assignment1

April 29, 2023

[46]:
```
# CS614 Assignment #1
## Vision
### Author: Cory W. Mauer
```

0.1 Pitch:

My Original idea for this project was inspired when scrolling through instagram looking at popular tatoo artists work. The challenge with this is I am often looking for ideas for work, without wanting to make a direct copy of something that already exists. Based on this idea I theorized the idea of building web site that enabled users to view autogenerated art images as a means for inspiration toward their own artwork.

0.2 Data source:

During my initial investigation of data sets to use for this project I initial was looking to either track down an existing data set of tattoo related art, however unsuprisingly I was not able to find any.

Following that discovery I decided I wanted to track down more general art related data. While I was able to find some, most had inconsistent large and/or inconsistent image sizing which made the process more difficult.

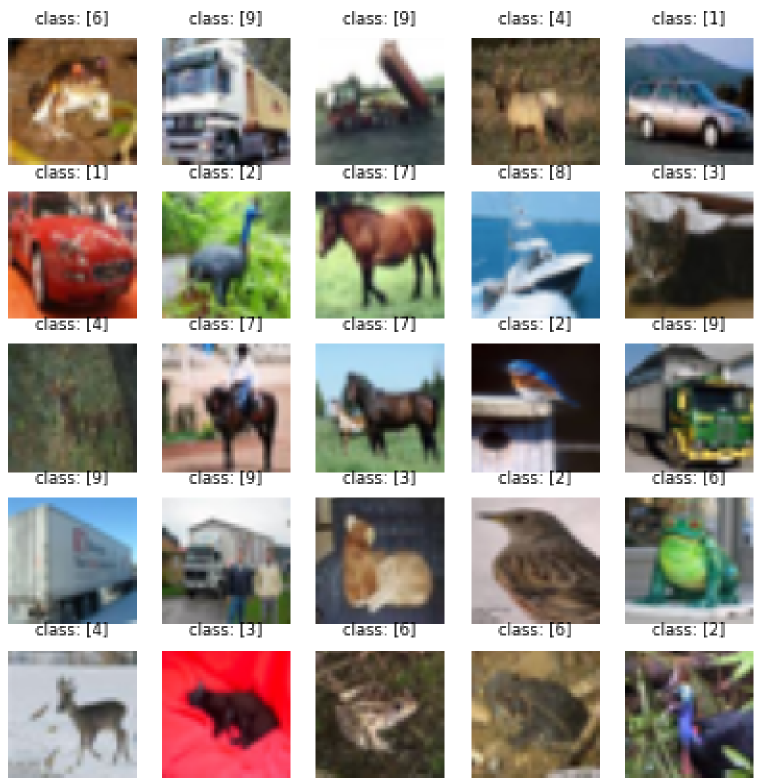
For this project I finally decided to leverage the Ciphar-10 data set (https://www.tensorflow.org/datasets/catalog/cifar10) as it provided my with an easily accessible large, color, and tagged dataset leveraging a wide variety of classes. Originally I expected very poor performance when using this data set, however after using it I was happily surprised by the results. They say that art imitates life, and in this case the combination of data from seemingly randomly classes tends to actually produce some very interesting images.

here are same examples from the data set with class labels

[47]:
```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from matplotlib import pyplot as plt


(train_images, train_labels), (_, _) = cifar10.load_data()
```

```python
figure_size = {"length": 5, "width":5}
fig = plt.figure(figsize=(figure_size["length"],figure_size["width"]))
for i in range(figure_size["length"]* figure_size["width"]):
    plt.subplot(figure_size["length"], figure_size["width"], i+1)
    plt.title(f"class: {train_labels[i]}", fontsize=6)
    plt.imshow((train_images[i, :, :, :]))
    plt.axis('off')
```



Given more time and compute resources, I would like to build or aquire larger and more targeted datasets. With these larger data sets I would hope to build higher resolution artwork that can better target specific use cases.

