Images Classification with CNNs

Lesson of Content

1. Build Model
2. Technique Training
3. Serving model

Overview GoogleNet and MobileNet

GoogleNet:

- Cho phép mạng xử lý thông tin từ nhiều kích thước bộ lọc tích chập cùng một lúc. Giúp mô hình nhận diện nhiều đặc trưng ở kích thước khác nhau.
- Ít tham số hơn và tốc độ tính toán nhanh hơn so với VGG và Resnet.
- Là mô hình khởi đầu cho ý tưởng có nhiều output, cũng là khởi nguồn ý tưởng của yolo sau này.

Nhược điểm:

- Phức tạp trong việc thiết kế và hiểu rõ cấu trúc mô hình.
- Mạng vẫn bị vanishing khi quá sâu (Cho đến khi Resnet ra đời mới giải quyết)
- Vẫn chưa đủ nhẹ để triển khai được mô hình trên cấp độ mobile

Mobilenet:

- Hiệu quả về tính toán, hiệu quả về dung lượng lưu trữ
- Phù hợp cho bài toán thời gian thực
- Tùy chỉnh linh hoạt được theo độ rộng (width multiplier)

Nhược điểm:

- Hiệu suất mô hình không cao bằng các mạng truyền thống

GoogleNet

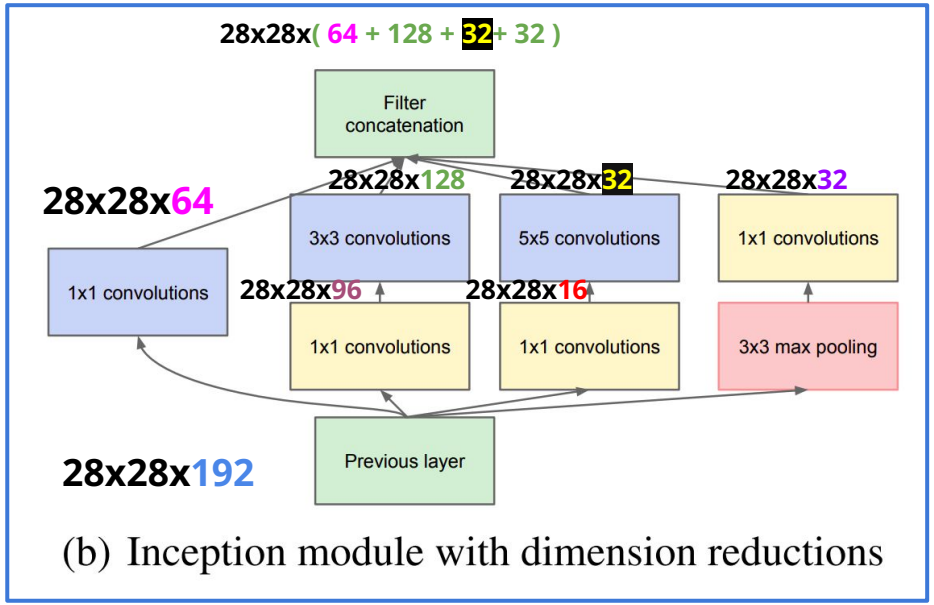
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture

Model: **GoogleNet**
Paper: https://arxiv.org/pdf/1409.4842.pdf
Years: **11.2014**

```
GoogLeNet(
  (conv1): BasicConv2d(…
  )
  (maxpool1): MaxPool2d(kernel_size
  (conv2): BasicConv2d(…
  )
  (conv3): BasicConv2d(…
  )
  (maxpool2): MaxPool2d(kernel_size
  (inception3a): Inception(…
  )
  (inception3b): Inception(…
  )
  (maxpool3): MaxPool2d(kernel_size
  (inception4a): Inception(…
  )
  (inception4b): Inception(…
  )
  (inception4c): Inception(…
  )
  (inception4d): Inception(…
  )
  (inception4e): Inception(…
  )
  (maxpool4): MaxPool2d(kernel_size
  (inception5a): Inception(…
  )
  (inception5b): Inception(…
  )
  (aux1): InceptionAux(…
  )
  (aux2): InceptionAux(…
  )
  (avgpool): AdaptiveAvgPool2d(outp
  (dropout): Dropout(p=0.2, inplace
  (fc): Linear(in_features=1024, ou
)
```

