, x):
18        x = self.base_model(x)
19        class_logits = self.classifier(x)
20        bbox_coords = torch.sigmoid(self.regressor(x))
21        return class_logits, bbox_coords
22

```

Trong mô hình này, chúng ta thực hiện các bước sau:

- **Base model:** Sử dụng pre-trained ResNet18
- **Classification head:** Thêm một lớp fully connected mới để thực hiện nhiệm vụ phân loại.
- **Regression head:** Thêm một lớp fully connected khác để dự đoán tọa độ bounding box, với mỗi tọa độ được chuẩn hóa về khoảng [0, 1] thông qua hàm **sigmoid**.

Cài Đặt Model

Chúng ta sẽ khởi tạo mô hình, thiết lập thiết bị, **loss function** và **optimizer**.

```

1 # Model
2 model = TwoHeadedModel()

```

```
3
4 # Device
5 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
6 model.to(device)
7
8 # Loss and Optimizer
9 criterion_class = nn.CrossEntropyLoss()
10 criterion_bbox = nn.MSELoss()
11 optimizer = optim.Adam(model.parameters(), lr=0.001)
12
```

Training loop

Chúng ta sẽ huấn luyện mô hình với cả hai head và theo dõi hiệu suất trên test set.

```
1 # Training Loop
2 num_epochs = 10
3 for epoch in range(num_epochs):
4     model.train()
5     for batch_idx, (data, targets, bboxes) in enumerate(train_loader):
6         data = data.to(device)
7         targets = targets.to(device)
8         bboxes = bboxes.to(device)
9
10        scores, pred_bboxes = model(data)
11        loss_class = criterion_class(scores, targets)
12        loss_bbox = criterion_bbox(pred_bboxes, bboxes)
13        loss = loss_class + loss_bbox # Combine losses
14
15        optimizer.zero_grad()
16        loss.backward()
17        optimizer.step()
18
19    # Validation
20    model.eval()
21    with torch.no_grad():
22        correct = 0
23        total = 0
24        total_loss_bbox = 0
25        total_samples = 0
26        for data, targets, bboxes in val_loader:
27            data = data.to(device)
28            targets = targets.to(device)
29            bboxes = bboxes.to(device)
30
31            scores, pred_bboxes = model(data)
32            _, predictions = scores.max(1)
33            correct += (predictions == targets).sum()
34            total += targets.size(0)
35
36            # Calculate bbox loss for monitoring (optional)
37            total_loss_bbox += criterion_bbox(pred_bboxes, bboxes).item() * data.
38            size(0)
39            total_samples += data.size(0)
40
41        avg_loss_bbox = total_loss_bbox / total_samples
```

```
41
42     print(f'Epoch {epoch+1}/{num_epochs}, Validation Accuracy: {float(correct
43 )/float(total)*100:.2f}%, '
44         f'Avg. Bbox Loss: {avg_loss_bbox:.4f}')
```



Hình 3: Kết quả của mô hình sau khi huấn luyện 10 epoch.

Bài toán 3: Classification (> 2 objects in an image) + Bounding Box Regression

Giới thiệu

Trong bài toán thứ ba, chúng ta mở rộng phạm vi của **classification** bằng cách xử lý hình ảnh **chứa nhiều hơn hai object** và thực hiện **bounding box regression** để định vị các object này. Bài toán này đòi hỏi mô hình không chỉ phân loại các object mà còn xác định chính xác vị trí của từng object trong hình ảnh.

Import và Tải Dữ liệu

Dầu tiên, chúng ta sẽ import các thư viện cần thiết và tải bộ dữ liệu từ kagglehub.

```
1 import kagglehub
2
3 # Download latest version
4 data_dir = kagglehub.dataset_download("andrewmd/dog-and-cat-detection")
5 print("Path to dataset files:", data_dir)
6
```

Import Các Thư Viện Cần Thiết

Tiếp theo, chúng ta sẽ import các thư viện cần thiết để xử lý dữ liệu, xây dựng mô hình và thực hiện các thao tác đánh giá.

```
1 import os
2 import torch
3 import random
4 import numpy as np
5 import pandas as pd
6 import seaborn as sns
7 import torch.nn as nn
8 import torch.optim as optim
9 import matplotlib.pyplot as plt
10 import torch.nn.functional as F
11 import matplotlib.patches as patches
12 import xml.etree.ElementTree as ET
13 import tqdm.notebook as tqdm
14
15 from PIL import Image
16 from torchvision import transforms, models
17 from torch.utils.data import Dataset, DataLoader
18 from sklearn.metrics import confusion_matrix
19 from sklearn.model_selection import train_test_split
20 from torchvision.models.resnet import ResNet18_Weights, ResNet50_Weights
21
```

Định Nghĩa Class Dataset với Nhiều object

Chúng ta sẽ định nghĩa một lớp **MyDataset** để xử lý các hình ảnh chứa nhiều object và thực hiện các thao tác như merge hình ảnh, chia thành các patch, và chuẩn bị dữ liệu cho mô hình.

```
1 class MyDataset(Dataset):
2     def __init__(self, annotations_dir, image_dir, transform=None):
3         self.annotations_dir = annotations_dir
4         self.image_dir = image_dir
5         self.transform = transform
6         self.image_files = self.filter_images_with_multiple_objects()
7
8     def filter_images_with_multiple_objects(self):
9         valid_image_files = []
10        for f in os.listdir(self.image_dir):
11            if os.path.isfile(os.path.join(self.image_dir, f)):
12                img_name = f
13                annotation_name = os.path.splitext(img_name)[0] + ".xml"
14                annotation_path = os.path.join(self.annotations_dir,
annotation_name)
15
16                if self.count_objects_in_annotation(annotation_path) == 1:
17                    valid_image_files.append(img_name)
18        return valid_image_files
19
20    def count_objects_in_annotation(self, annotation_path):
21        try:
22            tree = ET.parse(annotation_path)
23            root = tree.getroot()
24            count = 0
25            for obj in root.findall("object"):
26                count += 1
27            return count
28        except FileNotFoundError:
29            return 0
30
31    def parse_annotation(self, annotation_path):
32        tree = ET.parse(annotation_path)
33        root = tree.getroot()
34
35        # Get image size for normalization
36        image_width = int(root.find("size/width").text)
37        image_height = int(root.find("size/height").text)
38
39        label = None
40        bbox = None
41        for obj in root.findall("object"):
42            name = obj.find("name").text
43            if label is None: # Take the first label
44                label = name
45            # Get bounding box coordinates
46            xmin = int(obj.find("bndbox/xmin").text)
47            ymin = int(obj.find("bndbox/ymin").text)
48            xmax = int(obj.find("bndbox/xmax").text)
49            ymax = int(obj.find("bndbox/ymax").text)
50
51            # Normalize bbox coordinates to [0, 1]
52            bbox = [
53                xmin / image_width,
54                ymin / image_height,
55                xmax / image_width,
56                ymax / image_height,
57            ]
58
```

```
58
59     # Convert label to numerical representation (0 for cat, 1 for dog)
60     label_num = 0 if label == "cat" else 1 if label == "dog" else -1
61
62     return label_num, torch.tensor(bbox, dtype=torch.float32)
63
64     def __len__(self):
65         return len(self.image_files)
66
67     def __getitem__(self, idx):
68         img1_file = self.image_files[idx]
69         img1_path = os.path.join(self.image_dir, img1_file)
70
71         annotation_name = os.path.splitext(img1_file)[0] + ".xml"
72         img1_annotations = self.parse_annotation(
73             os.path.join(self.annotations_dir, annotation_name)
74         )
75
76         idx2 = random.randint(0, len(self.image_files) - 1)
77         img2_file = self.image_files[idx2]
78         img2_path = os.path.join(self.image_dir, img2_file)
79
80         annotation_name = os.path.splitext(img2_file)[0] + ".xml"
81         img2_annotations = self.parse_annotation(
82             os.path.join(self.annotations_dir, annotation_name)
83         )
84
85         img1 = Image.open(img1_path).convert("RGB")
86         img2 = Image.open(img2_path).convert("RGB")
87
88         # Horizontal merge
89         merged_image = Image.new(
90             "RGB", (img1.width + img2.width, max(img1.height, img2.height))
91         )
92         merged_image.paste(img1, (0, 0))
93         merged_image.paste(img2, (img1.width, 0))
94         merged_w = img1.width + img2.width
95         merged_h = max(img1.height, img2.height)
96
97         merged_annotations = []
98
99         # No change for objects from img1, already normalized
100        merged_annotations.append(
101            {"bbox": img1_annotations[1].tolist(), "label": img1_annotations[0]}
102        )
103
104        # Adjust bbox coordinates for objects from img2 AND normalize
105        new_bbox = [
106            (img2_annotations[1][0] * img2.width + img1.width) / merged_w, # Normalize xmin
107            img2_annotations[1][1] * img2.height / merged_h, # Normalize ymin
108            (img2_annotations[1][2] * img2.width + img1.width) / merged_w, # Normalize xmax
109            img2_annotations[1][3] * img2.height / merged_h, # Normalize ymax
110        ]
111
```

```

112     merged_annotations.append({"bbox": new_bbox, "label": img2_annotations
113     [0]})

