Name: Jason Zhou
Mentor: Dr. Dongjin Song
Status Report #: 18
Time Spent on Research This Week: 25.5
Cumulative Time Spent on Research: 157.25
Miles Traveled to/from Mentor This Week: 0
Cumulative Miles Traveled to/from Mentor: 0
==========

Monday, February 14th, 2022:

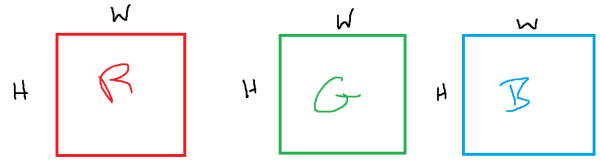
Today, I had a dentist appointment, so I had to reschedule Dr. Song. Our weekly zoom meeting will be held on Wednesday.

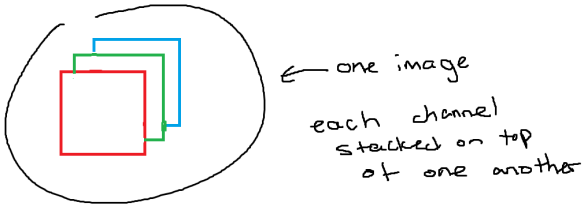
Wednesday, February 16th, 2022: (0.5 Hrs)

Thursday, February 17th, 2022: (3 Hrs)

On this day, I wanted to try making a model for a convolutional autoencoder; however, first I needed to understand how convolutional layers were formatted on Pytorch[1].

Apparently, instead of the height and width of an image, the convolutional layers of a network take in the number of channel, which is another dimension added to an image to store extra data. As such, it was at this point that I learned that images are actually three dimensional and not two dimensional. One dimension is for the height, one dimension is for the width, and one dimension is for the channel. For images, channels are usually used to handle the RGB values of the image, which gives the image its color.





(An example of the channels that I am describing in the above paragraph. As stated, each image has 3 channels that correspond to R, G, or B values. When stacked on top of eachother, they create the colors that people see on the image)

[1] Convolutional layers are primarily used to handle image data.

Besides, the number of channels, I also needed the filter [2]size and stride[3]. After reviewing these topics, I needed to learn how these parameters affected the size of an image because for each layer, the neural network needs to know the exact size of the image that is being inputted.

I quickly googled the equation to calculate the dimensions (N x N) of an image after a convolution, and here is what I found: [(H+2p-f)/s + 1] by [(W+2p-f)/s + 1].

- H = height of the inputted image
- W = width of the inputted image
- p = padding[4]
- f = the size of the fulter
- s = stride

(NOTE: the equation used above is a gross oversimplification of the actual equation used to calculate the size of an image after convolution)

As long as I followed this equation and inputted the right sizes for the neural network to expect, I should have no issues.

Friday, February 18th, 2022: (5 Hrs)

After learning about convolutional layers and TC layers, I tried making my own convolutional autoencoder(CAE) from scratch. As a start, I would be using the MNIST dataset because it is a very easy dataset to use and would be good to use for my first attempt at a CAE.

If I had to summarize the process of making the CAE, I would say it was less than optimal. I was constantly met with errors related to the shape of the inputs I had given the neural network. Despite learning the equations, it was still hard to apply them and challenging to interpret the error messages that the console gave me. Nonetheless, after struggling for many hours, I finally got my neural network to work.

---

[2] A Filter is an N by N array that passes over an image to apply certain effects to an image. In this context, a filter is used to reduce the size of an image, which makes it easier for a neural network to learn. The reduction of the image is commonly known as convolution.

[3] Stride is how many spaces a filter travels after it processes an area of data. This is also used to reduce the size of an image when using a convolutional layer on an image.