GoogleNet

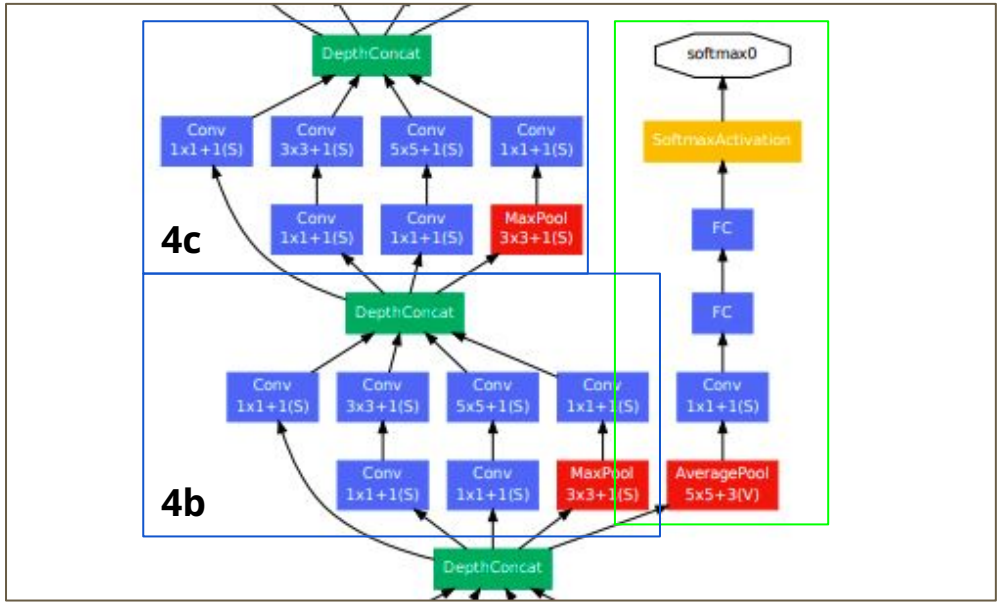


(b) Inception module with dimension reductions

```python
class InceptionBlock(nn.Module):
    def __init__(self, in_channels, ch1x1, ch3x3red, ch3x3, ch5x5red, ch5x5, pool_proj):
        super(InceptionBlock, self).__init__()
        self.branch1 = nn.Sequential(
            nn.Conv2d(in_channels, ch1x1, kernel_size=1, stride=1),
            nn.BatchNorm2d(ch1x1)
        )

        self.branch2 = nn.Sequential(
            nn.Conv2d(in_channels, ch3x3red, kernel_size=1, stride=1),
            nn.BatchNorm2d(ch3x3red),

            nn.Conv2d(ch3x3red, ch3x3, kernel_size=3, stride=1, padding = 1),
            nn.BatchNorm2d(ch3x3)
        )

        self.branch3 = nn.Sequential(
            nn.Conv2d(in_channels, ch5x5red, kernel_size=1),
            nn.BatchNorm2d(ch5x5red),

            nn.Conv2d(ch5x5red, ch5x5, kernel_size=5, padding=2),
            nn.BatchNorm2d(ch5x5),
        )

        self.branch4 = nn.Sequential(
            nn.MaxPool2d(kernel_size=3, stride=1, padding=1),
            nn.Conv2d(in_channels, pool_proj, kernel_size=1),
            nn.BatchNorm2d(pool_proj),
        )
    def forward(self, x):
        return torch.cat([self.branch1(x), self.branch2(x), self.branch3(x), self.branch4(x)], 1)
```

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj
max pool	3×3/2	28×28×192	0						
inception (3a)		28×28×256	2	64	96	128	16	32	32
inception (3b)		28×28×480	2	128	128	192	32	96	64