0.3 Model and Data Justification

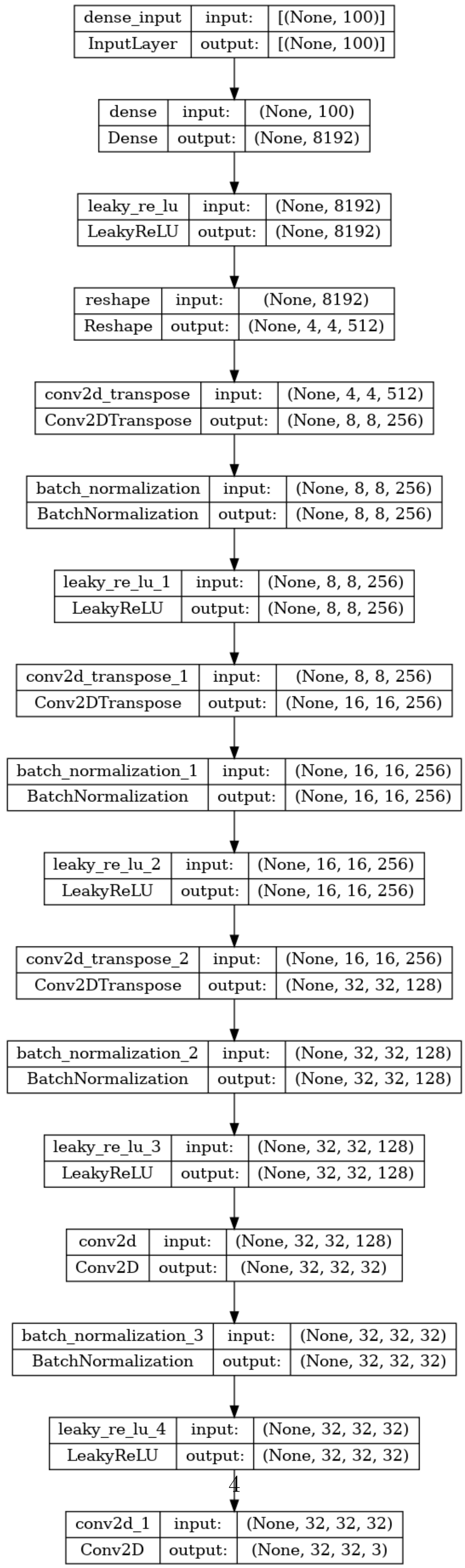
For this task I chose to use a GAN for the image generation. Prior to this project I had read about GAN's at a high level but never actually implemented one. Given more time and compute resources I would have liked to have explored a more cutting edge model such as a diffusion model.

During my GAN development, I orignally started with a simple GAN consisting of convolutional transpose layers with relatively few chanels.

best gan generator

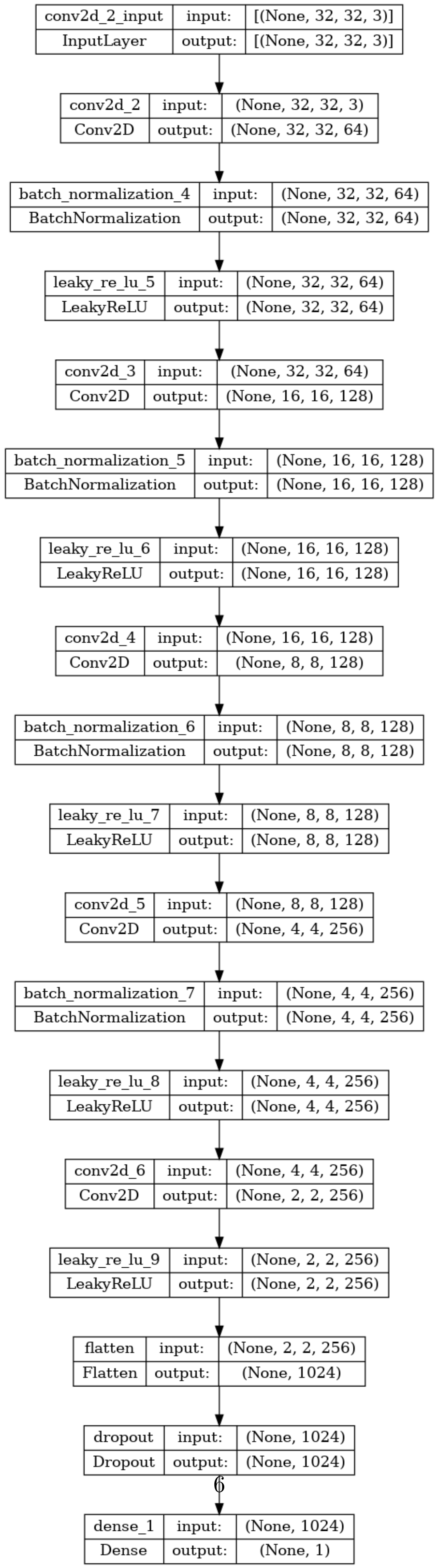
[48]:

```
from IPython.display import display, Image
display(Image(filename="./model_plots/generator_plot.png", height=400,␣
↪width=400))
```