114     # Convert merged image to tensor
115     if self.transform:
116         merged_image = self.transform(merged_image)
117     else:
118         merged_image = transforms.ToTensor()(merged_image)

119     # Convert annotations to 1D tensors, with shape (4,) for bbox and (1,)
120     for label:
121         annotations = torch.zeros((len(merged_annotations), 5))
122         for i, ann in enumerate(merged_annotations):
123             annotations[i] = torch.cat((torch.tensor(ann["bbox"]),
124                                         torch.tensor([ann["label"]])))

125     return merged_image, annotations
126

```

Trong class MyDataset, chúng ta thực hiện các bước sau:

- **Merge hình ảnh:** Chọn ngẫu nhiên hai hình ảnh và ghép chúng lại thành một hình ảnh duy nhất bằng cách dán hai hình ảnh cạnh nhau.
- **Điều chỉnh annotations:** Điều chỉnh tọa độ bounding box của object trong patch mới sau khi ghép hình ảnh.
- **Trả về dữ liệu:** Mỗi mẫu dữ liệu bao gồm 4 patch hình ảnh và các annotations tương ứng cho từng patch.
- **Trình bày dữ liệu:** Dữ liệu trả về sẽ có dạng (bounding box, class), với shape là [2, 5].

Phân Tích và Chuẩn Bị Dữ Liệu

Sau khi định nghĩa lớp dataset, chúng ta tiến hành phân tích và chuẩn bị dữ liệu cho bài toán này.

```

1 # Data directory
2 annotations_dir = os.path.join(data_dir, 'annotations')
3 image_dir = os.path.join(data_dir, 'images')
4
5 # Define transformations
6 transform = transforms.Compose([
7     transforms.Resize((224, 224)),
8     transforms.ToTensor()
9 ])
10
11 # Create dataset and dataloaders
12 dataset = MyDataset(annotations_dir, image_dir, transform=transform)
13 train_dataset, val_dataset = train_test_split(dataset, test_size=0.2,
14                                              random_state=42)
15 train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
16 val_loader = DataLoader(val_dataset, batch_size=8, shuffle=False)

```

Xây Dựng Model với Hai Head

Chúng ta sẽ xây dựng một mô hình có hai head: một để **classification** và một để dự đoán **bounding box**. Trong trường hợp này, chúng ta sử dụng ResNet50 làm backbone cho mô hình.

```

1 class SimpleYOLO(nn.Module):
2     def __init__(self, num_classes):
3         super(SimpleYOLO, self).__init__()
4         self.backbone = models.resnet50(weights=ResNet50_Weights.DEFAULT)
5         self.num_classes = num_classes
6
7         # Remove the final classification layer of ResNet
8         self.backbone = nn.Sequential(*list(self.backbone.children())[:-2])
9
10        # Add the YOLO head
11        self.fcs = nn.Linear(
12            2048, 2 * 2 * (4 + self.num_classes)
13        ) # 2 is for the number of grid cell
14
15    def forward(self, x):
16        # x shape: (batch_size, C, H, W)
17        features = self.backbone(x)
18        features = F.adaptive_avg_pool2d(features, (1, 1)) # shape: (batch_size,
19        2048, 1, 1)
20        features = features.view(features.size(0), -1) # shape: (batch_size,
21        2048)
22        features = self.fcs(features)
23
24    return features
25

```

Cài Đặt Model

Chúng ta sẽ khởi tạo mô hình, thiết lập thiết bị, **loss function** và **optimizer**.

```

1 # Initialize model, criterion, and optimizer
2 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
3 num_classes = 2 # Assuming two classes: dog and cat
4 class_to_idx = {'dog': 0, 'cat': 1}
5
6 model = SimpleYOLO(num_classes=num_classes).to(device)
7 optimizer = optim.Adam(model.parameters(), lr=0.001)
8

```

Training Loop

Chúng ta sẽ huấn luyện mô hình với cả hai head và theo dõi hiệu suất trên tập kiểm tra (validation set).

```

1 def calculate_loss(output, targets, device, num_classes):
2     mse_loss = nn.MSELoss()
3     ce_loss = nn.CrossEntropyLoss()
4

```

```
5     batch_size = output.shape[0]
6     total_loss = 0
7
8     output = output.view(batch_size, 2, 2, 4 + num_classes) # Reshape to (
9         batch_size, grid_y, grid_x, 4 + num_classes)
10
11    for i in range(batch_size): # Iterate through each image in the batch
12        for j in range(len(targets[i])): # Iterate through objects in the image
13
14            # Determine which grid cell the object's center falls into
15            # Assuming bbox coordinates are normalized to [0, 1]
16            bbox_center_x = (targets[i][j][0] + targets[i][j][2]) / 2
17            bbox_center_y = (targets[i][j][1] + targets[i][j][3]) / 2
18
19            grid_x = int(bbox_center_x * 2) # Multiply by number of grid cells
20                (2 in this case)
21            grid_y = int(bbox_center_y * 2)
22
23            # 1. Classification Loss for the responsible grid cell
24            # Convert label to one-hot encoding only for this example
25            # One hot encoding
26            label_one_hot = torch.zeros(num_classes, device=device)
27            label_one_hot[int(targets[i][j][4])] = 1
28
29            # Classification loss (using CrossEntropyLoss)
30            classification_loss = ce_loss(output[i, grid_y, grid_x, 4:], 
31                label_one_hot)
32
33            # 2. Regression Loss for the responsible grid cell
34            bbox_target = targets[i][j][:4].to(device)
35            regression_loss = mse_loss(output[i, grid_y, grid_x, :4], bbox_target
36        )
37
38            # 3. No Object Loss (for other grid cells)
39            no_obj_loss = 0
40            for other_grid_y in range(2):
41                for other_grid_x in range(2):
42                    if other_grid_y != grid_y or other_grid_x != grid_x:
43                        # MSE loss for predicting no object (all zeros)
44                        no_obj_loss += mse_loss(output[i, other_grid_y,
45                            other_grid_x, :4], torch.zeros(4, device=device))
46
47            total_loss += classification_loss + regression_loss + no_obj_loss
48
49    return total_loss / batch_size # Average loss over the batch
50
51
52    def evaluate_model(model, data_loader, device, num_classes):
53        model.eval()
54        running_loss = 0.0
55        all_predictions = []
56        all_targets = []
57
58        with torch.no_grad():
59            for images, targets in tqdm.tqdm(data_loader, desc="Validation", leave=
59 False):
60                images = images.to(device)
61
62                output = model(images)
```

```
58         total_loss = calculate_loss(output, targets, device, num_classes)
59         running_loss += total_loss.item()
60
61     # Reshape output to (batch_size, grid_y, grid_x, 4 + num_classes)
62     output = output.view(images.shape[0], 2, 2, 4 + num_classes)
63
64     # Collect predictions and targets for mAP calculation
65     for batch_idx in range(images.shape[0]):
66         for target in targets[batch_idx]:
67             # Determine responsible grid cell
68             bbox_center_x = (target[0] + target[2]) / 2
69             bbox_center_y = (target[1] + target[3]) / 2
70             grid_x = int(bbox_center_x * 2)
71             grid_y = int(bbox_center_y * 2)
72
73             # Class prediction (index of max probability)
74             prediction = output[batch_idx, grid_y, grid_x, 4:].argmax().item()
75             all_predictions.append(prediction)
76
77             all_targets.append(target[4].item())
78
79     val_loss = running_loss / len(data_loader)
80
81     # Convert lists to tensors for PyTorch's metric functions
82     all_predictions = torch.tensor(all_predictions, device=device)
83     all_targets = torch.tensor(all_targets, device=device)
84
85     # Calculate accuracy
86     val_accuracy = (all_predictions == all_targets).float().mean()
87
88     return val_loss, val_accuracy.item()
89
90 def train_model(
91     model, train_loader, val_loader, optimizer, num_epochs, device, num_classes
92 ):
93     best_val_accuracy = 0.0
94     train_losses = []
95     val_losses = []
96     train_accuracies = []
97     val_accuracies = []
98
99     for epoch in tqdm.tqdm(range(num_epochs), desc="Epochs"):
100         model.train()
101         running_loss = 0.0
102
103         for images, targets in tqdm.tqdm(train_loader, desc="Batches", leave=False):
104             images = images.to(device)
105
106             optimizer.zero_grad()
107             output = model(images)
108
109             total_loss = calculate_loss(output, targets, device, num_classes)
110
111             total_loss.backward()
112             optimizer.step()
113             running_loss += total_loss.item()
114
```

```

115     epoch_loss = running_loss / len(train_loader)
116     train_losses.append(epoch_loss)
117
118     # Validation
119     val_loss, val_accuracy = evaluate_model(model, val_loader, device,
120     num_classes)
120     val_losses.append(val_loss)
121     val_accuracies.append(val_accuracy)
122
123     print(
124         f"Epoch {epoch+1}/{num_epochs}, Train Loss: {epoch_loss:.4f},
125         Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}"
126     )
127
128     # Save the best model
129     if val_accuracy > best_val_accuracy:
130         best_val_accuracy = val_accuracy
131         torch.save(model.state_dict(), "best_model.pth")
132
133 return train_losses, val_losses, train_accuracies, val_accuracies

```

Inference

Sau khi huấn luyện, chúng ta có thể sử dụng mô hình để thực hiện dự đoán trên các hình ảnh mới.

