Homework 2 – Image Generation

The goal of this work is to generate a series of images using conditional GANs. We will also use these new images as data augmentation for a classification problem and determine whether they will help improve accuracy.

The dataset contains 450 images of the game paper, rock, scissors (150 for each class). link: https://www.kaggle.com/datasets/sanikamal/rock-paper-scissors-dataset

cGAN Model

The cGAN architecture consists of two main components: the generator and the discriminator. The generator takes both random noise and conditional information as inputs and attempts to generate realistic samples that match the provided conditions. The discriminator, on the other hand, takes both real and generated samples along with the conditional information and aims to distinguish between them.

During training, the generator and discriminator are trained in a competitive manner. The generator tries to produce samples that fool the discriminator into classifying them as real, while the discriminator aims to correctly classify real samples as real and generated samples as fake. The conditional information helps guide the generation process, allowing the generator to produce samples that align with the desired conditions.

GAN Model-Generator

The Generator consists of three main components: the label_conditioned_generator, the latent layer, and the model.

The label_conditioned_generator is a sequential module. It starts with an Embedding layer, which maps categorical labels to continuous embeddings. It takes 3-dimensional inputs and converts them into 100-dimensional embeddings. The output of the Embedding layer is passed through a Linear layer, which maps the 100-dimensional embeddings to a 16-dimensional feature space. This component is responsible for incorporating label information into the generation process.

The latent layer is also a sequential module. It begins with a Linear layer, which takes 100-dimensional random noise vectors as input and maps them to a higher-dimensional space of 1024 features. The output of the Linear layer is then passed through a LeakyReLU activation function with a negative slope of 0.2. This component helps to introduce randomness and variability into the generated samples.

The model is the main component of the Generator and is responsible for transforming the latent representations into image-like samples. It is composed of a sequence of ConvTranspose2d, BatchNorm2d, ReLU, and Tanh layers. The ConvTranspose2d layers perform transposed convolution operations, which are used to upsample the input features and generate higher-resolution images. Each ConvTranspose2d layer is followed by a BatchNorm2d layer, which normalizes the features and helps stabilize the training process. The ReLU activation function is applied after each BatchNorm2d layer, introducing non-linearity and allowing the model to capture complex patterns. The final ConvTranspose2d layer has an output channel size of 3, corresponding to RGB images, and is followed by a Tanh activation function, which scales the output values between -1 and 1 to generate realistic images.

```
Generator(
  (label_conditioned_generator): Sequential(
    (0): Embedding(3, 100)
    (1): Linear(in_features=100, out_features=16, bias=True)
  )
  (latent): Sequential(
    (0): Linear(in_features=100, out_features=1024, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (model): Sequential(
    (0): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

GAN Model-Discriminator

The Discriminator consists of two main components: the label_condition_disc and the model.

The label_condition_disc is a sequential module. It starts with an Embedding layer, which maps categorical labels to continuous embeddings. It takes 3-dimensional inputs and converts them into 100-dimensional embeddings. The output of the Embedding layer is passed through a Linear layer, which maps the 100-dimensional embeddings to a high-dimensional space of 49,152 features. This component is responsible for incorporating label information into the discrimination process.

The model is the main component of the Discriminator and is responsible for classifying the input as real or generated. It is composed of a sequence of Conv2d, BatchNorm2d, LeakyReLU, Flatten, Dropout, Linear, and Sigmoid layers. The Conv2d layers perform convolution operations on the input images, extracting features at different scales. Each Conv2d layer is followed by a BatchNorm2d layer, which normalizes the features and helps stabilize the training process. The LeakyReLU activation function with a negative slope of 0.2 is applied after each BatchNorm2d layer, introducing non-linearity and allowing the model to capture complex patterns. The Flatten layer reshapes the output from the convolutional layers into a 1-dimensional vector. The Dropout layer randomly sets elements of the vector to zero during training, preventing overfitting. The Linear layer maps the flattened vector to a single output, indicating the probability that the input is real.

The output is then passed through a Sigmoid activation function, scaling the output to a range between 0 and 1, representing the probability of the input being real.

```
Discriminator(
  (label_condition_disc): Sequential(
    (0): Embedding(3, 100)
    (1): Linear(in_features=100, out_features=49152, bias=True)
  )
  (model): Sequential(
    (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(3, 3), padding=(2, 2), bias=False)
    (1): BatchNorm2d(128, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
    (3): Conv2d(128, 256, kernel_size=(4, 4), stride=(3, 3), padding=(2, 2), bias=False)
    (4): BatchNorm2d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (5): LeakyReLU(negative_slope=0.2, inplace=True)
    (6): Conv2d(256, 512, kernel_size=(4, 4), stride=(3, 3), padding=(2, 2), bias=False)
    (7): BatchNorm2d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (8): LeakyReLU(negative_slope=0.2, inplace=True)
    (9): Flatten(start_dim=1, end_dim=-1)
    (10): Dropout(p=0.4, inplace=False)
    (11): Linear(in_features=4608, out_features=1, bias=True)
    (12): Sigmoid()
  )
)
```

Training

I used a Adam optimizer with a learning rate (lr) of 0.0002 and a momentum of 0.1.
Beta parameters: 0.5 and 0.999. Loss: Binary Cross Entropy Loss. The Batch size is equal to 18.

Examples of generated images

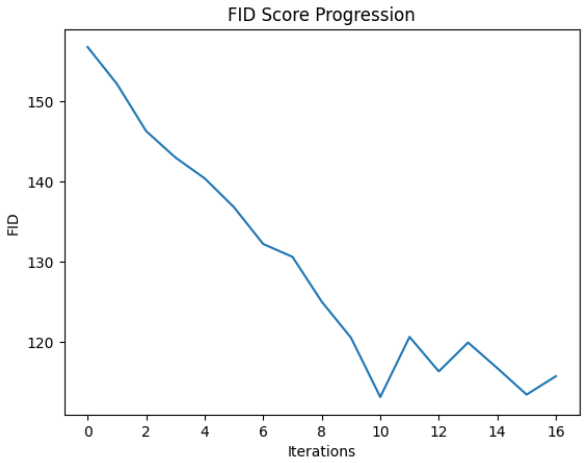
Paper:

Rock:

Scissors:

FID e IS

The function calculate_fid_given_paths is used for calculate FID. It takes two paths as arguments: the path to the directory containing real images and the path to the temporary directory containing generated images. FID depends on the batch size (18) and the number of iterations (17).



IS= 2.48

GAN-augmented classification results

The architecture of the model used is the same as the first model of the first homework but with fewer parameters.

```
CNN(
  (conv_layer): Sequential(
    (0): Conv2d(3, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (7): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): ReLU()
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc_layers): Sequential(
    (0): Linear(in_features=16384, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=3, bias=True)
  )
)
```

No difference in results with and without GAN-based data augmentation:

Using original dataset:
test accuracy: 1.00

Using generated data:
test accuracy: 1.00