HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BỘ MÔN THỰC TẬP CƠ SỞ

THỰC TẬP CƠ SỞ

Giảng viên hướng dẫn	**: KIM NGỌC BÁCH**
Họ và tên sinh viên	**: NGUYỄN XUÂN HẢI**
Mã sinh viên	**: B22DCCN271**
Lớp	**: D22CQCN07-B**
Nhóm	**: 13**

Báo Cáo Hàng Tuần Lần 5 ngày (6-12/4/2025)

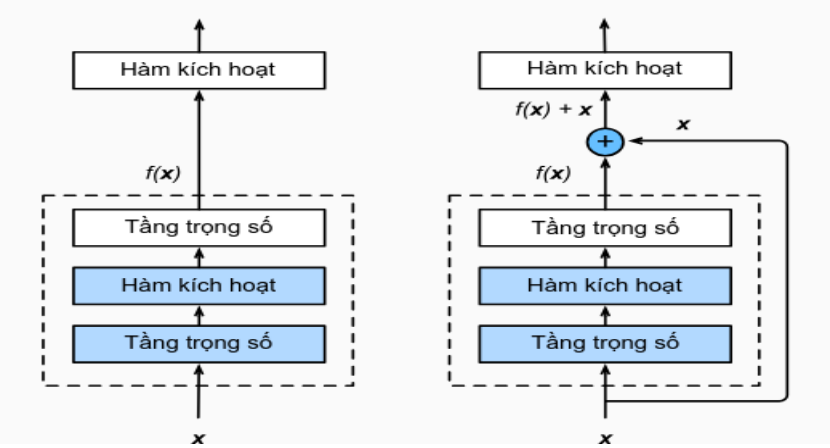
Hà Nội - 2025

I. NHIỆM VỤ CỦA TUẦN NÀY

1. Xây dựng mô hình

Có 2 mô hình CNN nổi tiếng và nhiều chương trình dùng hiện nay là RESNET và EFFICENT NET

a, RESNET



- Ý tưởng: dùng các **residual block** để giải quyết vấn đề khi mất **Gradient** quá nhanh.
- **Cấu trúc đặc trưng:** Có các skip connections kiểu `F(x) + x`.
- Sơ đồ khối: x → [Conv → BN → ReLU → Conv → BN] → + → ReLU → y
  |_____↑

- Có các phiên bản nổi tiếng : RESNET -18, RESNET -34, RESNET -50, RESNET -101

Ưu điểm:

- Dễ huấn luyện, ổn định, phù hợp cho việc fine-tune.(Resnet-18/-34)
- Dễ hiểu, phổ biến trong nghiên cứu và giảng dạy.

Nhược điểm:

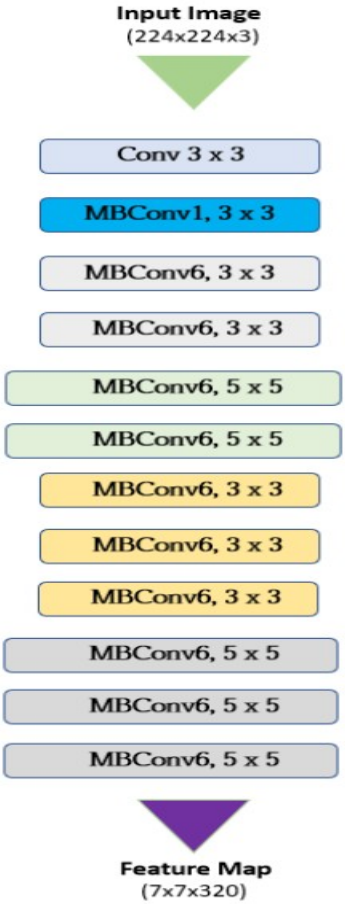
- Mô hình lớn, tiêu tốn tài nguyên hơn.(Resnet -101)
- Không được tối ưu tốt cho tốc độ/hiệu suất trên thiết bị yếu.

