

```

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

```

```

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

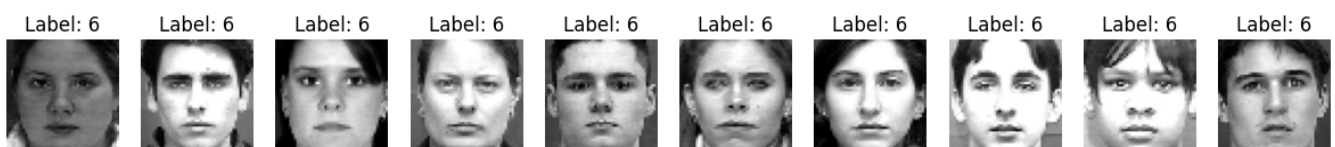
```

Mounted at /content/drive

```

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

```



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



RGB Image Resized



Grayscale Image



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

```

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

```

```

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

```



```

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

```

```

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

```



```

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

```

```

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

```



```

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

```

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



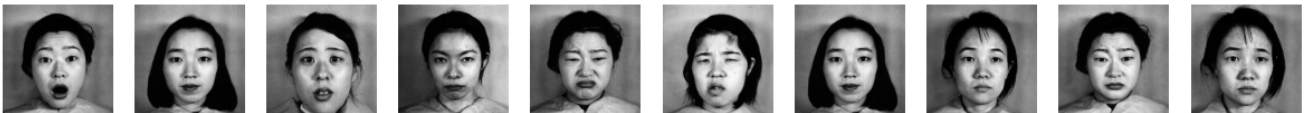
```

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

```



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



```

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

```

```

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

```

```

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

```



```

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

```

```

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

```

```

1 # # (Dropout و BatchNorm) تفعيل وضع التدريب
2
3
4 epochs = 3
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'JAFKE Epoch {epoch+1}: Accuracy={epoch_acc:.4f}, Loss={running_l
21

```

```

→ JAFKE Epoch 1: Accuracy=0.0141, Loss=3.5984
  JAFKE Epoch 2: Accuracy=0.0469, Loss=3.3067
  JAFKE Epoch 3: Accuracy=0.0704, Loss=3.2142

```

```

1 model.classifier[1] = nn.Linear(model.last_channel, num_classes_ck)
2 model = model.to(device)
3
4 for epoch in range(3):
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:.4f}')
20

```

 CK+ Epoch 1: Accuracy=0.5815, Loss=1.5426
 CK+ Epoch 2: Accuracy=0.7011, Loss=1.0489
 CK+ Epoch 3: Accuracy=0.7348, Loss=0.9035

```

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.moun

```

1 import cv2
2 resized_image = cv2.resize(image, (128, 128), interpolation=cv2.INTER_CUBIC)

```



```
-----  
error                                Traceback (most recent call last)  
/tmp/ipython-input-71-1231681638.py in <cell line: 0>()  
    1 import cv2  
----> 2 resized_image = cv2.resize(image, (128, 128), interpolation=cv2.INTER_CUBIC)  
  
error: OpenCV(4.11.0) /io/opencv/modules/imgproc/src/resize.cpp:4208: error:  
(-215:Assertion failed) !ssize.empty() in function 'resize'
```

1 Start coding or generate with AI.



Resized RGB Image



Double-click (or enter) to edit

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



```
-----  
NameError                                Traceback (most recent call last)  
/tmp/ipython-input-1-2671325200.py in <cell line: 0>()  
----> 1 plt.figure(figsize=(10,10), dpi=200) # dpi = dots per inch  
      2 plt.imshow(image)  
      3 plt.axis('off')  
      4 plt.show()
```

NameError: name 'plt' is not defined

```
1 gray_image = cv2.imread('/content/drive/MyDrive/Datasets/jaffe/KA.AN1.39.tif')  
2
```

```
1 rgb_image = cv2.cvtColor(gray_image, cv2.COLOR_GRAY2RGB)  
2 resized_rgb = cv2.resize(rgb_image, (128,128), interpolation=cv2.INTER_CUBIC)  
3  
4 plt.imshow(resized_rgb)  
5 plt.title('Resized RGB Image')  
6 plt.axis('off')  
7 plt.show()  
8
```



Resized RGB Image



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