dense_input | input: | [(None, 100)]
InputLayer | output: | [(None, 100)]
dense | input: | (None, 100)
Dense | output: | (None, 8192)
leaky_re_lu | input: | (None, 8192)
LeakyReLU | output: | (None, 8192)
reshape | input: | (None, 8192)
Reshape | output: | (None, 4, 4, 512)
conv2d_transpose | input: | (None, 4, 4, 512)
Conv2DTranspose | output: | (None, 8, 8, 256)
batch_normalization | input: | (None, 8, 8, 256)
BatchNormalization | output: | (None, 8, 8, 256)
leaky_re_lu_1 | input: | (None, 8, 8, 256)
LeakyReLU | output: | (None, 8, 8, 256)
conv2d_transpose_1 | input: | (None, 8, 8, 256)
Conv2DTranspose | output: | (None, 16, 16, 256)
batch_normalization_1 | input: | (None, 16, 16, 256)
BatchNormalization | output: | (None, 16, 16, 256)
leaky_re_lu_2 | input: | (None, 16, 16, 256)
LeakyReLU | output: | (None, 16, 16, 256)
conv2d_transpose_2 | input: | (None, 16, 16, 256)
Conv2DTranspose | output: | (None, 32, 32, 128)
batch_normalization_2 | input: | (None, 32, 32, 128)
BatchNormalization | output: | (None, 32, 32, 128)
leaky_re_lu_3 | input: | (None, 32, 32, 128)
LeakyReLU | output: | (None, 32, 32, 128)
conv2d | input: | (None, 32, 32, 128)
Conv2D | output: | (None, 32, 32, 32)
batch_normalization_3 | input: | (None, 32, 32, 32)
BatchNormalization | output: | (None, 32, 32, 32)
leaky_re_lu_4 | input: | (None, 32, 32, 32)
LeakyReLU | output: | (None, 32, 32, 32)
conv2d_1 | input: | (None, 32, 32, 32)
Conv2D | output: | (None, 32, 32, 3)

best gan discriminator

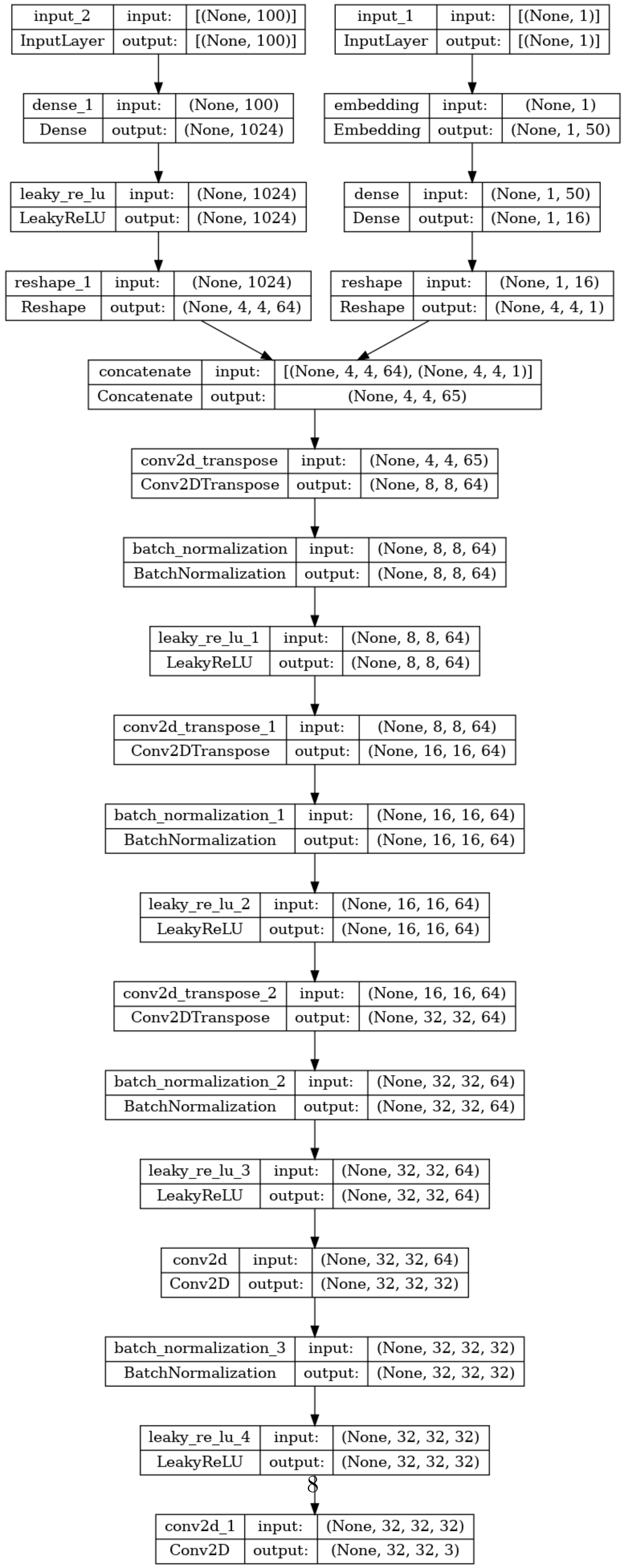
[49]:
```
display(Image(filename="./model_plots/discriminator_plot.png", height=400,␣
↪width=400))
```