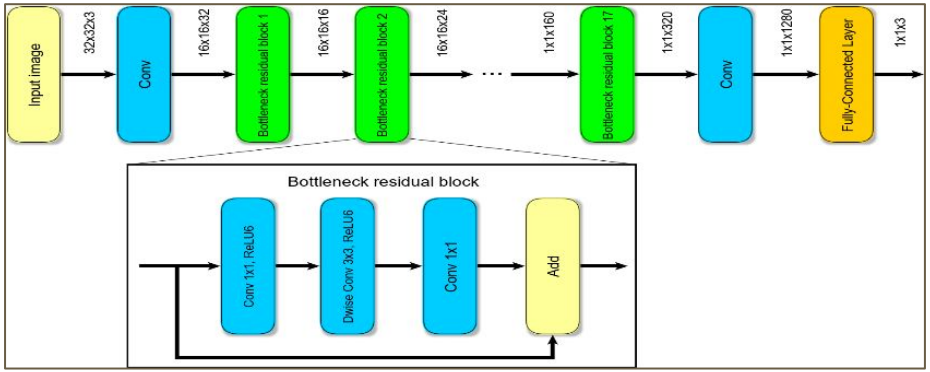
GoogleNet

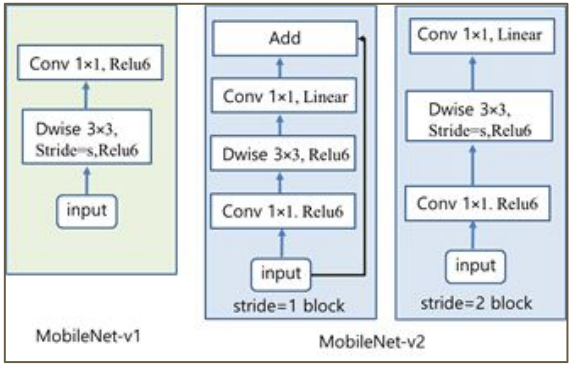



```python
class InceptionAux(nn.Module):
    def __init__(self, in_channels, num_classes):
        super(InceptionAux, self).__init__()
        self.avg = nn.AvgPool2d(kernel_size=5, stride=3)
        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, 128, kernel_size=1),
            nn.BatchNorm2d(128)
        )
        self.fc1 = nn.Linear(2048, 1024) # 2048 = 4x4x128
        self.fc2 = nn.Linear(1024, num_classes)
        self.dropout = nn.Dropout(0.7)

    def forward(self, x):
        out = self.avg(x)
        out = self.conv(out)
        out = out.view(out.size(0), -1) # Flatten this output
        out = self.fc1(out)
        out = self.fc2(out)
        out = self.dropout(out)
        return out
```

- An average pooling layer with 5×5 filter size and stride 3, resulting in an $4\times4\times512$ output for the (4a), and $4\times4\times528$ for the (4d) stage.
- A 1×1 convolution with 128 filters for dimension reduction and rectified linear activation.
- A fully connected layer with 1024 units and rectified linear activation.
- A dropout layer with 70% ratio of dropped outputs.
- A linear layer with softmax loss as the classifier (predicting the same 1000 classes as the main classifier, but removed at inference time).

MobileNet v2





Model: **MobileNet v2**
Paper: https://arxiv.org/pdf/1801.04381.pdf
Years: **03.2019**

```
MobileNetV2(
  (features): Sequential(
    (0): Conv2dNormActivation(…
    )
    (1): InvertedResidual(…
    )
    (2): InvertedResidual(…
    )
    (3): InvertedResidual(…
    )
    (4): InvertedResidual(…
    )
    (5): InvertedResidual(…
    )
    (6): InvertedResidual(…
    )
    (7): InvertedResidual(…
    )
    (8): InvertedResidual(…
    )
    (9): InvertedResidual(…
    )
    (10): InvertedResidual(…
    )
    (11): InvertedResidual(…
    )
    (12): InvertedResidual(…
    )
    (13): InvertedResidual(…
    )
    (14): InvertedResidual(…
    )
    (15): InvertedResidual(…
    )
    (16): InvertedResidual(…
    )
    (17): InvertedResidual(…
    )
    (18): Conv2dNormActivation(…
    )
  )
  (classifier): Sequential(
    (0): Dropout(p=0.2, inplace=False)
    (1): Linear(in_features=1280, out_features=1000, bias=True)
  )
)
```

MobileNet v2

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Table 2: MobileNetV2 : Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated n times. All layers in the same sequence have the same number c of output channels. The first layer of each sequence has a stride s and all others use stride 1. All spatial convolutions use 3×3 kernels. The expansion factor t is always applied to the input size as described in Table 1.

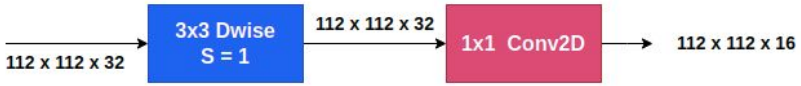
t: Expansion Factor, dùng để mở rộng kích thước mô hình.
c: Số channels output.
n: Số lần lặp lại của khối này.
s: stride.

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

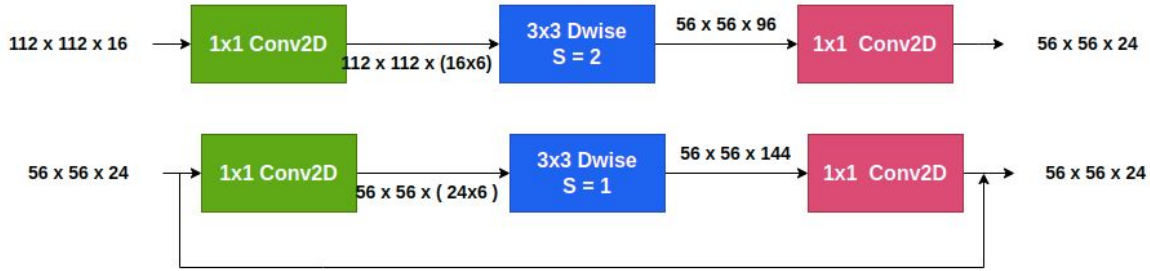
Table 1: *Bottleneck residual block* transforming from k to k' channels, with stride s, and expansion factor t.

