```
import numpy as np
import pandas as pd


from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
data_path = '/content/drive/MyDrive/6 Emotions for image classification'
test="/content/drive/MyDrive/6 Emotions for image classification/anger"
train="/content/drive/MyDrive/6 Emotions for image classification/happy"
val="/content/drive/MyDrive/6 Emotions for image classification/fear"


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

def create_cnn_model(input_shape=(64, 64, 3), num_classes=10):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=input_shape),
        MaxPooling2D((2, 2)),

        Conv2D(64, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),

        Conv2D(128, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),

        Conv2D(256, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),

        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

model = create_cnn_model()
model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_4"
```

Layer (type)	Output Shape	Param #
conv2d_16 (Conv2D)	(None, 64, 64, 32)	896
max_pooling2d_16 (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_17 (Conv2D)	(None, 32, 32, 64)	18,496
max_pooling2d_17 (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_18 (Conv2D)	(None, 16, 16, 128)	73,856
max_pooling2d_18 (MaxPooling2D)	(None, 8, 8, 128)	0
conv2d_19 (Conv2D)	(None, 8, 8, 256)	295,168
max_pooling2d_19 (MaxPooling2D)	(None, 4, 4, 256)	0
flatten_4 (Flatten)	(None, 4096)	0
dense_8 (Dense)	(None, 512)	2,097,664
dropout_3 (Dropout)	(None, 512)	0
dense_9 (Dense)	(None, 10)	5,130

```
Total params: 2,491,210 (9.50 MB)
Trainable params: 2,491,210 (9.50 MB)
```

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
import os
import random

def create_cnn_model(input_shape=(64, 64, 1)):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2, 2)),

        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),

        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),

        Conv2D(256, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),

        Flatten(),
        Dense(128, activation='relu'),
        Dense(10, activation='softmax')  # Assuming 10 output classes
    ])
    return model

# Create the model
model = create_cnn_model()
model.summary()

def display_images_with_cnn_output(image_folder, model, num_images=20, img_size=(64, 64)):
    image_files = [
        os.path.join(image_folder, f)
        for f in os.listdir(image_folder)
        if os.path.isfile(os.path.join(image_folder, f))
    ]
    random_image_files = random.sample(image_files, min(num_images, len(image_files)))

    fig, axes = plt.subplots(len(random_image_files), 2, figsize=(10, len(random_image_files) * 2))

    for i, image_file in enumerate(random_image_files):
        image = cv2.imread(image_file, cv2.IMREAD_GRAYSCALE)
        resized_image = cv2.resize(image, img_size)
        processed_image = resized_image.astype("float32") / 255.0
        input_image = np.expand_dims(processed_image, axis=[0, -1])

        prediction = model.predict(input_image)[0]

        axes[i, 0].imshow(resized_image, cmap="gray")
        axes[i, 0].axis("off")
        axes[i, 0].set_title(f"Original: {os.path.basename(image_file)}")

        axes[i, 1].text(0.5, 0.5, f"Prediction: {prediction}", ha='center', va='center', fontsize=12)
        axes[i, 1].axis("off")

    plt.tight_layout()
    plt.show()

# Call the function
image_folder = "/content/drive/MyDrive/6 Emotions for image classification/anger"  # Replace with your dataset path
display_images_with_cnn_output(image_folder, model, num_images=20, img_size=(64, 64))
```

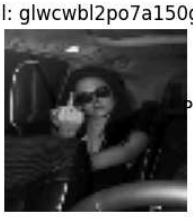
Model: "sequential_5"

Layer (type)	Output Shape	Param #
conv2d_20 (Conv2D)	(None, 62, 62, 32)	320
max_pooling2d_20 (MaxPooling2D)	(None, 31, 31, 32)	0
conv2d_21 (Conv2D)	(None, 29, 29, 64)	18,496
max_pooling2d_21 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_22 (Conv2D)	(None, 12, 12, 128)	73,856
max_pooling2d_22 (MaxPooling2D)	(None, 6, 6, 128)	0
conv2d_23 (Conv2D)	(None, 4, 4, 256)	295,168
max_pooling2d_23 (MaxPooling2D)	(None, 2, 2, 256)	0
flatten_5 (Flatten)	(None, 1024)	0
dense_10 (Dense)	(None, 128)	131,200
dense_11 (Dense)	(None, 10)	1,290

Total params: 520,330 (1.98 MB)
Trainable params: 520,330 (1.98 MB)
Non-trainable params: 0 (0.00 B)

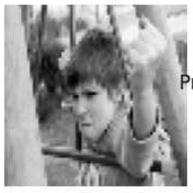
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 319ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
```

Original: glwcwbl2po7a150gp4.jpg



Prediction: [0.10266884 0.0976698 0.10046215 0.10261572 0.09524395 0.10076533 0.09742123 0.10156593 0.10219086 0.09939624]

Original: OIP.dn2dWcF-BTEj-RKzuFdqhwAAAA.jpg



Prediction: [0.10347217 0.09643664 0.09978675 0.10439316 0.09090319 0.10156061 0.09732837 0.10390867 0.1045128 0.09769763]