conv2d_2_input
InputLayer
input:
output:
[(None, 32, 32, 3)]
[(None, 32, 32, 3)]
conv2d_2
Conv2D
input:
output:
(None, 32, 32, 3)
(None, 32, 32, 64)
batch_normalization_4
BatchNormalization
input:
output:
(None, 32, 32, 64)
(None, 32, 32, 64)
leaky_re_lu_5
LeakyReLU
input:
output:
(None, 32, 32, 64)
(None, 32, 32, 64)
conv2d_3
Conv2D
input:
output:
(None, 32, 32, 64)
(None, 16, 16, 128)
batch_normalization_5
BatchNormalization
input:
output:
(None, 16, 16, 128)
(None, 16, 16, 128)
leaky_re_lu_6
LeakyReLU
input:
output:
(None, 16, 16, 128)
(None, 16, 16, 128)
conv2d_4
Conv2D
input:
output:
(None, 16, 16, 128)
(None, 8, 8, 128)
batch_normalization_6
BatchNormalization
input:
output:
(None, 8, 8, 128)
(None, 8, 8, 128)
leaky_re_lu_7
LeakyReLU
input:
output:
(None, 8, 8, 128)
(None, 8, 8, 128)
conv2d_5
Conv2D
input:
output:
(None, 8, 8, 128)
(None, 4, 4, 256)
batch_normalization_7
BatchNormalization
input:
output:
(None, 4, 4, 256)
(None, 4, 4, 256)
leaky_re_lu_8
LeakyReLU
input:
output:
(None, 4, 4, 256)
(None, 4, 4, 256)
conv2d_6
Conv2D
input:
output:
(None, 4, 4, 256)
(None, 2, 2, 256)
leaky_re_lu_9
LeakyReLU
input:
output:
(None, 2, 2, 256)
(None, 2, 2, 256)
flatten
Flatten
input:
output:
(None, 2, 2, 256)
(None, 1024)
dropout
Dropout
input:
output:
(None, 1024)
(None, 1024)
dense_1
Dense
input:
output:
(None, 1024)
(None, 1)

I did start exploring using classification in effort to create a conditional GAN, but didn't have great success with training.

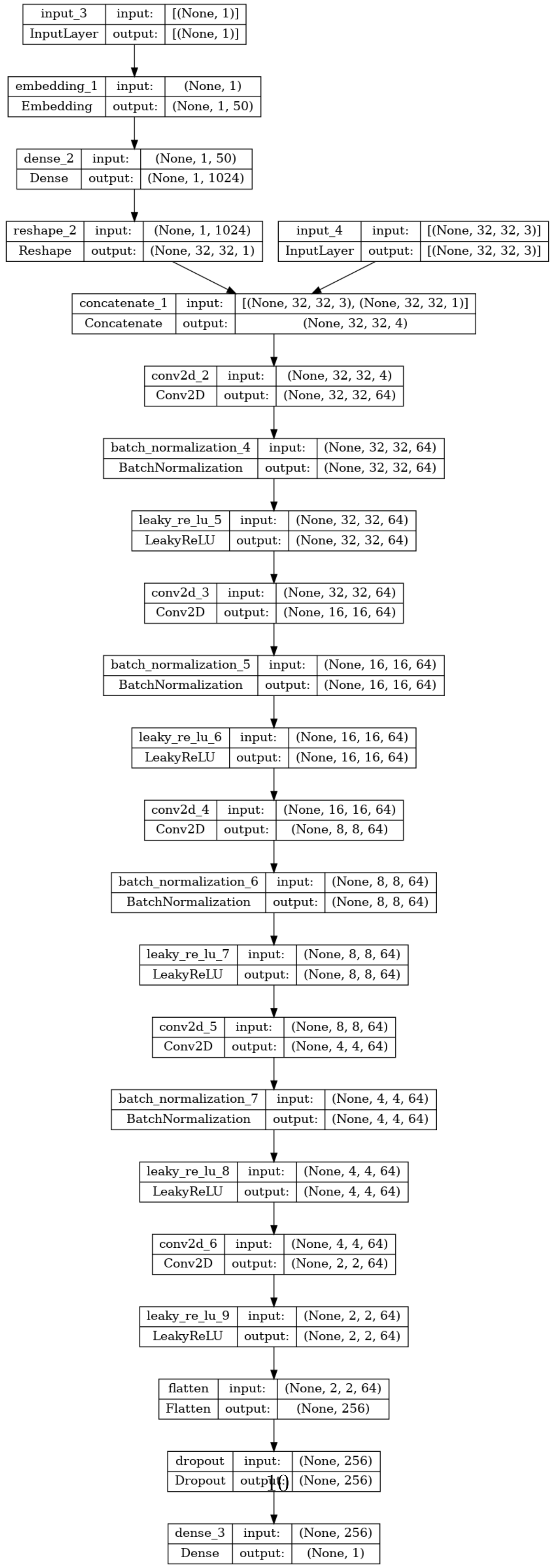
cgan generator

[50]:
```
display(Image(filename="./model_plots/cgan_gen_plot.png", height=400,␣
↪width=400))
```


input_2
InputLayer
input:
output:
[(None, 100)]
[(None, 100)]
input_1
InputLayer
input:
output:
[(None, 1)]
[(None, 1)]
dense_1
Dense
input:
output:
(None, 100)
(None, 1024)
embedding
Embedding
input:
output:
(None, 1)
(None, 1, 50)
leaky_re_lu
LeakyReLU
input:
output:
(None, 1024)
(None, 1024)
dense
Dense
input:
output:
(None, 1, 50)
(None, 1, 16)
reshape_1
Reshape
input:
output:
(None, 1024)
(None, 4, 4, 64)
reshape
Reshape
input:
output:
(None, 1, 16)
(None, 4, 4, 1)
concatenate
Concatenate
input:
output:
[(None, 4, 4, 64), (None, 4, 4, 1)]
(None, 4, 4, 65)
conv2d_transpose
Conv2DTranspose
input:
output:
(None, 4, 4, 65)
(None, 8, 8, 64)
batch_normalization
BatchNormalization
input:
output:
(None, 8, 8, 64)
(None, 8, 8, 64)
leaky_re_lu_1
LeakyReLU
input:
output:
(None, 8, 8, 64)
(None, 8, 8, 64)
conv2d_transpose_1
Conv2DTranspose
input:
output:
(None, 8, 8, 64)
(None, 16, 16, 64)
batch_normalization_1
BatchNormalization
input:
output:
(None, 16, 16, 64)
(None, 16, 16, 64)
leaky_re_lu_2
LeakyReLU
input:
output:
(None, 16, 16, 64)
(None, 16, 16, 64)
conv2d_transpose_2
Conv2DTranspose
input:
output:
(None, 16, 16, 64)
(None, 32, 32, 64)
batch_normalization_2
BatchNormalization
input:
output:
(None, 32, 32, 64)
(None, 32, 32, 64)
leaky_re_lu_3
LeakyReLU
input:
output:
(None, 32, 32, 64)
(None, 32, 32, 64)
conv2d
Conv2D
input:
output:
(None, 32, 32, 64)
(None, 32, 32, 32)
batch_normalization_3
BatchNormalization
input:
output:
(None, 32, 32, 32)
(None, 32, 32, 32)
leaky_re_lu_4
LeakyReLU
input:
output:
(None, 32, 32, 32)
(None, 32, 32, 32)
conv2d_1
Conv2D
input:
output:
(None, 32, 32, 32)
(None, 32, 32, 3)

cgan discriminator

[51]:
```
display(Image(filename="./model_plots/cgan_discriminator_plot.png", height=400,␣
↪width=400))
```


input_3 | input: | [(None, 1)]
InputLayer | output: | [(None, 1)]
embedding_1 | input: | (None, 1)
Embedding | output: | (None, 1, 50)
dense_2 | input: | (None, 1, 50)
Dense | output: | (None, 1, 1024)
reshape_2 | input: | (None, 1, 1024)
Reshape | output: | (None, 32, 32, 1)
input_4 | input: | [(None, 32, 32, 3)]
InputLayer | output: | [(None, 32, 32, 3)]
concatenate_1 | input: | [(None, 32, 32, 3), (None, 32, 32, 1)]
Concatenate | output: | (None, 32, 32, 4)
conv2d_2 | input: | (None, 32, 32, 4)
Conv2D | output: | (None, 32, 32, 64)
batch_normalization_4 | input: | (None, 32, 32, 64)
BatchNormalization | output: | (None, 32, 32, 64)
leaky_re_lu_5 | input: | (None, 32, 32, 64)
LeakyReLU | output: | (None, 32, 32, 64)
conv2d_3 | input: | (None, 32, 32, 64)
Conv2D | output: | (None, 16, 16, 64)
batch_normalization_5 | input: | (None, 16, 16, 64)
BatchNormalization | output: | (None, 16, 16, 64)
leaky_re_lu_6 | input: | (None, 16, 16, 64)
LeakyReLU | output: | (None, 16, 16, 64)
conv2d_4 | input: | (None, 16, 16, 64)
Conv2D | output: | (None, 8, 8, 64)
batch_normalization_6 | input: | (None, 8, 8, 64)
BatchNormalization | output: | (None, 8, 8, 64)
leaky_re_lu_7 | input: | (None, 8, 8, 64)
LeakyReLU | output: | (None, 8, 8, 64)
conv2d_5 | input: | (None, 8, 8, 64)
Conv2D | output: | (None, 4, 4, 64)
batch_normalization_7 | input: | (None, 4, 4, 64)
BatchNormalization | output: | (None, 4, 4, 64)
leaky_re_lu_8 | input: | (None, 4, 4, 64)
LeakyReLU | output: | (None, 4, 4, 64)
conv2d_6 | input: | (None, 4, 4, 64)
Conv2D | output: | (None, 2, 2, 64)
leaky_re_lu_9 | input: | (None, 2, 2, 64)
LeakyReLU | output: | (None, 2, 2, 64)
flatten | input: | (None, 2, 2, 64)
Flatten | output: | (None, 256)
dropout | input: | (None, 256)
Dropout | output: | (None, 256)
dense_3 | input: | (None, 256)
Dense | output: | (None, 1)

