```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("shawon10/ckplus")
path = path + '/CK+48'

print("Path to dataset files:", path)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/shawon10/ckplus?dataset_version_number=1...
100%|██████████| 3.63M/3.63M [00:00<00:00, 31.8MB/s]Extracting files...

Path to dataset files: /root/.cache/kagglehub/datasets/shawon10/ckplus/versions/1/CK+48
```

```
import os

# Define the folder path
folder_path = path

# List items in the folder
items = os.listdir(folder_path)

# Print the items
for item in items:
    print(item)
```

```
disgust
contempt
happy
sadness
fear
surprise
anger
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Data generators
```

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    validation_split=0.2
)
```

```
train_generator = train_datagen.flow_from_directory(
    path,
    target_size=(48, 48),
    color_mode='grayscale',
    batch_size=32,
    class_mode='categorical',
    subset='training'
)
```

```
Found 788 images belonging to 7 classes.
```

```
validation_generator = train_datagen.flow_from_directory(
    path,
    target_size=(48, 48),
    color_mode='grayscale',
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 193 images belonging to 7 classes.
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

```
# Define the CNN model
model = Sequential([
    # First convolutional layer
    Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)),
    MaxPooling2D(2, 2),

    # Second convolutional layer
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),

    # Third convolutional layer
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),

    # Flatten and dense layers
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(7, activation='softmax')  # 7 for our 7 emotions
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Compile the model
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model.summary()
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 46, 46, 32)	320
max_pooling2d_3 (MaxPooling2D)	(None, 23, 23, 32)	0
conv2d_4 (Conv2D)	(None, 21, 21, 64)	18,496
max_pooling2d_4 (MaxPooling2D)	(None, 10, 10, 64)	0
conv2d_5 (Conv2D)	(None, 8, 8, 128)	73,856
max_pooling2d_5 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten_1 (Flatten)	(None, 2048)	0
dense_2 (Dense)	(None, 128)	262,272
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 7)	903

```
Total params: 355,847 (1.36 MB)
Trainable params: 355,847 (1.36 MB)
Non-trainable params: 0 (0.00 B)
```

```
# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // 32,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // 32,
    epochs=25
)
```

```
Epoch 1/25
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: You
  self._warn_if_super_not_called()
```

