

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM
KHOA CƠ KHÍ CHẾ TẠO MÁY



HCMUTE

BÁO CÁO GIỮA KÌ
TRÍ TUỆ NHÂN TẠO
NHẬN DIỆN HÌNH ẢNH BẰNG PHƯƠNG PHÁP DEEP LEARNING
(Convolutional Neural Network - CNN)

GIẢNG VIÊN HƯỚNG DẪN:	PGS.TS NGUYỄN TRƯỜNG THỊNH
Mã Học Phần:	222ARIN337629E
HỌ & TÊN SINH VIÊN:	Nguyễn Ngọc Nhân
MSSV:	20146262

TP Hồ Chí Minh, ngày 30 tháng 4 năm 2023

Mục lục

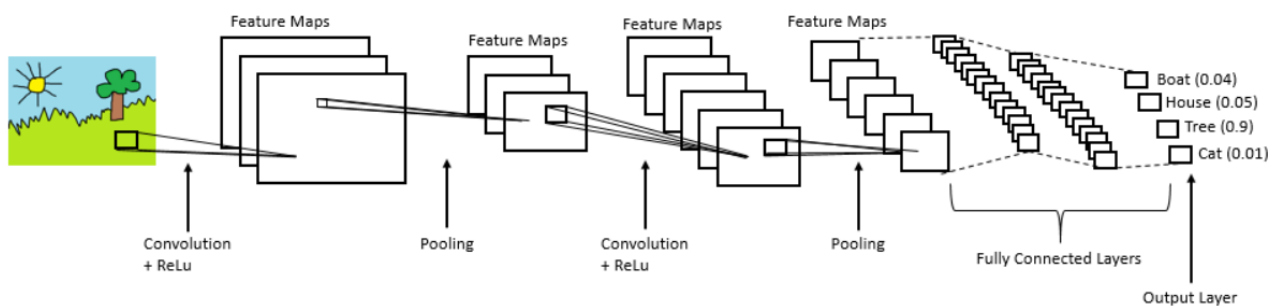
1	Introduction - Giới thiệu	3
2	Methodology - Phương pháp	3
2.1	Convolution Layer - Lớp tính chập	3
2.2	Stride - Bước nhảy	4
2.3	Padding - Đường viền	5
2.4	Rectified Linear Unit (Relu) - Hàm phi tuyến	5
2.5	Pooling Layer - Lớp gộp	6
2.6	Forward Pass and Backward Pass - chạy thuận và chạy nghịch	6
3	Implementation - Thực hiện	7
3.1	Prediction of your future based on hand palm outline or face or fingerprint. . .	8
3.2	5 kinds of flowers (rose, lotus, water lily, apricot, daisy, pink)	17
3.3	10 Vietnamese dishes (bún bò, chè, bánh xèo.....)	25
3.4	VN banknotes (5000vnd, 10000vnd, 20k, 50k, 100k, 500k)	33
3.5	Recognition of all members of class from face images (you collected)	41
4	Results - Kết quả	49
5	Conclusions - Kết luận	49
6	References	50

1 Introduction - Giới thiệu

- Convolutional Neural Network (CNN – Mạng nơ-ron tích chập) là một trong những mô hình Deep Learning (DL) tiên tiến để nhận dạng và phân loại hình ảnh. Nó giúp cho chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao như hiện nay.

- CNN được áp dụng rộng rãi trong nhiều lĩnh vực hiện nay, bao gồm việc nhận diện khuôn mặt, phân loại bệnh qua ảnh chụp từ y khoa, phát hiện các object (vật thể) qua các video và hình ảnh. Với sự phát triển liên tục, CNN đã đóng vai trò quan trọng trong việc giải quyết các vấn đề về lĩnh vực thị giác máy tính.

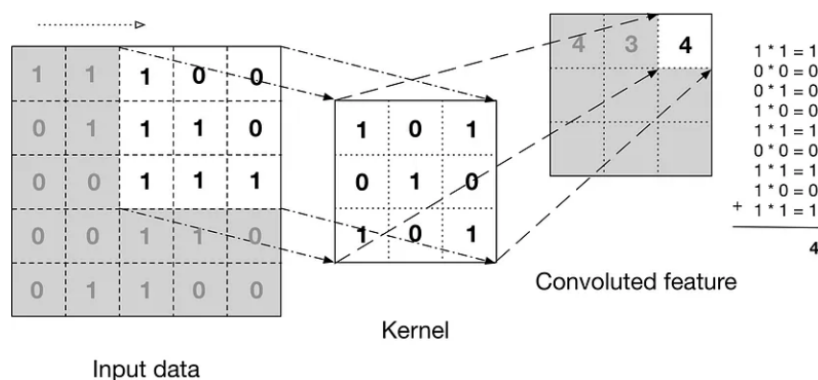
2 Methodology - Phương pháp







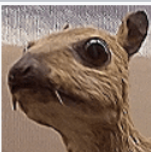
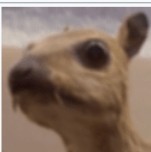
2.1 Convolution Layer - Lớp tích chập

- Là lớp đầu tiên để trích xuất các tính năng từ hình ảnh đầu vào. Nó sử dụng phép tính toán convolution (tích chập) giữa các ma trận. Tức là, nó áp dụng một kernel (còn được gọi là filter hoặc weight) đến một vùng của ảnh đầu vào để tạo ra một feature map (bản đồ đặc trưng) mới.

- Việc tính toán này được thực hiện bằng cách áp dụng phép nhân ma trận giữa kernel và các phần tử tương ứng trên vùng ảnh đầu vào, sau đó tính tổng các kết quả và lưu vào feature map. Quá trình này được thực hiện trên toàn bộ ảnh đầu vào để tạo ra nhiều feature map khác nhau, từ đó trích xuất các đặc trưng của ảnh để phục vụ cho các mục đích như phân loại, nhận dạng, phát hiện.

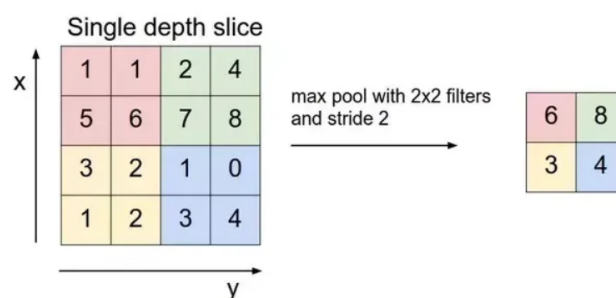


- Sự kết hợp của 1 hình ảnh với các bộ lọc khác nhau có thể giúp chúng ta tìm ra cạnh, làm mờ và làm sắc nét bằng cách áp dụng các bộ lọc.
- Ví dụ dưới đây cho thấy hình ảnh tích chập khác nhau sau khi áp dụng các Kernel khác nhau.

Operation	Kernel ω	Image result $g(x,y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

2.2 Stride - Bước nhảy

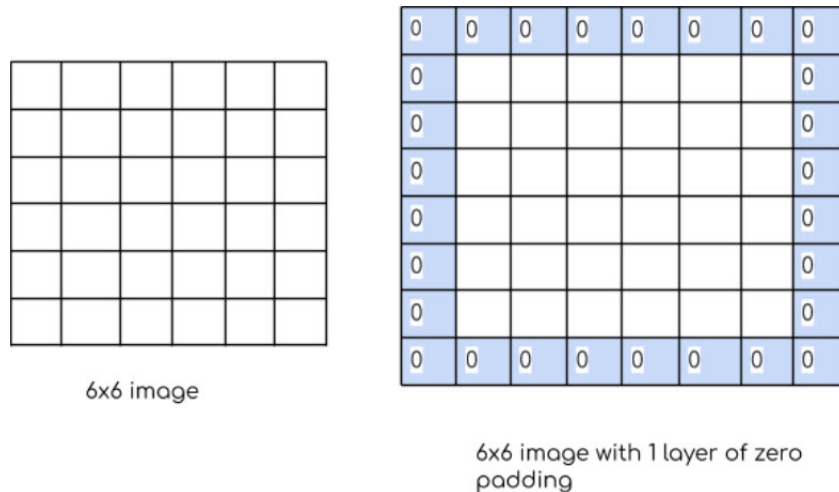
- Stride giúp ta xác định bao nhiêu pixel bước đi mỗi lần trượt bộ lọc tích chập trên ma trận đầu vào.
- Ví dụ khi stride là 1 tức là di chuyển các kernel 1 pixel. Khi stride là 2 tức là di chuyển các kernel đi 2 pixel và tiếp tục như vậy.
- Ví dụ hình ảnh dưới đây mô tả lớp chập hoạt động với stride là 2.



2.3 Padding - Đường viền

- Để giải quyết một số trường hợp kernel không phù hợp với ảnh đầu vào vì sau mỗi lần thực hiện phép tính convolution xong thì kích thước ma trận output - Y đều nhỏ hơn ma trận input - X. Ta chọn tăng kích thước ảnh bằng cách thêm padding. Tức là thêm giá trị 0 ở viền ngoài ma trận X.

- Ví dụ dưới đây là thêm giá trị 0 ở viền với padding = 1.



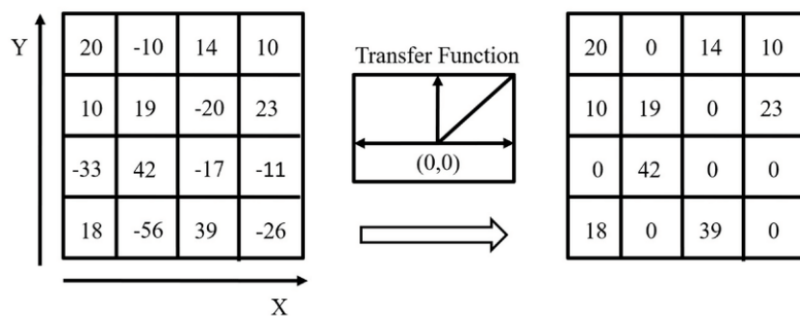
- Phép tính này được gọi là convolution với padding=1. Khi padding = k nghĩa là ta thêm k vector 0 vào mỗi phía của ma trận.

2.4 Rectified Linear Unit (Relu) - Hàm phi tuyến

- Giúp mạng neural học các đặc trưng tốt hơn và tăng tốc quá trình hội tụ trong quá trình huấn luyện.

- Giảm thiểu hiện tượng Gradient Vanishing (tức là khi gradient giảm đáng kể đến mức rất gần bằng 0 làm các trọng số của mạng neuron được cập nhật rất chậm hoặc thậm chí không cập nhật được nữa).

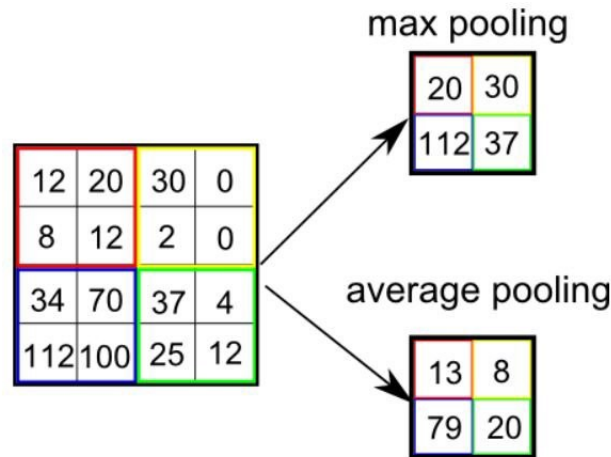
- Đầu ra $f(x) = \max(0, x)$. Tức là ma trận sau khi thực hiện phép tính chập và áp dụng hàm Relu ta sẽ ra được một ma trận không âm



- Tuy nhiên các node có giá trị nhỏ hơn 0, qua ReLU activation sẽ thành 0, hiện tượng này gọi là “Dying ReLU“. Khi các node bị chuyển thành 0 sẽ không còn ý nghĩa với linear activation ở lớp tiếp theo và các hệ số tương ứng từ node đấy cũng không được cập nhật với gradient descent (vì bằng 0 nên không thể gradient được). \Rightarrow Leaky ReLU ra đời.

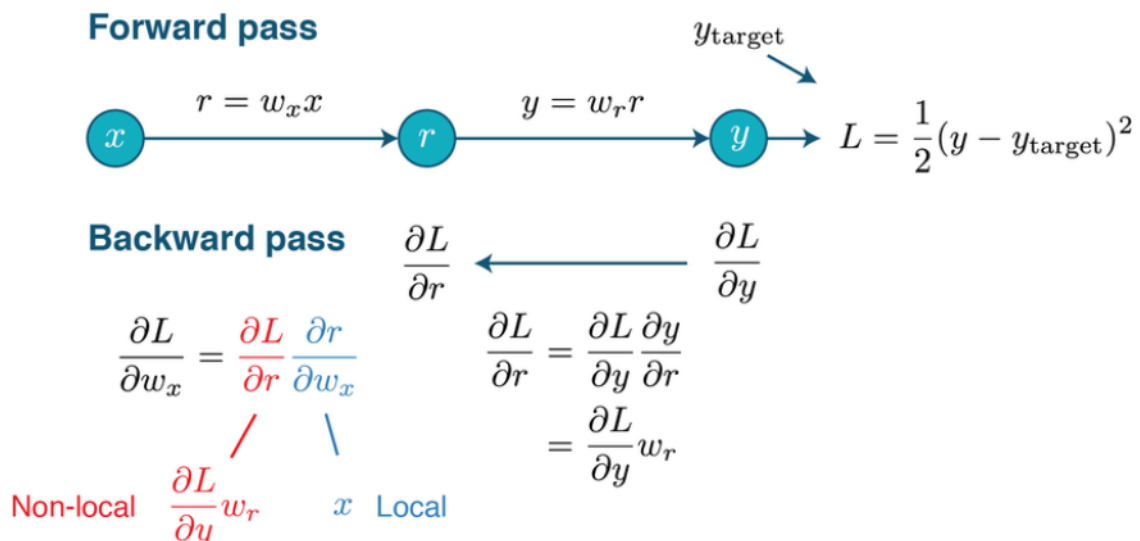
2.5 Pooling Layer - Lớp gộp

- Để giảm bớt kích thước đầu vào và trích xuất các đặc trưng quan trọng.
- Thường được dùng giữa các convolutional layer.
- Có 2 loại pooling layer phổ biến là max pooling và average pooling:
 - + Max pooling là pooling layer sẽ chọn ra giá trị lớn nhất trong mỗi khối đó để đưa ra đầu ra.
 - + Average pooling là pooling layer sẽ lấy giá trị trung bình trong mỗi khối để đưa ra đầu ra.