Gọi thư viện RESNET từ torchvision:

```
import torchvision.models as models
import torch.nn as nn


# Tải mô hình ResNet-18 pretrained
model = models.resnet18(pretrained=True)
# Thay lớp fully connected cuối cùng (1000 → số lớp bạn muốn)
num_classes = 15  # Ví dụ: 15 loài động vật
model.fc = nn.Linear(model.fc.in_features, num_classes)
```

b, EFFICIENT NET



Khác với cách truyền thống (chỉ tăng chiều sâu hoặc chiều rộng), EfficientNet dùng cách **phóng to mô hình cân bằng** theo cả:

1. **Chiều sâu (depth)** – số layer
2. **Chiều rộng (width)** – số filter/channel mỗi layer
3. **Kích thước ảnh đầu vào (resolution)** -độ phân giải ảnh đầu vào

Tất cả được scale một cách đồng bộ bằng hệ số **φ** theo công thức:
depth: $d = \alpha^{\varphi}$
width: $w = \beta^{\varphi}$
resolution: $r = \gamma^{\varphi}$
→ with constraint: $\alpha * \beta^2 * \gamma^2 \approx 2$
- Các phiên bản :

Tên	Depth	Width	Resolution	Params	Accuracy
B0	1.0×	1.0×	224×224	5.3M	77.1%
B1	1.1×	1.0×	240×240	7.8M	79.1%
B2	1.2×	1.1×	260×260	9.2M	80.1%

Tên	Depth	Width	Resolution	Params	Accuracy
B3	1.4×	1.2×	300×300	12M	81.6%
B4	1.8×	1.4×	380×380	19M	82.9%

Ưu điểm:

- **Hiệu suất cao vượt trội** so với ResNet cùng kích cỡ.
- **Nhẹ**, chạy tốt trên cả CPU lẫn mobile.
- Dễ scale theo yêu cầu tài nguyên.

Nhược điểm:

- Cấu trúc phức tạp hơn ResNet.
- Khó sử dụng với những người mới.

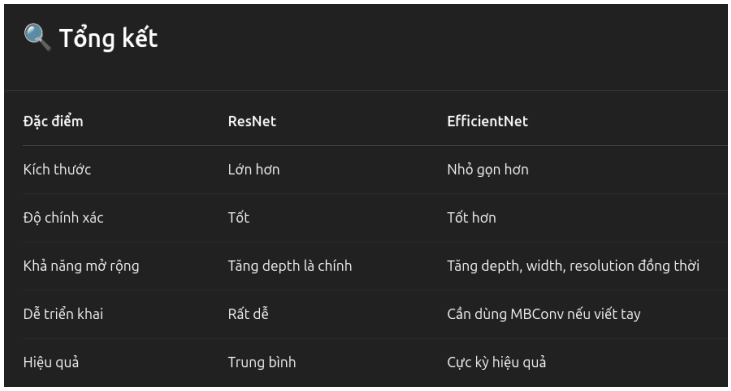
Gọi EfficientNet từ `torchvision`

```
from torchvision.models import efficientnet_b0
import torch.nn as nn

# Tải EfficientNet-B0 pretrained
model = efficientnet_b0(pretrained=True)

# Thay lớp cuối cùng
num_classes = 15  # Số loài cần phân loại
model.classifier[1] = nn.Linear(model.classifier[1].in_features, num_classes)
```

- So sánh giữa 2 loại Model CNN:



- Dựa theo điều kiện Tranning của máy, độ chính xác và hiệu quả, em sẽ lựa chọn Model EfficientNet làm cấu trúc lớp CNN.

```python
import torch
import torch.nn as nn
from torchvision.models import EfficientNet_B0_Weights, efficientnet_b0

def get_efficient(number_classes):  1 usage
    #Call model Efficient B0
    model = efficientnet_b0(weights = EfficientNet_B0_Weights.DEFAULT)

    #Extract in_features from model (1280)
    in_features = model.classifier[1].in_features
    #Replace classifier layer
    model.classifier[1] = nn.Linear(in_features, number_classes)
    return model

if __name__ == "__main__":
    model = get_efficient(19)
    batch_size = 64
    x = torch.rand(batch_size, 3, 224, 224)
    output = model(x)
    print(output)
```