Original: pexels1.jpg

Prediction: [0.10334346 0.0966695 0.09937106 0.1041842 0.0903846 0.10318813 0.09657434 0.10291263 0.10555349 0.09781863]

Original: h3x8kpywna15busbgj.jpg

Prediction: [0.10445835 0.09562701 0.09918924 0.10614935 0.08891232 0.10170811 0.09588935 0.10550179 0.10589346 0.09667104]

Original: gs2trkwi7hos146z4w.jpg

Prediction: [0.10169248 0.09807735 0.09945707 0.10374696 0.09219062 0.10005464
0.09895124 0.10383494 0.10465316 0.09734158]

Original: woman-at-PC_WTF_16x9.jpg

Prediction: [0.10449998 0.09587776 0.09986382 0.10542466 0.09001335 0.10089631
0.09583024 0.10482362 0.10542505 0.09734525]

Original: g6xgrntszdu0qzwpji.jpg

Prediction: [0.10468552 0.09587098 0.09925377 0.10629433 0.08949299 0.10174555
0.09700631 0.10504603 0.10442863 0.09617586]

Original: ive-had-it.jpg

Prediction: [0.10235722 0.09705966 0.09957598 0.10565813 0.08943037 0.10181143
0.09730233 0.10430273 0.10584748 0.09665468]

Original: white-background-conceptual-studio-shoot-of-a-angry-indian-bearded-man.jpg

Prediction: [0.10333037 0.09705131 0.10028402 0.1045671 0.08953992 0.1017512
0.09583752 0.10499366 0.10640714 0.09623777]

Original: OIP.6t0EWZ09nvNTr_GNGp-zKwAAAA.jpg

Prediction: [0.10364828 0.09704276 0.09988506 0.10621583 0.08998174 0.10074753
0.09626916 0.10476081 0.10417396 0.09727494]

Original: road-rage-criminal-offense-law-offices-of-anthony-carbone-1024x682.jpg

Prediction: [0.10442021 0.09657348 0.10050119 0.10759274 0.08914748 0.10052708
0.09581031 0.10457331 0.10380489 0.09704933]

Original: senior-businessman-making-threatening-video-call.jpg

Prediction: [0.10337877 0.09705297 0.10067925 0.10591346 0.09162665 0.10039806
0.0960933 0.10367929 0.10361758 0.09756063]

Original: OIP.mZOr1_UhZJePTPnSDiLjJAHaEo.jpg

Prediction: [0.10411293 0.0962512 0.09948318 0.1049577 0.09066077 0.10149523 0.09511539 0.10546596 0.10508979 0.09736773]

Original: mad-male-employee-blaming-female-colleague-for-mistake.jpg

Prediction: [0.10339199 0.09694543 0.10006097 0.10614919 0.08953382 0.1010569 0.09551883 0.10444666 0.1054482 0.09744804]

Original: gjd2m0twkxuuko294r.jpg

Prediction: [0.10376717 0.09673867 0.10015713 0.10515578 0.0898734 0.10160913 0.09661642 0.10397214 0.10468061 0.09742958]

Original: y-modern-office-people-emotions-business-concept-copy-space-wide-angle-photo.jpg

Prediction: [0.10350711 0.09656072 0.09944376 0.10413741 0.09136178 0.10169161 0.09664792 0.1039343 0.10505571 0.09765971]

Original: gg8x0zudzwpy7arvmb.jpg

Prediction: [0.10450967 0.09845087 0.10035817 0.10451085 0.09174979 0.10053273 0.09592044 0.10367015 0.10366955 0.09662784]

Original: xtremely-bossy-and-took-joy-telling-people-how-to-live.-He-didnt-know-Jesus..jpg

Prediction: [0.10251418 0.09750212 0.09980875 0.10462269 0.09175111 0.10018508 0.09780472 0.10424546 0.10448306 0.09708278]

Original: throwing-documents-yelling-burn-out-looking-screen-blunder-mistake-183655610.jpg

Prediction: [0.10327753 0.09687518 0.10012099 0.10422469 0.09044473 0.10063402 0.09747571 0.10421389 0.10510709 0.09762611]

Original: g5bu83q50sftz7rx9n.jpg

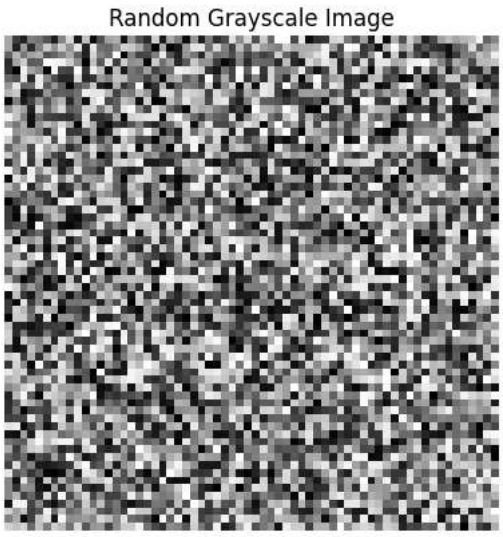
Prediction: [0.10278586 0.09724572 0.10039161 0.10534213 0.09041177 0.10054862 0.09761957 0.10409296 0.10453653 0.09702526]

