**BỘ GIÁO DỤC VÀ ĐÀO TẠO**
**TRƯỜNG ĐẠI HỌC VĂN LANG**
**KHOA CÔNG NGHỆ THÔNG TIN**

# BÁO CÁO
# BỘ MÔN
# HỌC MÁY VÀ ỨNG DỤNG.

## ĐỀ TÀI
## THUẬT TOÁN CNN

**Lớp: 241_71ITAI41203_01**
**GV: Thầy Huỳnh Thái Học.**

Học kì 1/2024 - 2025 ( từ tháng 9 – 12

## LỜI CẢM ƠN

Chúng em xin gửi lời cảm ơn chân thành đến thầy Huỳnh Thái Học- người đảm nhiệm dạy thực hành môn **Học máy và ứng dụng** ở học kì này của khoa Công nghệ thông tin trường Đại học Văn Lang, đã tận tình hướng dẫn và chỉ dẫn nhóm chúng em trong quá trình thực hiện bài báo cáo tiểu luận cũng như xây dựng đồ án. Dù trong thời gian có hạn, nhưng nhờ sự giúp đỡ của thầy, chúng em đã hoàn thành bài báo cáo và đạt được kết quả mong muốn trong quá trình nghiên cứu của mình.

Chúng em chân thành xin lỗi nếu có một số sai sót và khuyết điểm có trong đồ án của chúng em. Chúng em rất trân trọng và biết ơn sự kiên nhẫn, hỗ trợ và động viên của các thầy. Những lời khuyên và sự hướng dẫn của các thầy giáo đã giúp chúng em tiến bộ và phát triển hơn về kỹ năng nghiên cứu cũng như giải quyết vấn đề. Chúng em cảm thấy may mắn và tự hào khi được làm việc và học tập dưới sự chỉ dẫn của các thầy giáo tận tình và giàu kinh nghiệm.

Một lần nữa, chúng em xin chân thành cảm ơn và mong rằng các thầy luôn khỏe mạnh, hạnh phúc và thành công trong sự nghiệp giảng dạy.

*Ngày 16/11/2024*

# MỤC LỤC

# I. GIỚI THIỆU:

## 1.Lý do chọn đề tài

Chọn đề tài thuật toán **CNN (Convolutional Neural Network**) thường được xem là một quyết định hợp lý và có cơ sở mạnh mẽ. Dưới đây là một số lý do chính:

**Khả năng xử lý hình ảnh vượt trội**

CNN đặc biệt mạnh mẽ trong việc xử lý và nhận diện các mẫu trong dữ liệu hình ảnh. Các lớp convolution giúp tự động học các đặc trưng quan trọng từ hình ảnh mà không cần can thiệp từ con người.

**Ứng dụng rộng rãi**

CNN được áp dụng rộng rãi trong nhiều lĩnh vực như nhận diện khuôn mặt, phân loại đối tượng, nhận diện chữ viết tay, và thậm chí trong các ứng dụng y tế như chẩn đoán từ hình ảnh y khoa (X-quang, MRI).

**Hiệu suất cao**

CNN đã chứng minh hiệu suất cao trong nhiều bài thi và cuộc thi về thị giác máy tính. Các mô hình CNN hàng đầu thường đạt kết quả tốt nhất trong các benchmark hình ảnh như ImageNet.

**Khả năng mở rộng và tùy chỉnh**

CNN có nhiều biến thể và có thể được tùy chỉnh dễ dàng để phù hợp với nhiều bài toán khác nhau. Một số biến thể phổ biến như ResNet, VGG, và Inception đã được thiết kế để cải thiện hiệu suất trên nhiều loại dữ liệu khác nhau.

**Hỗ trợ tốt từ các thư viện phần mềm**

Các thư viện học sâu như TensorFlow, PyTorch cung cấp nhiều công cụ và tiện ích hỗ trợ triển khai và huấn luyện các mô hình CNN một cách dễ dàng. Điều này giúp tiết kiệm thời gian và công sức cho các nhà nghiên cứu và kỹ sư.

**Tiềm năng nghiên cứu**

CNN là một trong những chủ đề nóng trong nghiên cứu trí tuệ nhân tạo và học sâu. Nhiều vấn đề như cải tiến hiệu suất, giảm độ phức tạp, và áp dụng CNN vào các lĩnh vực mới liên tục được khám phá.

**Dễ học và áp dụng**

Với sự phổ biến của các khóa học và tài liệu trực tuyến, việc học và áp dụng CNN đã trở nên dễ dàng hơn bao giờ hết. Điều này giúp người mới bắt đầu nhanh chóng làm quen và thử nghiệm với các mô hình CNN.

**Khả năng tổng quát hóa**

CNN không chỉ áp dụng được trong nhận diện hình ảnh mà còn có thể mở rộng sang các lĩnh vực khác như xử lý ngôn ngữ tự nhiên, dự đoán chuỗi thời gian, và thậm chí là chơi game.

### 2.Kết chương

Những lý do trên khiến việc chọn đề tài về CNN trở nên hấp dẫn và đầy hứa hẹn, không chỉ trong việc nghiên cứu mà còn trong việc ứng dụng thực tiễn để giải quyết các vấn đề trong nhiều lĩnh vực khác nhau.

## II. CƠ SỞ LÝ THUYẾT

### 1. Khái niệm về học máy ứng dụng

**Học máy ứng dụng** là một nhánh của học máy (Machine Learning - ML) liên quan đến việc sử dụng các thuật toán học máy để giải quyết các bài toán thực tiễn trong nhiều lĩnh vực khác nhau. Dưới đây là một cái nhìn chi tiết về học máy ứng dụng:

**Định nghĩa:**

**Học máy ứng dụng** (Applied Machine Learning) là quá trình sử dụng các kỹ thuật và mô hình học máy để phân tích, xử lý và dự đoán từ dữ liệu thực tế. Mục tiêu là tối ưu hóa hiệu suất của các hệ thống và đưa ra quyết định dựa trên dữ liệu.

## 2. Quy trình và ứng dụng

### Quy trình học máy ứng dụng

Quy trình học máy ứng dụng thường bao gồm các bước sau:

**Thu thập dữ liệu**: Thu thập dữ liệu từ nhiều nguồn khác nhau, có thể là cảm biến, cơ sở dữ liệu, web scraping, hoặc thông tin từ các hệ thống hiện có.

**Tiền xử lý dữ liệu**: Làm sạch và chuẩn bị dữ liệu cho việc phân tích. Bao gồm việc loại bỏ các giá trị bị thiếu, chuẩn hóa các đặc trưng, và trích xuất các đặc trưng quan trọng.

**Khám phá dữ liệu**: Sử dụng các kỹ thuật phân tích và trực quan hóa dữ liệu để hiểu rõ hơn về đặc điểm và cấu trúc của dữ liệu.

**Chọn và xây dựng mô hình**: Lựa chọn các thuật toán học máy phù hợp và xây dựng mô hình dựa trên dữ liệu đã chuẩn bị.

**Đánh giá mô hình**: Sử dụng các kỹ thuật đánh giá như phân chia dữ liệu thành tập huấn luyện và tập kiểm tra, sử dụng cross-validation để kiểm tra hiệu suất của mô hình.

**Triển khai mô hình**: Triển khai mô hình vào hệ thống sản xuất và tích hợp với các quy trình hiện có.

**Giám sát và bảo trì mô hình**: Theo dõi hiệu suất của mô hình sau khi triển khai và cập nhật mô hình khi cần thiết để đảm bảo nó tiếp tục hoạt động tốt.

### Ứng dụng của học máy

Học máy ứng dụng được sử dụng trong nhiều lĩnh vực khác nhau, bao gồm nhưng không giới hạn:

**Y tế**: Chẩn đoán bệnh, phân tích hình ảnh y khoa, dự đoán kết quả điều trị.
**Tài chính**: Dự đoán rủi ro, phân tích thị trường, phát hiện gian lận.
**Tiếp thị**: Phân tích hành vi khách hàng, đề xuất sản phẩm, tối ưu hóa chiến dịch quảng cáo.
**Công nghệ thông tin**: Phát hiện tấn công mạng, phân loại email, nhận diện giọng nói.
**Giao thông vận tải**: Điều khiển xe tự lái, dự đoán lưu lượng giao thông, tối ưu hóa lịch trình vận tải.

### Các thuật toán phổ biến trong học máy ứng dụng

Một số thuật toán học máy phổ biến được sử dụng trong học máy ứng dụng bao gồm:

**Hồi quy tuyến tính (Linear Regression)**: Dự đoán giá trị liên tục.
**Rừng ngẫu nhiên (Random Forest)**: Phân loại và hồi quy.

