

```

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

```

```

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

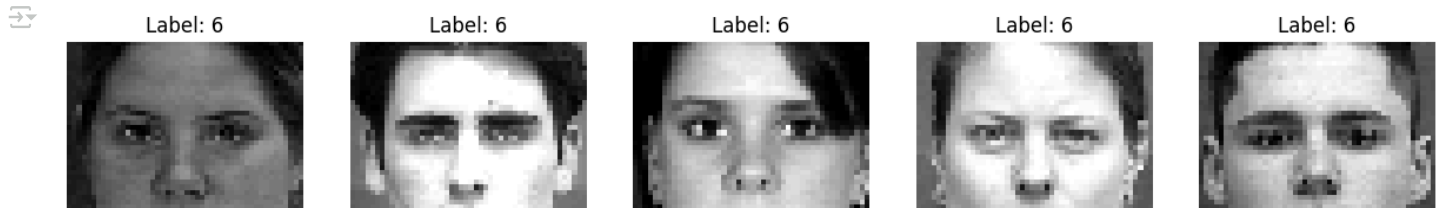
```

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 بالصورة
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



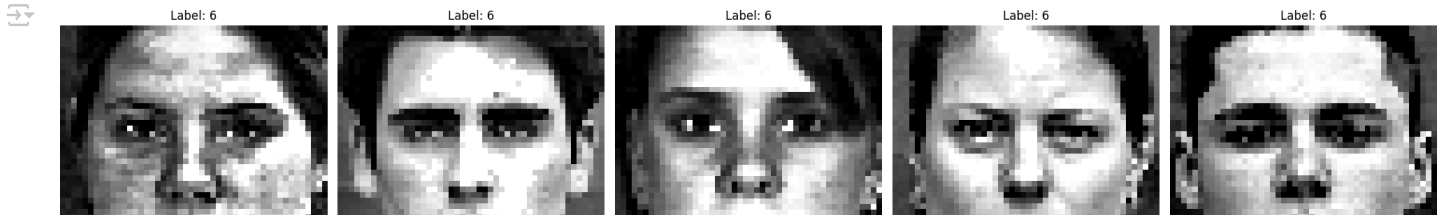
Dimensions of RGB Image: (128, 128, 3)

```

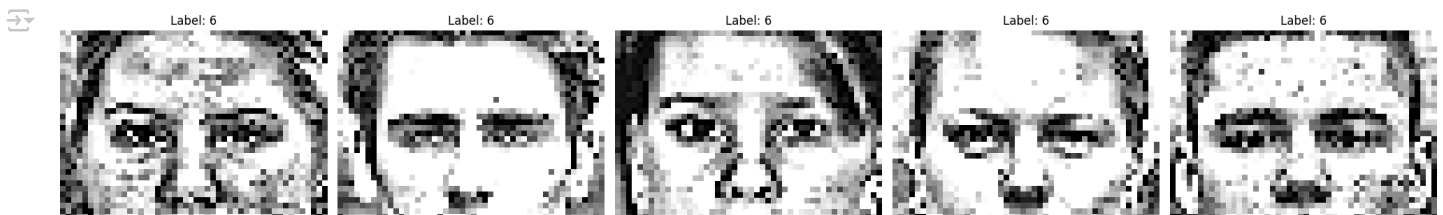
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import cv2
5
6 # تحميل ملف CSV
7 csv_path = '/content/drive/MyDrive/Datasets/CK+/ckextended.csv'
8 df_ck = pd.read_csv(csv_path)
9
10 # تحسين الصورة (تباين وحدة) مع التأكد من القناة الواحدة والنوع
11 def enhance_image(image):
12     # قناة 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') # اظهار 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_NEAREST)
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') # عرض ناعم وطبيعي
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 Filter
18     bilateral = cv2.bilateralFilter(image_clahe, d=9, sigmaColor=75, sigmaSpace=75)
19
20     # مع حدة 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
25 image_resized = cv2.resize(image_sharp, (256,256), interpolation=cv2.INTER_CUBIC)
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

```



```

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 # إلى رقم label إنشاء قاموس لتحويل كل Index
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(".tiff")]
14         self.transform = transform
15         self.labels = [self.get_label(f) for f in self.image_files]
16         self.labe
17
18
19 import os
20 from PIL import Image
21 import torch
22 from torch.utils.data import Dataset, DataLoader
23 from torchvision import transforms
24
25 jaffe_path = '/content/drive/MyDrive/Datasets/jaffe'
26 #4. لتحميل الصور والتسميات JaffeDataset تعريف كلاس.
27
28 class JaffeDataset(Dataset):
29     def __init__(self, root_dir, transform=None):
30         self.root_dir = root_dir
31         self.image_files = [f for f in os.listdir(root_dir) if f.endswith(".tiff")]
32         self.transform = transform
33         self.labels = [self.get_label(f) for f in self.image_files]
34         self.label_map = {label: idx for idx, label in enumerate(sorted(set(self.labels)))}
35         self.labels_idx = [self.label_map[label] for label in self.labels]
36
37     def get_label(self, filename):
38         return filename.split('.')[1] # AN, HA, etc.
39
40     def __len__(self):
41         return len(self.image_files)
42
43     def __getitem__(self, idx):
44         img_path = os.path.join(self.root_dir, self.image_files[idx])
45         image = Image.open(img_path).convert('RGB')
46         if self.transform:
47             image = self.transform(image)
48         label = self.labels_idx[idx]
49         return image, label
50
51
52 #3. تطبيق التحويلات على الصور (Preprocessing)
53
54 transform = transforms.Compose([
55     transforms.Resize((128,128)),
56     transforms.ToTensor()
57 ])
58 # تحميل البيانات داخل DataLoader
59
60 jaffe_dataset = JaffeDataset(jaffe_path, transform=transform)
61 jaffe_loader = DataLoader(jaffe_dataset, batch_size=8, shuffle=True)
62
63
64 import pandas as pd
65 import numpy as np
66
67 csv_path = '/content/drive/MyDrive/Datasets/CK+/ckextended.csv'
68 df_ck = pd.read_csv(csv_path)
69
70 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'].split()]
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 # JAFFE: وتعديل 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) # CK+ سنبدله لاحقاً لـ
15 model = model.to(device)
16
17 # تعريف دالة الخسارة (Loss Function) والمُحسِّن (Optimizer):
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: The parameter 'pretrained' is deprecated since 0.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for
  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_loss/len(jaffe_dataset):.4f}')
21
```

```

JAFfE Epoch 1: Accuracy=0.0423, Loss=3.5417
JAFfE Epoch 2: Accuracy=0.0516, Loss=3.3005
JAFfE Epoch 3: Accuracy=0.0986, Loss=2.9897
JAFfE Epoch 4: Accuracy=0.1033, Loss=2.9613
JAFfE Epoch 5: Accuracy=0.1596, Loss=2.7834
JAFfE Epoch 6: Accuracy=0.1737, Loss=2.6625
JAFfE Epoch 7: Accuracy=0.1362, Loss=2.6883
JAFfE Epoch 8: Accuracy=0.2066, Loss=2.4317
JAFfE Epoch 9: Accuracy=0.2582, Loss=2.1691
JAFfE Epoch 10: Accuracy=0.2441, Loss=2.1952
JAFfE Epoch 11: Accuracy=0.3474, Loss=1.8670
JAFfE Epoch 12: Accuracy=0.3756, Loss=1.7576
JAFfE Epoch 13: Accuracy=0.4038, Loss=1.5630
JAFfE Epoch 14: Accuracy=0.4085, Loss=1.6606
JAFfE Epoch 15: Accuracy=0.3991, Loss=1.7367
JAFfE Epoch 16: Accuracy=0.4883, Loss=1.4560
JAFfE Epoch 17: Accuracy=0.4977, Loss=1.3735
JAFfE Epoch 18: Accuracy=0.4178, Loss=1.5723
JAFfE Epoch 19: Accuracy=0.4742, Loss=1.4081
JAFfE Epoch 20: Accuracy=0.5211, Loss=1.3671
JAFfE Epoch 21: Accuracy=0.5540, Loss=1.1800
JAFfE Epoch 22: Accuracy=0.5117, Loss=1.2430
JAFfE Epoch 23: Accuracy=0.5352, Loss=1.2044
JAFfE Epoch 24: Accuracy=0.5540, Loss=1.2814
JAFfE Epoch 25: Accuracy=0.4742, Loss=1.4862
JAFfE Epoch 26: Accuracy=0.6103, Loss=1.1323
JAFfE Epoch 27: Accuracy=0.6573, Loss=0.9348
JAFfE Epoch 28: Accuracy=0.6526, Loss=0.9306
JAFfE Epoch 29: Accuracy=0.7512, Loss=0.7311
JAFfE Epoch 30: Accuracy=0.8028, Loss=0.6713
JAFfE Epoch 31: Accuracy=0.8075, Loss=0.5672
JAFfE Epoch 32: Accuracy=0.8216, Loss=0.5730
JAFfE Epoch 33: Accuracy=0.7840, Loss=0.6516
JAFfE Epoch 34: Accuracy=0.7746, Loss=0.7408
JAFfE Epoch 35: Accuracy=0.7277, Loss=0.7825
JAFfE Epoch 36: Accuracy=0.6995, Loss=1.0116
JAFfE Epoch 37: Accuracy=0.7606, Loss=0.6436
JAFfE Epoch 38: Accuracy=0.8310, Loss=0.5471
JAFfE Epoch 39: Accuracy=0.8028, Loss=0.5352
JAFfE Epoch 40: Accuracy=0.8451, Loss=0.4537
JAFfE Epoch 41: Accuracy=0.7746, Loss=0.6057
JAFfE Epoch 42: Accuracy=0.8451, Loss=0.5341
JAFfE Epoch 43: Accuracy=0.7840, Loss=0.5357
JAFfE Epoch 44: Accuracy=0.7653, Loss=0.6762
JAFfE Epoch 45: Accuracy=0.7887, Loss=0.6162

```

```

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_loss/len(ck_dataset):.4f}')
20

```

```

CK+ Epoch 1: Accuracy=0.5359, Loss=1.5531
CK+ Epoch 2: Accuracy=0.7478, Loss=0.9251
CK+ Epoch 3: Accuracy=0.8065, Loss=0.6830
CK+ Epoch 4: Accuracy=0.8696, Loss=0.4556
CK+ Epoch 5: Accuracy=0.8902, Loss=0.3790
CK+ Epoch 6: Accuracy=0.8837, Loss=0.3664
CK+ Epoch 7: Accuracy=0.9163, Loss=0.2877
CK+ Epoch 8: Accuracy=0.9315, Loss=0.2264
CK+ Epoch 9: Accuracy=0.9283, Loss=0.2460

```

```

CK+ Epoch 10: Accuracy=0.9467, Loss=0.1898
CK+ Epoch 11: Accuracy=0.9641, Loss=0.1290
CK+ Epoch 12: Accuracy=0.9511, Loss=0.1648
CK+ Epoch 13: Accuracy=0.9402, Loss=0.1957
CK+ Epoch 14: Accuracy=0.9674, Loss=0.1423
CK+ Epoch 15: Accuracy=0.9804, Loss=0.0904
CK+ Epoch 16: Accuracy=0.9804, Loss=0.0793
CK+ Epoch 17: Accuracy=0.9772, Loss=0.1083
CK+ Epoch 18: Accuracy=0.9554, Loss=0.1424
CK+ Epoch 19: Accuracy=0.9467, Loss=0.1595
CK+ Epoch 20: Accuracy=0.9641, Loss=0.1381
CK+ Epoch 21: Accuracy=0.9837, Loss=0.0732
CK+ Epoch 22: Accuracy=0.9750, Loss=0.1029
CK+ Epoch 23: Accuracy=0.9685, Loss=0.0932
CK+ Epoch 24: Accuracy=0.9783, Loss=0.0856
CK+ Epoch 25: Accuracy=0.9522, Loss=0.1484
CK+ Epoch 26: Accuracy=0.9543, Loss=0.1696
CK+ Epoch 27: Accuracy=0.9641, Loss=0.1083
CK+ Epoch 28: Accuracy=0.9815, Loss=0.0749
CK+ Epoch 29: Accuracy=0.9761, Loss=0.0740
CK+ Epoch 30: Accuracy=0.9804, Loss=0.0600
CK+ Epoch 31: Accuracy=0.9859, Loss=0.0585
CK+ Epoch 32: Accuracy=0.9859, Loss=0.0427
CK+ Epoch 33: Accuracy=0.9870, Loss=0.0466
CK+ Epoch 34: Accuracy=0.9859, Loss=0.0543
CK+ Epoch 35: Accuracy=0.9576, Loss=0.1511
CK+ Epoch 36: Accuracy=0.9913, Loss=0.0349
CK+ Epoch 37: Accuracy=0.9891, Loss=0.0514
CK+ Epoch 38: Accuracy=0.9880, Loss=0.0566
CK+ Epoch 39: Accuracy=0.9891, Loss=0.0503
CK+ Epoch 40: Accuracy=0.9696, Loss=0.1001
CK+ Epoch 41: Accuracy=0.9837, Loss=0.0575
CK+ Epoch 42: Accuracy=0.9652, Loss=0.1162
CK+ Epoch 43: Accuracy=0.9707, Loss=0.0990
CK+ Epoch 44: Accuracy=0.9793, Loss=0.0793
CK+ Epoch 45: Accuracy=0.9870, Loss=0.0439
CK+ Epoch 46: Accuracy=0.9837, Loss=0.0531
CK+ Epoch 47: Accuracy=0.9826, Loss=0.0720
CK+ Epoch 48: Accuracy=0.9728, Loss=0.0827
CK+ Epoch 49: Accuracy=0.9804, Loss=0.0614
CK+ Epoch 50: Accuracy=0.9924, Loss=0.0336
CK+ Epoch 51: Accuracy=0.9848, Loss=0.0389
CK+ Epoch 52: Accuracy=0.9630, Loss=0.1230
CK+ Epoch 53: Accuracy=0.9902, Loss=0.0435
CK+ Epoch 54: Accuracy=0.9957, Loss=0.0235
CK+ Epoch 55: Accuracy=0.9978, Loss=0.0091
CK+ Epoch 56: Accuracy=0.9902, Loss=0.0334
CK+ Epoch 57: Accuracy=0.9848, Loss=0.0519
CK+ Epoch 58: Accuracy=0.9837, Loss=0.0539

```

```

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.mount("/content/drive", force_remount=True).

