

```

1 حذف جميع البيانات المرتبطة لتسهيل عملية الربط من جديد#
2
3 %reset -sf
4
5

```

```

1 ربط كولا ب ب الريدف#
2 from google.colab import drive
3 drive.mount('/content/drive')

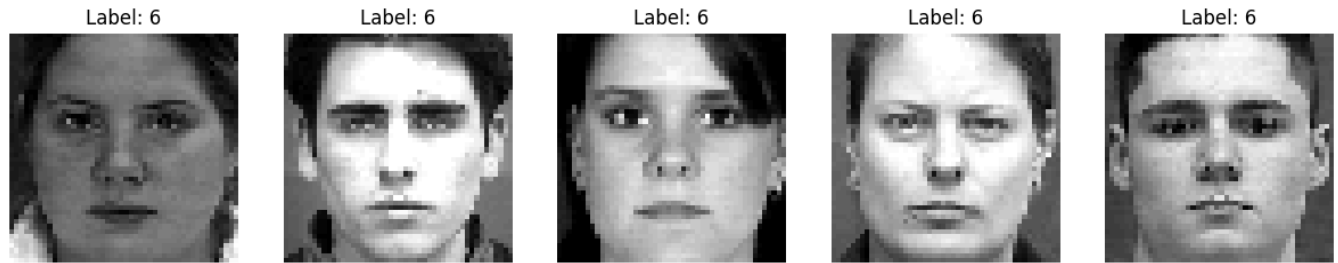
```

Mounted at /content/drive

```

1 استيراد الحزم الأساسية#
2 import pandas as pd
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 # تحميل ملفات CSV
7 csv_path = '/content/drive/MyDrive/Datasets/CK+/ckextended.csv'
8
9 # دالة تحسين جودة الصورة (تباين + تنعيم + حدة + تكبير)
10 df_ck = pd.read_csv(csv_path)
11
12 # دالة لعرض عينات الصور بعد تحسينها
13 def display_ck_images(df, num_samples=5):
14     plt.figure(figsize=(15,5))
15     for i in range(num_samples):
16         pixel_values = [int(p) for p in df.iloc[i]['pixels'].split()]
17         image = np.array(pixel_values).reshape(48, 48)
18         label = df.iloc[i]['emotion']
19 # رسم الصورة
20         plt.subplot(1, num_samples, i+1)
21         plt.imshow(image, cmap='gray')
22         plt.title(f'Label: {label}')
23         plt.axis('off')
24     plt.show()
25 # استدعاء الدالة لعرض أول 5 صور بعد تحسينها
26 display_ck_images(df_ck, num_samples=5)
27

```



```

1 # استدعاء الدالة لعرض أول 5 صور بعد تحسينها
2 import cv2
3
4 # تحويل CSV إلى صورة RGB و Grayscale
5 pixel_values = [int(p) for p in df_ck.iloc[0]['pixels'].split()]
6 image_48 = np.array(pixel_values, dtype='uint8').reshape(48, 48)
7
8 # تحويل إلى 3 قنوات لتصبح RGB
9 image_rgb = cv2.cvtColor(image_48, cv2.COLOR_GRAY2RGB)
10
11 # إعادة الحجم إلى 128x128
12 image_rgb_resized = cv2.resize(image_rgb, (128,128))
13 image_gray = cv2.cvtColor(image_rgb_resized, cv2.COLOR_RGB2GRAY)
14
15 # تجهيز الشكل لعرض صورتين جنباً إلى جنب
16
17 plt.subplot(1,2,1)
18 plt.imshow(image_rgb_resized)
19 plt.title("RGB Image Resized")
20 plt.axis("off")
21
22 plt.subplot(1,2,2)
23 plt.imshow(image_gray, cmap='gray')
24 plt.title("Grayscale Image")
25 plt.axis("off")
26 # عرض الشكل النهائي بالصور
27
28 plt.show()
29 # طباعة أبعاد الصورة الملونة بعد التصغير
30
31 print(f"Dimensions of RGB Image: {image_rgb_resized.shape}")
32

```



RGB Image Resized



Grayscale Image



Dimensions of RGB Image: (128, 128, 3)

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import cv2
5
6 # تحميل ملف CSV
7 csv_path = '/content/drive/MyDrive/Datasets/CK+/ckextended.csv'
8 df_ck = pd.read_csv(csv_path)
9
10 # تحسين الصورة (تباين وحدة) مع التأكد من القناة الواحدة والنوع
11 def enhance_image(image):
12     # وقناة واحدة uint8 تأكيد التحويل إلى
13     if len(image.shape) == 3:
14         image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
15     image = image.astype('uint8')
16
17     image_enhanced = cv2.equalizeHist(image)
18     gaussian = cv2.GaussianBlur(image_enhanced, (9,9), 10.0)
19     image_sharp = cv2.addWeighted(image_enhanced, 1.5, gaussian, -0.5, 0)
20     return image_sharp
21
22 # عرض الصور مع اظهار مربعات البكسل
23 def display_ck_images(df, num_samples=5):
24     plt.figure(figsize=(20,5))
25     for i in range(num_samples):
26         pixel_values = [int(p) for p in df.iloc[i]['pixels'].split()]
27         image = np.array(pixel_values).reshape(48, 48).astype('uint8')
28
29         # تحسين الصورة
30         image = enhance_image(image)
31

```

```

32     label = df.iloc[i]['emotion']
33
34     plt.subplot(1, num_samples, i+1)
35     plt.imshow(image, cmap='gray', interpolation='nearest') # البكسل واضح
36     plt.title(f'Label: {label}', fontsize=12)
37     plt.axis('off')
38
39     plt.tight_layout()
40     plt.show()
41
42 # استدعاء الدالة
43 display_ck_images(df_ck, num_samples=5)
44

```



```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import cv2
5
6 # تحميل البيانات
7 csv_path = '/content/drive/MyDrive/Datasets/CK+/ckextended.csv'
8 df_ck = pd.read_csv(csv_path)
9
10 # حدة + تكبير + تحسين الصورة: CLAHE
11 def enhance_image(image):
12     image = image.astype('uint8')
13
14     # تحسين التباين باستخدام CLAHE
15     clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
16     image_clahe = clahe.apply(image)
17
18     # تعزيز الحواف: Laplacian filter
19     laplacian = cv2.Laplacian(image_clahe, cv2.CV_8U)
20     enhanced = cv2.addWeighted(image_clahe, 1.5, laplacian, -0.5, 0)
21
22     # تكبير الصورة حتى تظهر التفاصيل
23     enhanced_resized = cv2.resize(enhanced, (192,192), interpolation=cv2.INTER
24

```

```

25     return enhanced_resized
26
27 # عرض الصور مع تفاصيل واضحة ومربعات بكسل كبيرة
28 def display_ck_images(df, num_samples=5):
29     plt.figure(figsize=(20,5))
30     for i in range(num_samples):
31         pixel_values = [int(p) for p in df.iloc[i]['pixels'].split()]
32         image = np.array(pixel_values).reshape(48, 48).astype('uint8')
33
34         # تحسين الصورة وتكبيرها
35         enhanced_image = enhance_image(image)
36         label = df.iloc[i]['emotion']
37
38         plt.subplot(1, num_samples, i+1)
39         plt.imshow(enhanced_image, cmap='gray', interpolation='nearest')
40         plt.title(f'Label: {label}', fontsize=12)
41         plt.axis('off')
42
43     plt.tight_layout()
44     plt.show()
45
46 # عرض الصور المحسنة
47 display_ck_images(df_ck, num_samples=5)
48

```



```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import cv2
5
6 # تحميل البيانات
7 csv_path = '/content/drive/MyDrive/Datasets/CK+/ckextended.csv'
8 df_ck = pd.read_csv(csv_path)
9
10 def enhance_image(image):
11     image = image.astype('uint8')
12
13     # تحسين التباين باستخدام CLAHE

```

```

14 clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
15 image_clahe = clahe.apply(image)
16
17 # تحسين الحدة باستخدام unsharp masking
18 gaussian = cv2.GaussianBlur(image_clahe, (0,0), sigmaX=1.0)
19 image_sharp = cv2.addWeighted(image_clahe, 1.5, gaussian, -0.5, 0)
20
21 return image_sharp
22
23 def display_ck_images(df, num_samples=5):
24     plt.figure(figsize=(20,5))
25     for i in range(num_samples):
26         pixel_values = [int(p) for p in df.iloc[i]['pixels'].split()]
27         image = np.array(pixel_values).reshape(48, 48).astype('uint8')
28
29         # تحسين الصورة
30         enhanced_image = enhance_image(image)
31         label = df.iloc[i]['emotion']
32
33         plt.subplot(1, num_samples, i+1)
34         plt.imshow(enhanced_image, cmap='gray', interpolation='bicubic') #
35         plt.title(f'Label: {label}', fontsize=12)
36         plt.axis('off')
37
38     plt.tight_layout()
39     plt.show()
40
41 # عرض الصور المحسنة بمظهر ناعم
42 display_ck_images(df_ck, num_samples=5)
43

```



```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import cv2
5
6 # تحميل البيانات
7 csv_path = '/content/drive/MyDrive/Datasets/CK+/ckextended.csv'

```

```
8 df_ck = pd.read_csv(csv_path)
9
10 def enhance_image(image):
11     image = image.astype('uint8')
12
13     # تحسين التباين باستخدام CLAHE
14     clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
15     image_clahe = clahe.apply(image)
16
17     # لتنعيم الضوضاء مع الحفاظ على الحواف Bilateral فلتير
18     bilateral = cv2.bilateralFilter(image_clahe, d=9, sigmaColor=75, sigmaSp
19
20     # حدة مع Unsharp Mask
21     gaussian = cv2.GaussianBlur(bilateral, (0,0), sigmaX=1.5)
22     image_sharp = cv2.addWeighted(bilateral, 1.7, gaussian, -0.7, 0)
23
24     # بوضوح عالي 256x256 تكبير الصورة إلى 256
25     image_resized = cv2.resize(image_sharp, (256,256), interpolation=cv2.INT
26
27     return image_resized
28
29 def display_ck_images(df, num_samples=5):
30     plt.figure(figsize=(25,5))
31     for i in range(num_samples):
32         pixel_values = [int(p) for p in df.iloc[i]['pixels'].split()]
33         image = np.array(pixel_values).reshape(48, 48).astype('uint8')
34
35         # تحسين الصورة وتكبيرها
36         enhanced_image = enhance_image(image)
37         label = df.iloc[i]['emotion']
38
39         plt.subplot(1, num_samples, i+1)
40         plt.imshow(enhanced_image, cmap='gray')
41         plt.title(f'Label: {label}', fontsize=14)
42         plt.axis('off')
43
44     plt.tight_layout()
45     plt.show()
46
47 # عرض الصور المحسنة بجودة فائقة
48 display_ck_images(df_ck, num_samples=5)
49
50
```



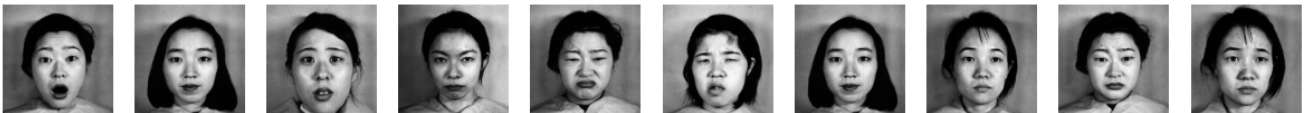
```

1 import os
2 from PIL import Image
3 import matplotlib.pyplot as plt
4 # تحديد مسار مجلد صور قاعدة بيانات JAFFE
5
6 jaffe_path = '/content/drive/MyDrive/Datasets/jaffe'
7 # صورة المرور على أول num_samples
8
9 def display_jaffe_samples(directory, num_samples=5):
10     images = [f for f in os.listdir(directory) if f.endswith('.tiff')]
11     plt.figure(figsize=(15,5))
12     for i, img_name in enumerate(images[:num_samples]):
13         img_path = os.path.join(directory, img_name)
14         img = Image.open(img_path)
15         plt.subplot(1, num_samples, i+1)
16         plt.imshow(img, cmap='gray')
17         plt.title(img_name)
18         plt.axis('off')
19     plt.show()
20 # استدعاء الدالة لعرض أول 10 صور من قاعدة بيانات JAFFE
21
22 display_jaffe_samples(jaffe_path, num_samples=10)
23

```



KL.SU2.165.tiff M.NE1.177.tiff M.FE2.111.tiff A.AN3.213.tiff KL.DI1.170.tiff KM.DI1.20.tiff M.NE3.179.tiff UY.NE1.134.tiff KL.FE2.175.tiff UY.FE1.152.tiff



```

1 import os
2 from PIL import Image
3 import torch
4 from torch.utils.data import Dataset, DataLoader
5 from torchvision import transforms
6
7 jaffe_path = '/content/drive/MyDrive/Datasets/jaffe'
8
9 # إنشاء قاموس لتحويل كل label إلى رقم Index
10 class JaffeDataset(Dataset):

```

```

10 class JaffeDataset(Dataset):
11     def __init__(self, root_dir, transform=None):
12         self.root_dir = root_dir
13         self.image_files = [f for f in os.listdir(root_dir) if f.endswith(
14             self.transform = transform
15             self.labels = [self.get_label(f) for f in self.image_files]
16             self.labe
17

```

```

1 import os
2 from PIL import Image
3 import torch
4 from torch.utils.data import Dataset, DataLoader
5 from torchvision import transforms
6
7 jaffe_path = '/content/drive/MyDrive/Datasets/jaffe'
8 #4. لتحميل الصور والتسميات JaffeDataset تعريف كلاس
9
10 class JaffeDataset(Dataset):
11     def __init__(self, root_dir, transform=None):
12         self.root_dir = root_dir
13         self.image_files = [f for f in os.listdir(root_dir) if f.endswith(
14             self.transform = transform
15             self.labels = [self.get_label(f) for f in self.image_files]
16             self.label_map = {label: idx for idx, label in enumerate(sorted(
17                 self.labels_idx = [self.label_map[label] for label in self.labels
18
19     def get_label(self, filename):
20         return filename.split('.')[1] # AN, HA, etc.
21
22     def __len__(self):
23         return len(self.image_files)
24
25     def __getitem__(self, idx):
26         img_path = os.path.join(self.root_dir, self.image_files[idx])
27         image = Image.open(img_path).convert('RGB')
28         if self.transform:
29             image = self.transform(image)
30         label = self.labels_idx[idx]
31         return image, label
32
33
34 #تطبيق التحويلات على الصور (Preprocessing)
35
36
37 transform = transforms.Compose([
38     transforms.Resize((128, 128)),

```

```

39     transforms.ToTensor()
40 ])
41 # تحميل البيانات داخل DataLoader
42
43 jaffe_dataset = JaffeDataset(jaffe_path, transform=transform)
44 jaffe_loader = DataLoader(jaffe_dataset, batch_size=8, shuffle=True)
45

```

```

1 import pandas as pd
2 import numpy as np
3
4 csv_path = '/content/drive/MyDrive/Datasets/CK+/ckextended.csv'
5 df_ck = pd.read_csv(csv_path)
6
7 class CKPlusDataset(Dataset):
8     def __init__(self, dataframe, transform=None):
9         self.df = dataframe
10        self.transform = transform
11
12    def __len__(self):
13        return len(self.df)
14
15    def __getitem__(self, idx):
16        pixel_values = [int(p) for p in self.df.iloc[idx]['pixels']].
17        image = np.array(pixel_values, dtype='uint8').reshape(48, 48)
18        image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
19        image = cv2.resize(image, (128,128))
20        image = Image.fromarray(image)
21        label = self.df.iloc[idx]['emotion']
22        if self.transform:
23            image = self.transform(image)
24        return image, label
25
26 ck_dataset = CKPlusDataset(df_ck, transform=transform)
27 ck_loader = DataLoader(ck_dataset, batch_size=8, shuffle=True)
28

```

```

1 from torchvision import models
2 import torch.nn as nn
3 # تحديد الجهاز المناسب للتدريب (GPU أو CPU):
4
5
6 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
7
8 num_classes_jaffe = len(set(jaffe_dataset.labels))

```