```
1 jaffe_path = '/content/drive/MyDrive/Datasets/jaffe'
2
3 class JaffeDataset(Dataset):
4     def __init__(self, root_dir, transform=None):
5         self.root_dir = root_dir
6         self.image_files = [f for f in os.listdir(root_dir) if f.endswith(".
7         self.transform = transform
8         self.labels = [self.get_label(f) for f in self.image_files]
9         self.label_map = {label: idx for idx, label in enumerate(sorted(set(
```

```

10         self.labels_idx = [self.label_map[label] for label in self.labels]
11
12     def get_label(self, filename):
13         return filename.split('.')[1] # AN, HA, etc.
14
15     def __len__(self):
16         return len(self.image_files)
17
18     def __getitem__(self, idx):
19         img_path = os.path.join(self.root_dir, self.image_files[idx])
20         image = Image.open(img_path).convert('RGB')
21         if self.transform:
22             image = self.transform(image)
23         label = self.labels_idx[idx]
24         return image, label
25
26 jaffe_dataset = JaffeDataset(jaffe_path, transform=transform)
27 jaffe_loader = DataLoader(jaffe_dataset, batch_size=8, shuffle=True)
28 num_classes_jaffe = len(set(jaffe_dataset.labels))
29

```

```

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

```

```
26
27     def forward(self, x):
28         return self.model(x)
29

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

1 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
2
3 model = get_mobilenetv2(num_classes_jaffe)
4 model = model.to(device)
5 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
6 criterion = nn.CrossEntropyLoss()
7
8 epochs = 5
9 train_losses, train_accs = [], []
10
11 for epoch in range(epochs):
12     model.train()
13     running_loss, correct = 0.0, 0
14     for inputs, labels in jaffe_loader:
15         inputs, labels = inputs.to(device), labels.to(device)
16         optimizer.zero_grad()
17         outputs = model(inputs)
18         loss = criterion(outputs, labels)
19         loss.backward()
20         optimizer.step()
21
22         running_loss += loss.item() * inputs.size(0)
23         _, preds = torch.max(outputs, 1)
24         correct += torch.sum(preds == labels.data)
25
26     epoch_loss = running_loss / len(jaffe_dataset)
27     epoch_acc = correct.double() / len(jaffe_dataset)
28
29     train_losses.append()
30     train_accs.append(epoch_acc.item())
31
```

```

32     print(f"Epoch {epoch+1}/{epochs}: Loss={epoch_loss:.4f}, Accuracy={epoch_
33

```

```

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

```

```

-----
TypeError                                Traceback (most recent call last)
/tmp/ipython-input-67-188733086.py in <cell line: 0>()
    27     epoch_acc = correct.double() / len(jaffe_dataset)
    28
---> 29     train_losses.append()
    30     train_accs.append(epoch_acc.item())
    31