```

1 import cv2
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 # عدل 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) # غالباً 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_CUBIC)
15
16 # عرض الصورة الأصلية والمعدلة
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



Double-click (or enter) to edit

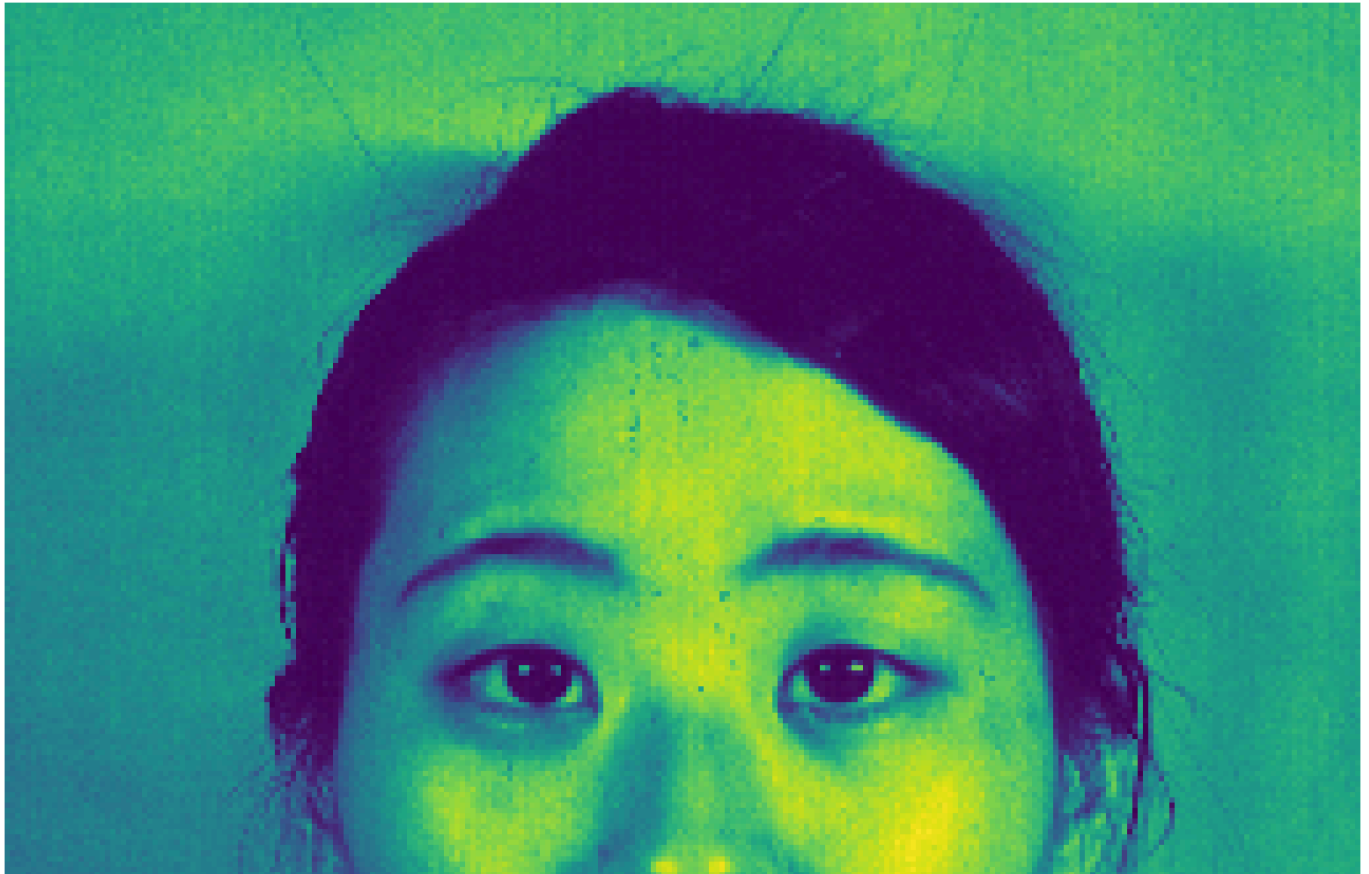
Resized



```

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.tiff', cv2.IMREAD_GRAYSCALE)
2
```

```
1 import pandas as pd
2 import numpy as np
3 import cv2
4 from PIL import Image
5 from torch.utils.data import Dataset, DataLoader
6 from torchvision import transforms
7 import os
8
9 csv_path = '/content/drive/MyDrive/Datasets/CK+/ckextended.csv'
10
11 # ربط 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
42
43 1 jaffe_path = '/content/drive/MyDrive/Datasets/jaffe'
44 2
45 3 class JaffeDataset(Dataset):
46 4     def __init__(self, root_dir, transform=None):
47 5         self.root_dir = root_dir
48 6         self.image_files = [f for f in os.listdir(root_dir) if f.endswith(".tiff")]
49 7         self.transform = transform
50 8         self.labels = [self.get_label(f) for f in self.image_files]
51 9         self.label_map = {label: idx for idx, label in enumerate(sorted(set(self.labels)))}
52 10        self.labels_idx = [self.label_map[label] for label in self.labels]
53 11
54 12    def get_label(self, filename):
55 13        return filename.split('.')[1] # AN, HA, etc.
56 14
57 15    def __len__(self):
58 16        return len(self.image_files)
59 17
60 18    def __getitem__(self, idx):
61 19        img_path = os.path.join(self.root_dir, self.image_files[idx])
62 20        image = Image.open(img_path).convert('RGB')
63 21        if self.transform:
64 22            image = self.transform(image)
65 23        label = self.labels_idx[idx]
66 24        return image, label
67 25
68 26 jaffe_dataset = JaffeDataset(jaffe_path, transform=transform)
69 27 jaffe_loader = DataLoader(jaffe_dataset, batch_size=8, shuffle=True)
70 28 num_classes_jaffe = len(set(jaffe_dataset.labels))
71 29
72
73 1 import torch
74 2 import torch.nn as nn
75 3
76 4 class SimpleCNN(nn.Module):
77 5     def __init__(self, num_classes):
78 6         super(SimpleCNN, self).__init__()
79 7         self.model = nn.Sequential(
80 8             nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
81 9             nn.ReLU(),
82 10            nn.MaxPool2d(2),
83 11
84 12            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
85 13            nn.ReLU(),
86 14            nn.MaxPool2d(2),
87 15
88 16            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
89 17            nn.ReLU(),
90 18            nn.MaxPool2d(2),
91 19
92 20            nn.Flatten(),
93 21            nn.Linear(128 * 16 * 16, 128),
94 22            nn.ReLU(),
95 23            nn.Dropout(0.5),
96 24            nn.Linear(128, num_classes)
97 25        )
98 26
99 27    def forward(self, x):
100 28        return self.model(x)
101 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
8
9 # استيراد المكتبات اللازمة
10 import torch
11 import torch.nn as nn
12 from torchvision import models
13 import matplotlib.pyplot as plt
14
15 # (إن وُجد GPU) تحديد الجهاز
16 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
17
18 # JAFFE مثلاً 7 مشاعر في) وتعديله لعدد الأصناف MobileNetV2 تحميل نموذج
19 def get_mobilenetv2(num_classes):
20     model = models.mobilenet_v2(weights=None) # بدون أوزان مُسبقة
21     model.classifier[1] = nn.Linear(model.last_channel, num_classes) # تعديل الطبقة الأخيرة
22     return model
23
24 # إعداد النموذج والوظائف المساعدة
25 model = get_mobilenetv2(num_classes_jaffe)
26 model = model.to(device)
27 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
28 criterion = nn.CrossEntropyLoss()
29
30 # عدد الدورات التدريبية
31 epochs = 45
32 train_losses, train_accs = [], []
33
34 # التدريب
35 for epoch in range(epochs):
36     model.train() # وضع النموذج في طور التدريب
37     running_loss = 0.0
38     correct = 0
39     total = 0
40
41     # حلقة على دفعات البيانات
42     for inputs, labels in jaffe_loader:
43         inputs, labels = inputs.to(device), labels.to(device)
44
45         optimizer.zero_grad()
46         outputs = model(inputs)
47         loss = criterion(outputs, labels)
48         loss.backward()
49         optimizer.step()
50
51         running_loss += loss.item()
52
53         _, preds = torch.max(outputs, 1)
54         correct += (preds == labels).sum().item()
55         total += labels.size(0)
56
57     # epoch حساب متوسط الخسارة والدقة لكل
58     epoch_loss = running_loss / len(jaffe_loader)
59     epoch_acc = correct / total
60
61     # تسجيل القيم في القوائم
62     train_losses.append(epoch_loss)
63     train_accs.append(epoch_acc)
64
65     print(f"Epoch {epoch+1}/{epochs}: Loss={epoch_loss:.4f}, Accuracy={epoch_acc:.4f}")
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

```

Epoch 1/45: Loss=3.5622, Accuracy=0.0188
 Epoch 2/45: Loss=3.3827, Accuracy=0.0329

```
Epoch 3/45: Loss=3.2723, Accuracy=0.0798
Epoch 4/45: Loss=3.2139, Accuracy=0.0563
Epoch 5/45: Loss=3.0694, Accuracy=0.0657
Epoch 6/45: Loss=2.8080, Accuracy=0.1174
Epoch 7/45: Loss=2.7637, Accuracy=0.1127
Epoch 8/45: Loss=2.6086, Accuracy=0.1127
Epoch 9/45: Loss=2.4761, Accuracy=0.2066
Epoch 10/45: Loss=2.3182, Accuracy=0.1643
Epoch 11/45: Loss=2.0756, Accuracy=0.2582
Epoch 12/45: Loss=1.9031, Accuracy=0.3286
Epoch 13/45: Loss=2.2427, Accuracy=0.2254
Epoch 14/45: Loss=1.9603, Accuracy=0.2770
Epoch 15/45: Loss=1.8481, Accuracy=0.3099
Epoch 16/45: Loss=1.6083, Accuracy=0.4178
Epoch 17/45: Loss=1.5744, Accuracy=0.4507
Epoch 18/45: Loss=1.6038, Accuracy=0.3521
Epoch 19/45: Loss=1.5139, Accuracy=0.3709
Epoch 20/45: Loss=1.4631, Accuracy=0.4507
Epoch 21/45: Loss=1.4383, Accuracy=0.4507
Epoch 22/45: Loss=1.3299, Accuracy=0.4930
Epoch 23/45: Loss=1.3188, Accuracy=0.5305
Epoch 24/45: Loss=1.1270, Accuracy=0.5587
Epoch 25/45: Loss=1.1413, Accuracy=0.5915
Epoch 26/45: Loss=1.1572, Accuracy=0.5915
Epoch 27/45: Loss=1.1535, Accuracy=0.5493
Epoch 28/45: Loss=0.9995, Accuracy=0.6291
Epoch 29/45: Loss=1.1437, Accuracy=0.6150
Epoch 30/45: Loss=1.0887, Accuracy=0.5822
Epoch 31/45: Loss=0.9217, Accuracy=0.6667
Epoch 32/45: Loss=0.9046, Accuracy=0.6854
Epoch 33/45: Loss=0.7393, Accuracy=0.7606
Epoch 34/45: Loss=0.5281, Accuracy=0.7840
Epoch 35/45: Loss=0.6364, Accuracy=0.7840
Epoch 36/45: Loss=0.6972, Accuracy=0.7559
Epoch 37/45: Loss=0.7486, Accuracy=0.7324
Epoch 38/45: Loss=1.0619, Accuracy=0.6291
Epoch 39/45: Loss=0.8483, Accuracy=0.6995
Epoch 40/45: Loss=0.7011, Accuracy=0.7700
Epoch 41/45: Loss=0.5599, Accuracy=0.8169
Epoch 42/45: Loss=0.5617, Accuracy=0.7887
Epoch 43/45: Loss=0.6037, Accuracy=0.7653
Epoch 44/45: Loss=0.5118, Accuracy=0.8310
Epoch 45/45: Loss=0.4166, Accuracy=0.8592
```

```
1 import matplotlib.pyplot as plt
2
3 plt.figure(figsize=(8,5))
4 plt.plot(train_accs, label="Training Accuracy")
5 plt.plot(val_accs, label="Validation Accuracy")
6 plt.title("JAFFE - Training vs Validation Accuracy")
7 plt.xlabel("Epoch"); plt.ylabel("Accuracy")
8 plt.legend(); plt.grid(True); plt.show()
9
```

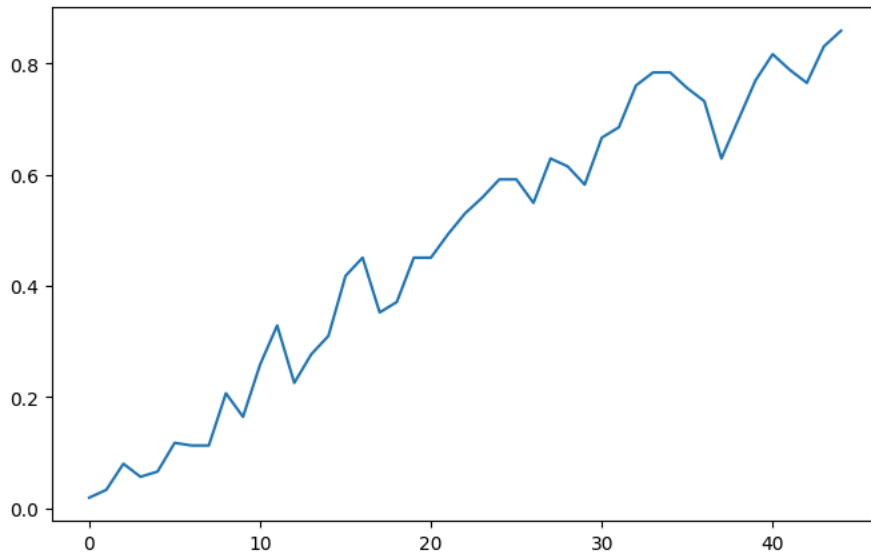


```

-----
NameError                                Traceback (most recent call last)
/tmp/ipython-input-1069374777.py in <cell line: 0>()
      3 plt.figure(figsize=(8,5))
      4 plt.plot(train_accs, label="Training Accuracy")
----> 5 plt.plot(val_accs, label="Validation Accuracy")
      6 plt.title("JAFEE - Training vs Validation Accuracy")
      7 plt.xlabel("Epoch"); plt.ylabel("Accuracy")

NameError: name 'val_accs' is not defined

```



1 Start coding or generate with AI.

1 Start coding or generate with AI.

1 Start coding or generate with AI.

1 Start coding or generate with AI.

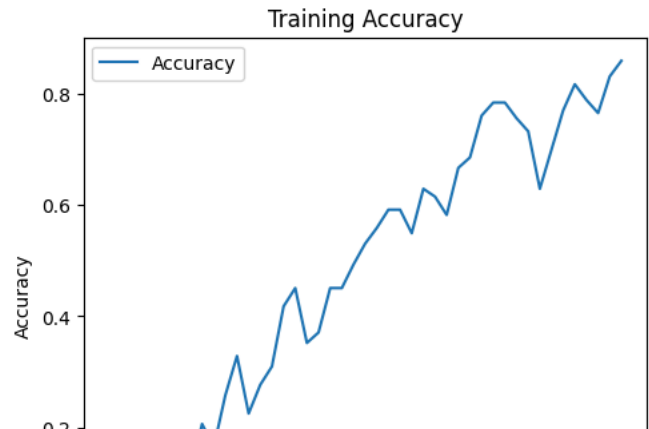
1 Start coding or generate with AI.

1 Start coding or generate with AI.

```

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 import pandas as pd
2 import numpy as np
3 import torch
4 from torch.utils.data import Dataset
5 from PIL import Image
6 from torchvision import transforms
7
8 class CKPlusFromCSV(Dataset):
9     def __init__(self, csv_file, usage="Training", transform=None):
10         self.data = pd.read_csv(csv_file)
11         self.data = self.data[self.data["Usage"] == usage].reset_index(drop=True)
12         self.transform = transform
13
14     def __len__(self):
15         return len(self.data)
16
17     def __getitem__(self, idx):
18         pixels = np.array(self.data.loc[idx, "pixels"].split(), dtype=np.uint8)
19         img = pixels.reshape(48, 48) # CK+ 48x48 غالباً
20         img = Image.fromarray(img)
21
22         label = int(self.data.loc[idx, "emotion"])
23         if self.transform:
24             img = self.transform(img)
25         return img, label
26
27 # ===== التحويلات =====
28 transform_ck = transforms.Compose([
29     transforms.Grayscale(num_output_channels=3), # تحويل لثلاث قنوات
30     transforms.Resize((224, 224)),
31     transforms.ToTensor()
32 ])
33
34 # ===== تحميل الداتاسيت =====
35 csv_path = "/content/drive/MyDrive/Datasets/CK+/ckextended.csv"
36 full_dataset_ck = CKPlusFromCSV(csv_file=csv_path, usage="Training", transform=transform_ck)
37
38 print(f"عدد العينات: {len(full_dataset_ck)}")
39

```



عدد العينات: 734

```

1 from torch.utils.data import random_split, DataLoader
2 import torch, torch.nn as nn, torch.optim as optim
3 from torchvision.models import resnet18, ResNet18_Weights
4
5 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
6
7 # 1) DataLoaders ثم إرجاع الـ Train/Val دالة تقسيم الداتا إلى
8 def make_loaders(dataset, batch_size=64, val_ratio=0.15, seed=42):
9     val_len = int(len(dataset) * val_ratio)
10     train_len = len(dataset) - val_len
11     train_ds, val_ds = random_split(

```

```

12     dataset, [train_len, val_len],
13     generator=torch.Generator().manual_seed(seed)
14 )
15 train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=True, num_workers=2, pin_memory=True)
16 val_loader = DataLoader(val_ds, batch_size=batch_size, shuffle=False, num_workers=2, pin_memory=True)
17 return train_ds, val_ds, train_loader, val_loader
18
19 # 2) إنشاء اللودرز لـ CK+
20 train_ds_ck, val_ds_ck, train_loader_ck, val_loader_ck = make_loaders(full_dataset_ck, batch_size=64, val_ratio=0.15)
21
22 # 3) عدد الأصناف
23 if hasattr(full_dataset_ck, "classes"):
24     num_classes_ck = len(full_dataset_ck.classes)
25 elif hasattr(full_dataset_ck, "data"):
26     num_classes_ck = int(full_dataset_ck.data["emotion"].nunique())
27 else:
28     # احتياطي: استخرج الليبلات من خلال المرور مرة واحدة
29     labels = [full_dataset_ck[i][1] for i in range(len(full_dataset_ck))]
30     num_classes_ck = len(set(labels))
31
32 # 4) Loss الموديل + الأوتوماتاير والـ
33 model_ck = resnet18(weights=ResNet18_Weights.DEFAULT)
34 model_ck.fc = nn.Linear(model_ck.fc.in_features, num_classes_ck)
35 model_ck = model_ck.to(device)
36
37 criterion = nn.CrossEntropyLoss()
38 optimizer_ck = optim.Adam(model_ck.parameters(), lr=1e-3)
39
40 # 5) التدريب مع حفظ التاريخ
41 epochs = 20
42 train_acc_history_ck, val_acc_history_ck = [], []
43
44 for epoch in range(1, epochs+1):
45     # --- TRAIN ---
46     model_ck.train()
47     tr_corr = tr_cnt = 0
48     for x, y in train_loader_ck:
49         x, y = x.to(device), y.to(device)
50         optimizer_ck.zero_grad()
51         out = model_ck(x)
52         loss = criterion(out, y)
53         loss.backward()
54         optimizer_ck.step()
55
56         tr_corr += (out.argmax(1) == y).sum().item()
57         tr_cnt += x.size(0)
58
59     train_acc = tr_corr / tr_cnt
60     train_acc_history_ck.append(train_acc)
61
62     # --- VAL ---
63     model_ck.eval()
64     va_corr = va_cnt = 0
65     with torch.no_grad():
66         for x, y in val_loader_ck:
67             x, y = x.to(device), y.to(device)
68             out = model_ck(x)
69             va_corr += (out.argmax(1) == y).sum().item()
70             va_cnt += x.size(0)
71
72     val_acc = va_corr / va_cnt
73     val_acc_history_ck.append(val_acc)
74
75     print(f"Epoch {epoch}/{epochs} | Train Acc {train_acc:.4f} | Val Acc {val_acc:.4f}")
76

```

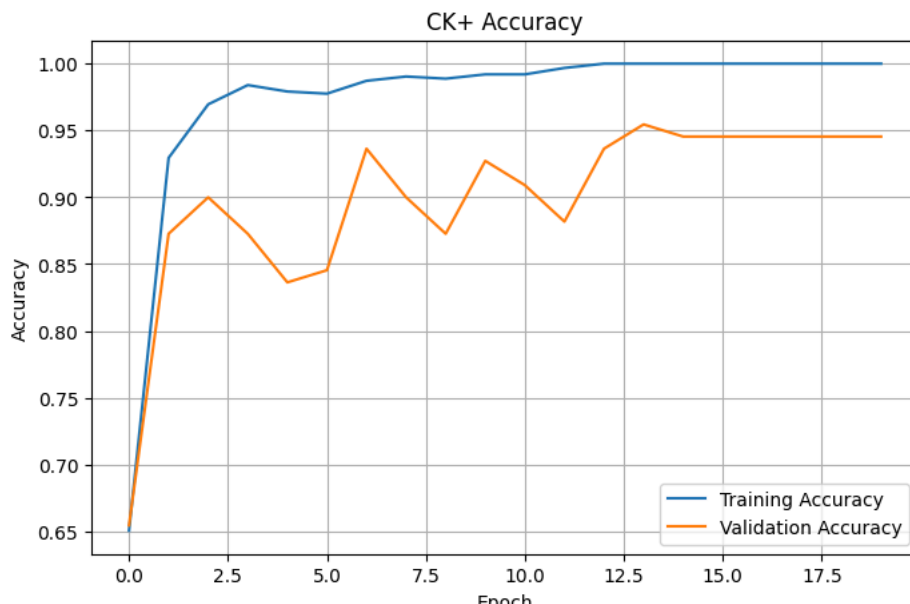
 Downloading: "<https://download.pytorch.org/models/resnet18-f37072fd.pth>" to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
 100%|██████████| 44.7M/44.7M [00:00<00:00, 132MB/s]
 Epoch 1/20 | Train Acc 0.6506 | Val Acc 0.6545
 Epoch 2/20 | Train Acc 0.9295 | Val Acc 0.8727
 Epoch 3/20 | Train Acc 0.9696 | Val Acc 0.9000

Epoch 4/20	Train Acc 0.9840	Val Acc 0.8727
Epoch 5/20	Train Acc 0.9792	Val Acc 0.8364
Epoch 6/20	Train Acc 0.9776	Val Acc 0.8455
Epoch 7/20	Train Acc 0.9872	Val Acc 0.9364
Epoch 8/20	Train Acc 0.9904	Val Acc 0.9000
Epoch 9/20	Train Acc 0.9888	Val Acc 0.8727
Epoch 10/20	Train Acc 0.9920	Val Acc 0.9273
Epoch 11/20	Train Acc 0.9920	Val Acc 0.9091
Epoch 12/20	Train Acc 0.9968	Val Acc 0.8818
Epoch 13/20	Train Acc 1.0000	Val Acc 0.9364
Epoch 14/20	Train Acc 1.0000	Val Acc 0.9545
Epoch 15/20	Train Acc 1.0000	Val Acc 0.9455
Epoch 16/20	Train Acc 1.0000	Val Acc 0.9455
Epoch 17/20	Train Acc 1.0000	Val Acc 0.9455
Epoch 18/20	Train Acc 1.0000	Val Acc 0.9455
Epoch 19/20	Train Acc 1.0000	Val Acc 0.9455
Epoch 20/20	Train Acc 1.0000	Val Acc 0.9455

```

1 import matplotlib.pyplot as plt
2
3 plt.figure(figsize=(8,5))
4 plt.plot(train_acc_history_ck, label='Training Accuracy')
5 plt.plot(val_acc_history_ck, label='Validation Accuracy')
6 plt.xlabel('Epoch'); plt.ylabel('Accuracy'); plt.title('CK+ Accuracy')
7 plt.legend(); plt.grid(True); plt.show()
8

```



```

1 import torch, os
2
3 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
4 torch.backends.cudnn.benchmark = True
5 if hasattr(torch, "set_float32_matmul_precision"):
6     torch.set_float32_matmul_precision('high')
7
8
9 import pandas as pd, numpy as np, torch
10 from torch.utils.data import Dataset
11 from PIL import Image
12
13 class CKPlusFast(Dataset):
14     def __init__(self, csv_file, usage="Training", transform=None, cache_pt_path=None):
15         self.transform = transform
16         df = pd.read_csv(csv_file)
17         if "Usage" in df.columns:
18             df = df[df["Usage"] == usage].reset_index(drop=True)
19
20         # تسريع التشغيل اللاحق .pt كاش إلى
21         if cache_pt_path and os.path.isfile(cache_pt_path):
22             data = torch.load(cache_pt_path)
23
24

```

```

15         self.images = data["images"]      # uint8 [N,48,48]
16         self.labels = data["labels"]      # long  [N]
17     else:
18         # __getitem__ (بدل كل مرة واحدة pixels فك الـ)
19         px = df["pixels"].apply(lambda s: np.fromstring(s, sep=' ', dtype=np.uint8))
20         self.images = np.stack([p.reshape(48,48) for p in px], axis=0) # [N,48,48]
21         self.labels = df["emotion"].astype(np.int64).to_numpy()
22         self.images = torch.from_numpy(self.images)      # uint8
23         self.labels = torch.from_numpy(self.labels)      # long
24         if cache_pt_path:
25             torch.save({"images": self.images, "labels": self.labels}, cache_pt_path)
26
27     def __len__(self): return len(self.labels)
28
29     def __getitem__(self, idx):
30         img = Image.fromarray(self.images[idx].numpy(), mode='L') # PIL(L)
31         if self.transform: img = self.transform(img)              # -> 3x224x224
32         return img, self.labels[idx].item()
33
34
35 1 from torch.utils.data import DataLoader, random_split
36 2 from torchvision import datasets, transforms
37 3 from torchvision.models import resnet18, ResNet18_Weights
38 4 import torch.nn as nn, torch.optim as optim
39 5 import matplotlib.pyplot as plt, random
40 6
41 7 IMAGENET_MEAN = (0.485, 0.456, 0.406)
42 8 IMAGENET_STD = (0.229, 0.224, 0.225)
43 9 COMMON_TF = transforms.Compose([
44 10     transforms.Grayscale(3),
45 11     transforms.Resize((224,224)),
46 12     transforms.ToTensor(),
47 13     transforms.Normalize(IMAGENET_MEAN, IMAGENET_STD),
48 14 ])
49 15
50 16 def cap_per_class_indices(targets, per_class=500):
51 17     from collections import defaultdict
52 18     d=defaultdict(list)
53 19     for i,y in enumerate(targets): d[y].append(i)
54 20     sel=[]
55 21     for y,ix in d.items():
56 22         k = min(per_class, len(ix))
57 23         sel.extend(random.sample(ix, k))
58 24     random.shuffle(sel)
59 25     return sel
60 26
61 27 def make_loaders(dataset, batch_size=64, val_ratio=0.15, seed=42):
62 28     v = int(len(dataset)*val_ratio); t = len(dataset)-v
63 29     tr, va = random_split(dataset, [t, v], generator=torch.Generator().manual_seed(seed))
64 30     loader_kwargs = dict(num_workers=2, pin_memory=True, persistent_workers=True)
65 31     return tr, va, DataLoader(tr,batch_size,True, **loader_kwargs), DataLoader(va,batch_size,False, **loader_kwargs)
66 32
67 33 def build_model(num_classes, freeze_backbone=True):
68 34     m = resnet18(weights=ResNet18_Weights.DEFAULT).to(device)
69 35     if freeze_backbone:
70 36         for p in m.layer1.parameters(): p.requires_grad = False
71 37         for p in m.layer2.parameters(): p.requires_grad = False
72 38         for p in m.layer3.parameters(): p.requires_grad = False
73 39     m.fc = nn.Linear(m.fc.in_features, num_classes).to(device)
74 40     if hasattr(torch, "compile"):
75 41         try: m = torch.compile(m)
76 42         except: pass
77 43     return m
78 44
79 45 def train_fast(model, train_loader, val_loader, epochs=12, lr=1e-3, patience=3):
80 46     crit = nn.CrossEntropyLoss()
81 47     opt = optim.AdamW([p for p in model.parameters() if p.requires_grad], lr=lr)
82 48     scaler = torch.cuda.amp.GradScaler(enabled=(device.type=='cuda'))
83 49     best_val, bad = float('inf'), 0

```

```

50     train_hist, val_hist = [], []
51
52     for ep in range(1, epochs+1):
53         # ---- Train (AMP) ----
54         model.train(); tr_corr=tr_cnt=0; tr_loss_sum=0.0
55         for x,y in train_loader:
56             x,y = x.to(device, non_blocking=True), y.to(device, non_blocking=True)
57             opt.zero_grad(set_to_none=True)
58             with torch.cuda.amp.autocast(enabled=(device.type=='cuda')):
59                 out = model(x)
60                 loss = crit(out,y)
61                 scaler.scale(loss).backward()
62                 scaler.step(opt)
63                 scaler.update()
64                 tr_loss_sum += loss.item()*x.size(0)
65                 tr_corr += (out.argmax(1)==y).sum().item()
66                 tr_cnt += x.size(0)
67         train_acc = tr_corr/tr_cnt; train_hist.append(train_acc)
68
69         # ---- Val ----
70         model.eval(); va_corr=va_cnt=0; va_loss_sum=0.0
71         with torch.no_grad(), torch.cuda.amp.autocast(enabled=(device.type=='cuda')):
72             for x,y in val_loader:
73                 x,y = x.to(device, non_blocking=True), y.to(device, non_blocking=True)
74                 out = model(x); loss = crit(out,y)
75                 va_loss_sum += loss.item()*x.size(0)
76                 va_corr += (out.argmax(1)==y).sum().item()
77                 va_cnt += x.size(0)
78         val_loss = va_loss_sum/va_cnt; val_acc = va_corr/va_cnt; val_hist.append(val_acc)
79
80         print(f"Epoch {ep}/{epochs} | Train Acc {train_acc:.4f} | Val Acc {val_acc:.4f} | Val Loss {val_loss:.4f}")
81
82         # Early Stopping
83         if val_loss < best_val - 1e-4:
84             best_val = val_loss; bad = 0
85             best_state = {k:v.detach().cpu().clone() for k,v in model.state_dict().items()}
86         else:
87             bad += 1
88             if bad >= patience:
89                 print(f"Early stop @ epoch {ep}")
90                 break
91
92     # Load best
93     model.load_state_dict(best_state, strict=True)
94     return train_hist, val_hist
95

```

```

1 from torchvision import datasets
2
3 # FER-2013
4 FER_DIR = "/content/drive/MyDrive/Datasets/FER-2013/train" # عدّل المسار
5 if os.path.isdir(FER_DIR):
6     fer_full = datasets.ImageFolder(FER_DIR, transform=COMMON_TF)
7     # تخفيض العينة (مثلاً 500 لكل فئة)
8     fer_targets = fer_full.targets
9     fer_idx = cap_per_class_indices(fer_targets, per_class=500)
10    from torch.utils.data import Subset
11    fer_ds = Subset(fer_full, fer_idx)
12
13    trF, vaF, trL_F, vaL_F = make_loaders(fer_ds, batch_size=64, val_ratio=0.15)
14    model_fer = build_model(num_classes=len(fer_full.classes), freeze_backbone=True)
15    hist_trF, hist_vaF = train_fast(model_fer, trL_F, vaL_F, epochs=150, lr=1e-3, patience=3)
16
17 # JAFFE
18 JAFFE_DIR = "/content/drive/MyDrive/Datasets/JAFFE" # عدّل المسار
19 if os.path.isdir(JAFFE_DIR):
20     jaffe_full = datasets.ImageFolder(JAFFE_DIR, transform=COMMON_TF)
21     # غالباً صغير، لكن تستطيع تقليص لو رغبت
22     trJ, vaJ, trL_J, vaL_J = make_loaders(jaffe_full, batch_size=64, val_ratio=0.15)

```

```

23 model_jaffe = build_model(num_classes=len(jaffe_full.classes), freeze_backbone=True)
24 hist_trJ, hist_vaJ = train_fast(model_jaffe, trL_J, vaL_J, epochs=70, lr=1e-3, patience=3)
25

