Introduction to Deep Learning - Assignment 2

Generative models and sequence modelling

Jelte Oldenhof (3834069), Burak Ağaçhan (4186095), Praneeth Dathu (4174089)

December 13, 2024

1 Task 1: Generative Models

1.1 Introduction and Motivation

Generative models like Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) can learn data distributions and produce new samples similar to real data. Here, we applied both models to two datasets: a relatively uniform face dataset and a more diverse, smaller flower dataset. We first tuned hyperparameters and architectures using the simpler face dataset, then applied the setup to the more challenging flower dataset.

Our objectives were to:

1. Understand VAEs and GANs at a fundamental level.
2. Examine how architectural components (e.g., convolutional layers, batch normalization) and hyperparameters (e.g., learning rates, latent dimensions, KL weights) affect performance.
3. Compare results on a uniform dataset (faces) versus a diverse, limited dataset (flowers).

1.2 Datasets and Preprocessing

1.2.1 Face Dataset

We used the FFHQ (Flickr-Faces-HQ) dataset [1], which was provided in the assignment's notebook. This dataset consists of high-quality, well-aligned face images collected from Flickr. For our experiments, we selected a subset and resized images to $64 \times 64 \times 3$, normalizing pixel values to [0,1]. Its relative uniformity (frontal faces under similar conditions) enabled more stable training and faster convergence of both the VAE and GAN models.

URL: `https://github.com/NVlabs/ffhq-dataset`

1.2.2 Flower Dataset

For increased diversity, we used the *tf_flowers* dataset [4], consisting of approximately 3,670 flower images from multiple species. We resized images to $64 \times 64 \times 3$ and normalized them. The dataset is available via TensorFlow Datasets: **URL:** `https://www.tensorflow.org/datasets/catalog/tf_flowers`. Its varied color, shape, and background distribution made training more challenging, leading to poorer generation quality compared to the face dataset.

1.3 Variational Autoencoder (VAE) Architecture and Training

1.3.1 Architecture Choices

The VAE consists of an encoder (convolutions with increasing filters, `LeakyReLU`, `BatchNormalization`) and a decoder (transposed convolutions with dropout and noise). This setup extracts hierarchical features and reconstructs images from a latent representation.