```

9 num_classes_ck = df_ck['emotion'].nunique()
10
11 # تعديل الطبقة النهائية لتناسب مع عدد الفئات في MobileNetV2 تحميل:
12
13 model = models.mobilenet_v2(pretrained=False)
14 model.classifier[1] = nn.Linear(model.last_channel, num_classes_jaffe) # 1
15 model = model.to(device)
16
17 # (Optimizer) والمُحسِّن (Loss Function) تعريف دالة الخسارة:
18
19 criterion = nn.CrossEntropyLoss()
20 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
21

```

```

→ /usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: 1
  warnings.warn(
  /usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: 1
  warnings.warn(msg)

```

```

1 # # (Dropout و BatchNorm) تفعيل وضع التدريب
2
3
4 epochs = 45
5 for epoch in range(epochs):
6     model.train()
7     running_loss, correct = 0.0, 0
8     for inputs, labels in jaffe_loader:
9         inputs, labels = inputs.to(device), labels.to(device)
10        optimizer.zero_grad()
11        outputs = model(inputs)
12        loss = criterion(outputs, labels)
13        loss.backward()
14        optimizer.step()
15        _, preds = torch.max(outputs, 1)
16        correct += torch.sum(preds == labels.data)
17        running_loss += loss.item() * inputs.size(0)
18        # حساب دقة وخسارة العنصر
19    epoch_acc = correct.double() / len(jaffe_dataset)
20    print(f'JAFfE Epoch {epoch+1}: Accuracy={epoch_acc:.4f}, Loss={running_l
21

```

```

→ JAFfE Epoch 1: Accuracy=0.0423, Loss=3.5438
JAFfE Epoch 2: Accuracy=0.0188, Loss=3.3483
JAFfE Epoch 3: Accuracy=0.0657, Loss=3.1323
JAFfE Epoch 4: Accuracy=0.1174, Loss=3.1331
JAFfE Epoch 5: Accuracy=0.1502, Loss=2.9160
JAFfE Epoch 6: Accuracy=0.1268, Loss=2.8051

```

```
JAFFE Epoch 7: Accuracy=0.1972, Loss=2.5492
JAFFE Epoch 8: Accuracy=0.2347, Loss=2.2958
JAFFE Epoch 9: Accuracy=0.2254, Loss=2.3355
JAFFE Epoch 10: Accuracy=0.1972, Loss=2.3865
JAFFE Epoch 11: Accuracy=0.2300, Loss=2.1692
JAFFE Epoch 12: Accuracy=0.3052, Loss=1.9462
JAFFE Epoch 13: Accuracy=0.3427, Loss=1.7142
JAFFE Epoch 14: Accuracy=0.3850, Loss=1.6863
JAFFE Epoch 15: Accuracy=0.3756, Loss=1.6615
JAFFE Epoch 16: Accuracy=0.3615, Loss=1.8404
JAFFE Epoch 17: Accuracy=0.3944, Loss=1.6018
JAFFE Epoch 18: Accuracy=0.4225, Loss=1.5044
JAFFE Epoch 19: Accuracy=0.4554, Loss=1.4628
JAFFE Epoch 20: Accuracy=0.3944, Loss=1.6035
JAFFE Epoch 21: Accuracy=0.4695, Loss=1.5398
JAFFE Epoch 22: Accuracy=0.5305, Loss=1.2598
JAFFE Epoch 23: Accuracy=0.5587, Loss=1.3144
JAFFE Epoch 24: Accuracy=0.5681, Loss=1.2057
JAFFE Epoch 25: Accuracy=0.5446, Loss=1.2230
JAFFE Epoch 26: Accuracy=0.6526, Loss=0.9523
JAFFE Epoch 27: Accuracy=0.6197, Loss=1.0407
JAFFE Epoch 28: Accuracy=0.6009, Loss=1.2214
JAFFE Epoch 29: Accuracy=0.6573, Loss=0.9386
JAFFE Epoch 30: Accuracy=0.7089, Loss=0.8329
JAFFE Epoch 31: Accuracy=0.7277, Loss=0.7701
JAFFE Epoch 32: Accuracy=0.7700, Loss=0.7737
JAFFE Epoch 33: Accuracy=0.7418, Loss=0.7128
JAFFE Epoch 34: Accuracy=0.7559, Loss=0.8233
JAFFE Epoch 35: Accuracy=0.6714, Loss=0.8986
JAFFE Epoch 36: Accuracy=0.6901, Loss=0.9655
JAFFE Epoch 37: Accuracy=0.6385, Loss=0.9148
JAFFE Epoch 38: Accuracy=0.6995, Loss=0.9137
JAFFE Epoch 39: Accuracy=0.6901, Loss=0.9961
JAFFE Epoch 40: Accuracy=0.7934, Loss=0.6329
JAFFE Epoch 41: Accuracy=0.7793, Loss=0.6761
JAFFE Epoch 42: Accuracy=0.8404, Loss=0.5692
JAFFE Epoch 43: Accuracy=0.8545, Loss=0.4898
JAFFE Epoch 44: Accuracy=0.8967, Loss=0.4159
JAFFE Epoch 45: Accuracy=0.9014, Loss=0.3309
```

```
1 model.classifier[1] = nn.Linear(model.last_channel, num_classes_ck)
2 model = model.to(device)
3
4 for epoch in range(190):
5     model.train()
6     running_loss, correct = 0.0, 0
7     for inputs, labels in ck_loader:
8         inputs, labels = inputs.to(device), labels.to(device)
9         optimizer.zero_grad()
10        outputs = model(inputs)
11        loss = criterion(outputs, labels)
12        loss.backward()
```

```
13     optimizer.step()
14     _, preds = torch.max(outputs, 1)
15     correct += torch.sum(preds == labels.data)
16     running_loss += loss.item() * inputs.size(0)
17
18     epoch_acc = correct.double() / len(ck_dataset)
19     print(f'CK+ Epoch {epoch+1}: Accuracy={epoch_acc:.4f}, Loss={running_lo
20
```



```

CK+ Epoch 178: Accuracy=0.9963, Loss=0.1012
CK+ Epoch 179: Accuracy=0.9935, Loss=0.0250
CK+ Epoch 180: Accuracy=0.9957, Loss=0.0192
CK+ Epoch 181: Accuracy=0.9989, Loss=0.0094
CK+ Epoch 182: Accuracy=0.9989, Loss=0.0117
CK+ Epoch 183: Accuracy=0.9989, Loss=0.0038
CK+ Epoch 184: Accuracy=1.0000, Loss=0.0029
CK+ Epoch 185: Accuracy=1.0000, Loss=0.0016
CK+ Epoch 186: Accuracy=1.0000, Loss=0.0014
CK+ Epoch 187: Accuracy=1.0000, Loss=0.0010
CK+ Epoch 188: Accuracy=1.0000, Loss=0.0009
CK+ Epoch 189: Accuracy=1.0000, Loss=0.0008
CK+ Epoch 190: Accuracy=1.0000, Loss=0.0022

```

```

1 from google.colab import drive
2 drive.mount('/content/drive')
3
4 import cv2
5 import numpy as np
6 import matplotlib.pyplot as plt
7 from PIL import Image
8

```

➡ Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour

```

1 import cv2
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 # عدّل حسب مجلدك مسار الصورة من Google Drive
6 image_path = '/content/drive/MyDrive/Datasets/jaffe/NM.NE3.94.tiff'
7
8 # تحميل الصورة
9 image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE) # بالياً رمادية jaffe الصور في
10
11 # التحقق من التحميل
12 if isinstance(image, np.ndarray):
13     # تغيير الحجم
14     resized_image = cv2.resize(image, (128, 128), interpolation=cv2.INTER_CU
15
16     # عرض الصورة الأصلية والمعدّلة
17     plt.figure(figsize=(10,5), dpi=150)
18
19     plt.subplot(1, 2, 1)
20     plt.imshow(image, cmap='gray')
21     plt.title("Original")
22     plt.axis('off')
23

```

```
24 plt.subplot(1, 2, 2)
25 plt.imshow(resized_image, cmap='gray')
26 plt.title("Resized")
27 plt.axis('off')
28
29 plt.show()
30 else:
31     print("❌ لم يتم تحميل الصورة. تأكد من المسار.")
32
```



Original

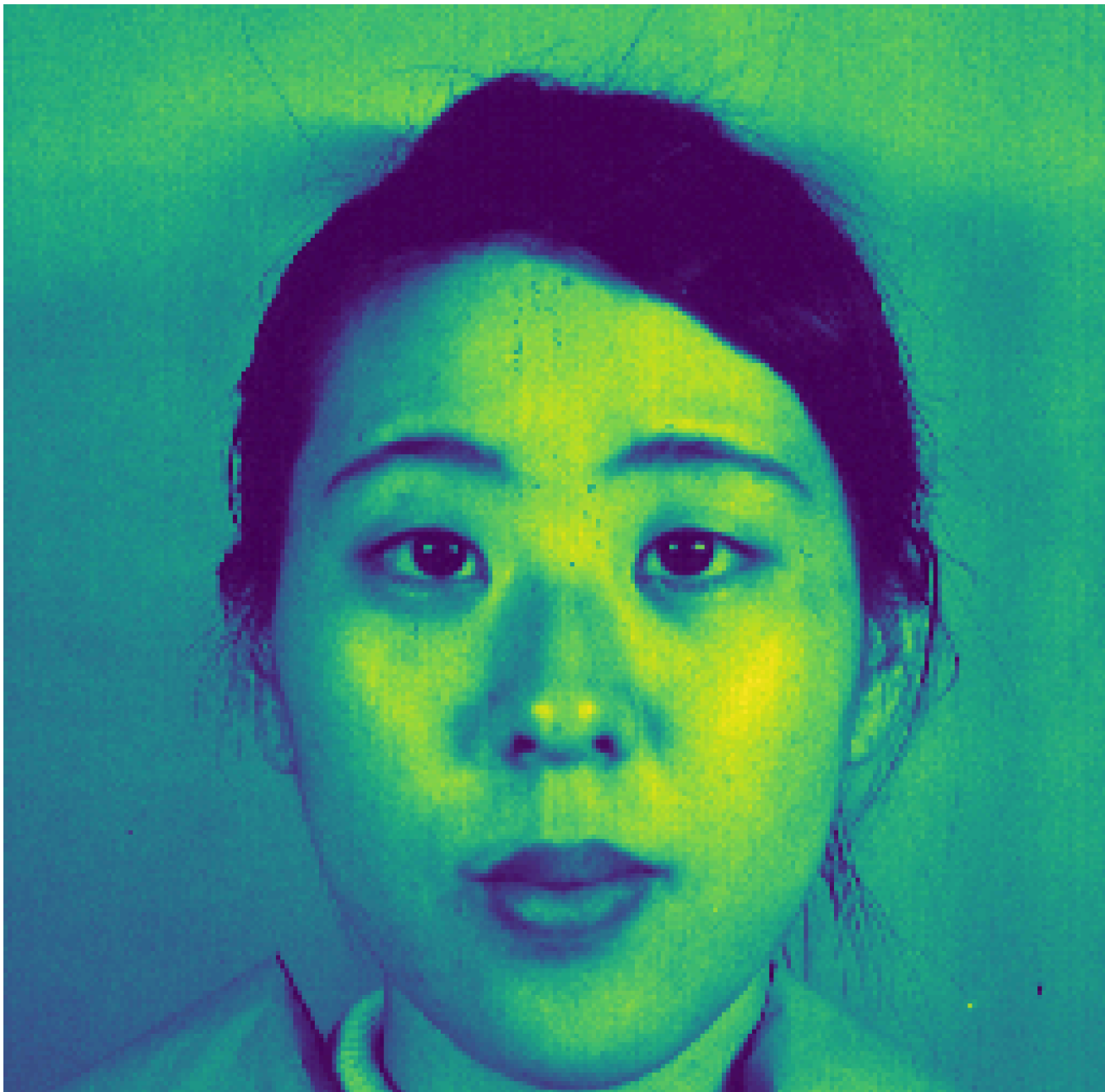


Resized



Double-click (or enter) to edit

```
1 plt.figure(figsize=(10,10), dpi=200) # dpi = dots per inch
2 plt.imshow(image)
3 plt.axis('off')
4 plt.show()
5
```



```
1 gray_image = cv2.imread('/content/drive/MyDrive/Datasets/jaffe/KA.AN1.39.tif')
2
3
4
5
6
7
8
9 csv_path = '/content/drive/MyDrive/Datasets/CK+/ckextended.csv'
10
```

```

11 #لربط البيانات مع التحويل المطلوب CKPlusDataset يتم إنشاء كائن من كلاس#.
12 class CKPlusDataset(Dataset):
13     def __init__(self, dataframe, transform=None):
14         self.df = dataframe
15         self.transform = transform
16
17     def __len__(self):
18         return len(self.df)
19
20     def __getitem__(self, idx):
21         pixel_values = [int(p) for p in self.df.iloc[idx]['pixels'].split()]
22         image = np.array(pixel_values, dtype='uint8').reshape(48, 48)
23         image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
24         image = cv2.resize(image, (128,128))
25         image = Image.fromarray(image)
26         label = self.df.iloc[idx]['emotion']
27         if self.transform:
28             image = self.transform(image)
29         return image, label
30
31 df_ck = pd.read_csv(csv_path)
32
33 transform = transforms.Compose([
34     transforms.ToTensor()
35 ])
36
37 ck_dataset = CKPlusDataset(df_ck, transform=transform)
38 ck_loader = DataLoader(ck_dataset, batch_size=8, shuffle=True)
39 #عدد الأصناف (المشاعر) في قاعدة البيانات
40 num_classes_ck = df_ck['emotion'].nunique()
41

```

```

1 jaffe_path = '/content/drive/MyDrive/Datasets/jaffe'
2
3 class JaffeDataset(Dataset):
4     def __init__(self, root_dir, transform=None):
5         self.root_dir = root_dir
6         self.image_files = [f for f in os.listdir(root_dir) if f.endswith('.')]
7         self.transform = transform
8         self.labels = [self.get_label(f) for f in self.image_files]
9         self.label_map = {label: idx for idx, label in enumerate(sorted(set(
10             self.labels_idx = [self.label_map[label] for label in self.labels]
11
12     def get_label(self, filename):
13         return filename.split('.')[1] # AN, HA, etc.
14

```

```
15     def __len__(self):
16         return len(self.image_files)
17
18     def __getitem__(self, idx):
19         img_path = os.path.join(self.root_dir, self.image_files[idx])
20         image = Image.open(img_path).convert('RGB')
21         if self.transform:
22             image = self.transform(image)
23         label = self.labels_idx[idx]
24         return image, label
25
26 jaffe_dataset = JaffeDataset(jaffe_path, transform=transform)
27 jaffe_loader = DataLoader(jaffe_dataset, batch_size=8, shuffle=True)
28 num_classes_jaffe = len(set(jaffe_dataset.labels))
29
```

```
1 import torch
2 import torch.nn as nn
3
4 class SimpleCNN(nn.Module):
5     def __init__(self, num_classes):
6         super(SimpleCNN, self).__init__()
7         self.model = nn.Sequential(
8             nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
9             nn.ReLU(),
10            nn.MaxPool2d(2),
11
12            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
13            nn.ReLU(),
14            nn.MaxPool2d(2),
15
16            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
17            nn.ReLU(),
18            nn.MaxPool2d(2),
19
20            nn.Flatten(),
21            nn.Linear(128 * 16 * 16, 128),
22            nn.ReLU(),
23            nn.Dropout(0.5),
24            nn.Linear(128, num_classes)
25        )
26
27     def forward(self, x):
28         return self.model(x)
29
```

```

1 from torchvision import models
2
3 def get_mobilenetv2(num_classes):
4     model = models.mobilenet_v2(pretrained=False)
5     model.classifier[1] = nn.Linear(model.last_channel, num_classes)
6     return model
7

