CAP6610 Project Report 3

Justin Ho
justinho@ufl.edu

March 21, 2023

1 Introduction

As a reminder from the last report, I modified an existing Conditional GAN (CGAN) to generate images from the CIFAR-10 dataset with surprisingly high fidelity. However, when I went to go find a Conditional VAE (CVAE) to use for experimentation, there was no modern code of a CVAE for TensorFlow. A majority of the code online was written for PyTorch and the code that was for TensorFlow (TF) was for TF1 which is dramatically different from the current API version which is 2. So these past weeks I've spent time learning the TF 2 API and transforming an existing modern TF VAE example into a CVAE [2].

2 Transforming a VAE to a CVAE

I've used TensorFlow before for some very basic projects, and I have modified models before, but to convert a VAE to a CVAE, I knew it would be more involved than just changing the model: it would involve modifying the training loop as well. As a reminder, the CVAE is different from the traditional VAE in the sense that the label of the image is used as a "condition" that is passed to the encoder and the decoder [1]. This extra information from the label is crucial as it allows the CVAE to learn which distribution to draw from when it sees a particular class. With a fully trained CVAE, all you would need to do is to pass the latent space, and a class label and hypothetically the CVAE would generate an image of the class you selected. This sounds easy in theory but this process was quite involved and required a lot of fine-tuning to get the model to even run.

To approach this I explored the TF 2 API and found that there were three different ways to train a model: using the default train_step, subclassing your train_step from the TensorFlow model API, and implementing a train_step that takes a model. The example that I began modifying used the latter technique [2]. To start, I had to modify the train_step function to take in a tf.dataset that had both the images AND the label. The prior code only took in the label. I then had to modify the model input sizes to take in the one-hot-encoded condition for both the encoder and the decoder. Once that was

done I had to do some clever data manipulation to transform the data into the correct form for the model to process end to end. This process was extremely laborious so I'll spare you the details but I was able to get great output with both models.

3 A broad comparison of the CGAN and CVAE

Both the CGAN and the CVAE took a while to train with the CVAE taking 51 seconds and the CGAN taking 40 seconds. The CGAN required far fewer epochs to get discernable results than the CVAE which required quite a bit of time to get good results at 20 epochs versus 75 epochs. It also seems that there is much more available documentation and research into the CGAN than the CVAE which made it much harder to implement and optimize. However, regardless of the downsides of the CVAE, once I got it working it was quite interesting to explore and experiment with the model to see what features it has learned.

4 Output Image Analysis

The images produced by the CVAE were typical of prior VAEs. Typically, the images produced by the VAE were blurrier than their GAN counterparts and this trend holds for the CVAE. However, one advantage that is working in the CVAE's favor is that it does not suffer from the artifacts that are present in the CGAN's images. I also noticed that editing the latent space did not yield much of a difference until you got under 80 in which the images were indiscernible. I thought the images produced by the CVAE were better than that of the CGAN though the loss of sharpness sometimes made the images produced hard to distinguish. You can find all the results talked about here in the figures attached in the appendix.

5 Conclusion and Next Steps

Overall I am pretty pleased and proud of what I was able to accomplish with this report. I felt like my understanding of GANs and VAEs improved significantly, especially regarding their implementation and behavior. I learned a lot about how TF works and I also learned quite a bit about how to take an existing model and heavily augment it for a different task. I am proud to say that once I upload my code after this semester is over, I will be among the first to have uploaded a TF 2 example with the CIFAR-10 dataset. In the last report, I said I was going to explore the StyleGAN architecture but there is no VAE counterpart so I will pivot toward implementing metrics like FID now that I am generating images that are not MNIST. To make it clear my goal from now to the end of the semester is to run these "mini-experiments" and explore the different classes of GANS and VAE and how they compare. Once I have a rough picture of the field for the last report I will combine all the techniques that I have learned and perform experimentation among all the techniques I have used to see the best combination of the bunch.