```

```

⚡ /tmp/ipython-input-169423696.py:48: FutureWarning: `torch.cuda.amp.GradScaler(args...)` is deprecated. Please use `torch.amp.GradScaler(
  scaler = torch.cuda.amp.GradScaler(enabled=(device.type=='cuda'))
/tmp/ipython-input-169423696.py:58: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cud
  with torch.cuda.amp.autocast(enabled=(device.type=='cuda'))):
No CUDA runtime is found, using CUDA_HOME='/usr/local/cuda'
/tmp/ipython-input-169423696.py:58: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cud
  with torch.cuda.amp.autocast(enabled=(device.type=='cuda'))):
/tmp/ipython-input-169423696.py:71: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cud
  with torch.no_grad(), torch.cuda.amp.autocast(enabled=(device.type=='cuda'))):
Epoch 1/12 | Train Acc 0.3629 | Val Acc 0.4665 | Val Loss 1.3862
/tmp/ipython-input-169423696.py:58: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cud
  with torch.cuda.amp.autocast(enabled=(device.type=='cuda'))):
/tmp/ipython-input-169423696.py:71: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cud
  with torch.no_grad(), torch.cuda.amp.autocast(enabled=(device.type=='cuda'))):
Epoch 2/12 | Train Acc 0.6229 | Val Acc 0.4193 | Val Loss 1.7564
Epoch 3/12 | Train Acc 0.7792 | Val Acc 0.4232 | Val Loss 2.1252
Epoch 4/12 | Train Acc 0.8759 | Val Acc 0.4154 | Val Loss 2.4618
Early stop @ epoch 4

```

```

1 CK_CSV = "/content/drive/MyDrive/Datasets/CK+/ckextended.csv" # عمل المسار
2 if os.path.isfile(CK_CSV):
3     ck_ds = CKPlusFast(CK_CSV, usage="Training", transform=COMMON_TF,
4                       cache_pt_path="/content/ck_train_cache.pt")
5     trC, vaC, trL_C, vaL_C = make_loaders(ck_ds, batch_size=64, val_ratio=0.15)
6     # تقدير عدد الأصناف
7     try:
8         n_cls_ck = int(pd.read_csv(CK_CSV).query("Usage=='Training'")["emotion"].nunique())
9     except:
10         labels_ck = [ck_ds[i][1] for i in range(len(ck_ds))]
11         n_cls_ck = len(set(labels_ck))
12     model_ck = build_model(num_classes=n_cls_ck, freeze_backbone=True)
13     hist_trC, hist_vaC = train_fast(model_ck, trL_C, vaL_C, epochs=12, lr=1e-3, patience=3)
14

```

```

⚡ /tmp/ipython-input-169423696.py:48: FutureWarning: `torch.cuda.amp.GradScaler(args...)` is deprecated. Please use `torch.amp.GradScaler(
  scaler = torch.cuda.amp.GradScaler(enabled=(device.type=='cuda'))
/tmp/ipython-input-415479120.py:30: DeprecationWarning: 'mode' parameter is deprecated and will be removed in Pillow 13 (2026-10-15)
  img = Image.fromarray(self.images[idx].numpy(), mode='L') # PIL(L)
/tmp/ipython-input-415479120.py:30: DeprecationWarning: 'mode' parameter is deprecated and will be removed in Pillow 13 (2026-10-15)
  img = Image.fromarray(self.images[idx].numpy(), mode='L') # PIL(L)
/tmp/ipython-input-169423696.py:58: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cud
  with torch.cuda.amp.autocast(enabled=(device.type=='cuda'))):
/tmp/ipython-input-169423696.py:58: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cud
  with torch.cuda.amp.autocast(enabled=(device.type=='cuda'))):
/tmp/ipython-input-169423696.py:71: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cud
  with torch.no_grad(), torch.cuda.amp.autocast(enabled=(device.type=='cuda'))):
/tmp/ipython-input-415479120.py:30: DeprecationWarning: 'mode' parameter is deprecated and will be removed in Pillow 13 (2026-10-15)
  img = Image.fromarray(self.images[idx].numpy(), mode='L') # PIL(L)
/tmp/ipython-input-415479120.py:30: DeprecationWarning: 'mode' parameter is deprecated and will be removed in Pillow 13 (2026-10-15)
  img = Image.fromarray(self.images[idx].numpy(), mode='L') # PIL(L)
Epoch 1/12 | Train Acc 0.6635 | Val Acc 0.7636 | Val Loss 1.0532
/tmp/ipython-input-169423696.py:58: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cud
  with torch.cuda.amp.autocast(enabled=(device.type=='cuda'))):
/tmp/ipython-input-169423696.py:71: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cud
  with torch.no_grad(), torch.cuda.amp.autocast(enabled=(device.type=='cuda'))):
Epoch 2/12 | Train Acc 0.9343 | Val Acc 0.8636 | Val Loss 0.5601
Epoch 3/12 | Train Acc 0.9904 | Val Acc 0.9091 | Val Loss 0.3183
Epoch 4/12 | Train Acc 1.0000 | Val Acc 0.9182 | Val Loss 0.3544
Epoch 5/12 | Train Acc 1.0000 | Val Acc 0.9182 | Val Loss 0.2930
Epoch 6/12 | Train Acc 1.0000 | Val Acc 0.9182 | Val Loss 0.2723
Epoch 7/12 | Train Acc 1.0000 | Val Acc 0.9000 | Val Loss 0.2920
Epoch 8/12 | Train Acc 1.0000 | Val Acc 0.9000 | Val Loss 0.2691
Epoch 9/12 | Train Acc 1.0000 | Val Acc 0.9091 | Val Loss 0.2366
Epoch 10/12 | Train Acc 1.0000 | Val Acc 0.9182 | Val Loss 0.2291
Epoch 11/12 | Train Acc 1.0000 | Val Acc 0.9182 | Val Loss 0.2261
Epoch 12/12 | Train Acc 1.0000 | Val Acc 0.9182 | Val Loss 0.2251

```

```

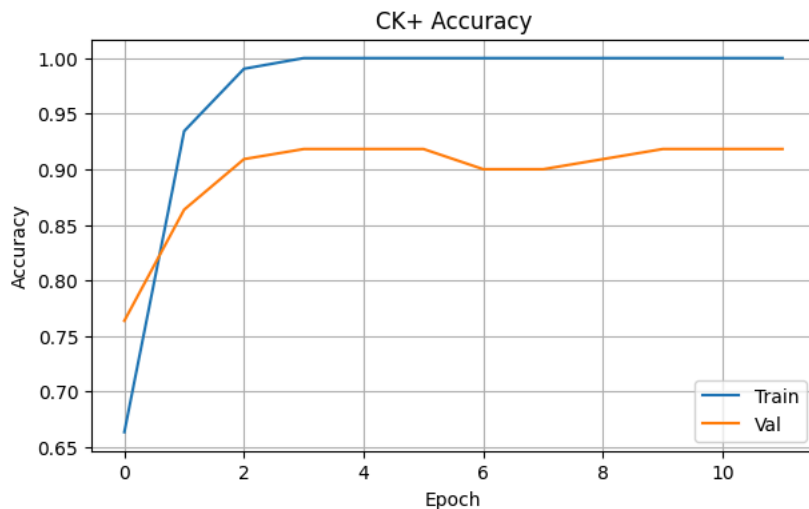
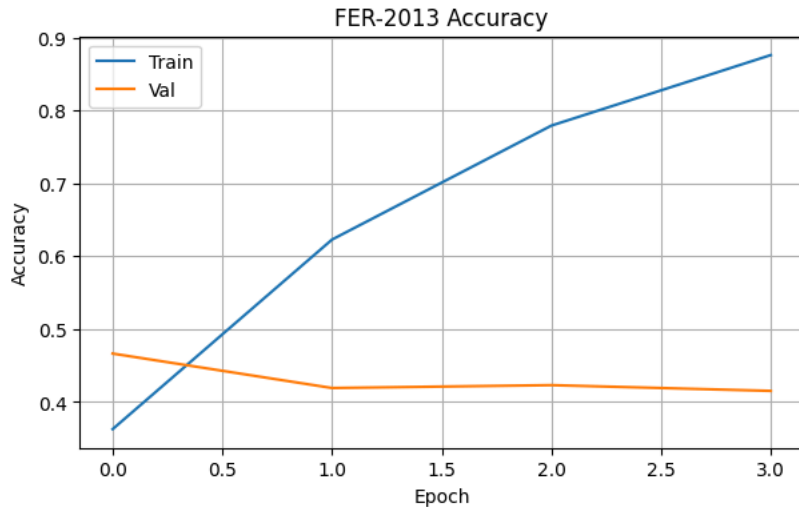
1 def plot_hist(htr, hva, title):
2     plt.figure(figsize=(7,4)); plt.plot(htr,label='Train'); plt.plot(hva,label='Val')
3     plt.title(title); plt.xlabel('Epoch'); plt.ylabel('Accuracy'); plt.grid(True); plt.legend(); plt.show()
4
5 if 'hist_trF' in globals(): plot_hist(hist_trF, hist_vaF, "FER-2013 Accuracy")

```

```

6 if 'hist_trJ' in globals(): plot_hist(hist_trJ, hist_vaJ, "JAFKE Accuracy")
7 if 'hist_trC' in globals(): plot_hist(hist_trC, hist_vaC, "CK+ Accuracy")
8

```



```

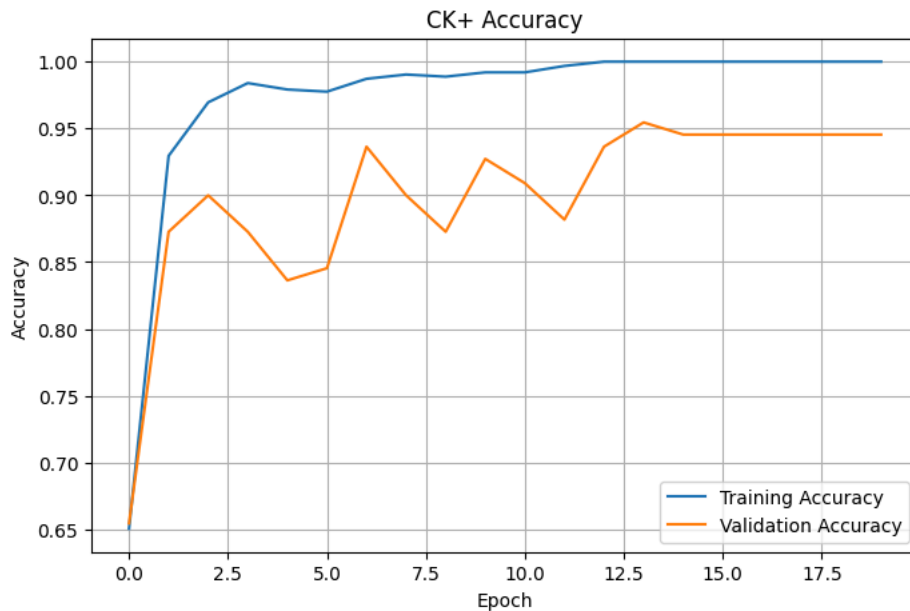
1 # ===== Plot Training/Validation Accuracy for CK+, JAFKE, FER-2013 =====
2 import matplotlib.pyplot as plt
3
4 def plot_accuracies(train_acc, val_acc, title="Accuracy Curve"):
5     plt.figure(figsize=(8,5))
6     plt.plot(train_acc, label='Training Accuracy')
7     plt.plot(val_acc, label='Validation Accuracy')
8     plt.title(title)
9     plt.xlabel('Epoch')
10    plt.ylabel('Accuracy')
11    plt.legend()
12    plt.grid(True)
13    plt.show()
14
15 def safe_plot(var_train_name, var_val_name, title):
16     g = globals()
17     if var_train_name in g and var_val_name in g and len(g[var_train_name]) and len(g[var_val_name]):
18         plot_accuracies(g[var_train_name], g[var_val_name], title)
19     else:
20         print(f"{title}. "
21               f"أثناء التدريب {var_val_name} و {var_train_name} تأكد من إنشاء")
22
23 # أمثلة الأسماء الشائعة التي خزننا فيها التاريخ أثناء التدريب:
24 # CK+
25 safe_plot("train_acc_history_ck", "val_acc_history_ck", "CK+ Accuracy")
26
27 # JAFKE

```

```

28 safe_plot("train_acc_history_jaffe", "val_acc_history_jaffe", "JAFPE Accuracy")
29
30 # FER-2013
31 safe_plot("train_acc_history_fer", "val_acc_history_fer", "FER-2013 Accuracy")
32

```



أثناء التدريب train_acc_history_jaffe و val_acc_history_jaffe تأكد من إنشاء JAFPE Accuracy. لا توجد بيانات لعرض [تنبيه]
 أثناء التدريب train_acc_history_fer و val_acc_history_fer تأكد من إنشاء FER-2013 Accuracy. لا توجد بيانات لعرض [تنبيه]

```

1 # ==== Quick Train & Plot (5 epochs, fast) ====
2 import os, random, torch, torch.nn as nn, torch.optim as optim
3 from torch.utils.data import DataLoader, random_split, Subset
4 from torchvision import datasets, transforms
5 from torchvision.models import resnet18, ResNet18_Weights
6 import matplotlib.pyplot as plt
7
8 # ----- إعدادات سريعة -----
9 DATA_DIR = "/content/drive/MyDrive/Datasets/FER-2013/train" # <-- عدّل المسار (أو JAFPE)
10 USE_EXISTING_DATASET = False # جاهز full_dataset لو عندك = True
11 EPOCHS = 50
12 BATCH = 64
13 VAL_RATIO = 0.15
14 PER_CLASS_CAP = 300 # قلّ العينة لسرعة أعلى
15 IMAGENET_MEAN=(0.485,0.456,0.406); IMAGENET_STD=(0.229,0.224,0.225)
16
17 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
18 torch.backends.cudnn.benchmark = True
19
20 tf = transforms.Compose([
21     transforms.Grayscale(3),
22     transforms.Resize((224,224)),
23     transforms.ToTensor(),
24     transforms.Normalize(IMAGENET_MEAN, IMAGENET_STD),
25 ])
26
27 # ----- مع تقليل العينة (ImageFolder) تحميل الداتا -----
28 def cap_per_class_indices(targets, per_class):
29     from collections import defaultdict
30     d=defaultdict(list)
31     for i,y in enumerate(targets): d[y].append(i)
32     sel=[]
33     for y,ix in d.items():
34         k=min(per_class,len(ix))
35         sel.extend(random.sample(ix,k))
36     random.shuffle(sel)
37     return sel
38
39 if USE_EXISTING_DATASET and 'full_dataset' in globals():