0.4 Commented examples

The following provides some commented code snip its of the model. for more details see the training script `cifar10_gan.py`

[52]:
```python
from tensorflow.keras import layers
# Generator model
def make_generator_model():
    model = tf.keras.Sequential()
    n_nodes = [4, 4, 512]
    model.add(layers.Dense(n_nodes[0] * n_nodes[1]* n_nodes[2], input_dim=100))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Reshape(n_nodes))
    model.add(layers.Conv2DTranspose(256, (4,4), strides=(2,2), padding='same'))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Conv2DTranspose(256, (4,4), strides=(2,2), padding='same'))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Conv2D(32, (3,3), padding='same'))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Conv2D(3, (3,3), activation='tanh', padding='same'))
    return model

generator = make_generator_model()
print(f"generator \n {generator.summary()}")
```

```
Model: "sequential_8"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 8192)              827392

 leaky_re_lu_40 (LeakyReLU)  (None, 8192)              0

 reshape_4 (Reshape)         (None, 4, 4, 512)         0

 conv2d_transpose_12 (Conv2D  (None, 8, 8, 256)        2097408
 Transpose)

 batch_normalization_32 (Bat  (None, 8, 8, 256)        1024
```

```
 chNormalization)

 leaky_re_lu_41 (LeakyReLU)  (None, 8, 8, 256)         0

 conv2d_transpose_13 (Conv2D  (None, 16, 16, 256)      1048832
 Transpose)

 batch_normalization_33 (Bat  (None, 16, 16, 256)      1024
 chNormalization)

 leaky_re_lu_42 (LeakyReLU)  (None, 16, 16, 256)       0

 conv2d_transpose_14 (Conv2D  (None, 32, 32, 128)      524416
 Transpose)

 batch_normalization_34 (Bat  (None, 32, 32, 128)      512
 chNormalization)

 leaky_re_lu_43 (LeakyReLU)  (None, 32, 32, 128)       0

 conv2d_28 (Conv2D)          (None, 32, 32, 32)        36896

 batch_normalization_35 (Bat  (None, 32, 32, 32)       128
 chNormalization)

 leaky_re_lu_44 (LeakyReLU)  (None, 32, 32, 32)        0

 conv2d_29 (Conv2D)          (None, 32, 32, 3)         867

=================================================================
Total params: 4,538,499
Trainable params: 4,537,155
Non-trainable params: 1,344
_________________________________________________________________
generator
 None
```