**Máy vector hỗ trợ (Support Vector Machine - SVM)**: Phân loại.

**Mạng nơ-ron tích chập (Convolutional Neural Network - CNN)**: Nhận diện hình ảnh.
**Mạng nơ-ron hồi quy (Recurrent Neural Network - RNN)**: Xử lý chuỗi thời gian và ngôn ngữ tự nhiên.

## 3.Kết chương

Học máy ứng dụng là một lĩnh vực mạnh mẽ và linh hoạt, cung cấp các công cụ và kỹ thuật để giải quyết các bài toán thực tiễn thông qua phân tích và dự đoán từ dữ liệu. Nó đóng một vai trò quan trọng trong việc cải thiện hiệu suất và ra quyết định trong nhiều lĩnh vực khác nhau.

## III. CÁC BƯỚC THỰC HIỆN THUẬT TOÁN

### 1. Bước lấy dữ liệu

**CIFAR-10** là một bộ dữ liệu được phát triển bởi **Canadian Institute for Advanced Research (CIFAR)** và được công bố vào năm 2009. Bộ dữ liệu này được tạo ra từ một phần của bộ dữ liệu **80 million Tiny Images**.

**Mục đích của CIFAR-10:**

CIFAR-10 được sử dụng rộng rãi trong nghiên cứu và huấn luyện các thuật toán học máy và trí tuệ nhân tạo, đặc biệt là trong lĩnh vực **nhận diện hình ảnh**. Bộ dữ liệu này gồm 60,000 hình ảnh 32x32 màu sắc thuộc 10 danh mục khác nhau, với mỗi danh mục có 6,000 hình ảnh. Các danh mục bao gồm: máy bay, xe hơi, chim, mèo, hươu, chó, ếch, ngựa, tàu, và xe nặng.

**Dữ liệu đầu vào, đầu ra (input, output):**

**Đầu vào (input)**: Bộ dữ liệu CIFAR-10 bao gồm các hình ảnh 32x32 màu sắc được lưu trữ dưới dạng các giá trị RGB (Red, Green, Blue). Mỗi hình ảnh được chuyển đổi thành một mảng số và được sử dụng như đầu vào cho các mô hình học máy.

**Đầu ra (output)**: Mục tiêu của các mô hình học máy khi sử dụng CIFAR-10 là phân loại các hình ảnh thành một trong 10 danh mục khác nhau. Đầu ra của mô hình là một danh mục gồm các danh mục có thể bao gồm: máy bay, xe hơi, chim, mèo, hươu, chó, ếch, ngựa, tàu, và xe nặng.

**Ứng dụng:**

CIFAR-10 thường được sử dụng để huấn luyện các mạng nơ-ron tích chập (Convolutional Neural Networks - CNNs) và để đánh giá hiệu suất của các thuật toán mới. Bộ dữ liệu này cũng được sử dụng như một tiêu chuẩn để so sánh hiệu suất giữa các nhóm nghiên cứu và các thuật toán khác nhau.

**CODE và Kết quả:**

**+ Import các thư viện cần thiết như**


**+ Download dữ liệu CIFAR10 sau đó chia thành 2 tập dữ liệu training và testing.**

```python
# Define the transformation to normalize the data
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Download and load the CIFAR-10 training dataset
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=100,
                                          shuffle=True, num_workers=2)

# Download and load the CIFAR-10 testing dataset
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=100,
                                         shuffle=False, num_workers=2)

# Classes in CIFAR-10 dataset
classes = ('plane', 'car', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck')

# Let's print some information to verify
print(f'Training dataset size: {len(trainset)}')
print(f'Testing dataset size: {len(testset)}')
```

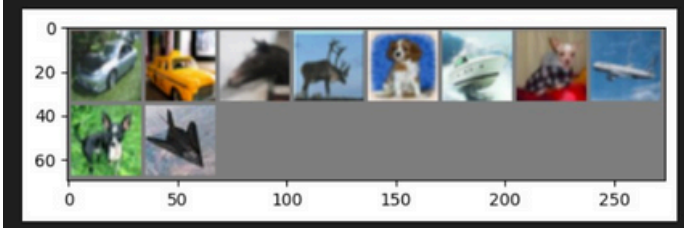
✓ 1.7s Python

```
Files already downloaded and verified
Files already downloaded and verified
Training dataset size: 50000
Testing dataset size: 10000
```