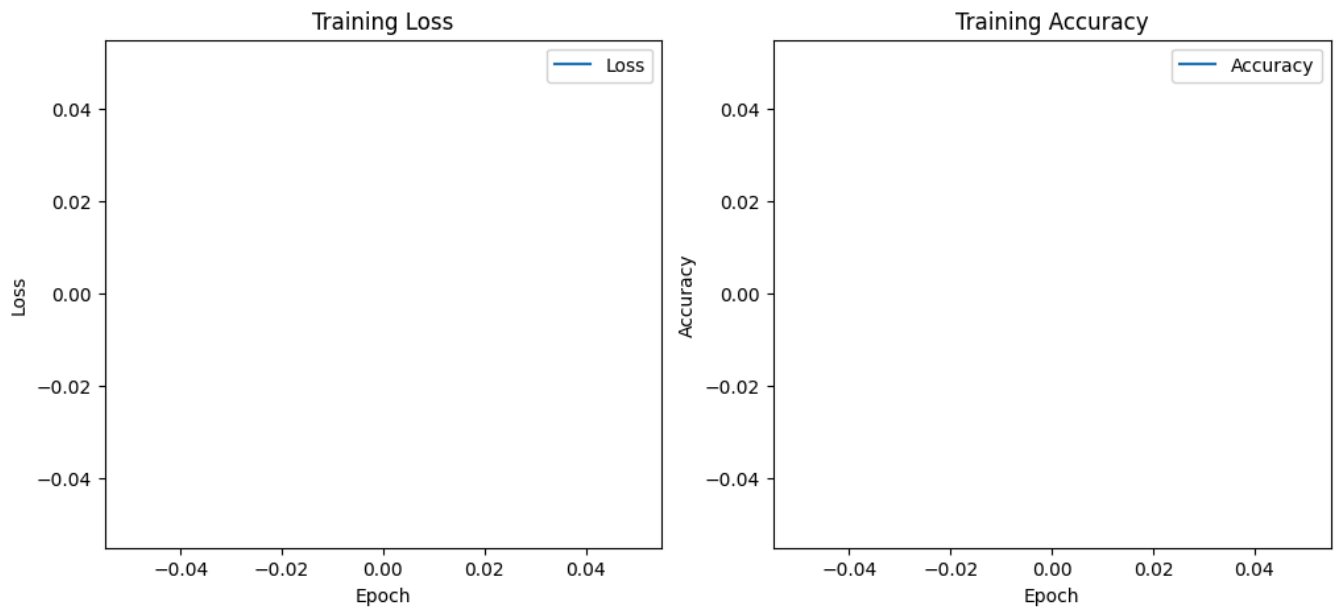
```

TypeError: list.append() takes exactly one argument (0 given)

```

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

```



```

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

```



```

17 -0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 6
18 -2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 1 0 0 0 2 1

```

```

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

```

```

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

```

```

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

```

```

1 jaffe_path = '/content/drive/MyDrive/Datasets/jaffe'
2
3 class JaffeDataset(Dataset):
4     def __init__(self, root_dir, transform=None):
5         self.root_dir = root_dir
6         self.image_files = [f for f in os.listdir(root_dir) if f.endswith(".
7         self.transform = transform
8         self.labels = [self.get_label(f) for f in self.image_files]
9         self.label_map = {label: idx for idx, label in enumerate(sorted(set(
10        self.labels_idx = [self.label_map[label] for label in self.labels]
11
12    def get_label(self, filename):
13        return filename.split('.')[1] # AN, HA, etc.
14
15    def __len__(self):
16        return len(self.image_files)
17
18    def __getitem__(self, idx):
19        img_path = os.path.join(self.root_dir, self.image_files[idx])
20        image = Image.open(img_path).convert('RGB')
21        if self.transform:
22            image = self.transform(image)
23        label = self.labels_idx[idx]
24        return image, label
25
26 transform = T.Compose([
27     T.RandomHorizontalFlip(),
28     T.RandomVerticalFlip(),
29     T.RandomRotation(10),
30     T.Resize((128,128)),
31     T.ToTensor()
32 ])
33
34 jaffe_dataset = JaffeDataset(jaffe_path, transform=transform)
35 jaffe_loader = DataLoader(jaffe_dataset, batch_size=8, shuffle=True)

```

```
36 num_classes_jaffe = len(set(jaffe_dataset.labels))
37

1 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=2):
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: 3.5536 Acc: 0.0282

Epoch 2/2

Loss: 3.4113 Acc: 0.0235

```

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 def evaluate_model(model_name, y_true, y_pred, y_proba):
23     labels = list(jaffe_dataset.label_map.keys())
24     accuracy = accuracy_score(y_true, y_pred)
25     precision = precision_score(y_true, y_pred, average='weighted', zero_div
26     recall = recall_score(y_true, y_pred, average='weighted')
27     f1 = f1_score(y_true, y_pred, average='weighted')
28     cm = confusion_matrix(y_true, y_pred)
29
30     print(f"[{model_name}] Accuracy: {accuracy:.4f}")
31     print("Precision:", precision)
32     print("Recall:", recall)
33     print("F1-score:", f1)
34     print("Classification Report:\n", classification_report(y_true, y_pred,
35
36
37     plt.figure(figsize=(6,5))
38     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, y
39     plt.title('Confusion Matrix')
40     plt.show()