```

1 def inference(model, image_path, transform, device, class_to_idx, threshold=0.5):
2     model.eval()
3     image = Image.open(image_path).convert("RGB")
4     original_width, original_height = image.size
5
6     # Resize the image to match the input size expected by the model (e.g., 448
7     #x448)
8     resized_image = image.resize((448, 448))
9     resized_width, resized_height = resized_image.size
10
11    # Apply the same transformations used during training
12    transformed_image = transform(resized_image).unsqueeze(0).to(device)
13
14    with torch.no_grad():
15        output = model(transformed_image)
16        output = output.view(1, 2, 2, 4 + len(class_to_idx))  # Reshape for 2x2
17        grid
18
19        fig, ax = plt.subplots(1)
20        ax.axis("off")
21        ax.imshow(resized_image)  # Display resized image
22
23        for grid_y in range(2):
24            for grid_x in range(2):
25                # Get the class prediction and bounding box for the current grid cell
26                class_pred = output[0, grid_y, grid_x, 4:].argmax().item()
27                bbox = output[0, grid_y, grid_x, :4].tolist()  # Predicted bbox
28
29                # Confidence (probability of the predicted class)
30                confidence = torch.softmax(output[0, grid_y, grid_x, 4:], dim=0)[

```

```

29         class_pred
30     ].item()
31
32     # Scale the bounding box back to the resized image size
33     # Assuming bbox coordinates are normalized to [0, 1] within the grid
34     cell
35         x_min = bbox[0] * (resized_width / 2) + grid_x * (resized_width / 2)
36         y_min = bbox[1] * (resized_height / 2) + grid_y * (resized_height / 2)
37
38         x_max = bbox[2] * (resized_width / 2) + grid_x * (resized_width / 2)
39         y_max = bbox[3] * (resized_height / 2) + grid_y * (resized_height / 2)
40
41     # Draw the bounding box and label on the image if confidence is above
42     threshold
43     if confidence > threshold:
44         rect = patches.Rectangle(
45             (x_min, y_min),
46             x_max - x_min,
47             y_max - y_min,
48             linewidth=1,
49             edgecolor="r",
50             facecolor="none",
51         )
52         ax.add_patch(rect)
53         plt.text(
54             x_min,
55             y_min,
56             f"{{list(class_to_idx.keys())[class_pred]}: {confidence:.2f}}",
57             color="white",
58             fontsize=12,
59             bbox=dict(facecolor="red", alpha=0.5),
60         )
61
62     plt.show()
63
64 # Load the best model
65 model.load_state_dict(torch.load("best_model.pth"))
66
67 # Inference on a sample image
68 # image_path = os.path.join(image_dir, "cat.100.jpg")
69 image_path = "/mnt/c/Study/OD Project/good_1.jpg"
70 inference(model, image_path, transform, device, class_to_idx, threshold=0.5)

```

Kết luận

Qua bài toán thứ ba, chúng ta đã mở rộng khả năng của mô hình trong việc xử lý các hình ảnh chứa nhiều object. Bằng cách kết hợp **classification** và **bounding box regression**, chúng ta có thể không chỉ phân loại các object mà còn xác định chính xác vị trí của chúng trong hình ảnh. Việc sử dụng các kỹ thuật như **patching** và **two-headed models** giúp cải thiện hiệu suất và khả năng mở rộng của hệ thống.



Hình 4: Dự đoán của mô hình sau khi huấn luyện 10 epoch.

Bài toán 4: YOLOv1

Trong bài toán thứ tư, chúng ta sẽ triển khai mô hình **YOLOv1** cho bài toán **Object Detection**. YOLOv1 là một trong những mô hình tiên tiến nhất cho việc phát hiện và định vị đối tượng trong hình ảnh với tốc độ nhanh và hiệu quả cao. Mô hình này sử dụng một mạng neural đơn giản để dự đoán các bounding box và lớp của chúng trong một lần duy nhất, giúp giảm thiểu thời gian xử lý so với các phương pháp truyền thống.

Giới thiệu về YOLOv1

YOLOv1 (You Only Look Once version 1) là một kiến trúc mạng neural được thiết kế để thực hiện **Object Detection** một cách hiệu quả. Thay vì phân chia hình ảnh thành nhiều vùng và xử lý từng vùng một cách riêng biệt như các phương pháp trước đây, YOLOv1 xử lý toàn bộ hình ảnh cùng một lúc. Điều này không chỉ tăng tốc độ xử lý mà còn cải thiện độ chính xác trong việc phát hiện và định vị các đối tượng.

Mô hình YOLOv1 được xây dựng dựa trên kiến trúc **Darknet**, một mạng convolutional mạnh mẽ với các lớp CNN sâu, kết hợp với các lớp fully connected để dự đoán các bounding box và xác suất lớp cho từng grid cell trong hình ảnh.

Import và Tải Dữ liệu

Dầu tiên, chúng ta cần import các thư viện cần thiết và tải dữ liệu từ bộ dữ liệu VOC (Visual Object Classes). Bộ dữ liệu này cung cấp các hình ảnh cùng với các annotations về các đối tượng trong hình ảnh, phù hợp cho việc huấn luyện và đánh giá mô hình **Object Detection**.

```

1 import os
2
3 import numpy as np
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 from matplotlib import patches
7 from collections import Counter
8 import cv2
9 from glob import glob
10 from tqdm import tqdm
11 from termcolor import colored
12
13 import torch
14 from torch import nn
15 from torch import optim
16 from torch.utils.data import DataLoader
17
18 import torchvision
19 from torchvision import transforms
20
21 import albumentations as A
22 from albumentations.pytorch import ToTensorV2
23

```

Định Nghĩa Dataset Custom cho YOLOv1

Chúng ta sẽ định nghĩa một lớp dataset tùy chỉnh `CustomVOCDataset` kế thừa từ `torchvision.datasets.VOCDetection`. Lớp này sẽ xử lý việc chuyển đổi annotations từ định dạng VOC sang định dạng YOLO, chuẩn bị dữ liệu cho mô hình YOLOv1.

```

1 class CustomVOCDataset(torchvision.datasets.VOCDetection):
2     def __init__(self, class_mapping, S=7, B=2, C=20, custom_transforms=None):
3         # Initialize YOLO-specific configuration parameters.
4         self.S = S # Grid size S x S
5         self.B = B # Number of bounding boxes
6         self.C = C # Number of classes
7         self.class_mapping = class_mapping # Mapping of class names to class
8         indices
9         self.custom_transforms = custom_transforms
10
11     def __getitem__(self, index):
12         # Get an image and its target (annotations) from the VOC dataset.
13         image, target = super(CustomVOCDataset, self).__getitem__(index)
14         img_width, img_height = image.size
15
16         # Convert target annotations to YOLO format bounding boxes.
17         boxes = convert_to_yolo_format(target, img_width, img_height, self.
18             class_mapping)

```

```
17
18     just_boxes = boxes[:,1:]
19     labels = boxes[:,0]
20
21     # Tranforme
22     if self.custom_transforms:
23         sample = {
24             'image': np.array(image),
25             'bboxes': just_boxes,
26             'labels': labels
27         }
28
29         sample = self.custom_transforms(**sample)
30         image = sample['image']
31         boxes = sample['bboxes']
32         labels = sample['labels']
33
34     # Create an empty label matrix for YOLO ground truth.
35     label_matrix = torch.zeros((self.S, self.S, self.C + 5 * self.B))
36
37     boxes = torch.tensor(boxes, dtype=torch.float32)
38     labels = torch.tensor(labels, dtype=torch.float32)
39     image = torch.as_tensor(image, dtype=torch.float32)
40
41     # Iterate through each bounding box in YOLO format.
42     for box, class_label in zip(boxes, labels):
43         x, y, width, height = box.tolist()
44         class_label = int(class_label)
45
46         # Calculate the grid cell (i, j) that this box belongs to.
47         i, j = int(self.S * y), int(self.S * x)
48         x_cell, y_cell = self.S * x - j, self.S * y - i
49
50         # Calculate the width and height of the box relative to the grid cell
51
52         width_cell, height_cell = (
53             width * self.S,
54             height * self.S,
55         )
56
57         # If no object has been found in this specific cell (i, j) before:
58         if label_matrix[i, j, 20] == 0:
59             # Mark that an object exists in this cell.
60             label_matrix[i, j, 20] = 1
61
62             # Store the box coordinates as an offset from the cell boundaries
63
64             box_coordinates = torch.tensor(
65                 [x_cell, y_cell, width_cell, height_cell]
66             )
67
68             # Set the box coordinates in the label matrix.
69             label_matrix[i, j, 21:25] = box_coordinates
70
71             # Set the one-hot encoding for the class label.
72             label_matrix[i, j, class_label] = 1
73
74
75     return image, label_matrix
```

Trong lớp CustomVOCdataset, chúng ta thực hiện các bước sau:

- **Khởi tạo:** Xác định các tham số đặc thù của YOLO như kích thước grid (S), số lượng bounding boxes mỗi grid cell (B), và số lớp (C).
- **Chuyển đổi annotations:** Chuyển đổi annotations từ định dạng VOC sang định dạng YOLO, bao gồm việc tính toán tọa độ trung tâm và kích thước của bounding box, cũng như ánh xạ lớp đối tượng.
- **Áp dụng biến đổi:** Sử dụng các biến đổi dữ liệu (transforms) để tăng cường dữ liệu và chuẩn bị hình ảnh cho mô hình.
- **Tạo ma trận nhãn:** Tạo một ma trận nhãn rỗng cho YOLO, sau đó điền thông tin về các bounding box và lớp đối tượng vào ma trận này.

Chuyển Đổi Annotations sang Định Dạng YOLO

Hàm convert_to_yolo_format chịu trách nhiệm chuyển đổi annotations từ định dạng VOC sang định dạng YOLO, bao gồm việc tính toán tọa độ trung tâm, chiều rộng, chiều cao của bounding box và ánh xạ lớp đối tượng.