2.6 Forward Pass and Backward Pass - chạy thuận và chạy nghịch

- Forward Pass là quá trình đưa dữ liệu đầu vào đi qua một mạng neural network, các phép tính được thực hiện từ lớp đầu vào đến lớp đầu ra và cho ra kết quả đầu ra.
- Backward pass là quá trình tính toán đạo hàm (gradient) của hàm mất mát theo các tham số của mạng neural network. Quá trình này được thực hiện bằng thuật toán backpropagation, từ lớp đầu ra trở về lớp đầu vào, giúp cho việc cập nhật các trọng số mạng để tối ưu hoá mô hình được thực hiện một cách hiệu quả.



3 Implementation - Thực hiện

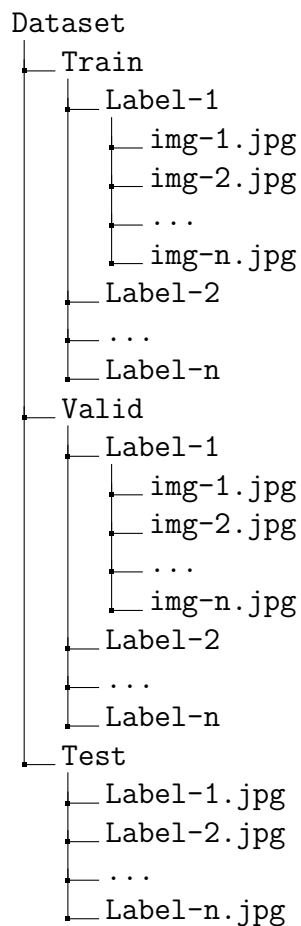
Các bước thực hiện

- + **Step 1:** Thu thập dữ liệu
- + **Step 2:** Xử lý dữ liệu
- + **Step 3:** Xây dựng model CNN
- + **Step 4:** Viết hàm loss đánh giá
- + **Step 5:** Nhập ảnh đầu vào từ tập test và demo kết quả

Công việc đã thực hiện

- ☒ Xây dựng model CNN với frame-work.
 - ☒ Pytorch
 - ☐ Keras
 - ☐ TensorFlow
- ☒ Demo kết quả.
 - ☒ Demo kết quả trên ảnh
 - ☐ Demo kết quả trên video
- ☒ Trực quan hóa và theo dõi các chỉ số quá trình huấn luyện.
 - ☒ TensorBoard

Cấu trúc dataset chung.



3.1 Prediction of your future based on hand palm outline or face or fingerprint.

Code đầy đủ ở đây [6]

- Đầu tiên ta sẽ import các thư viện cần thiết cho việc xây dựng và thực thi model
- Ở đây ta sẽ dùng Cuda cho việc xây model trên GPU và nó giúp ta tăng tổng quá trình train và thực thi model

```
In [1]: import numpy as np
        from tqdm import tqdm
        import torch
        import torch.nn as nn
        import torchvision
        import torch.nn.functional as F
        from torch.utils.data import DataLoader
        from torchvision import datasets, transforms, models
        from torchvision.transforms import ToTensor
        import matplotlib.pyplot as plt
        from torch.utils.tensorboard import SummaryWriter

        device = torch.device('cuda' if torch.cuda.is_available() else 'cpu'
                               )
        print(device)
```

Out [1]: cuda

- Tiếp Theo ta sẽ xây dựng hàm **transform** để biến đổi trên dữ liệu (data augmentation) hoặc chuẩn hóa (data normalization) dữ liệu trước khi đưa vào huấn luyện mô hình.
- Việc sử dụng hàm **transform** giúp cho mô hình huấn luyện được đào tạo trên nhiều dữ liệu đa dạng hơn, giúp mô hình học được các đặc trưng tổng quát hơn và tránh overfitting trên dữ liệu huấn luyện.
- Quy trình của hàm **transform**: Thay đổi kích thước ảnh về dạng (128x128) \Rightarrow đưa ảnh về dạng các cạnh dựa vào hàm **Canny()** \Rightarrow Đưa ma trận ảnh về dạng **tensor** để làm việc với pytorch.

```
In [2]: from PIL import Image
        import torch
        from torchvision import transforms
        import cv2

        class CannyEdgeTransform:
            def __init__(self, threshold1=100, threshold2=220):
                self.threshold1 = threshold1
                self.threshold2 = threshold2

            def __call__(self, img):
                # Chuyển PIL.Image sang numpy array
                img_np = np.array(img)
                # Chuyển sang ảnh grayscale
                img_gray = cv2.cvtColor(img_np, cv2.COLOR_BGR2GRAY)
                # Áp dụng edge detection bằng Canny
                edges = cv2.Canny(img_gray, self.threshold1, self.threshold2
                                  )
```



```

        # Chuyển lại PIL.Image
        edges_pil = Image.fromarray(edges)
        # Áp dụng transform của Pytorch
        edges_tensor = transforms.ToTensor()(edges_pil)
        return edges_tensor

transform = transforms.Compose([
    transforms.Resize(124), # thay đổi kích thước bên ngang nhất
                             # thành 124 pixel
    transforms.CenterCrop(124), # cắt kích thước bên dài nhất ra
                                # khoảng 124 pixel từ trung
                                # tâm
    CannyEdgeTransform(threshold1=100, threshold2=220) # chuyển ảnh
                                                         # về dạng canny edge để thay
                                                         # được nét mặt người
])
x_train = FaceDataset(root=("/kaggle/input/face-class/Face/Train"),
                       transform = transform)
x_val = FaceDataset(root = ('/kaggle/input/face-class/Face/Val'),
                    transform = transform)

```

- Đầu vào là hình ảnh mặt người sau đó ta sẽ xác định và cắt vùng ảnh chứa mặt người về kích thước (124x124) sau đó ta sẽ dùng hàm **CannyEddgeTransform()** để chuyển về dạng các cạnh.

```

In [3]: import matplotlib.pyplot as plt

        # Load ảnh gốc
        img = Image.open('/kaggle/input/face-class/Face/Train/NguyenNgocNhan
                           /img1.jpg')

        # Hiện thị ảnh gốc
        plt.subplot(1, 2, 1) # 1 hàng, 2 cột, vị trí 1
        plt.imshow(img)

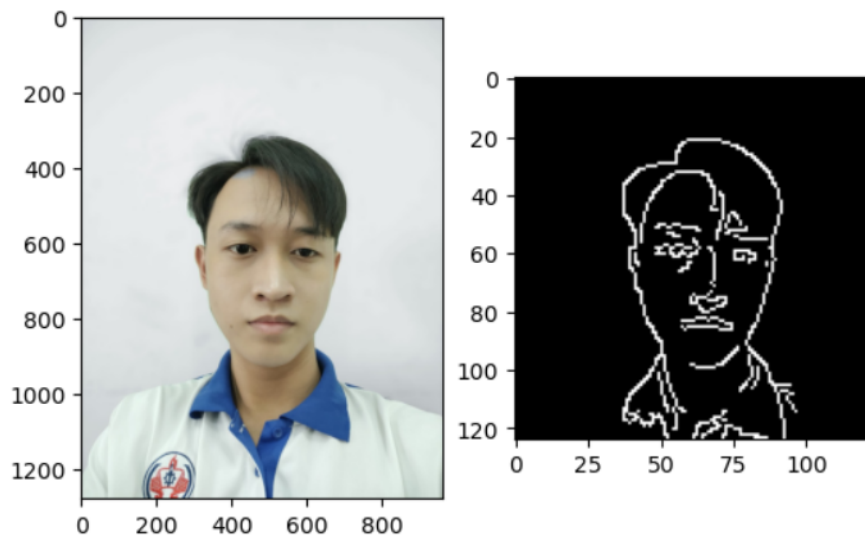
        # Chuyển tensor về numpy array và đổi lại thứ tự các kênh màu
        transformed_img = transform(img).numpy().transpose(1, 2, 0)
        # Tensor: (C, H, W) -> OpenCV, Matlab: (H, W, C)

        # Hiện thị ảnh sau khi áp dụng transform
        plt.subplot(1, 2, 2) # 1 hàng, 2 cột, vị trí 2
        plt.imshow(transformed_img, cmap='gray')

        # Hiện thị dòng thời
        plt.show()

```

Out [3]:



```
In [4]: print("Tong class: ",len(x_train.classes))
        class_predict_future = ['Ca si', 'Game thu', 'Trader', 'Ky su PID',
                                'Ky su']
        print("Cac du doan co the: ",class_predict_future)
```

Out [4]: Tong class: 5

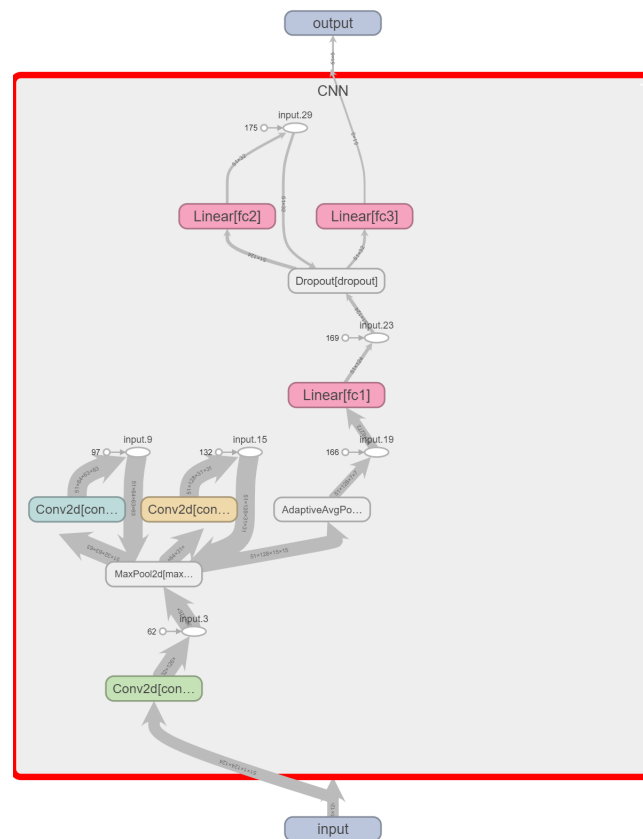
Cac du doan co the: ['Ca si', 'Game thu', 'Trader', 'Ky su PID', 'Ky su']

- Tiếp theo ta sẽ tiến hành thiết lập siêu tham số (hyperparameter) và chia dữ liệu theo batch cho model
- Ở đây ta sử dụng Weight Decay (L2 Regularization) bằng 0.0001 để giảm thiểu Overfitting [5]

```
In [5]: batch = 128 # Khai bao kích thước batch
        epochs = 12 # Thuc hien 80 epochs
        learning_rate = 1e-3 # Khao bao tốc độ học là 10-3
        weight_decay = 1e-4 # Khai bao weight decay là 10-4 để giảm thiểu
                               overfitting

        train_loader = DataLoader(x_train, batch_size = batch, shuffle =
                                   True)
        val_loader = DataLoader(x_val, batch_size = batch, shuffle = True)
```

- Tiếp theo ta sẽ tiến hành xây dựng model CNN
- Dưới đây là hình ảnh cấu trúc model CNN. Chi tiết xem thêm tại đây[11]



```
In [6]: class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.maxpool = nn.MaxPool2d(kernel_size = 2, stride = 2)
        self.conv1 = nn.Conv2d(1,32,3,stride = 1, padding = 2)
        self.conv2 = nn.Conv2d(32,64,3,stride = 1, padding = 1)
        self.conv3 = nn.Conv2d(64,128,3,stride = 1, padding = 1)
        self.avgpool = nn.AdaptiveAvgPool2d((7, 7))
        self.dropout = nn.Dropout(p = 0.1)
        self.fc1 = nn.Linear(7*7*128, 124)
        self.fc2 = nn.Linear(124,32)
        self.fc3 = nn.Linear(32, 5)

    def forward(self, x):
        x = self.maxpool(F.leaky_relu(self.conv1(x)))
        x = self.maxpool(F.leaky_relu(self.conv2(x)))
        x = self.maxpool(F.leaky_relu(self.conv3(x)))
        x = self.avgpool(x)
        x = x.view(-1, 7*7*128)
        x = self.dropout(F.leaky_relu(self.fc1(x)))
        x = self.dropout(F.leaky_relu(self.fc2(x)))
        x = self.fc3(x)

        return x

model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate,
                               weight_decay = weight_decay)
```

```
In [7]: from prettytable import PrettyTable

def count_parameters(model):
    table = PrettyTable(["Modules", "Parameters"])
    total_params = 0
    for name, parameter in model.named_parameters():
        if not parameter.requires_grad:
            continue
        param = parameter.numel()
        table.add_row([name, param])
        total_params+=param
    print(table)
    print(f"Total Trainable Params: {total_params}")
    return total_params

count_parameters(model)
```