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

# Define CNN model for grayscale images
print("Model for Grayscale images:")
def create_cnn_model(input_shape=(64, 64, 1), num_classes=10):
    model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=input_shape),
        tf.keras.layers.MaxPooling2D((2, 2)),

        tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        tf.keras.layers.MaxPooling2D((2, 2)),

        tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        tf.keras.layers.MaxPooling2D((2, 2)),

        tf.keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
        tf.keras.layers.MaxPooling2D((2, 2)),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(512, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Initialize model for grayscale images
model_gray = create_cnn_model(input_shape=(64, 64, 1))
model_gray.summary()

# Function to display a random grayscale image
def display_random_gray_image():
    random_image = np.random.rand(64, 64)
    plt.imshow(random_image, cmap="gray")  # Use 'gray' colormap for grayscale images
    plt.axis("off")
    plt.title("Random Grayscale Image")
    plt.show()

display_random_gray_image()

# Generate synthetic grayscale dataset for training
x_train_gray = np.random.rand(1000, 64, 64, 1)  # (Samples, Height, Width, Channels=1)
y_train_gray = tf.keras.utils.to_categorical(np.random.randint(10, size=(1000,)), num_classes=10)

# Train model for 50 epochs
model_gray.fit(x_train_gray, y_train_gray, epochs=50, batch_size=32, verbose=1)
```

Model for Grayscale images:
Model: "sequential_6"

Layer (type)	Output Shape	Param #
conv2d_24 (Conv2D)	(None, 64, 64, 32)	320
max_pooling2d_24 (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_25 (Conv2D)	(None, 32, 32, 64)	18,496
max_pooling2d_25 (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_26 (Conv2D)	(None, 16, 16, 128)	73,856
max_pooling2d_26 (MaxPooling2D)	(None, 8, 8, 128)	0
conv2d_27 (Conv2D)	(None, 8, 8, 256)	295,168
max_pooling2d_27 (MaxPooling2D)	(None, 4, 4, 256)	0
flatten_6 (Flatten)	(None, 4096)	0
dense_12 (Dense)	(None, 512)	2,097,664
dropout_4 (Dropout)	(None, 512)	0
dense_13 (Dense)	(None, 10)	5,130

Total params: 2,490,634 (9.50 MB)
Trainable params: 2,490,634 (9.50 MB)
Non-trainable params: 0 (0.00 B)




```
Epoch 1/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 70ms/step - accuracy: 0.1231 - loss: 2.3095
Epoch 2/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.1046 - loss: 2.2964
Epoch 3/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1102 - loss: 2.2858
Epoch 4/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1138 - loss: 2.2935
Epoch 5/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1205 - loss: 2.2922
Epoch 6/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1172 - loss: 2.2914
Epoch 7/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1240 - loss: 2.2845
Epoch 8/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1282 - loss: 2.2930
Epoch 9/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1302 - loss: 2.2872
Epoch 10/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1327 - loss: 2.2890
Epoch 11/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1076 - loss: 2.2991
Epoch 12/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1058 - loss: 2.2868
Epoch 13/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.0874 - loss: 2.2905
Epoch 14/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1093 - loss: 2.2854
Epoch 15/50
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.0992 - loss: 2.2870
Epoch 16/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1126 - loss: 2.2885
Epoch 17/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1036 - loss: 2.2947
Epoch 18/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1039 - loss: 2.2946
Epoch 19/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.0956 - loss: 2.2894
Epoch 20/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1341 - loss: 2.2934
Epoch 21/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.0984 - loss: 2.2939
Epoch 22/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1210 - loss: 2.2870
Epoch 23/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1282 - loss: 2.2891
Epoch 24/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1298 - loss: 2.2896
Epoch 25/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1135 - loss: 2.2883
Epoch 26/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1456 - loss: 2.2884
Epoch 27/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1027 - loss: 2.3042
Epoch 28/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1026 - loss: 2.2949
Epoch 29/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1240 - loss: 2.2866
Epoch 30/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1269 - loss: 2.2871
Epoch 31/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1106 - loss: 2.2888
Epoch 32/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1131 - loss: 2.2859
Epoch 33/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.1074 - loss: 2.2952
Epoch 34/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.1333 - loss: 2.2849
Epoch 35/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.1182 - loss: 2.2937
Epoch 36/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.1140 - loss: 2.2925
Epoch 37/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.1347 - loss: 2.2856
Epoch 38/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.1057 - loss: 2.2932
Epoch 39/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.1115 - loss: 2.2858
Epoch 40/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.1140 - loss: 2.2971
Epoch 41/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.1209 - loss: 2.2898
Epoch 42/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.1152 - loss: 2.2878
Epoch 43/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1058 - loss: 2.2863
Epoch 44/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1388 - loss: 2.2838
Epoch 45/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1057 - loss: 2.2932
Epoch 46/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1211 - loss: 2.2879
Epoch 47/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1098 - loss: 2.2855
Epoch 48/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1060 - loss: 2.2854
Epoch 49/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1423 - loss: 2.2928
Epoch 50/50
```