```

21

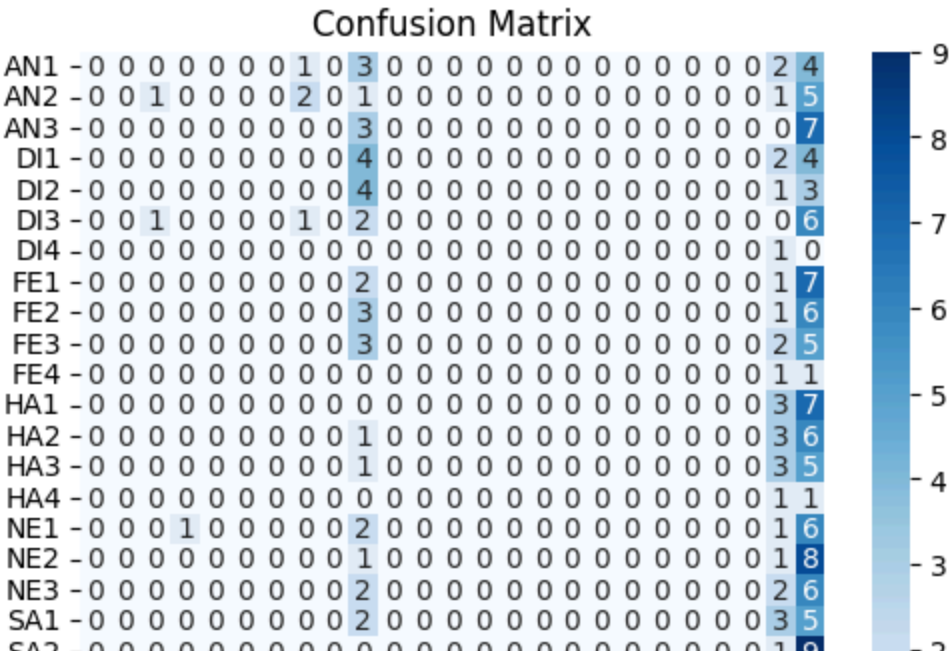
22 `evaluate_model("ResNet18 on JAFFE", y_true, y_pred, y_proba)`

23



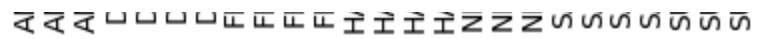
[ResNet18 on JAFFE] Accuracy: 0.0563
Precision: 0.009428280207561158
Recall: 0.056338028169014086
F1-score: 0.015584813227189222
Classification Report:

	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	0.00	0.00	0.00	10
DI4	0.00	0.00	0.00	1
FE1	0.00	0.00	0.00	10
FE2	0.00	0.00	0.00	10
FE3	0.08	0.30	0.12	10
FE4	0.00	0.00	0.00	2
HA1	0.00	0.00	0.00	10
HA2	0.00	0.00	0.00	10
HA3	0.00	0.00	0.00	9
HA4	0.00	0.00	0.00	2
NE1	0.00	0.00	0.00	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	0.00	0.00	0.00	10
SA3	0.00	0.00	0.00	10
SA5	0.00	0.00	0.00	1
SU1	0.00	0.00	0.00	10
SU2	0.07	0.30	0.12	10
SU3	0.05	0.60	0.09	10
accuracy			0.06	213
macro avg	0.01	0.05	0.01	213
weighted avg	0.01	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 # تعريف CKPlusDataset الذي يرث من torch.utils.data.Dataset
2 # إلى شكل يمكن التعامل معه بالتدريب DataFrame المخزنة في CK+ هذا الكلاس مخصص لتحويل بيانات
3
4 class CKPlusDataset(Dataset):
5     def __init__(self, dataframe, transform=None):
6         self.df = dataframe
7         self.transform = transform
8         # عدد العينات في البيانات، أي عدد الصفوف في DataFrame
9
10    def __len__(self):
11        return len(self.df)
12        # تحويل القائمة إلى مصفوفة NumPy 48x48 بشكل صورة رمادية
13        # وتحويلها إلى تدرج رمادي PIL إلى كائن صورة NumPy تحويل الصورة من
14        # (augmentations + Resize + ToTensor) تطبيق التحويلات
15
16    def __getitem__(self, idx):
17        pixel_values = [int(p) for p in self.df.iloc[idx]['pixels'].split()]
18        image = np.array(pixel_values, dtype='uint8').reshape(48, 48)
19        image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
20        image = cv2.resize(image, (128,128))
21        image = Image.fromarray(image)
22        label = self.df.iloc[idx]['emotion']
23        if self.transform:
24            image = self.transform(image)
25        return image, label-
26
27 # تعريف مجموعة التحويلات التي سنطبق على صور CK+
28
29 transform = T.Compose([
30     T.RandomHorizontalFlip(),
31     T.RandomVerticalFlip(),
32     T.RandomRotation(10),
33     T.Resize((128,128)),
34     T.ToTensor()
35 ])
36
37 ck_dataset = CKPlusDataset(df_ck, transform=transform)
38 ck_loader = DataLoader(ck_dataset, batch_size=8, shuffle=True)
39 num_classes_ck = df_ck['emotion'].nunique()
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

```