Modules	Parameters
conv1.weight	288
conv1.bias	32
conv2.weight	18432
conv2.bias	64
conv3.weight	73728
conv3.bias	128
fc1.weight	777728
fc1.bias	124
fc2.weight	3968
fc2.bias	32
fc3.weight	160
fc3.bias	5

Total Trainable Params: 874689

Out [7]: 874689

- Tổng có 874689 tham số

- Tiếp theo ta sẽ tiến hành train và đánh giá model

```
In [8]:

n_total_steps = len(train_loader)
run_loss = 0.0
run_acc = 0.0
for epoch in range(epochs):
    for i, (images, labels) in enumerate(tqdm(train_loader)):
        images = images.reshape(-1,1,124,124).to(device)
        labels = labels.to(device)
        _, labels = torch.max(labels.data, 1)
        outputs = model(images)
        loss = criterion(outputs, labels)
        run_loss += loss.item() #for tensorboard
        _, pred = torch.max(outputs.data, 1)
        run_acc += (pred==labels).sum().item() #for tensorboard

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (i+1) % 2 == 0:
        with torch.no_grad():
            n_correct = 0
            n_samples = 0
            n_class_correct = [0 for i in range(5)]
            n_class_samples = [0 for i in range(5)]

            for images_val, labels_val in val_loader:
                images_val = images_val.reshape(-1,1,124,124).to(device)
                labels_val = labels_val.to(device)
                _, labels_val = torch.max(labels_val.data, 1)
                output_val = model(images_val)
                _, predict_val = torch.max(output_val.data, 1)
                n_samples += labels_val.size(0)
                n_correct += (predict_val == labels_val).sum().item()
                loss_val = F.mse_loss(labels_val.float(), predict_val.float())

                for i in range(26):
                    label_val = labels_val[i]
                    pred_val = predict_val[i]
                    if label_val == pred_val:
                        n_class_correct[label_val] += 1
                        n_class_samples[label_val] += 1

            acc_val = n_correct/(n_samples)
            run_acc_result = run_acc/((2-1)*128+pred.size(0))
            print(f'Epoch[{epoch+1}/{epochs}]:  Loss_Train: {(run_loss/2):.2f}, Acc_Train: {run_acc_result:.2f} , Loss_Val: {loss_val:.2f} , Acc_Val: {acc_val:.2f} ')

        run_loss = 0.0
        run_acc = 0.0

print('Finished Training')
```

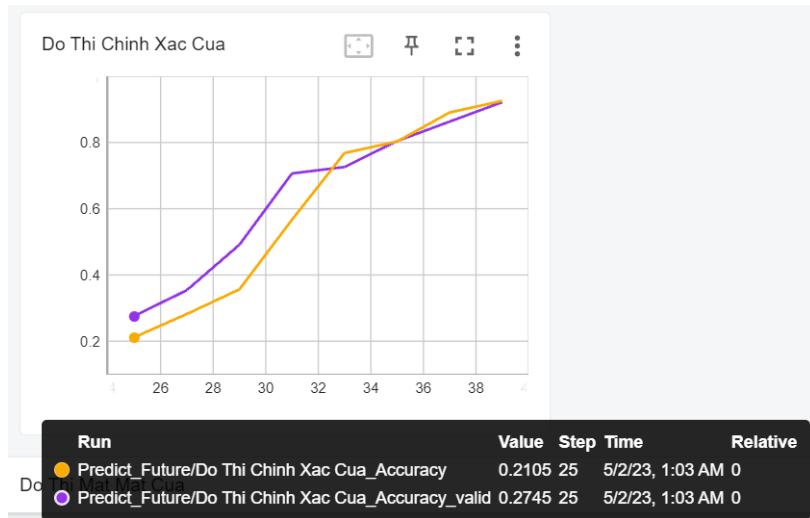
Out [8]:

```
100%[=====] 2/2 [00:15<00:00, 7.99s/it]
Epoch[1/12]: Loss_Train: 1.60, Acc_Train: 0.17, Loss_Val: 4.41, Acc_Val: 0.14
100%[=====] 2/2 [00:15<00:00, 7.77s/it]
Epoch[2/12]: Loss_Train: 1.56, Acc_Train: 0.36, Loss_Val: 2.41, Acc_Val: 0.29
100%[=====] 2/2 [00:15<00:00, 7.96s/it]
Epoch[3/12]: Loss_Train: 1.44, Acc_Train: 0.30, Loss_Val: 2.49, Acc_Val: 0.45
100%[=====] 2/2 [00:15<00:00, 7.68s/it]
Epoch[4/12]: Loss_Train: 1.19, Acc_Train: 0.62, Loss_Val: 1.12, Acc_Val: 0.76
100%[=====] 2/2 [00:15<00:00, 7.93s/it]
Epoch[5/12]: Loss_Train: 0.89, Acc_Train: 0.75, Loss_Val: 1.29, Acc_Val: 0.75
100%[=====] 2/2 [00:15<00:00, 7.73s/it]
Epoch[6/12]: Loss_Train: 0.54, Acc_Train: 0.78, Loss_Val: 1.35, Acc_Val: 0.80
100%[=====] 2/2 [00:16<00:00, 8.11s/it]
Epoch[7/12]: Loss_Train: 0.38, Acc_Train: 0.90, Loss_Val: 0.22, Acc_Val: 0.90
100%[=====] 2/2 [00:15<00:00, 7.68s/it]
Epoch[8/12]: Loss_Train: 0.32, Acc_Train: 0.90, Loss_Val: 0.53, Acc_Val: 0.90
100%[=====] 2/2 [00:16<00:00, 8.09s/it]
Epoch[9/12]: Loss_Train: 0.11, Acc_Train: 0.97, Loss_Val: 0.35, Acc_Val: 0.92
100%[=====] 2/2 [00:15<00:00, 7.69s/it]
Epoch[10/12]: Loss_Train: 0.20, Acc_Train: 0.94, Loss_Val: 0.16, Acc_Val: 0.96
100%[=====] 2/2 [00:16<00:00, 8.03s/it]
Epoch[11/12]: Loss_Train: 0.21, Acc_Train: 0.94, Loss_Val: 0.25, Acc_Val: 0.96
100%[=====] 2/2 [00:15<00:00, 7.70s/it]
Epoch[12/12]: Loss_Train: 0.10, Acc_Train: 0.99, Loss_Val: 0.25, Acc_Val: 0.96
Finished Training
```

```
In [9]: with torch.no_grad():
        n_correct = 0
        n_samples = 0
        n_class_correct = [0 for i in range(5)]
        n_class_samples = [0 for i in range(5)]
        for images, labels in val_loader:
            images = images.reshape(-1,1,124,124).to(device)
            labels = labels.to(device)
            _, labels = torch.max(labels.data, 1)
            output = model(images)
            _, predict = torch.max(output.data, 1)
            n_samples += labels.size(0)
            n_correct += (predict == labels).sum().item()
        for i in range(51):
            label = labels[i]
            pred = predict[i]
            if label == pred:
                n_class_correct[label] += 1
            n_class_samples[label] += 1
        acc = 100.0*n_correct/n_samples
        print(f'Accuracy of the network: {acc} %')
```

Out [9]: Accuracy of the network: 96.07843137254902 %

- Sau 12 epochs ta có thể thấy độ chính xác trên tập validation là 96%
- Dưới đây là đồ thị của độ chính xác. Màu tím của tập train và màu cam của tập validation. Chi tiết xem thêm tại đây [11]



- Dưới đây là đồ thị của độ mất mát. Màu xanh của tập train và màu hồng của tập validation. Chi tiết xem thêm tại đây [11]



- Tiến hành lưu và load model

```
In [10]:# Save model
        FILE = "model_face.pth"
        torch.save(model.state_dict(), FILE)
        # Load model
        loaded_model = CNN().to(device)
        loaded_model.load_state_dict(torch.load(FILE))
        loaded_model.eval()
```

Out [10]:

```
CNN(
  (maxpool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
                        ceil_mode=False)
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (dropout): Dropout(p=0.1, inplace=False)
  (fc1): Linear(in_features=6272, out_features=124, bias=True)
  (fc2): Linear(in_features=124, out_features=32, bias=True)
  (fc3): Linear(in_features=32, out_features=6, bias=True)
)
```

```
In [11]: # Tien hanh du doan tuong lai voi input la anh mat nguoi
        from torchvision import transforms
        import PIL.Image as Image
        url = '/kaggle/input/face-class/Face/Test/Trader.jpg'
        # Load anh
        img = Image.open(url)
        # Hien thi anh
        plt.imshow(img)
        plt.show()
        # Ap dung transform
        img_transformed = transform(img)
        # App load model
        test = loaded_model(img_transformed.to(device))
        print("Tuong lai lam: ",class_predict_future[torch.max(test.data,1)[
                                                                1].data])
```



Out [11]: Tuong lai Lam: Trader

3.2 5 kinds of flowers (rose, lotus, water lily, apricot, daisy, pink)

Code đầy đủ ở đây [7]

- Đầu tiên ta sẽ import các thư viện cần thiết cho việc xây dựng và thực thi model
- Ở đây ta sẽ dùng Cuda cho việc xây model trên GPU và nó giúp ta tăng tổng quá trình train và thực thi model

```
In [1]: import numpy as np
        from tqdm import tqdm
        import torch
        import torch.nn as nn
        import torchvision
        import torch.nn.functional as F
        from torch.utils.data import DataLoader
        from torchvision import datasets, transforms, models
        from torchvision.transforms import ToTensor
        import matplotlib.pyplot as plt
        from torch.utils.tensorboard import SummaryWriter

        device = torch.device('cuda' if torch.cuda.is_available() else 'cpu'
                               )
        print(device)
```

Out [1]: cuda

- Tiếp Theo ta sẽ xây dựng hàm transform để biến đổi trên dữ liệu (data augmentation) hoặc chuẩn hóa (data normalization) dữ liệu trước khi đưa vào huấn luyện mô hình.
- Việc sử dụng hàm transform giúp cho mô hình huấn luyện được đào tạo trên nhiều dữ liệu đa dạng hơn, giúp mô hình học được các đặc trưng tổng quát hơn và tránh overfitting trên dữ liệu huấn luyện.
- Quy trình của hàm transform: Thay đổi kích thước ảnh về dạng (54x54) ⇒ Chuyển về dạng tensor ⇒ Chuẩn hóa ma trận.

```
In [2]: transform = transforms.Compose([
        # xoay ngẫu nhiên ảnh với góc quay trong khoảng từ -10 đến 10 độ
        transforms.RandomRotation(10),
        # lật ngang ảnh theo chiều ngang với xác suất 50%
        transforms.RandomHorizontalFlip(),
        # thay đổi kích thước bên ngang nhất thành 54 pixel
        transforms.Resize(54),
        # cắt kích thước bên dài nhất ra khoảng 54 pixel từ trung tâm
        transforms.CenterCrop(54),
        transforms.ToTensor(), # Chuyển ma trận ảnh sang dạng tensor
                                # thì mới xây dựng được
                                # model bằng pytorch

        # để chuẩn hóa giá trị của ảnh
        # với giá trị trung bình và độ lệch chuẩn của các kênh màu
        # trong ảnh
        transforms.Normalize([0.485, 0.456, 0.406],
                              [0.229, 0.224, 0.225])
    ])
```

```

class FlowersDataset(torch.utils.data.Dataset):
    def __init__(self, root, transform=None):
        self.data = datasets.ImageFolder(root, transform=transform)
        self.classes = self.data.classes
        self.num_classes = len(self.classes)

    def __getitem__(self, index):
        img, label = self.data[index]
        one_hot_label = F.one_hot(torch.tensor(label), num_classes=
                                   self.num_classes).
                                   float()# Áp dụng one
                                   hot cho label

        return img, one_hot_label

    def __len__(self):
        return len(self.data)

x_train = FlowersDataset(root=("/kaggle/input/flower-s/FLOWERS/Train
                               "), transform = transform)
x_val = FlowersDataset(root = ('/kaggle/input/flower-s/FLOWERS/Valid
                               '), transform = transform)

```

```

In [3]: class_name = x_train.classes
        print("Tong class: ",len(class_name))
        print("Ten cac loai: ",class_name)

```

Out [3]:Tong class: 5

Ten cac loai: ['Apricots', 'Daisy', 'Lotus', 'Pink', 'Waterlily']

- Tiếp theo ta sẽ tiến hành thiết lập siêu tham số (hyperparameter) và chia dữ liệu theo batch cho model

- Ở đây ta sử dụng Weight Decay (L2 Regularization) bằng 0.0001 để giảm thiểu Overfitting [5]

```

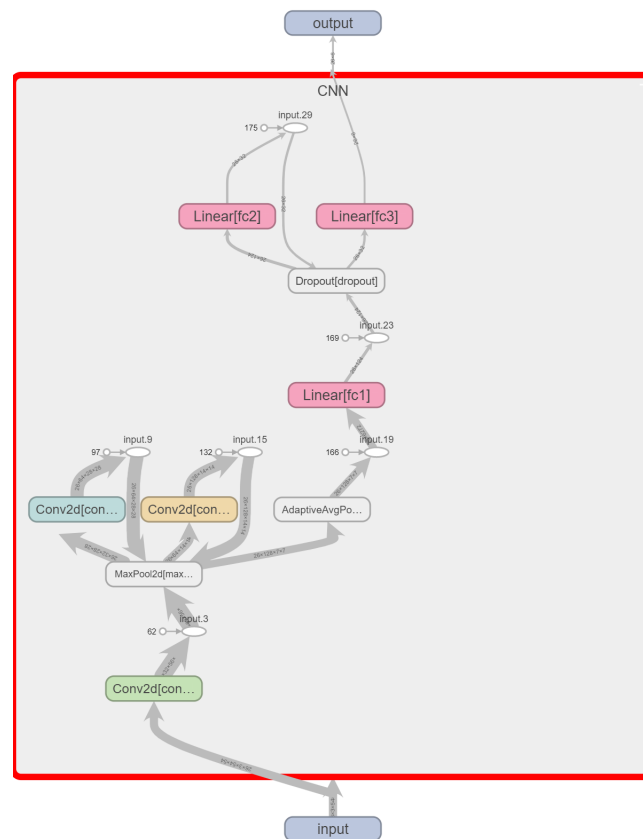
In [4]: batch = 128 # Khai bao kích thước batch
        epochs = 80 # Thuc hien 80 epochs
        learning_rate = 1e-3 # Khao bao tốc độ học là 10-3
        weight_decay = 1e-4 # Khai bao weight decay là 10-4 để giảm thiểu
                               overfitting

        # Chuyen doi dataset thanh iterable có thể duyệt qua theo batch size
        train_loader = DataLoader(x_train, batch_size = batch, shuffle =
                                   True)
        val_loader = DataLoader(x_val, batch_size = batch, shuffle = True)