[53]:

```
# Discriminator model
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (3,3), padding='same', input_shape=(32,32,3)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Conv2D(128, (3,3), strides=(2,2), padding='same'))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Conv2D(128, (3,3), strides=(2,2), padding='same'))
```

```python
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Conv2D(256, (3,3), strides=(2,2), padding='same'))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Conv2D(256, (3,3), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Flatten())
    model.add(layers.Dropout(0.25))
    model.add(layers.Dense(1, activation='sigmoid'))
    return model

discriminator = make_discriminator_model()
print(f"discriminator \n {discriminator.summary()}")
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_30 (Conv2D)          (None, 32, 32, 64)        1792

 batch_normalization_36 (Bat  (None, 32, 32, 64)       256
 chNormalization)

 leaky_re_lu_45 (LeakyReLU)  (None, 32, 32, 64)        0

 conv2d_31 (Conv2D)          (None, 16, 16, 128)       73856

 batch_normalization_37 (Bat  (None, 16, 16, 128)      512
 chNormalization)

 leaky_re_lu_46 (LeakyReLU)  (None, 16, 16, 128)       0

 conv2d_32 (Conv2D)          (None, 8, 8, 128)         147584

 batch_normalization_38 (Bat  (None, 8, 8, 128)        512
 chNormalization)

 leaky_re_lu_47 (LeakyReLU)  (None, 8, 8, 128)         0

 conv2d_33 (Conv2D)          (None, 4, 4, 256)         295168

 batch_normalization_39 (Bat  (None, 4, 4, 256)        1024
 chNormalization)

 leaky_re_lu_48 (LeakyReLU)  (None, 4, 4, 256)         0
```

```
 conv2d_34 (Conv2D)          (None, 2, 2, 256)         590080

 leaky_re_lu_49 (LeakyReLU)  (None, 2, 2, 256)         0

 flatten_4 (Flatten)         (None, 1024)              0

 dropout_4 (Dropout)         (None, 1024)              0

 dense_9 (Dense)             (None, 1)                 1025

=================================================================
Total params: 1,111,809
Trainable params: 1,110,657
Non-trainable params: 1,152
_________________________________________________________________
discriminator
 None
```

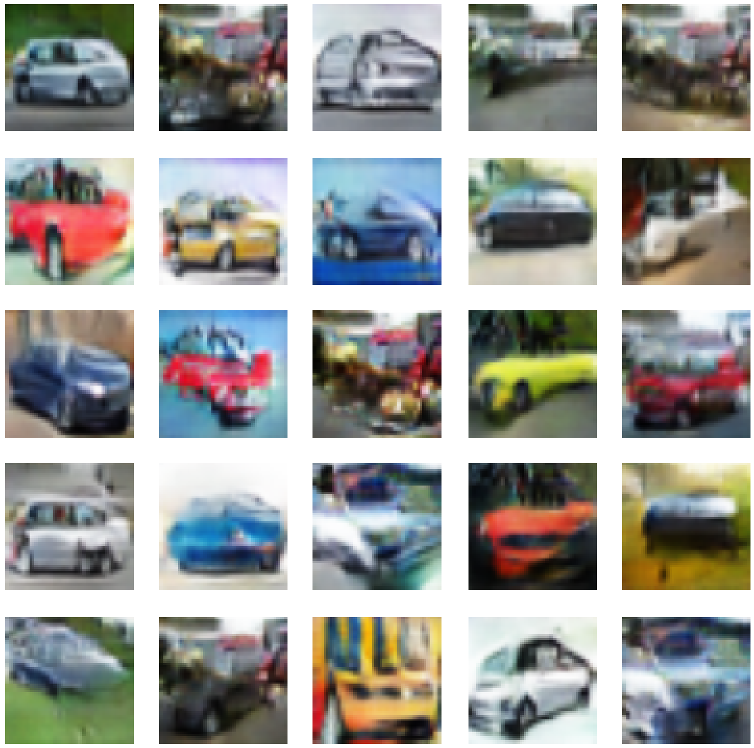
0.5 Testing

Initial testing using these gan models focused on training data from a single class from the cifar data set. In this case we will demonstrate the model being trained with automobile images. Notice the quality of the relatively high quality of the images that make it easy to tell that a car. The homogenous nature of the images all being cars allow for a model that cleary produces car images.

[54]:
```
generator = tf.keras.models.load_model("./car_generator")
figure_size = {"length": 5, "width": 5}
number_of_generated_images = figure_size['length'] * figure_size['width']
noise = tf.random.normal([number_of_generated_images, 100])
generated_images = generator.predict(noise)
generated_images = (generated_images * 127.5 + 127.5).astype(int)

fig = plt.figure(figsize=(figure_size["length"],figure_size["width"]), dpi= 100)
for i in range(figure_size["length"]* figure_size["width"]):
    plt.subplot(figure_size["length"], figure_size["width"], i+1)
    plt.imshow((generated_images[i, :, :, :]))
    plt.axis('off')
```

```
WARNING:tensorflow:No training configuration found in save file, so the model
was *not* compiled. Compile it manually.
1/1 [==============================] - 0s 134ms/step
```