1 # استيراد المكتبات اللازمة
2 import torch
3 import torch.nn as nn
4 from torchvision import models
5 import matplotlib.pyplot as plt
6
7 # (إن وُجد GPU) تحديد الجهاز
8 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
9
10 # (JAFPE مثلاً 7 مشاعر في) وتعديله لعدد الأصناف MobileNetV2 تحميل نموذج
11 def get_mobilenetv2(num_classes):
12     model = models.mobilenet_v2(weights=None) # بدون أوزان مُسبقة
13     model.classifier[1] = nn.Linear(model.last_channel, num_classes) # الأخيرة
14     return model
15
16 # إعداد النموذج والوظائف المساعدة
17 model = get_mobilenetv2(num_classes_jaffe)
18 model = model.to(device)
19 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
20 criterion = nn.CrossEntropyLoss()
21
22 # عدد الدورات التدريبية
23 epochs = 45
24 train_losses, train_accs = [], []
25
26 # التدريب
27 for epoch in range(epochs):
28     model.train() # وضع النموذج في طور التدريب
29     running_loss = 0.0
30     correct = 0
31     total = 0
32
33     # حلقة على دفعات البيانات
34     for inputs, labels in jaffe_loader:
35         inputs, labels = inputs.to(device), labels.to(device)
36
37         optimizer.zero_grad()

```

```

38     outputs = model(inputs)
39     loss = criterion(outputs, labels)
40     loss.backward()
41     optimizer.step()
42
43     running_loss += loss.item()
44
45     _, preds = torch.max(outputs, 1)
46     correct += (preds == labels).sum().item()
47     total += labels.size(0)
48
49     # حساب متوسط الخسارة والدقة لكل epoch
50     epoch_loss = running_loss / len(jaffe_loader)
51     epoch_acc = correct / total
52
53     # ✅ تسجيل القيم في القوائم
54     train_losses.append(epoch_loss)
55     train_accs.append(epoch_acc)
56
57     print(f"Epoch {epoch+1}/{epochs}: Loss={epoch_loss:.4f}, Accuracy={epoch.
58

```



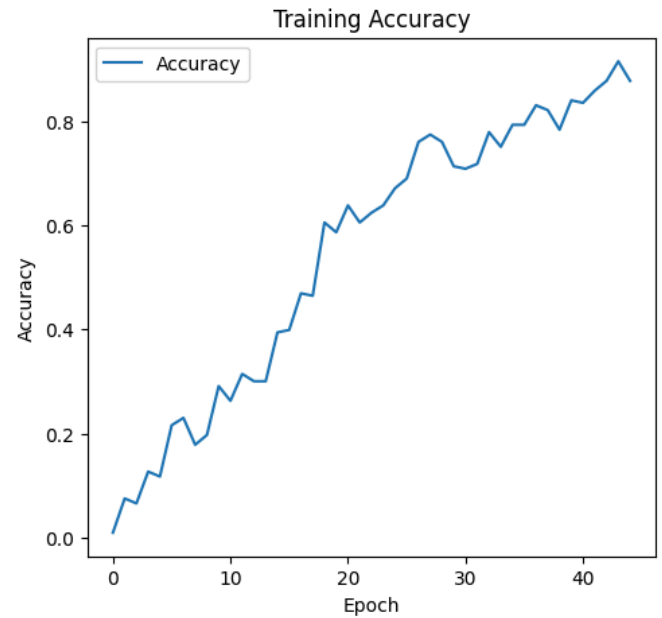
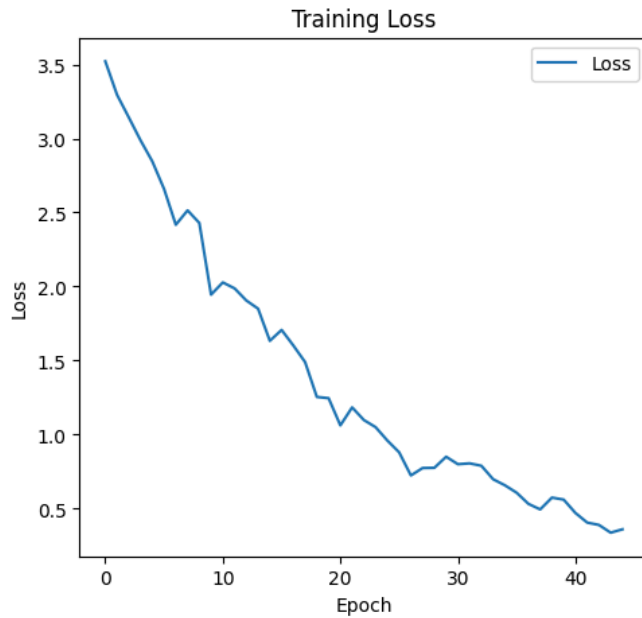
```

Epoch 1/45: Loss=3.5208, Accuracy=0.0094
Epoch 2/45: Loss=3.2943, Accuracy=0.0751
Epoch 3/45: Loss=3.1417, Accuracy=0.0657
Epoch 4/45: Loss=2.9868, Accuracy=0.1268
Epoch 5/45: Loss=2.8451, Accuracy=0.1174
Epoch 6/45: Loss=2.6588, Accuracy=0.2160
Epoch 7/45: Loss=2.4172, Accuracy=0.2300
Epoch 8/45: Loss=2.5144, Accuracy=0.1784
Epoch 9/45: Loss=2.4295, Accuracy=0.1972
Epoch 10/45: Loss=1.9455, Accuracy=0.2911
Epoch 11/45: Loss=2.0280, Accuracy=0.2629
Epoch 12/45: Loss=1.9875, Accuracy=0.3146
Epoch 13/45: Loss=1.9059, Accuracy=0.3005
Epoch 14/45: Loss=1.8506, Accuracy=0.3005
Epoch 15/45: Loss=1.6338, Accuracy=0.3944
Epoch 16/45: Loss=1.7074, Accuracy=0.3991
Epoch 17/45: Loss=1.6022, Accuracy=0.4695
Epoch 18/45: Loss=1.4901, Accuracy=0.4648
Epoch 19/45: Loss=1.2549, Accuracy=0.6056
Epoch 20/45: Loss=1.2466, Accuracy=0.5869
Epoch 21/45: Loss=1.0631, Accuracy=0.6385
Epoch 22/45: Loss=1.1844, Accuracy=0.6056
Epoch 23/45: Loss=1.0991, Accuracy=0.6244
Epoch 24/45: Loss=1.0509, Accuracy=0.6385
Epoch 25/45: Loss=0.9606, Accuracy=0.6714
Epoch 26/45: Loss=0.8814, Accuracy=0.6901
Epoch 27/45: Loss=0.7251, Accuracy=0.7606
Epoch 28/45: Loss=0.7751, Accuracy=0.7746
Epoch 29/45: Loss=0.7766, Accuracy=0.7606
Epoch 30/45: Loss=0.8513, Accuracy=0.7136

```

```
Epoch 31/45: Loss=0.8008, Accuracy=0.7089
Epoch 32/45: Loss=0.8067, Accuracy=0.7183
Epoch 33/45: Loss=0.7902, Accuracy=0.7793
Epoch 34/45: Loss=0.6993, Accuracy=0.7512
Epoch 35/45: Loss=0.6585, Accuracy=0.7934
Epoch 36/45: Loss=0.6083, Accuracy=0.7934
Epoch 37/45: Loss=0.5334, Accuracy=0.8310
Epoch 38/45: Loss=0.4956, Accuracy=0.8216
Epoch 39/45: Loss=0.5754, Accuracy=0.7840
Epoch 40/45: Loss=0.5619, Accuracy=0.8404
Epoch 41/45: Loss=0.4718, Accuracy=0.8357
Epoch 42/45: Loss=0.4069, Accuracy=0.8592
Epoch 43/45: Loss=0.3909, Accuracy=0.8779
Epoch 44/45: Loss=0.3387, Accuracy=0.9155
Epoch 45/45: Loss=0.3613, Accuracy=0.8779
```

```
1 import matplotlib.pyplot as plt
2
3 plt.figure(figsize=(12,5))
4
5 plt.subplot(1,2,1)
6 plt.plot(train_losses, label='Loss')
7 plt.title('Training Loss')
8 plt.xlabel('Epoch')
9 plt.ylabel('Loss')
10 plt.legend()
11
12 plt.subplot(1,2,2)
13 plt.plot(train_accs, label='Accuracy')
14 plt.title('Training Accuracy')
15 plt.xlabel('Epoch')
16 plt.ylabel('Accuracy')
17 plt.legend()
18
19 plt.show()
20
```



```

1 from sklearn.metrics import confusion_matrix, classification_report
2 import seaborn as sns
3
4 y_true = []
5 y_pred = []
6
7 model.eval()
8 with torch.no_grad():
9     for inputs, labels in jaffe_loader:
10         inputs = inputs.to(device)
11         outputs = model(inputs)
12         _, preds = torch.max(outputs, 1)
13         y_true.extend(labels.numpy())
14         y_pred.extend(preds.cpu().numpy())
15
16 cm = confusion_matrix(y_true, y_pred)
17 print("Classification Report:\n", classification_report(y_true, y_pred))
18
19 plt.figure(figsize=(6,5))
20 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
21 plt.title('Confusion Matrix')
22 plt.xlabel('Predicted')
23 plt.ylabel('True')
24 plt.show()
25

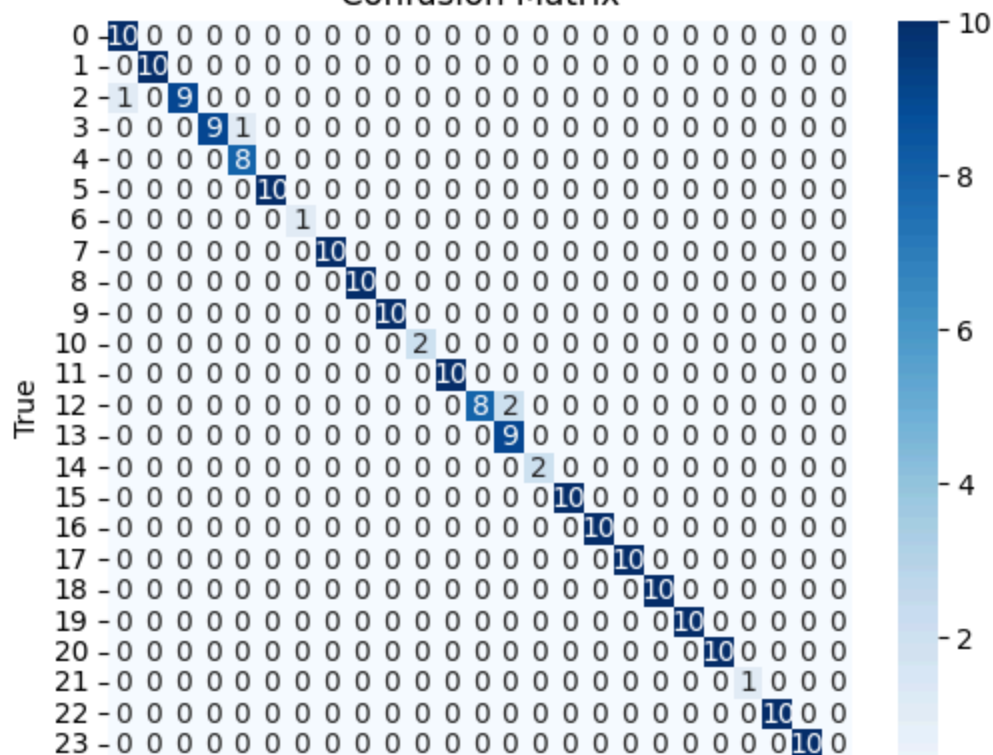
```

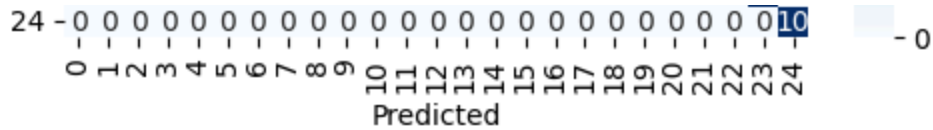


Classification Report:

	precision	recall	f1-score	support
0	0.91	1.00	0.95	10
1	1.00	1.00	1.00	10
2	1.00	0.90	0.95	10
3	1.00	0.90	0.95	10
4	0.89	1.00	0.94	8
5	1.00	1.00	1.00	10
6	1.00	1.00	1.00	1
7	1.00	1.00	1.00	10
8	1.00	1.00	1.00	10
9	1.00	1.00	1.00	10
10	1.00	1.00	1.00	2
11	1.00	1.00	1.00	10
12	1.00	0.80	0.89	10
13	0.82	1.00	0.90	9
14	1.00	1.00	1.00	2
15	1.00	1.00	1.00	10
16	1.00	1.00	1.00	10
17	1.00	1.00	1.00	10
18	1.00	1.00	1.00	10
19	1.00	1.00	1.00	10
20	1.00	1.00	1.00	10
21	1.00	1.00	1.00	1
22	1.00	1.00	1.00	10
23	1.00	1.00	1.00	10
24	1.00	1.00	1.00	10
accuracy			0.98	213
macro avg	0.98	0.98	0.98	213
weighted avg	0.98	0.98	0.98	213

Confusion Matrix





```

1 import os
2 import numpy as np
3 import torch
4 from torch.utils.data import Dataset, DataLoader
5 from torchvision import datasets, transforms as T, models
6 from torch import nn, optim
7 import warnings
8 warnings.filterwarnings("ignore")
9 from tqdm import tqdm
10 from sklearn.metrics import confusion_matrix, accuracy_score, precision_score
11 import matplotlib.pyplot as plt
12 import seaborn as sns
13 from PIL import Image
14 import cv2
15 import pandas as pd
16

```

```

1 jaffe_path = '/content/drive/MyDrive/Datasets/jaffe'
2
3 class JaffeDataset(Dataset):
4     def __init__(self, root_dir, transform=None):
5         self.root_dir = root_dir
6         self.image_files = [f for f in os.listdir(root_dir) if f.endswith('.')]
7         self.transform = transform
8         self.labels = [self.get_label(f) for f in self.image_files]
9         self.label_map = {label: idx for idx, label in enumerate(sorted(set(
10             self.labels_idx = [self.label_map[label] for label in self.labels]
11
12     def get_label(self, filename):
13         return filename.split('.')[1] # AN, HA, etc.
14
15     def __len__(self):
16         return len(self.image_files)
17
18     def __getitem__(self, idx):
19         img_path = os.path.join(self.root_dir, self.image_files[idx])
20         image = Image.open(img_path).convert('RGB')
21         if self.transform:
22             image = self.transform(image)
23         label = self.labels_idx[idx]
24         return image, label

```

```

25
26 transform = T.Compose([
27     T.RandomHorizontalFlip(),
28     T.RandomVerticalFlip(),
29     T.RandomRotation(10),
30     T.Resize((128,128)),
31     T.ToTensor()
32 ])
33
34 jaffe_dataset = JaffeDataset(jaffe_path, transform=transform)
35 jaffe_loader = DataLoader(jaffe_dataset, batch_size=8, shuffle=True)
36 num_classes_jaffe = len(set(jaffe_dataset.labels))
37

```

```

1 jaffe_path = '/content/drive/MyDrive/Datasets/jaffe'
2
3 class JaffeDataset(Dataset):
4     def __init__(self, root_dir, transform=None):
5         self.root_dir = root_dir
6         self.image_files = [f for f in os.listdir(root_dir) if f.endswith(
7             self.transform = transform
8             self.labels = [self.get_label(f) for f in self.image_files]
9             self.label_map = {label: idx for idx, label in enumerate(sorted(
10                 self.labels_idx = [self.label_map[label] for label in self.labels
11
12     def get_label(self, filename):
13         return filename.split('.')[1] # AN, HA, etc.
14
15     def __len__(self):
16         return len(self.image_files)
17
18     def __getitem__(self, idx):
19         img_path = os.path.join(self.root_dir, self.image_files[idx])
20         image = Image.open(img_path).convert('RGB')
21         if self.transform:
22             image = self.transform(image)
23         label = self.labels_idx[idx]
24         return image, label
25
26 transform = T.Compose([
27     T.RandomHorizontalFlip(),
28     T.RandomVerticalFlip(),
29     T.RandomRotation(10),
30     T.Resize((128,128)),
31     T.ToTensor()
32 ])

```