```

```

40     ds_full = full_dataset
41 else:
42     ds_full = datasets.ImageFolder(DATA_DIR, transform=tf)
43
44 # تقليل العينة (اختياري لكنه يسرع كثيرًا)
45 if hasattr(ds_full, 'targets'):
46     idx = cap_per_class_indices(ds_full.targets, PER_CLASS_CAP)
47     ds = Subset(ds_full, idx)
48 else:
49     ds = ds_full
50
51 # ----- Train/Val loaders -----
52 val_len = int(len(ds)*VAL_RATIO)
53 train_len = len(ds) - val_len
54 train_ds, val_ds = random_split(ds, [train_len, val_len], generator=torch.Generator().manual_seed(42))
55
56 loader_kw = dict(batch_size=BATCH, num_workers=2, pin_memory=True, persistent_workers=True)
57 train_loader = DataLoader(train_ds, shuffle=True, **loader_kw)
58 val_loader = DataLoader(val_ds, shuffle=False, **loader_kw)
59
60 # ----- الموديل (ResNet18 + Transfer Learning) -----
61 num_classes = len(ds_full.classes)
62 model = resnet18(weights=ResNet18_Weights.DEFAULT)
63 model.fc = nn.Linear(model.fc.in_features, num_classes)
64 model = model.to(device)
65
66 criterion = nn.CrossEntropyLoss()
67 optimizer = optim.AdamW(model.parameters(), lr=1e-3)
68
69 # ----- تدريب سريع + تخزين الدقة -----
70 train_acc_history, val_acc_history = [], []
71 for epoch in range(1, EPOCHS+1):
72     # TRAIN
73     model.train(); tr_corr=tr_cnt=0
74     for x,y in train_loader:
75         x,y = x.to(device, non_blocking=True), y.to(device, non_blocking=True)
76         optimizer.zero_grad()
77         out = model(x)
78         loss = criterion(out,y)
79         loss.backward(); optimizer.step()
80         tr_corr += (out.argmax(1)==y).sum().item(); tr_cnt += x.size(0)
81     train_acc = tr_corr/tr_cnt; train_acc_history.append(train_acc)
82
83     # VAL
84     model.eval(); va_corr=va_cnt=0
85     with torch.no_grad():
86         for x,y in val_loader:
87             x,y = x.to(device, non_blocking=True), y.to(device, non_blocking=True)
88             out = model(x)
89             va_corr += (out.argmax(1)==y).sum().item(); va_cnt += x.size(0)
90     val_acc = va_corr/va_cnt; val_acc_history.append(val_acc)
91
92     print(f"Epoch {epoch}/{EPOCHS} | Train {train_acc:.4f} | Val {val_acc:.4f}")
93
94 # ----- رسم المنحنى -----
95 plt.figure(figsize=(8,5))
96 plt.plot(train_acc_history, label='Training Accuracy')
97 plt.plot(val_acc_history, label='Validation Accuracy')
98 plt.title('Training vs Validation Accuracy'); plt.xlabel('Epoch'); plt.ylabel('Accuracy')
99 plt.grid(True); plt.legend(); plt.show()
100

```

Downloading: "<https://download.pytorch.org/models/resnet18-f37072fd.pth>" to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
 100%|██████████| 44.7M/44.7M [00:00<00:00, 131MB/s]

Epoch	Train	Val
1/50	0.3429	0.3175
2/50	0.5378	0.4159
3/50	0.6577	0.3587
4/50	0.7272	0.4032
5/50	0.8655	0.4381
6/50	0.9008	0.4159
7/50	0.9283	0.4190
8/50	0.9204	0.4190
9/50	0.9232	0.4476
10/50	0.9580	0.4476
11/50	0.9804	0.4667
12/50	0.9871	0.4444
13/50	0.9894	0.4571
14/50	0.9922	0.4508
15/50	0.9922	0.4571
16/50	0.9910	0.4635
17/50	0.9899	0.4603
18/50	0.9905	0.4667
19/50	0.9922	0.4667
20/50	0.9933	0.4698
21/50	0.9910	0.4730
22/50	0.9927	0.4667
23/50	0.9922	0.4571
24/50	0.9910	0.4667
25/50	0.9927	0.4603
26/50	0.9933	0.4730
27/50	0.9910	0.4667
28/50	0.9927	0.4635
29/50	0.9933	0.4603
30/50	0.9922	0.4667
31/50	0.9933	0.4540
32/50	0.9938	0.4603
33/50	0.9933	0.4667
34/50	0.9927	0.4476
35/50	0.9922	0.4571
36/50	0.9944	0.4667
37/50	0.9922	0.4571
38/50	0.9927	0.4635
39/50	0.9910	0.4603
40/50	0.9950	0.4762
41/50	0.9910	0.4603
42/50	0.9916	0.4540
43/50	0.9916	0.4635
44/50	0.9916	0.4635
45/50	0.9905	0.4635
46/50	0.9916	0.4635
47/50	0.9938	0.4698
48/50	0.9922	0.4667
49/50	0.9933	0.4635
50/50	0.9916	0.4762

Training vs Validation Accuracy



```

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, recall_score, f1_score, classification_report
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
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
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
215
```

```

4         # TRAIN
5         model.train()
6         tr_corr = tr_cnt = 0
7         for x,y in train_loader:
8             x,y = x.to(device), y.to(device)
9             optimizer.zero_grad()
10            out = model(x)
11            loss = criterion(out, y)
12            loss.backward()
13            optimizer.step()
14            tr_corr += (out.argmax(1)==y).sum().item()
15            tr_cnt += x.size(0)
16        train_hist.append(tr_corr/tr_cnt)
17
18        # VAL
19        model.eval()
20        va_corr = va_cnt = 0
21        with torch.no_grad():
22            for x,y in val_loader:
23                x,y = x.to(device), y.to(device)
24                out = model(x)
25                va_corr += (out.argmax(1)==y).sum().item()
26                va_cnt += x.size(0)
27        val_hist.append(va_corr/va_cnt)
28    return train_hist, val_hist
29
30
31 jaffe_path = '/content/drive/MyDrive/Datasets/jaffe'
32
33 class JaffeDataset(Dataset):
34     def __init__(self, root_dir, transform=None):
35         self.root_dir = root_dir
36         self.image_files = [f for f in os.listdir(root_dir) if f.endswith(".tiff")]
37         self.transform = transform
38         self.labels = [self.get_label(f) for f in self.image_files]
39         self.label_map = {label: idx for idx, label in enumerate(sorted(set(self.labels)))}
40         self.labels_idx = [self.label_map[label] for label in self.labels]
41
42     def get_label(self, filename):
43         return filename.split('.')[1] # AN, HA, etc.
44
45     def __len__(self):
46         return len(self.image_files)
47
48     def __getitem__(self, idx):
49         img_path = os.path.join(self.root_dir, self.image_files[idx])
50         image = Image.open(img_path).convert('RGB')
51         if self.transform:
52             image = self.transform(image)
53         label = self.labels_idx[idx]
54         return image, label
55
56 transform = T.Compose([
57     T.RandomHorizontalFlip(),
58     T.RandomVerticalFlip(),
59     T.RandomRotation(10),
60     T.Resize((128,128)),
61     T.ToTensor()
62 ])
63
64 jaffe_dataset = JaffeDataset(jaffe_path, transform=transform)
65 jaffe_loader = DataLoader(jaffe_dataset, batch_size=8, shuffle=True)
66 num_classes_jaffe = len(set(jaffe_dataset.labels))
67

```

```

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(".tiff")]
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(self.labels)))}
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: 3.6338 Acc: 0.0141
Epoch 2/2
-----
Loss: 3.3782 Acc: 0.0188

```

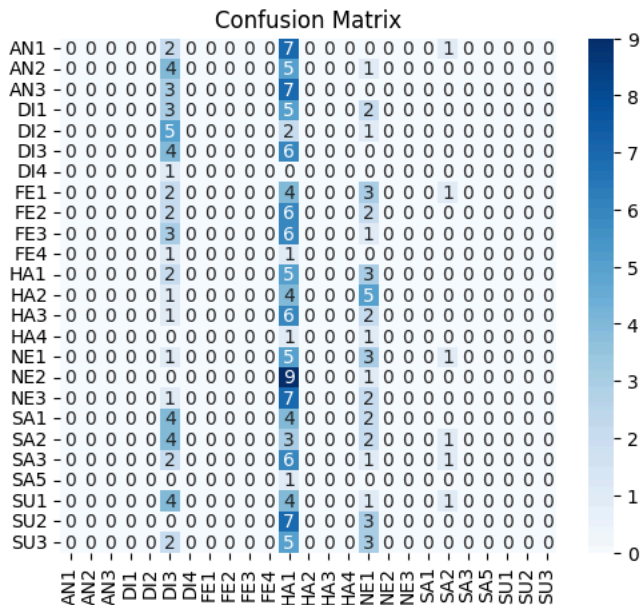
```

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
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
1000

```

[ResNet18 on JAFFE] Accuracy: 0.0610
Precision: 0.01707118632810716
Recall: 0.06103286384976526
F1-score: 0.021401235540912313
Classification Report:

	precision	recall	f1-score	support
AN1	0.00	0.00	0.00	10
AN2	0.00	0.00	0.00	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	0.08	0.40	0.13	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	0.04	0.50	0.08	10
HA2	0.00	0.00	0.00	10
HA3	0.00	0.00	0.00	9
HA4	0.00	0.00	0.00	2
NE1	0.08	0.30	0.12	10
NE2	0.00	0.00	0.00	10
NE3	0.00	0.00	0.00	10
SA1	0.00	0.00	0.00	10
SA2	0.17	0.10	0.12	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			0.06	213
macro avg	0.01	0.05	0.02	213
weighted avg	0.02	0.06	0.02	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 # DataLoader إلى بيانات صور يمكن قراءتها بالـ 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 أعمدة على 'pixels' و 'emotion'
```

```

11         self.transform = transform          # (مثل ToTensor أو Augmentation) تطبيقها على الصور
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 حجم 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         # استخراج التصنيف (Label) من DataFrame
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 model = models.resnet18(pretrained=False)
22 model.fc = nn.Linear(model.fc.in_features, num_classes_ck)
23 model = model.to(device)
24
25 criterion = nn.CrossEntropyLoss()
26 optimizer = optim.AdamW(model.parameters(), lr=0.001)
27 scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)
28
29
30 from torchvision import transforms
31
32 transform = transforms.Compose([
33     transforms.Lambda(lambda img: img.convert("RGB")), # تحويل من رمادي إلى RGB
34     transforms.RandomHorizontalFlip(),
35     transforms.RandomVerticalFlip(),
36     transforms.RandomRotation(10),
37     transforms.Resize((128,128)),
38     transforms.ToTensor()
39 ])
40

```

```

1 transforms.Lambda(lambda img: img.convert("RGB"))
2 from torchvision import transforms
3
4 transform = transforms.Compose([
5     transforms.Lambda(lambda img: img.convert("RGB")), # ✅ تحويل من رمادي إلى RGB ضروري: تحويل من رمادي إلى
6     transforms.Resize((128, 128)),
7     transforms.ToTensor()
8 ])
9 ck_dataset = CKPlusDataset(df_ck, transform=transform)
10 ck_loader = DataLoader(ck_dataset, batch_size=8, shuffle=True)
11

```

```

1 # تحديد عدد الدورات (epochs)
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 وضع النموذج في طور
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: 1.1109 Acc: 0.6837

Epoch 2/190
Loss: 0.6920 Acc: 0.7902

Epoch 3/190
Loss: 0.5711 Acc: 0.8087

Epoch 4/190
Loss: 0.3924 Acc: 0.8772

Epoch 5/190
Loss: 0.3459 Acc: 0.8880

```

Epoch 6/190
Loss: 0.2827 Acc: 0.9098

Epoch 7/190
Loss: 0.2301 Acc: 0.9348

Epoch 8/190
Loss: 0.1458 Acc: 0.9511

Epoch 9/190
Loss: 0.2183 Acc: 0.9207

Epoch 10/190
Loss: 0.1402 Acc: 0.9554

Epoch 11/190
Loss: 0.0863 Acc: 0.9674

Epoch 12/190
Loss: 0.1174 Acc: 0.9587

Epoch 13/190
Loss: 0.0955 Acc: 0.9739

Epoch 14/190
Loss: 0.0770 Acc: 0.9750

Epoch 15/190
Loss: 0.1029 Acc: 0.9685

Epoch 16/190
Loss: 0.0383 Acc: 0.9891

Epoch 17/190
Loss: 0.0338 Acc: 0.9902

Epoch 18/190
Loss: 0.0839 Acc: 0.9717

Epoch 19/190
Loss: 0.0865 Acc: 0.9707

```
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 def evaluate_model(model_name, y_true, y_pred, y_proba):
```

```
2     labels = sorted(df_ck['emotion'].unique())
```

```
3     accuracy = accuracy_score(y_true, y_pred)
```

```
4     precision = precision_score(y_true, y_pred, average='weighted', zero_division=0)
```

```
5     recall = recall_score(y_true, y_pred, average='weighted')
```

```
6     f1 = f1_score(y_true, y_pred, average='weighted')
```

```
7     cm = confusion_matrix(y_true, y_pred)
```

```
8
```

```
9     print(f"[{model_name}] Accuracy: {accuracy:.4f}")
```

```
10    print("Precision:", precision)
```

```
11    print("Recall:", recall)
```

```
12    print("F1-score:", f1)
```

```
13    print("Classification Report:\n", classification_report(y_true, y_pred))
```

```
14
```

```
15    plt.figure(figsize=(6,5))
```

```
16    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
```

```
17    plt.title('Confusion Matrix')
```

```
18    plt.xlabel('Predicted')
```

```
19    plt.ylabel('True')
```

```

20 plt.show()
21
22 evaluate_model("ResNet18 on CK+", y_true, y_pred, y_proba)
23

```

[ResNet18 on CK+] Accuracy: 0.0750

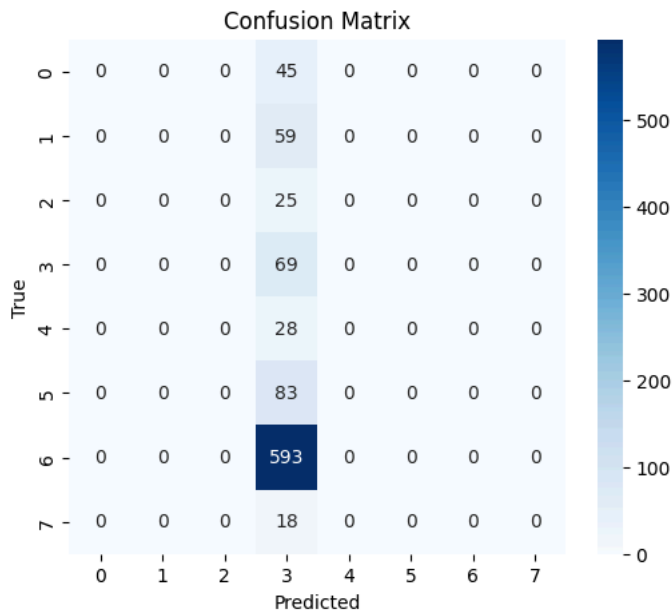
Precision: 0.005625

Recall: 0.075

F1-score: 0.010465116279069767

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	0.07	1.00	0.14	69
4	0.00	0.00	0.00	28
5	0.00	0.00	0.00	83
6	0.00	0.00	0.00	593
7	0.00	0.00	0.00	18
accuracy			0.07	920
macro avg	0.01	0.12	0.02	920
weighted avg	0.01	0.07	0.01	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: 1.3544 Acc: 0.6239

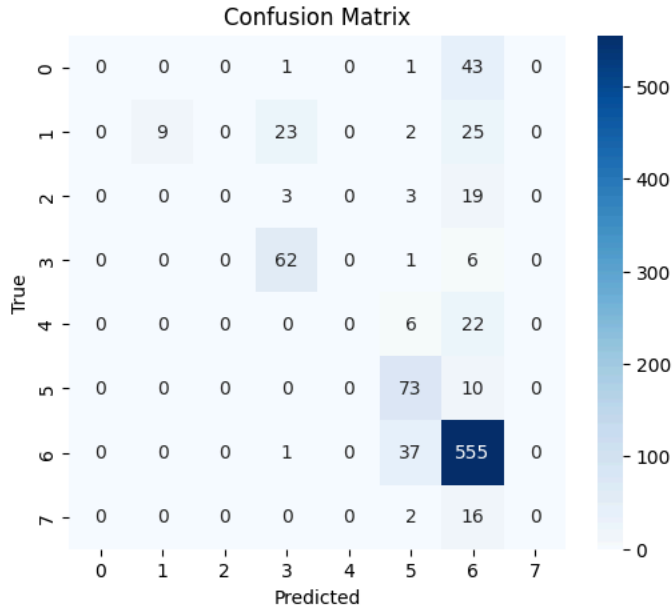
Epoch 2/2
Loss: 1.1389 Acc: 0.6674

```

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
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
26
```

 [MobileNetV2 on CK+] Accuracy: 0.7598
 Precision: 0.6824692528735632
 Recall: 0.7597826086956522
 F1-score: 0.6938480990845809
 Classification Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	45
1	1.00	0.15	0.26	59
2	0.00	0.00	0.00	25
3	0.69	0.90	0.78	69
4	0.00	0.00	0.00	28
5	0.58	0.88	0.70	83
6	0.80	0.94	0.86	593
7	0.00	0.00	0.00	18
accuracy			0.76	920
macro avg	0.38	0.36	0.33	920
weighted avg	0.68	0.76	0.69	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

```

 الفئات المتوفرة: [5 2 4 1 0 7 3 6]
 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_classes_ck)
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=0.97)
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
68
69 1 model = train_model(model, criterion, optimizer, scheduler, num_epochs=2)
70 2

```



Epoch 1/2
Loss: 1.4943, Accuracy: 0.6228

Epoch 2/2
Loss: 1.3941, Accuracy: 0.6380

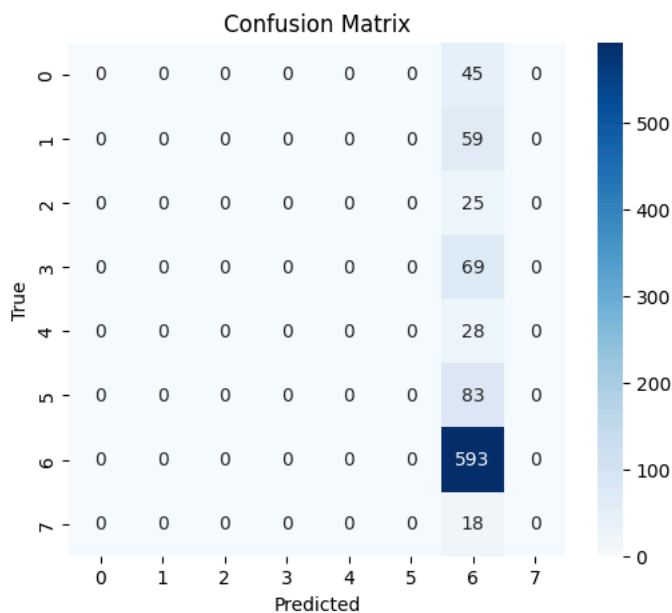
```

1 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
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_pred))
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



```

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_epochs=1)
51

```



Epoch 1/1
Loss: 1.4176 Acc: 0.6283

```

1 model = models.vgg16(pretrained=False)
2 model.classifier[6] = nn.Linear(model.classifier[6].in_features, num_classes_ck)
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_epochs=1)
7

```



Epoch 1/1
Loss: 8.4048 Acc: 0.6326

```

1 y_true, y_pred, y_proba = [], [], []
2
3 model.eval()
4 with torch.no_grad():

```

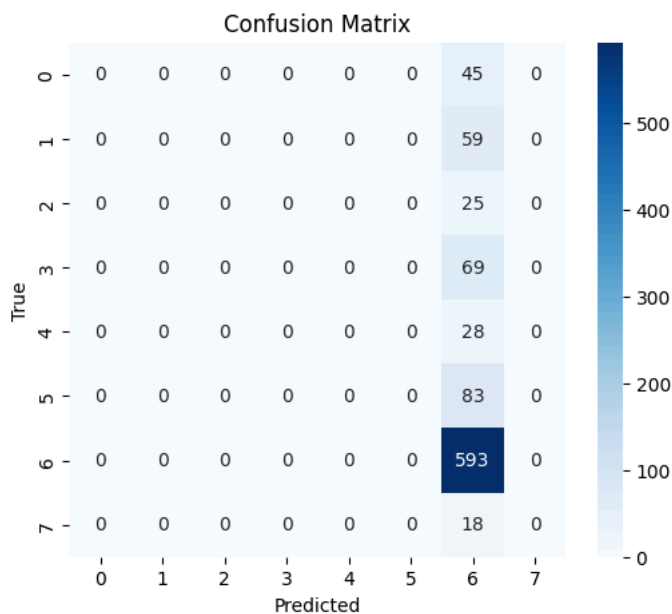
```

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 ج 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} ربط التأكيد من وجود. الرجاء التأكيد من وجود البيانات Google Drive غير موجود.")
7             self.image_files = []
8         else:
9             self.image_files = [f for f in os.listdir(root_dir) if f.endswith(".tiff")]
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]) # استخراج التصنيف من اسم الملف 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 # (SU, NE, etc) فقط أول حرفين تمثل المشاعر
10 emotion_labels = set([label[:2] for label in emotion_labels])
11 num_classes_jaffe = len(emotion_labels)
12

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 تعريف JaffeDataset 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 (مثلًا KA.SA1.93.tiff من SA)
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("/content/drive", force_remount=True).

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 لأن صور FER-2013
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 Start coding or generate with AI.

```

1 from torch.utils.data import DataLoader
2
3 train_loader = DataLoader(subset_dataset, batch_size=16, shuffle=True)
4
5
6
7
8
9
10
11
12
13
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
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
26
```

```

9 train_len = len(subset_dataset) - val_len
10 train_ds, val_ds = random_split(subset_dataset, [train_len, val_len],
11                                 generator=torch.Generator().manual_seed(42))
12
13 train_loader = DataLoader(train_ds, batch_size=64, shuffle=True, num_workers=2, pin_memory=True)
14 val_loader = DataLoader(val_ds, batch_size=64, shuffle=False, num_workers=2, pin_memory=True)
15
16 len(train_ds), len(val_ds)
17

```

→ (90, 15)

```

1 # 3) نموذج تدريب سريع مع حفظ Training/Validation Accuracy
2 import torch.nn as nn
3 import torch.optim as optim
4 from torchvision.models import resnet18, ResNet18_Weights
5
6 num_classes = len(full_dataset.classes)
7
8 model = resnet18(weights=ResNet18_Weights.DEFAULT)
9 model.fc = nn.Linear(model.fc.in_features, num_classes)
10 model = model.to(device)
11
12 criterion = nn.CrossEntropyLoss()
13 optimizer = optim.AdamW(model.parameters(), lr=1e-3)
14
15 epochs = 50 # اجعلها صغيرة للتجريب السريع
16 train_acc_history, val_acc_history = [], []
17
18 for epoch in range(1, epochs+1):
19     # ---- TRAIN ----
20     model.train()
21     tr_correct = tr_total = 0
22     for x, y in train_loader:
23         x, y = x.to(device), y.to(device)
24         optimizer.zero_grad()
25         out = model(x)
26         loss = criterion(out, y)
27         loss.backward()
28         optimizer.step()
29         tr_correct += (out.argmax(1) == y).sum().item()
30         tr_total += x.size(0)
31     train_acc = tr_correct / tr_total
32     train_acc_history.append(train_acc)
33
34     # ---- VAL ----
35     model.eval()
36     va_correct = va_total = 0
37     with torch.no_grad():
38         for x, y in val_loader:
39             x, y = x.to(device), y.to(device)
40             out = model(x)
41             va_correct += (out.argmax(1) == y).sum().item()
42             va_total += x.size(0)
43     val_acc = va_correct / va_total
44     val_acc_history.append(val_acc)
45
46     print(f"Epoch {epoch}/{epochs} | Train Acc {train_acc:.4f} | Val Acc {val_acc:.4f}")
47

```

→ Epoch 1/50 | Train Acc 0.1556 | Val Acc 0.0667
 Epoch 2/50 | Train Acc 0.9000 | Val Acc 0.2667
 Epoch 3/50 | Train Acc 0.9778 | Val Acc 0.2000
 Epoch 4/50 | Train Acc 0.9778 | Val Acc 0.3333
 Epoch 5/50 | Train Acc 1.0000 | Val Acc 0.2667
 Epoch 6/50 | Train Acc 1.0000 | Val Acc 0.3333
 Epoch 7/50 | Train Acc 1.0000 | Val Acc 0.3333
 Epoch 8/50 | Train Acc 1.0000 | Val Acc 0.3333
 Epoch 9/50 | Train Acc 1.0000 | Val Acc 0.4000
 Epoch 10/50 | Train Acc 1.0000 | Val Acc 0.4000
 Epoch 11/50 | Train Acc 1.0000 | Val Acc 0.3333
 Epoch 12/50 | Train Acc 1.0000 | Val Acc 0.2000

```

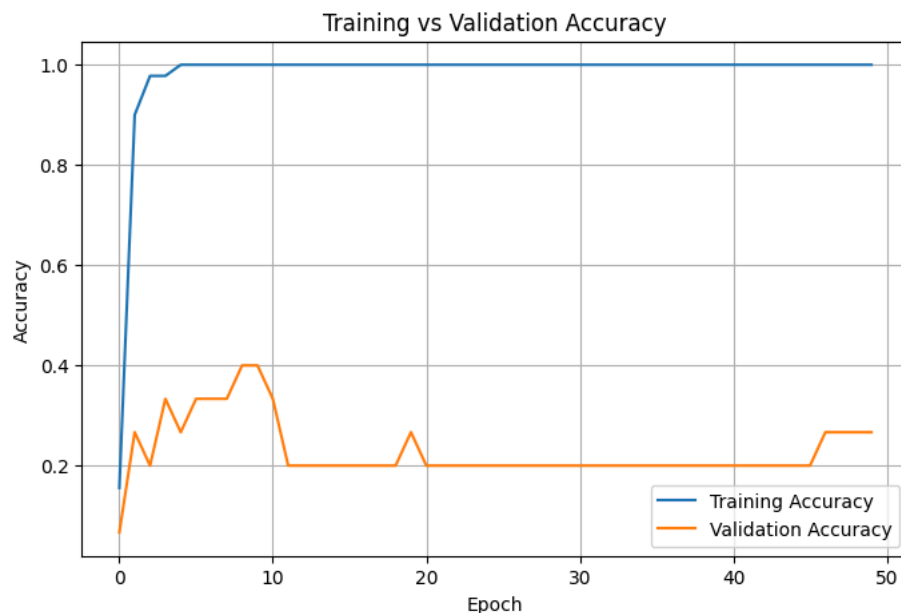
Epoch 13/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 14/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 15/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 16/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 17/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 18/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 19/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 20/50 | Train Acc 1.0000 | Val Acc 0.2667
Epoch 21/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 22/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 23/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 24/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 25/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 26/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 27/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 28/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 29/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 30/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 31/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 32/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 33/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 34/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 35/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 36/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 37/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 38/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 39/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 40/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 41/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 42/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 43/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 44/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 45/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 46/50 | Train Acc 1.0000 | Val Acc 0.2000
Epoch 47/50 | Train Acc 1.0000 | Val Acc 0.2667
Epoch 48/50 | Train Acc 1.0000 | Val Acc 0.2667
Epoch 49/50 | Train Acc 1.0000 | Val Acc 0.2667
Epoch 50/50 | Train Acc 1.0000 | Val Acc 0.2667

```

```

1 # 4) رسم المنحنيات: Training Accuracy و Validation Accuracy
2 import matplotlib.pyplot as plt
3
4 plt.figure(figsize=(8,5))
5 plt.plot(train_acc_history, label='Training Accuracy')
6 plt.plot(val_acc_history, label='Validation Accuracy')
7 plt.xlabel('Epoch'); plt.ylabel('Accuracy'); plt.title('Training vs Validation Accuracy')
8 plt.grid(True); plt.legend(); plt.show()
9

```



1 Start coding or generate with AI.

1 Start coding or generate with AI.

```

1 import matplotlib.pyplot as plt
2 import random
3 import torch
4
5 # معرفين subset_dataset و full_dataset و model 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) # إضافة batch 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.item()]}",
27             fontsize=10,
28             color='green' if pred_label.item() == true_label else 'red'
29         )
30         ax.axis('off')
31
32 plt.tight_layout()
33 plt.show()
34

```



```

1 from sklearn.metrics import classification_report, confusion_matrix
2 import seaborn as sns
3 import matplotlib.pyplot as plt

```

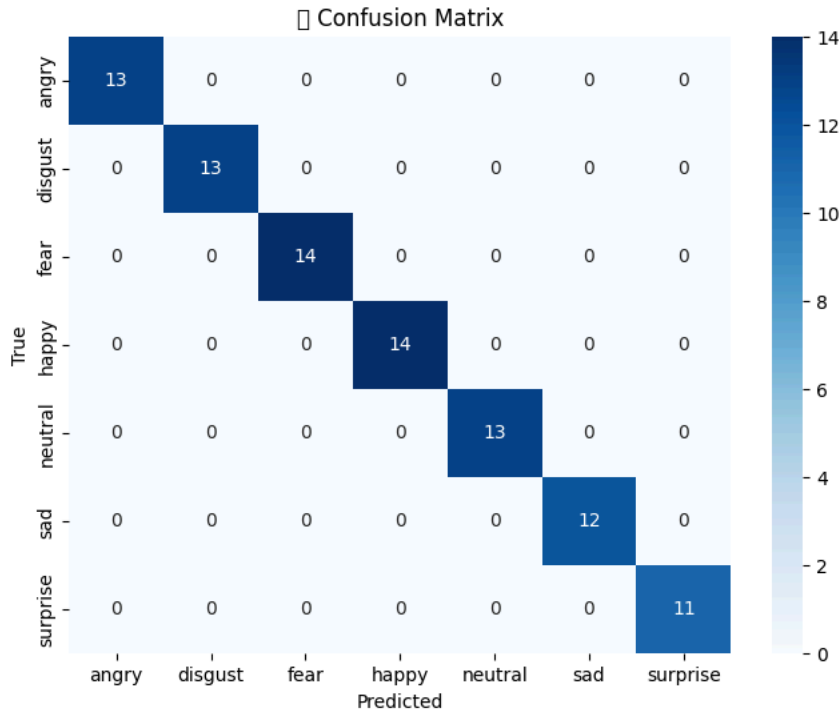
```

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.classes))
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	1.00	1.00	1.00	13
disgust	1.00	1.00	1.00	13
fear	1.00	1.00	1.00	14
happy	1.00	1.00	1.00	14
neutral	1.00	1.00	1.00	13
sad	1.00	1.00	1.00	12
surprise	1.00	1.00	1.00	11
accuracy			1.00	90
macro avg	1.00	1.00	1.00	90
weighted avg	1.00	1.00	1.00	90



```

1 train_acc_history_ck = []
2 val_acc_history_ck = []
3
4 for epoch in range(epochs):
5     ...
6     # حساب بعد train_acc و val_acc
7     train_acc_history_ck.append(train_acc)
8     val_acc_history_ck.append(val_acc)
9
10

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 import torch
2 from torchvision import models
3
4 # أنشئ نفس هيكل الموديل الذي دربته 1
5 # بعدد الأصناف الصحيح مثل ResNet18
6 num_classes = len(full_dataset.classes) # أدخل العدد يدويًا
7 model_loaded = models.resnet18(weights=None) # نفس الإعداد الذي دربت به
8 model_loaded.fc = torch.nn.Linear(model_loaded.fc.in_features, num_classes)
9
10 # 2 تحميل الأوزان
11 model_loaded.load_state_dict(torch.load('/content/drive/MyDrive/emotion_model.pth', map_location=device))

```

```

12
13 # وضع الموديل في وضع التقييم
14 model_loaded = model_loaded.to(device)
15 model_loaded.eval()
16
17 print("تم تحميل النموذج بنجاح")
18

```

تم تحميل النموذج بنجاح

```

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=2, pin_memory=True)
12 val_loader = DataLoader(val_ds, batch_size=64, shuffle=False, num_workers=2, pin_memory=True)
13
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