Detail Bottleneck Operator

Input	Operator	t	c	n	s
$112^2 \times 32$	bottleneck	1	16	1	1



Input	Operator	t	c	n	s
$112^2 \times 16$	bottleneck	6	24	2	2

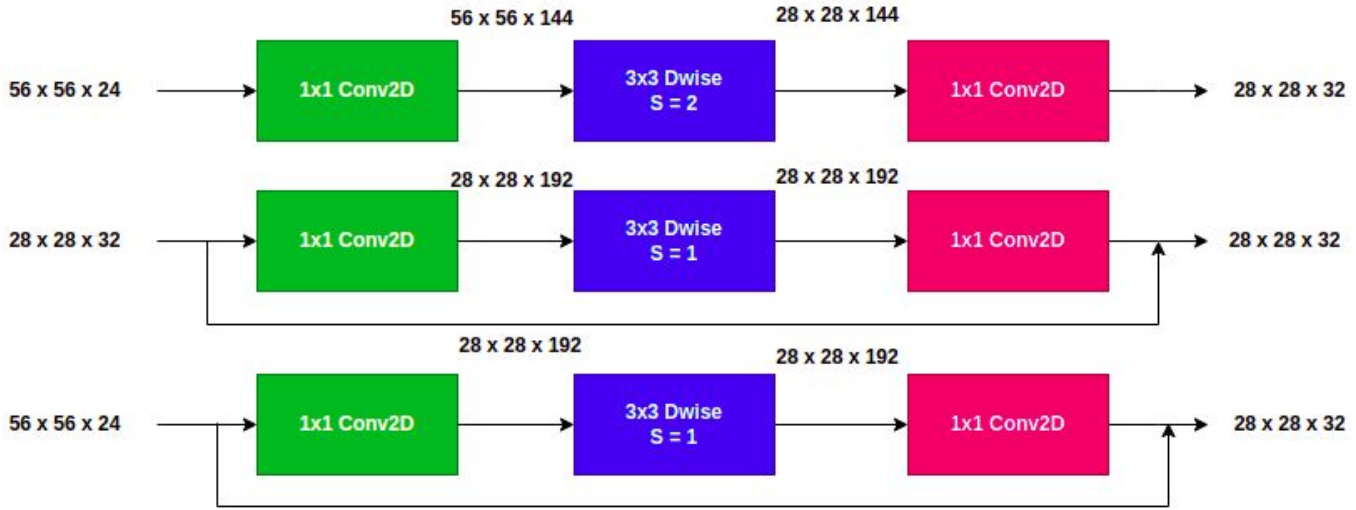


Để có skip-connection thì:

- Stride = 1
- Channels không đổi

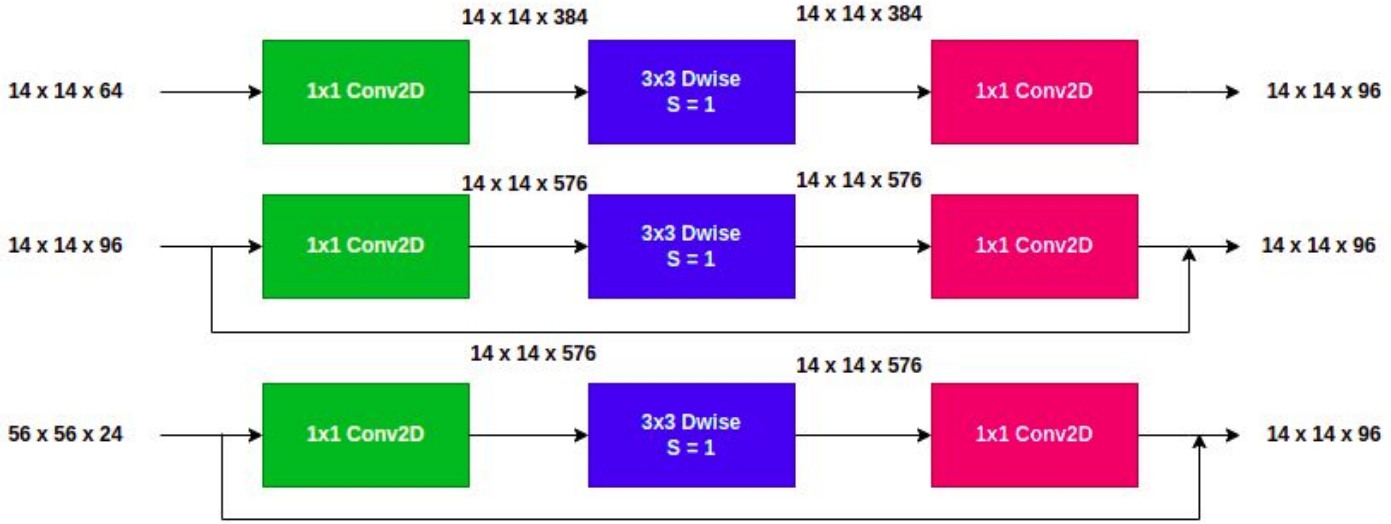
Detail Bottleneck Operator

Input	Operator	t	c	n	s
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$					

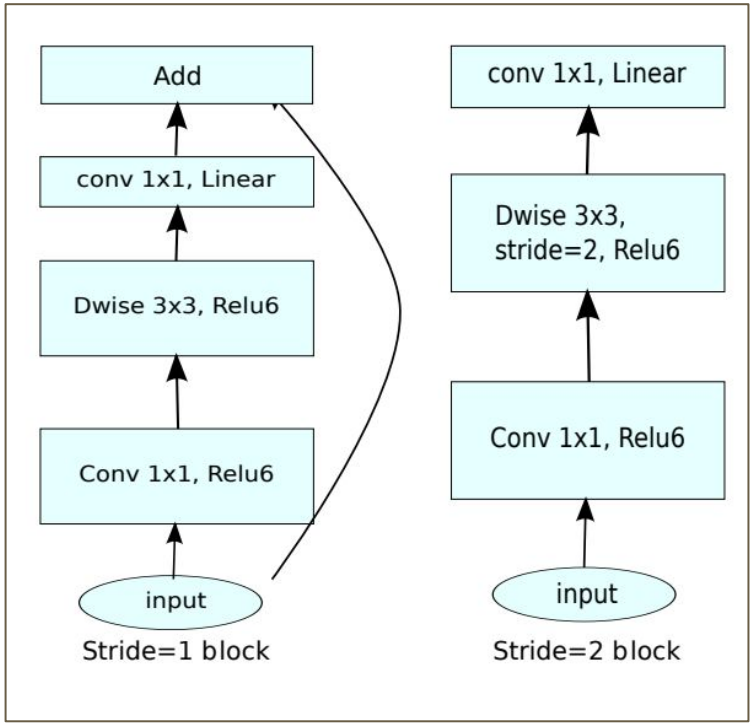


Detail Bottleneck Operator

Input	Operator	t	c	n	s
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2



Detail Bottleneck Operator




```python
class InvertedResidual(nn.Module):
    def __init__(self, in_channels, out_channels, stride, expand_ratio):
        super(InvertedResidual, self).__init__()
        self.stride = stride
        hidden_dim = round(in_channels * expand_ratio)

        # Stride = 1 và channels không đổi
        self.use_res_connect = self.stride == 1 and in_channels == out_channels

        layers = []

        # Khi expand_ratio khác 1, thêm tầng Conv2d và tầng BatchNorm để tăng số kênh
        if expand_ratio != 1:
            layers.append(nn.Conv2d(in_channels, hidden_dim, kernel_size=1, stride=1, padding=0, bias=False))
            layers.append(nn.BatchNorm2d(hidden_dim))
            layers.append(nn.ReLU6(inplace=True))

        # Thêm tầng Conv2d với kernel size 3 và stride có thể thay đổi (depthwise separable convolution)
        layers.extend([
            nn.Conv2d(hidden_dim, hidden_dim, kernel_size=3, stride=stride, padding=1, groups=hidden_dim, bias=False),
            nn.BatchNorm2d(hidden_dim),
            nn.ReLU6(inplace=True),
            # Tầng Conv2d cuối cùng để giảm số kênh trở lại
            nn.Conv2d(hidden_dim, out_channels, kernel_size=1, stride=1, padding=0, bias=False),
            nn.BatchNorm2d(out_channels)
        ])

        # Tạo một chuỗi các tầng đã xây dựng
        self.layers = nn.Sequential(*layers)

    def forward(self, x):
        # Nếu điều kiện sử dụng kết nối shortcut đúng, thực hiện kết nối shortcut
        if self.use_res_connect:
            return x + self.layers(x)
        else:
            # Nếu không, thực hiện các tầng theo chuỗi đã xây dựng
            return self.layers(x)
```

- Kiến trúc mạng:
- Mobilenet, GoogleNet

- Tiền xử lý dữ liệu:
- Normalization
- Data Augmentation

- Các hàm tối ưu:
- Adam, Momentum, Lion
- Scheduler

- Chống overfit:
- Dropout (Day 01)
- Cross-Validation
- Early stopping (Build using pytorch)

- Transfer Learning + Chiến lược Tuning (Bài sau)

Data Preprocessing

```python
# Chuẩn bị transform
from torchvision import transforms

train_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(p=0.1), # lật theo chiều ngang
    transforms.RandomVerticalFlip(p=0.1),   # lật theo chiều dọc
    transforms.RandomRotation(degrees=15),  # Xoay ảnh
    transforms.ColorJitter(
        brightness=0.2,
        contrast=0.2,
        saturation=0.2,
        hue=0.2), # Chỉnh về màu sắc
    transforms.GaussianBlur(kernel_size=3), # Làm mờ
    transforms.RandomResizedCrop(size=224, scale=(0.8, 1.0)), # Crop ngẫu nhiên
    transforms.RandomGrayscale(p=0.1), # Chuyển về ảnh xám
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
test_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```


Original


Blur


Horizontal flip


Gray

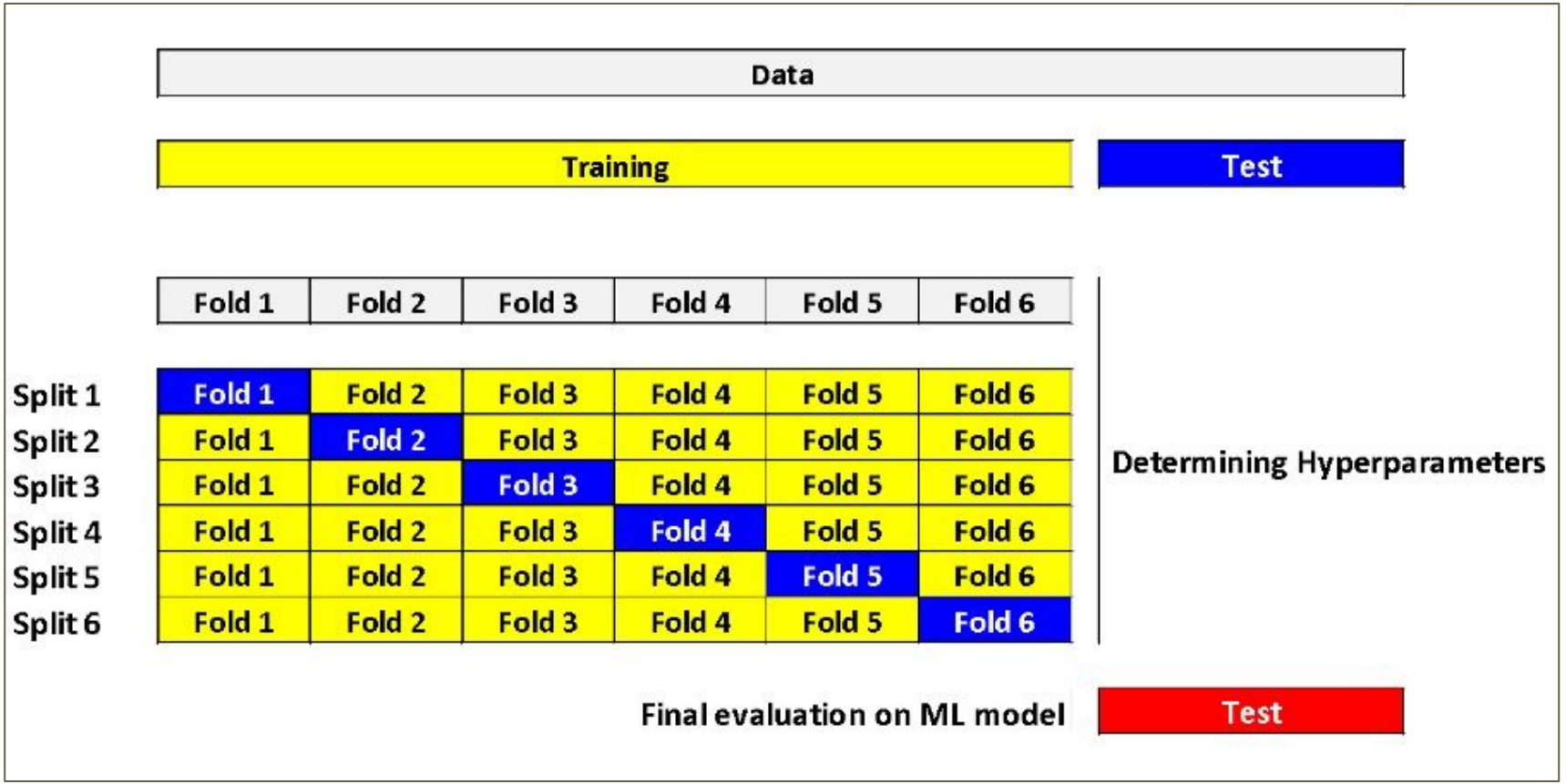

Rotate


Vertical flip

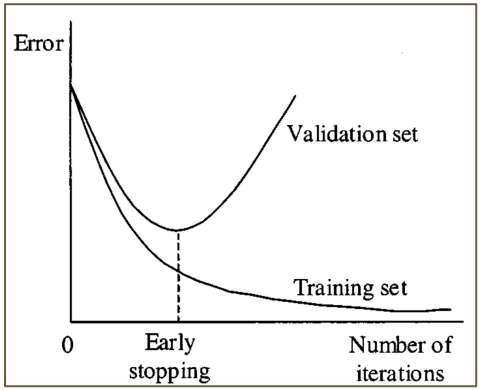
Lưu ý: Chỉ transform dữ liệu trên tập **train** để tăng cường, còn tập **test** chỉ cần chuẩn hóa dữ liệu.

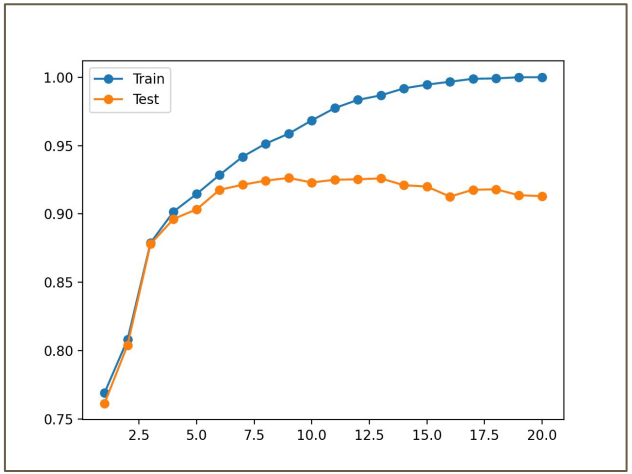
Links: https://pytorch.org/vision/main/transforms.html

Cross Validation



Early Stopping