```

1 def convert_to_yolo_format(target, img_width, img_height, class_mapping):
2     """
3         Convert annotation data from VOC format to YOLO format.
4
5     Parameters:
6         target (dict): Annotation data from VOCDetection dataset.
7         img_width (int): Width of the original image.
8         img_height (int): Height of the original image.
9         class_mapping (dict): Mapping from class names to integer IDs.
10
11    Returns:
12        torch.Tensor: Tensor of shape [N, 5] for N bounding boxes,
13                      each with [class_id, x_center, y_center, width, height].
14    """
15    # Extract the list of annotations from the target dictionary.
16    annotations = target['annotation']['object']
17
18    # Get the real width and height of the image from the annotation.
19    real_width = int(target['annotation']['size']['width'])
20    real_height = int(target['annotation']['size']['height'])
21
22    # Ensure that annotations is a list, even if there's only one object.
23    if not isinstance(annotations, list):
24        annotations = [annotations]
25
26    # Initialize an empty list to store the converted bounding boxes.
27    boxes = []
28
29    # Loop through each annotation and convert it to YOLO format.
30    for anno in annotations:
31        xmin = int(anno['bndbox']['xmin']) / real_width
32        xmax = int(anno['bndbox']['xmax']) / real_width
33        ymin = int(anno['bndbox']['ymin']) / real_height
34        ymax = int(anno['bndbox']['ymax']) / real_height
35

```

```
36     # Calculate the center coordinates, width, and height of the bounding box
37
38     x_center = (xmin + xmax) / 2
39     y_center = (ymin + ymax) / 2
40     width = xmax - xmin
41     height = ymax - ymin
42
43     # Retrieve the class name from the annotation and map it to an integer ID
44
45     class_name = anno['name']
46     class_id = class_mapping[class_name] if class_name in class_mapping else
47     0
48
49     # Append the YOLO formatted bounding box to the list.
50     boxes.append([class_id, x_center, y_center, width, height])
51
52     # Convert the list of boxes to a torch tensor.
53     return np.array(boxes)
```

Định Nghĩa Hàm Tính IoU và Non-Maximum Suppression

Các hàm `intersection_over_union` và `non_max_suppression` được sử dụng để tính toán **Intersection over Union (IoU)** giữa các bounding boxes và thực hiện **Non-Maximum Suppression (NMS)** để loại bỏ các bounding boxes trùng lặp, chỉ giữ lại các bounding boxes có độ tin cậy cao nhất.

```
1 def intersection_over_union(boxes_preds, boxes_labels, box_format="midpoint"):
2     """
3         Calculate the Intersection over Union (IoU) between bounding boxes.
4
5     Parameters:
6         boxes_preds (tensor): Predicted bounding boxes (BATCH_SIZE, 4)
7         boxes_labels (tensor): Ground truth bounding boxes (BATCH_SIZE, 4)
8         box_format (str): Box format, can be "midpoint" or "corners".
9
10    Returns:
11        tensor: Intersection over Union scores for each example.
12    """
13
14    # Check if the box format is "midpoint"
15    if box_format == "midpoint":
16        # Calculate coordinates of top-left (x1, y1) and bottom-right (x2, y2)
17        # points for predicted boxes
18        box1_x1 = boxes_preds[..., 0:1] - boxes_preds[..., 2:3] / 2
19        box1_y1 = boxes_preds[..., 1:2] - boxes_preds[..., 3:4] / 2
20        box1_x2 = boxes_preds[..., 0:1] + boxes_preds[..., 2:3] / 2
21        box1_y2 = boxes_preds[..., 1:2] + boxes_preds[..., 3:4] / 2
22
23        # Calculate coordinates of top-left (x1, y1) and bottom-right (x2, y2)
24        # points for ground truth boxes
25        box2_x1 = boxes_labels[..., 0:1] - boxes_labels[..., 2:3] / 2
26        box2_y1 = boxes_labels[..., 1:2] - boxes_labels[..., 3:4] / 2
27        box2_x2 = boxes_labels[..., 0:1] + boxes_labels[..., 2:3] / 2
28        box2_y2 = boxes_labels[..., 1:2] + boxes_labels[..., 3:4] / 2
29
30    # Check if the box format is "corners"
```

```

29     if box_format == "corners":
30         # Extract coordinates for predicted boxes
31         box1_x1 = boxes_preds[..., 0:1]
32         box1_y1 = boxes_preds[..., 1:2]
33         box1_x2 = boxes_preds[..., 2:3]
34         box1_y2 = boxes_preds[..., 3:4]
35
36         # Extract coordinates for ground truth boxes
37         box2_x1 = boxes_labels[..., 0:1]
38         box2_y1 = boxes_labels[..., 1:2]
39         box2_x2 = boxes_labels[..., 2:3]
40         box2_y2 = boxes_labels[..., 3:4]
41
42     # Calculate coordinates of the intersection rectangle
43     x1 = torch.max(box1_x1, box2_x1)
44     y1 = torch.max(box1_y1, box2_y1)
45     x2 = torch.min(box1_x2, box2_x2)
46     y2 = torch.min(box1_y2, box2_y2)
47
48     # Compute the area of the intersection rectangle, clamp(0) to handle cases
49     # where they do not overlap
50     intersection = (x2 - x1).clamp(0) * (y2 - y1).clamp(0)
51
52     # Calculate the areas of the predicted and ground truth boxes
53     box1_area = abs((box1_x2 - box1_x1) * (box1_y2 - box1_y1))
54     box2_area = abs((box2_x2 - box2_x1) * (box2_y2 - box2_y1))
55
56     # Calculate the Intersection over Union, adding a small epsilon to avoid
57     # division by zero
58     return intersection / (box1_area + box2_area - intersection + 1e-6)
59
60

```

```

1 def non_max_suppression(bboxes, iou_threshold, threshold, box_format="corners"):
2     """
3         Perform Non-Maximum Suppression on a list of bounding boxes.
4         Parameters:
5             bboxes (list): List of bounding boxes, each represented as [class_pred,
6                         prob_score, x1, y1, x2, y2].
6             iou_threshold (float): IoU threshold to determine correct predicted
7                         bounding boxes.
8             threshold (float): Threshold to discard predicted bounding boxes (
9                         independent of IoU).
10            box_format (str): "midpoint" or "corners" to specify the format of
11                         bounding boxes.
12            Returns:
13                list: List of bounding boxes after performing NMS with a specific IoU
14                         threshold.
15            """
16
17            # Check the data type of the input parameter
18            assert type(bboxes) == list
19
20            # Filter predicted bounding boxes based on probability threshold
21            bboxes = [box for box in bboxes if box[1] > threshold]
22
23            # Sort bounding boxes by probability in descending order
24            bboxes = sorted(bboxes, key=lambda x: x[1], reverse=True)
25
26

```

```
21 # List to store bounding boxes after NMS
22 bboxes_after_nms = []
23
24
25 # Continue looping until the list of bounding boxes is empty
26 while bboxes:
27     # Get the bounding box with the highest probability
28     chosen_box = bboxes.pop(0)
29
30     # Remove bounding boxes with IoU greater than the specified threshold
31     # with the chosen box
32     bboxes = [
33         box
34         for box in bboxes
35         if box[0] != chosen_box[0]
36         or intersection_over_union(
37             torch.tensor(chosen_box[2:]),
38             torch.tensor(box[2:]),
39             box_format=box_format,
40         )
41         < iou_threshold
42     ]
43
44     # Add the chosen bounding box to the list after NMS
45     bboxes_after_nms.append(chosen_box)
46
47 # Return the list of bounding boxes after NMS
48 return bboxes_after_nms
```

Tính Toán Trung Bình Độ Chính Xác (mAP)

Hàm `mean_average_precision` được sử dụng để tính toán giá trị trung bình của độ chính xác (mAP) cho mô hình, một chỉ số quan trọng đánh giá hiệu quả của các mô hình **Object Detection**.