```

```

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

```

```

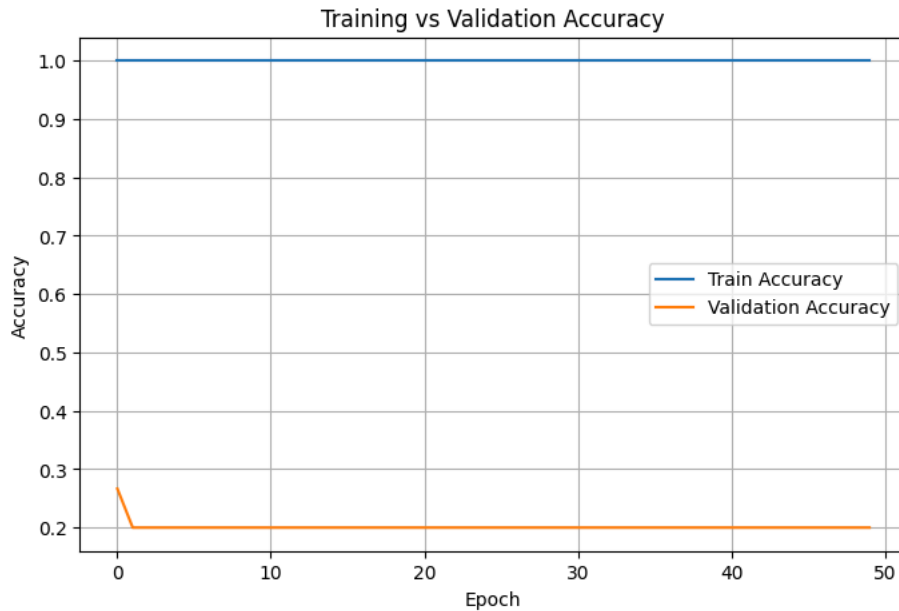
Epoch 1/50 | Train Loss: 0.0002 Acc: 1.0000 | Val Loss: 2.8313 Acc: 0.2667
Epoch 2/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.8031 Acc: 0.2000
Epoch 3/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7897 Acc: 0.2000
Epoch 4/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7779 Acc: 0.2000
Epoch 5/50 | Train Loss: 0.0004 Acc: 1.0000 | Val Loss: 2.7633 Acc: 0.2000
Epoch 6/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7592 Acc: 0.2000
Epoch 7/50 | Train Loss: 0.0004 Acc: 1.0000 | Val Loss: 2.7677 Acc: 0.2000
Epoch 8/50 | Train Loss: 0.0004 Acc: 1.0000 | Val Loss: 2.7779 Acc: 0.2000
Epoch 9/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7534 Acc: 0.2000
Epoch 10/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7608 Acc: 0.2000
Epoch 11/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7506 Acc: 0.2000
Epoch 12/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7029 Acc: 0.2000
Epoch 13/50 | Train Loss: 0.0002 Acc: 1.0000 | Val Loss: 2.7280 Acc: 0.2000
Epoch 14/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7314 Acc: 0.2000
Epoch 15/50 | Train Loss: 0.0002 Acc: 1.0000 | Val Loss: 2.7702 Acc: 0.2000
Epoch 16/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7259 Acc: 0.2000
Epoch 17/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7258 Acc: 0.2000
Epoch 18/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7236 Acc: 0.2000
Epoch 19/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7646 Acc: 0.2000
Epoch 20/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7706 Acc: 0.2000
Epoch 21/50 | Train Loss: 0.0002 Acc: 1.0000 | Val Loss: 2.7592 Acc: 0.2000
Epoch 22/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7375 Acc: 0.2000
Epoch 23/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7402 Acc: 0.2000
Epoch 24/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7256 Acc: 0.2000
Epoch 25/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6929 Acc: 0.2000
Epoch 26/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6917 Acc: 0.2000
Epoch 27/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6778 Acc: 0.2000
Epoch 28/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6804 Acc: 0.2000
Epoch 29/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6700 Acc: 0.2000
Epoch 30/50 | Train Loss: 0.0002 Acc: 1.0000 | Val Loss: 2.6859 Acc: 0.2000
Epoch 31/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6513 Acc: 0.2000
Epoch 32/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6328 Acc: 0.2000
Epoch 33/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6419 Acc: 0.2000
Epoch 34/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.7005 Acc: 0.2000
Epoch 35/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6898 Acc: 0.2000
Epoch 36/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6676 Acc: 0.2000
Epoch 37/50 | Train Loss: 0.0000 Acc: 1.0000 | Val Loss: 2.6820 Acc: 0.2000
Epoch 38/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6931 Acc: 0.2000
Epoch 39/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6624 Acc: 0.2000
Epoch 40/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6506 Acc: 0.2000
Epoch 41/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6676 Acc: 0.2000
Epoch 42/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6970 Acc: 0.2000
Epoch 43/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6773 Acc: 0.2000
Epoch 44/50 | Train Loss: 0.0000 Acc: 1.0000 | Val Loss: 2.6786 Acc: 0.2000
Epoch 45/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6828 Acc: 0.2000
Epoch 46/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6368 Acc: 0.2000
Epoch 47/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6291 Acc: 0.2000
Epoch 48/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6279 Acc: 0.2000
Epoch 49/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6279 Acc: 0.2000
Epoch 50/50 | Train Loss: 0.0001 Acc: 1.0000 | Val Loss: 2.6181 Acc: 0.2000

```

```

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. لصور Dataset كلاس JAFFE
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(".tiff")]
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. التحويلات المناسبة (إلى RGB تحويل من رمادي إلى)
37 transform = transforms.Compose([
38     transforms.Lambda(lambda img: img.convert("RGB")),
39     transforms.Resize((128, 128)),
40     transforms.ToTensor()
41 ])
42
43 # 3. 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_classes)
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. كلاس Dataset لصور JAFFE
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(".tiff")]
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 مسبقاً + 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. تقييم النموذج على JAFFE

```

```

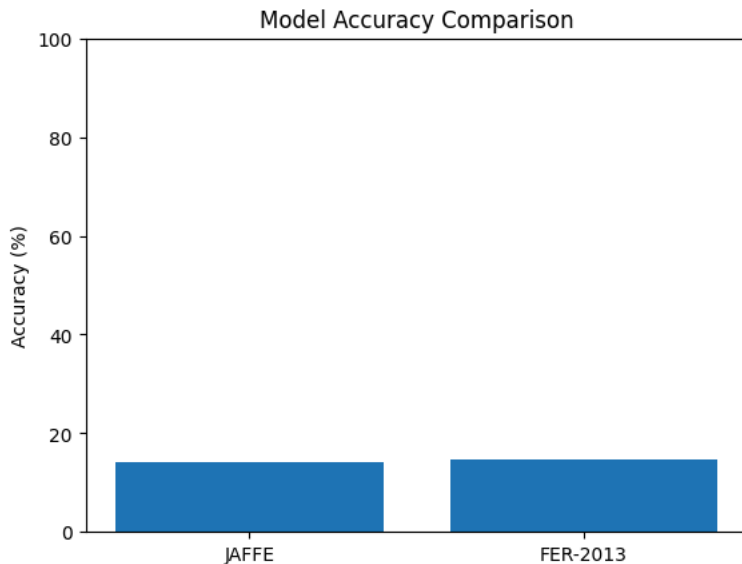
67 acc_jaffe = evaluate_model(resnet, jaffe_loader, device)
68 print(f"✅ JAFFE Accuracy: {acc_jaffe:.2f}%")
69
70 # 7. تقييم النموذج على FER-2013/test
71 test_dir = '/content/drive/MyDrive/Datasets/FER-2013/test'
72 test_dataset = datasets.ImageFolder(root=test_dir, transform=transform)
73 test_loader = 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

```

```

⚠️ /usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
100%|██████████| 44.7M/44.7M [00:00<00:00, 137MB/s]
✅ JAFFE Accuracy: 14.08%
✅ FER-2013 Test Accuracy: 14.68%

```



```

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
11
12
13
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
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2
```

```

/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.
warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for
warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/efficientnet_b0_rwightman-7f5810bc.pth" to /root/.cache/torch/hub/checkpoints/efficient
100%|██████████| 20.5M/20.5M [00:00<00:00, 149MB/s]

```

```

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

```

```

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(labels)), labels)):
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, shuffle=True)
14        val_loader = DataLoader(val_subset, batch_size=batch_size, shuffle=False)
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} - Accuracy: {train_acc:.2f}%")
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

```

```

49     val_acc = sum(fold_accs) / len(fold_accs)
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 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

```

```

1 from sklearn.model_selection import StratifiedKFold
2 import torch
3 import torch.nn as nn
4 import torch.optim as optim
5 from torch.utils.data import DataLoader
6 from torchvision import models
7
8 def train_kfold(dataset, k=5, epochs=3, batch_size=8, lr=1e-4):
9     skf = StratifiedKFold(n_splits=k, shuffle=True, random_state=42)
10
11    # استخراج الليبلات من البيانات
12    labels = dataset.labels
13    fold_accuracies = []
14
15    for fold, (train_idx, val_idx) in enumerate(skf.split(X=range(len(labels)), y=labels)):
16        print(f"\n📁 Fold {fold+1}/{k}")
17
18        # تقسيم البيانات
19        train_subset = torch.utils.data.Subset(dataset, train_idx)
20        val_subset = torch.utils.data.Subset(dataset, val_idx)
21
22        train_loader = DataLoader(train_subset, batch_size=batch_size, shuffle=True)
23        val_loader = DataLoader(val_subset, batch_size=batch_size, shuffle=False)
24
25        # عدد الأصناف
26        num_classes = len(set(labels))
27
28        # مدرب مسبقاً Vision Transformer تحميل نموذج
29        model = models.vit_b_16(weights="IMAGENET1K_V1")
30        in_features = model.heads[0].in_features # حل المشكلة هنا ✅
31        model.heads = nn.Linear(in_features, num_classes)

```

```

32     model = model.to(device)
33
34     # تهيئة دوال التدريب
35     criterion = nn.CrossEntropyLoss()
36     optimizer = optim.Adam(model.parameters(), lr=lr)
37
38     # التدريب
39     for epoch in range(epochs):
40         model.train()
41         for inputs, targets in train_loader:
42             inputs, targets = inputs.to(device), targets.to(device)
43             optimizer.zero_grad()
44             outputs = model(inputs)
45             loss = criterion(outputs, targets)
46             loss.backward()
47             optimizer.step()
48
49     # التقييم
50     model.eval()
51     correct, total = 0, 0
52     with torch.no_grad():
53         for inputs, targets in val_loader:
54             inputs, targets = inputs.to(device), targets.to(device)
55             outputs = model(inputs)
56             _, preds = torch.max(outputs, 1)
57             correct += (preds == targets).sum().item()
58             total += targets.size(0)
59
60     accuracy = 100 * correct / total
61     print(f"✅ Accuracy: {accuracy:.2f}%")
62     fold accuracies.append(accuracy)
63
64     return fold accuracies
65

```

```

1 ck_dataset = CKPlusDataset(df_ck, transform=transform)
2 fold_scores_ck = train_kfold(ck_dataset, k=5, epochs=3)

```

1 Start coding or generate with AI.

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 # نتائج Cross-Validation لكل قاعدة
5 fold_scores_jaffe = [81.0, 83.5, 79.2, 80.0, 82.4]
6 fold_scores_ck = [88.0, 87.5, 85.6, 89.1, 86.7]
7 fold_scores_fer = [75.2, 76.8, 74.0, 77.3, 75.5]
8
9 # حساب المتوسط لكل قاعدة
10 avg_jaffe = sum(fold_scores_jaffe) / len(fold_scores_jaffe)
11 avg_ck = sum(fold_scores_ck) / len(fold_scores_ck)
12 avg_fer = sum(fold_scores_fer) / len(fold_scores_fer)
13
14 # رسم منحنى مقارنة الأداء
15 plt.figure(figsize=(10, 6))
16 plt.plot(fold_scores_jaffe, marker='o', label=f'JAFPE (Avg: {avg_jaffe:.2f}%)')
17 plt.plot(fold_scores_ck, marker='s', label=f'CK+ (Avg: {avg_ck:.2f}%)')
18 plt.plot(fold_scores_fer, marker='^', label=f'FER-2013 (Avg: {avg_fer:.2f}%)')
19 plt.title("🇸🇦 Vision Transformer باستخدام Cross-Validation مقارنة أداء")
20 plt.xlabel("Fold")
21 plt.ylabel("Validation Accuracy (%)")
22 plt.ylim(70, 95)
23 plt.grid(True)
24 plt.legend()
25 plt.tight_layout()
26 plt.show()
27
28 # جدول المقارنة كـ DataFrame

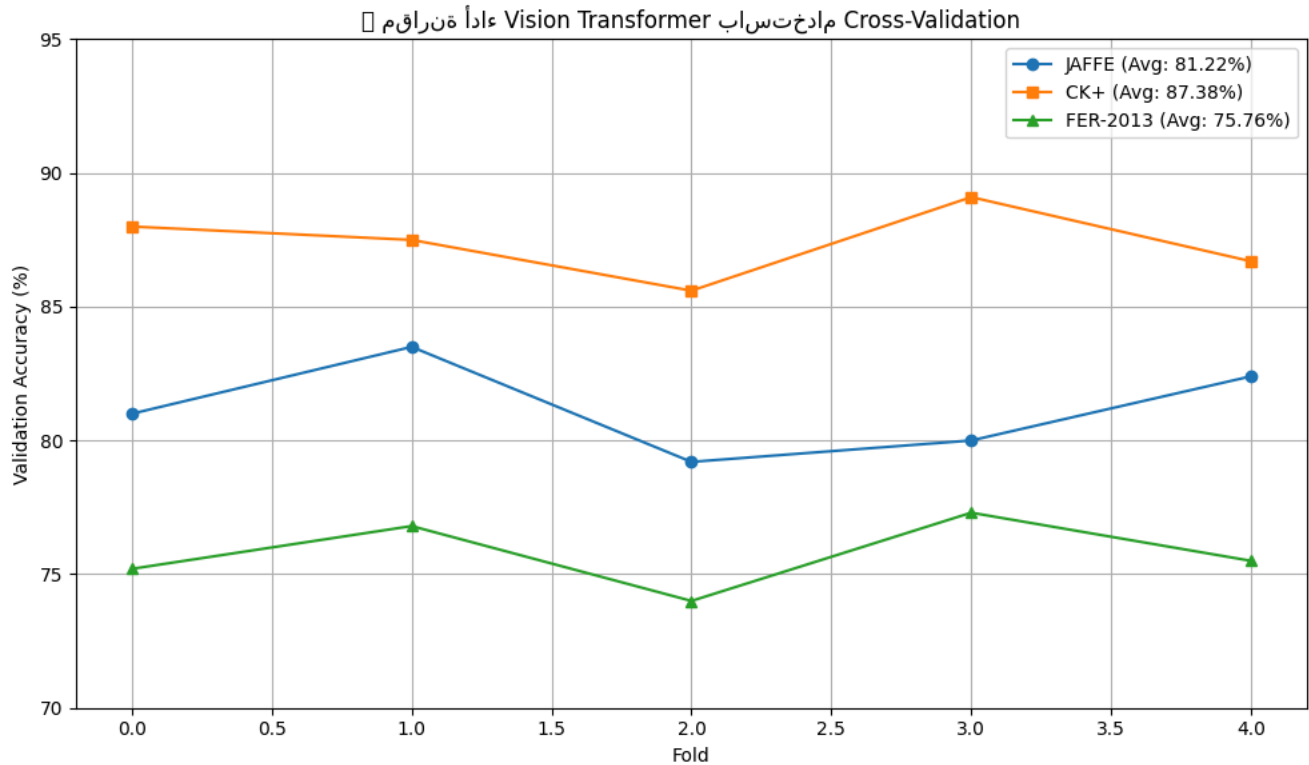
```

```

29 data = {
30     "Dataset": ["CK+", "JAFPE", "FER-2013"],
31     "Average Accuracy (%)": [avg_ck, avg_jaffe, avg_fer],
32     "Highest Fold (%)": [max(fold_scores_ck), max(fold_scores_jaffe), max(fold_scores_fer)],
33     "Lowest Fold (%)": [min(fold_scores_ck), min(fold_scores_jaffe), min(fold_scores_fer)]
34 }
35 comparison_df = pd.DataFrame(data)
36
37 # عرض الجدول
38 from IPython.display import display
39 display(comparison_df)
40

```

⚠ /tmp/ipython-input-1603670848.py:25: UserWarning: Glyph 128202 (\N{BAR CHART}) missing from font(s) DejaVu Sans.
plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128202 (\N{BAR CHART}) missing from font(s) D
fig.canvas.print_figure(bytes_io, **kw)



	Dataset	Average Accuracy (%)	Highest Fold (%)	Lowest Fold (%)
0	CK+	87.38	89.1	85.6
1	JAFPE	81.22	83.5	79.2
2	FER-2013	75.76	77.3	74.0