References

[1] Sohn, Kihyuk and Yan, Xinchen and Lee, Honglak, Learning Structured Output Representation Using Deep Conditional Generative Models, 2015, Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2, https://dl.acm.org/doi/10.5555/2969442.2969628

[2] https://www.tensorflow.org/tutorials/generative/cvae

6 Appendix

Code: https://colab.research.google.com/drive/1t_tgNITp57bjTzbuXphkmXOgj2q808vy?usp=sharing

Figure 1: On the left is a car generated by the CGAN and on the right is a car generated by the CVAE. Note the lack of artifacting and also note that the car image generated by the CVAE is much smoother than the one generated by the CGAN

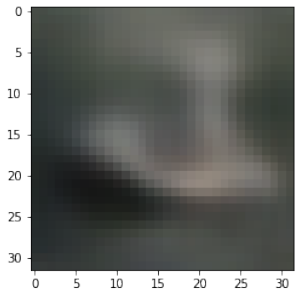


Figure 2: An image of a boat generated by the CVAE

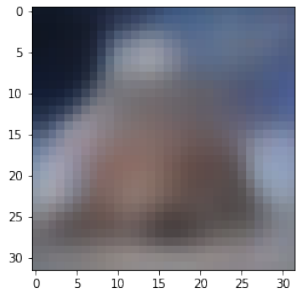


Figure 3: An image of a cat generated by the CVAE with a latent space of 70 and with 75 epochs. Note the cat is not discernable. All of the above results were generated with an epoch of 75 and a latent space of 110

```
!pip install -q git+https://github.com/tensorflow/docs
```

```
       Preparing metadata (setup.py) ... done
```

```
from tensorflow import keras
from tensorflow.keras import layers

from tensorflow_docs.vis import embed
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import imageio
```

```
batch_size = 32
num_channels = 3
num_classes = 10
image_size = 32
latent_dim = 128
```

```
# We'll use all the available examples from both the training and test
# sets.
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

test_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test))
test_dataset = test_dataset.shuffle(buffer_size=1024).batch(batch_size)

all_images = np.concatenate([x_train, x_test])
all_labels = np.concatenate([y_train, y_test])

# Scale the pixel values to [0, 1] range, add a channel dimension to
# the images, and one-hot encode the labels.
all_images = all_images.astype("float32") / 255.0
all_labels = keras.utils.to_categorical(all_labels, 10)

dataset = tf.data.Dataset.from_tensor_slices((all_images, all_labels))
dataset = dataset.shuffle(buffer_size=1024).batch(batch_size)

print(f"Shape of training images: {all_images.shape}")
print(f"Shape of training labels: {all_labels.shape}")
```

```
     Shape of training images: (60000, 32, 32, 3)
     Shape of training labels: (60000, 10)
```

```
generator_in_channels = latent_dim + num_classes
discriminator_in_channels = num_channels + num_classes
print(generator_in_channels, discriminator_in_channels)
```

```
     138 13
```

Conditional Variational Auto-Encoder

```
from IPython import display

import glob
import imageio
import matplotlib.pyplot as plt
import numpy as np
import PIL
import tensorflow as tf
import tensorflow_probability as tfp
import time
```