```
1 def mean_average_precision(
2     pred_boxes, true_boxes, iou_threshold=0.5, box_format="midpoint", num_classes
3     =20):
4     """
5         Calculate the mean average precision (mAP).
6
7         Parameters:
8             pred_boxes (list): A list containing predicted bounding boxes with each
9                 box defined as [train_idx, class_pred, prob_score, x1, y1, x2, y2].
10            true_boxes (list): Similar to pred_boxes but containing information about
11                true boxes.
12            iou_threshold (float): IoU threshold, where predicted boxes are
13                considered correct.
14            box_format (str): "midpoint" or "corners" used to specify the format of
15                the boxes.
16            num_classes (int): Number of classes.
17
18        Returns:
19            float: The mAP value across all classes with a specific IoU threshold.
20        """
21
```

```
17     # List to store mAP for each class
18     average_precisions = []
19
20     # Small epsilon to stabilize division
21     epsilon = 1e-6
22
23     for c in range(num_classes):
24         detections = []
25         ground_truths = []
26
27         # Iterate through all predictions and targets, and only add those
28         # belonging to
29         # the current class 'c'.
30         for detection in pred_boxes:
31             if detection[1] == c:
32                 detections.append(detection)
33
34         for true_box in true_boxes:
35             if true_box[1] == c:
36                 ground_truths.append(true_box)
37
38         # Find the number of boxes for each training example.
39         # The Counter here counts the number of target boxes we have
40         # for each training example, so if image 0 has 3, and image 1 has 5,
41         # we'll have a dictionary like:
42         # amount_bboxes = {0: 3, 1: 5}
43         amount_bboxes = Counter([gt[0] for gt in ground_truths])
44
45         # We then loop through each key, val in this dictionary and convert it to
46         # the following (for the same example):
47         # amount_bboxes = {0: torch.tensor([0, 0, 0]), 1: torch.tensor([0, 0, 0,
48         0, 0])}
49         for key, val in amount_bboxes.items():
50             amount_bboxes[key] = torch.zeros(val)
51
52         # Sort by box probability, index 2 is the probability
53         detections.sort(key=lambda x: x[2], reverse=True)
54         TP = torch.zeros((len(detections)))
55         FP = torch.zeros((len(detections)))
56         total_true_bboxes = len(ground_truths)
57
58         # If there are no ground truth boxes for this class, it can be safely
59         # skipped
60         if total_true_bboxes == 0:
61             continue
62
63         for detection_idx, detection in enumerate(detections):
64             # Only consider ground truth boxes with the same training index as
65             # the prediction
66             ground_truth_img = [
67                 bbox for bbox in ground_truths if bbox[0] == detection[0]
68             ]
69
70             num_gts = len(ground_truth_img)
71             best_iou = 0
72
73             for idx, gt in enumerate(ground_truth_img):
74                 iou = intersection_over_union(
75                     torch.tensor(detection[3:]),
76                     torch.tensor(gt[3:]),
77                     epsilon=epsilon)
78
79                 if iou > best_iou:
80                     best_iou = iou
81
82             if best_iou > 0:
83                 TP[detection_idx] += 1
84                 FP[idx] += 1
85
86             else:
87                 FP[idx] += 1
88
89         precision = TP / (TP + FP)
90         recall = TP / total_true_bboxes
91
92         if len(detections) > 0:
93             average_precisions.append(precision * recall)
94
95     average_precision = sum(average_precisions) / len(average_precisions)
96
97     return average_precision
```

```

71             torch.tensor(gt[3:]),
72             box_format=box_format,
73         )
74
75         if iou > best_iou:
76             best_iou = iou
77             best_gt_idx = idx
78
79         if best_iou > iou_threshold:
80             # Only detect ground truth once
81             if amount_bboxes[detection[0]][best_gt_idx] == 0:
82                 # True positive and mark this bounding box as seen
83                 TP[detection_idx] = 1
84                 amount_bboxes[detection[0]][best_gt_idx] = 1
85             else:
86                 FP[detection_idx] = 1
87
88         # If IOU is lower, the detection result is false positive
89     else:
90         FP[detection_idx] = 1
91
92     TP_cumsum = torch.cumsum(TP, dim=0)
93     FP_cumsum = torch.cumsum(FP, dim=0)
94     recalls = TP_cumsum / (total_true_bboxes + epsilon)
95     precisions = torch.divide(TP_cumsum, (TP_cumsum + FP_cumsum + epsilon))
96     precisions = torch.cat((torch.tensor([1]), precisions))
97     recalls = torch.cat((torch.tensor([0]), recalls))
98     # Use torch.trapz for numerical integration
99     average_precisions.append(torch.trapz(precisions, recalls))
100
101 return sum(average_precisions) / len(average_precisions)
102

```

Định Nghĩa Kiến Trúc Mạng YOLOv1

Mô hình YOLOv1 được xây dựng dựa trên kiến trúc Darknet, bao gồm các lớp convolutional và max-pooling, kết hợp với các lớp fully connected để dự đoán bounding boxes và lớp đối tượng.

```

1 """
2 Information about the architectural configuration:
3 A Tuple is structured as (kernel_size, number of filters, stride, padding).
4 "M" simply represents max-pooling with a 2x2 pool size and 2x2 kernel.
5 The list is structured according to the data blocks, and ends with an integer
       representing the number of repetitions.
6 """
7
8 # Describing convolutional and max-pooling layers, as well as the number of
   repetitions for convolutional blocks.
9 architecture_config = [
10     (7, 64, 2, 3),  # Convolutional block 1
11     "M",           # Max-pooling layer 1
12     (3, 192, 1, 1), # Convolutional block 2
13     "M",           # Max-pooling layer 2
14     (1, 128, 1, 0), # Convolutional block 3
15     (3, 256, 1, 1), # Convolutional block 4
16     (1, 256, 1, 0), # Convolutional block 5

```

```
17     (3, 512, 1, 1), # Convolutional block 6
18     "M",           # Max-pooling layer 3
19     [(1, 256, 1, 0), (3, 512, 1, 1), 4], # Convolutional block 7 (repeated 4
times)
20     (1, 512, 1, 0), # Convolutional block 8
21     (3, 1024, 1, 1),# Convolutional block 9
22     "M",           # Max-pooling layer 4
23     [(1, 512, 1, 0), (3, 1024, 1, 1), 2], # Convolutional block 10 (repeated 2
times)
24     (3, 1024, 1, 1),# Convolutional block 11
25     (3, 1024, 2, 1),# Convolutional block 12
26     (3, 1024, 1, 1),# Convolutional block 13
27     (3, 1024, 1, 1),# Convolutional block 14
28 ]
29
30 # A convolutional block is defined with Conv2d, BatchNorm2d, and LeakyReLU layers
.
31 class CNNBlock(nn.Module):
32     def __init__(self, in_channels, out_channels, **kwargs):
33         super(CNNBlock, self).__init__()
34         self.conv = nn.Conv2d(in_channels, out_channels, bias=False, **kwargs)
35         self.batchnorm = nn.BatchNorm2d(out_channels)
36         self.leakyrelu = nn.LeakyReLU(0.1)
37
38     def forward(self, x):
39         return self.leakyrelu(self.batchnorm(self.conv(x)))
40
41 # The YOLOv1 model is defined with convolutional layers and fully connected
layers (fcs).
42 class Yolov1(nn.Module):
43     def __init__(self, in_channels=3, **kwargs):
44         super(Yolov1, self).__init__()
45         self.architecture = architecture_config
46         self.in_channels = in_channels
47         self.darknet = self._create_conv_layers(self.architecture)
48         self.fcs = self._create_fcs(**kwargs)
49
50     def forward(self, x):
51         x = self.darknet(x)
52         return self.fcs(torch.flatten(x, start_dim=1))
53
54     # Function to create convolutional layers based on the predefined
architecture.
55     def _create_conv_layers(self, architecture):
56         layers = []
57         in_channels = self.in_channels
58
59         for x in architecture:
60             if type(x) == tuple:
61                 layers += [
62                     CNNBlock(
63                         in_channels, x[1], kernel_size=x[0], stride=x[2], padding
=x[3],
64                         )
65                     ]
66                     in_channels = x[1]
67
68             elif type(x) == str:
69                 layers += [nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2))]
```

```

70
71     elif type(x) == list:
72         conv1 = x[0]
73         conv2 = x[1]
74         num_repeats = x[2]
75
76         for _ in range(num_repeats):
77             layers += [
78                 CNNBlock(
79                     in_channels,
80                     conv1[1],
81                     kernel_size=conv1[0],
82                     stride=conv1[2],
83                     padding=conv1[3],
84                 )
85             ]
86             layers += [
87                 CNNBlock(
88                     conv1[1],
89                     conv2[1],
90                     kernel_size=conv2[0],
91                     stride=conv2[2],
92                     padding=conv2[3],
93                 )
94             ]
95             in_channels = conv2[1]
96
97     return nn.Sequential(*layers)
98
99 # Function to create fully connected layers based on input parameters such as
100 # grid size, number of boxes, and number of classes.
101 def _create_fcs(self, split_size, num_boxes, num_classes):
102     S, B, C = split_size, num_boxes, num_classes
103
104     return nn.Sequential(
105         nn.Flatten(),
106         nn.Linear(1024 * S * S, 4096),
107         nn.Dropout(0.0),
108         nn.LeakyReLU(0.1),
109         nn.Linear(4096, S * S * (C + B * 5)),
110     )

```

Trong kiến trúc này:

- **Convolutional Layers:** Các lớp convolutional được xây dựng theo cấu hình đã định nghĩa trong `architecture_config`, bao gồm các lớp `CNNBlock` kết hợp với `BatchNorm2d` và `LeakyReLU`.
- **Fully Connected Layers:** Sau các lớp convolutional, các lớp fully connected được sử dụng để dự đoán các bounding boxes và lớp của chúng. Lớp đầu tiên `Linear` kết nối từ các đặc trưng đã trích xuất sang không gian cao hơn, sau đó sử dụng lớp `Dropout` để giảm thiểu overfitting và lớp `LeakyReLU` cho phép gradient flow tốt hơn.

Định Nghĩa Hàm Mất Của YOLOv1

Hàm mất (YoloLoss) được thiết kế để tối ưu hóa các dự đoán của mô hình YOLOv1, bao gồm việc tính toán mất mát cho các bounding boxes, xác suất đối tượng và lớp của đối tượng.