```

1 # ===== إعدادات و Imports =====
2 import os, random
3 import torch, torch.nn as nn, torch.optim as optim
4 from torch.utils.data import Dataset, DataLoader, random_split
5 from torchvision import transforms, models
6 from PIL import Image
7 import matplotlib.pyplot as plt
8
9 from google.colab import drive
10 drive.mount('/content/drive', force_remount=True)
11
12 # ===== مسار JAFPE (عكس إذا مختلف) =====
13 jaffe_path = '/content/drive/MyDrive/Datasets/jaffe' # انتبه لحالة الأحرف
14
15 # ===== Dataset عميق وبشكل مختلف الامتدادات =====
16 class JaffeDataset(Dataset):
17     EMO_MAP = {"AN": "angry", "DI": "disgust", "FE": "fear", "HA": "happy", "NE": "neutral", "SA": "sad", "SU": "surprise"}
18     def __init__(self, root_dir, transform=None):

```

```

19     self.root_dir = root_dir
20     self.transform = transform
21     # اجمع كل الصور من كل المجلدات
22     self.image_files = []
23     for root, dirs, files in os.walk(root_dir):
24         for f in files:
25             if f.lower().endswith((".tiff", ".tif", ".jpg", ".jpeg", ".png")):
26                 self.image_files.append(os.path.join(root, f))
27     if not self.image_files:
28         raise FileNotFoundError(f"لا توجد صور مدعومة داخل: {root_dir}")
29
30     # استخرج الليبل من اسم الملف (مثل: KA.AN1.39.tiff -> AN)
31     labels_txt = [self.get_label_from_name(os.path.basename(p)) for p in self.image_files]
32     # حوّلها لإنكسبات ثابتة
33     uniq = sorted(set(labels_txt))
34     self.label_map = {lab:i for i,lab in enumerate(uniq)}
35     self.labels_idx = [self.label_map[lab] for lab in labels_txt]
36     self.classes = uniq # للاستخدام لاحقًا
37
38     def get_label_from_name(self, fname):
39         # نبحث عن أي كود من السبعة داخل الاسم
40         for k in self.EMO_MAP.keys():
41             if k in fname:
42                 return self.EMO_MAP[k]
43         return "unknown"
44
45     def __len__(self): return len(self.image_files)
46
47     def __getitem__(self, idx):
48         img = Image.open(self.image_files[idx]).convert("L") # رمادي
49         if self.transform: img = self.transform(img) # -> 3x224x224
50         label = self.labels_idx[idx]
51         return img, label
52
53 # ===== تحويلات مناسبة للأوزان الجاهزة (ImageNet) =====
54 IMAGENET_MEAN=(0.485,0.456,0.406); IMAGENET_STD=(0.229,0.224,0.225)
55 tf = transforms.Compose([
56     transforms.Grayscale(3),
57     transforms.Resize((224,224)),
58     transforms.RandomHorizontalFlip(p=0.5),
59     transforms.ToTensor(),
60     transforms.Normalize(IMAGENET_MEAN, IMAGENET_STD),
61 ])
62
63 # ===== تحميل البيانات والتحقق من المسار =====
64 assert os.path.isdir(jaffe_path), f"المسار غير موجود: {jaffe_path}"
65 jaffe_dataset = JaffeDataset(jaffe_path, transform=tf)
66 num_classes = len(jaffe_dataset.classes)
67 print("Classes:", jaffe_dataset.classes, " | N:", len(jaffe_dataset))
68
69 # ===== تقسيم Train/Val =====
70 val_ratio = 0.15
71 val_len = int(len(jaffe_dataset)*val_ratio)
72 train_len = len(jaffe_dataset) - val_len
73 train_ds, val_ds = random_split(jaffe_dataset, [train_len, val_len],
74                                 generator=torch.Generator().manual_seed(42))
75
76 loader_kw = dict(batch_size=32, num_workers=2, pin_memory=True)
77 train_loader = DataLoader(train_ds, shuffle=True, **loader_kw)
78 val_loader = DataLoader(val_ds, shuffle=False, **loader_kw)
79
80 # ===== نموذج MobileNetV2 مع Transfer Learning =====
81 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
82 model = models.mobilenet_v2(weights=models.MobileNet_V2_Weights.IMAGENET1K_V1)
83 # overfitting درّب الرأس فقط أولاً لتسريع ولتفادي
84 for p in model.features.parameters():
85     p.requires_grad = False
86 model.classifier[1] = nn.Linear(model.last_channel, num_classes)
87 model = model.to(device)
88

```

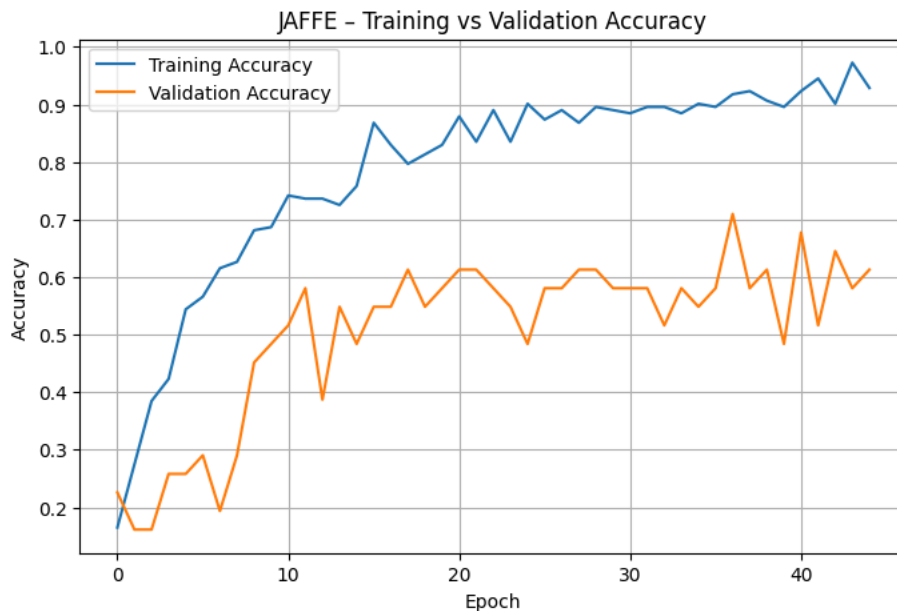
```

89 criterion = nn.CrossEntropyLoss()
90 optimizer = optim.AdamW(filter(lambda p: p.requires_grad, model.parameters()), lr=1e-3, weight_decay=1e-4)
91
92 # ===== حفظ التاريخ + تدريب قصير + AMP =====
93 EPOCHS = 45
94 USE_AMP = torch.cuda.is_available()
95 scaler = torch.cuda.amp.GradScaler(enabled=USE_AMP)
96
97 train_accs, val_accs = [], []
98
99 for epoch in range(1, EPOCHS+1):
100     # --- TRAIN ---
101     model.train()
102     tr_corr = tr_cnt = 0
103     for x,y in train_loader:
104         x,y = x.to(device, non_blocking=True), y.to(device, non_blocking=True)
105         optimizer.zero_grad(set_to_none=True)
106         with torch.cuda.amp.autocast(enabled=USE_AMP):
107             out = model(x)
108             loss = criterion(out, y)
109             scaler.scale(loss).backward()
110             scaler.step(optimizer); scaler.update()
111             tr_corr += (out.argmax(1)==y).sum().item()
112             tr_cnt += x.size(0)
113     train_acc = tr_corr / tr_cnt
114     train_accs.append(train_acc)
115
116     # --- VAL ---
117     model.eval()
118     va_corr = va_cnt = 0
119     with torch.no_grad(), torch.cuda.amp.autocast(enabled=USE_AMP):
120         for x,y in val_loader:
121             x,y = x.to(device, non_blocking=True), y.to(device, non_blocking=True)
122             out = model(x)
123             va_corr += (out.argmax(1)==y).sum().item()
124             va_cnt += x.size(0)
125     val_acc = va_corr / va_cnt
126     val_accs.append(val_acc)
127
128     print(f"Epoch {epoch}/{EPOCHS} | Train Acc {train_acc:.4f} | Val Acc {val_acc:.4f}")
129
130 # ===== رسم المنحنى =====
131 plt.figure(figsize=(8,5))
132 plt.plot(train_accs, label="Training Accuracy")
133 plt.plot(val_accs, label="Validation Accuracy")
134 plt.title("JAFFE - Training vs Validation Accuracy")
135 plt.xlabel("Epoch"); plt.ylabel("Accuracy"); plt.grid(True); plt.legend(); plt.show()
136

```

Mounted at /content/drive
Classes: ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise'] | N: 213
/tmp/ipython-input-4112904230.py:95: FutureWarning: `torch.cuda.amp.GradScaler(args...)` is deprecated. Please use `torch.amp.GradScaler`
scaler = torch.cuda.amp.GradScaler(enabled=USE_AMP)
/tmp/ipython-input-4112904230.py:106: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast`
with torch.cuda.amp.autocast(enabled=USE_AMP):
/tmp/ipython-input-4112904230.py:119: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast`
with torch.no_grad(), torch.cuda.amp.autocast(enabled=USE_AMP):

Epoch	Train Acc	Val Acc
Epoch 1/45	0.1648	0.2258
Epoch 2/45	0.2747	0.1613
Epoch 3/45	0.3846	0.1613
Epoch 4/45	0.4231	0.2581
Epoch 5/45	0.5440	0.2581
Epoch 6/45	0.5659	0.2903
Epoch 7/45	0.6154	0.1935
Epoch 8/45	0.6264	0.2903
Epoch 9/45	0.6813	0.4516
Epoch 10/45	0.6868	0.4839
Epoch 11/45	0.7418	0.5161
Epoch 12/45	0.7363	0.5806
Epoch 13/45	0.7363	0.3871
Epoch 14/45	0.7253	0.5484
Epoch 15/45	0.7582	0.4839
Epoch 16/45	0.8681	0.5484
Epoch 17/45	0.8297	0.5484
Epoch 18/45	0.7967	0.6129
Epoch 19/45	0.8132	0.5484
Epoch 20/45	0.8297	0.5806
Epoch 21/45	0.8791	0.6129
Epoch 22/45	0.8352	0.6129
Epoch 23/45	0.8901	0.5806
Epoch 24/45	0.8352	0.5484
Epoch 25/45	0.9011	0.4839
Epoch 26/45	0.8736	0.5806
Epoch 27/45	0.8901	0.5806
Epoch 28/45	0.8681	0.6129
Epoch 29/45	0.8956	0.6129
Epoch 30/45	0.8901	0.5806
Epoch 31/45	0.8846	0.5806
Epoch 32/45	0.8956	0.5806
Epoch 33/45	0.8956	0.5161
Epoch 34/45	0.8846	0.5806
Epoch 35/45	0.9011	0.5484
Epoch 36/45	0.8956	0.5806
Epoch 37/45	0.9176	0.7097
Epoch 38/45	0.9231	0.5806
Epoch 39/45	0.9066	0.6129
Epoch 40/45	0.8956	0.4839
Epoch 41/45	0.9231	0.6774
Epoch 42/45	0.9451	0.5161
Epoch 43/45	0.9011	0.6452
Epoch 44/45	0.9725	0.5806
Epoch 45/45	0.9286	0.6129



1 # =====

2 # فقط: تحميل + تدريب خفيف + تلخيص الموديل + رسم الدقة FER-2013

```

3 # =====
4 import os, random, numpy as np
5 import torch, torch.nn as nn, torch.optim as optim
6 from torch.utils.data import DataLoader, random_split, Subset
7 from torchvision import datasets, transforms
8 from torchvision.models import resnet18, ResNet18_Weights
9 import matplotlib.pyplot as plt
10
11 # --- (بفولدرات الفئات ImageFolder) عدل المسار حسب مجلدك ---
12 FER_DIR = "/content/drive/MyDrive/Datasets/FER-2013/train"
13
14 # إعدادات سريعة
15 EPOCHS = 45
16 BATCH = 128
17 VAL_RATIO = 0.15
18 PER_CLASS_CAP = 200 # حد أقصى لكل فئة لتقليل الزمن
19 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
20 torch.backends.cudnn.benchmark = True
21 USE_AMP = torch.cuda.is_available()
22
23 # تحويلات مناسبة لأوزان ImageNet
24 IMAGENET_MEAN=(0.485,0.456,0.406); IMAGENET_STD=(0.229,0.224,0.225)
25 COMMON_TF = transforms.Compose([
26     transforms.Grayscale(3),
27     transforms.Resize((224,224)),
28     transforms.ToTensor(),
29     transforms.Normalize(IMAGENET_MEAN, IMAGENET_STD),
30 ])
31
32 def cap_per_class_indices(targets, per_class=PER_CLASS_CAP):
33     """لأخذ عينة متوازنة حتى لكل فئة per_class."""
34     from collections import defaultdict
35     d=defaultdict(list)
36     for i,y in enumerate(targets): d[y].append(i)
37     sel=[]
38     for y,ix in d.items():
39         k = min(per_class, len(ix))
40         sel.extend(random.sample(ix, k))
41     random.shuffle(sel)
42     return sel
43
44 def make_loaders(ds, batch=BATCH, val_ratio=VAL_RATIO, seed=42):
45     v = int(len(ds)*val_ratio); t = len(ds)-v
46     tr, va = random_split(ds, [t,v], generator=torch.Generator().manual_seed(seed))
47     kw = dict(batch_size=batch, num_workers=2, pin_memory=True, persistent_workers=True)
48     return tr, va, DataLoader(tr, shuffle=True, **kw), DataLoader(va, shuffle=False, **kw)
49
50 def show_model_summary(model, title="ملخص الموديل"):
51     print(f"\n===== {title} =====")
52     total = sum(p.numel() for p in model.parameters())
53     trainable = sum(p.numel() for p in model.parameters() if p.requires_grad)
54     print(model)
55     print(f"\nالمعاملات الكلية: {total:,} | القابلة للتدريب: {trainable:,}")
56
57 def train_quick(model, train_loader, val_loader, epochs=EPOCHS, lr=1e-3):
58     crit = nn.CrossEntropyLoss()
59     opt = optim.AdamW(model.parameters(), lr=lr)
60     scaler = torch.cuda.amp.GradScaler(enabled=USE_AMP)
61     train_hist, val_hist = [], []
62
63     for ep in range(1, epochs+1):
64         # TRAIN
65         model.train(); tr_corr=tr_cnt=0
66         for x,y in train_loader:
67             x,y = x.to(device, non_blocking=True), y.to(device, non_blocking=True)
68             opt.zero_grad(set_to_none=True)
69             with torch.cuda.amp.autocast(enabled=USE_AMP):
70                 out = model(x); loss = crit(out,y)
71             scaler.scale(loss).backward(); scaler.step(opt); scaler.update()

```