AIML402: Assignment 3 report

Junyi Shen (8386129)
October 8, 2024

In this assignment, I implemented a conditional variational autoencoder (CVAE), which can learn from the Sign Language MNIST dataset, and finished the following three tasks:

1. Generate images of a consonant or a vowel letter.
2. Generate images of a specific letter.
3. Generate images with the rotation angle 0, 90, 180, or 270.

For each task, I wrote a Python script using TensorFlow and trained a model separately to achieve the task. I submitted three scripts and three saved models corresponding to the three tasks. Their names are myModel_task1.py, myModel_task1.keras, myModel_task2.py, myModel_task2.keras, myModel_task3.py, and myModel_task3.keras.

In the first part of this report, I will introduce the design of my CVAE model, including its architecture, parameters, and how it can complete the three tasks. In the second part of the report, I will illustrate and explain the performance of my model. In this part, some generated images will be demonstrated to show how the model performs in each task. Good image examples and bad image examples will all be shown for comparison. In the third part of the report, I will discuss the advantages and disadvantages of CVAE against VAE with some example use cases. Finally, I will make a brief conclusion about the work done for this assignment.

1 CVAE Model Design

In this part, I will first briefly introduce the fundamental architecture of my CVAE. Then, for the three tasks, I will explain how each model can achieve each task.

A CVAE model consists of an encoder, a decoder, a loss function, and conditional information. I will introduce each part of my CVAE model and explain the parameters I used.

1.1 Encoder

The encoder's task is to compress the image into the latent representation with conditional information. In the encoder, a 28×28 image and the additional condition are the input. The shape of the input layer is (28, 28, 1 + *add_dimension*), where *add_dimension* is the number of conditional channels. If there are more than two classes, it is equal to the number of classes. For binary classification, this value is 1.

Then the data goes through three convolutional layers, whose number of filters are 128, 64, and 128, respectively. All three convolutional layers have a kernel size of 3, strides of 2, and an activation function of ReLU. Here is the reason why I chose 128, 64, and 128 as the filter numbers:

(1) 128 filters for the first convolutional layer: Because the Sign Language MNIST dataset is complex and the data contains many details. I started with a relatively large filter number to capture more features at the beginning.

(2) 64 filters for the second convolutional layer: The second layer is for filtering features. The filter number decreases to 64 because it can help the model focus on abstracting a small number of high-level features and reduce noise.

(3) 128 filters for the third convolutional layer: The third layer is for expanding features. I increased the number of filters back to 128 to allow the model to explore more high-level features.

The output of the convolutional layers is flattened and passed through a dense layer, which produces the latent space. The output shape is (*latent_dim* + *latent_dim*). The two parts of the output are the mean and log variance. The mean is the center of the distribution and log variance is the spread of the distribution, determining how much the distribution of the samples varies from the mean.

The latent dimension is set to 512 because for complex data with many detailed features like the Sign Language MNIST, we need to use a large latent dimension to store the featured representation in the latent space. If the latent dimension is too small, the generated images will not be clear enough since not enough features are stored in the latent space.

I applied a batch normalization after each convolutional layer. The output of each layer is normalized before it is passed to the next layer. This method can reduce the risk of vanishing or exploding gradients and help faster convergence.

1.2 Decoder

The decoder's job is to reconstruct the image from the latent space representation along with conditional information. In the decoder, the input is the latent vector and the conditional input. The shape of the input layer is (*latent_dim* + *add_dimension*), where *latent_dim* is the dimension in latent space and *add_dimension* is the number of conditional channels.

Then the data goes through a 7×7×*latent_dim* dense layer to enable the decoder to capture complex features of the image. After that, the output is reshaped into 7×7×*latent_dim* and passed two transposed convolutional layers, whose filters are 128 and 64, respectively. The reason why I chose 128 filters first and then 64 filters is that it allows the model to first expand the representation and extract more detailed features from the latent space. Then, the 64-filter layer focuses on a smaller number of high-level features to reconstruct a clearer image while avoiding overfitting.