```python
best_loss = float('inf')  # Khởi tạo giá trị best_loss bằng vô cùng lớn
early_stopping_patience = 5  # Số lần không cải thiện cho phép trước khi dừng
n_consecutive_worse = 0  # Số lần liên tiếp không cải thiện
max_epochs = 100  # Số lượng epochs tối đa

for epoch in range(max_epochs):
    # Huấn luyện mô hình trong một epoch và tính loss
    train_loss = train_one_epoch()

    # Đánh giá mô hình trên tập validation và tính loss
    val_loss = validate_model()

    # Kiểm tra xem val_loss có tốt hơn best_loss không
    if val_loss < best_loss:
        best_loss = val_loss
        n_consecutive_worse = 0  # Reset số lần không cải thiện
        save_model()  # Lưu mô hình hiện tại với val_loss tốt nhất

    else:
        n_consecutive_worse += 1

    # Kiểm tra xem đã đạt được điều kiện dừng chưa
    if n_consecutive_worse >= early_stopping_patience:
        print("Early stopping: No improvement in validation loss for {} epochs.".format(early_stopping_patience))
        break

print("Training finished.")
```

What is FastAPI ?

https://fastapi.tiangolo.com/

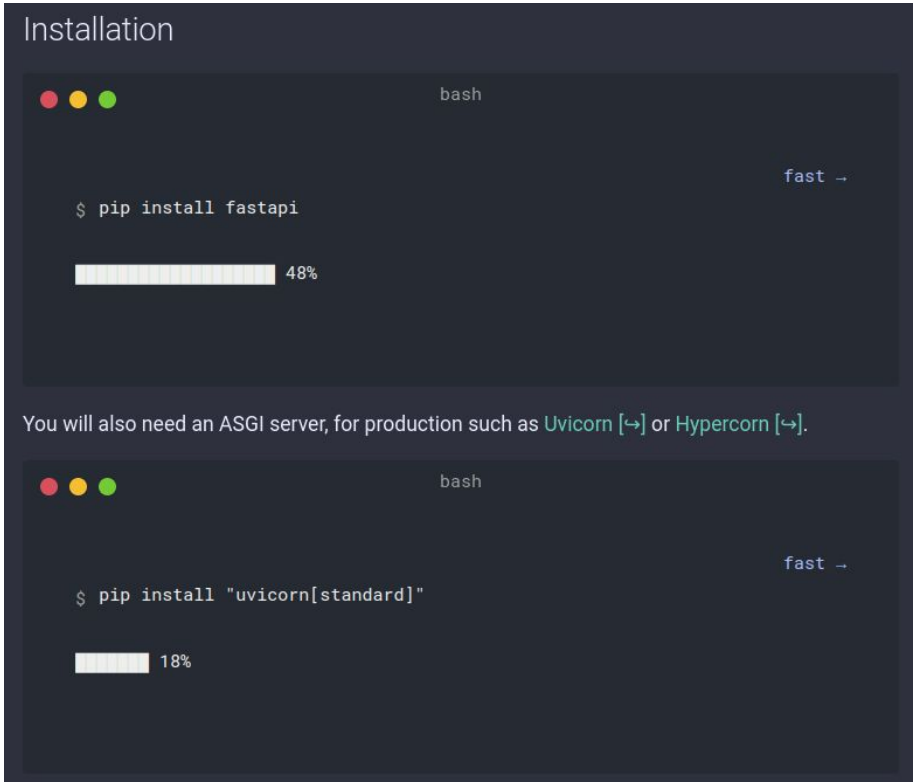
- Hỗ trợ được nhiều phương thức: **Post**, **Get**, Put, Delete.

- Quản lý được định dạng output và input của mô hình.

- Dễ tích hợp và debug mô hình.

FastAPI

High performance, easy to learn,
fast to code, ready for production



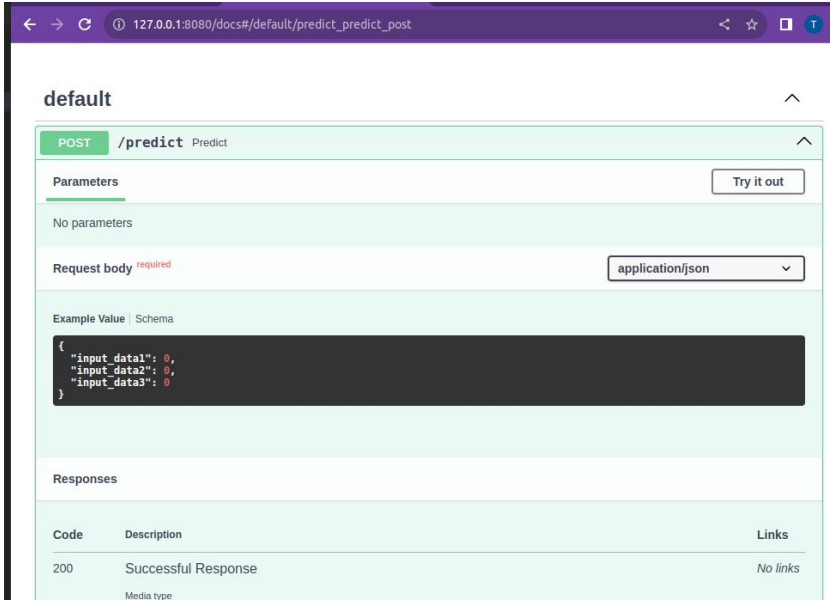
FastAPI in Basics