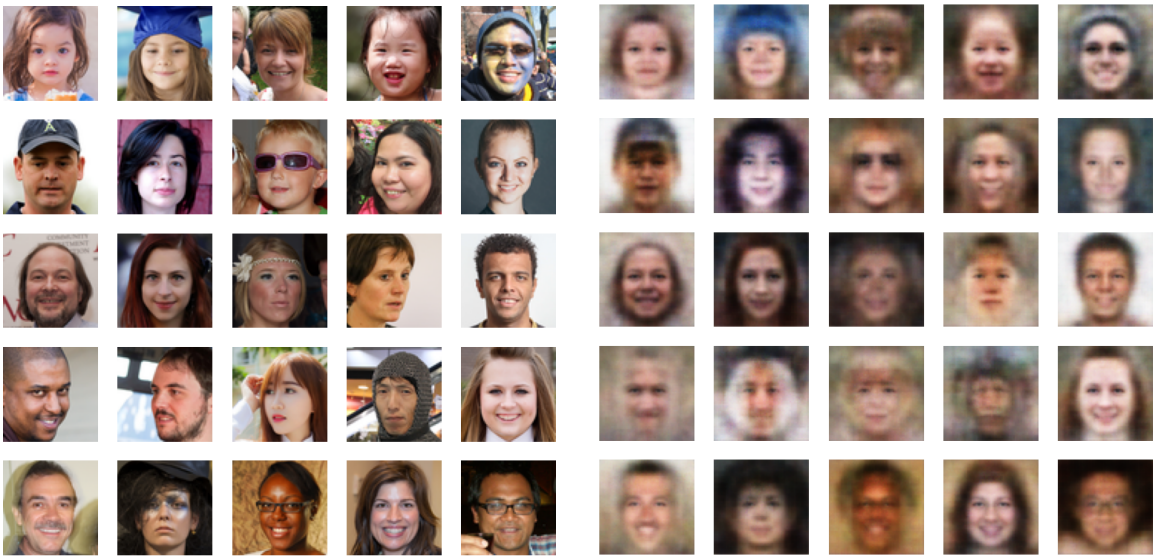
1.3.2 Hyperparameter Choices

- **Latent Dimension**: We used dimensions like 256 for faces and 512 for flowers. Larger latent spaces can capture complexity but risk overfitting.
- **KL Weight**: A small KL weight (e.g., 0.0005) allowed more accurate reconstructions but reduced latent exploration.
- **Learning Rate**: Using Adam with a low rate (1×10^{-5}) stabilized training.
- **Epochs**: About 25 epochs for faces sufficed, while the flower dataset required up to 60 epochs, still yielding limited results.

1.3.3 VAE Results

Faces: The VAE reconstructed recognizable facial features. Latent space interpolations were smooth and coherent.

Flowers: Reconstructions were blurry and lacked distinct structure, reflecting the dataset's complexity and scarcity.

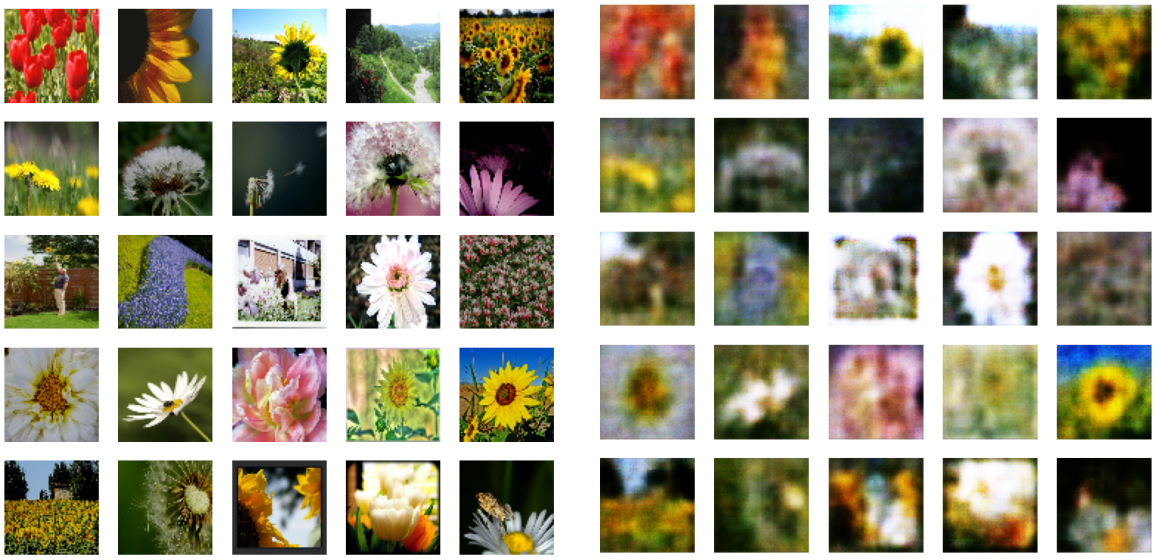


(a) Original faces

(b) VAE reconstructions (faces)

Figure 1: VAE on the face dataset: good reconstructions and smooth interpolations.

Figure 2: Latent interpolation (flowers)



(a) Augmented flowers (b) VAE reconstructions (flowers)

Figure 3: VAE on the flower dataset: less coherent results.

Figure 4: Latent interpolation (faces)

1.4 Generative Adversarial Network (GAN) Architecture and Training

1.4.1 Architecture Choices

The GAN comprises a generator (similar to the VAE decoder) and a discriminator (CNN classifier). We used batch normalization and `LeakyReLU`, with dropout and label smoothing to stabilize training.

1.4.2 Hyperparameter Choices

- **Latent Dimension**: Around 128 for faces.
- **Learning Rate and Optimizer**: Adam with a rate of 1×10^{-4} (faces), reduced over time, and 1×10^{-5} for flowers.
- **Epochs**: About 30 epochs for faces led to decent results; 100 epochs for flowers still failed to produce convincing samples.