Both transposed convolutional layers in the decoder have a kernel size of 3, a stride of 2, and an activation function of ReLU. The ReLU function can not only introduce non-linearity, but it can also prevent the vanishing gradient problem during training.

The last layer in the decoder is a transposed convolutional layer with a single filter

and no activation function. This layer is used to reconstruct the output image.

1.3 Loss function

The loss function of my CVAE model calculates the loss by combining reconstruction loss and Kullback_Leibler (KL) divergence.

After encoding the image data and the conditional information, the encoder outputs the mean (center of latent distribution) and log variance (spread of distribution). Using the reparameterization trick, it samples from the latent space representation. Then the sample is decoded along with the conditional information, and the result is used for computing the reconstruction loss. Then the KL divergence is computed, which assesses how close the learned latent distribution is to a standard normal distribution. Finally, the reconstruction loss and the KL divergence are added together to get the total loss. This loss function can make the autoencoder be variational since it will make the latent contribution close to a normal distribution around the center.

1.4 Conditional information: Task 1

For task 1, the conditional information is consonant (0) or vowel (1). Because it is a binary classification, it does not need to perform one-hot encoding. The original labels in the Sign Language MNIST dataset are 0 to 8 and 10 to 24 (9 and 25 are excluded because the letter 'J' and 'Z' are not in the dataset). I mapped the label to 0 and 1, and changed the training and test data labels before training and testing the model. The input of the encoder is the image data concatenating with 0 (consonant) or 1 (vowel). As a result, the latent space representation contains the conditional information after training.

To generate images, the conditional information is concatenated with the random sample in the latent space representation before decoding. By this method, the model can generate images with certain conditions.

1.5 Conditional information: Task 2

For task 2, the conditional information is the letter index. However, because the original label in the dataset is 0 to 24 but excludes 9, we need to convert it to 0 to 23 before performing one-hot encoding. After converting, the label will be one-hot encoded ([1, 0, 0, ...], [0, 1, 0, ...], ...) and concatenated with the image data as the input of the encoder. To generate an image, the one-hot encoded label will be concatenated with a random sample in the latent space representation and the decoder will generate an image of a specific letter.

1.6 Conditional information: Task 3

For Task 3, I first need to generate rotated image data. For each image in the Sign Language MNIST dataset, I used *tf.image.rot90* function of TensorFlow to rotate the images, and create a label (0, 1, 2, or 3) for each image to represent the rotation angle (0, 90, 180, 270, respectively). Then, save the rotated dataset as a *.npz* file.

The input of the encoder is the image data concatenating with the one-hot encoding of one of the four angles 0, 90, 180, and 270, which is [1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], or [0, 0, 0, 1], respectively. When generating images, this one-hot encoding is concatenated with the random sample and decoder can generate images with the corresponding angle.

1.7 Summary of architecture

In summary, the architecture and parameters of my CVAE model is as follows:

	myModel_task1	myModel_task2	myModel_task3
Number of classes	2	24	4
Conditional information	0 or 1	one-hot encoding of 24 labels	one-hot encoding of 4 labels
Latent dimension	512		
Encoder layers	3 convolutional layers (filter number 128, 64, 128), 1 dense layer		
Decoder layers	1 dense layer, 3 transposed convolutional layers (filter number 128, 64, 1)		
Loss function	Reconstruction loss (use cross entropy) and KL divergence		
Learning rate	0.001		

Table 1: CVAE model architecture and parameters

2 Performance analysis

In this part, I will provide some results (images) my model generated and compare good examples and bad examples for each task. Also, I will explain the reason for the performance differences.

For each task, the model was trained for 100 epochs under the Sign Language MNIST dataset. For task 3, because there are four training images for each image data, the training dataset is four times as the other two tasks.

2.1 Task 1

In task 1, as Figure 1 and Figure 2 show, the model can generate different sign images depending on whether they are consonant or vowel. The model does not have information about the corresponding letter of each training image. Some generated images look like a specific letter (good examples), but some images could look like a mixture of multiple letters (bad examples). The images generated in task 1 are blurrier than in task 2, as the model would mix up the features of different images in the latent space.

