```
!pip install timm
```

```
Requirement already satisfied: timm in /usr/local/lib/python3.10/dist-packages (0.9.16)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from timm) (2.2.1+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (from timm) (0.17.1+cu121)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from timm) (6.0.1)
Requirement already satisfied: huggingface_hub in /usr/local/lib/python3.10/dist-packages (from timm) (0.20.3)
Requirement already satisfied: safetensors in /usr/local/lib/python3.10/dist-packages (from timm) (0.4.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from huggingface_hub->timm) (3.13.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface_hub->timm) (
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from huggingface_hub->timm) (2.31.0)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-packages (from huggingface_hub->timm) (4.66
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface_hu
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.10/dist-packages (from huggingface_hub->timm) (24
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->timm) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch->timm) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->timm) (3.1.3)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch->t
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch-
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch->t
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in /usr/local/lib/python3.10/dist-packages (from torch->timm)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in /usr/local/lib/python3.10/dist-packages (from torch->timm)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in /usr/local/lib/python3.10/dist-packages (from torch->timm)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in /usr/local/lib/python3.10/dist-packages (from torch->timm
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in /usr/local/lib/python3.10/dist-packages (from torch->t
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in /usr/local/lib/python3.10/dist-packages (from torch->t
Requirement already satisfied: nvidia-nccl-cu12==2.19.3 in /usr/local/lib/python3.10/dist-packages (from torch->timm) (2.
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch->timm) (
Requirement already satisfied: triton==2.2.0 in /usr/local/lib/python3.10/dist-packages (from torch->timm) (2.2.0)
Requirement already satisfied: nvidia-nvjitlink-cu12 in /usr/local/lib/python3.10/dist-packages (from nvidia-cusolver-cu1
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision->timm) (1.25.2)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision->timm)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch->timm) (2.1
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->huggin
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface_hub->t
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface_
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface_
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch->timm) (1.3.0)
```

```
#IMPORT ALL LIBRARIES
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import torchvision
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder
import timm

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import sys
from tqdm.notebook import tqdm


from torchvision.datasets import ImageFolder
from torch.utils.data import Dataset

class CelebDataset(Dataset):
    def __init__(self, data_dir, transform=None):
        self.data = ImageFolder(data_dir, transform=transform)

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

    def __getitem__(self, idx):
        return self.data[idx]

    @property
    def classes(self):
        return self.data.classes
```

```
import zipfile
import os

# Define paths
zip_file_path = 'Dataset.zip'
extraction_path = 'dataset'

# Create the extraction directory if it doesn't exist
if not os.path.exists(extraction_path):
    os.makedirs(extraction_path)

# Open the zip file for reading
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    # Extract all contents of the zip file
    zip_ref.extractall(extraction_path)

print(f"Datasets extracted to: {extraction_path}")
```

```
 Datasets extracted to: dataset
```

```
dataset = CelebDataset(
    data_dir='dataset/train'
)
```

```
data_dir = 'dataset/train'
target_to_class = {v: k for k, v in ImageFolder(data_dir).class_to_idx.items()}
print(target_to_class)
```

```
 {0: 'Arnold_Schwarzenegger', 1: 'George_W_Bush', 2: 'Junichiro_Koizumi', 3: 'Tony_Blair', 4: 'Vladimir_Putin'}
```

TRANSFORMING

```
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
])

data_dir = 'dataset/train'
dataset = CelebDataset(data_dir, transform)
```

```
for image, label in dataset:
    break
```

DATA LOADERS

```
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)
for images, labels in dataloader:
    break
images.shape, labels.shape
```

```
 (torch.Size([32, 3, 128, 128]), torch.Size([32]))
```

```
labels
```

```
 tensor([1, 1, 1, 2, 1, 4, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 2, 3, 1, 0, 1, 1, 3, 1,
         1, 1, 3, 2, 2, 4, 1, 3])
```

PYTORCH MODEL

```
class SimpleCelebClassifer(nn.Module):
    def __init__(self, num_classes=5):
        super(SimpleCelebClassifer, self).__init__()
        self.base_model = timm.create_model('efficientnet_b0', pretrained=True)
        self.features = nn.Sequential(*list(self.base_model.children())[:-1])

        enet_out_size = 1280
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(enet_out_size, num_classes)
        )
```

```
    def forward(self, x):
        x = self.features(x)
        output = self.classifier(x)
        return output


model = SimpleCelebClassifer(num_classes=5)
print(str(model)[:500])
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: UserWarni
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (http
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public mo
  warnings.warn(
model.safetensors:  100%                                                21.4M/21.4M  [00:00<00:00,  63.4MB/s]
SimpleCelebClassifer(
  (base_model): EfficientNet(
    (conv_stem): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bia
    (bn1): BatchNormAct2d(
      32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (blocks): Sequential(
      (0): Sequential(
        (0): DepthwiseSeparableConv(
          (conv_dw): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1
```

```
example_out = model(images)
example_out.shape
```

```
torch.Size([32, 5])
```