```
1 class YoloLoss(nn.Module):
2     """
3         Calculate the loss for the YOLO (v1) model.
4     """
5
6     def __init__(self, S=7, B=2, C=20):
7         super(YoloLoss, self).__init__()
8         self.mse = nn.MSELoss(reduction="sum")
9
10    """
11        S is the grid size of the image (7),
12        B is the number of bounding boxes (2),
13        C is the number of classes (in VOC dataset, it's 20).
14    """
15    self.S = S
16    self.B = B
17    self.C = C
18
19    # These are YOLO-specific constants, representing the weight
20    # for no object loss (lambda_noobj) and box coordinates loss (
21    # lambda_coord).
22    self.lambda_noobj = 0.5
23    self.lambda_coord = 5
24
25    def forward(self, predictions, target):
26        # Reshape the predictions to the shape (BATCH_SIZE, S*S(C+B*5))
27        predictions = predictions.reshape(-1, self.S, self.S, self.C + self.B *
28                                         5)
29
30        # Calculate Intersection over Union (IoU) for the two predicted bounding
31        # boxes
32        # with the target bounding box.
33        iou_b1 = intersection_over_union(predictions[:, :, 21:25], target[:, :,
34                                         21:25])
35        iou_b2 = intersection_over_union(predictions[:, :, 26:30], target[:, :,
36                                         21:25])
37        ious = torch.cat([iou_b1.unsqueeze(0), iou_b2.unsqueeze(0)], dim=0)
38
39        # Get the box with the highest IoU among the two predictions.
40        # Note that bestbox will have an index of 0 or 1, indicating which box is
41        # better.
42        iou_maxes, bestbox = torch.max(ious, dim=0)
43        exists_box = target[:, :, 20].unsqueeze(3) # This represents Iobj_i in
44        # the paper
45
46        # ===== #
47        # FOR BOX COORDINATES #
48        # ===== #
49
50        # Set the boxes with no objects to zero. Choose one of the two
51        # predictions
52        # based on the bestbox index calculated earlier.
53        box_predictions = exists_box * (
54            (
55                bestbox * predictions[:, :, 26:30]
56                + (1 - bestbox) * predictions[:, :, 21:25]
57            )
58        )
```

```
51     box_targets = exists_box * target[..., 21:25]
52
53     # Take the square root of width and height to ensure positive values.
54     box_predictions[..., 2:4] = torch.sign(box_predictions[..., 2:4]) * torch
55     .sqrt(
56         torch.abs(box_predictions[..., 2:4] + 1e-6)
57     )
58     box_targets[..., 2:4] = torch.sqrt(box_targets[..., 2:4])
59
60     box_loss = self.mse(
61         torch.flatten(box_predictions, end_dim=-2),
62         torch.flatten(box_targets, end_dim=-2),
63     )
64
65     # ===== #
66     # FOR OBJECT LOSS #
67     # ===== #
68
69     # pred_box represents the confidence score of the box with the highest
70     # IoU.
71     pred_box = (
72         bestbox * predictions[..., 25:26] + (1 - bestbox) * predictions[...,
73         20:21]
74     )
75
76     object_loss = self.mse(
77         torch.flatten(exists_box * pred_box),
78         torch.flatten(exists_box * target[..., 20:21]),
79     )
80
81     # ===== #
82     # FOR NO OBJECT LOSS #
83     # ===== #
84
85     no_object_loss = self.mse(
86         torch.flatten((1 - exists_box) * predictions[..., 20:21], start_dim
87         =1),
88         torch.flatten((1 - exists_box) * target[..., 20:21], start_dim=1),
89     )
90
91     no_object_loss += self.mse(
92         torch.flatten((1 - exists_box) * predictions[..., 25:26], start_dim
93         =1),
94         torch.flatten((1 - exists_box) * target[..., 20:21], start_dim=1)
95     )
96
97     # ===== #
98     # FOR CLASS LOSS #
99     # ===== #
100
101    class_loss = self.mse(
102        torch.flatten(exists_box * predictions[..., :20], end_dim=-2, ),
103        torch.flatten(exists_box * target[..., :20], end_dim=-2, ),
104    )
105
106    # Calculate the final loss by combining the above components.
107    loss = (
108        self.lambda_coord * box_loss # First term
```

```
105         + object_loss # Second term
106         + self.lambda_noobj * no_object_loss # Third term
107         + class_loss # Fourth term
108     )
109
110     return loss
111
```

Cài Đặt và Huấn Luyện Mô Hình YOLOv1

Chúng ta sẽ cài đặt mô hình YOLOv1, thiết lập các siêu tham số, và tiến hành huấn luyện mô hình trên bộ dữ liệu VOC.

```
1 # Set the random seed for reproducibility.
2 seed = 123
3 torch.manual_seed(seed)
4
5 # Hyperparameters and configurations
6 # Learning rate for the optimizer.
7 LEARNING_RATE = 2e-5
8 # Specify whether to use "cuda" (GPU) or "cpu" for training.
9 DEVICE = "cuda"
10 # Originally 64 in the research paper, but using a smaller batch size due to GPU
    limitations.
11 BATCH_SIZE = 16
12 # Number of training epochs.
13 EPOCHS = 300
14 # Number of worker processes for data loading.
15 NUM_WORKERS = 2
16 # If True, DataLoader will pin memory to transfer data to the GPU faster.
17 PIN_MEMORY = True
18 # If False, the training process will not load a pre-trained model.
19 LOAD_MODEL = False
20 # Specify the file name for the pre-trained model if LOAD_MODEL is True.
21 LOAD_MODEL_FILE = "yolov1.pth.tar"
22
```

Định Nghĩa Các Biến Đổi Dữ Liệu

Chúng ta sẽ sử dụng thư viện `albumentations` để áp dụng các biến đổi dữ liệu nhằm tăng cường dữ liệu và cải thiện khả năng tổng quát hóa của mô hình.

```
1 WIDTH = 448
2 HEIGHT = 448
3
4 def get_train_transforms():
5     return A.Compose([
6         A.OneOf([
7             A.HueSaturationValue(hue_shift_limit=0.2,
8                 sat_shift_limit= 0.2, val_shift_limit=0.2, p=0.9),
9             A.RandomBrightnessContrast(brightness_limit=0.2,
10                 contrast_limit=0.2, p=0.9)], p=0.9),
11         A.ToGray(p=0.01),
12         A.HorizontalFlip(p=0.2),
13         A.VerticalFlip(p=0.2),
```

```

10             A.Resize(height=WIDTH, width=WIDTH, p=1),
11             #   A.Cutout(num_holes=8, max_h_size=64, max_w_size=64,
12             fill_value=0, p=0.5),
13             ToTensorV2(p=1.0)],
14             p=1.0,
15             bbox_params=A.BboxParams(format='yolo', min_area=0,
16             min_visibility=0, label_fields=['labels'])
17     )
18
19 def get_valid_transforms():
20     return A.Compose([A.Resize(height=WIDTH, width=WIDTH, p=1.0),
21                     ToTensorV2(p=1.0)],
22                     p=1.0,
23                     bbox_params=A.BboxParams(format='yolo', min_area=0,
24                     min_visibility=0, label_fields=['labels'])
25     )

```

Định Nghĩa Mapping Lớp Đôi Tượng

Chúng ta định nghĩa một từ điển `class_mapping` để ánh xạ các tên lớp đối tượng trong bộ dữ liệu VOC sang các chỉ số số học, giúp dễ dàng xử lý trong quá trình huấn luyện.

```

1 class_mapping = {
2     'aeroplane': 0,
3     'bicycle': 1,
4     'bird': 2,
5     'boat': 3,
6     'bottle': 4,
7     'bus': 5,
8     'car': 6,
9     'cat': 7,
10    'chair': 8,
11    'cow': 9,
12    'diningtable': 10,
13    'dog': 11,
14    'horse': 12,
15    'motorbike': 13,
16    'person': 14,
17    'pottedplant': 15,
18    'sheep': 16,
19    'sofa': 17,
20    'train': 18,
21    'tvmonitor': 19
22 }
23

```

Định Nghĩa Hàm Huấn Luyện và Đánh Giá

Chúng ta sẽ định nghĩa các hàm `train_fn` và `test_fn` để thực hiện quá trình huấn luyện và đánh giá mô hình trên tập dữ liệu huấn luyện và tập dữ liệu kiểm tra.