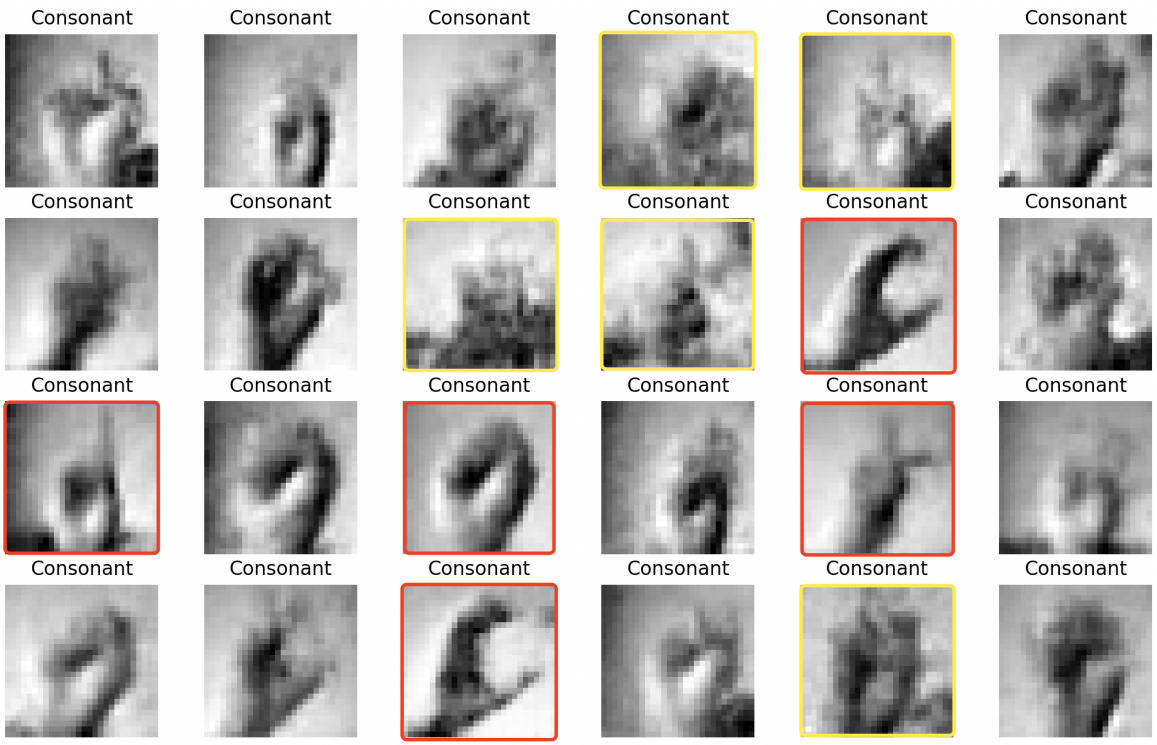

Figure 1: Task 1. Examples of consonant sign images. (Red: relatively good examples. Yellow: relatively bad examples)