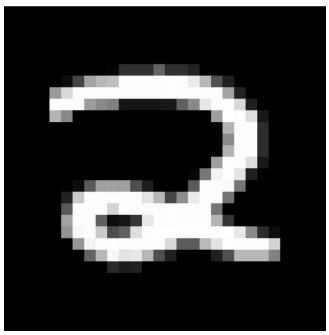
[4] Padding is used to preserve data, especially important data that resides in the corner of an image. This is done by increasing the size of the image. This is important because sometimes a filter can gloss over this kind of data, leading toloss of important data.

```
class CAE(nn.Module):
    def __init__(self):
        super().__init__() #super

        #layers
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 32, 5),
            nn.Conv2d(32, 64, 5),
            nn.Conv2d(64, 128, 5),
        )

        self.decoder = nn.Sequential(
            nn.Conv2d(128, 64, 5),
            nn.Conv2d(64, 32, 5),
            nn.Conv2d(32, 1, 5),
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

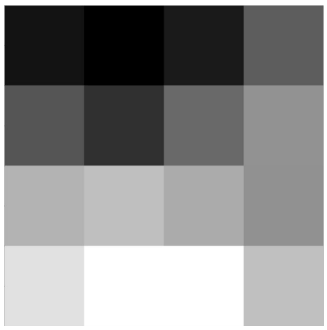
(This is an image of the neural network I created. Although it looks simple, this is the final result after I streamlined the code and worked out the major compile-time errors.)

After finishing the architecture for the network, I passed in an image into the model. Also, I am fully aware that I did not train the network, I was just curious to see what the output would look like.

Here are the results:



(This is the image I started with. In other words, the image that I inputted into the model)



(This is an image of the output. As one can see, although the output is supposed to look like the input, it looks nothing like a two. This was to be expected because I did not train the model; however, the problem is that the output is not the right size. In other words, the output does not have the same number of pixels as the input. Regardless of whether I trained the model or not, the number of pixels should have remained constant.)

Friday, February 19th, 2022: (3 Hrs)

After yesterday, I realized that the reason the number of pixels were not the same was because I only used convolutional layers. These layers are only able to decrease the size of the image, not enlargen them. Thus, after doing some more research online, I learned of the existence of transposed convolutional (TC) layers: the exact opposite of convolutional layers. In other words, a TC layer reverses a convolutional layer. This is important because my research deals with autoencoders and, thus, must encode and decode the data. Because I need convolutional layers to encode the data, I need TC layers to decode [5]the data.

A TC layer essentially takes in the same things as a convolutional layer; however, these parameters now have the opposite effect. For example, in convolutional layers, filter sizes were used to reduce the size of an image. As such, while being used for a TC layer, filter sizes will be used to enlargen the image.

As with convolutional layers, I also needed to learn the equation for the size of an image after deconvolution[6].

[(H-1)*s - 2p + f - 1 + op + 1] by [(W-1)*s - 2p + f - 1 + op + 1]

- H = height
- W = Width
- p = padding
- f = filter size
- s = stride
- op = output_padding (irrelevant to what I am doing right now)

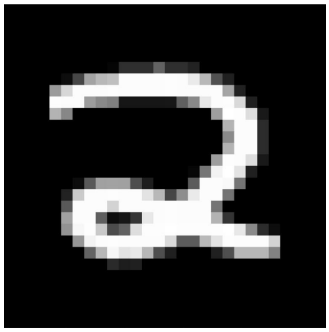
(NOTE: the equation used above is a gross oversimplification of the actual equation used to calculate the size of an image after deconvolution)

Saturday, February 20th, 2022: (2 Hrs)

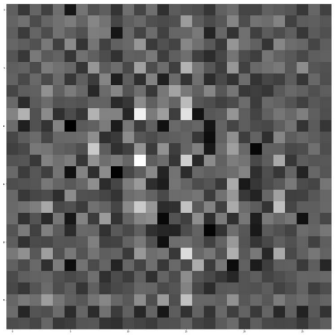
Now that I had learned about TC layers, I decided to replace the decoder section of my network with TC layers. Just like last time, I was hit with a flurry of errors that took a while to sort through. After overcoming these challenges, I decided to actually train my network this time and pass an input into it.

Here are the results:

[5] Decoding is the opposite of encoding. Thus, in order to decode, I need a neural network layer that does the opposite of a convolutional layer: a TC layer.
[6] Deconvolution is a synonym for transpose convolution: the opposite of convolution.



(This is the image I started with. In other words, the image that I inputted into the model. It is the same one I used last time.)



(This is the output. At the very least, the output is now the same size as the input. However, the problem is that despite training my network this time, it does not resemble a two in the slightest.)

As one can see, I still have some things to adjust in my CAE.

Monday, February 21st, 2022: (1 Hrs)

My weekly meeting with my mentor was moved to Wednesday due to some family issues that he had to deal with.

Wednesday, February 23rd, 2022: (1 Hrs)

Before going into my zoom meeting today, I tried to work out some of the issues with my CAE but was unsuccessful.

In the end, I decided to ask Dr. Song for some assistance, and after listening to some of his advice, I was able to resolve my problem. The model now works fine when working with MNIST data. Now, the problem will be to modify my current CAE so that it will be able to handle Mel spectrograms, a visual of sound.

Thursday, February, 24th, 2022: (1 Hrs)

On this day, I spent my time adjusting the values and hyperparameters of the neural network to be able to take in and process Mel spectrograms. This involved changing the channel inputs and stride and filter values so that the input would be properly convolved and deconvolved.

Saturday, February 26th, 2022: (7 Hrs)

After adjusting the values, I had to prepare the batches[7] for the data.

Here is some of my code:

```
#creating the batches

BATCH_SIZE = 32
BATCHES = []

copy = training_data.tolist()

iterations = int((len(copy))/BATCH_SIZE)
for i in range(iterations):
    start = random.randint(0, len(copy) - BATCH_SIZE)
    end = start + BATCH_SIZE
    batch = copy[start:end]
    BATCHES.append(batch)
    copy = copy[:start] + copy[end:] #taking out the batch from the training_data
    #to prevent duplicate numbers from appearing across batches

print(len(copy))
print(f'Shape: {len(BATCHES)}, {len(BATCHES[0])}')
```

(A picture of my code that shows the batching process. I define a size for each batch and use this value to determine the number of iterations that must be done to extract the total batches from the data. I then define a random space of 32 numbers within the array [8]of the training data and save it to a variable. This variable is added to a python list[9], and the 32 number space defined earlier is spliced [10]from the training data array.)

Coding this process took much longer than I expected it to because of all of the different type that I had to handle. To me, python lists and numpy arrays are essentially the same thing (although they are technically different), so it was hard for me to realize that the mismatch errors I was getting was because I had forgotten to convert from data type to another datatype.

After defining the batching process, I had to create the training loop that would be use to train the neural network.

---

[7]The process of batching entails inputting groups of data into a model one at a time instead of simply throwing in all of the data at once. This is supposed to decrease training time.
[8] A method of data collection
[9] Another Method of data collection
[10] Removed

```
#training

model = CAE()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr = 0.0001)

NUM_OF_EPOCH = 250

outputs = []

for epoch in range(NUM_OF_EPOCH):
    for i in range(len(batched_training_data)):
        output = model(batched_training_data[i].view(-1,1,10,474))
        loss = criterion(output, batched_training_data[i])