```python
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
def read_root():
    return {"Hello": "World"}


@app.get("/items/{item_id}")
def read_item(item_id: int, q: str = None):
    return {"item_id": item_id, "q": q}
```

```python
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class Data(BaseModel):
    text: str
    lang: str


@app.post("/text/")
def extract_entities(data: Data):
    return {"message": data.text}
```

FastAPI in ML/DL

```python
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

# Mô phỏng một mô hình Machine Learning đơn giản với nhiều biến đầu vào
def simple_ml_model(input_data):
    # Đây là một ví dụ đơn giản, mô hình này tính tổng các biến đầu vào
    return sum(input_data)

class PredictionRequest(BaseModel):
    input_data1: float
    input_data2: float
    input_data3: float
    # Thêm các biến đầu vào khác cần thiết

class PredictionResponse(BaseModel):
    prediction: float

@app.post("/predict", response_model=PredictionResponse)
async def predict(input_data: PredictionRequest):
    # Nhận dự đoán từ mô hình Machine Learning
    input_values = [input_data.input_data1, input_data.input_data2, input_data.input_data3]
    prediction = simple_ml_model(input_values)

    return {"prediction": prediction}
```



Run API:

```
uvicorn api:app --reload --port 8080
```

Link để test API

http://127.0.0.1:8080/docs

What is Streamlit ?

- Là một thư viện giúp triển khai nhanh một ứng dụng demo.

- Không cần phải biết frontend

- Tương thích với các thư viện của AI: Pytorch, sklearn, opencv, tensorflow, ...

Thường có hai cách dùng:

- Xem streamlit như một giao diện và backend sẽ gọi hoàn toàn qua fastapi.

- Chỉ sử dụng streamlit làm luôn phần frontend và backend.

https://streamlit.io/

How to use Streamlit

servering > app.py

```python
import streamlit as st
import requests

# Streamlit UI
st.title("Image Classification with FastAPI")

# Upload image
uploaded_image = st.file_uploader("Upload an image...", type=["jpg", "png", "jpeg"])

if uploaded_image is not None:
    # Display the uploaded image
    st.image(uploaded_image, caption="Uploaded Image", use_column_width=True)

    # Make a request to the FastAPI endpoint for classification
    url = "http://localhost:8000/classify/"  # Đảm bảo rằng FastAPI đang chạy ở cổng 8000
    files = {"file": uploaded_image}
    response = requests.post(url, files=files)

    if response.status_code == 200:
        result = response.json()
        class_label = result["class_label"]
        st.success(f"Predicted Class: {class_label}")
    else:
        st.error("Failed to classify the image. Please try again.")
```

```
(fpt) tari@Nitro-AN515-56:~/Documents/AI Tutor/CV/Topic 2/day03/servering$ streamlit run app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.133:8501
```

3. Serving model

How to use Streamlit

```python
import io
import streamlit as st
from PIL import Image
from utils import *

# Load the pre-trained MobileNetV2 model
device = 'cuda'
model = load_models(
    path = 'models/mobinet_v2_classification.pth',
    device = device
)
# Streamlit UI
st.title("Image Classification with Streamlit")

# Upload image
uploaded_image = st.file_uploader("Upload an image...", type=["jpg", "png", "jpeg"])

if uploaded_image is not None:
    # Display the uploaded image
    st.image(uploaded_image, caption="Uploaded Image", use_column_width=True)

    # Perform inference with the model
    image = Image.open(uploaded_image)
    input_tensor = preprocess_image(image, device)

    with torch.no_grad():
        output = model(input_tensor)

    # Get the predicted class label
    _, predicted_class = output.max(1)
    class_label = class_labels[predicted_class]

    st.success(f"Predicted Class: {class_label}")
```

```python
# Load the pre-trained MobileNetV2 model ( chỉ một lần)
@st.cache_resource()
def load_model():
    print("Here")
    model = load_models(
        path = 'models/mobinet_v2_classification.pth',
        device = device
    )
    return model
# Load model
model = load_model()
```