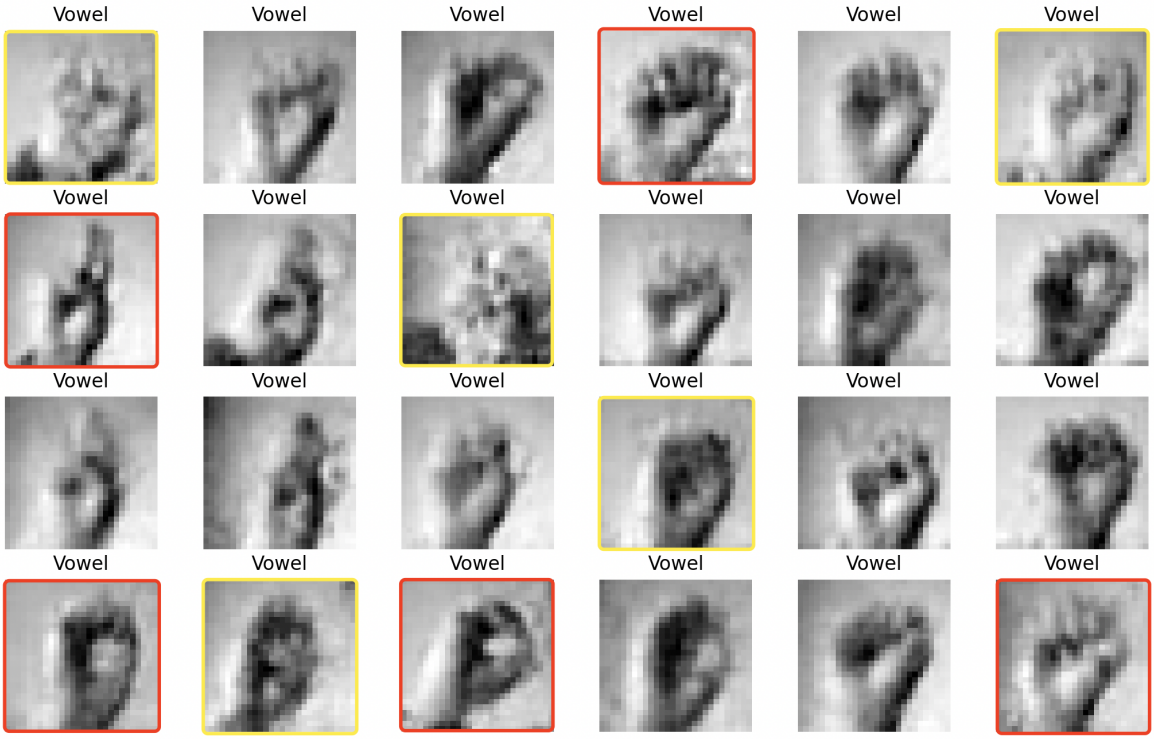

Figure 2: Task 1. Examples of vowel sign images. (Red: relatively good examples. Yellow: relatively bad examples)

2.2 Task 2

In task 2, the generated images are clear because the model knows exactly what letter the input images correspond to. As shown in Figure 3, almost all the images are clear,