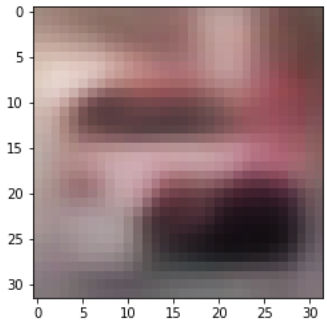
```
def generate_and_save_images(model, epoch, test_sample):
  mean, logvar = model.encode(test_sample)
  z = model.reparameterize(mean, logvar)
  predictions = model.sample(z)
  fig = plt.figure(figsize=(4, 4))

  for i in range(predictions.shape[0]):
    plt.subplot(4, 4, i + 1)
```

```
      plt.imshow(predictions[i, :, :, 0], cmap='gray')
      plt.axis('off')

    # tight_layout minimizes the overlap between 2 sub-plots
    plt.savefig('image_at_epoch_{:04d}_vae_uc.png'.format(epoch))
    plt.show()


  class CVAE(tf.keras.Model):
    """Convolutional variational autoencoder."""

    def __init__(self, latent_dim):
      super(CVAE, self).__init__()
      self.latent_dim = latent_dim
      self.encoder = tf.keras.Sequential(
          [
              tf.keras.layers.InputLayer(input_shape=((32, 32, discriminator_in_channels))),

              tf.keras.layers.Conv2D(
                  filters=32, kernel_size=4, strides=(2, 2), activation='relu'),
              tf.keras.layers.BatchNormalization(),
              tf.keras.layers.ReLU(),

              tf.keras.layers.Conv2D(
                  filters=32, kernel_size=4, strides=(2, 2), activation='relu'),
              tf.keras.layers.BatchNormalization(),
              tf.keras.layers.ReLU(),

              tf.keras.layers.Conv2D(
                  filters=32, kernel_size=4, strides=(2, 2), activation='relu'),
              tf.keras.layers.BatchNormalization(),
              tf.keras.layers.ReLU(),

              # No activation
              tf.keras.layers.Flatten(),
              tf.keras.layers.Dense(latent_dim + latent_dim),
              tf.keras.layers.Reshape(target_shape=(latent_dim * 2,)),
          ]
      )


      self.decoder = tf.keras.Sequential(
          [
              tf.keras.layers.InputLayer(input_shape=(latent_dim + num_classes,)),
              tf.keras.layers.Dense(units=8*8*32, activation=tf.nn.relu),
              tf.keras.layers.Reshape(target_shape=(8, 8, 32)),

              tf.keras.layers.UpSampling2D(size=(2, 2)),
              tf.keras.layers.Conv2D(192, (4, 4), strides=(1, 1), padding='same', activation='relu'),
              layers.BatchNormalization(),

              tf.keras.layers.UpSampling2D(size=(2, 2)),
              tf.keras.layers.Conv2D(96, (4, 4), strides=(1, 1), padding='same', activation='relu'),
              layers.BatchNormalization(),

              # No activation
              tf.keras.layers.Conv2D(3, (4, 4), strides=(1, 1), padding='same', use_bias=False)
          ]
      )

    @tf.function
    def sample(self, eps=None):
      if eps is None:
        eps = tf.random.normal(shape=(1, self.latent_dim))
      return self.decode(eps, apply_sigmoid=True)

    def encode(self, x):
      mean, logvar = tf.split(self.encoder(x), num_or_size_splits=2, axis=1)
      return mean, logvar

    def reparameterize(self, mean, logvar):
      eps = tf.random.normal(shape=mean.shape)
      return eps * tf.exp(logvar * .5) + mean

    def decode(self, z, apply_sigmoid=False):
      logits = self.decoder(z)
      if apply_sigmoid:
        probs = tf.sigmoid(logits)
```

```
        probs = tf.sigmoid(logits)
        return probs
      return logits


optimizer = tf.keras.optimizers.Adam(1e-4)


def log_normal_pdf(sample, mean, logvar, raxis=1):
  log2pi = tf.math.log(2. * np.pi)
  return tf.reduce_sum(
      -.5 * ((sample - mean) ** 2. * tf.exp(-logvar) + logvar + log2pi),
      axis=raxis)


def compute_loss(model, x, real, one_hot_labels):
  mean, logvar = model.encode(x)
  z = model.reparameterize(mean, logvar)
  z_comb = tf.concat([z, one_hot_labels], 1)
  x_logit = model.decode(z_comb)
  cross_ent = tf.nn.sigmoid_cross_entropy_with_logits(logits=x_logit, labels=real)
  logpx_z = -tf.reduce_sum(cross_ent, axis=[1, 2, 3])
  logpz = log_normal_pdf(z, 0., 0.)
  logqz_x = log_normal_pdf(z, mean, logvar)
  return -tf.reduce_mean(logpx_z + logpz - logqz_x)


@tf.function
def train_step(model, batch, optimizer):
  """Executes one training step and returns the loss.

  This function computes the loss and gradients, and uses the latter to
  update the model's parameters.
  """

  real_images, one_hot_labels = batch


  image_one_hot_labels = one_hot_labels[:, :, None, None]
  image_one_hot_labels = tf.repeat(