```

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: 1.4377 Acc: 0.6152

```

```

Epoch 2/2
Loss: 1.1417 Acc: 0.6815

```

```

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

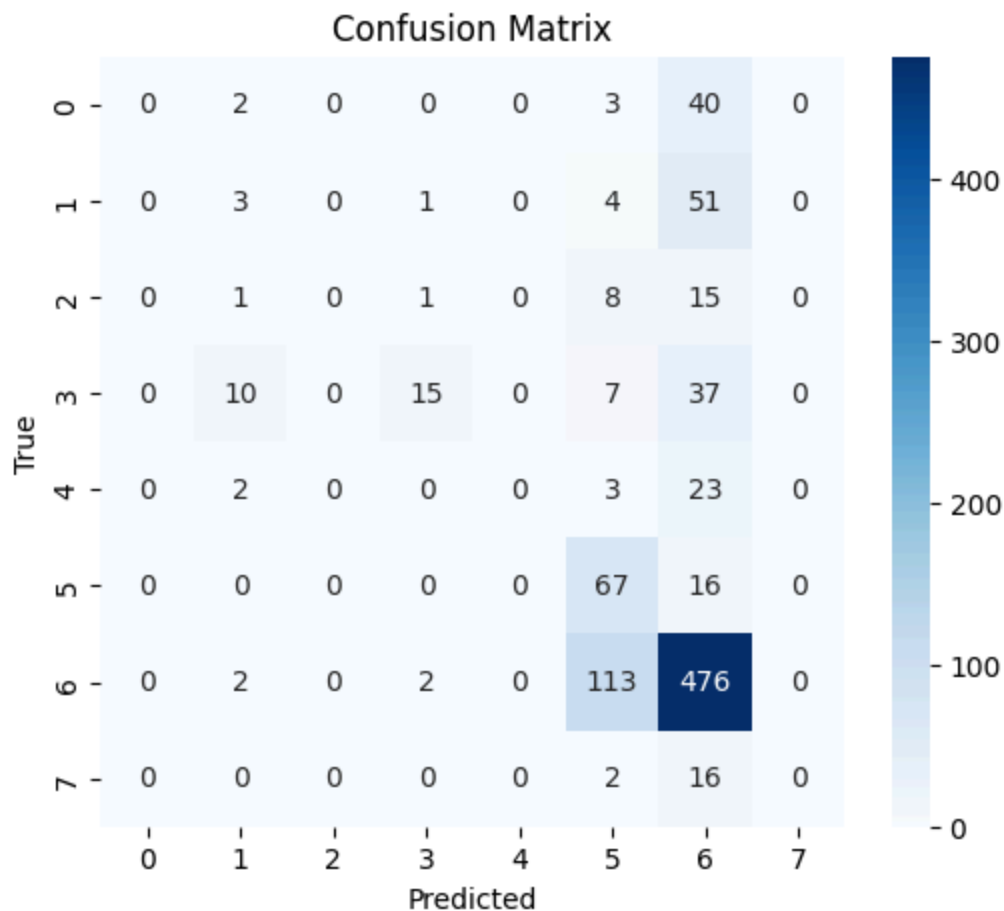
```

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



[ResNet18 on CK+] Accuracy: 0.6098
 Precision: 0.5532431203647419
 Recall: 0.6097826086956522
 F1-score: 0.5564397009156161
 Classification Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	45
1	0.15	0.05	0.08	59
2	0.00	0.00	0.00	25
3	0.79	0.22	0.34	69
4	0.00	0.00	0.00	28
5	0.32	0.81	0.46	83
6	0.71	0.80	0.75	593
7	0.00	0.00	0.00	18
accuracy			0.61	920
macro avg	0.25	0.23	0.20	920
weighted avg	0.55	0.61	0.56	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: 1.4453 Acc: 0.6250