## + Hiển thị 10 ảnh đầu tiên trong tập dữ liệu testing

```python
def imshow(img):
    img = img * 0.5 + 0.5
    np_img = img.numpy()
    plt.imshow(np.transpose(np_img, (1, 2, 0)))
    plt.show()

for i, (images, labels) in enumerate(trainloader, 0):
    imshow(torchvision.utils.make_grid(images[:10]))
    break
```

✓ 8.1s Python




## + Khởi tạo hàm loss function và phương thức optimizer

```python
n_features = 32 * 32 * 3
model = getModel(n_features)
lr = 0.01
optim = SGD(params = model.parameters(), lr = lr)
loss_fn = nn.CrossEntropyLoss()
model
```

✓ 0.0s Python

```
Sequential(
  (0): Flatten(start_dim=1, end_dim=-1)
  (1): Linear(in_features=3072, out_features=256, bias=True)
  (2): ReLU()
  (3): Linear(in_features=256, out_features=10, bias=True)
)
```

## + Xây dựng hàm đánh giá model

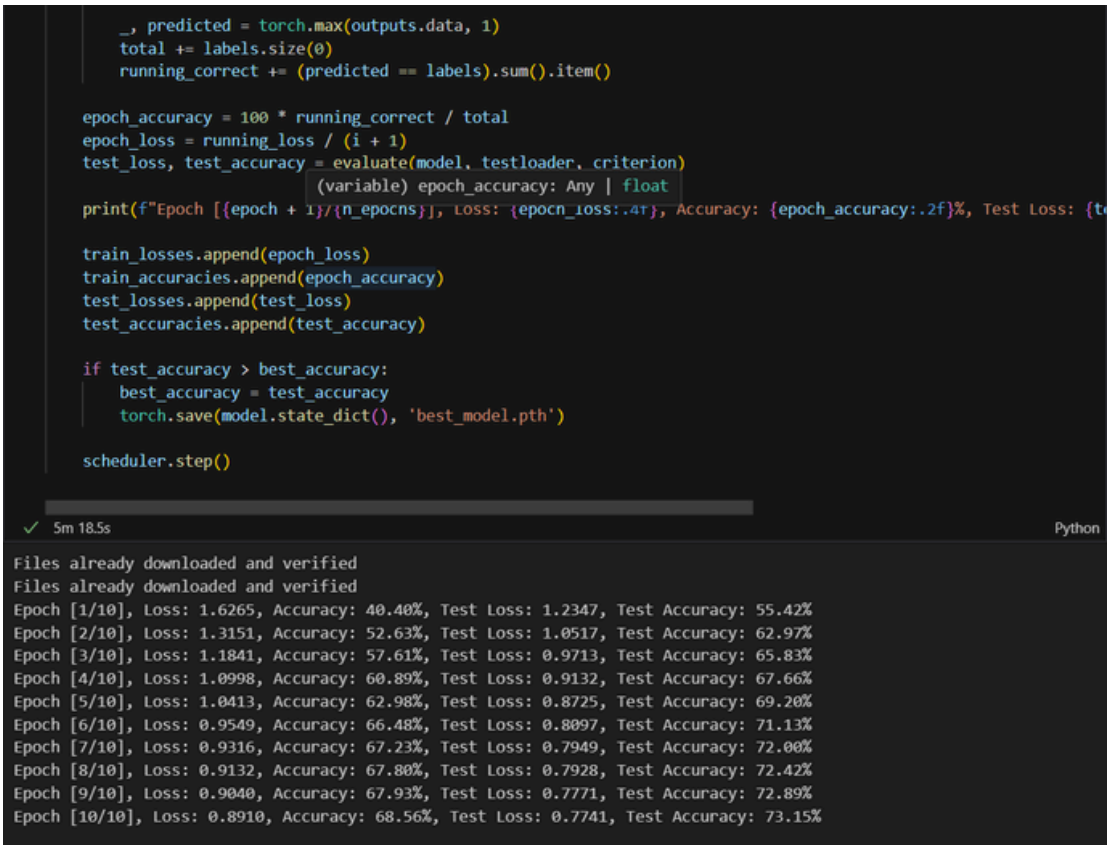
```python
def evaluate(model, testloader, criterion):
    model.eval()
    test_loss = 0.0
    correct = 0
    total = 0
    for images, labels in testloader:
        images, labels = images.to(device), labels.to(device)

        outputs = model(images)
        loss = criterion(outputs, labels)
        test_loss += loss.item()

        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    test_loss = test_loss / len(testloader)
    return test_loss, accuracy
```