      image_one_hot_labels, repeats=[image_size * image_size]
  )
  image_one_hot_labels = tf.reshape(
      image_one_hot_labels, (-1, image_size, image_size, num_classes)
  )
  X = tf.concat([real_images, image_one_hot_labels], -1)
  with tf.GradientTape() as tape:
    loss = compute_loss(model, X, real_images, one_hot_labels)
  gradients = tape.gradient(loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))


epochs = 100
# set the dimensionality of the latent space to a plane for visualization later
latent_dim = 110

model = CVAE(latent_dim)


for epoch in range(1, epochs + 1):
  start_time = time.time()
  for batch in dataset:
    train_step(model, batch, optimizer)
  end_time = time.time()
  print('Epoch: {}, time elapse for current epoch: {}'
    .format(epoch, end_time - start_time))
```

```
Epoch: 44, time elapse for current epoch: 51.15254545211792
Epoch: 45, time elapse for current epoch: 51.13719701766968
Epoch: 46, time elapse for current epoch: 51.154871225357056
Epoch: 47, time elapse for current epoch: 51.14214253425598
Epoch: 48, time elapse for current epoch: 51.15955686569214
Epoch: 49, time elapse for current epoch: 51.13997793197632
Epoch: 50, time elapse for current epoch: 51.12781858444214
Epoch: 51, time elapse for current epoch: 51.09955167770386
Epoch: 52, time elapse for current epoch: 51.09057354927063
Epoch: 53, time elapse for current epoch: 51.09539842605591
Epoch: 54, time elapse for current epoch: 51.09111142158508
Epoch: 55, time elapse for current epoch: 51.09191703796387
Epoch: 56, time elapse for current epoch: 51.118308782577515
Epoch: 57, time elapse for current epoch: 51.16083788871765
Epoch: 58, time elapse for current epoch: 51.140204668045044
Epoch: 59, time elapse for current epoch: 51.10639452934265
Epoch: 60, time elapse for current epoch: 51.11641502380371
Epoch: 61, time elapse for current epoch: 51.15155053138733
Epoch: 62, time elapse for current epoch: 51.241127729415894
Epoch: 63, time elapse for current epoch: 81.90204572677612
Epoch: 64, time elapse for current epoch: 51.68192481994629
Epoch: 65, time elapse for current epoch: 51.08146667480469
Epoch: 66, time elapse for current epoch: 51.35550904273987
Epoch: 67, time elapse for current epoch: 51.22075366973877
Epoch: 68, time elapse for current epoch: 51.26197624206543
Epoch: 69, time elapse for current epoch: 51.275999307632446
Epoch: 70, time elapse for current epoch: 51.266908168792725
Epoch: 71, time elapse for current epoch: 51.200406074523926
Epoch: 72, time elapse for current epoch: 51.22204089164734
Epoch: 73, time elapse for current epoch: 51.14965462684631
Epoch: 74, time elapse for current epoch: 51.14586329460144
Epoch: 75, time elapse for current epoch: 51.19008779525757
Epoch: 76, time elapse for current epoch: 51.204936027526855
Epoch: 77, time elapse for current epoch: 51.253639459609985
Epoch: 78, time elapse for current epoch: 51.258516788482666
Epoch: 79, time elapse for current epoch: 51.26776623725891
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-13-f16ba88d420a> in <module>
      2   start_time = time.time()
      3   for batch in dataset:
----> 4     train_step(model, batch, optimizer)
      5   end_time = time.time()
      6   print('Epoch: {}, time elapse for current epoch: {}'

                                    6 frames
/usr/local/lib/python3.9/dist-packages/tensorflow/python/eager/execute.py in
quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     50   try:
     51     ctx.ensure_initialized()
---> 52     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     53                                         inputs, attrs, num_outputs)
     54   except core._NotOkStatusException as e:

KeyboardInterrupt:

SEARCH STACK OVERFLOW
```

```
z = tf.random.normal(shape=(1, latent_dim))
conditioned_array = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
conditioned_array = np.expand_dims(conditioned_array, axis=0)
z_comb = tf.concat([z, conditioned_array], 1)


picture = model.sample(z_comb)[0]


plt.imshow(picture)
plt.show()
```




```
z = tf.random.normal(shape=(1, latent_dim))
conditioned_array = [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
conditioned_array = np.expand_dims(conditioned_array, axis=0)
z_comb = tf.concat([z, conditioned_array], 1)
picture = model.sample(z_comb)[0]
plt.imshow(picture)
plt.show()
```