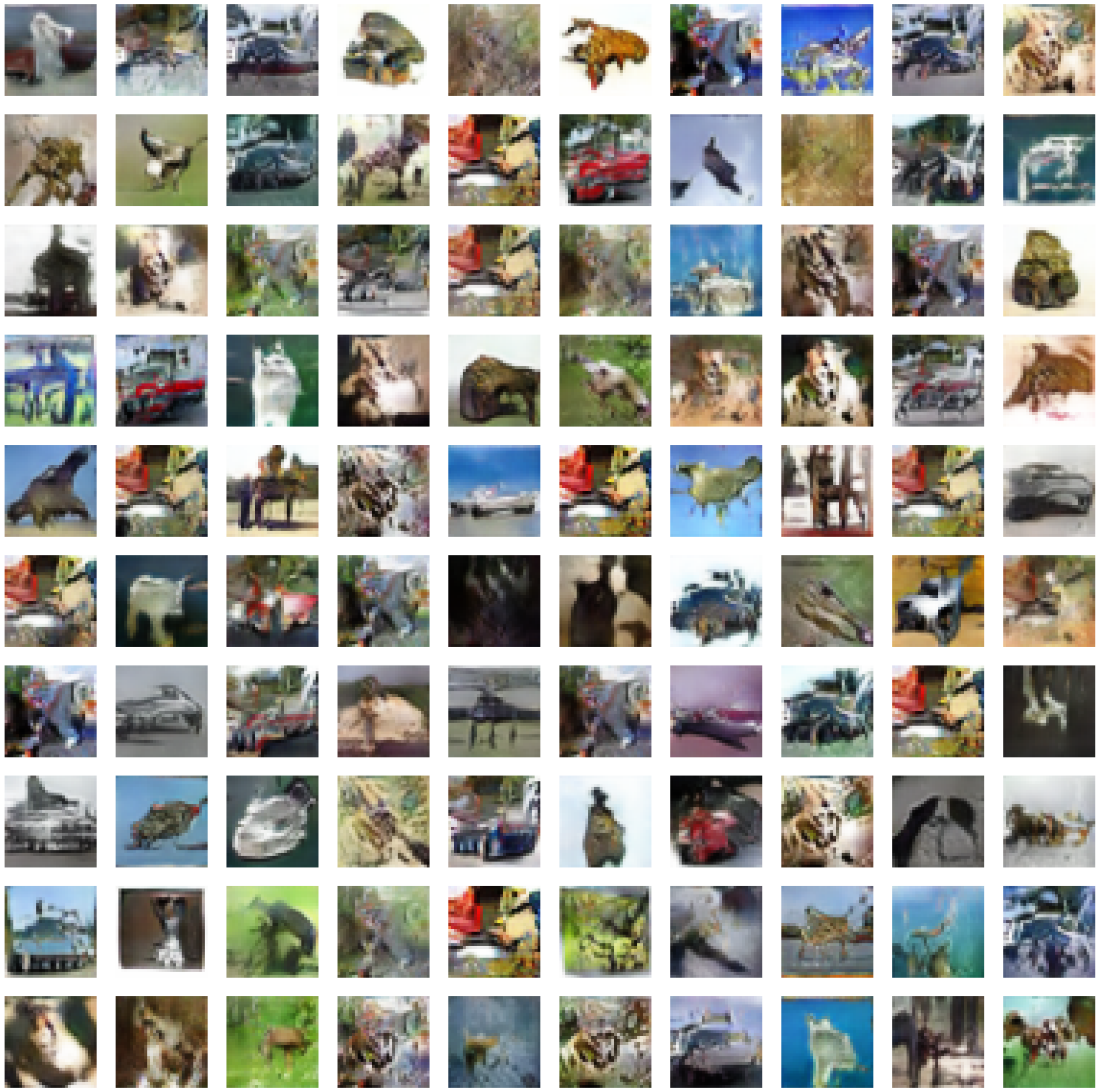


After this I decided look into how I could make artistic looking images. I found that by training my models with all classes of image from the cifar dataset I was able to create images that mix features of each of the classes and make unique but familiar images.

[55]:
```
generator = tf.keras.models.load_model("./gan10_generator")
figure_size = {"length": 10, "width":10}
number_of_generated_images = figure_size['length'] * figure_size['width']
noise = tf.random.normal([number_of_generated_images, 100])
generated_images = generator.predict(noise)
generated_images = (generated_images * 127.5 + 127.5).astype(int)

fig = plt.figure(figsize=(figure_size["length"],figure_size["width"]), dpi= 800)
for i in range(figure_size["length"]* figure_size["width"]):
    plt.subplot(figure_size["length"], figure_size["width"], i+1)
    plt.imshow((generated_images[i, :, :, :]))
    plt.axis('off')
```

```
WARNING:tensorflow:No training configuration found in save file, so the model
was *not* compiled. Compile it manually.
4/4 [==============================] - 0s 20ms/step
```



0.6 Code and instructions to run it

Code can be found on my github at cs614(https://github.com/cwma86/cs614) The model training code exists in `./cifar10_gan.py`. the pre-trained model can be found at `gan10_generator` and `gan10_discriminator`

to add more training epoch run

```
./cifar10_gan.py --gen gan10_generator --disc gan10_discriminator
```

to see super resolved version of the generated images

```
./super_resolution.py gan10_generator
```