✓ 0.0s Python

## + Bắt đầu training và đánh giá model.

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.optim.lr_scheduler import StepLR
from sklearn.metrics import precision_score, recall_score, accuracy_score

transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=128,
                                         shuffle=False, num_workers=2)

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 8 * 8, 512)
        self.fc2 = nn.Linear(512, 10)
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 8 * 8)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

```python
model = Net()
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
scheduler = StepLR(optimizer, step_size=5, gamma=0.2)

n_epochs = 10
train_losses = []
train_accuracies = []
test_losses = []
test_accuracies = []

def evaluate(model, dataloader, loss_fn):
    model.eval()
    test_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in dataloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = loss_fn(outputs, labels)
            test_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = 100 * correct / total
    test_loss /= len(dataloader)
    return test_loss, accuracy

best_accuracy = 0.0

for epoch in range(n_epochs):
    model.train()
    running_loss = 0.0
    running_correct = 0
    total = 0
    for i, (inputs, labels) in enumerate(trainloader, 0):
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()

        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
```



```
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        running_correct += (predicted == labels).sum().item()

    epoch_accuracy = 100 * running_correct / total
    epoch_loss = running_loss / (i + 1)
    test_loss, test_accuracy = evaluate(model, testloader, criterion)
                               (variable) epoch_accuracy: Any | float
    print(f"Epoch [{epoch + 1}/{n_epochs}], Loss: {epoch_loss:.4f}, Accuracy: {epoch_accuracy:.2f}%, Test Loss: {t

    train_losses.append(epoch_loss)
    train_accuracies.append(epoch_accuracy)
    test_losses.append(test_loss)
    test_accuracies.append(test_accuracy)

    if test_accuracy > best_accuracy:
        best_accuracy = test_accuracy
        torch.save(model.state_dict(), 'best_model.pth')

    scheduler.step()
```

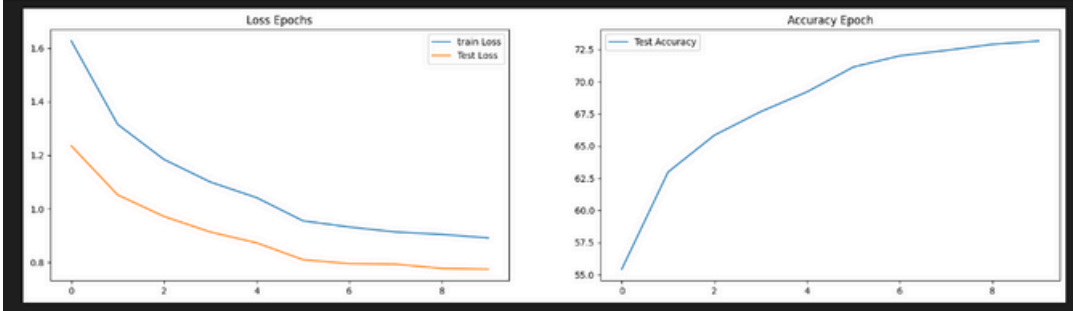
✓ 5m 18.5s Python

```
Files already downloaded and verified
Files already downloaded and verified
Epoch [1/10], Loss: 1.6265, Accuracy: 40.40%, Test Loss: 1.2347, Test Accuracy: 55.42%
Epoch [2/10], Loss: 1.3151, Accuracy: 52.63%, Test Loss: 1.0517, Test Accuracy: 62.97%
Epoch [3/10], Loss: 1.1841, Accuracy: 57.61%, Test Loss: 0.9713, Test Accuracy: 65.83%
Epoch [4/10], Loss: 1.0998, Accuracy: 60.89%, Test Loss: 0.9132, Test Accuracy: 67.66%
Epoch [5/10], Loss: 1.0413, Accuracy: 62.98%, Test Loss: 0.8725, Test Accuracy: 69.20%
Epoch [6/10], Loss: 0.9549, Accuracy: 66.48%, Test Loss: 0.8097, Test Accuracy: 71.13%
Epoch [7/10], Loss: 0.9316, Accuracy: 67.23%, Test Loss: 0.7949, Test Accuracy: 72.00%
Epoch [8/10], Loss: 0.9132, Accuracy: 67.80%, Test Loss: 0.7928, Test Accuracy: 72.42%
Epoch [9/10], Loss: 0.9040, Accuracy: 67.93%, Test Loss: 0.7771, Test Accuracy: 72.89%
Epoch [10/10], Loss: 0.8910, Accuracy: 68.56%, Test Loss: 0.7741, Test Accuracy: 73.15%
```