```

```

Epoch 2/2
Loss: 1.4050 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

```

```

1 evaluate_model("MobileNetV2 on CK+", y_true, y_pred, y_proba)
2

```

➞ [MobileNetV2 on CK+] Accuracy: 0.6424

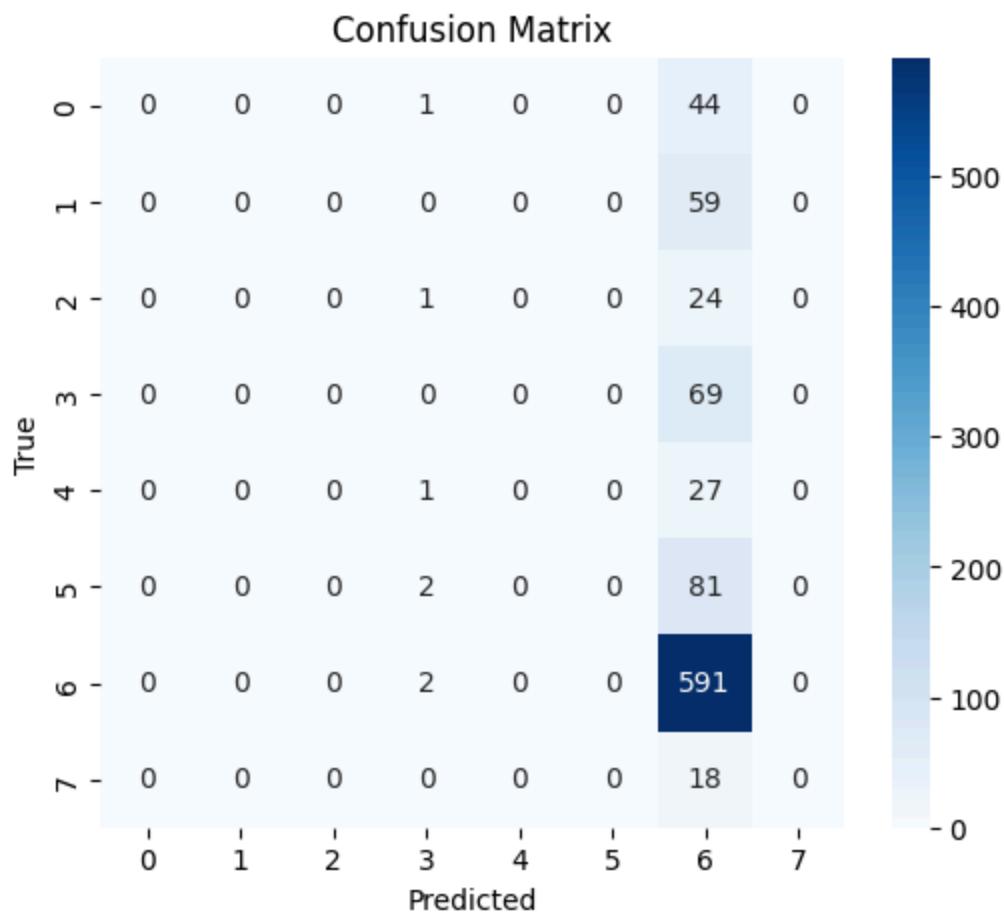
Precision: 0.4172377256059812

Recall: 0.6423913043478261

F1-score: 0.505893816040187

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

```

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