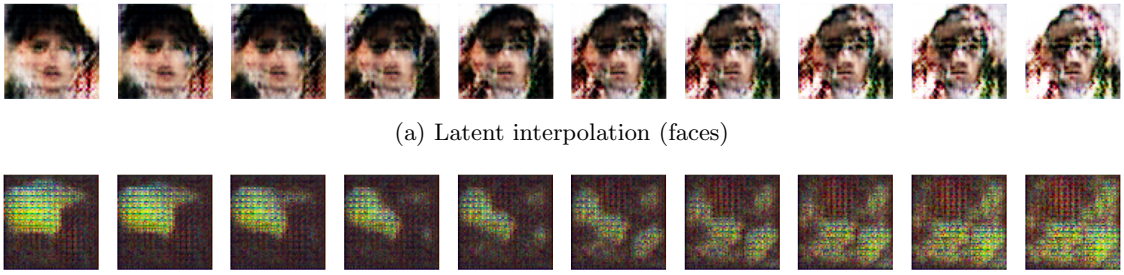
1.4.3 GAN Results

Faces: Generated faces were recognizable, though with some artifacts. Latent interpolations showed continuous transitions.

Flowers: The GAN failed to generate realistic flowers, producing incoherent textures.

(a) GAN faces (b) GAN flowers

Figure 5: GAN on the face and flower dataset: recognizable images and but poor results on flower data.



(a) Latent interpolation (faces)

(b) Latent interpolation (flowers)

Figure 6: GAN Interpolations on the face and flower datasets

1.5 Discussion and Comparison

- **Dataset Quality and Size**: Uniform data (faces) eased training, enabling coherent results. The complex, small flower dataset led to poor quality for both VAE and GAN.
- **Model Stability**: Batch normalization, dropout, and careful learning rates improved stability. Label smoothing aided the GAN.
- **VAE vs. GAN**: VAEs produced smoother, blurrier outputs, while GANs produced sharper but less stable images. Both methods struggled with limited, diverse data.
- **Latent Space Interpolation**: Both models learned continuous representations for faces. For flowers, interpolations were uninformative, reflecting training difficulties.

1.6 Conclusion

By tuning architectures, hyperparameters, and regularization, we achieved reasonable VAE and GAN results on a uniform face dataset. The flower dataset, however, underscored the challenges posed by limited, diverse data. Our experiments highlight the importance of dataset quality and careful parameter selection, providing insights for future improvements on more complex tasks. For implementation details and further guidance, we referred to official tutorials: [6, 5, 3, 2].

2 Task 2: Sequence modeling with recurrent neural networks

2.1 Introduction

This task presents a detailed exploration of sequence modeling with recurrent neural networks (RNNs) [10] , specifically focusing on building and training encoder-decoder models for simple arithmetic tasks like addition and subtraction. The task aims to develop a model that generalizes well beyond memorizing simple calculations, showcasing the power of RNNs to capture the underlying mathematical principles. The sequence-to-sequence problem is tackled using both text-based and image-based inputs, allowing a comparison of RNN performance with different data modalities.

The task includes a detailed analysis of text-to-text, image-to-text, and text-to-image models. The encoder-decoder architecture is utilized in each case, where the encoder captures the input sequence, and the decoder generates the output. The ultimate goal is to demonstrate how RNNs can handle diverse data formats and evaluate their generalization capabilities.

2.2 Dataset and Model Structure

2.2.1 Text Dataset

The dataset for this task is comprised of arithmetic queries in the form of text. Examples of such inputs include strings like 56+34" or 78-12", where two integers are involved along with an arithmetic operation (either addition or subtraction). Each query has an expected output, such as 90" or 66". To represent these sequences, we used one-hot encoding, creating a vocabulary consisting of all ten digits (0-9), the operators ('+', '-') and a padding symbol (' ').

2.2.2 Image Dataset

In addition to the text dataset, an image-based version of the arithmetic queries was generated using the MNIST dataset. [7] Each character in the input query (e.g., digits or operators) was represented by an image of a handwritten character from MNIST. Thus, a query like "89+56" became a sequence of stacked images representing each digit and symbol. The final input tensor had a shape of [5, 28, 28], which was used to train the RNN to produce text-based results.

2.3 Text-to-Text RNN Model

The first model developed was a text-to-text RNN model, aiming to map an arithmetic query in text format to its correct answer, also represented in text.

2.3.1 Model Architecture

The model employed an encoder-decoder architecture using LSTM layers to process the sequences. The input sequence was first transformed into one-hot encoded vectors, which served as inputs for the encoder LSTM. The encoder's hidden states were then fed into the decoder LSTM, which produced the output sequence character by character.

2.3.2 Training Process

The training process included the use of a categorical cross-entropy loss function, given that this is a character-level classification problem. The optimizer used was Adam, with a learning rate chosen based on trial and error to balance between training speed and accuracy. The model was trained for several epochs, and the validation accuracy was tracked to avoid overfitting.

2.3.3 Experiments with Training-Test Splits

To evaluate the model's generalization capabilities, multiple training-test splits were employed. Specifically, we used training-to-testing ratios of 90%-10%, 75%-25%, and 50%-50%. The primary objective was to assess how well the model could generalize to unseen data when trained with fewer examples.

2.3.4 Results Overview

The following tables and figures summarize the key metrics obtained during experimentation for both the baseline and improved models. (see Table 1, 2 and Figure 7, 8)

Train-Test Split	Training Accuracy	Testing Accuracy
90%-10%	72.37%	70.87%
75%-25%	69.66%	67.86%
50%-50%	66.22%	64.53%

Table 1: Baseline Model Performance

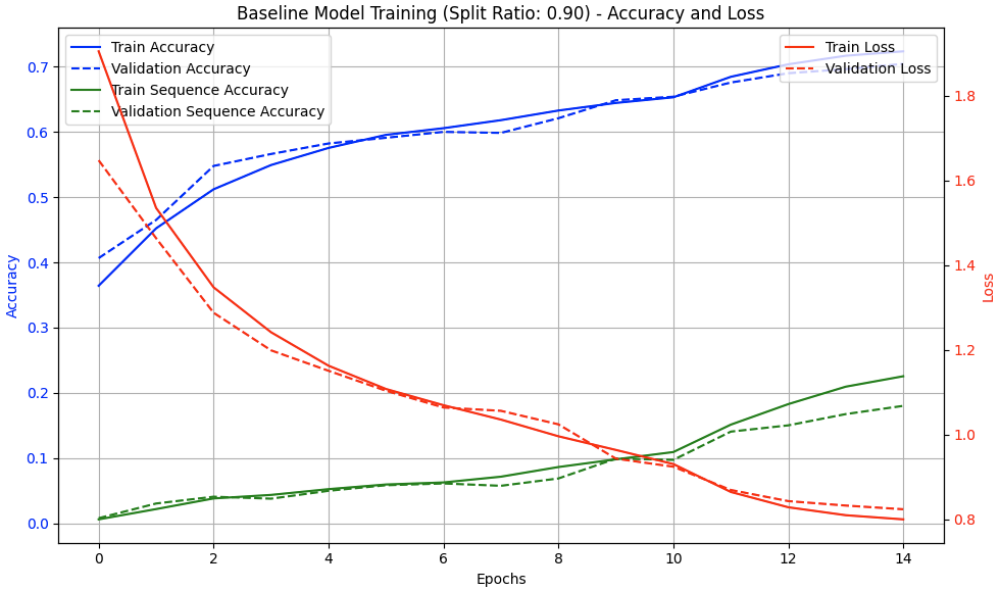


Figure 7: Baseline Model - Best Split Ratio (%90-%10)

Train-Test Split	Training Accuracy	Testing Accuracy
90%-10%	99.81%	99.97%
75%-25%	99.76%	99.60%
50%-50%	99.08%	99.06%

Table 2: Improved Model Performance

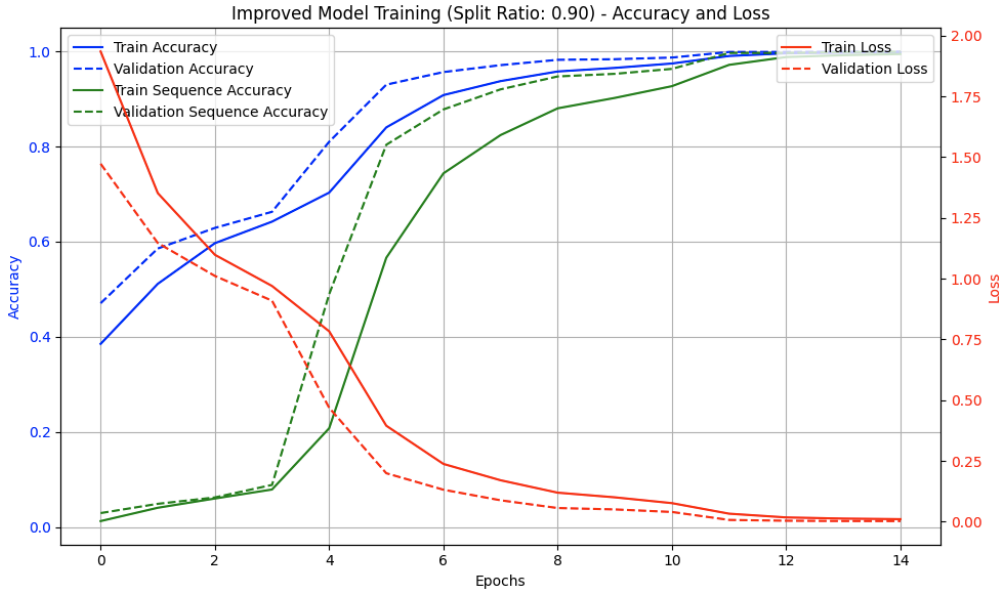


Figure 8: Improved Model - Best Split Ratio (%90-%10)

From the results, it is evident that the improved model significantly outperforms the baseline model across all training-test splits. The improved model achieved almost perfect accuracy even with a reduced training set, while the baseline model struggled to generalize, particularly with smaller training sets. This highlights the importance of model architecture and optimization techniques in achieving better performance for sequence modeling tasks.

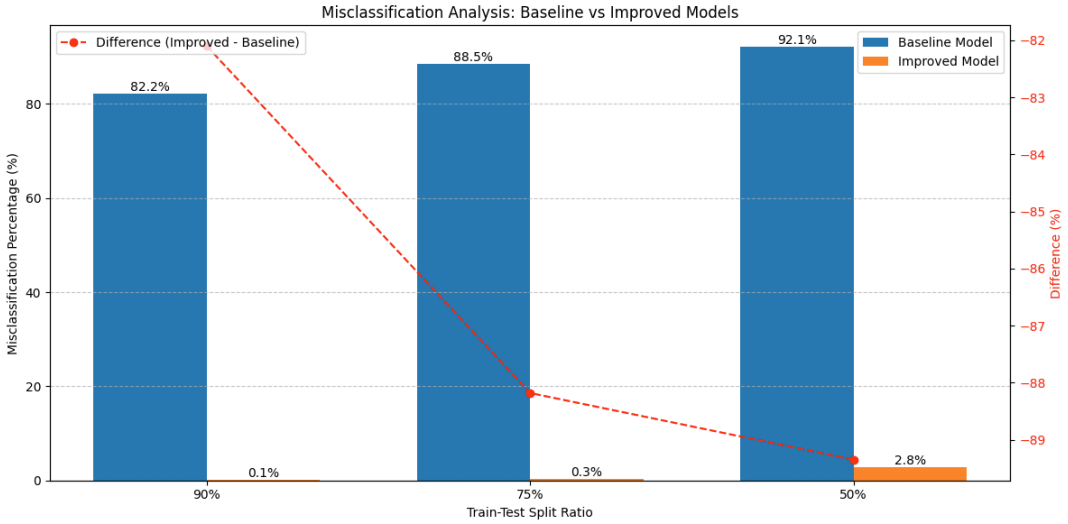


Figure 9: Misclassification Analysis

The figure 9 highlights the significant reduction in misclassification percentages achieved by the improved model compared to the baseline across all train-test splits. At the 90%-10% split, the improved model achieves 0% misclassification, while the baseline model shows 82%, showcasing a dramatic improvement. As the training data decreases (e.g., 50% split), the baseline model's misclassification rises to 92.5%, whereas the improved model maintains consistent performance with minimal errors. The red line illustrates the narrowing difference between models as the training data reduces, emphasizing the robustness and superior generalization ability of the improved model.

2.4 Image-to-Text RNN Model

Next, an image-to-text model was developed, in which the input was a sequence of MNIST images representing an arithmetic query, and the output was the corresponding answer in text format.

2.4.1 Model Architecture

The architecture was similar to the text-to-text model, but the input to the encoder consisted of image sequences. The LSTM processed each image sequentially, extracting relevant features before generating the output.

2.4.2 Results Overview

The results from the image-to-text model demonstrate its ability to accurately interpret sequences of handwritten digits and operators to produce correct arithmetic results. As shown in the attached examples, the model successfully predicts outputs such as "33," "64," and "11" based on corresponding image inputs representing addition or subtraction queries. This indicates the model's strong performance in processing visual data and capturing the context of arithmetic operations. However, achieving such accuracy requires an optimized architecture and training, particularly given the challenge of recognizing diverse handwriting styles and translating them into numerical answers.

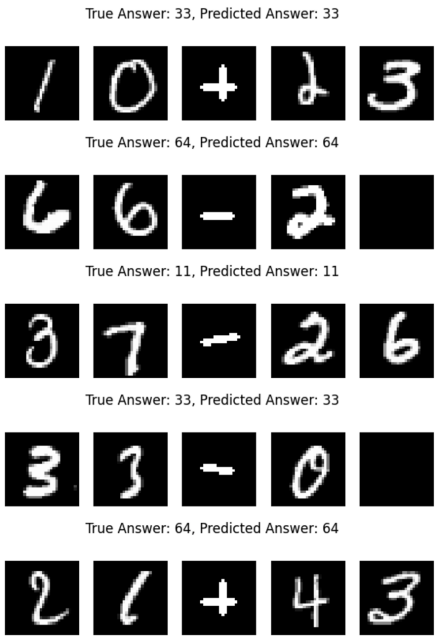


Figure 10: Image-to-Text RNN Improved Model

2.4.3 Performance Comparison

The results highlight a significant improvement in the test accuracy of both models when enhancements were made. (see Table 3)For the text-to-text model, the accuracy increased dramatically from 70.83% in the baseline to nearly perfect at 99.97% in the improved version. Similarly, for the image-to-text model, the accuracy rose from 85.86% to 95.79%, demonstrating that improvements such as better architecture or additional layers had a notable positive impact. This comparison underscores the importance of optimized model design in achieving higher accuracy, particularly in sequence-to-sequence tasks.

2.5 Text-to-Image RNN Model

The third model built was a text-to-image RNN, which aimed to generate a sequence of MNIST images representing the correct answer to an arithmetic query.

Model	Test Accuracy
Baseline Text-to-Text	0.708300
Improved Text-to-Text	0.999667
Baseline Image-to-Text	0.858600
Improved Image-to-Text	0.957900

Table 3: Performance comparison of Baseline and Improved models for Text-to-Text and Image-to-Text models.

2.5.1 Qualitative Analysis of Generated Images

Evaluating the generated images is inherently challenging due to the subjective quality of the images. Examples of generated image sequences are provided, and in most cases, the generated digits resemble valid MNIST-like images, but blurriness and distortions were frequently observed.