```
33
34 jaffe_dataset = JaffeDataset(jaffe_path, transform=transform)
35 jaffe_loader = DataLoader(jaffe_dataset, batch_size=8, shuffle=True)
36 num_classes_jaffe = len(set(jaffe_dataset.labels))
37

1 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
2
3 model = models.resnet18(pretrained=False)
4 model.fc = nn.Linear(model.fc.in_features, num_classes_jaffe)
5 model = model.to(device)
6
7 criterion = nn.CrossEntropyLoss()
8 optimizer = optim.AdamW(model.parameters(), lr=0.001)
9 scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)
10

1 def train_model(model, criterion, optimizer, scheduler, num_epochs=30):
2     for epoch in range(num_epochs):
3         print(f"Epoch {epoch+1}/{num_epochs}")
4         print('-' * 10)
5
6         model.train()
7         running_loss = 0.0
8         running_corrects = 0
9
10        for inputs, labels in jaffe_loader:
11            inputs, labels = inputs.to(device), labels.to(device)
12
13            optimizer.zero_grad()
14            outputs = model(inputs)
15            loss = criterion(outputs, labels)
16            loss.backward()
17            optimizer.step()
18
19            _, preds = torch.max(outputs, 1)
20            running_loss += loss.item() * inputs.size(0)
21            running_corrects += torch.sum(preds == labels.data)
22
23        scheduler.step()
24        epoch_loss = running_loss / len(jaffe_dataset)
25        epoch_acc = running_corrects.double() / len(jaffe_dataset)
26
27        print(f"Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}")
28    return model
```

```

29
30 model = train_model(model, criterion, optimizer, scheduler, num_epochs=2)
31

```



Epoch 1/2

Loss: 3.6091 Acc: 0.0188

Epoch 2/2

Loss: 3.4231 Acc: 0.0376

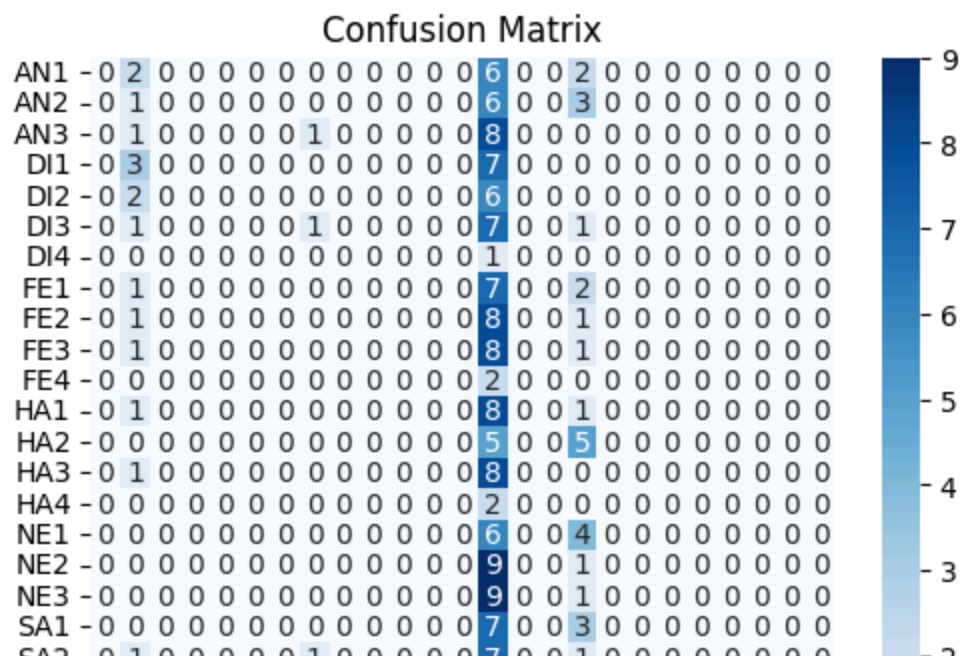
```

1 y_true, y_pred, y_proba = [], [], []
2
3 model.eval()
4 with torch.no_grad():
5     for inputs, labels in jaffe_loader:
6         inputs = inputs.to(device)
7         outputs = model(inputs)
8         probabilities = torch.softmax(outputs, dim=1)
9         preds = torch.argmax(probabilities, axis=1)
10
11     y_true.extend(labels.numpy())
12     y_pred.extend(preds.cpu().numpy())
13     y_proba.extend(probabilities.cpu().numpy())
14
15 y_true = np.array(y_true)
16 y_pred = np.array(y_pred)
17 y_proba = np.array(y_proba)
18
19
20
21 #
22
23 def evaluate_model(model_name, y_true, y_pred, y_proba):
24     labels = list(jaffe_dataset.label_map.keys())
25     accuracy = accuracy_score(y_true, y_pred)
26     precision = precision_score(y_true, y_pred, average='weighted', zero_div
27     recall = recall_score(y_true, y_pred, average='weighted')
28     f1 = f1_score(y_true, y_pred, average='weighted')
29     cm = confusion_matrix(y_true, y_pred)
30
31     print(f"[{model_name}] Accuracy: {accuracy:.4f}")
32     print("Precision:", precision)
33     print("Recall:", recall)
34     print("F1-score:", f1)
35     print("Classification Report:\n", classification_report(y_true, y_pred,
36

```

```
17 plt.figure(figsize=(6,5))
18 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, y
19 plt.title('Confusion Matrix')
20 plt.show()
21
22 evaluate_model("ResNet18 on JAFFE", y_true, y_pred, y_proba)
23
```

	precision	recall	f1-score	support
AN1	0.00	0.00	0.00	10
AN2	0.06	0.10	0.07	10
AN3	0.00	0.00	0.00	10
DI1	0.00	0.00	0.00	10
DI2	0.00	0.00	0.00	8
DI3	0.00	0.00	0.00	10
DI4	0.00	0.00	0.00	1
FE1	0.00	0.00	0.00	10
FE2	0.00	0.00	0.00	10
FE3	0.00	0.00	0.00	10
FE4	0.00	0.00	0.00	2
HA1	0.00	0.00	0.00	10
HA2	0.00	0.00	0.00	10
HA3	0.05	0.89	0.10	9
HA4	0.00	0.00	0.00	2
NE1	0.00	0.00	0.00	10
NE2	0.03	0.10	0.05	10
NE3	0.00	0.00	0.00	10
SA1	0.00	0.00	0.00	10
SA2	0.00	0.00	0.00	10
SA3	0.00	0.00	0.00	10
SA5	0.00	0.00	0.00	1
SU1	0.00	0.00	0.00	10
SU2	0.00	0.00	0.00	10
SU3	0.00	0.00	0.00	10
accuracy			0.05	213
macro avg	0.01	0.04	0.01	213
weighted avg	0.01	0.05	0.01	213





```

1 csv_path = '/content/drive/MyDrive/Datasets/CK+/ckextended.csv'
2 df_ck = pd.read_csv(csv_path)
3

```

```

1 from torch.utils.data import Dataset
2 from PIL import Image
3 import numpy as np
4
5 # Dataset الذي يرث من CKPlusDataset تعريف كلاس
6 # DataLoa إلى بيانات صور يمكن قراءتها بال DataFrame المخزنة في CK+ هذا الكلاس يستخدم لتحويل بيانات
7
8 class CKPlusDataset(Dataset):
9     def __init__(self, dataframe, transform=None):
10         self.df = dataframe # DataFrame يحتوي على أعمدة 'pixels' و
11         self.transform = transform # التحويلات التي سيتم تطبيقها على الصور (مثل ToT
12
13     def __len__(self):
14         return len(self.df) # عدد العينات في مجموعة البيانات
15
16     def __getitem__(self, idx):
17         # تحويل سلسلة البكسلات إلى مصفوفة أرقام صحيحة
18         pixel_values = [int(p) for p in self.df.iloc[idx]['pixels'].split()]
19
20         # CK+ حسب قاعدة 48x48 بحجم 48D تحويل البكسلات إلى صورة 2
21         image = np.array(pixel_values).reshape(48, 48).astype(np.uint8)
22
23         # torchvision.transforms للتعامل مع PIL تحويل الصورة إلى كائن
24         image = Image.fromarray(image)
25
26         # تطبيق التحويلات المحددة إن وجدت
27         if self.transform:
28             image = self.transform(image)
29
30         # DataFrame من (Label) استخراج التصنيف
31         label = int(self.df.iloc[idx]['emotion'])
32
33         return image, label # هذا السطر الصحيح ✓
34

```

```

1 from torchvision import transforms
2 from torch.utils.data import DataLoader
3
4 # تعريف التحويلات التي تطبق على كل صورة
5 transform = transforms.Compose([
6     transforms.RandomHorizontalFlip(),
7     transforms.RandomVerticalFlip(),
8     transforms.RandomRotation(10),
9     transforms.Resize((128,128)),
10    transforms.ToTensor()
11 ])

```

```

12
13 # إنشاء مجموعة البيانات واللودر
14 ck_dataset = CKPlusDataset(df_ck, transform=transform)
15 ck_loader = DataLoader(ck_dataset, batch_size=8, shuffle=True)
16
17 # حساب عدد المشاعر (الكلاسات)
18 num_classes_ck = df_ck['emotion'].nunique()
19
20
21 1 model = models.resnet18(pretrained=False)
22 2 model.fc = nn.Linear(model.fc.in_features, num_classes_ck)
23 3 model = model.to(device)
24 4
25 5 criterion = nn.CrossEntropyLoss()
26 6 optimizer = optim.AdamW(model.parameters(), lr=0.001)
27 7 scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)
28 8
29
30 1 from torchvision import transforms
31 2
32 3 transform = transforms.Compose([
33 4     transforms.Lambda(lambda img: img.convert("RGB")), # تحويل من رمادي إلى RGB
34 5     transforms.RandomHorizontalFlip(),
35 6     transforms.RandomVerticalFlip(),
36 7     transforms.RandomRotation(10),
37 8     transforms.Resize((128,128)),
38 9     transforms.ToTensor()
39 10 ])
40 11
41
42 1 transforms.Lambda(lambda img: img.convert("RGB"))
43 2 from torchvision import transforms
44 3
45 4 transform = transforms.Compose([
46 5     transforms.Lambda(lambda img: img.convert("RGB")), # تحويل من رمادي إلى RGB
47 6     transforms.Resize((128, 128)),
48 7     transforms.ToTensor()
49 8 ])
50 9 ck_dataset = CKPlusDataset(df_ck, transform=transform)
51 10 ck_loader = DataLoader(ck_dataset, batch_size=8, shuffle=True)
52 11
53
54 1 # تحديد عدد الدورات (epochs)
55 2 epochs = 190

```

```

3
4
5 for epoch in range(epochs):
6     print(f"\nEpoch {epoch+1}/{epochs}")
7     model.train() # وضع النموذج في طور التدريب
8
9     running_loss = 0.0
10    correct = 0
11    total = 0
12
13    # حلقة على دفعات البيانات
14    for inputs, labels in ck_loader:
15        inputs, labels = inputs.to(device), labels.to(device)
16
17        # تصفير التدرجات السابقة
18        optimizer.zero_grad()
19
20        # تمرير الصور داخل النموذج للحصول على التنبؤات
21        outputs = model(inputs)
22
23        # حساب الخسارة
24        loss = criterion(outputs, labels)
25
26        # تحديث الأوزان بناءً على الخسارة
27        loss.backward()
28        optimizer.step()
29
30        # تجميع الخسارة
31        running_loss += loss.item()
32
33        # حساب التوقعات الصحيحة
34        _, preds = torch.max(outputs, 1)
35        correct += (preds == labels).sum().item()
36        total += labels.size(0)
37
38    # حساب المتوسط لكل epoch
39    epoch_loss = running_loss / len(ck_loader)
40    epoch_acc = correct / total
41
42    print(f"Loss: {epoch_loss:.4f}  Acc: {epoch_acc:.4f}")
43

```



Epoch 1/190
Loss: 1.1092 Acc: 0.6935

Epoch 2/190

Loss: 0.7597 Acc: 0.7837

Epoch 3/190

Loss: 0.5296 Acc: 0.8380

Epoch 4/190

Loss: 0.4512 Acc: 0.8587

Epoch 5/190

Loss: 0.3582 Acc: 0.8793

Epoch 6/190

Loss: 0.2690 Acc: 0.9120

Epoch 7/190

Loss: 0.2628 Acc: 0.9130

Epoch 8/190

Loss: 0.1977 Acc: 0.9370

Epoch 9/190

Loss: 0.1322 Acc: 0.9565

Epoch 10/190

Loss: 0.1125 Acc: 0.9652

Epoch 11/190

Loss: 0.1597 Acc: 0.9522

Epoch 12/190

Loss: 0.1103 Acc: 0.9641

Epoch 13/190

Loss: 0.0833 Acc: 0.9772

Epoch 14/190

Loss: 0.0693 Acc: 0.9750

Epoch 15/190

Loss: 0.0359 Acc: 0.9880

Epoch 16/190

Loss: 0.0337 Acc: 0.9924

Epoch 17/190

Loss: 0.1201 Acc: 0.9609

Epoch 18/190

Loss: 0.1133 Acc: 0.9674

Epoch 19/190

Loss: 0.0743 Acc: 0.9761

```
1 y_true, y_pred, y_proba = [], [], []
```

```
2
```

```
3 model.eval()
4 with torch.no_grad():
5     for inputs, labels in ck_loader:
6         inputs = inputs.to(device)
7         outputs = model(inputs)
8         probabilities = torch.softmax(outputs, dim=1)
9         preds = torch.argmax(probabilities, axis=1)
10
11     y_true.extend(labels.numpy())
12     y_pred.extend(preds.cpu().numpy())
13     y_proba.extend(probabilities.cpu().numpy())
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```



[ResNet18 on CK+] Accuracy: 1.0000

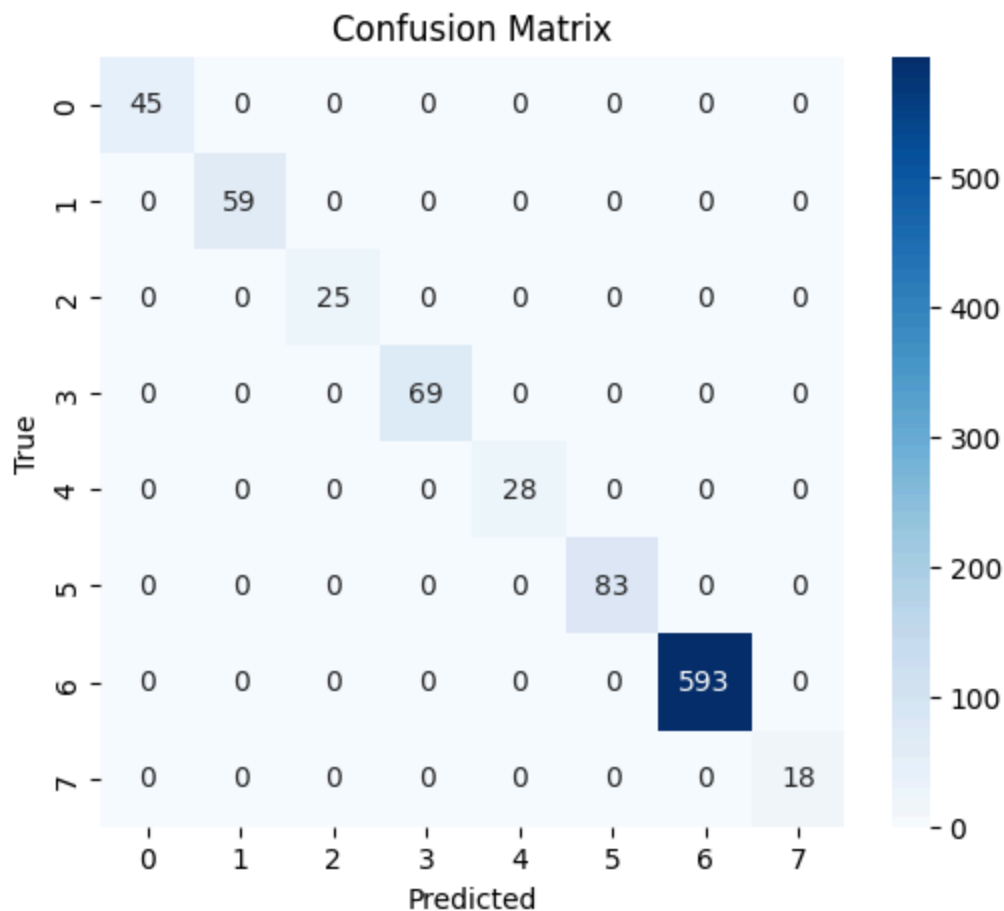
Precision: 1.0

Recall: 1.0

F1-score: 1.0

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	45
1	1.00	1.00	1.00	59
2	1.00	1.00	1.00	25
3	1.00	1.00	1.00	69
4	1.00	1.00	1.00	28
5	1.00	1.00	1.00	83
6	1.00	1.00	1.00	593
7	1.00	1.00	1.00	18
accuracy			1.00	920
macro avg	1.00	1.00	1.00	920
weighted avg	1.00	1.00	1.00	920



```

1 model = models.mobilenet_v2(pretrained=False)
2 model.classifier[1] = nn.Linear(model.last_channel, num_classes_ck)
3 model = model.to(device)
4
5 criterion = nn.CrossEntropyLoss()