```

- Tiếp theo ta sẽ tiến hành xây dựng model CNN

- Dưới đây là hình ảnh cấu trúc model CNN. Chi tiết xem thêm tại đây[12]



```
In [5]: class CNN(nn.Module):
def __init__(self):
    super(CNN, self).__init__()
    self.maxpool = nn.MaxPool2d(kernel_size = 2, stride = 2)
    self.conv1 = nn.Conv2d(3,32,3,stride = 1, padding = 2)
    self.conv2 = nn.Conv2d(32,64,3,stride = 1, padding = 1)
    self.conv3 = nn.Conv2d(64,128,3,stride = 1, padding = 1)
    self.avgpool = nn.AdaptiveAvgPool2d((7, 7))
    self.dropout = nn.Dropout(p = 0.1)
    self.fc1 = nn.Linear(7*7*128, 124)
    self.fc2 = nn.Linear(124,32)
    self.fc3 = nn.Linear(32, 6)

def forward(self, x):
    x = self.maxpool(F.leaky_relu(self.conv1(x)))
    x = self.maxpool(F.leaky_relu(self.conv2(x)))
    x = self.maxpool(F.leaky_relu(self.conv3(x)))
    x = self.avgpool(x)
    x = x.view(-1, 7*7*128)
    x = self.dropout(F.leaky_relu(self.fc1(x)))
    x = self.dropout(F.leaky_relu(self.fc2(x)))
    x = self.fc3(x)
    return x

model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate,
                                weight_decay = weight_decay)
```

```
In [6]: from prettytable import PrettyTable

def count_parameters(model):
    table = PrettyTable(["Modules", "Parameters"])
    total_params = 0
    for name, parameter in model.named_parameters():
        if not parameter.requires_grad:
            continue
        param = parameter.numel()
        table.add_row([name, param])
        total_params+=param
    print(table)
    print(f"Total Trainable Params: {total_params}")
    return total_params

count_parameters(model)
```

```
+-----+-----+
|  Modules  | Parameters |
+-----+-----+
| conv1.weight |    864    |
| conv1.bias   |     32    |
| conv2.weight |   18432   |
| conv2.bias   |     64    |
| conv3.weight |   73728   |
| conv3.bias   |    128    |
| fc1.weight   |  777728   |
| fc1.bias     |    124    |
| fc2.weight   |   3968    |
| fc2.bias     |     32    |
| fc3.weight   |    192    |
| fc3.bias     |      6    |
+-----+-----+
Total Trainable Params: 875298
```

Out [6]:875298

- Tổng có 875298 tham số
- Tiếp theo ta sẽ tiến hành train và đánh giá model

```

In [7]:
n_total_steps = len(train_loader)

run_loss = 0.0
run_acc = 0.0
for epoch in range(epochs):
    for i, (images, labels) in enumerate(tqdm(train_loader)):
        images = images.reshape(-1,3,54,54).to(device)
        labels = labels.to(device)
        _, labels = torch.max(labels.data, 1)

        outputs = model(images)
        loss = criterion(outputs, labels)

        run_loss += loss.item() #for tensorboard
        _, pred = torch.max(outputs.data, 1)
        run_acc += (pred==labels).sum().item() #for tensorboard

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (i+1) % 3 == 0:
        with torch.no_grad():
            n_correct = 0
            n_samples = 0
            n_class_correct = [0 for i in range(5)]
            n_class_samples = [0 for i in range(5)]

            for images_val, labels_val in val_loader:
                images_val = images_val.reshape(-1,3,54,54).to(device)
                labels_val = labels_val.to(device)
                _, labels_val = torch.max(labels_val.data, 1)

                output_val = model(images_val)
                _, predict_val = torch.max(output_val.data, 1)
                n_samples += labels_val.size(0)
                n_correct += (predict_val == labels_val).sum().item()
                loss_val = F.mse_loss(labels_val.float(), predict_val.float())

                for i in range(26):
                    label_val = labels_val[i]
                    pred_val = predict_val[i]
                    if label_val == pred_val:
                        n_class_correct[label_val] += 1

                n_class_samples[label_val] += 1

            acc_val = n_correct/(n_samples)
            run_acc_total = run_acc/((3-1)*128+pred.size(0))
            print(f'Epoch[{epoch+1}/{epochs}]: Loss of Train: {(
                run_loss/3):.2f},
                Accuracy of Train: {
                run_acc_total:.2f},
                Loss of Val: {loss_val
                :.2f} , Acc of Val: {
                acc_val:.2f}')

print('Finished Training')

```

Out [7]:

```
100%[=====] 3/3 [00:06<00:00, 2.05s/it]
Epoch[1/80]: Loss_Train: 1.67, Acc_Train: 0.32, Loss_Val: 1.54 , Acc_Val: 0.27
100%[=====] 3/3 [00:05<00:00, 1.85s/it]
Epoch[2/80]: Loss_Train: 1.48, Acc_Train: 0.36, Loss_Val: 2.08 , Acc_Val: 0.38
100%[=====] 3/3 [00:05<00:00, 1.86s/it]
Epoch[3/80]: Loss_Train: 1.39, Acc_Train: 0.35, Loss_Val: 2.27 , Acc_Val: 0.35
100%[=====] 3/3 [00:05<00:00, 1.79s/it]
Epoch[4/80]: Loss_Train: 1.40, Acc_Train: 0.40, Loss_Val: 1.46 , Acc_Val: 0.69
100%[=====] 3/3 [00:05<00:00, 1.89s/it]
Epoch[5/80]: Loss_Train: 1.30, Acc_Train: 0.45, Loss_Val: 1.65 , Acc_Val: 0.65
...
100%[=====] 3/3 [00:05<00:00, 1.78s/it]
Epoch[76/80]: Loss_Train: 0.07, Acc_Train: 0.99, Loss_Val: 0.23 , Acc_Val: 0.88
100%[=====] 3/3 [00:05<00:00, 1.80s/it]
Epoch[77/80]: Loss_Train: 0.09, Acc_Train: 0.98, Loss_Val: 0.42 , Acc_Val: 0.81
100%[=====] 3/3 [00:05<00:00, 1.80s/it]
Epoch[78/80]: Loss_Train: 0.10, Acc_Train: 0.97, Loss_Val: 0.50 , Acc_Val: 0.92
100%[=====] 3/3 [00:05<00:00, 1.82s/it]
Epoch[79/80]: Loss_Train: 0.06, Acc_Train: 0.98, Loss_Val: 0.73 , Acc_Val: 0.81
100%[=====] 3/3 [00:05<00:00, 1.72s/it]
Epoch[80/80]: Loss_Train: 0.07, Acc_Train: 0.97, Loss_Val: 0.69 , Acc_Val: 0.85
Finished Training
```

```
In [8]: with torch.no_grad():
        n_correct = 0
        n_samples = 0
        n_class_correct = [0 for i in range(5)]
        n_class_samples = [0 for i in range(5)]

        for images, labels in val_loader:
            images = images.reshape(-1,3,54,54).to(device)
            labels = labels.to(device)
            _, labels = torch.max(labels.data, 1)
            output = model(images)

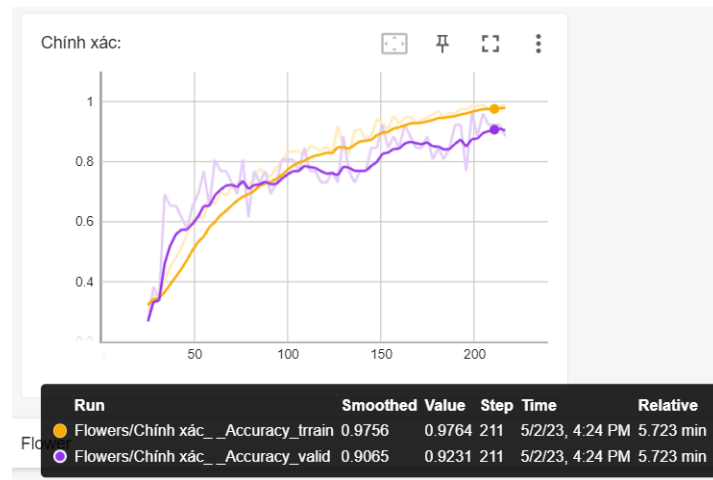
            _, predict = torch.max(output.data, 1)
            n_samples += labels.size(0)
            n_correct += (predict == labels).sum().item()

            for i in range(26):
                label = labels[i]
                pred = predict[i]
                if label == pred:
                    n_class_correct[label] +=1

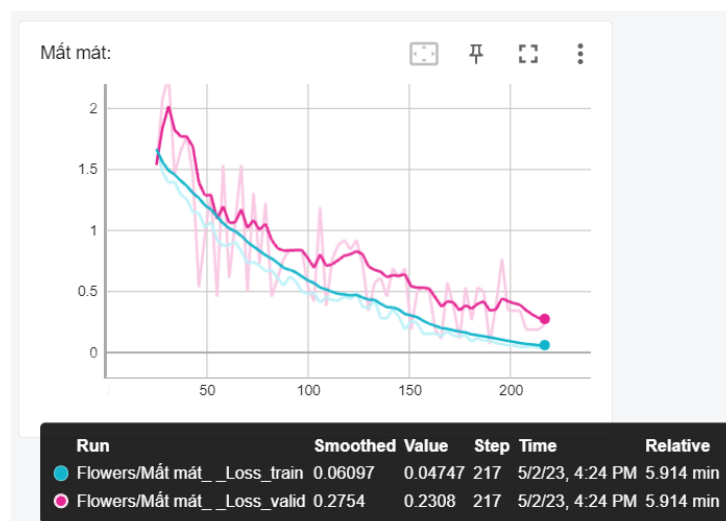
            n_class_samples[label] +=1
        acc = 100.0*n_correct/n_samples
        print(f'Accuracy of the network: {acc} %')
```

Out [8]: Accuracy of the network: 96.15384615384616 %

- Sau 80 epochs ta có thể thấy độ chính xác trên tập validation là 96 %
- Dưới đây là đồ thị của độ chính xác. Màu cam của tập train và màu tím của tập validation. Chi tiết xem thêm tại đây [12]



- Dưới đây là đồ thị của độ mất mát. Màu hồng của tập train và màu xanh của tập validation. Chi tiết xem thêm tại đây [12]



- Tiến hành lưu và load model

```
In [9]:# Save model
FILE = "model.pth"
torch.save(model.state_dict(), FILE)
# Load model
loaded_model = CNN().to(device)
loaded_model.load_state_dict(torch.load(FILE))
loaded_model.eval()
```

Out [9]:

```
CNN(
  (maxpool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
                        ceil_mode=False)
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (dropout): Dropout(p=0.1, inplace=False)
  (fc1): Linear(in_features=6272, out_features=124, bias=True)
  (fc2): Linear(in_features=124, out_features=32, bias=True)
  (fc3): Linear(in_features=32, out_features=6, bias=True)
)
```

- Cuối cùng ta tiến hành thử nghiệm model với ảnh từ tập test.

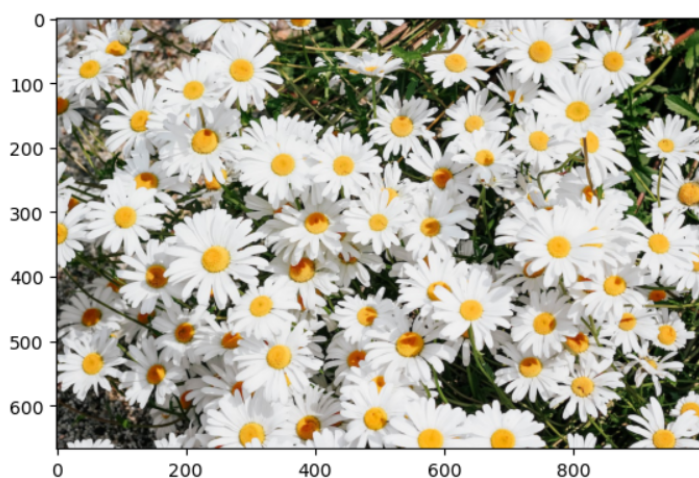
```
In [10]: from torchvision import transforms
import PIL.Image as Image

url = '/kaggle/input/flower-s/FLOWERS/Test/Daisy.jpg'
# Load ảnh
img = Image.open(url)

# Hien thi ảnh
plt.imshow(img)
plt.show()
# Áp dụng transform

img_transformed = transform(img)

test = loaded_model(img_transformed.to(device))
print(class_name[torch.max(test.data, 1)[1].data])
```



Out [10]: Daisy

3.3 10 Vietnamese dishes (bún bò, chè, bánh xèo.....)

Code đầy đủ ở đây [8]

- Đầu tiên ta sẽ import các thư viện cần thiết cho việc xây dựng và thực thi model
- Ở đây ta sẽ dùng Cuda cho việc xây model trên GPU và nó giúp ta tăng tổng quá trình train và thực thi model

```
In [1]: import numpy as np
        from tqdm import tqdm
        import torch
        import torch.nn as nn
        import torchvision
        import torch.nn.functional as F
        from torch.utils.data import DataLoader
        from torchvision import datasets, transforms, models
        from torchvision.transforms import ToTensor
        import matplotlib.pyplot as plt
        from torch.utils.tensorboard import SummaryWriter

        device = torch.device('cuda' if torch.cuda.is_available() else 'cpu'
                               )

        print(device)
```

Out [1]: cuda

- Tiếp Theo ta sẽ xây dựng hàm transform để biến đổi trên dữ liệu (data augmentation) hoặc chuẩn hóa (data normalization) dữ liệu trước khi đưa vào huấn luyện mô hình.
- Việc sử dụng hàm transform giúp cho mô hình huấn luyện được đào tạo trên nhiều dữ liệu đa dạng hơn, giúp mô hình học được các đặc trưng tổng quát hơn và tránh overfitting trên dữ liệu huấn luyện.
- Quy trình của hàm transform: Thay đổi kích thước ảnh về dạng (224x224) \Rightarrow Chuyển về dạng tensor \Rightarrow Chuẩn hóa ma trận.

```
In [2]: transform = transforms.Compose([
        transforms.RandomRotation(10),
        transforms.RandomHorizontalFlip(),
        transforms.Resize(224),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                               [0.229, 0.224, 0.225])
    ])

    class FoodDataset(torch.utils.data.Dataset):
        def __init__(self, root, transform = None):
            self.data = datasets.ImageFolder(root, transform = transform)

            self.classes = self.data.classes
            self.num_classes = len(self.classes)

        def __getitem__(self, index):
            img, label = self.data[index]
            one_hot_label = F.one_hot(torch.tensor(label), num_classes=
                                     self.num_classes).
                                     float()

            return img, one_hot_label
```

```

def __len__(self):
    return len(self.data)

x_train = FoodDataset( root=("/kaggle/input/vn-food/Food/Train"),
                        transform = transform)
x_val = FoodDataset(root = ('/kaggle/input/vn-food/Food/Val'),
                    transform = transform)

```

- Tiếp theo ta sẽ tiến hành thiết lập siêu tham số (hyperparameter) và chia dữ liệu theo batch cho model

- Ở đây ta sử dụng Weight Decay (L2 Regularization) bằng 0.0001 để giảm thiểu Overfitting [5]

```

In [3]: class_name = x_train.classes
        print("Tong class: ",len(class_name))
        print("Ten cac loai: ",class_name)

        batch = 128
        epochs = 80
        learning_rate = 1e-3
        weight_decay = 1e-4

        train_loader = DataLoader(x_train, batch_size = batch, shuffle =
                                True)
        val_loader = DataLoader(x_val, batch_size = batch, shuffle = True)

        examples = iter(val_loader)
        example_data, example_targets = next(examples)

```

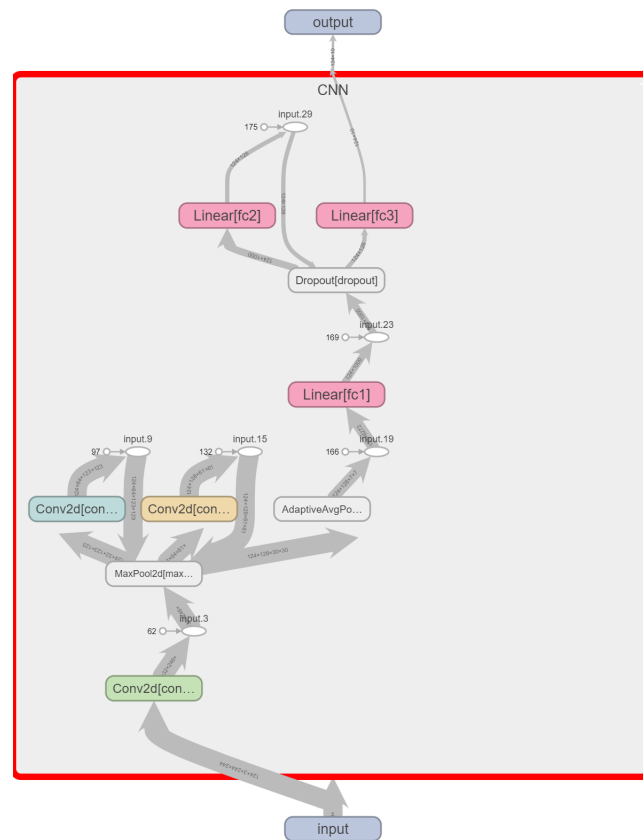
Out [3]: Tong class: 10

```

Ten cac loai: ['Banh bot loc', 'Banh chung', 'Banh khot', 'Banh mi',
               'Banh trang nuong', 'Bun bo Hue', 'Bun thit nuong',
               'Chao long', 'Com tam', 'Xoi xeo']

```

- Tiếp theo ta sẽ tiến hành xây dựng model CNN
- Dưới đây là hình ảnh cấu trúc model CNN. Chi tiết xem thêm tại đây[13]



```
In [4]: class CNN(nn.Module):
def __init__(self):
    super(CNN, self).__init__()
    self.maxpool = nn.MaxPool2d(kernel_size = 2, stride = 2)
    self.conv1 = nn.Conv2d(3,32,3,stride = 1, padding = 2)
    self.conv2 = nn.Conv2d(32,64,3,stride = 1, padding = 1)
    self.conv3 = nn.Conv2d(64,128,3,stride = 1, padding = 1)
    self.avgpool = nn.AdaptiveAvgPool2d((7, 7))
    self.dropout = nn.Dropout(p = 0.5)
    self.fc1 = nn.Linear(7*7*128, 1000)
    self.fc2 = nn.Linear(1000,128)
    self.fc3 = nn.Linear(128, 10)
def forward(self, x):
    x = self.maxpool(F.leaky_relu(self.conv1(x)))
    x = self.maxpool(F.leaky_relu(self.conv2(x)))
    x = self.maxpool(F.leaky_relu(self.conv3(x)))
    x = self.avgpool(x)
    x = x.view(-1, 7*7*128)
    x = self.dropout(F.leaky_relu(self.fc1(x)))
    x = self.dropout(F.leaky_relu(self.fc2(x)))
    x = self.fc3(x)
    return x

model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate,
                               weight_decay = weight_decay)
```

```
In [7]: from prettytable import PrettyTable

def count_parameters(model):
    table = PrettyTable(["Modules", "Parameters"])
    total_params = 0
    for name, parameter in model.named_parameters():
        if not parameter.requires_grad:
            continue
        param = parameter.numel()
        table.add_row([name, param])
        total_params += param
    print(table)
    print(f"Total Trainable Params: {total_params}")
    return total_params

count_parameters(model)
```

```
+-----+-----+
|  Modules  | Parameters |
+-----+-----+
| conv1.weight |    864    |
| conv1.bias   |     32    |
| conv2.weight |   18432   |
| conv2.bias   |     64    |
| conv3.weight |   73728   |
| conv3.bias   |    128    |
| fc1.weight   |  6272000  |
| fc1.bias     |    1000   |
| fc2.weight   |   128000  |
| fc2.bias     |    128    |
| fc3.weight   |    1280   |
| fc3.bias     |     10    |
+-----+-----+
```

Total Trainable Params: 6495666

Out [7]: 6495666

- Tổng có 6495666 tham số
- Tiếp theo ta sẽ tiến hành train và đánh giá model

```

In [8]:
n_total_steps = len(train_loader)

run_loss = 0.0
run_acc = 0.0
for epoch in range(epochs):
    for i, (images, labels) in enumerate(tqdm(train_loader)):
        images = images.reshape(-1,3,244,244).to(device)
        labels = labels.to(device)
        _, labels = torch.max(labels.data, 1)
        outputs = model(images)
        loss = criterion(outputs, labels)

        run_loss += loss.item() #for tensorboard
        _, pred = torch.max(outputs.data, 1)
        run_acc += (pred==labels).sum().item() #for tensorboard

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (i+1) % 3 == 0:
        with torch.no_grad():
            n_correct = 0
            n_samples = 0
            n_class_correct = [0 for i in range(10)]
            n_class_samples = [0 for i in range(10)]

            for images_val, labels_val in val_loader:
                images_val = images_val.reshape(-1,3,244,244).to(device)
                labels_val = labels_val.to(device)
                _, labels_val = torch.max(labels_val.data, 1)

                output_val = model(images_val)
                _, predict_val = torch.max(output_val.data, 1)
                n_samples += labels_val.size(0)
                n_correct += (predict_val == labels_val).sum().item()
                loss_val = F.mse_loss(labels_val.float(), predict_val.float())

                for i in range(26):
                    label_val = labels_val[i]
                    pred_val = predict_val[i]
                    if label_val == pred_val:
                        n_class_correct[label_val] += 1

                    n_class_samples[label_val] += 1

            acc_val = n_correct/(n_samples)
            acc_val_total = run_acc/((3-1)*128+pred.size(0))

            print(f'Epoch[{epoch+1}/{epochs}]: Loss_Train: {(run_loss/3):.2f}, Acc_Train: {acc_val_total:.2f}, Loss_Val: {loss_val:.2f}, Acc_Val: {acc_val:.2f}')

print('Finished Training')

```

Out [8]:

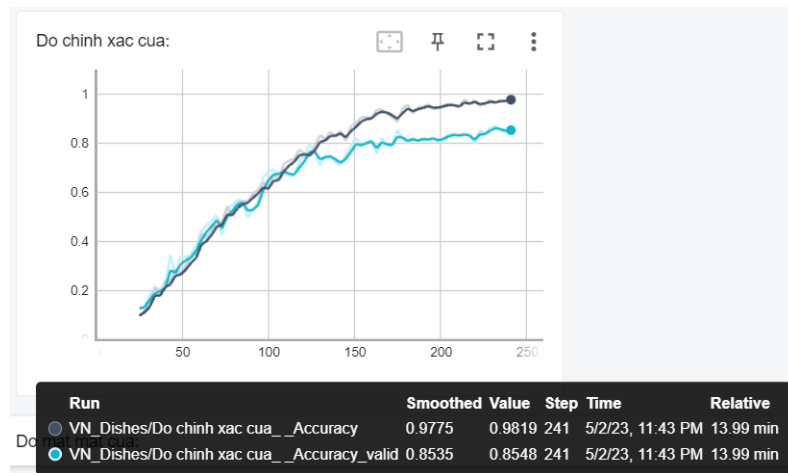
```
100%[=====] 3/3 [00:12<00:00, 4.16s/it]
Epoch[1/80]: Loss_Train: 2.32, Acc_Train: 0.10 , Loss_Val: 12.56 , Acc_Val: 0.13
100%[=====] 3/3 [00:11<00:00, 3.98s/it]
Epoch[2/80]: Loss_Train: 2.29, Acc_Train: 0.12 , Loss_Val: 13.07 , Acc_Val: 0.14
100%[=====] 3/3 [00:11<00:00, 3.86s/it]
Epoch[3/80]: Loss_Train: 2.27, Acc_Train: 0.15 , Loss_Val: 12.53 , Acc_Val: 0.19
100%[=====] 3/3 [00:11<00:00, 3.80s/it]
Epoch[4/80]: Loss_Train: 2.20, Acc_Train: 0.22 , Loss_Val: 15.38 , Acc_Val: 0.21
100%[=====] 3/3 [00:11<00:00, 3.91s/it]
Epoch[5/80]: Loss_Train: 2.16, Acc_Train: 0.18 , Loss_Val: 14.32 , Acc_Val: 0.21
...
100%[=====] 3/3 [00:11<00:00, 3.85s/it]
Epoch[76/80]: Loss_Train: 0.12, Acc_Train: 0.96 , Loss_Val: 1.44 , Acc_Val: 0.90
100%[=====] 3/3 [00:11<00:00, 3.86s/it]
Epoch[77/80]: Loss_Train: 0.05, Acc_Train: 0.99 , Loss_Val: 1.93 , Acc_Val: 0.86
100%[=====] 3/3 [00:11<00:00, 3.94s/it]
Epoch[78/80]: Loss_Train: 0.12, Acc_Train: 0.97 , Loss_Val: 3.54 , Acc_Val: 0.82
100%[=====] 3/3 [00:11<00:00, 3.83s/it]
Epoch[79/80]: Loss_Train: 0.11, Acc_Train: 0.96 , Loss_Val: 2.65 , Acc_Val: 0.81
100%[=====] 3/3 [00:11<00:00, 3.84s/it]
Epoch[80/80]: Loss_Train: 0.11, Acc_Train: 0.97 , Loss_Val: 3.52 , Acc_Val: 0.82
Finished Training
```

```
In [9]: with torch.no_grad():
        n_correct = 0
        n_samples = 0
        n_class_correct = [0 for i in range(10)]
        n_class_samples = [0 for i in range(10)]
        for images, labels in val_loader:
            images = images.reshape(-1,3,244,244).to(device)
            labels = labels.to(device)
            _, labels = torch.max(labels.data, 1)
            output = model(images)
            _, predict = torch.max(output.data, 1)
            n_samples += labels.size(0)
            n_correct += (predict == labels).sum().item()
            for i in range(87):
                label = labels[i]
                pred = predict[i]
                if label == pred:
                    n_class_correct[label] +=1

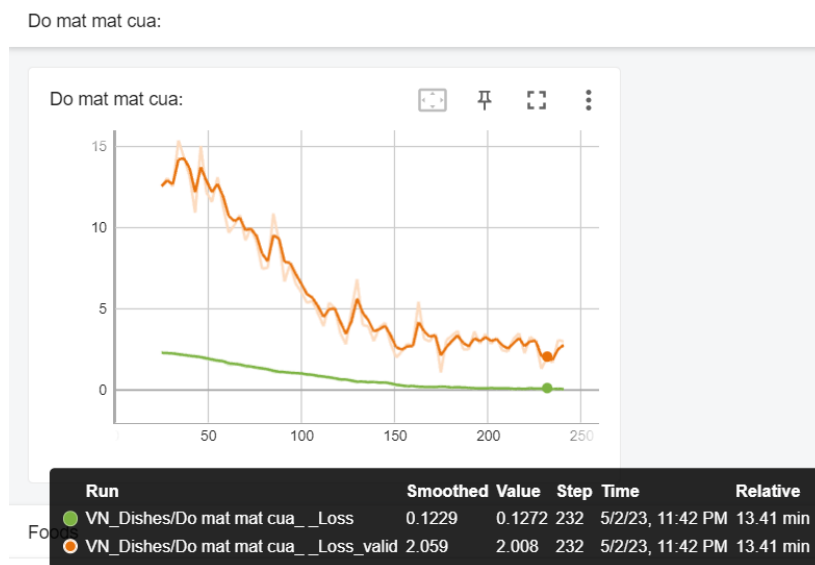
            n_class_samples[label] +=1
        acc = 100.0*n_correct/n_samples
        print(f'Accuracy of the network: {acc} %')
```