```

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

```

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

```
1 model = train_model(model, criterion, optimizer, scheduler, num_epochs=2)
2
```



```
Epoch 1/2
Loss: 1.5492, Accuracy: 0.6033
```

```
Epoch 2/2
Loss: 1.3721, Accuracy: 0.6391
```

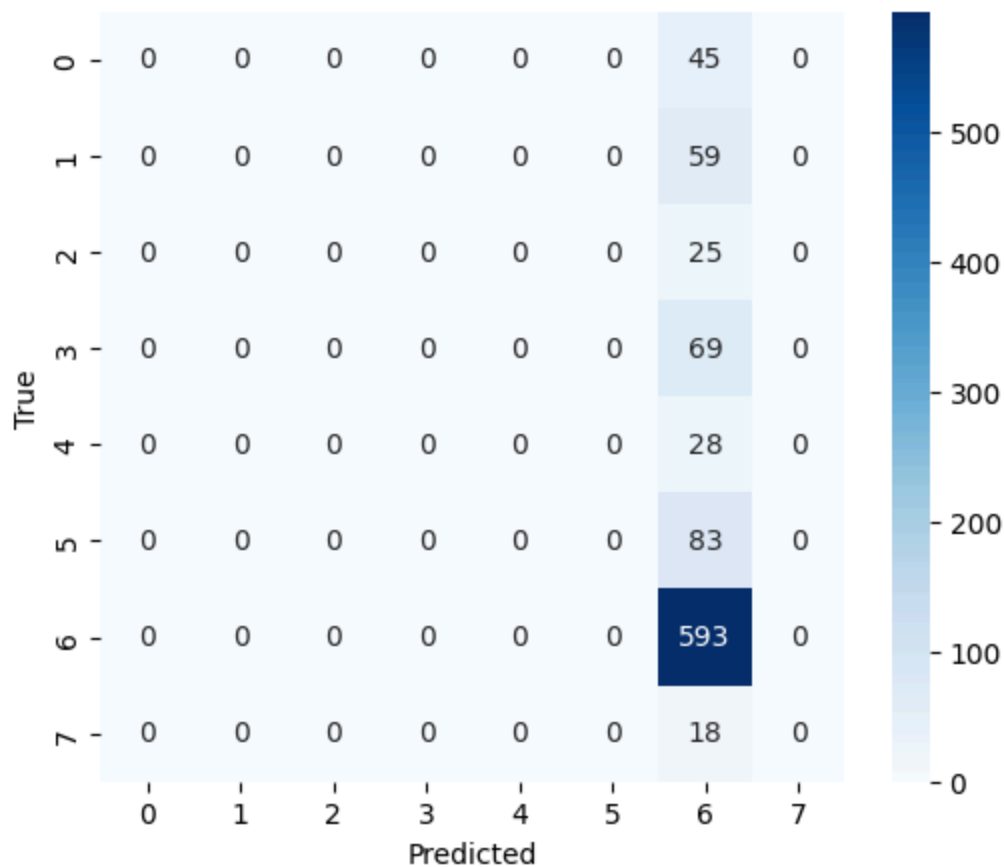
```
1 from sklearn.metrics import accuracy_score, precision_score, recall_score, f
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4
5 y_true, y_pred = [], []
6
7 model.eval()
8 with torch.no_grad():
9     for inputs, labels in ck_loader:
10         inputs = inputs.to(device)
11         outputs = 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

Confusion Matrix



```

1 model = models.densenet121(pretrained=False)
2 model.classifier = nn.Linear(model.classifier.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_epoch
7

```



Epoch 1/1
Loss: 1.4313, Accuracy: 0.6315

```

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

```



Epoch 1/1
Loss: 2.7828, Accuracy: 0.6402

```

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: 0.0196
Precision: 0.0029644095683425842
Recall: 0.01956521739130435