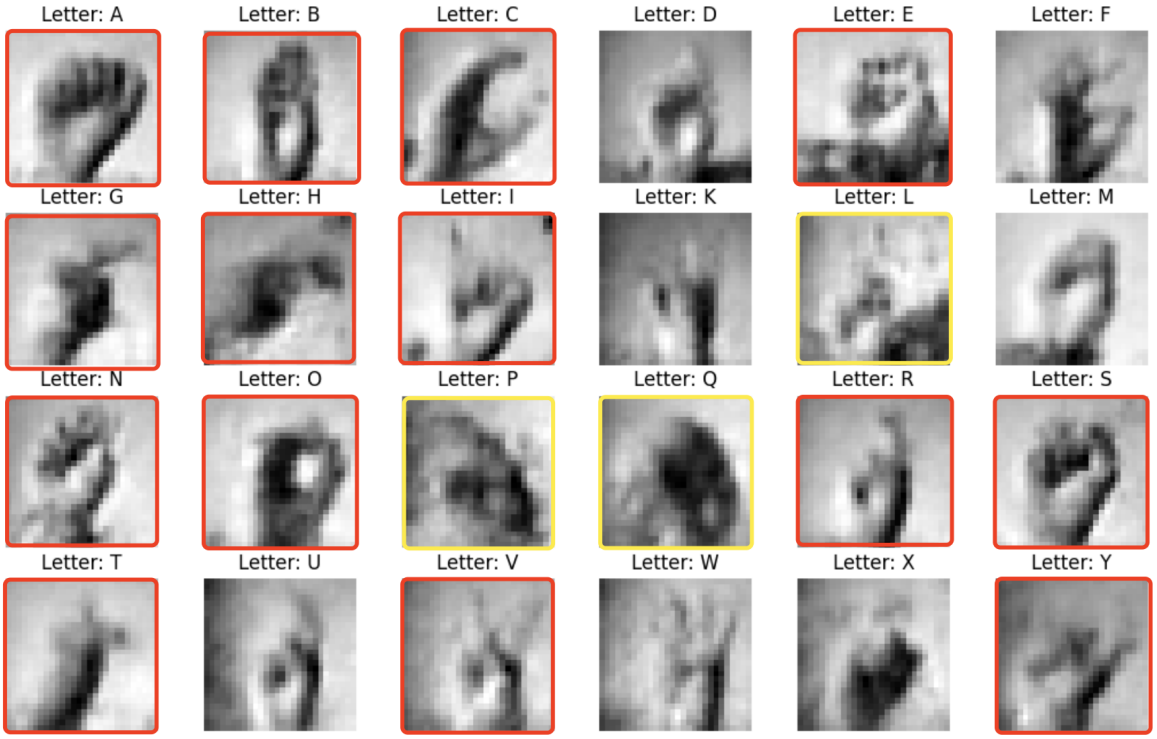

Figure 3: Task 2. Examples of all letter signs. (Red: relatively good examples. Yellow: relatively bad examples)

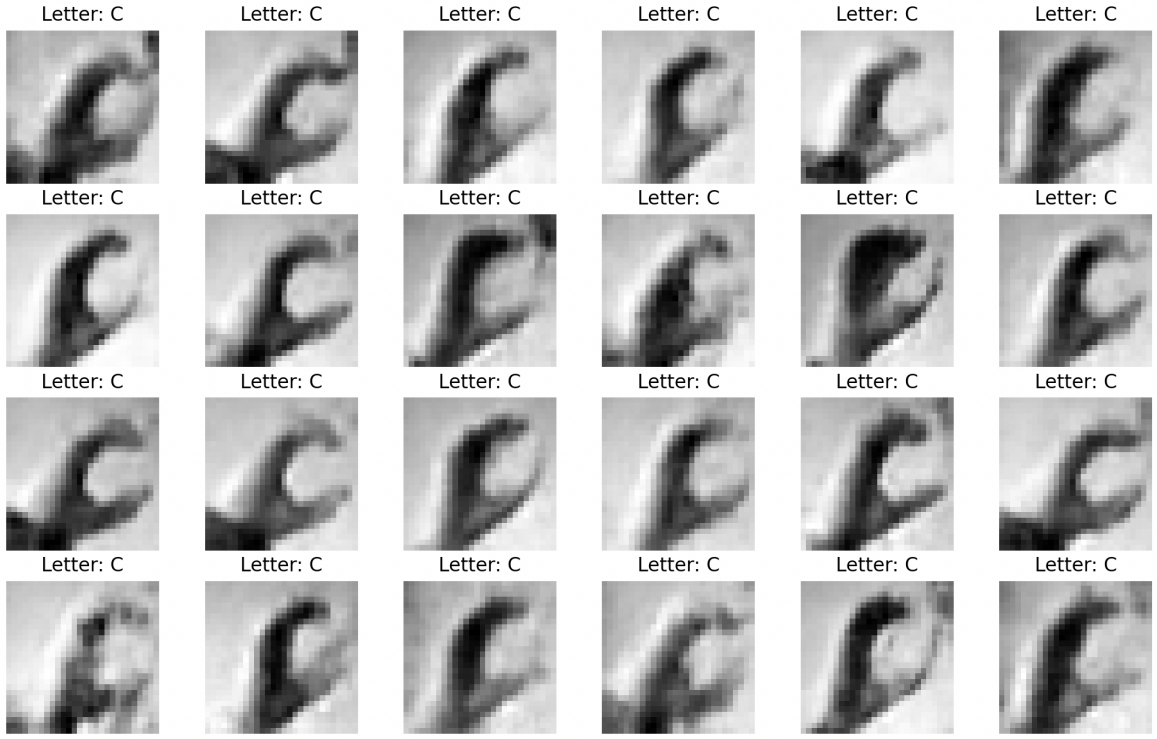

Figure 4: Task 2. Examples of letter C. Almost all the images generated are clear.