```

```

6 optimizer = optim.AdamW(model.parameters(), lr=0.001)
7 scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)
8
1 epochs = 2
2 for epoch in range(epochs):
3     print(f"\nEpoch {epoch+1}/{epochs}")
4     model.train()
5     running_loss, correct = 0.0, 0
6
7     for inputs, labels in ck_loader:
8         inputs, labels = inputs.to(device), labels.to(device)
9         optimizer.zero_grad()
10        outputs = model(inputs)
11        loss = criterion(outputs, labels)
12        loss.backward()
13        optimizer.step()
14
15        _, preds = torch.max(outputs, 1)
16        correct += torch.sum(preds == labels.data)
17        running_loss += loss.item() * inputs.size(0)
18
19    epoch_loss = running_loss / len(ck_dataset)
20    epoch_acc = correct.double() / len(ck_dataset)
21    print(f"Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}")
22

```



```

Epoch 1/2
Loss: 1.4110 Acc: 0.6228

```

```

Epoch 2/2
Loss: 1.2530 Acc: 0.6489

```

```

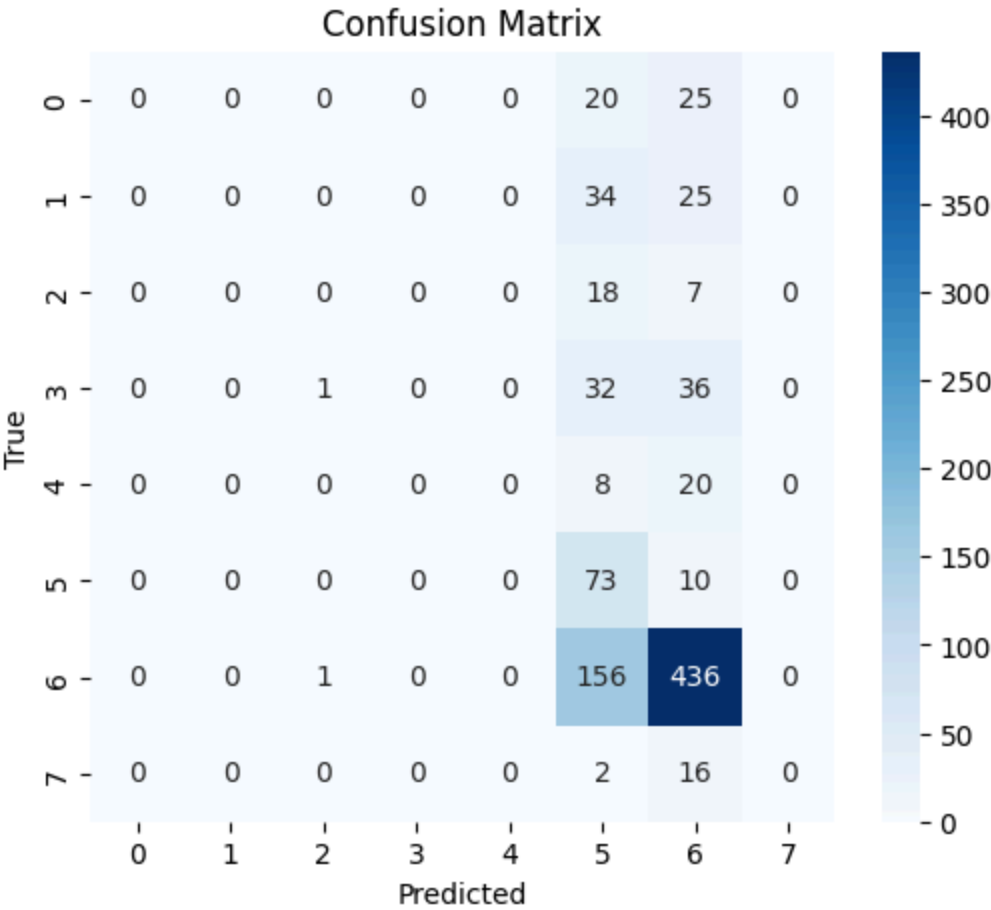
1 y_true, y_pred, y_proba = [], [], []
2
3 model.eval()
4 with torch.no_grad():
5     for inputs, labels in ck_loader:
6         inputs = inputs.to(device)
7         outputs = model(inputs)
8         probabilities = torch.softmax(outputs, dim=1)
9         preds = torch.argmax(probabilities, axis=1)
10
11    y_true.extend(labels.numpy())
12    y_pred.extend(preds.cpu().numpy())

```

```
13         y_proba.extend(probabilities.cpu().numpy())
14
1
2 evaluate_model("MobileNetV2 on CK+", y_true, y_pred, y_proba)
```

 [MobileNetV2 on CK+] Accuracy: 0.5533
Precision: 0.5079493681350478
Recall: 0.5532608695652174
F1-score: 0.5121360733389817
Classification Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	45
1	0.00	0.00	0.00	59
2	0.00	0.00	0.00	25
3	0.00	0.00	0.00	69
4	0.00	0.00	0.00	28
5	0.21	0.88	0.34	83
6	0.76	0.74	0.75	593
7	0.00	0.00	0.00	18
accuracy			0.55	920
macro avg	0.12	0.20	0.14	920
weighted avg	0.51	0.55	0.51	920



```

1 import pandas as pd
2 import numpy as np
3 import torch
4 from torch.utils.data import Dataset, DataLoader
5 from torchvision import transforms, models
6 from PIL import Image
7 import cv2
8
9 csv_path = '/content/drive/MyDrive/Datasets/CK+/ckextended.csv'
10 df_ck = pd.read_csv(csv_path)
11
12 # التأكد من الفئات وعددها
13 print("الفئات المتوفرة:", df_ck['emotion'].unique())
14 num_classes_ck = df_ck['emotion'].nunique()
15 print("عدد الفئات:", num_classes_ck)
16
17 # تصحيح: إزالة أي فئة خارج النطاق إذا وجدت
18 df_ck = df_ck[df_ck['emotion'] < num_classes_ck].reset_index(drop=True)
19

```

↩ [5 2 4 1 0 7 3 6] الفئات المتوفرة:
عدد الفئات: 8

```

1 class CKPlusDataset(Dataset):
2     def __init__(self, dataframe, transform=None):
3         self.df = dataframe
4         self.transform = transform
5
6     def __len__(self):
7         return len(self.df)
8
9     def __getitem__(self, idx):
10        pixel_values = [int(p) for p in self.df.iloc[idx]['pixels'].split()]
11        image = np.array(pixel_values, dtype='uint8').reshape(48, 48)
12        image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
13        image = cv2.resize(image, (128,128))
14        image = Image.fromarray(image)
15        label = self.df.iloc[idx]['emotion']
16        if self.transform:
17            image = self.transform(image)
18        return image, label
19
20 transform = transforms.Compose([
21     transforms.RandomHorizontalFlip(),
22     transforms.RandomVerticalFlip(),
23     transforms.RandomRotation(10),

```

```

24     transforms.Resize((128,128)),
25     transforms.ToTensor()
26 ])
27
28 ck_dataset = CKPlusDataset(df_ck, transform=transform)
29 ck_loader = DataLoader(ck_dataset, batch_size=8, shuffle=True)
30
31
32 1 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
33 2
34 3 model = models.efficientnet_b0(pretrained=False)
35 4 model.classifier[1] = torch.nn.Linear(model.classifier[1].in_features, num_c
36 5 model = model.to(device)
37 6
38 7 criterion = torch.nn.CrossEntropyLoss()
39 8 optimizer = torch.optim.AdamW(model.parameters(), lr=0.001)
40 9 scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.'
41 10
42
43 1 def train_model(model, criterion, optimizer, scheduler, num_epochs=2
44 2     for epoch in range(num_epochs):
45 3         print(f"\nEpoch {epoch+1}/{num_epochs}")
46 4         model.train()
47 5         running_loss, correct = 0.0, 0
48 6
49 7         for inputs, labels in ck_loader:
50 8             inputs, labels = inputs.to(device), labels.to(device)
51 9
52 10            optimizer.zero_grad()
53 11            outputs = model(inputs)
54 12            loss = criterion(outputs, labels)
55 13            loss.backward()
56 14            optimizer.step()
57 15
58 16            _, preds = torch.max(outputs, 1)
59 17            correct += torch.sum(preds == labels.data)
60 18            running_loss += loss.item() * inputs.size(0)
61 19
62 20            scheduler.step()
63 21            epoch_loss = running_loss / len(ck_dataset)
64 22            epoch_acc = correct.double() / len(ck_dataset)
65 23            print(f"Loss: {epoch_loss:.4f}, Accuracy: {epoch_acc:.4f}")
66 24     return model
67 25

```

```
1 model = train_model(model, criterion, optimizer, scheduler, num_epoc
2
```



Epoch 1/2
Loss: 1.5203, Accuracy: 0.6152

Epoch 2/2
Loss: 1.3856, Accuracy: 0.6304

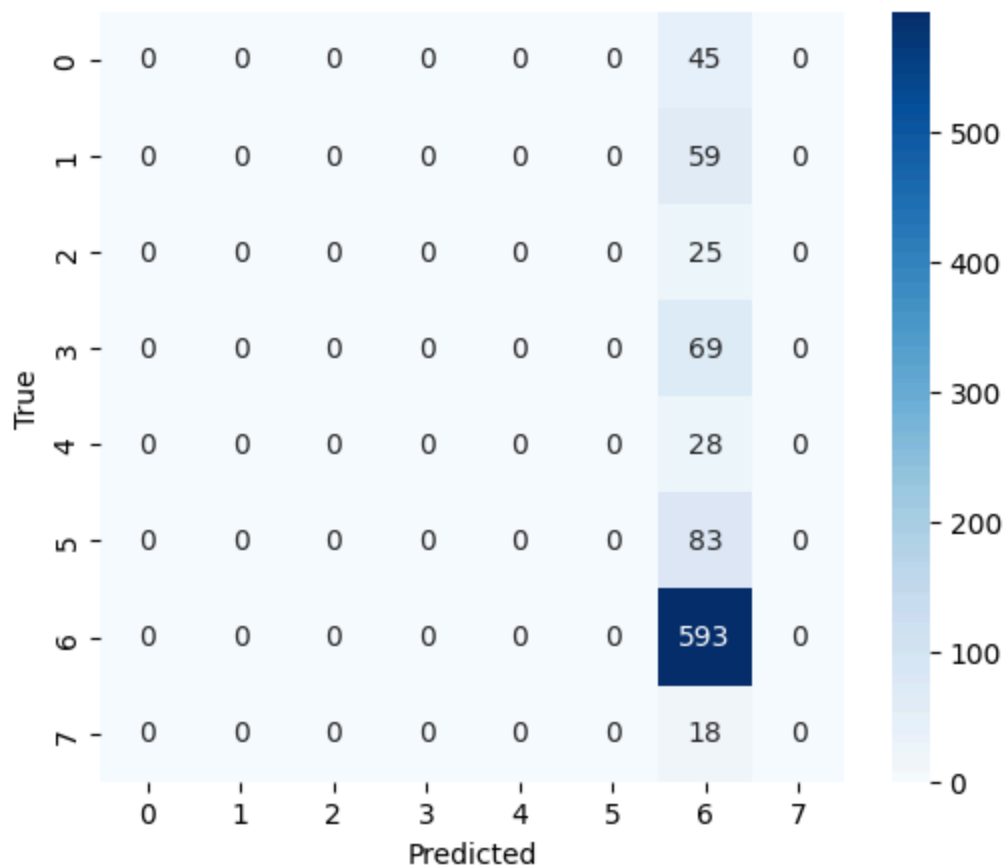
```
1 from sklearn.metrics import accuracy_score, precision_score, recall_
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4
5 y_true, y_pred = [], []
6
7 model.eval()
8 with torch.no_grad():
9     for inputs, labels in ck_loader:
10         inputs = inputs.to(device)
11         outputs = model(inputs)
12         _, preds = torch.max(outputs, 1)
13
14         y_true.extend(labels.numpy())
15         y_pred.extend(preds.cpu().numpy())
16
17 cm = confusion_matrix(y_true, y_pred)
18 print("Classification Report:\n", classification_report(y_true, y_pr
19
20 plt.figure(figsize=(6,5))
21 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
22 plt.title('Confusion Matrix')
23 plt.xlabel('Predicted')
24 plt.ylabel('True')
25 plt.show()
26
```



Classification Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	45
1	0.00	0.00	0.00	59
2	0.00	0.00	0.00	25
3	0.00	0.00	0.00	69
4	0.00	0.00	0.00	28
5	0.00	0.00	0.00	83
6	0.64	1.00	0.78	593
7	0.00	0.00	0.00	18
accuracy			0.64	920
macro avg	0.08	0.12	0.10	920
weighted avg	0.42	0.64	0.51	920

Confusion Matrix



```

1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 from torchvision import models
5 from torch.optim import lr_scheduler
6
7 # 1. تعريف عدد الكلاسات حسب بيانات CK+
8 num_classes_ck = df_ck['emotion'].nunique()
9

```

```

10 # 2. وتعديله DenseNet121 تحميل نموذج
11 model = models.densenet121(pretrained=False)
12 model.classifier = nn.Linear(model.classifier.in_features, num_classes_ck)
13 model = model.to(device)
14
15 # 3. تعريف الخسارة، المحسن، وجدولة التعلم
16 criterion = nn.CrossEntropyLoss()
17 optimizer = optim.AdamW(model.parameters(), lr=0.001)
18 exp_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
19
20 # 4. تعريف دالة التدريب
21 def train_model(model, criterion, optimizer, scheduler, num_epochs=1):
22     for epoch in range(num_epochs):
23         print(f"\nEpoch {epoch+1}/{num_epochs}")
24         model.train()
25         running_loss = 0.0
26         correct = 0
27         total = 0
28
29         for inputs, labels in ck_loader:
30             inputs, labels = inputs.to(device), labels.to(device)
31
32             optimizer.zero_grad()
33             outputs = model(inputs)
34             loss = criterion(outputs, labels)
35             loss.backward()
36             optimizer.step()
37
38             running_loss += loss.item()
39             _, preds = torch.max(outputs, 1)
40             correct += (preds == labels).sum().item()
41             total += labels.size(0)
42
43         scheduler.step()
44         epoch_loss = running_loss / len(ck_loader)
45         epoch_acc = correct / total
46         print(f"Loss: {epoch_loss:.4f}  Acc: {epoch_acc:.4f}")
47     return model
48
49 # 5. تنفيذ التدريب
50 model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epoch
51

```



Epoch 1/1

Loss: 1.4346 Acc: 0.6163

```

1 model = models.vgg16(pretrained=False)
2 model.classifier[6] = nn.Linear(model.classifier[6].in_features, num_classes)
3 model = model.to(device)
4
5 optimizer = optim.AdamW(model.parameters(), lr=0.001)
6 model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epoch
7