Out [8]: Accuracy of the network: 83.06451612903226 %

- Sau 80 epochs ta có thể thấy độ chính xác trên tập validation là 83 %
- Dưới đây là đồ thị của độ chính xác. Màu xanh đen của tập train và màu xanh sáng của tập validation. Chi tiết xem thêm tại đây [13]



- Dưới đây là đồ thị của độ mất mát. Màu cam của tập train và màu xanh lá của tập validation. Chi tiết xem thêm tại đây [13]



- Tiến hành lưu và load model

```
In [9]:# Save model
FILE = "model.pth"
torch.save(model.state_dict(), FILE)
# Load model
loaded_model = CNN().to(device)
loaded_model.load_state_dict(torch.load(FILE))
loaded_model.eval()
```

```

Out [9]:CNN(
  (maxpool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
                        ceil_mode=False)
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (dropout): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=6272, out_features=1000, bias=True)
  (fc2): Linear(in_features=1000, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=10, bias=True)
)

```

- Cuối cùng ta tiến hành thử nghiệm model với ảnh từ tập test.

```

In [10]:from torchvision import transforms
import PIL.Image as Image
url = '/kaggle/input/vn-food/Food/Test/comtam.jpg'

# Load ảnh
img = Image.open(url)
# Hien thi ảnh
plt.imshow(img)
plt.show()

# Ap dung transform
img_transformed = transform(img)
test = loaded_model(img_transformed.to(device))
print(class_name[torch.max(test.data,1)[1].data])

```



Out[10]:Com tam

3.4 VN banknotes (5000vnd, 10000vnd, 20k, 50k, 100k, 500k)

Code đầy đủ ở đây [9]

- Đầu tiên ta sẽ import các thư viện cần thiết cho việc xây dựng và thực thi model
- Ở đây ta sẽ dùng Cuda cho việc xây model trên GPU và nó giúp ta tăng tổng quá trình train và thực thi model

```
In [1]: import numpy as np
        from tqdm import tqdm
        import torch
        import torch.nn as nn
        import torchvision
        import torch.nn.functional as F
        from torch.utils.data import DataLoader
        from torchvision import datasets, transforms, models
        from torchvision.transforms import ToTensor
        import matplotlib.pyplot as plt
        from torch.utils.tensorboard import SummaryWriter

        device = torch.device('cuda' if torch.cuda.is_available() else 'cpu'
                               )

        print(device)
```

Out [1]: cuda

- Tiếp Theo ta sẽ xây dựng hàm transform để biến đổi trên dữ liệu (data augmentation) hoặc chuẩn hóa (data normalization) dữ liệu trước khi đưa vào huấn luyện mô hình.
- Việc sử dụng hàm transform giúp cho mô hình huấn luyện được đào tạo trên nhiều dữ liệu đa dạng hơn, giúp mô hình học được các đặc trưng tổng quát hơn và tránh overfitting trên dữ liệu huấn luyện.
- Quy trình của hàm transform: Thay đổi kích thước ảnh về dạng (64x64) ⇒ Chuyển về dạng tensor ⇒ Chuẩn hóa ma trận.

```
In [2]: transform = transforms.Compose([
        transforms.RandomRotation(10),
        transforms.RandomHorizontalFlip(),
        transforms.Resize((64,64)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                              [0.229, 0.224, 0.225])
        ])

        class MoneyDataset(torch.utils.data.Dataset):
            def __init__(self, root, transform = None):
                self.data = datasets.ImageFolder(root, transform = transform
                )

                self.classes = self.data.classes
                self.num_classes = len(self.classes)

            def __getitem__(self, index):
                img, label = self.data[index]
                one_hot_label = F.one_hot(torch.tensor(label), num_classes=
                self.num_classes).
                float()

                return img, one_hot_label
```

```

def __len__(self):
    return len(self.data)

x_train = MoneyDataset( root=("/kaggle/input/vn-money/Money/Train"),
                        transform = transform)
x_val = MoneyDataset(root = ('/kaggle/input/vn-money/Money/Val'),
                    transform = transform)

```

- Tiếp theo ta sẽ tiến hành thiết lập siêu tham số (hyperparameter) và chia dữ liệu theo batch cho model
- Ở đây ta sử dụng Weight Decay (L2 Regularization) bằng 0.0001 để giảm thiểu Overfitting [5]

```

In [3]: class_name = x_train.classes
print("Tong class: ",len(class_name))
print("Ten cac loai: ",class_name)

batch = 128
epochs = 80
learning_rate = 3*1e-3
weight_decay = 1e-4

train_loader = DataLoader(x_train, batch_size = batch, shuffle =
                          True)
val_loader = DataLoader(x_val, batch_size = batch, shuffle = True)

examples = iter(val_loader)
example_data, example_targets = next(examples)

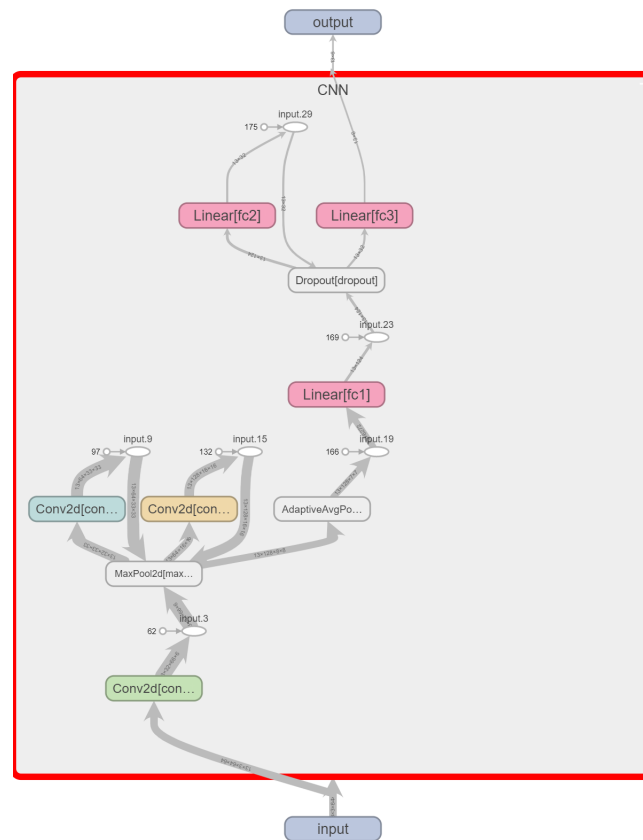
```

```

Out [3]: Tong class:  6
Ten cac loai:  ['100k', '10k', '20k', '500k', '50k', '5k']

```

- Tiếp theo ta sẽ tiến hành xây dựng model CNN
- Dưới đây là hình ảnh cấu trúc model CNN. Chi tiết xem thêm tại đây[14]



```
In [4]: class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.maxpool = nn.MaxPool2d(kernel_size = 2, stride = 2)
        self.conv1 = nn.Conv2d(3,32,3,stride = 1, padding = 2)
        self.conv2 = nn.Conv2d(32,64,3,stride = 1, padding = 1)
        self.conv3 = nn.Conv2d(64,128,3,stride = 1, padding = 1)
        self.avgpool = nn.AdaptiveAvgPool2d((7, 7))
        self.dropout = nn.Dropout(p = 0.1)
        self.fc1 = nn.Linear(7*7*128, 124)
        self.fc2 = nn.Linear(124,32)
        self.fc3 = nn.Linear(32, 6)

    def forward(self, x):
        x = self.maxpool(F.leaky_relu(self.conv1(x)))
        x = self.maxpool(F.leaky_relu(self.conv2(x)))
        x = self.maxpool(F.leaky_relu(self.conv3(x)))
        x = self.avgpool(x)
        x = x.view(-1, 7*7*128)
        x = self.dropout(F.leaky_relu(self.fc1(x)))
        x = self.dropout(F.leaky_relu(self.fc2(x)))
        x = self.fc3(x)
        return x

model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate,
                               weight_decay = weight_decay)
```

```
In [5]: from prettytable import PrettyTable

def count_parameters(model):
    table = PrettyTable(["Modules", "Parameters"])
    total_params = 0
    for name, parameter in model.named_parameters():
        if not parameter.requires_grad:
            continue
        param = parameter.numel()
        table.add_row([name, param])
        total_params+=param
    print(table)
    print(f"Total Trainable Params: {total_params}")
    return total_params

count_parameters(model)
```

```
+-----+-----+
|  Modules  | Parameters |
+-----+-----+
| conv1.weight |    864    |
| conv1.bias   |     32    |
| conv2.weight |   18432   |
| conv2.bias   |     64    |
| conv3.weight |   73728   |
| conv3.bias   |    128    |
| fc1.weight   |  777728   |
| fc1.bias     |    124    |
| fc2.weight   |   3968    |
| fc2.bias     |     32    |
| fc3.weight   |    192    |
| fc3.bias     |      6    |
+-----+-----+
```

Total Trainable Params: 875298

Out [5]: 875298

- Tổng có 875298 tham số
- Tiếp theo ta sẽ tiến hành train và đánh giá model

```

In [6]:
n_total_steps = len(train_loader)
run_loss = 0.0
run_acc = 0.0
for epoch in range(epochs):
    for i, (images, labels) in enumerate(tqdm(train_loader)):
        images = images.reshape(-1,3,64,64).to(device)
        labels = labels.to(device)
        _, labels = torch.max(labels.data, 1)
        outputs = model(images)
        loss = criterion(outputs, labels)
        run_loss += loss.item() #for tensorboard
        _, pred = torch.max(outputs.data, 1)
        run_acc += (pred==labels).sum().item() #for tensorboard

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (i+1) % 3 == 0:
        with torch.no_grad():
            n_correct = 0
            n_samples = 0
            n_class_correct = [0 for i in range(6)]
            n_class_samples = [0 for i in range(6)]

            for images_val, labels_val in val_loader:
                images_val = images_val.reshape(-1,3,64,64).to(device)
                labels_val = labels_val.to(device)
                _, labels_val = torch.max(labels_val.data, 1)

                output_val = model(images_val)
                _, predict_val = torch.max(output_val.data, 1)
                n_samples += labels_val.size(0)
                n_correct += (predict_val == labels_val).sum().item()
                loss_val = F.mse_loss(labels_val.float(), predict_val.
                                     float())

                for i in range(13):
                    label_val = labels_val[i]
                    pred_val = predict_val[i]
                    if label_val == pred_val:
                        n_class_correct[label_val] += 1

                n_class_samples[label_val] += 1

            acc_val = n_correct/(n_samples)
            acc_total = run_acc/((3-1)*128+pred.size(0))
            print(f'Epoch[{epoch+1}/{epochs}]: Loss_Train: {(run_loss/3)
                :.2f}, Acc_Train: {
                acc_total:.2f} ,
                Loss_Val: {loss_val:.
                2f} , Acc_Val: {
                acc_val:.2f} ')

print('Finished Training')