```
24/24 ━━━━━━━━━━━━━━━━━━━━ 6s 166ms/step - accuracy: 0.2335 - loss: 1.8875 - val_accuracy: 0.2552 - val_loss: 1.8218
Epoch 2/25
 1/24 ━━━━━━━━━━━━━━━━━━━━ 3s 140ms/step - accuracy: 0.2188 - loss: 1.9175/usr/local/lib/python3.11/dist-packages/ke
  self._interrupted_warning()
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.2188 - loss: 1.9175 - val_accuracy: 0.2552 - val_loss: 1.8234
Epoch 3/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 5s 194ms/step - accuracy: 0.2378 - loss: 1.8363 - val_accuracy: 0.3021 - val_loss: 1.8017
Epoch 4/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.2188 - loss: 1.7873 - val_accuracy: 0.3073 - val_loss: 1.7997
Epoch 5/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 112ms/step - accuracy: 0.2669 - loss: 1.8035 - val_accuracy: 0.4062 - val_loss: 1.6686
Epoch 6/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6250 - loss: 1.4976 - val_accuracy: 0.4115 - val_loss: 1.6542
Epoch 7/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 5s 218ms/step - accuracy: 0.4364 - loss: 1.5443 - val_accuracy: 0.4740 - val_loss: 1.4140
Epoch 8/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.4688 - loss: 1.5034 - val_accuracy: 0.4844 - val_loss: 1.3817
Epoch 9/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 8s 114ms/step - accuracy: 0.5036 - loss: 1.2823 - val_accuracy: 0.5312 - val_loss: 1.2746
Epoch 10/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.3438 - loss: 1.5999 - val_accuracy: 0.5677 - val_loss: 1.2714
Epoch 11/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 5s 130ms/step - accuracy: 0.5362 - loss: 1.2068 - val_accuracy: 0.5469 - val_loss: 1.1717
Epoch 12/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.5938 - loss: 1.1804 - val_accuracy: 0.5104 - val_loss: 1.2126
Epoch 13/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 4s 114ms/step - accuracy: 0.5849 - loss: 1.1236 - val_accuracy: 0.5990 - val_loss: 1.0588
Epoch 14/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.4062 - loss: 1.1753 - val_accuracy: 0.5729 - val_loss: 1.0394
Epoch 15/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 6s 162ms/step - accuracy: 0.5912 - loss: 1.0511 - val_accuracy: 0.6615 - val_loss: 0.9659
Epoch 16/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.5938 - loss: 1.0271 - val_accuracy: 0.6198 - val_loss: 1.0362
Epoch 17/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 5s 181ms/step - accuracy: 0.6287 - loss: 0.9535 - val_accuracy: 0.6771 - val_loss: 0.8589
Epoch 18/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.5312 - loss: 1.2738 - val_accuracy: 0.6562 - val_loss: 0.9636
Epoch 19/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 5s 197ms/step - accuracy: 0.6333 - loss: 0.9202 - val_accuracy: 0.6510 - val_loss: 0.9342
Epoch 20/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.5938 - loss: 0.8645 - val_accuracy: 0.6302 - val_loss: 0.9107
```

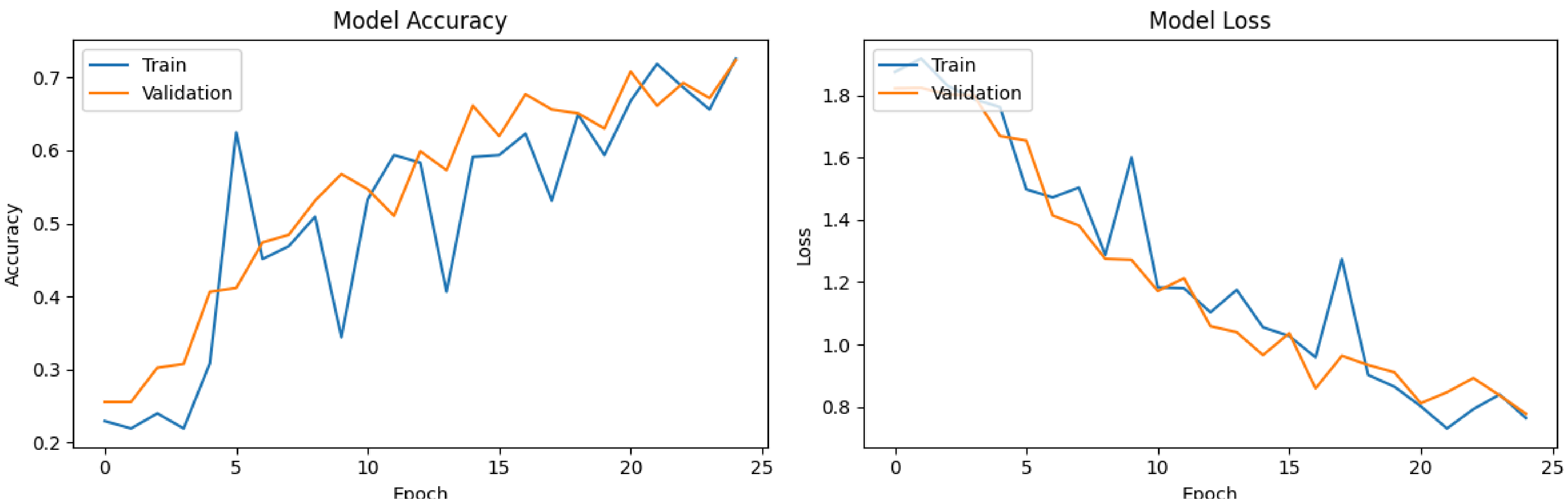
```
Epoch 21/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 112ms/step - accuracy: 0.6725 - loss: 0.8342 - val_accuracy: 0.7083 - val_loss: 0.8117
Epoch 22/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.7188 - loss: 0.7306 - val_accuracy: 0.6615 - val_loss: 0.8467
Epoch 23/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 4s 114ms/step - accuracy: 0.6865 - loss: 0.8101 - val_accuracy: 0.6927 - val_loss: 0.8922
Epoch 24/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6562 - loss: 0.8393 - val_accuracy: 0.6719 - val_loss: 0.8365
Epoch 25/25
24/24 ━━━━━━━━━━━━━━━━━━━━ 4s 163ms/step - accuracy: 0.7109 - loss: 0.7787 - val_accuracy: 0.7240 - val_loss: 0.7777
```