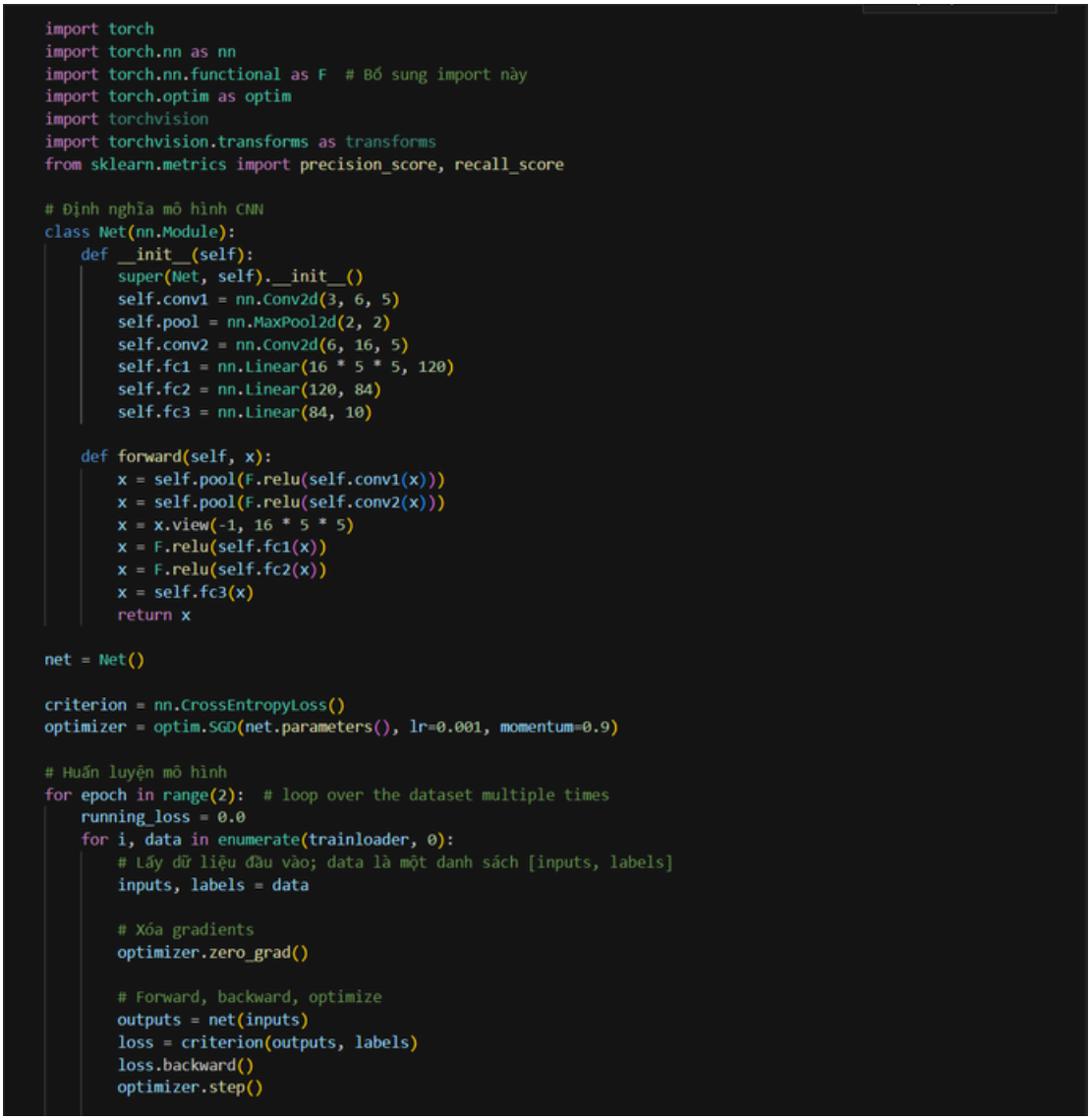
**+ Biểu đồ sau khi train và test**

```python
import matplotlib.pyplot as plt

# Assuming 'train_losses', 'test_losses', and 'test_accuracies' are already defined

plt.figure(figsize=(20, 5))

plt.subplot(121)
plt.title('Loss Epochs')
plt.plot(train_losses, label='train Loss')
plt.plot(test_losses, label='Test Loss')
plt.legend()

plt.subplot(122)
plt.title('Accuracy Epoch')
plt.plot(test_accuracies, label='Test Accuracy')
plt.legend()

plt.show()
```

