7.CallingKerasCGAN

April 11, 2024

[1]:
```
%pwd
```

[1]:
```
'D:\\Desktop\\Deep Learning\\GAN for Face expression Classification\\Research'
```

[2]:
```
import os
```

[3]:
```
os.chdir("../")
```

[4]:
```
%pwd
```

[4]:
```
'D:\\Desktop\\Deep Learning\\GAN for Face expression Classification'
```

[5]:
```
import logging
from pathlib import Path
logging.basicConfig(
    # filename='extract_data.log',
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S'
)
```

[11]:
```
import os
from pathlib import Path
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from keras.models import load_model
import keras

# Define the directory where the models are saved
save_dir = Path(os.getcwd()) / "Model" / "CGAN"

# Load the generator model
generator_model_path = save_dir / "generator_model.keras"
generator = load_model(generator_model_path)

generator.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0003),
```

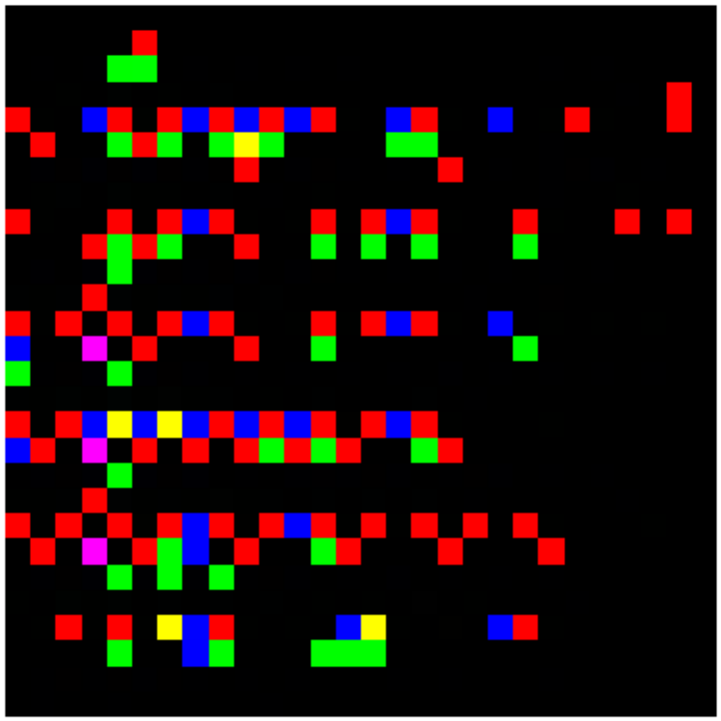
```
    loss=keras.losses.BinaryCrossentropy(from_logits=True)
)
```

```
D:\Desktop\Deep Learning\GAN for Face expression Classification\venv\Lib\site-
packages\keras\src\saving\saving_lib.py:418: UserWarning: Skipping variable
loading for optimizer 'adam', because it has 18 variables whereas the saved
optimizer has 2 variables.
  trackable.load_own_variables(weights_store.get(inner_path))
```

[12]:
```
# Function to generate images for a specified class
def generate_images_for_class(desired_class, num_interpolation=100):
    # Choose the latent dimension size
    latent_dim = generator.input_shape[1] - 10

    # Sample noise for the interpolation
    interpolation_noise = np.random.normal(size=(1, latent_dim))
    interpolation_noise = np.repeat(interpolation_noise,␣
 ↪repeats=num_interpolation, axis=0)

    # Convert the desired class to one-hot encoded vector
    label = np.zeros((1, 10))
    label[0, desired_class] = 1

    # Calculate the interpolation labels
    percent_second_label = np.linspace(0, 1, num_interpolation)[:, None]
    interpolation_labels = label * (1 - percent_second_label) + label *␣
 ↪percent_second_label

    # Combine noise and labels and run inference with the generator
    noise_and_labels = np.concatenate([interpolation_noise,␣
 ↪interpolation_labels], axis=1)
    fake_images = generator.predict(noise_and_labels)

    return fake_images

# Function to display and save the generated image
def display_and_save_image(image_array, save_path):
    # Convert the image array to a PIL Image
    image = Image.fromarray((255 * image_array).astype(np.uint8))

    # Display the image using matplotlib
    plt.imshow(image)
    plt.axis('off')
    plt.show()

    # Save the image
    image.save(save_path)
```

[14]:
```
# Generate and display the image for the desired class
desired_class = 1  # Change this to the desired class
fake_image = generate_images_for_class(desired_class)

# Define the directory to save the generated image
save_path = save_dir / f"generated_image_class_{desired_class}.png"

# Display and save the image
display_and_save_image(fake_image[0], save_path)
```

```
4/4             0s 13ms/step
```



[ ]: