Introduction to Deep Learning: Assignment 2

David Hof (3821013)
Tanisha Kasar (4459431)
Gaurisankar Jayadas (4374886)

Task 1: Generative Models

Introduction

For this task, we used a self-chosen image dataset to retrain two pre-defined generative models: a variational autoencoder (VAE) and a generative adversarial network (GAN). After retraining these models, we systematically generated new data examples to explore their learned latent spaces.

Methods

The pre-defined VAE and GAN were originally trained on a downsampled subset (20,000 images; 64x64x3) of the Flickr-Faces-HQ Dataset, which features 70,000 images (1024x1024x3; png format) of human faces.

Both models were composed of 1) a downsampling architecture (VAE: encoder; GAN: discriminator) and 2) an upsampling architecture (VAE: decoder; GAN: generator). Downsampling and upsampling were realized using convolutional networks and deconvolutional networks, respectively.

The working principle of a VAE is this: An observation is passed through the encoder, which predicts the mean and standard deviation of the corresponding posterior probability distribution over the latent variable vector (low-dimensional representation). A new sample is then "drawn" from this distribution by multiplying the predicted standard deviation by a Gaussian noise vector and adding it to the predicted mean. The Gaussian noise component is generated from an independent computational branch to keep stochasticity out of the main computational pipeline and thereby enable backpropagation (reparameterization trick). The sampled latent vector is subsequently passed to the decoder, which predicts the original observation. This architecture uses a negative evidence lower bound (ELBO) loss function, which depends on how accurate the final prediction is and how similar the estimated posterior over the latent variable is to the corresponding prior (i.e., a multivariate standard normal distribution).

The working principle of a GAN is this: Random noise vectors are passed through the generator, which creates sample images. These sample images together with real images are passed to the discriminator, which predicts whether each image is real or generated. The (in)ability of the discriminator to accurately distinguish real and generated images provides a training signal for the generator. The discriminator, on the other hand, is effectively trained like a standard binary classifier. During training, the weights of the discriminator and of the generator are updated in turns, which makes the discriminator better at telling real from "fake" and the generator better at "fooling" the discriminator.

Figure 1: Example Images from the Butterfly and Moths Image Classification Dataset

The fundamental difference between VAEs and GANs is that VAEs are explicitly probabilistic, learning a probability distribution over the training data, whereas GANs implicitly approximate the training data distribution by training a generator to produce samples that are indistinguishable from real data.

We chose to retrain the generative models on the Butterfly & Moths Image Classification Dataset, which features 12,594 training, 500 test, and 500 validation images (224x224x3; jpg format) of 100 butterfly/moth species (Figure 1). Here, we only used the training images. In a preprocessing step, we rescaled the images to render their dimensions consistent with those of the face images. The pixel values were subsequently scaled to the range [0, 1] for the VAE and [-1, 1] for the GAN.

To adapt the pre-defined VAE and GAN to this new dataset, we made some slight changes to the model configurations.

While the encoder and decoder in the pre-defined VAE used a series of convolutional/deconvolutional layers with a fixed number of channels, we adjusted this to a hierarchical channel structure. That is, during downsampling, as the size of the representation decreased by a factor of two per layer, the number of channels now increased by a factor of two—and vice versa for upsampling. The outermost convolutional and deconvolutional layers featured 32 channels, the innermost 256 channels. Furthermore, the dimensionality of the latent variable was decreased from 64 in the pre-defined VAE to 16 and batch size was increased from 8 to 16. All other hyperparameters were left unchanged: optimizer: Adam (learning rate = 1e-3); (de)convolutional kernels: 3x3; stride: 2; (de)convolutional layer activation function: ReLU; activation function at the end of encoder/decoder: sigmoid; number of epochs: 20.

For the GAN, we decreased the dimensionality of the latent variable from 256 in the pre-defined model to 16. We changed the fixed number of channels in the convolutional/deconvolutional layers from 128 to 64 and decreased batch size from 64 to 32. All other hyperparameters were left unchanged: optimizer: Adam (learning rate = 2e-4, beta1=0.5); (de)convolutional kernels: 3x3; stride: 2; (de)convolutional layer activation function: ReLU; activation function at the end of generator: Tanh; activation function at the end of discriminator: sigmoid; number of epochs: 20.

After training was completed, we generated two randomly sampled noise vectors per model (scaled by a factor of $\frac{1}{6}$ [VAE] or using 2SD-truncation [GAN] to limit their spread) and fed them to the VAE decoder and GAN generator to produce new image samples. We

then linearly interpolated between the two images to explore the models' latent spaces.

Results

The retrained VAE yielded reasonably plausible, albeit not particularly high-resolution image generations (Figure 2). Qualitatively speaking, the objects in most of these images could easily be recognized as butterflies/moths.

The results of the VAE latent space exploration via linear interpolation between two randomly sampled noise vectors are shown in Figure 3.

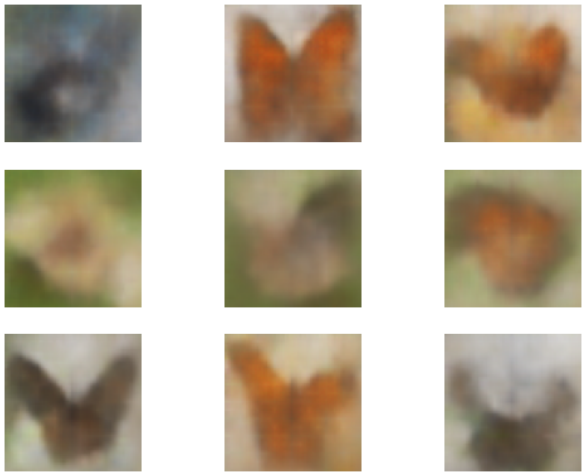

Figure 2: Example Image Generations from the Retrained VAE

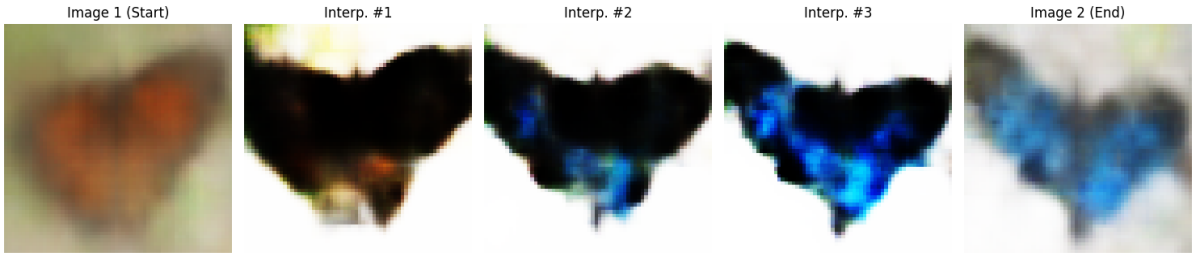

Figure 3: Linear Interpolation Between Two Images Generated by the VAE Decoder

The retrained GAN yielded relatively low-quality image generations (Figure 4). Only in some of these images could the objects in them be recognized as butterflies/moths.

The results of the GAN latent space exploration via linear interpolation between two randomly sampled noise vectors are shown in Figure 5.

Discussion

Retraining the VAE and GAN models proved to be a more challenging task than we first assumed. Systematic hyperparameter tuning is necessary to enable a generative model to effectively mimic its training data. This requires a decent amount of patience.

Once (re)trained, the VAE and GAN generated qualitatively different image samples. The VAE's generations were quite realistic and displayed a moderate amount of variety in terms of color and shapes; butterflies/moths were clearly recognizable in these images.

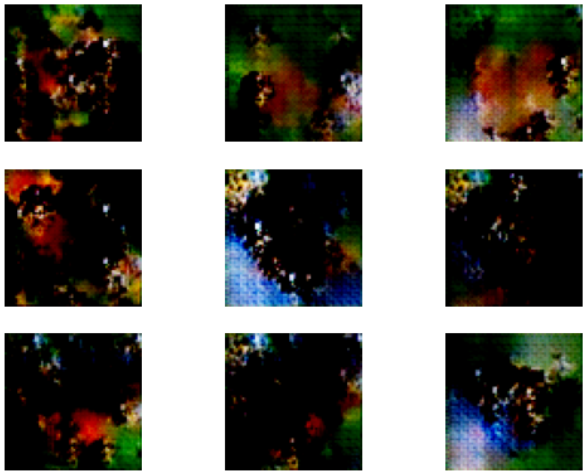

Figure 4: Example Image Generations from the Retrained GAN

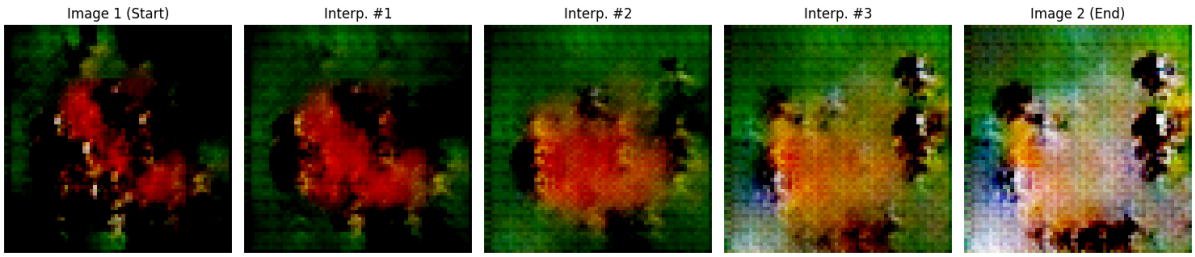

Figure 5: Linear Interpolation Between Two Images Generated by the GAN Generator

However, the image resolution was suboptimal and the insects appeared relatively blurry. By contrast, the GAN's generations were much less realistic and had an abstract or even artistic look; many of these images looked similar (mode dropping), it was much more difficult to identify butterflies/moths in them, and the overall image quality was poor.

The differences in image quality and appearance between the two models are likely due to their inherently different approaches to modeling the distribution of the training data (explicitly probabilistic vs. implicitly probabilistic). These mathematical differences might have interacted with the statistical peculiarities of the dataset we chose for retraining. The GAN, in particular, was very difficult to train on this data. The training process itself (i.e., the evolution of the loss) was highly unstable and fully overcoming this would have likely required more advanced techniques that were beyond the scope of this assignment.

Lastly, the linear interpolations through the models' latent spaces were surprisingly smooth. This demonstrated that both VAEs and GANs have well-behaved latent spaces.

Task 2: Sequence Modeling with RNNs

2.1-2.3 Introduction

[Tanisha's part]

2.1-2.3 Methods

[Tanisha's part]

2.1-2.3 Results

[Tanisha's part]

2.1-2.3 Discussion

[Tanisha's part]

2.4 Introduction

In this task, we are given a textual query and aim to produce an image of the answer. We also have a basic arithmetic query between two-digit numbers and must generate a two- or three-digit answer in image format. We have a custom dataset made using the MNIST dataset; the textual and image queries and the answers are in the same format.

Our model seeks to address a simplified version of this task: When provided with a textual query, it generates a series of images that ideally depict a correct and coherent "response." To accomplish this, the model uses recurrent neural networks (RNNs) to encode the text input and decode it into a sequence of image representations. This is followed by convolutional neural network (CNN) layers synthesising visually meaningful images. Here, evaluation is subjective, and we choose to do it manually.

2.4 Methods

First, we encode the text; the model takes a text query as a one-hot encoded fixed-length vector that feeds into an LSTM layer for encoding. The latent vector captures the semantic meaning of the input query. Once the latent representation is acquired from the text encoder (which includes an LSTM output and a dense layer), this vector is replicated for each time step of the intended output sequence length. Subsequently, an additional LSTM layer handles the repeated latent vector sequence to generate a time series of high-level representations that will be converted into images.

The next step is to decode the high-level representations into images. We use a `TimeDistributed` layer that applies a fully connected network, reshaping outputs into a spatial format suitable for convolutional decoding. The CNN decoder aims to produce coherent image features and textures.

We use the custom data we created beforehand to train the model: the text query and the corresponding image sequences, and we use a 90%-10% split. We use the mean squared error (MSE) to compare the generated image pixels to the ground truth. For stable training, we employ `adam`.

2.4 Results

Since the problem is simple enough and we have managed to train the model to an accuracy of 85.7% (test) after training for 50 epochs, we have considerable generalisation for the task at hand. Mind that we have a training accuracy of 85.8%, which means we have a good generalised model. The results are lower resolution and a bit blurred but nonetheless legible; they lack finer details and boundaries. The results have good semantic accuracy and resemble the shape of the real ground truth answers, which we aimed to achieve.

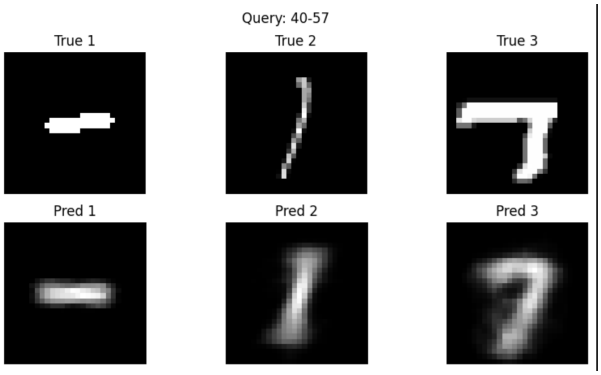

Figure 6: Result for text-image RNN with CNN for image synthesis.

2.4 Discussion

We have a comparatively simple model that functions well for the simple task at hand. We also have high-quality training data for the task, which has significantly impacted our results. We use MSE loss, which focuses on pixel-level accuracy; this means that smooth and blurry images are expected. If we had used a GAN, this might have resulted in sharper images. Aesthetically, the results may not be at par with state-of-the-art models, but we were able to achieve semantic inference from them, which means we have succeeded in what we are aiming for.

2.5 Introduction

In this subsection, we aim to experiment with an extra LSTM layer in the encoder and the single LSTM layer we used in our original model implementation for text-to-text, image-to-text, and text-to-image models. Here, our assumption is that an additional LSTM layer in the encoder will increase the representational power of the network, help it capture more intricate details, and potentially improve the reasoning capabilities of the models.

2.5 Methods

The original architecture used a single encoder LSTM layer followed by a decoder mechanism. Here, we add a new LSTM layer with the first LSTM layer's `return_sequences=True` to pass the entire output sequence into the second LSTM layer, making the network deeper. This should enable the encoder to provide richer, more abstract representations before decoding. The rest of the network is left intact, and the training for the original and updated model is kept the same.

2.5 Results

Text-to-Text: In the original model, we trained for 40 epochs to achieve a test accuracy of 98.9%. We also checked the answers manually and saw that the answers always had errors, often within +1/-1 or +10/-10. One digit consistently being wrong indicated that the model was generalising fairly well but still made arithmetic mistakes. Thus, accuracy alone did not reflect the model's reasoning capabilities.

In the updated model, the test accuracy is 99.7%, but manual inspection shows mixed results. Some answers have +1/-1 errors, while others deviate more than expected. De-

spite the higher accuracy, we did not observe a substantial improvement in generalisation or arithmetic reasoning across all cases.

Here, Fig 7 shows the accuracy and loss plots.

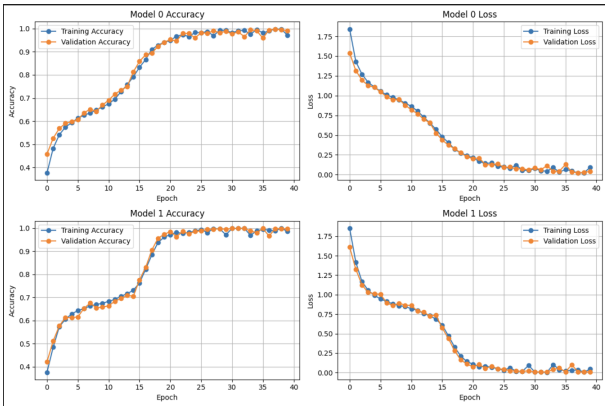

Figure 7: Result for text-to-text accuracy and loss.

Image-to-Text: In the original model, we trained for 25 epochs, and we noticed that the model seemed to reach a training accuracy of 95% after 20 epochs, as in the previous case of test-to-text. Here, we achieve a training accuracy of 98.7%. Still, the model does not generalise well, as the test accuracy is merely 57.7%, which is expected as we have a more complicated problem than the previous one. Manually testing the answers showed that we did get some answers in an error range of +1/-1 percentage in some cases. We also noticed that the validation loss took a dip initially and then continued to increase.

This time, the updated model showed considerable improvement. The training accuracy is only 82.5%, but the test accuracy is 83.2%, which means our additional LSTM layer has improved the model's generalisation. In manual testing, we found more accurate examples and fewer errors, and the percentage of deviation from the ground truth answer was better.

Here, Fig 8 shows the accuracy and loss plots.

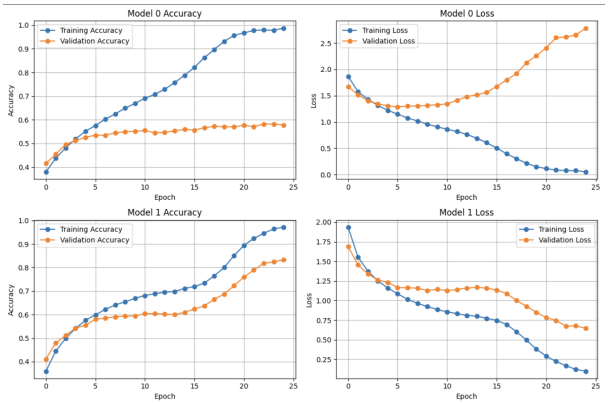

Figure 8: Result for image-to-text accuracy and loss.

Text-to-Image: The original model had a test accuracy of 85.7%, and the updated model had an accuracy of 86%. Adding a new LSTM layer does not significantly increase the model's performance. Manual testing shows that the original model has fewer blurry images than the updated model.

2.5 Discussion

Text-to-Text: Our intuition was that adding a new LSTM layer to the encoder would provide a more representative latent space. While accuracy increased, there was no clear improvement in arithmetic reasoning or generalisation. The model may be overfitting as it becomes more complex, capturing noise rather than genuinely improving arithmetic inference.

Interestingly, the updated model reached around 70% accuracy faster than the original model, suggesting quicker learning. It then showed a steep increase above 95% within the next five epochs, followed by saturation and slow improvement. In contrast, the original model displayed a smoother learning curve, suggesting it was steadily incorporating nuances. Additional epochs and complexity might have led to overfitting rather than true generalisation improvements.

Image-to-Text: Using an LSTM layer to create a more representative latent space has worked out, per our reasoning. The fact that we have a more complicated problem where we have an image-based query rather than a textual query helps us understand how we needed more abstraction for the latent space. We see a considerable improvement in the model's generalisation by adding this LSTM layer to the encoder to achieve this. This one change showed a jump of around 25% in test performance. We also see how the plots are so different; the original model did very badly at generalising the problem, and we see that validation loss plateaus and then rises again. It goes beyond the starting point by the end. The updated network has a much clearer plot that represents the improvement in performance that we could verify manually, too. Here, we can see a clear example of how the problem's complexity benefits directly from adding an LSTM layer.

Text-to-Image: Here, an additional LSTM layer does not do much in accuracy. However, manual testing shows that the updated model has more blurry images. This could be because we are converting the text sequence at the beginning into the latent space, which is much simpler than an image query, as in the previous example. We have 5 x MNIST images as the query in the previous example of image-to-text vs the character string query in this example. The LSTM layers help generalise more complex examples with additional abstraction. At the same time, it overgeneralises the case when the task is too simple, as is the case here.

Contributions

- **David Hof**: Task 1 code, Task 1 report
- **Tanisha Kasar**: Task 2 code, Task 2 report
- **Gaurisankar Jayadas**: Task 2 code, Task 2 report