2. Khởi tạo

```python
def img_classifier(args):  1 usage
    warnings.filterwarnings( action: "ignore", category=UserWarning)
    transform = Compose([
        ToTensor(),
        Resize((args.size, args.size)),
        Normalize( mean: [0.485, 0.456, 0.406],
                   std: [0.229, 0.224, 0.225])
    ])

    train_data = Animals(root_path = args.root_path, is_train = True, transform = transform)
    train_loader = DataLoader(train_data, batch_size= args.batch_size, shuffle=True, num_workers = args.num_workers, drop_last=True)

    test_data = Animals(root_path = args.root_path, is_train = False, transform = transform)
    test_loader = DataLoader(test_data, batch_size= args.batch_size, shuffle=False, num_workers= args.num_workers, drop_last=False)

    model = get_efficient(19)
    #Loss function
    criterion = nn.CrossEntropyLoss()
    #optimize function
    optimize = SGD(model.parameters(), lr= args.learning_rate, momentum= args.momentum)
    num_iter = len(train_data)
    train_loss = []
```

- Khởi tạo 2 bộ train_loader và test_loader để có thể xử lí đồng thời nhiều bức ảnh từ bộ dataset.
- Biến đổi ảnh cho đúng định dạng mà Model cần:

+ Vì EfficientNet-B0 cần ảnh 224-224 pixel nên phải **Resize** ảnh về (224,224)
+ **Chuẩn hóa ảnh** theo **mean và std** đã dùng khi huấn luyện EfficientNet trên ImageNet, làm ảnh đầu vào có phân phối giống với ảnh gốc mà EfficientNet từng học.

- Gọi Model **EfficientNet**, khởi tạo **Loss function** và **Optimize function**
Chú ý: Sử dụng argparse() để viết chương trình có thể nhận tham số từ dòng lệnh.

3. Huấn luyện mô hình

```
for epoch in range(start_epoch, args.epoch):
    # train_step
    model.train()
    train_loss = []
    progress_bar = tqdm(train_loader, colour = "cyan")

    for iter, (image, label) in enumerate(progress_bar):
        output = model(image)
        loss = criterion(output, label)
        optimize.zero_grad()
        loss.backward()
        optimize.step()
        train_loss.append(loss.item())
        avg_loss = np.mean(train_loss)
        writer.add_scalar( tag: 'train_loss', avg_loss, epoch * num_iter + iter)

        progress_bar.set_description("train_epoch {}/{}".format( *args: epoch+1, args.epoch))
        progress_bar.set_postfix(loss=f"{avg_loss:.4f}")
```

Quá trình huấn luyện
1. Lấy 1 batch ảnh và nhãn
2. Cho ảnh đi qua model → ra output
3. Tính loss giữa output và nhãn thật
4. Xóa gradient cũ
5. Tính gradient mới (backward)
6. Cập nhật trọng số model
7. Ghi nhận loss để theo dõi quá trình học

```
----------------------------------------------------------------
Total params: 4,031,887
Trainable params: 4,031,887
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 173.64
Params size (MB): 15.38
Estimated Total Size (MB): 189.60
----------------------------------------------------------------
train_epoch 1/100:   1%|          | 9/601 [03:58<4:21:51, 26.54s/it, loss=2.9337]
```

4. Kiểm thử mô hình

```python
#valid
all_label = []
all_predict = []
all_loss = []
model.eval()
with torch.no_grad():
    progress_bar = tqdm(test_loader, colour = "yellow")
    for (image, label) in progress_bar:
        predict = model(image)
        loss = criterion(predict, label)
        max_predict = torch.argmax(predict, dim=1)

        all_loss.append(loss.item())
        all_predict.extend(max_predict.tolist())
        all_label.extend(label.tolist())

loss = np.mean(all_loss)
acc = accuracy_score(all_label, all_predict)
progress_bar.set_description("test_epoch: {}/{}".format( *args: epoch + 1, args.epoch))
progress_bar.set_postfix(loss=f"{loss:.4f}")
```

- model.eval() : Đặt mô hình sang chế độ `eval()`, giúp:
    - Tắt các layer như `Dropout`, `BatchNorm` hoạt động đúng kiểu đánh giá.

- torch.no_grad(): Tắt tính toán gradient để:

    - **Tiết kiệm RAM**, tránh lưu lại các bước backward.
    - **Chạy nhanh hơn**, vì không cần tính toán cho việc huấn luyện.

Quá trình kiểm thử:
1. Dự đoán đầu ra từ mô hình
2. Tính loss giữa dự đoán và nhãn thực tế
3. Lấy nhãn dự đoán có xác suất cao nhất
4. Lưu kết quả để tính toán sau
5. Tính loss trung bình và accuracy cuối cùng
6. Cập nhật tiến trình hiển thị

5. Triển khai lên git