✓ 0.2s Python



## 2. Sử dụng thuật toán CNN cho ra precision và recall như sau



```python
import torch
import torch.nn as nn
import torch.nn.functional as F  # Bổ sung import này
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from sklearn.metrics import precision_score, recall_score

# Định nghĩa mô hình CNN
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

# Huấn luyện mô hình
for epoch in range(2):  # loop over the dataset multiple times
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # Lấy dữ liệu đầu vào; data là một danh sách [inputs, labels]
        inputs, labels = data

        # Xóa gradients
        optimizer.zero_grad()

        # Forward, backward, optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

```python
            # In ra thông tin
            running_loss += loss.item()
            if i % 200 == 199:    # In ra mỗi 200 mini-batches
                print(f'[Epoch: {epoch + 1}, Mini-batch: {i + 1}] loss: {running_loss / 200:.3f}')
                running_loss = 0.0

    print('Finished Training')

    # Dự đoán và tính toán Precision và Recall
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for data in testloader:
            images, labels = data
            outputs = net(images)
            _, predicted = torch.max(outputs, 1)
            all_preds.extend(predicted.numpy())
            all_labels.extend(labels.numpy())

    precision = precision_score(all_labels, all_preds, average='macro') *100
    recall = recall_score(all_labels, all_preds, average='macro') *100

    print(f'Precision: {precision:.2f}%')
    print(f'Recall: {recall:.2f}%')
```

✓ 31.8s Python