only except letters L, P, and Q. Figure 4 shows that the generated images of the letter C are all clear. This is because the pattern of letter C is simple and easy for the model

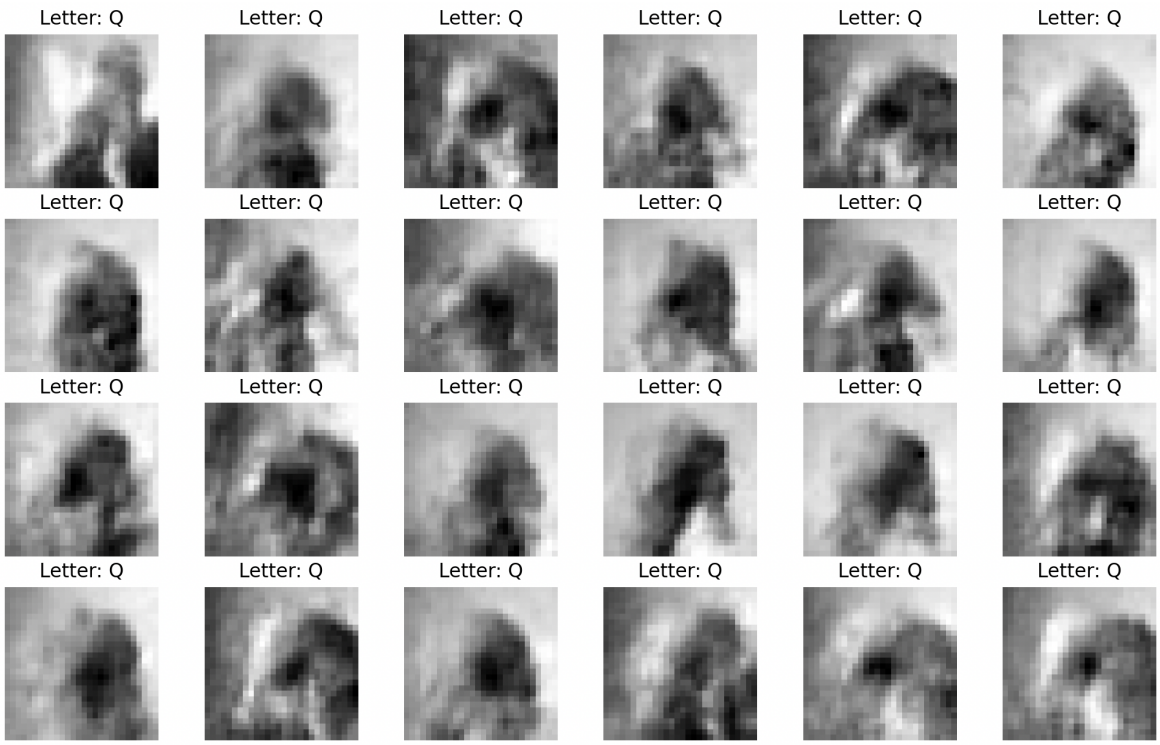

Figure 5: Task 2. Examples of letter Q. Many images generated are unrecognizable.

to extract. In contrast, in Figure 5, letter Q images are hard to recognize. The reason is the patterns of the letter Q are complex, which is hard for the model to learn.

2.3 Task 3

In task 3, as shown in Figure 6 and Figure 7, the generated images are the blurriest among the three tasks. This is because in this task, the model only knows the angle of the input images and has no other information. As a result, it is difficult for the model to separate the image patterns of each letter, and it would mix up image patterns from different letters. However, it is clear that the model learns the angle of the rotation. Thus, for each angle, it is distinguishable about what angle the image rotates. More convolutional layers and more filters would improve the performance, but it also requires more training time and computing power.