```



Epoch 1/1
Loss: 4.0951 Acc: 0.6283

```

1 y_true, y_pred, y_proba = [], [], []
2
3 model.eval()
4 with torch.no_grad():
5     for inputs, labels in ck_loader:
6         inputs = inputs.to(device)
7         outputs = model(inputs)
8         probabilities = torch.softmax(outputs, dim=1)
9         preds = torch.argmax(probabilities, axis=1)
10
11     y_true.extend(labels.numpy())
12     y_pred.extend(preds.cpu().numpy())
13     y_proba.extend(probabilities.cpu().numpy())
14
15 y_true = np.array(y_true)
16 y_pred = np.array(y_pred)
17 y_proba = np.array(y_proba)
18
19 evaluate_model("اسم الموديل", y_true, y_pred, y_proba)
20

```



[اسم الموديل] Accuracy: 0.6446

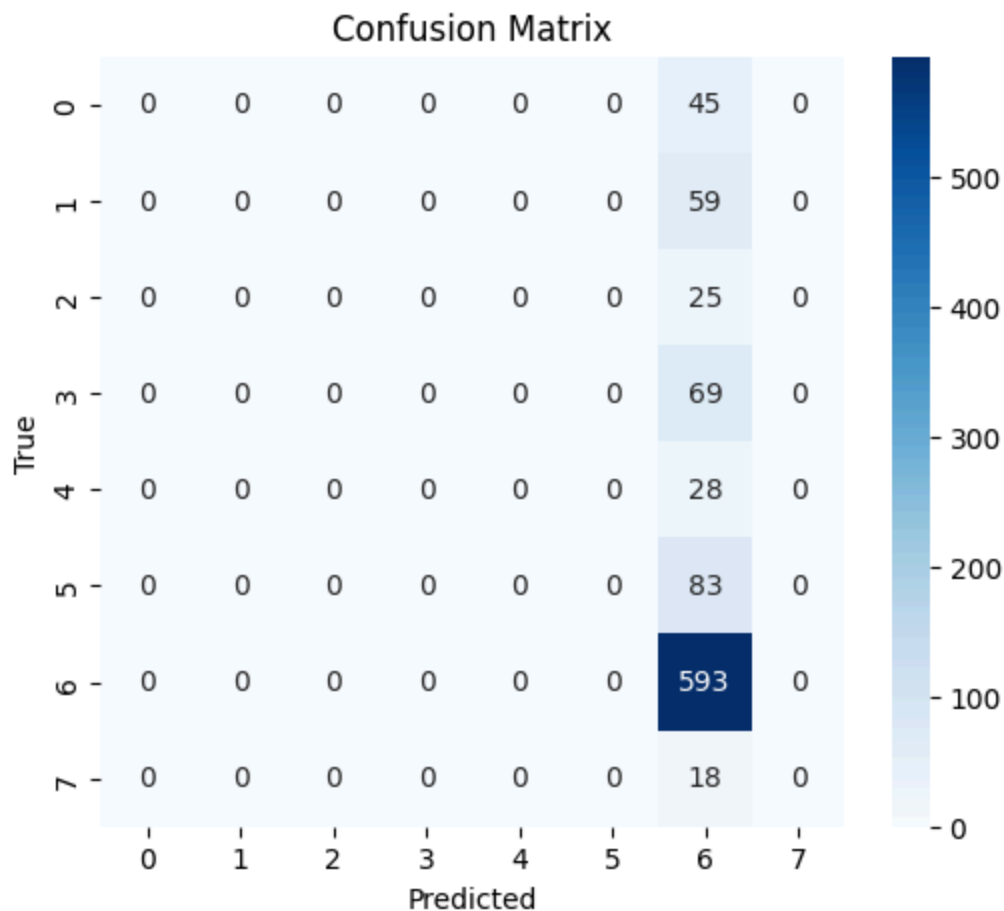
Precision: 0.41546431947069945

Recall: 0.6445652173913043

F1-score: 0.5052573349808903

Classification Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	45
1	0.00	0.00	0.00	59
2	0.00	0.00	0.00	25
3	0.00	0.00	0.00	69
4	0.00	0.00	0.00	28
5	0.00	0.00	0.00	83
6	0.64	1.00	0.78	593
7	0.00	0.00	0.00	18
accuracy			0.64	920
macro avg	0.08	0.12	0.10	920
weighted avg	0.42	0.64	0.51	920



```

1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 from torchvision import models, transforms
5 from torch.utils.data import Dataset, DataLoader

```

```

6 from PIL import Image
7 import os
8 import cv2
9 import numpy as np

1 transform = transforms.Compose([
2     transforms.RandomHorizontalFlip(),
3     transforms.RandomVerticalFlip(),
4     transforms.RandomRotation(20),
5     transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
6     transforms.Resize((224,224)),
7     transforms.ToTensor(),
8     transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
9 ])
10

1 # 2. Dataset لـ JAFFE
2 class JaffeDataset(Dataset):
3     def __init__(self, root_dir, transform=None):
4         self.root_dir = root_dir
5         if not os.path.exists(root_dir):
6             print(f"المجلد {root_dir} غير موجود. الرجاء التأكد من ربط المجلد {root_dir} بـ Google Drive")
7             self.image_files = []
8         else:
9             self.image_files = [f for f in os.listdir(root_dir) if f.endswith(
10                 self.transform = transform
11
12     def __len__(self):
13         return len(self.image_files)
14
15     def __getitem__(self, idx):
16         img_path = os.path.join(self.root_dir, self.image_files[idx])
17         image = Image.open(img_path).convert('RGB')
18         if self.transform:
19             image = self.transform(image)
20         label = int(self.image_files[idx].split('.')[1]) # أو حسب التنظيم المتبع
21         return image, label

1 emotion_labels = set()
2
3 for f in os.listdir(jaffe_path):
4     if f.endswith(".tiff"):
5         parts = f.split('.')
6         if len(parts) >= 2:
7             emotion_labels.add(parts[1]) # نأخذ الرمز الأول فقط ← "SU2" مثل

```

```

8
9 # فقط أول حرفين تمثل المشاعر (SU, NE, etc)
10 emotion_labels = set([label[:2] for label in emotion_labels])
11 num_classes_jaffe = len(emotion_labels)
12

```



```

-----
NameError                                Traceback (most recent call last)
/tmp/ipython-input-1014668409.py in <cell line: 0>()
      1 emotion_labels = set()
      2
----> 3 for f in os.listdir(jaffe_path):
      4     if f.endswith(".tiff"):
      5         parts = f.split('.')

NameError: name 'jaffe_path' is not defined

```

Next steps: [Explain error](#)

```

1 import os
2 from PIL import Image
3 import torch
4 from torch.utils.data import Dataset
5 from torchvision import transforms
6
7 # JAFFE لمعالجة بيانات JaffeDataset تعريف كلاس
8 class JaffeDataset(Dataset):
9     def __init__(self, root_dir, transform=None):
10         self.root_dir = root_dir
11         self.image_files = [f for f in os.listdir(root_dir) if f.endswith(".tiff")]
12         self.transform = transform
13
14         # تعريف قاموس تحويل رموز المشاعر إلى أرقام
15         self.emotion_dict = {
16             'AN': 0, # Angry
17             'DI': 1, # Disgust
18             'FE': 2, # Fear
19             'HA': 3, # Happy
20             'NE': 4, # Neutral
21             'SA': 5, # Sad
22             'SU': 6 # Surprise
23         }
24
25     def __len__(self):
26         return len(self.image_files)
27

```

```

28     def __getitem__(self, idx):
29         # تحميل الصورة
30         image_path = os.path.join(self.root_dir, self.image_files[idx])
31         image = Image.open(image_path).convert('L') # الصورة رمادية
32
33         # تطبيق التحويلات إن وجدت
34         if self.transform:
35             image = self.transform(image)
36
37         # استخراج رمز المشاعر من اسم الملف (مثلاً SA من KA.SA1.93.tiff)
38         emotion_code = self.image_files[idx].split('.')[1][:2]
39         label = self.emotion_dict[emotion_code]
40
41         return image, label
42

```

```

1 from google.colab import drive
2 drive.mount('/content/drive')
3
4 !ls "/content/drive/MyDrive/Datasets/FER-2013/train"
5

```

⇒ Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount(
 angry disgust fear happy neutral sad surprise

```

1 from torchvision import datasets, transforms
2
3 transform = transforms.Compose([
4     transforms.Grayscale(num_output_channels=3), # رمادية FER-2013 لأن صور
5     transforms.Resize((224, 224)),
6     transforms.ToTensor()
7 ])
8
9 train_dir = "/content/drive/MyDrive/Datasets/FER-2013/train"
10 full_dataset = datasets.ImageFolder(root=train_dir, transform=transform)
11
12 print(f"عدد الصور: {len(full_dataset)}")
13 print(f"الفئات: {full_dataset.classes}")
14

```

⇒ عدد الصور: 26580
 الفئات: ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']

```

1 import random
2 from torch.utils.data import Subset

```

```

3 from collections import defaultdict
4
5 samples_per_class = 1000 # الحد الأقصى لكل فئة
6
7 # استخراج الليبلات (labels)
8 targets = full_dataset.targets
9 class_indices = defaultdict(list)
10
11 # تجميع الفهارس لكل فئة
12 for idx, label in enumerate(targets):
13     class_indices[label].append(idx)
14
15 # اختيار عينات عشوائية من كل فئة
16 selected_indices = []
17 for cls, indices in class_indices.items():
18     if len(indices) > samples_per_class:
19         selected_indices.extend(random.sample(indices, samples_per_class))
20     else:
21         selected_indices.extend(indices) # لو أقل من 1000 نأخذ الكل
22
23 # إنشاء Subset
24 subset_dataset = Subset(full_dataset, selected_indices)
25
26 print(f"📦 Subset جاهز: {len(subset_dataset)} صورة.")
27

```

📦 Subset جاهز: 6393 صورة.

```

1 from torch.utils.data import DataLoader
2
3 train_loader = DataLoader(subset_dataset, batch_size=16, shuffle=True)
4
5
6
7 # إنشاء قائمة جديدة تحتوي على 1000 عينة لكل فئة فقط
8 samples_per_class = 1000
9 targets = full_dataset.targets
10 class_indices = {cls: [] for cls in range(len(full_dataset.classes))}
11
12 # جمع المؤشرات حسب الفئة
13 for idx, label in enumerate(targets):
14     if len(class_indices[label]) < samples_per_class:
15         class_indices[label].append(idx)
16
17 # دمج المؤشرات معًا
18

```

```

13 final_indices = [idx for indices in class_indices.values() for idx in indice
14 subset_dataset = Subset(full_dataset, final_indices)

1 import torch, torch.nn as nn, torch.optim as optim
2 from torch.utils.data import DataLoader, Subset
3 from torchvision import models, transforms
4 import random
5
6 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
7
8 # بـ 3 قنوات Tensor يعيد full_dataset مثال: تأكد أن
9
10
11 # عشوائي صغير للتجريب subset
12 if 'subset_dataset' not in globals():
13     n = min(len(full_dataset), 1000)
14     idx = random.sample(range(len(full_dataset)), n)
15     subset_dataset = Subset(full_dataset, idx)
16
17 # نموذج
18 model = models.resnet18(pretrained=False)
19 model.fc = nn.Linear(model.fc.in_features, len(full_dataset.classes))
20 model = model.to(device)
21
22 # تدريب
23 train_loader = DataLoader(subset_dataset, batch_size=16, shuffle=True)
24 criterion = nn.CrossEntropyLoss()
25 optimizer = optim.Adam(model.parameters(), lr=1e-4)
26 epochs = 1000 # ستوقف مبكرًا على الأغلب
27
28 for epoch in range(epochs):
29     model.train()
30     running_loss, correct = 0.0, 0
31     for inputs, labels in train_loader:
32         inputs, labels = inputs.to(device), labels.to(device)
33
34         optimizer.zero_grad()
35         outputs = model(inputs)
36         loss = criterion(outputs, labels)
37         loss.backward()
38         optimizer.step()
39
40     running_loss += loss.item() * inputs.size(0)
41     preds = outputs.argmax(1)
42     correct += (preds == labels).sum().item()
43

```

```

44     epoch_loss = running_loss / len(subset_dataset)
45     epoch_acc  = correct / len(subset_dataset)
46     print(f"Epoch {epoch+1}/{epochs}, Loss: {epoch_loss:.4f}, Accura
47
48     # إيقاف مبكر بسيط: إذا وصلنا >97% عدة مرات متتالية، أوقف
49     if epoch >= 19 and epoch_acc > 0.97:
50         break
51

```

```

→ /usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: 1
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: 1
  warnings.warn(msg)
Epoch 1/1000, Loss: 1.8610, Accuracy: 0.2348
Epoch 2/1000, Loss: 1.6405, Accuracy: 0.3673
Epoch 3/1000, Loss: 1.4362, Accuracy: 0.4544
Epoch 4/1000, Loss: 1.2448, Accuracy: 0.5318
Epoch 5/1000, Loss: 0.9508, Accuracy: 0.6604
Epoch 6/1000, Loss: 0.5947, Accuracy: 0.7954
Epoch 7/1000, Loss: 0.3066, Accuracy: 0.9085
Epoch 8/1000, Loss: 0.1774, Accuracy: 0.9546
Epoch 9/1000, Loss: 0.1284, Accuracy: 0.9695
Epoch 10/1000, Loss: 0.1106, Accuracy: 0.9731
Epoch 11/1000, Loss: 0.0984, Accuracy: 0.9753
Epoch 12/1000, Loss: 0.1041, Accuracy: 0.9715
Epoch 13/1000, Loss: 0.1075, Accuracy: 0.9706
Epoch 14/1000, Loss: 0.1320, Accuracy: 0.9596
Epoch 15/1000, Loss: 0.0778, Accuracy: 0.9795
Epoch 16/1000, Loss: 0.0805, Accuracy: 0.9773
Epoch 17/1000, Loss: 0.0754, Accuracy: 0.9804
Epoch 18/1000, Loss: 0.0667, Accuracy: 0.9820
Epoch 19/1000, Loss: 0.0577, Accuracy: 0.9842
Epoch 20/1000, Loss: 0.0929, Accuracy: 0.9715