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    outputs.append((epoch, output, batched_training_data[i]))
    print(f'Epoch {epoch}, Loss: {loss.item():.4f}')
```

(A picture of the training loop plus some other technical elements. What I have done here can be summarized into these steps:

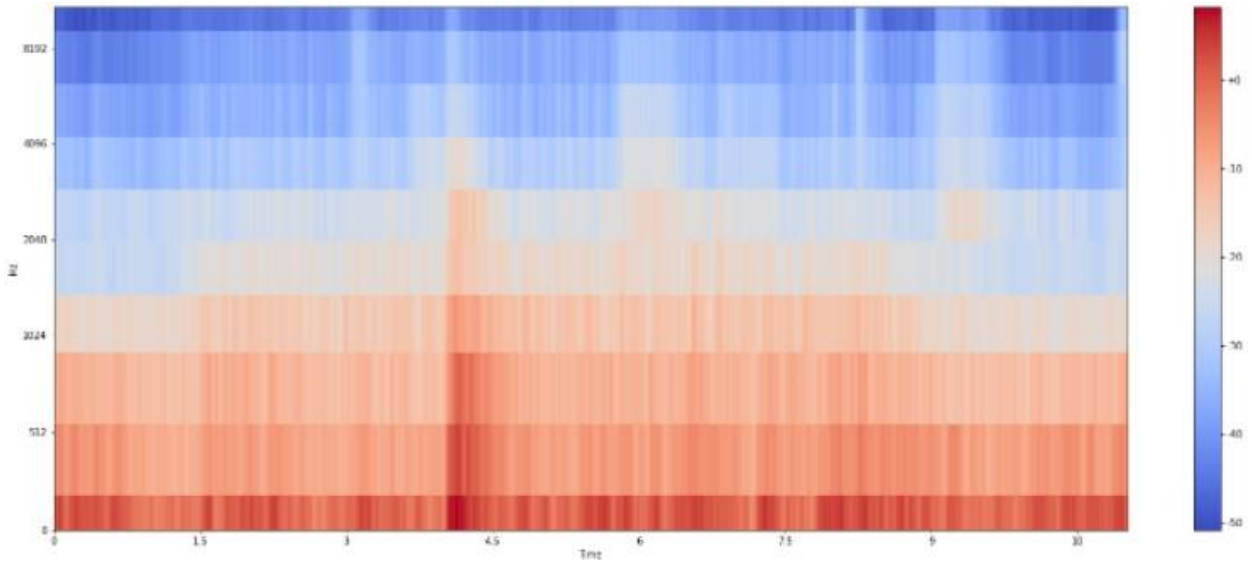
- Define neural network
- Define loss function[11] and optimizer[12]
- Set number of epochs
- Iterate over the batches
- Pass Batches into neural network
- Zero the gradient[13]
- Backpropagate[14]

)

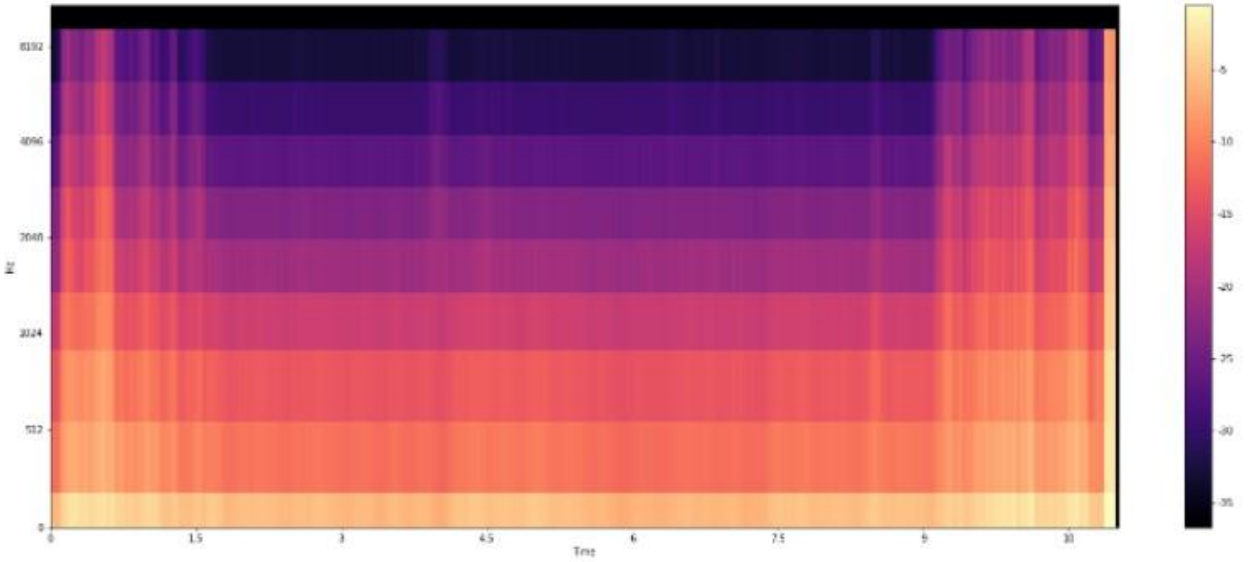
Although the process was a bit tedious because of the errors I encountered, I finally completed the training loop and began training my network. For this training, I would be using my own dataset (the Mel spectrograms). For the input, I would be passing in the entire Mel spectrogram (My mentor suggested that I try breaking up each mel spectrogram into several other inputs but I wanted to try this method first).

Here is the result of the training:

[11] Tells the neural network how wrong it was so that it may correct itself next time
[12] This is the function that actually corrects the network after the loss function tells it how wrong it is
[13] Calculus stuff used to ensure that the model is correcting itself properly
[14] Backpropagation is the process of using the loss function and optimizer



(The original image)



(The outputted image)

If the goal is to have the output be the reconstruction of the input, I would say that the model definitely needs to be modified and improved. First off, the colors are not even the same, and the actual patterns of the sound are the same either. However, this was to be expected because it was my first try. Additionally, at least it looks like a Mel spectrogram.

In summary, things can only go up from here!

Sunday, February 27th, 2022: (2 Hrs)

After seeing the results of my CAE's training, I took this day to investigate subjects such as activation functions[15] to try and see what I could do to improve my neural network.

In essence, there are three main types of activation functions that are used within the hidden layers of a network:

- ReLU, Range: (0, x) where x is the input passed through the ReLU function
- Sigmoid, Range: (0,1)
- Tanh, Range: (-1, 1)

Apparently, ReLU (which is the one I am using right now) is the most common activation function because it is the most practical and useful. Specifically, it does not fall victim to the weaknesses (the vanishing gradient problem[16]) of the other activation functions. As such, I will probably just keep using this activation function.

References

CONV2D. (n.d.). PyTorch. Retrieved February 28, 2022, from https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

CONVTRANSPOSE2D. (n.d.). PyTorch. Retrieved February 28, 2022, from https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html

Raschka, S. (2021, April 15). *L16.4 A convolutional autoencoder in PyTorch -- code example* [Video]. YouTube. https://www.youtube.com/watch?v=345wRyqKkQ0&t=366s

zhengquantao. (20, November). *RuntimeError: mat1 and mat2 shapes cannot be multiplied (64x13056 and 153600x2048)*. PyTorch. Retrieved February 28, 2022, from https://discuss.pytorch.org/t/runtimeerror-mat1-and-mat2-shapes-cannot-be-multiplied-64x13056-and-153600x2048/101315

[15] Functions that are used to confine calculated values within a network to a certain range of values. For instance, all values may be scaled to a range of -1 to 1. This keeps the network consistent and stable.
[16] I believe this is a problem that occurs with Sigmoid and Tanh, which causes the gradient to be ruined. If the gradient is ruined, the model cannot learn, which essentially makes machine learning useless. However, I do not have a solid idea of what this problem actually is. I will have to look into further.