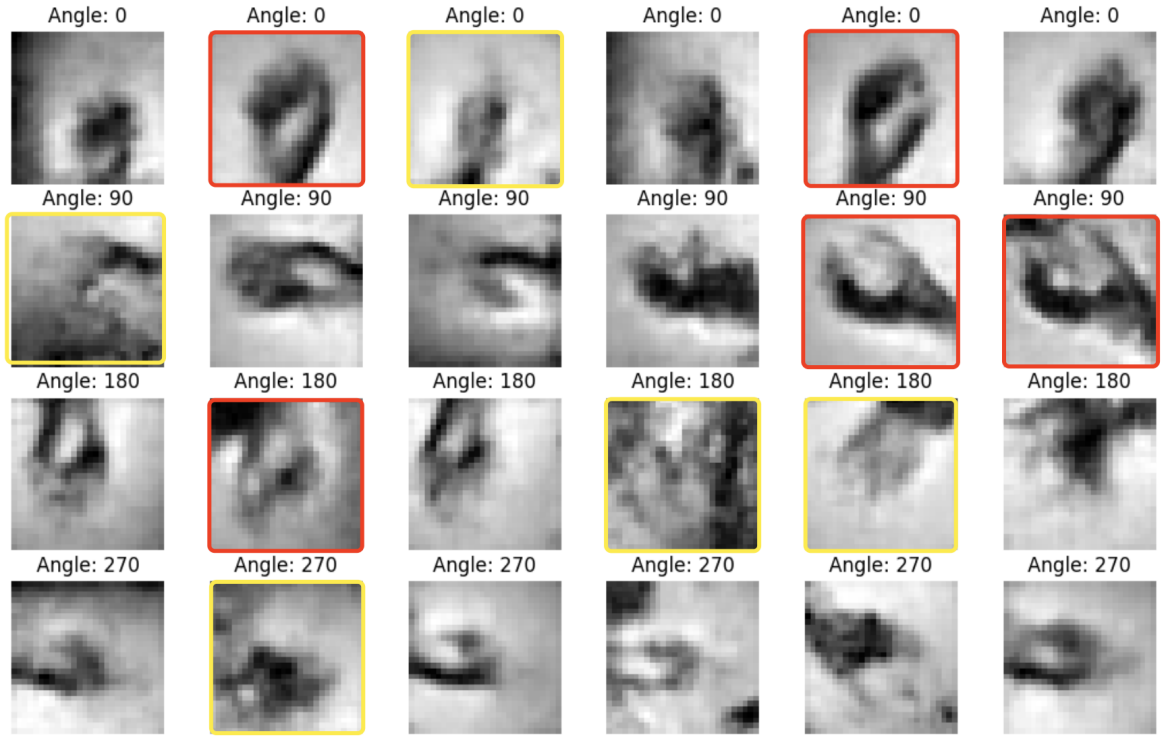

Figure 6: Task 3. Examples of four rotation angles (Red: relatively good examples. Yellow: relatively bad examples).