Starting Training Loop

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)


criterion(example_out, labels)
print(example_out.shape, labels.shape)
```

```
torch.Size([32, 5]) torch.Size([32])
```

Dataset

```
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
])

train_folder = 'dataset/train'
valid_folder = 'dataset/valid'
test_folder = 'dataset/test'

train_dataset = CelebDataset(train_folder, transform=transform)
val_dataset = CelebDataset(valid_folder, transform=transform)
test_dataset = CelebDataset(test_folder, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
```

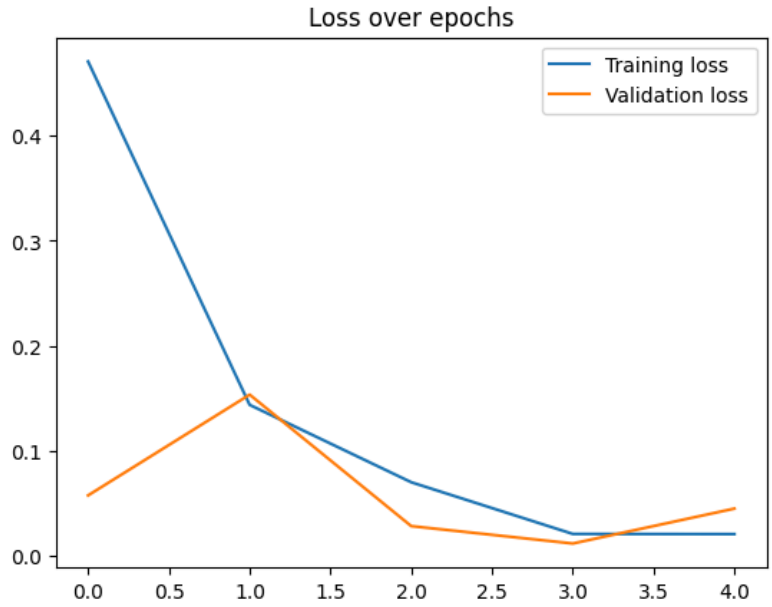
Training Loop

```
num_epochs = 5
train_losses, val_losses = [], []

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

model = SimpleCelebClassifer(num_classes=5)
model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

for epoch in range(num_epochs):
    # Training phase
    model.train()
    running_loss = 0.0
    for images, labels in tqdm(train_loader, desc='Training loop'):
        # Move inputs and labels to the device
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * labels.size(0)
    train_loss = running_loss / len(train_loader.dataset)
    train_losses.append(train_loss)

    # Validation phase
    model.eval()
    running_loss = 0.0
    with torch.no_grad():
        for images, labels in tqdm(val_loader, desc='Validation loop'):
            # Move inputs and labels to the device
            images, labels = images.to(device), labels.to(device)

            outputs = model(images)
            loss = criterion(outputs, labels)
            running_loss += loss.item() * labels.size(0)
    val_loss = running_loss / len(val_loader.dataset)
    val_losses.append(val_loss)
    print(f"Epoch {epoch+1}/{num_epochs} - Train loss: {train_loss}, Validation loss: {val_loss}")
```