```

Out [6]:

```
100%[=====] 3/3 [00:01<00:00, 2.07it/s]
Epoch[1/80]: Loss_Train: 1.73, Acc_Train: 0.41 , Loss_Val: 4.85 , Acc_Val: 0.15
100%[=====] 3/3 [00:01<00:00, 2.37it/s]
Epoch[2/80]: Loss_Train: 1.49, Acc_Train: 0.52 , Loss_Val: 10.38 , Acc_Val: 0.15
100%[=====] 3/3 [00:01<00:00, 2.27it/s]
Epoch[3/80]: Loss_Train: 1.11, Acc_Train: 0.75 , Loss_Val: 10.38 , Acc_Val: 0.15
100%[=====] 3/3 [00:01<00:00, 2.17it/s]
Epoch[4/80]: Loss_Train: 0.85, Acc_Train: 0.73 , Loss_Val: 8.08 , Acc_Val: 0.31
100%[=====] 3/3 [00:01<00:00, 2.19it/s]
Epoch[5/80]: Loss_Train: 0.69, Acc_Train: 0.74 , Loss_Val: 4.38 , Acc_Val: 0.31
...
Epoch[76/80]: Loss_Train: 0.02, Acc_Train: 0.99 , Loss_Val: 0.08 , Acc_Val: 0.92
100%[=====] 3/3 [00:01<00:00, 2.44it/s]
Epoch[77/80]: Loss_Train: 0.01, Acc_Train: 0.99 , Loss_Val: 0.08 , Acc_Val: 0.92
100%[=====] 3/3 [00:01<00:00, 2.45it/s]
Epoch[78/80]: Loss_Train: 0.01, Acc_Train: 1.00 , Loss_Val: 0.08 , Acc_Val: 0.92
100%[=====] 3/3 [00:01<00:00, 2.45it/s]
Epoch[79/80]: Loss_Train: 0.01, Acc_Train: 1.00 , Loss_Val: 0.15 , Acc_Val: 0.85 %
100%[=====] 3/3 [00:01<00:00, 2.39it/s]
Epoch[80/80]: Loss_Train: 0.00, Acc_Train: 1.00 , Loss_Val: 0.08 , Acc_Val: 0.92
Finished Training
```

```
In [7]: with torch.no_grad():
        n_correct = 0
        n_samples = 0
        n_class_correct = [0 for i in range(6)]
        n_class_samples = [0 for i in range(6)]

        for images, labels in val_loader:
            images = images.reshape(-1,3,64,64).to(device)
            labels = labels.to(device)
            _, labels = torch.max(labels.data, 1)

            output = model(images)

            _, predict = torch.max(output.data, 1)
            n_samples += labels.size(0)
            n_correct += (predict == labels).sum().item()

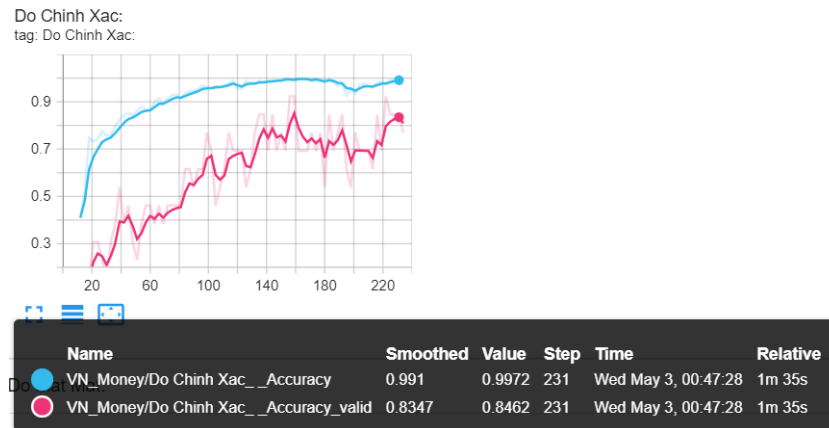
            for i in range(13):
                label = labels[i]
                pred = predict[i]
                if label == pred:
                    n_class_correct[label] +=1

            n_class_samples[label] +=1

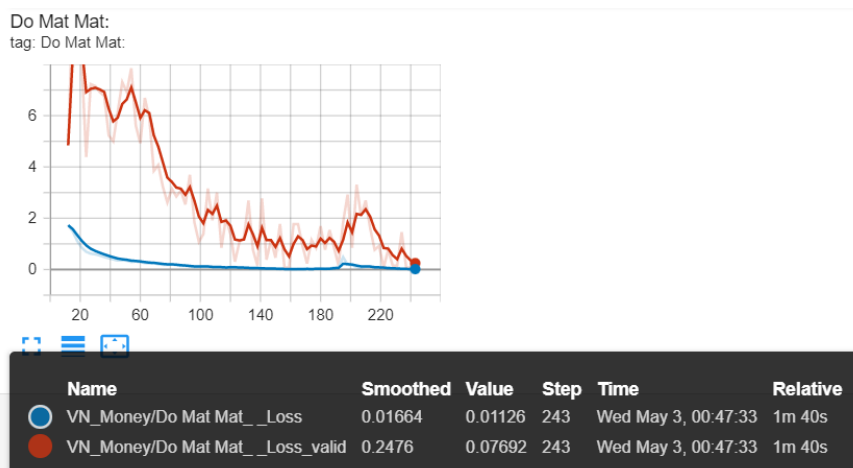
        acc = 100.0*n_correct/n_samples
        print(f'Accuracy of the network: {acc} %')
```

Out [7]: Accuracy of the network: 84.61538461538461 %

- Sau 80 epochs ta có thể thấy độ chính xác trên tập validation là 84 %
- Dưới đây là đồ thị của độ chính xác. Màu xanh của tập train và màu hồng của tập validation. Chi tiết xem thêm tại đây [14]



- Dưới đây là đồ thị của độ mất mát. Màu xanh của tập train và màu cam của tập validation. Chi tiết xem thêm tại đây [14]



- Tiến hành lưu và load model

```
In [8]: # Save model
FILE = "model.pth"
torch.save(model.state_dict(), FILE)

# Load model
loaded_model = CNN().to(device)
loaded_model.load_state_dict(torch.load(FILE)) # it takes the loaded
                                              dictionary, not the path file
                                              itself

loaded_model.eval()
```

Out [8]:

```
CNN(
  (maxpool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
                        ceil_mode=False)
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (dropout): Dropout(p=0.1, inplace=False)
  (fc1): Linear(in_features=6272, out_features=124, bias=True)
  (fc2): Linear(in_features=124, out_features=32, bias=True)
  (fc3): Linear(in_features=32, out_features=6, bias=True)
)
```

- Cuối cùng ta tiến hành thử nghiệm model với ảnh từ tập test.

```
In [9]: from torchvision import transforms
import PIL.Image as Image
url = '/kaggle/input/vn-money/Money/Test/500k.jpg'
# Load ảnh
img = Image.open(url)
# Hien thi ảnh
plt.imshow(img)
plt.show()
# Áp dụng transform
img_transformed = transform(img)
test = loaded_model(img_transformed.to(device))
print(class_name[torch.max(test.data, 1)[1].data])
```



Out [10]: 500k

3.5 Recognition of all members of class from face images (you collected)

Code đầy đủ ở đây [10]

- Đầu tiên ta sẽ import các thư viện cần thiết cho việc xây dựng và thực thi model
- Ở đây ta sẽ dùng Cuda cho việc xây model trên GPU và nó giúp ta tăng tổng quá trình train và thực thi model

```
In [1]: import numpy as np
        from tqdm import tqdm
        import torch
        import torch.nn as nn
        import torchvision
        import torch.nn.functional as F
        from torch.utils.data import DataLoader
        from torchvision import datasets, transforms, models
        from torchvision.transforms import ToTensor
        import matplotlib.pyplot as plt
        from torch.utils.tensorboard import SummaryWriter

        device = torch.device('cuda' if torch.cuda.is_available() else 'cpu'
                               )
        print(device)
```

Out [1]: cuda

- Tiếp Theo ta sẽ xây dựng hàm transform để biến đổi trên dữ liệu (data augmentation) hoặc chuẩn hóa (data normalization) dữ liệu trước khi đưa vào huấn luyện mô hình.
- Việc sử dụng hàm transform giúp cho mô hình huấn luyện được đào tạo trên nhiều dữ liệu đa dạng hơn, giúp mô hình học được các đặc trưng tổng quát hơn và tránh overfitting trên dữ liệu huấn luyện.
- Quy trình của hàm transform: Thay đổi kích thước ảnh về dạng (32x32) \Rightarrow Chuyển về dạng tensor \Rightarrow Chuẩn hóa ma trận.

```
In [2]: transform = transforms.Compose([
        transforms.RandomRotation(10),
        transforms.RandomHorizontalFlip(),
        transforms.Resize(32),
        transforms.CenterCrop(32),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                               [0.229, 0.224, 0.225])
    ])

    class FaceDataset(torch.utils.data.Dataset):
        def __init__(self, root, transform = None):
            self.data = datasets.ImageFolder(root, transform = transform)

            self.classes = self.data.classes
            self.num_classes = len(self.classes)

        def __getitem__(self, index):
            img, label = self.data[index]
            one_hot_label = F.one_hot(torch.tensor(label), num_classes=
                                      self.num_classes).
                                      float()
```

```

        return img, one_hot_label

    def __len__(self):
        return len(self.data)

x_train = FaceDataset( root=("/kaggle/input/face-class/Face/Train"),
                        transform = transform)
x_val = FaceDataset(root = ('/kaggle/input/face-class/Face/Val'),
                    transform = transform)

```

- Tiếp theo ta sẽ tiến hành thiết lập siêu tham số (hyperparameter) và chia dữ liệu theo batch cho model

- Ở đây ta sử dụng Weight Decay (L2 Regularization) bằng 0.0001 để giảm thiểu Overfitting [5]

```

In [3]: print("Tong class: ",len(x_train.classes))
        print("Ten cac loai: ",x_train.classes)
        class_name = ['Cao Minh Quan', 'Huynh Anh Duy', 'Le Tri Dung', '
                        Nguyen Hai Hoang', 'Nguyen
                        Ngoc Nhan']

        batch = 128
        epochs = 12
        learning_rate = 1e-3
        weight_decay = 1e-4

        train_loader = DataLoader(x_train, batch_size = batch, shuffle =
                                True)
        val_loader = DataLoader(x_val, batch_size = batch, shuffle = True)

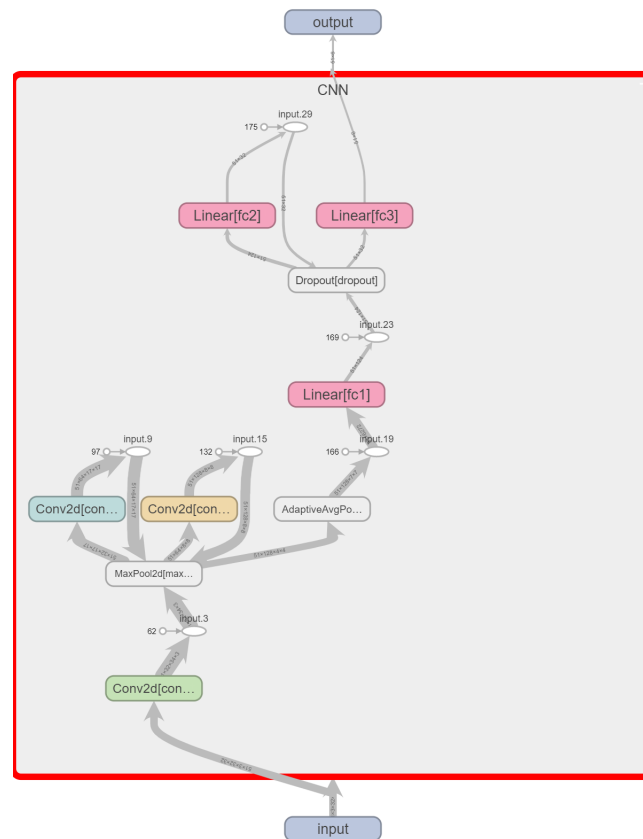
        examples = iter(val_loader)
        example_data, example_targets = next(examples)

```

Out [3]: Tong class: 5

Ten cac loai: ['CaoMinhQuan', 'HuynhAnhDUy', 'LeTriDung', 'NguyenHaiHoang', 'M

- Tiếp theo ta sẽ tiến hành xây dựng model CNN
- Dưới đây là hình ảnh cấu trúc model CNN. Chi tiết xem thêm tại đây[15]



```
In [4]: class CNN(nn.Module):
def __init__(self):
    super(CNN, self).__init__()
    self.maxpool = nn.MaxPool2d(kernel_size = 2, stride = 2)
    self.conv1 = nn.Conv2d(3,32,3,stride = 1, padding = 2)
    self.conv2 = nn.Conv2d(32,64,3,stride = 1, padding = 1)
    self.conv3 = nn.Conv2d(64,128,3,stride = 1, padding = 1)
    self.avgpool = nn.AdaptiveAvgPool2d((7, 7))
    self.dropout = nn.Dropout(p = 0.1)
    self.fc1 = nn.Linear(7*7*128, 124)
    self.fc2 = nn.Linear(124,32)
    self.fc3 = nn.Linear(32, 6)
def forward(self, x):
    x = self.maxpool(F.leaky_relu(self.conv1(x)))
    x = self.maxpool(F.leaky_relu(self.conv2(x)))
    x = self.maxpool(F.leaky_relu(self.conv3(x)))
    x = self.avgpool(x)
    x = x.view(-1, 7*7*128)
    x = self.dropout(F.leaky_relu(self.fc1(x)))
    x = self.dropout(F.leaky_relu(self.fc2(x)))
    x = self.fc3(x)
    return x
model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate,
                                weight_decay = weight_decay)
```

```
In [5]: from prettytable import PrettyTable

def count_parameters(model):
    table = PrettyTable(["Modules", "Parameters"])
    total_params = 0
    for name, parameter in model.named_parameters():
        if not parameter.requires_grad:
            continue
        param = parameter.numel()
        table.add_row([name, param])
        total_params+=param
    print(table)
    print(f"Total Trainable Params: {total_params}")
    return total_params

count_parameters(model)
```

```
+-----+-----+
|  Modules  | Parameters |
+-----+-----+
| conv1.weight |    864    |
| conv1.bias   |     32    |
| conv2.weight |   18432   |
| conv2.bias   |     64    |
| conv3.weight |   73728   |
| conv3.bias   |    128    |
| fc1.weight   |  777728   |
| fc1.bias     |    124    |
| fc2.weight   |   3968    |
| fc2.bias     |     32    |
| fc3.weight   |    192    |
| fc3.bias     |      6    |
+-----+-----+
```