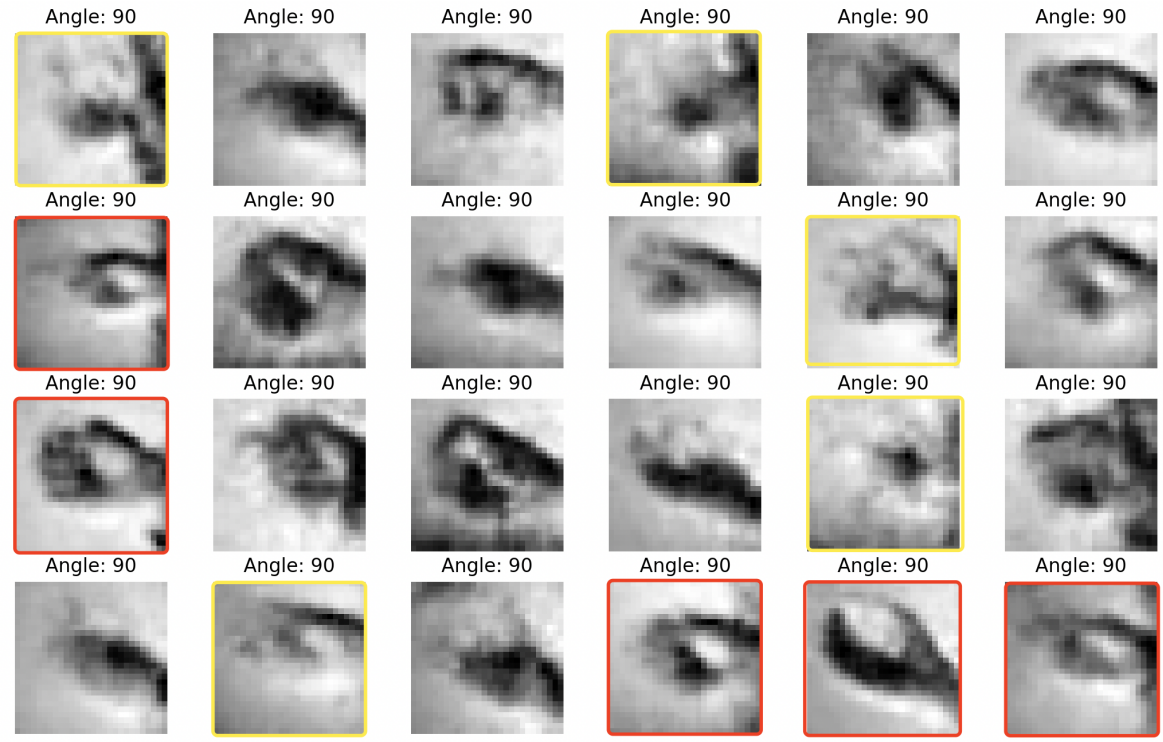

Figure 7: Task 3. Examples of rotating 90 degrees (Red: relatively good examples. Yellow: relatively bad examples).

3 Comparison between CVAE and VAE

In this section, I will discuss the pros and cons of CVAE compared to VAE with example cases where each would be more suitable.

3.1 Pros of CVAE against VAE

CVAE can generate outputs based on specific conditional inputs, making it more suitable for datasets with specific labels. With conditional information, the model can learn the dataset with the label and produce output based on the label. As a result, for specific labels, the model can generate better results compared to VAE. For example, task 2 of this assignment can produce the best result among the three tasks. This is because in task 2, the model learns the corresponding letter to each image, allowing the model to generate specific images based on the letter. However, in task 1 and task 3, the model does not know the corresponding letter of each image, so the images generated tend to be blurrier, as it would mix up different patterns from different images.

3.2 Example where CVAE is more suitable

For datasets with labeled data and we want to generate samples based on specific conditions, CVAE is a better choice since it will produce outputs that fit better to the specific labels. For example, to generate images of a specific animal, or generate a human face based on age and gender, CVAE is a better selection.

3.3 Cons of CVAE against VAE

CVAE does not work well with unlabeled data. CVAE requires conditional information to train the model and produce outputs. However, in many situations, no conditional information is provided, or creating the label for the data would be very complicated and impractical, especially in unsupervised learning. In this case, CVAE is not a proper choice.

3.4 Example where VAE is more suitable

In the real world, there are many cases where we can only obtain data without particular labels. For example, in detecting anomalous network attacks, VAE is a more suitable choice because the data is typically not labeled. VAE can be trained on normal traffic patterns, and anomalous network traffic patterns would cause higher reconstruction errors, signaling an anomaly.

4 Conclusion

In this assignment, I developed a CVAE model to train from the Sign Language MNIST dataset and finished three tasks: generate images based on whether they are consonants or vowels, generate images of a specific letter, and generate rotated images. Task 2 produced the best results because in this case, the model can better extract the features as the conditional information (letter) guides the model to learn the features of

corresponding letters.

The assignment deepened my understanding of neural networks, especially CVAE. In the process of developing, training, and testing the CVAE model, I tried different parameters to observe the results in order to optimize the generated images. Finally, I compared the advantages and disadvantages of CVAE against VAE, giving me a deeper understanding of their characteristics and situations where each technology is more suitable.