```
1 def train_fn(train_loader, model, optimizer, loss_fn, epoch):
2     mean_loss = []
3     mean_mAP = []
4
5     total_batches = len(train_loader)
6     display_interval = total_batches // 5 # Update after 20% of the total
7     batches.
8
9     for batch_idx, (x, y) in enumerate(train_loader):
10         x, y = x.to(DEVICE), y.to(DEVICE)
11         out = model(x)
12         loss = loss_fn(out, y)
13
14         optimizer.zero_grad()
15         loss.backward()
16         optimizer.step()
17
18         pred_boxes, true_boxes = get_bboxes_training(out, y, iou_threshold=0.5,
19             threshold=0.4)
20         mAP = mean_average_precision(pred_boxes, true_boxes, iou_threshold=0.5,
21             box_format="midpoint")
22
23         if batch_idx % display_interval == 0 or batch_idx == total_batches - 1:
24             print(f"Epoch: {epoch:3} \t Iter: {batch_idx:3}/{total_batches:3} \t
25             Loss: {loss.item():3.10f} \t mAP: {mAP.item():3.10f}")
26
27         avg_loss = sum(mean_loss) / len(mean_loss)
28         avg_mAP = sum(mean_mAP) / len(mean_mAP)
29         print(colored(f"Train \t loss: {avg_loss:3.10f} \t mAP: {avg_mAP:3.10f}", 'green'))
30
31     return avg_mAP
32
33 def test_fn(test_loader, model, loss_fn, epoch):
34     model.eval()
35     mean_loss = []
36     mean_mAP = []
37
38     for batch_idx, (x, y) in enumerate(test_loader):
39         x, y = x.to(DEVICE), y.to(DEVICE)
40         out = model(x)
41         loss = loss_fn(out, y)
42
43         pred_boxes, true_boxes = get_bboxes_training(out, y, iou_threshold=0.5,
44             threshold=0.4)
45         mAP = mean_average_precision(pred_boxes, true_boxes, iou_threshold=0.5,
46             box_format="midpoint")
47
48         mean_loss.append(loss.item())
49         mean_mAP.append(mAP.item())
50
51     avg_loss = sum(mean_loss) / len(mean_loss)
52     avg_mAP = sum(mean_mAP) / len(mean_mAP)
53     print(colored(f"Test \t loss: {avg_loss:3.10f} \t mAP: {avg_mAP:3.10f}", 'yellow'))
```

```
51     model.train()
52
53     return avg_mAP
54
55
```

Định Nghĩa Hàm Main để Huấn Luyện Mô Hình

Chúng ta sẽ định nghĩa hàm `train` để khởi tạo mô hình, thiết lập các lớp dữ liệu, và tiến hành huấn luyện mô hình YOLOv1 trên bộ dữ liệu VOC.

```
1 # Main function
2 def train():
3     # Initialize model, optimizer, loss
4     model = Yolov1(split_size=7, num_boxes=2, num_classes=20).to(DEVICE)
5     optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
6     loss_fn = YoloLoss()
7
8     # Load checkpoint if necessary
9     if LOAD_MODEL:
10         load_checkpoint(torch.load(LOAD_MODEL_FILE), model, optimizer)
11
12     # Create the full dataset
13     train_dataset = CustomVOCDataset(
14         root='./data',
15         year='2012',
16         image_set='train',
17         download=True,
18     )
19
20     train_dataset.init_config_yolo(class_mapping=class_mapping, custom_transforms=
21     =get_train_transforms())
22
23     testval_dataset = CustomVOCDataset(
24         root='./data',
25         year='2012',
26         image_set='val',
27         download=True,
28     )
29
30     testval_dataset.init_config_yolo(class_mapping=class_mapping,
31     custom_transforms=get_val_transforms())
32
33     # Split dataset into train, validation, and test sets using indices
34     dataset_size = len(testval_dataset)
35     val_size = int(0.15 * dataset_size)
36     test_size = dataset_size - val_size
37
38     val_indices = list(range(val_size))
39     test_indices = list(range(val_size, val_size + test_size))
40
41     # Create SubsetRandomSamplers
42     val_sampler = SubsetRandomSampler(val_indices)
43     test_sampler = SubsetRandomSampler(test_indices)
44
45     # Create DataLoaders using the samplers
```

```
44     train_loader = DataLoader(
45         dataset=train_dataset,
46         batch_size=BATCH_SIZE,
47         num_workers=NUM_WORKERS,
48         pin_memory=PIN_MEMORY,
49         drop_last=True,
50     )
51
52     val_loader = DataLoader(
53         dataset=testval_dataset,
54         batch_size=BATCH_SIZE,
55         num_workers=NUM_WORKERS,
56         pin_memory=PIN_MEMORY,
57         sampler=val_sampler, # Use the sampler here
58         drop_last=False,
59     )
60
61     test_loader = DataLoader(
62         dataset=testval_dataset,
63         batch_size=BATCH_SIZE,
64         num_workers=NUM_WORKERS,
65         pin_memory=PIN_MEMORY,
66         sampler=test_sampler, # Use the sampler here
67         drop_last=False,
68     )
69
70     best_mAP_train = 0
71     best_mAP_val = 0
72     best_mAP_test = 0
73
74     # Training loop
75     for epoch in range(EPOCHS):
76         train_mAP = train_fn(train_loader, model, optimizer, loss_fn, epoch)
77         val_mAP = val_test_fn(val_loader, model, loss_fn, epoch)
78         test_mAP = val_test_fn(test_loader, model, loss_fn, epoch, is_test=True)
79         # Pass is_test=True for test set
80
81         # Update best mAP values
82         if train_mAP > best_mAP_train:
83             best_mAP_train = train_mAP
84         if val_mAP > best_mAP_val:
85             best_mAP_val = val_mAP
86         # Save checkpoint when validation mAP improves
87         checkpoint = {
88             "state_dict": model.state_dict(),
89             "optimizer": optimizer.state_dict(),
90         }
91         save_checkpoint(checkpoint, filename=LOAD_MODEL_FILE)
92         if test_mAP > best_mAP_test:
93             best_mAP_test = test_mAP
94
95     print(colored(f"Best Train mAP: {best_mAP_train:.3f}", 'green'))
96     print(colored(f"Best Val mAP: {best_mAP_val:.3f}", 'blue'))
97     print(colored(f"Best Test mAP: {best_mAP_test:.3f}", 'yellow'))
```

Hàm test

Sau khi huấn luyện, chúng ta có thể sử dụng hàm `test` để thực hiện dự đoán trên các hình ảnh mới, hiển thị các bounding boxes và lớp đối tượng được phát hiện.

```
 1 def plot_image_with_labels(image, ground_truth_boxes, predicted_boxes,
 2     class_mapping):
 3     """Draw both ground truth and predicted bounding boxes on an image, with
 4     labels."""
 5
 6     # Inverting the class mapping for easy access of class names based on indices
 7     inverted_class_mapping = {v: k for k, v in class_mapping.items()}
 8
 9     # Convert the image to a numpy array and get its dimensions
10    im = np.array(image)
11    height, width, _ = im.shape
12
13    # Create a figure and axis for plotting
14    fig, ax = plt.subplots(1)
15    # Display the image
16    ax.imshow(im)
17
18    # Plot each ground truth box in green
19    for box in ground_truth_boxes:
20        # Extract label index and bounding box coordinates
21        label_index, box = box[0], box[2:]
22        # Calculate upper left coordinates
23        upper_left_x = box[0] - box[2] / 2
24        upper_left_y = box[1] - box[3] / 2
25        # Create a rectangle patch
26        rect = patches.Rectangle(
27            (upper_left_x * width, upper_left_y * height),
28            box[2] * width,
29            box[3] * height,
30            linewidth=1,
31            edgecolor="green",
32            facecolor="none",
33        )
34        # Add the rectangle to the plot
35        ax.add_patch(rect)
36        # Retrieve the class name and add it as text to the plot
37        class_name = inverted_class_mapping.get(label_index, "Unknown")
38        ax.text(upper_left_x * width, upper_left_y * height, class_name, color='white',
39                fontsize=12, bbox=dict(facecolor='green', alpha=0.2))
40
41    # Plot each predicted box in red
42    for box in predicted_boxes:
43        # Similar processing as for ground truth boxes
44        label_index, box = box[0], box[2:]
45        upper_left_x = box[0] - box[2] / 2
46        upper_left_y = box[1] - box[3] / 2
47        rect = patches.Rectangle(
48            (upper_left_x * width, upper_left_y * height),
49            box[2] * width,
50            box[3] * height,
```

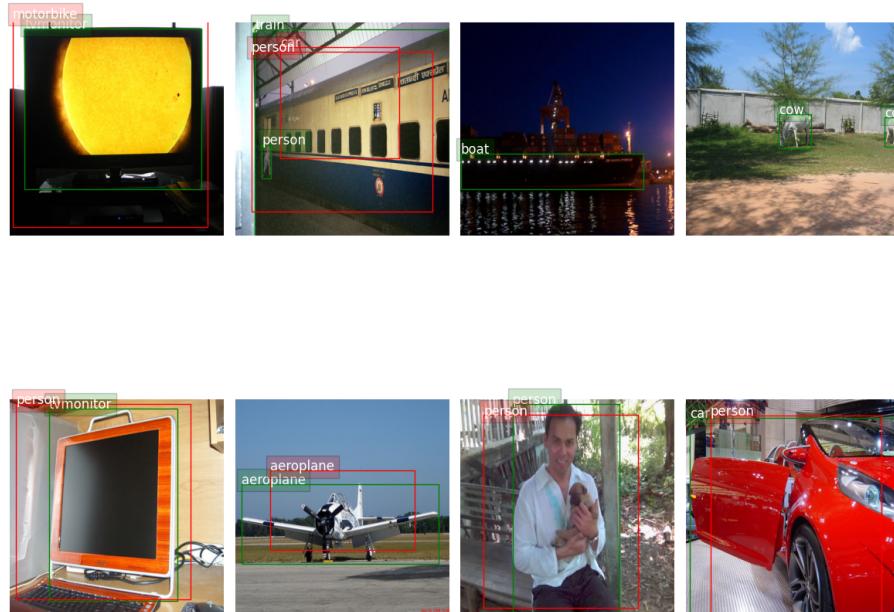
```