```
[Epoch: 1, Mini-batch: 200] loss: 2.304
[Epoch: 1, Mini-batch: 400] loss: 2.302
[Epoch: 2, Mini-batch: 200] loss: 2.296
[Epoch: 2, Mini-batch: 400] loss: 2.284
Finished Training
Precision: 17.93%
Recall: 19.22%
```

## 3. Bảng so sánh giữa các thuật toán

Bảng so sánh giữa các thuật toán

| Thuật toán | Precision | Recall |
|---|---|---|
| SVM | 28.99% | 22.49% |
| Random Forest | 16.10% | 20.59% |
| CNN | 17.93% | 19.22% |

+ Nhận xét:

**SVM (Support Vector Machine)**:

- **Precision**: Đạt 28.99%, cho thấy SVM có khả năng khá tốt trong việc phân loại chính xác các đối tượng mà nó dự đoán là dương tính. Điều này có nghĩa là trong số các dự đoán dương tính của SVM, phần lớn là đúng.
- **Recall**: Đạt 22.49%, cho thấy SVM có khả năng phát hiện các đối tượng dương tính trong tổng số các đối tượng thực sự dương tính là ở mức trung bình. SVM bỏ sót một số lượng đáng kể các đối tượng dương tính thực sự.

**Random Forest**:

- **Precision**: Đạt 16.10%, thấp hơn so với SVM. Điều này có nghĩa là Random Forest có tỷ lệ cao hơn trong việc dự đoán sai các đối tượng dương tính (FP nhiều hơn).
- **Recall**: Đạt 20.59%, hơi thấp hơn so với SVM nhưng cao hơn CNN. Điều này cho thấy Random Forest cũng bỏ sót một số lượng đối tượng dương tính, nhưng vẫn phát hiện được một phần đáng kể các đối tượng này.

**CNN (Convolutional Neural Network)**:

- **Precision**: Đạt 17.93%, cũng thấp hơn so với SVM nhưng cao hơn so với Random Forest. Điều này có nghĩa là CNN có tỷ lệ chính xác khá ổn trong các dự đoán dương tính của nó.
- **Recall**: Đạt 19.22%, thấp hơn so với cả SVM và Random Forest. Điều này cho thấy CNN bỏ sót nhiều đối tượng dương tính hơn so với hai thuật toán kia.

# IV. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## 1.Kết luận

**SVM (Support Vector Machine)**: tỏ ra vượt trội hơn cả về Precision lẫn Recall so với Random Forest và CNN trong trường hợp này. Điều này làm cho SVM trở thành lựa chọn tốt hơn khi cần một thuật toán với độ chính xác và khả năng phát hiện tốt.

**Random Forest**: mặc dù có độ chính xác thấp hơn nhưng vẫn giữ vững khả năng phát hiện đối tượng dương tính khá tốt, làm cho nó trở thành lựa chọn cân bằng giữa Precision và Recall.

**CNN (Convolutional Neural Network)**: có thể cần cải tiến thêm hoặc điều chỉnh để nâng cao hiệu suất, đặc biệt là ở chỉ số Recall.

## 2.Hướng phát triển

Học máy (machine learning) đang phát triển mạnh mẽ và có nhiều ứng dụng tiềm năng trong nhiều lĩnh vực khác nhau. Dưới đây là một số hướng phát triển chính:

**Học sâu (Deep Learning)**: Học sâu tiếp tục đóng vai trò quan trọng, với các mạng neural phức tạp hơn và khả năng xử lý dữ liệu lớn hơn.

**Học tự động (AutoML)**: Các công cụ học tự động giúp tối ưu hóa quá trình phát triển mô hình học máy, giúp các nhà phát triển không cần kiến thức sâu rộng về học máy.

**Học tập trực tuyến (Online Learning)**: Học tập trực tuyến cho phép mô hình học máy được cập nhật và điều chỉnh liên tục dựa trên dữ liệu mới.

**Học thống kê (Statistical Learning)**: Kết hợp các phương pháp thống kê và học máy để cải thiện độ chính xác và hiệu quả của các mô hình.

**Học với ít dữ liệu (Few-Shot Learning)**: Các phương pháp này giúp mô hình học máy có thể học từ ít dữ liệu, đặc biệt hữu ích trong các tình huống có dữ liệu hạn chế.

**Học bảo mật (Secure Learning)**: Đảm bảo tính bảo mật và bảo vệ dữ liệu khi sử dụng học máy, đặc biệt là trong các ứng dụng nhạy cảm như y tế và tài chính.

Các ứng dụng của học máy bao gồm:

**Y tế**: Chẩn đoán bệnh, phân tích dữ liệu y tế, phát triển thuốc mới.
**Kinh tế và tài chính**: Dự đoán xu hướng thị trường, phân tích rủi ro tài chính, tự động hóa giao dịch.
**Giao thông**: Phát triển xe tự lái, hệ thống giao thông thông minh, quản lý lưu lượng giao thông.
**Nội dung và truyền thông**: Tích hợp học máy vào các nền tảng truyền thông, tạo nội dung tương tác, phân tích hành vi người dùng.

**Nông nghiệp:** Phân tích dữ liệu từ các thiết bị khu vực, dự đoán mùa vụ, quản lý nông trại thông minh.

# Tài liệu tham khảo

[1] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. https://www.deeplearningbook.org/

[2] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention is All You Need*. *NeurIPS 2017*. https://arxiv.org/abs/1706.03762

[3] Chen, L., Chen, P., & Lin, Z.(2020). Artificial intelligence in education: A review. Ieee Access, 8, 75264-75278. https://ieeexplore.ieee.org/document/9069875

[4] Devlin, J., Chang, M.W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. *NAACL-HLT*. https://aclanthology.org/N19-1423.pdf

[5] Liu, Y., et al. (2019). *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. *arXiv preprint arXiv:1907.11692*. https://arxiv.org/pdf/1907.11692

[6] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language Models are Unsupervised Multitask Learners*. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[7] Bishop, C.M.(2006). *Pattern Recognition and Machine Learning*. Springer. https://link.springer.com/book/9780387310732

[8] Jurafsky, D., & Martin, J. H. (2019). *Speech and Language Processing* (3rd ed.). Pearson. https://idiom.ucsd.edu/~bakovic/compphon/Jurafsky,%20Martin.-Speech%20and%20Language%20Processing_%20An%20Introduction%20to%20Natural%20Language%20Processing%20(2007).pdf

[9] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., & Askell, A., et al. (2020). *Language Models are Few-Shot Learners*. *arXiv preprint arXiv:2005.14165*. https://arxiv.org/abs/2005.14165

[10] Zhang, T., & Mikolov, T. (2020). *Understanding Large Language Models*. *AI Journal*, 57(3), 123-135.https://arxiv.org/html/2402.06196v2