```python
import matplotlib.pyplot as plt

# Plot training & validation accuracy
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.tight_layout()
plt.show()

# Save the model
model.save('facial_expression_model.keras')
```



```
import kagglehub

# Download latest version
test_path = kagglehub.dataset_download("shuvoalok/ck-dataset")

print("Path to dataset files:", test_path)
```

```
Path to dataset files: /root/.cache/kagglehub/datasets/shuvoalok/ck-dataset/versions/1
```

```
# Define the folder path
folder_path = test_path

# List items in the folder
items = os.listdir(folder_path)

# Print the items
for item in items:
    print(item)
```

```
anger
sadness
disgust
fear
contempt
happy
surprise
```

```
# Test data generator
test_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    validation_split=0.2
)


# If you have a separate test set
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    test_path,
    target_size=(48, 48),
    color_mode='grayscale',
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)

# Evaluate the model
evaluation = model.evaluate(test_generator)
print(f"Test Loss: {evaluation[0]:.4f}")
print(f"Test Accuracy: {evaluation[1]:.4f}")
```

```
Found 981 images belonging to 7 classes.
31/31 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.7121 - loss: 0.8373
Test Loss: 0.5780
Test Accuracy: 0.8073
```

```python
from tensorflow.keras.preprocessing import image

# Define the class names (expressions) - replace with your actual classes
class_names = ['angry', 'contempt', 'disgust', 'fear', 'happy', 'sadness', 'surprise']  # Update with your classes

# Function to predict expression from an image
def predict_expression(img_path):
    img = image.load_img(img_path, target_size=(48, 48), color_mode='grayscale')
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0) / 255.0

    prediction = model.predict(img_array)
    expression_idx = np.argmax(prediction)
    expression = class_names[expression_idx]
    confidence = prediction[0][expression_idx]

    return img, expression, confidence


import random

test_images = []

# Get all image files from the test folder (assuming jpg, jpeg, or png)
for root, dirs, files in os.walk(test_path):
    for file in files:
        if file.lower().endswith(('.jpg', '.jpeg', '.png')):
            test_images.append(os.path.join(root, file))

# Randomly select a subset
if len(test_images) > 6:
    test_images = random.sample(test_images, 6)
```

```
print("test: ", test_images)
```

```
test:  ['/root/.cache/kagglehub/datasets/shuvoalok/ck-dataset/versions/1/happy/S124_007_00000022.png', '/root/.cache/
```

```
import numpy as np

# Number of images to display
num_images = min(6, len(test_images))

# Create a figure to display images with predictions
plt.figure(figsize=(15, 10))

for i in range(num_images):
    img_path = test_images[i]

    # Make prediction
    img, expression, confidence = predict_expression(img_path)

    # Display image with prediction
    plt.subplot(2, 3, i+1)
    plt.imshow(np.squeeze(img), cmap='gray')
    plt.title(f"Prediction: {expression}\nConfidence: {confidence:.2f}", fontsize=12)
    plt.axis('off')

plt.tight_layout()
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
```