```

```

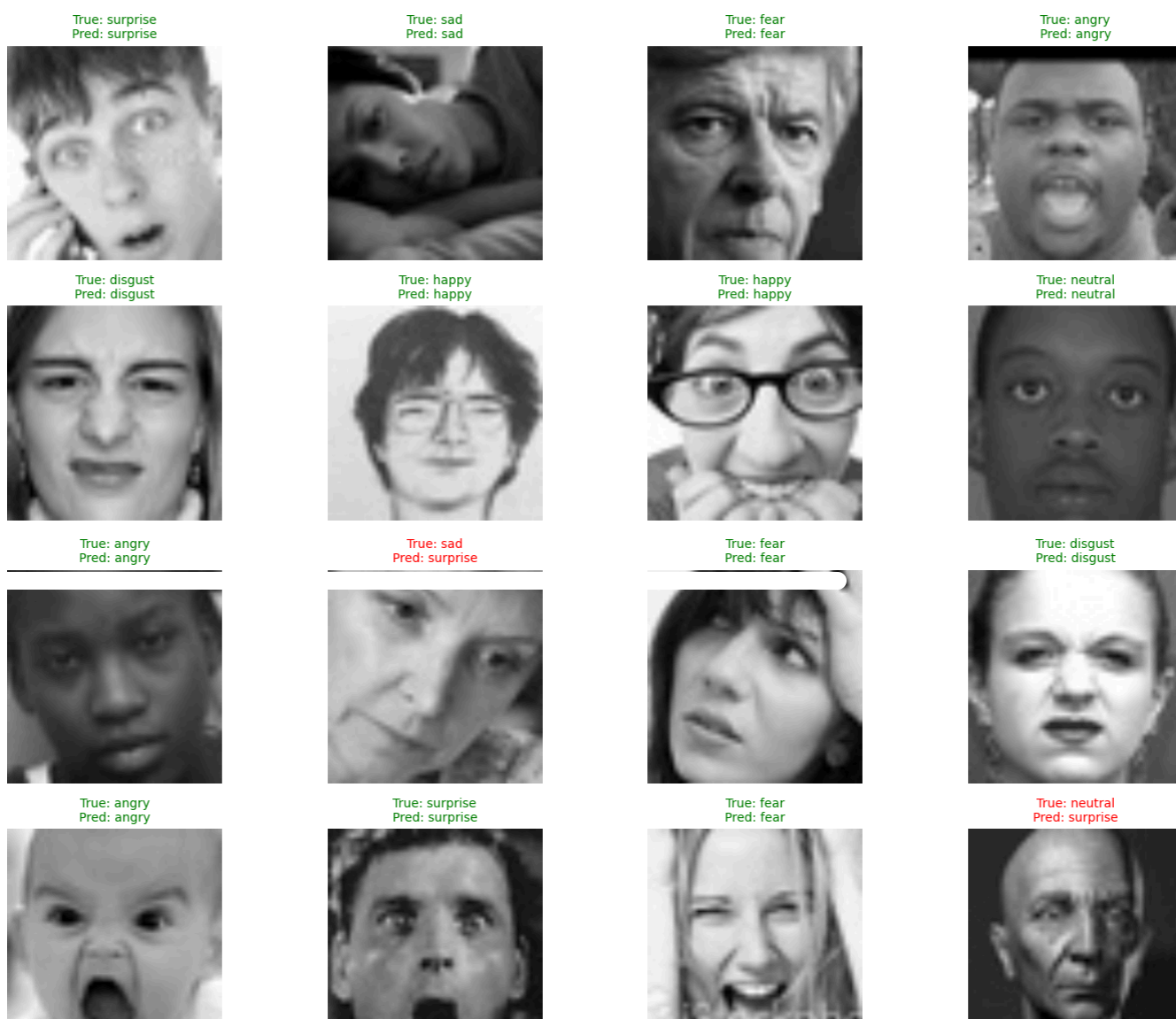
1 import matplotlib.pyplot as plt
2 import random
3 import torch
4
5 # معرفين model و full_dataset و subset_dataset تأكد أن
6
7 indices = random.sample(range(len(subset_dataset)), 8)
8 fig, axes = plt.subplots(2, 4, figsize=(15,6))
9
10 model.eval()
11 with torch.no_grad():
12     for ax, idx in zip(axes.flat, indices):
13         img, true_label = subset_dataset[idx]
14
15         # تجهيز الصورة للنموذج
16         inp = img.unsqueeze(0).to(device) # batch إضافة بعد

```

```

17     output = model(inp)
18     pred_label = output.argmax(1)
19
20     # تجهيز الصورة للعرض (من Tensor إلى numpy)
21     img_disp = img.permute(1, 2, 0).cpu().numpy()
22     img_disp = (img_disp - img_disp.min()) / (img_disp.max() - img_disp.min())
23
24     ax.imshow(img_disp)
25     ax.set_title(
26         f"True: {full_dataset.classes[true_label]}\nPred: {full_dataset.classes[pred_label]}"
27         ,
28         fontsize=10,
29         color='green' if pred_label.item() == true_label else 'red'
30     )
31     ax.axis('off')
32
33 plt.tight_layout()
34 plt.show()

```



```

1 from sklearn.metrics import classification_report, confusion_matrix
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4 import numpy as np
5
6 # 1. التنبؤ بالنتائج على نفس مجموعة التدريب المستخدمة
7 y_true = []
8 y_pred = []
9
10 model.eval()
11 with torch.no_grad():
12     for inputs, labels in train_loader:
13         inputs = inputs.to(device)
14         labels = labels.to(device)
15         outputs = model(inputs)
16         _, preds = torch.max(outputs, 1)
17         y_true.extend(labels.cpu().numpy())
18         y_pred.extend(preds.cpu().numpy())
19
20 # 2. حساب النتائج
21 print("📊 [Classification Report]")
22 print(classification_report(y_true, y_pred, target_names=full_dataset.classe
23
24 # 3. مصفوفة الالتباس
25 cm = confusion_matrix(y_true, y_pred)
26
27 plt.figure(figsize=(8,6))
28 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
29             xticklabels=full_dataset.classes,
30             yticklabels=full_dataset.classes)
31 plt.title("🔍 Confusion Matrix")
32 plt.xlabel("Predicted")
33 plt.ylabel("True")
34 plt.show()
35

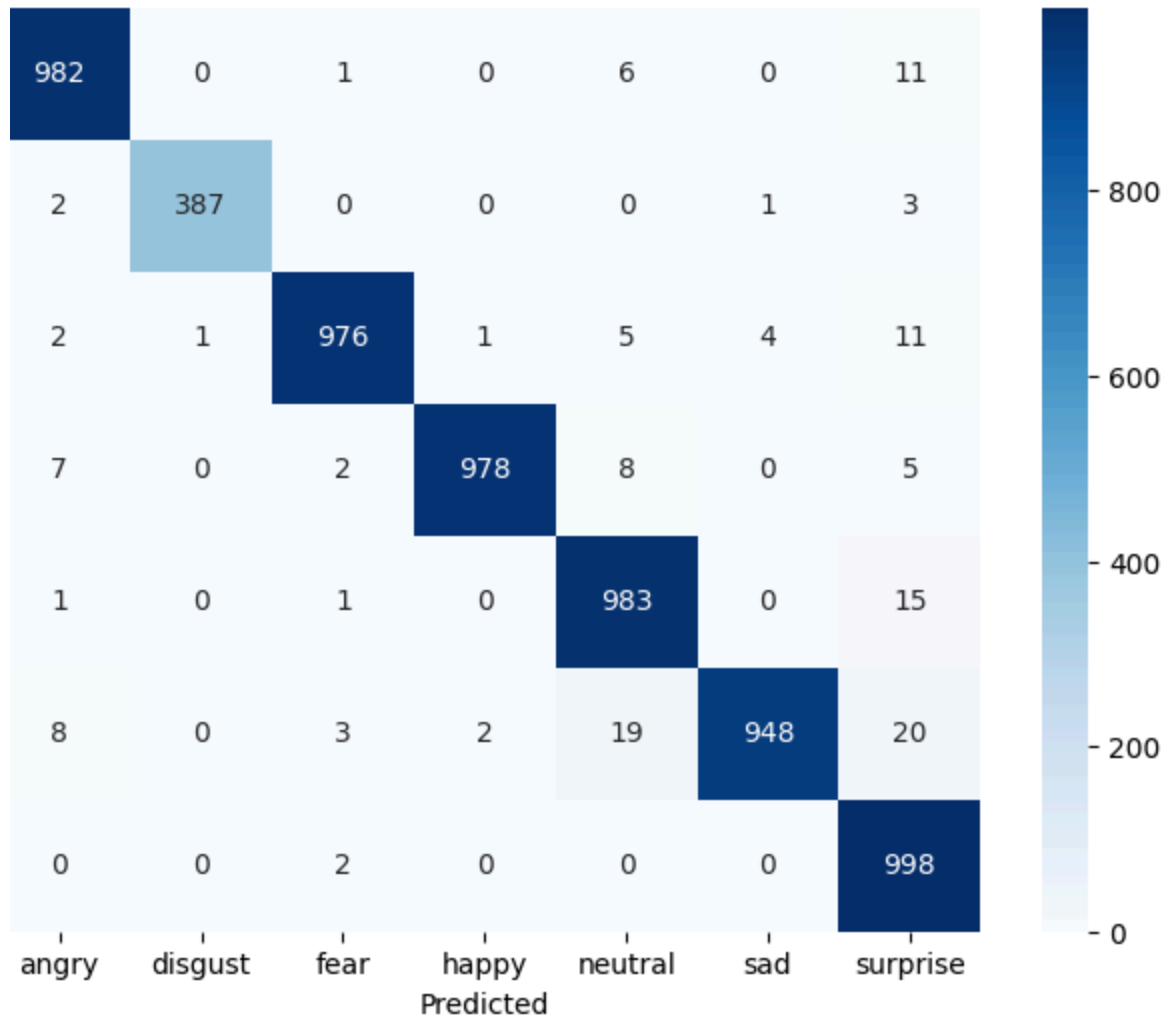
```



Classification Report]				
	precision	recall	f1-score	support
angry	0.98	0.98	0.98	1000
disgust	1.00	0.98	0.99	393
fear	0.99	0.98	0.98	1000
happy	1.00	0.98	0.99	1000
neutral	0.96	0.98	0.97	1000
sad	0.99	0.95	0.97	1000
surprise	0.94	1.00	0.97	1000
accuracy			0.98	6393
avg	0.98	0.98	0.98	6393
avg	0.98	0.98	0.98	6393

al/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyphs.print_figure(bytes_io, **kw)

Confusion Matrix



1 # حفظ النموذج بعد التدريب

2 torch.save(model.state_dict(), 'emotion_model.pth')

3

4 # حفظ النموذج داخل Google Drive

```

5 torch.save(model.state_dict(), '/content/drive/MyDrive/emotion_model.pth')
6

1 from torch.utils.data import random_split, DataLoader
2
3 # نسبة البيانات للتحقق (مثلاً 15%)
4 val_ratio = 0.15
5 val_len = int(len(subset_dataset) * val_ratio)
6 train_len = len(subset_dataset) - val_len
7
8 train_ds, val_ds = random_split(subset_dataset, [train_len, val_len],
9                                 generator=torch.Generator().manual_seed(42))
10
11 train_loader = DataLoader(train_ds, batch_size=64, shuffle=True, num_workers=
12 val_loader    = DataLoader(val_ds, batch_size=64, shuffle=False, num_workers=
13

1 for epoch in range(1, epochs+1):
2     ...
3     for x, y in val_loader:
4         ...
5

1 train_acc_history = []
2 val_acc_history = []
3
4 for epoch in range(1, epochs+1):
5     # ---- Training ----
6     model.train()
7     train_loss, train_correct, train_count = 0.0, 0, 0
8
9     for x, y in train_loader:
10         x, y = x.to(device), y.to(device)
11
12         optimizer.zero_grad()
13         logits = model(x)
14         loss = criterion(logits, y)
15         loss.backward()
16         optimizer.step()
17
18         train_loss += loss.item() * x.size(0)
19         train_correct += (logits.argmax(1) == y).sum().item()
20         train_count += x.size(0)
21
22     train_loss /= train_count

```

```

23 train_acc = train_correct / train_count
24 train_acc_history.append(train_acc)
25
26 # ---- Validation ----
27 model.eval()
28 val_loss, val_correct, val_count = 0.0, 0, 0
29
30 with torch.no_grad():
31     for x, y in val_loader:
32         x, y = x.to(device), y.to(device)
33         logits = model(x)
34         loss = criterion(logits, y)
35
36         val_loss += loss.item() * x.size(0)
37         val_correct += (logits.argmax(1) == y).sum().item()
38         val_count += x.size(0)
39
40 val_loss /= val_count
41 val_acc = val_correct / val_count
42 val_acc_history.append(val_acc)
43
44 print(f"Epoch {epoch}/{epochs} | "
45       f"Train Loss: {train_loss:.4f} Acc: {train_acc:.4f} | "
46       f"Val Loss: {val_loss:.4f} Acc: {val_acc:.4f}")
47

```



NameError Traceback (most recent call last)

[/tmp/ipython-input-2502302202.py](#) in <cell line: 0>()

```

29
30     with torch.no_grad():
---> 31         for x, y in val_loader:
32             x, y = x.to(device), y.to(device)
33             logits = model(x)

```

NameError: name 'val_loader' is not defined

Next steps: [Explain error](#)

```

1 import matplotlib.pyplot as plt
2
3 plt.figure(figsize=(8,5))
4 plt.plot(train_acc_history, label="Train Accuracy")
5 plt.plot(val_acc_history, label="Validation Accuracy")
6 plt.xlabel("Epoch")
7 plt.ylabel("Accuracy")
8 plt.title("Training vs Validation Accuracy")

```

```
9 plt.legend()
10 plt.grid(True)
11 plt.show()
12
```

...

```
1 # مسار مجلد test
2 test_dir = '/content/drive/MyDrive/Datasets/FER-2013'
3
4 # تحميل بيانات test
5 test_dataset = datasets.ImageFolder(root=test_dir, transform=transform)
6 test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
7
8 # اختبار النموذج
9 model.eval()
10 correct, total = 0, 0
11 with torch.no_grad():
12     for inputs, labels in test_loader:
13         inputs, labels = inputs.to(device), labels.to(device)
14         outputs = model(inputs)
15         _, preds = torch.max(outputs, 1)
16         total += labels.size(0)
17         correct += (preds == labels).sum().item()
18
```

```

19 print(f"✅ Test Accuracy: {100 * correct / total:.2f}%")
20

1 import os
2 from PIL import Image
3 import torch
4 import torch.nn as nn
5 import torch.optim as optim
6 from torch.utils.data import Dataset, DataLoader
7 from torchvision import transforms, models
8 import matplotlib.pyplot as plt
9
10 # 1. JAFFE Dataset class
11 class JaffeDataset(Dataset):
12     def __init__(self, root_dir, transform=None):
13         self.root_dir = root_dir
14         self.image_files = [f for f in os.listdir(root_dir) if f.endswith(".")]
15         self.transform = transform
16         self.emotion_dict = {
17             'AN': 0, 'DI': 1, 'FE': 2,
18             'HA': 3, 'NE': 4, 'SA': 5, 'SU': 6
19         }
20
21     def __len__(self):
22         return len(self.image_files)
23
24     def __getitem__(self, idx):
25         image_path = os.path.join(self.root_dir, self.image_files[idx])
26         image = Image.open(image_path).convert('L') # رمادية
27
28         if self.transform:
29             image = self.transform(image)
30
31         emotion_code = self.image_files[idx].split('.')[1][:2]
32         label = self.emotion_dict[emotion_code]
33
34         return image, label
35
36 # 2. RGB to Grayscale Conversion (تحويل من رمادي إلى التحويلات المناسبة)
37 transform = transforms.Compose([
38     transforms.Lambda(lambda img: img.convert("RGB")),
39     transforms.Resize((128, 128)),
40     transforms.ToTensor()
41 ])
42
43 # 3. Loading the data (تحميل البيانات)

```

```

44 jaffe_path = '/content/drive/MyDrive/Datasets/jaffe'
45 jaffe_dataset = JaffeDataset(jaffe_path, transform=transform)
46 jaffe_loader = DataLoader(jaffe_dataset, batch_size=8, shuffle=False)
47
48 # 4. (عدد الفئات 7 مشاعر)
49 num_classes = 7
50 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
51
52 # 5. تحميل النماذج وتعديل طبقات الإخراج
53 resnet = models.resnet18(pretrained=True)
54 resnet.fc = nn.Linear(resnet.fc.in_features, num_classes)
55 resnet = resnet.to(device)
56
57 mobilenet = models.mobilenet_v2(pretrained=True)
58 mobilenet.classifier[1] = nn.Linear(mobilenet.classifier[1].in_features, num
59 mobilenet = mobilenet.to(device)
60
61 # 6. دالة التقييم
62 def evaluate_model(model, dataloader, device):
63     model.eval()
64     correct = 0
65     total = 0
66     with torch.no_grad():
67         for inputs, labels in dataloader:
68             inputs, labels = inputs.to(device), labels.to(device)
69             outputs = model(inputs)
70             _, preds = torch.max(outputs, 1)
71             correct += (preds == labels).sum().item()
72             total += labels.size(0)
73     accuracy = 100 * correct / total
74     return accuracy
75
76 # 7. حساب الدقة لكل نموذج (قد تكون قليلة إذا لم يتم تدريبهم)
77 resnet_acc = evaluate_model(resnet, jaffe_loader, device)
78 mobilenet_acc = evaluate_model(mobilenet, jaffe_loader, device)
79
80 # 8. عرض النتائج
81 print(f"✅ ResNet18 Accuracy: {resnet_acc:.2f}%")
82 print(f"✅ MobileNetV2 Accuracy: {mobilenet_acc:.2f}%")
83
84 # 9. رسم بياني للمقارنة
85 results = {
86     'ResNet18': resnet_acc,
87     'MobileNetV2': mobilenet_acc
88 }
89