51
52     ax.add_patch(rect)
53     class_name = inverted_class_mapping.get(label_index, "Unknown")
54     ax.text(upper_left_x * width, upper_left_y * height, class_name, color='white', fontsize=12, bbox=dict(facecolor='red', alpha=0.2))
55
56 plt.show()
57
58 def test():
59     # Create a YOLO model object with specific hyperparameters.
60     model = Yolov1(split_size=7, num_boxes=2, num_classes=20).to(DEVICE)
61
62     # Load saved model weights and optimizer information from a file, if applicable.
63     if LOAD_MODEL:
64         model.load_state_dict(torch.load(LOAD_MODEL_FILE)['state_dict'])
65
66     # Prepare the test dataset and DataLoader for model evaluation
67     test_dataset = CustomVOCDataset(root='./data', image_set='val', download=False)
68     test_dataset.init_config_yolo(class_mapping=class_mapping, custom_transforms=get_valid_transforms())
69     test_loader = DataLoader(dataset=test_dataset, batch_size=BATCH_SIZE,
70                             num_workers=NUM_WORKERS, pin_memory=PIN_MEMORY,
71                             shuffle=False, drop_last=False)
72
73     model.eval()
74     # Iterate over the test dataset and process each batch
75     for x, y in test_loader:
76         x = x.to(DEVICE)
77         out = model(x)
78
79         # Convert model output to bounding boxes and apply non-max suppression
80         pred_bboxes = cellboxes_to_boxes(out)
81         gt_bboxes = cellboxes_to_boxes(y)
82
83         # Plot the first 8 images with their ground truth and predicted bounding
84         # boxes
85         for idx in range(8):
86             pred_box = non_max_suppression(pred_bboxes[idx], iou_threshold=0.5,
87             threshold=0.4, box_format="midpoint")
88             gt_box = non_max_suppression(gt_bboxes[idx], iou_threshold=0.5,
89             threshold=0.4, box_format="midpoint")
90
91             image = x[idx].permute(1,2,0).to("cpu")/255
92             plot_image_with_labels(image, gt_box, pred_box, class_mapping)
93
94         break # Stop after processing the first batch
95

```

Kết luận

Qua bài toán thử tư, chúng ta đã hoàn thiện quá trình xây dựng và huấn luyện mô hình **YOLOv1** cho bài toán **Object Detection**. Mô hình này không chỉ phân loại các đối tượng trong hình ảnh mà còn định vị chính xác vị trí của chúng bằng các bounding boxes. Việc áp dụng các kỹ thuật như Non-Maximum Suppression và tính toán **mean Average Precision (mAP)** giúp cải thiện độ chính xác và hiệu quả của hệ thống.



Hình 5: Kết quả dự đoán của mô hình YOLOv1 sau khi train 10 epoch. Kết quả đúng có đường viền màu xanh, kết quả dự đoán có đường viền màu đỏ.

Mô hình YOLOv1 đã chứng minh được khả năng phát hiện đối tượng nhanh chóng và chính xác, mở đường cho các phiên bản tiếp theo như YOLOv2, YOLOv3, và YOLOv4 với nhiều cải tiến hơn nữa về kiến trúc và hiệu suất.

Phần III: Câu hỏi trắc nghiệm

1. **Object Detection** nhằm mục đích gì?
 - (a) Phân loại hình ảnh.
 - (b) Nhận diện và định vị các object trong hình ảnh.
 - (c) Phân tích dữ liệu.
 - (d) Xử lý ngôn ngữ tự nhiên.
2. Một trong những mô hình phổ biến nhất cho **Object Detection** là:
 - (a) YOLO (You Only Look Once).
 - (b) ResNet50.
 - (c) BERT.
 - (d) GPT-4.
3. Trong **Object Detection**, **bounding box** được sử dụng để:
 - (a) Phân loại object.
 - (b) Tăng độ phân giải của hình ảnh.
 - (c) Xử lý màu sắc của hình ảnh.
 - (d) Định vị vị trí của object trong hình ảnh.
4. **Intersection over Union (IoU)** được sử dụng để:
 - (a) Đánh giá hiệu suất của mô hình phân loại.
 - (b) Đo lường độ chính xác của **bounding box**.
 - (c) Tăng tốc độ huấn luyện.
 - (d) Giảm thiểu độ lệch dữ liệu.
5. Một thách thức chính trong **Object Detection** là:
 - (a) Xử lý ngôn ngữ tự nhiên.
 - (b) Phân loại văn bản.
 - (c) Nhận diện các object trong điều kiện ánh sáng kém.
 - (d) Tăng độ phân giải của hình ảnh.
6. **Non-Maximum Suppression (NMS)** được sử dụng để:
 - (a) Tăng độ chính xác của phân loại.
 - (b) Loại bỏ các **bounding box** trùng lặp.
 - (c) Tối ưu hóa hàm loss.
 - (d) Điều chỉnh tỷ lệ học.
7. **Ý tưởng chính của YOLO V1 là gì?**
 - (a) Sử dụng một mô hình CNN để dự đoán box và score của các objects trong ảnh.
 - (b) Sử dụng nhiều mô hình CNN để dự đoán box và score của các objects trong ảnh.
 - (c) Sử dụng mô hình RNN để dự đoán box và score của các objects trong ảnh.

(d) Sử dụng một mô hình MLP để dự đoán box và score của các objects trong ảnh.

8. YOLO V1 chia hình ảnh đầu vào như thế nào?

- (a) Chia thành một lưới các vùng chồng chéo (Grid of overlapping regions).
- (b) Chia thành một lưới các vùng không chồng chéo.
- (c) Chia thành một vùng đơn.
- (d) Chia thành nhiều vùng dựa trên số lượng objects trong ảnh.

9. Nhược điểm của YOLO V1 là gì?

- (a) Không thể phát hiện được các objects nhỏ.
- (b) Không có khả năng xử lý đa objects trong cùng một vùng.
- (c) Không có khả năng xử lý các objects có kích thước khác nhau.
- (d) Tất cả các ý trên.

10. Sự khác nhau giữa tăng cường dữ liệu và tiền xử lý dữ liệu là gì?

- (a) Tăng cường dữ liệu dùng để tăng kích thước tập dữ liệu trong khi tiền xử lý dữ liệu dùng để làm sạch dữ liệu.
- (b) Tăng cường dữ liệu dùng để cải thiện chất lượng model trong khi tiền xử lý dữ liệu dùng để giảm overfitting.
- (c) Tăng cường dữ liệu dùng để tăng kích thước tập dữ liệu trong khi tiền xử lý dữ liệu dùng để chuẩn bị dữ liệu cho giai đoạn training.
- (d) Tăng cường dữ liệu dùng để giảm overfitting trong khi tiền xử lý dữ liệu dùng để tăng chất lượng model.

1 Phụ lục

1. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần bài tập, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

2. **Rubric:**

Object Detection Exercise - Rubric		
Câu	Kiến Thức	Dánh Giá
II.2.	<ul style="list-style-type: none"> - Kiến thức về YOLOv1. - Kiến thức về cách huấn luyện YOLOv1 cho bài toán Object Detection. 	<ul style="list-style-type: none"> - Biết cách ứng dụng YOLOv1 để xây dựng mô hình Object Detection.
II.3.	<ul style="list-style-type: none"> - Kiến thức về Convolutional Neural Networks (CNN). - Kiến thức về Classification và Localization trong Object Detection. - Khả năng xây dựng mô hình CNN sử dụng PyTorch cho bài toán Classification và Localization. 	<ul style="list-style-type: none"> - Hiểu được các khái niệm liên quan đến mô hình CNN và các nhiệm vụ Classification và Localization. - Khả năng lập trình và ứng dụng mô hình CNN vào giải quyết bài toán Classification và Localization trong Object Detection.

- *Hết* -