```
Training loop: 100%                                                 26/26 [01:33<00:00,  3.15s/it]
Validation loop: 100%                                               26/26 [00:15<00:00,  1.83it/s]
Epoch 1/5 - Train loss: 0.4708608351873629, Validation loss: 0.057306162064725705
Training loop: 100%                                                 26/26 [01:34<00:00,  3.41s/it]
Validation loop: 100%                                               26/26 [00:15<00:00,  1.86it/s]
Epoch 2/5 - Train loss: 0.1435946926023021, Validation loss: 0.1533446230310382
Training loop: 100%                                                 26/26 [01:33<00:00,  3.27s/it]
Validation loop: 100%                                               26/26 [00:15<00:00,  1.66it/s]
Epoch 3/5 - Train loss: 0.06983851192575513, Validation loss: 0.02802771284668283
Training loop: 100%                                                 26/26 [01:35<00:00,  3.37s/it]
Validation loop: 100%                                               26/26 [00:15<00:00,  1.93it/s]
Epoch 4/5 - Train loss: 0.020573520282226983, Validation loss: 0.011430205445433262
Training loop: 100%                                                 26/26 [01:33<00:00,  3.37s/it]
Validation loop: 100%                                               26/26 [00:15<00:00,  1.91it/s]
Epoch 5/5 - Train loss: 0.02042191235743689, Validation loss: 0.044708251069215214
```

Plotting the Loss

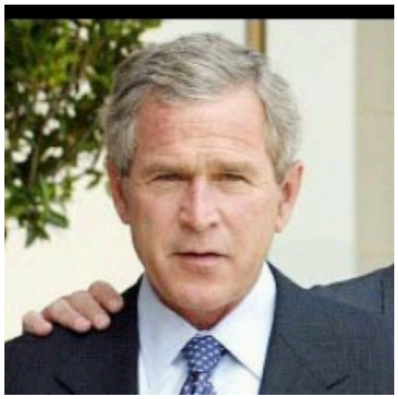
```
plt.plot(train_losses, label='Training loss')
plt.plot(val_losses, label='Validation loss')
plt.legend()
plt.title("Loss over epochs")
plt.show()
```

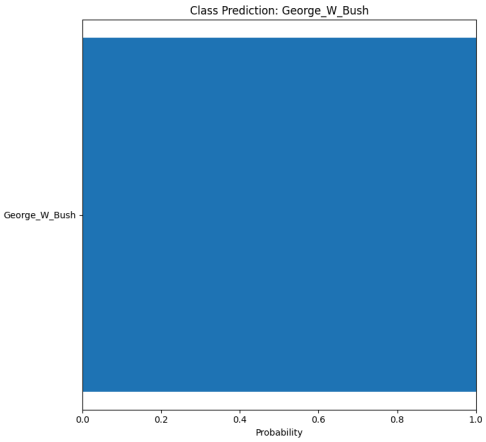


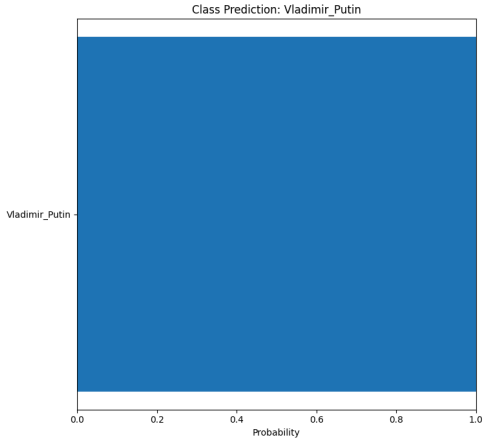
Evaluating the results

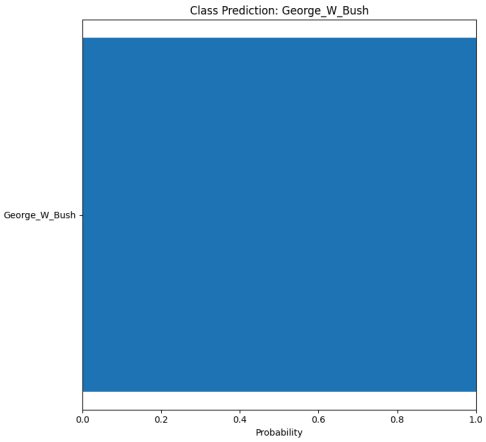
```
from PIL import Image

# Load and preprocess the image
def preprocess_image(image_path, transform):
    image = Image.open(image_path).convert("RGB")
    return image, transform(image).unsqueeze(0)

# Predict using the model
def predict(model, image_tensor, device):
    model.eval()
    with torch.no_grad():
        image_tensor = image_tensor.to(device)
        outputs = model(image_tensor)

        # Check if outputs is not None and has valid values
        if outputs is not None and torch.isfinite(outputs).all():
            probabilities, predicted_class = torch.max(outputs, 1)

            # Replace NaN values with zeros in probabilities
            probabilities[torch.isnan(probabilities)] = 0.0

            return probabilities.cpu().numpy().flatten(), predicted_class.item()
        else:
            # Return default values or handle the case appropriately
            return np.zeros(model.num_classes), -1  # Replace with appropriate values



# Visualization
def visualize_predictions(original_image, probabilities, predicted_class, class_names):
    fig, axarr = plt.subplots(1, 2, figsize=(14, 7))

    # Display image
    axarr[0].imshow(original_image)
    axarr[0].axis("off")

    # Display predictions
    if probabilities is not None and len(probabilities) > 0:
        class_name = class_names[predicted_class]
        axarr[1].barh([class_name], probabilities)
        axarr[1].set_xlabel("Probability")
        axarr[1].set_title("Class Prediction: {}".format(class_name))
        axarr[1].set_xlim(0, 1)
    else:
        axarr[1].set_xlabel("Probability")
        axarr[1].set_title("No valid predictions")
        axarr[1].set_xlim(0, 1)

    plt.tight_layout()
```

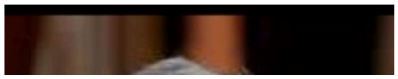
```
    plt.show()

from glob import glob
test_images = glob('dataset/train/*/*')
test_examples = np.random.choice(test_images, 10)

for example in test_examples:
    original_image, image_tensor = preprocess_image(example, transform)
    probabilities, predicted_class = predict(model, image_tensor, device)

    # Assuming dataset.classes gives the class names
    class_names = dataset.classes
    visualize_predictions(original_image, probabilities, predicted_class, class_names)
```

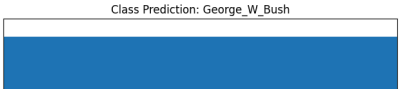



Class Prediction: George_W_Bush
George_W_Bush
0.0
0.2
0.4
0.6
0.8
1.0
Probability


Class Prediction: Vladimir_Putin
Vladimir_Putin
0.0
0.2
0.4
0.6
0.8
1.0
Probability


Class Prediction: George_W_Bush
George_W_Bush
0.0
0.2
0.4
0.6
0.8
1.0
Probability




Class Prediction: George_W_Bush