```

```

90 plt.bar(results.keys(), results.values())
91 plt.ylabel("Accuracy (%)")
92 plt.title("Model Comparison on JAFFE")
93 plt.ylim(0, 100)
94 plt.show()
95

```

```

1 import os
2 from PIL import Image
3 import torch
4 import torch.nn as nn
5 import torch.optim as optim
6 from torch.utils.data import Dataset, DataLoader
7 from torchvision import transforms, models, datasets
8 import matplotlib.pyplot as plt
9
10 # 1. JAFFE Dataset كلاس
11 class JaffeDataset(Dataset):
12     def __init__(self, root_dir, transform=None):
13         self.root_dir = root_dir
14         self.image_files = [f for f in os.listdir(root_dir) if f.endswith(".")]
15         self.transform = transform
16         self.emotion_dict = {
17             'AN': 0, 'DI': 1, 'FE': 2,
18             'HA': 3, 'NE': 4, 'SA': 5, 'SU': 6
19         }
20
21     def __len__(self):
22         return len(self.image_files)
23
24     def __getitem__(self, idx):
25         image_path = os.path.join(self.root_dir, self.image_files[idx])
26         image = Image.open(image_path).convert('L') # رمادية
27         if self.transform:
28             image = self.transform(image)
29         emotion_code = self.image_files[idx].split('.')[1][:2]
30         label = self.emotion_dict[emotion_code]
31         return image, label
32
33 # 2. إعداد التحويلات
34 transform = transforms.Compose([
35     transforms.Lambda(lambda img: img.convert("RGB")),
36     transforms.Resize((128, 128)),
37     transforms.ToTensor()
38 ])
39

```

```

40 # 3. تحميل بيانات JAFFE
41 jaffe_path = '/content/drive/MyDrive/Datasets/jaffe'
42 jaffe_dataset = JaffeDataset(jaffe_path, transform=transform)
43 jaffe_loader = DataLoader(jaffe_dataset, batch_size=8, shuffle=False)
44
45 # 4. مدرب مسبقًا + تعديل الطبقة الأخيرة ResNet18 تحميل نموذج
46 num_classes = 7
47 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
48
49 resnet = models.resnet18(pretrained=True)
50 resnet.fc = nn.Linear(resnet.fc.in_features, num_classes)
51 resnet = resnet.to(device)
52
53 # 5. دالة تقييم الدقة
54 def evaluate_model(model, dataloader, device):
55     model.eval()
56     correct, total = 0, 0
57     with torch.no_grad():
58         for inputs, labels in dataloader:
59             inputs, labels = inputs.to(device), labels.to(device)
60             outputs = model(inputs)
61             _, preds = torch.max(outputs, 1)
62             correct += (preds == labels).sum().item()
63             total += labels.size(0)
64     return 100 * correct / total
65
66 # 6. JAFFE تقييم النموذج على
67 acc_jaffe = evaluate_model(resnet, jaffe_loader, device)
68 print(f"✅ JAFFE Accuracy: {acc_jaffe:.2f}%")
69
70 # 7. FER-2013/test تقييم النموذج على
71 test_dir = '/content/drive/MyDrive/Datasets/FER-2013/test'
72 test_dataset = datasets.ImageFolder(root=test_dir, transform=transform)
73 test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
74
75 acc_fer = evaluate_model(resnet, test_loader, device)
76 print(f"✅ FER-2013 Test Accuracy: {acc_fer:.2f}%")
77
78 # 8. عرض النتائج في رسم بياني
79 results = {
80     'JAFFE': acc_jaffe,
81     'FER-2013': acc_fer
82 }
83
84 plt.bar(results.keys(), results.values())
85 plt.ylabel("Accuracy (%)")

```

```

86 plt.title("Model Accuracy Comparison")
87 plt.ylim(0, 100)
88 plt.show()
89

```

```

1 # مثال: آخر 2 بلوك في) فك تجميع بعض الطبقات فقط
2 for name, param in model.named_parameters():
3     if "layer4" in name or "fc" in name:
4         param.requires_grad = True
5     else:
6         param.requires_grad = False
7
8 # منخفض learning rate إعادة التدريب بـ
9 optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()))
10

```

```

1 transform = transforms.Compose([
2     transforms.RandomHorizontalFlip(),
3     transforms.RandomRotation(20),
4     transforms.ColorJitter(brightness=0.3, contrast=0.3),
5     transforms.Resize((128, 128)),
6     transforms.ToTensor()
7 ])
8

```

```

1 from sklearn.model_selection import train_test_split
2
3 # تقسيم يدوي للفهرس
4 indices = list(range(len(full_dataset)))
5 train_idx, val_idx = train_test_split(indices, test_size=0.2, stratify=target)
6
7 train_set = Subset(full_dataset, train_idx)
8 val_set = Subset(full_dataset, val_idx)
9
10 train_loader = DataLoader(train_set, batch_size=16, shuffle=True)
11 val_loader = DataLoader(val_set, batch_size=16, shuffle=False)
12

```

```

1 from torchvision import models
2 import torch.nn as nn
3
4 # عدد المشاعر
5 num_classes = len(full_dataset.classes) # حسب السياق test_dataset.classes أو
6

```

```

7 # تحميل النماذج المدربة مسبقًا
8 resnet = models.resnet18(pretrained=True)
9 mobilenet = models.mobilenet_v2(pretrained=True)
10
11 # تعديل الطبقة الأخيرة لتناسب مع عدد المشاعر
12 resnet.fc = nn.Linear(resnet.fc.in_features, num_classes)
13 mobilenet.classifier[1] = nn.Linear(mobilenet.classifier[1].in_features, num.
14

```

```

1 from torchvision import transforms
2
3 transform = transforms.Compose([
4     transforms.RandomHorizontalFlip(),
5     transforms.RandomRotation(20),
6     transforms.RandomResizedCrop(128, scale=(0.8, 1.0)),
7     transforms.ColorJitter(brightness=0.2, contrast=0.2),
8     transforms.ToTensor(),
9 ])
10

```

```

1 from torchvision.models import efficientnet_b0
2 model = efficientnet_b0(pretrained=True)
3 model.classifier[1] = nn.Linear(model.classifier[1].in_features, num_classes
4

```

```

1 for name, param in model.named_parameters():
2     param.requires_grad = "classifier" in name or "fc" in name
3

```

```

1 scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.
2

```

```

1 from sklearn.metrics import classification_report
2 print(classification_report(y_true, y_pred, target_names=full_dataset.classe
3

```

```

1 # تقييم النموذج على بيانات التحقق أو التدريب
2 correct = 0
3 total = 0
4 model.eval()
5
6 with torch.no_grad():
7     for inputs, labels in train_loader: # أو test_loader

```

```

8     inputs, labels = inputs.to(device), labels.to(device)
9     outputs = model(inputs)
10    _, preds = torch.max(outputs, 1)
11    correct += (preds == labels).sum().item()
12    total += labels.size(0)
13
14 accuracy = 100 * correct / total
15 print(f"✅ Final Accuracy: {accuracy:.2f}%")
16

```

```

1  إنشاء قائمتين لتخزين نتائج التدريب #
2
3  train_losses = []
4  train_accuracies = []
5
6  for epoch in range(epochs):
7      model.train()
8      running_loss = 0.0
9      correct = 0
10     total = 0
11     # من بيانات التدريب (batch) تمرير كل دفعة بيانات
12
13     for inputs, labels in train_loader:
14         inputs, labels = inputs.to(device), labels.to(device)
15         optimizer.zero_grad()
16         outputs = model(inputs)
17         loss = criterion(outputs, labels)
18         loss.backward()
19         optimizer.step()
20
21         _, preds = torch.max(outputs, 1)
22         correct += torch.sum(preds == labels).item()
23         total += labels.size(0)
24         running_loss += loss.item()
25     # حساب متوسط الخسارة والدقة لهذا الـ epoch
26
27     epoch_loss = running_loss / len(train_loader)
28     epoch_acc = correct / total
29     train_losses.append(epoch_loss)
30     train_accuracies.append(epoch_acc)
31
32     # طباعة النتائج
33
34     print(f"Epoch {epoch+1}/{epochs}, Loss: {epoch_loss:.4f}, Accuracy: {epoch_acc:.2f}%")
35

```

```

1 import matplotlib.pyplot as plt
2
3 epochs_range = range(1, epochs + 1)
4 plt.figure(figsize=(12, 5))
5
6 # Loss Curve
7 plt.subplot(1, 2, 1)
8 plt.plot(epochs_range, train_losses, marker='o')
9 plt.title('Training Loss')
10 plt.xlabel('Epoch')
11 plt.ylabel('Loss')
12
13 # Accuracy Curve
14 plt.subplot(1, 2, 2)
15 plt.plot(epochs_range, train_accuracies, marker='o')
16 plt.title('Training Accuracy')
17 plt.xlabel('Epoch')
18 plt.ylabel('Accuracy')
19
20 plt.tight_layout()
21 plt.show()
22

```

```

1 import random
2 # عينة عشوائية من 8 صور
3 indices = random.sample(range(len(subset_dataset)), 8)
4 fig, axes = plt.subplots(2, 4, figsize=(15,6))
5 # تأكد أن النموذج في وضع التقييم
6 model.eval()
7 with torch.no_grad():
8     for i, idx in enumerate(indices):
9         image, true_label = subset_dataset[idx]
10        img_input = image.unsqueeze(0).to(device)
11        output = model(img_input)
12        _, pred_label = torch.max(output, 1)
13
14        ax = axes[i // 4, i % 4]
15        ax.imshow(image.permute(1, 2, 0)) # تحويل إلى HWC
16        ax.axis('off')
17        ax.set_title(
18            f"True: {full_dataset.classes[true_label]}\nPred: {full_dataset.
19            color='green' if pred_label.item() == true_label else 'red'
20        )
21        # عرض التسمية الحقيقية والمتوقعة
22 plt.tight_layout()

```

```
23 plt.show()
24
```

```
1 # ✅ Cell 1: تحميل المكتبات الأساسية
2 import os
3 import random
4 import numpy as np
5 import pandas as pd
6 from PIL import Image
7 import torch
8 import torch.nn as nn
9 import torch.optim as optim
10 from torch.utils.data import Dataset, DataLoader, Subset
11 from torchvision import transforms, models
12 from sklearn.model_selection import StratifiedKFold
13 import matplotlib.pyplot as plt
14 from tqdm import tqdm
15 import warnings
16 warnings.filterwarnings("ignore")
17 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
18 print("Using device:", device)
19
```

```
1 # ✅ Cell 2: Augmentation + Preprocessing (متوافقة مع ViT)
2 transform = transforms.Compose([
3     transforms.Resize((224, 224)), # ✅ مطلوب ViT-B_16
4     transforms.RandomHorizontalFlip(),
5     transforms.RandomRotation(10),
6     transforms.ColorJitter(brightness=0.2, contrast=0.2),
7     transforms.ToTensor()
8 ])
9
```

```
1 # ✅ Cell 3 (مصححة): قاعدة Dataset كلاس JAFFE
2 class JaffeDataset(Dataset):
3     def __init__(self, root_dir, transform=None):
4         self.root_dir = root_dir
5         self.image_files = [f for f in os.listdir(root_dir) if f.endswith('.')]
6         self.label_map = {'AN':0, 'DI':1, 'FE':2, 'HA':3, 'NE':4, 'SA':5, 'SI':6}
7         self.labels = [self.get_label(f) for f in self.image_files]
8         self.transform = transform
9
10     def get_label(self, filename):
11         code = filename.split('.')[1][:2]
12         return self.label_map.get(code, 0)
```

```

13
14     def __len__(self):
15         return len(self.image_files)
16
17     def __getitem__(self, idx):
18         img_path = os.path.join(self.root_dir, self.image_files[idx])
19         image = Image.open(img_path).convert("RGB")
20         label = self.labels[idx]
21         if self.transform:
22             image = self.transform(image)
23         return image, label
24
25

```

1 #  Cell 4: وتعديله حسب عدد الفئات Vision Transformer تحميل نموذج

```

2 from torchvision.models import vit_b_16
3
4 def get_vit_model(num_classes):
5     model = vit_b_16(pretrained=True)
6     model.heads.head = nn.Linear(model.heads.head.in_features, num_classes)
7     return model.to(device)
8

```

1 #  Cell 5: K-Fold Cross Validation تدريب النموذج باستخدام

```

2 def train_kfold(dataset, k=5, epochs=3, batch_size=8, lr=1e-4):
3     skf = StratifiedKFold(n_splits=k, shuffle=True, random_state=42)
4     labels = dataset.labels
5     fold accuracies = []
6
7     for fold, (train_idx, val_idx) in enumerate(skf.split(np.zeros(len(label
8         print(f"\n  Fold {fold+1}/{k}")
9
10         train_subset = Subset(dataset, train_idx)
11         val_subset = Subset(dataset, val_idx)
12
13         train_loader = DataLoader(train_subset, batch_size=batch_size, shuff
14         val_loader = DataLoader(val_subset, batch_size=batch_size, shuffle=F
15
16         model = get_vit_model(num_classes=7)
17         optimizer = optim.Adam(model.parameters(), lr=lr)
18         criterion = nn.CrossEntropyLoss()
19
20         for epoch in range(epochs):
21             model.train()
22             running_loss, correct, total = 0.0, 0, 0

```

```

23
24         for inputs, labels in train_loader:
25             inputs, labels = inputs.to(device), labels.to(device)
26             optimizer.zero_grad()
27             outputs = model(inputs)
28             loss = criterion(outputs, labels)
29             loss.backward()
30             optimizer.step()
31             running_loss += loss.item()
32             _, preds = torch.max(outputs, 1)
33             correct += (preds == labels).sum().item()
34             total += labels.size(0)
35
36         train_acc = 100 * correct / total
37         print(f" Epoch {epoch+1}/{epochs} - Loss: {running_loss:.4f} - ,
38
39     # Evaluation on validation set
40     model.eval()
41     correct, total = 0, 0
42     with torch.no_grad():
43         for inputs, labels in val_loader:
44             inputs, labels = inputs.to(device), labels.to(device)
45             outputs = model(inputs)
46             _, preds = torch.max(outputs, 1)
47             correct += (preds == labels).sum().item()
48             total += labels.size(0)
49     val_acc = 100 * correct / total
50     fold accuracies.append(val_acc)
51     print(f"✅ Fold {fold+1} Validation Accuracy: {val_acc:.2f}%")
52
53     avg_acc = sum(fold accuracies) / len(fold accuracies)
54     print(f"\n🇸🇦 متوسط الدقة بعد {k}-Fold: {avg_acc:.2f}%")
55     return fold accuracies
56

```

```

1 # ✅ Cell 6: تشغيل Cross-Validation قاعدة JAFFE
2 jaffe_path = '/content/drive/MyDrive/Datasets/jaffe'
3 jaffe_dataset = JaffeDataset(jaffe_path, transform=transform)
4 fold_scores = train_kfold(jaffe_dataset, k=5, epochs=3)
5

```

```

1 # ✅ Cell CK+: لقاعدة Dataset كلاس CK+
2 class CKPlusDataset(Dataset):
3     def __init__(self, dataframe, transform=None):
4         self.df = dataframe

```

```

5         self.transform = transform
6
7     def __len__(self):
8         return len(self.df)
9
10    def __getitem__(self, idx):
11        pixels = [int(p) for p in self.df.iloc[idx]['pixels'].split()]
12        image = np.array(pixels).reshape(48, 48).astype(np.uint8)
13        image = Image.fromarray(image).convert("RGB") # تأكد من RGB
14        if self.transform:
15            image = self.transform(image)
16        label = int(self.df.iloc[idx]['emotion'])
17        return image, label
18

```

1 Start coding or generate with AI.

```

1 # ✅ Cell CK+: التحويل (transform) لـ ViT المناسب
2 transform = transforms.Compose([
3     transforms.Resize((224, 224)),
4     transforms.ToTensor()
5 ])
6

```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.