Total Trainable Params: 875298

Out [5]: 875298

- Tổng có 875298 tham số
- Tiếp theo ta sẽ tiến hành train và đánh giá model

```

In [6]:
n_total_steps = len(train_loader)

run_loss = 0.0
run_acc = 0.0
for epoch in range(epochs):
    for i, (images, labels) in enumerate(tqdm(train_loader)):
        images = images.reshape(-1,3,32,32).to(device)
        labels = labels.to(device)
        _, labels = torch.max(labels.data, 1)

        outputs = model(images)

        loss = criterion(outputs, labels)

        run_loss += loss.item() #for tensorboard
        _, pred = torch.max(outputs.data, 1)
        run_acc += (pred==labels).sum().item() #for tensorboard

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (i+1) % 2 == 0:
        with torch.no_grad():
            n_correct = 0
            n_samples = 0
            n_class_correct = [0 for i in range(5)]
            n_class_samples = [0 for i in range(5)]

            for images_val, labels_val in val_loader:
                images_val = images_val.reshape(-1,3,32,32).to(device)
                labels_val = labels_val.to(device)
                _, labels_val = torch.max(labels_val.data, 1)

                output_val = model(images_val)
                _, predict_val = torch.max(output_val.data, 1)
                n_samples += labels_val.size(0)
                n_correct += (predict_val == labels_val).sum().item()
                loss_val = F.mse_loss(labels_val.float(), predict_val.
                                     float())

                for i in range(26):
                    label_val = labels_val[i]
                    pred_val = predict_val[i]
                    if label_val == pred_val:
                        n_class_correct[label_val] += 1

                    n_class_samples[label_val] += 1
            acc_val = n_correct/(n_samples)
            acc_total = run_acc/((2-1)*128+pred.size(0))
            print(f'Epoch[{epoch+1}/{epochs}]:  Loss_Train: {(run_loss/2
                ):.2f}, Acc_Train: {
                acc_total:.2f},
                Loss_Val: {loss_val:.
                2f}, Acc_Val: {acc_val
                :.2f}')
```

```

print('Finished Training')

```

```

100%[=====] 2/2 [00:22<00:00, 11.44s/it]
Epoch[1/12]: Loss_Train: 1.74, Acc_Train: 0.18, Loss_Val: 4.47, Acc_Val: 0.16
100%[=====] 2/2 [00:17<00:00, 8.71s/it]
Epoch[2/12]: Loss_Train: 1.54, Acc_Train: 0.29, Loss_Val: 3.37, Acc_Val: 0.33
100%[=====] 2/2 [00:17<00:00, 8.67s/it]
Epoch[3/12]: Loss_Train: 1.34, Acc_Train: 0.39, Loss_Val: 2.08, Acc_Val: 0.51
100%[=====] 2/2 [00:16<00:00, 8.37s/it]
Epoch[4/12]: Loss_Train: 1.05, Acc_Train: 0.62, Loss_Val: 1.67, Acc_Val: 0.63
100%[=====] 2/2 [00:17<00:00, 8.90s/it]
Epoch[5/12]: Loss_Train: 0.78, Acc_Train: 0.79, Loss_Val: 0.37, Acc_Val: 0.75
100%[=====] 2/2 [00:17<00:00, 8.57s/it]
Epoch[6/12]: Loss_Train: 0.51, Acc_Train: 0.82, Loss_Val: 0.27, Acc_Val: 0.84
100%[=====] 2/2 [00:17<00:00, 8.90s/it]
Epoch[7/12]: Loss_Train: 0.40, Acc_Train: 0.87, Loss_Val: 0.49, Acc_Val: 0.78
100%[=====] 2/2 [00:17<00:00, 8.88s/it]
Epoch[8/12]: Loss_Train: 0.34, Acc_Train: 0.88, Loss_Val: 0.22, Acc_Val: 0.84
100%[=====] 2/2 [00:17<00:00, 8.54s/it]
Epoch[9/12]: Loss_Train: 0.24, Acc_Train: 0.89, Loss_Val: 0.18, Acc_Val: 0.82
100%[=====] 2/2 [00:17<00:00, 8.84s/it]
Epoch[10/12]: Loss_Train: 0.12, Acc_Train: 0.93, Loss_Val: 0.12, Acc_Val: 0.88
100%[=====] 2/2 [00:17<00:00, 8.64s/it]
Epoch[11/12]: Loss_Train: 0.11, Acc_Train: 0.95, Loss_Val: 0.25, Acc_Val: 0.90
100%[=====] 2/2 [00:17<00:00, 8.79s/it]
Epoch[12/12]: Loss_Train: 0.08, Acc_Train: 0.96, Loss_Val: 0.08, Acc_Val: 0.92
Finished Training

```

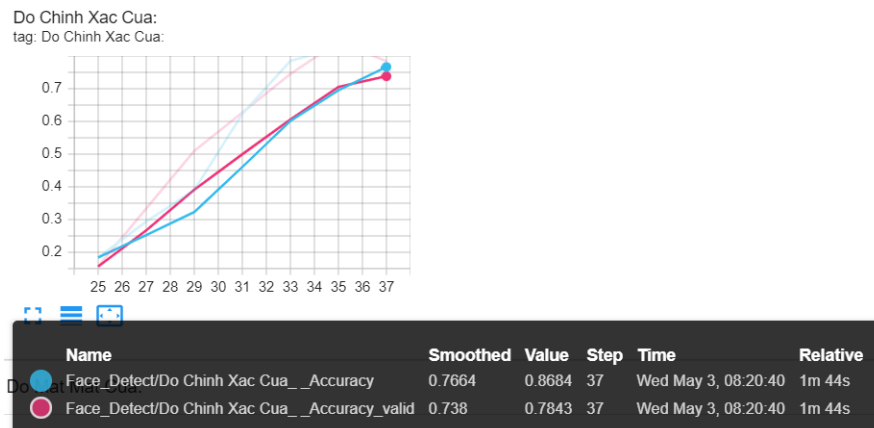
```

In [7]: with torch.no_grad():
        n_correct = 0
        n_samples = 0
        n_class_correct = [0 for i in range(5)]
        n_class_samples = [0 for i in range(5)]
        for images, labels in val_loader:
            images = images.reshape(-1,3,32,32).to(device)
            labels = labels.to(device)
            _, labels = torch.max(labels.data, 1)
            output = model(images)
            _, predict = torch.max(output.data, 1)
            n_samples += labels.size(0)
            n_correct += (predict == labels).sum().item()
        for i in range(51):
            label = labels[i]
            pred = predict[i]
            if label == pred:
                n_class_correct[label] +=1
            n_class_samples[label] +=1
        acc = 100.0*n_correct/n_samples
        print(f'Accuracy of the network: {acc} %')

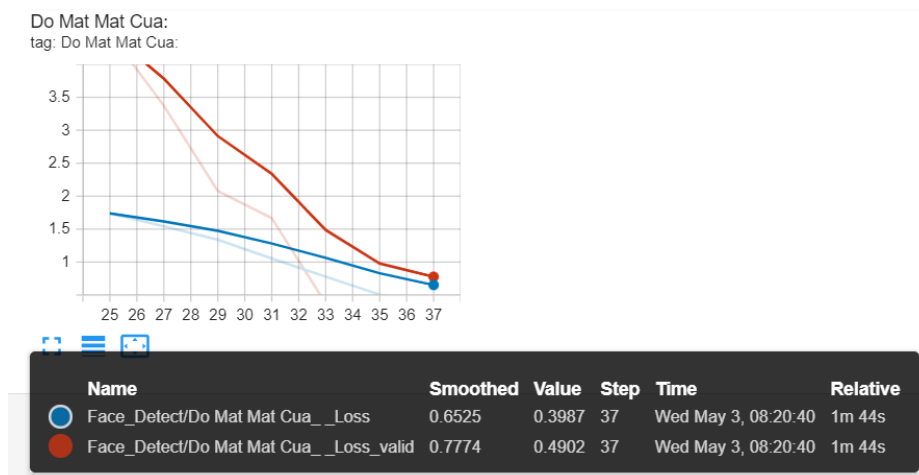
```

Out [8]: Accuracy of the network: 92.15686274509804 %

- Sau 12 epochs ta có thể thấy độ chính xác trên tập validation là 92 %
- Dưới đây là đồ thị của độ chính xác. Màu xanh của tập train và màu hồng của tập validation. Chi tiết xem thêm tại đây [15]



- Dưới đây là đồ thị của độ mất mát. Màu xanh của tập train và màu cam của tập validation. Chi tiết xem thêm tại đây [15]



- Tiến hành lưu và load model

```
In [9]: # Save model
        FILE = "model_face.pth"
        torch.save(model.state_dict(), FILE)

        # Load model
        loaded_model = CNN().to(device)
        loaded_model.load_state_dict(torch.load(FILE)) # it takes the loaded
                                                         dictionary, not the path file
                                                         itself

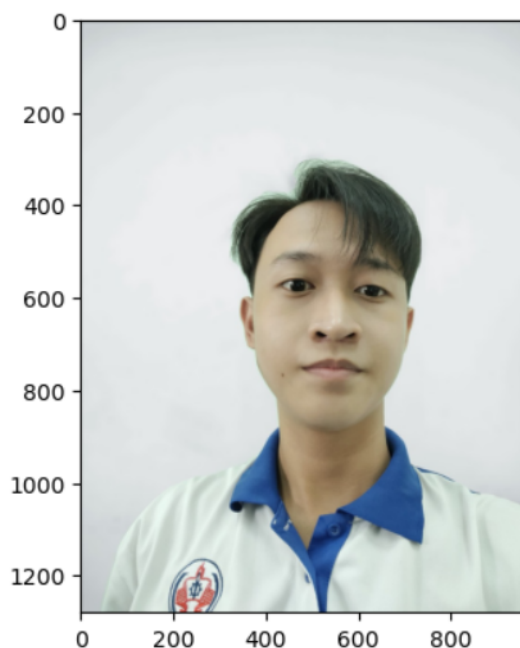
        loaded_model.eval()
```

Out [9]:

```
CNN(
  (maxpool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
                        ceil_mode=False)
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (dropout): Dropout(p=0.1, inplace=False)
  (fc1): Linear(in_features=6272, out_features=124, bias=True)
  (fc2): Linear(in_features=124, out_features=32, bias=True)
  (fc3): Linear(in_features=32, out_features=6, bias=True)
)
```

- Cuối cùng ta tiến hành thử nghiệm model với ảnh từ tập test.

```
In [10]: from torchvision import transforms
import PIL.Image as Image
url = '/kaggle/input/face-class/Face/Test/NguyenNgocNhan.jpg'
# Load ảnh
img = Image.open(url)
# Hien thi ảnh
plt.imshow(img)
plt.show()
# Áp dụng transform
img_transformed = transform(img)
# App load model
test = loaded_model(img_transformed.to(device))
print(class_name[torch.max(test.data,1)[1].data])
```



Out[10]: Nguyen Ngoc Nhan

4 Results - Kết quả

- Sau khi sử dụng CNN với độ chính xác trên 80% là một thành công hơn so với các mô hình machine learning trong việc xử lý ảnh và phân loại các đối tượng. Nó phụ thuộc rất nhiều vào các siêu tham số (hyperparameters) như epochs, batch size, dropout, weight decay, optimizer.
- Với CNN, chúng ta có thể tạo ra các mô hình phân loại ảnh chất lượng cao và đáng tin cậy, đặc biệt là trong các bài toán phân loại ảnh có tính chất phức tạp như phân loại hoa, phân loại động vật, nhận diện khuôn mặt, và nhiều bài toán khác. CNN đã trở thành một công cụ quan trọng trong lĩnh vực xử lý ảnh và thường được sử dụng để giải quyết các bài toán phân loại ảnh trong các ứng dụng thực tế.

5 Conclusions - Kết luận

- Convolutional neural network (CNN) là một loại mạng nơ-ron nhân tạo rất phổ biến trong việc xử lý ảnh và nhận dạng hình ảnh. Nó được thiết kế để phát hiện các đặc trưng của ảnh thông qua các lớp tích chập và kết hợp chúng lại để tạo ra một dự đoán cuối cùng.
- CNN đã đạt được những thành tựu lớn trong lĩnh vực nhận dạng hình ảnh, bao gồm nhận dạng khuôn mặt, phân loại hình ảnh, nhận dạng vật thể và nhiều ứng dụng khác. Nó cũng được sử dụng trong nhiều lĩnh vực khác nhau bao gồm xử lý ngôn ngữ tự nhiên, nhận dạng giọng nói, xử lý tín hiệu và nhiều ứng dụng khác.
- Mặc dù CNN đã đạt được nhiều thành tựu đáng kể, nhưng vẫn còn nhiều thách thức cần giải quyết, bao gồm độ chính xác của mô hình, kích thước mô hình và thời gian huấn luyện. Tuy nhiên các pretrain model hiện nay thường được xây dựng dựa trên mạng CNN và các biến thể của nó.

6 References

- [1] <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148>
- [2] <https://nttuan8.com/bai-6-convolutional-neural-network/>
- [3] <https://viblo.asia/p/deep-learning-tim-hieu-ve-mang-tich-chap-cnn-maGK73bOKj2>
- [4] <https://aicurious.io/blog/2019-09-23-cac-ham-kich-hoat-activation-function-trong-neural-networks>
- [5] <https://machinelearningcoban.com/2017/03/04/overfitting/>

GitHub: <https://github.com/tooniesnguyen/>

- [6] <https://github.com/tooniesnguyen/AI-UTE/blob/main/Notebooks/MidtermProject/predict-future.ipynb>
- [7] <https://github.com/tooniesnguyen/AI-UTE/blob/main/Notebooks/MidtermProject/miniproject-flowers.ipynb>
- [8] <https://github.com/tooniesnguyen/AI-UTE/blob/main/Notebooks/MidtermProject/vn-dishes.ipynb>
- [9] <https://github.com/tooniesnguyen/AI-UTE/blob/main/Notebooks/MidtermProject/predict-money.ipynb>
- [10] <https://github.com/tooniesnguyen/AI-UTE/blob/main/Notebooks/MidtermProject/face-detect.ipynb>

TensorBoard: <https://huggingface.co/Toonies/>

- [11] <https://huggingface.co/Toonies/PredictFuture/tensorboard>
- [12] <https://huggingface.co/Toonies/Flowers/tensorboard>
- [13] <https://huggingface.co/Toonies/TenDishes/tensorboard>
- [14] <https://huggingface.co/Toonies/Money/tensorboard>
- [15] <https://huggingface.co/Toonies/FaceDetect/tensorboard>