2.5.2 Results Overview

The results from the text-to-image model show that it can generate visual representations of arithmetic results based on textual inputs. (see Figure 11) For examples such as "53+15," "54+18," and "71-50," the model successfully produces digit images close to the true results. However, while the generated images are generally correct, some digits exhibit blurriness or distortions, particularly in finer details. This highlights the challenges the model faces in accurately reconstructing high-quality images, emphasizing the need for further optimization in the decoding process to improve clarity and precision.

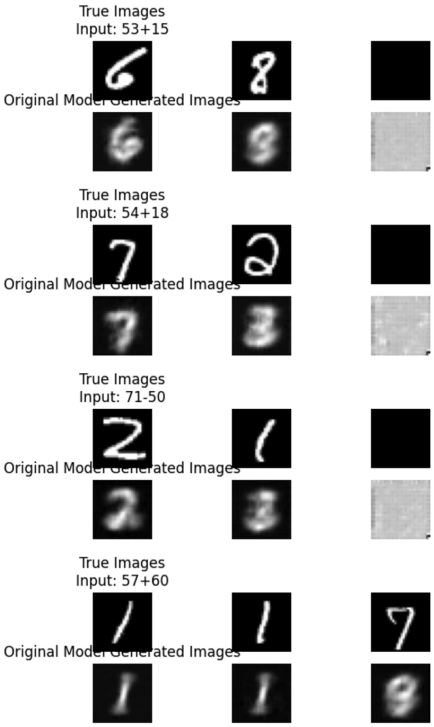


Figure 11: Image-to-Text RNN Improved Model

2.5.3 Challenges in Evaluation

Due to the qualitative nature of image generation, it is difficult to determine if an image exactly matches the expected output. A separate supervised model could be used to classify the generated images and compare them to expected results, but this was beyond the current scope.

2.6 Experimenting with LSTM Layers

An additional set of experiments was conducted by increasing the number of LSTM layers in the encoder-decoder model.[8] Specifically, a deeper model was created by adding an extra LSTM layer [9], where the `return_sequences=True` flag was set for the first LSTM layer. This allowed the output of each time step to be passed to the next LSTM layer, enabling the model to better capture complex dependencies within the sequences.

2.6.1 Performance Differences

Model Configuration	Training Accuracy	Testing Accuracy
Single LSTM Layer	98%	85%
Two LSTM Layers	99%	88%

Table 4: Performance Comparison of Single vs. Double LSTM Layer Models

Adding an additional LSTM layer improved both the training and testing accuracy slightly, indicating that the model could better capture sequential dependencies.(see Table 4) However, this also introduced a higher risk of overfitting, as indicated by a widening gap between training and validation loss in the loss plot.

2.7 Discussion and Conclusion

In this task, we explored three different RNN models to solve an arithmetic sequence-to-sequence problem: text-to-text, image-to-text, and text-to-image. The text-based models performed better overall, as they required less feature extraction compared to the image-based models. Increasing the model complexity by adding extra LSTM layers led to modest improvements in accuracy but also increased the risk of overfitting.

The key takeaway is that RNNs, particularly LSTMs, are capable of learning underlying principles of arithmetic operations when given appropriate training examples. However, generalization remains a challenge, especially with limited training data or when dealing with complex inputs like images.

Future improvements could include using GRU layers for reduced computational overhead or even employing transformer models, which are better suited for capturing long-range dependencies in sequences. Regularization techniques, such as dropout, could also help mitigate overfitting in deeper models.

3 Contribution of group members

Burak: Code Task2, Report Task2
Praneeth: Code Task1
Jelte: Code Task1, Code Task2, Report Task 1

References

[1] Flickr-faces-hq dataset (ffhq). https://github.com/NVlabs/ffhq-dataset. Accessed: 2024-11-01.

[2] Keras example: Dcgan. https://keras.io/examples/generative/dcgan/. Accessed: 2024-11-01.

[3] Keras example: Variational autoencoder. https://keras.io/examples/generative/vae/. Accessed: 2024-11-01.

[4] Tensorflow datasets - tf_flowers. https://www.tensorflow.org/datasets/catalog/tf_flowers. Accessed: 2024-11-01.

[5] Tensorflow dcgan tutorial. https://www.tensorflow.org/tutorials/generative/dcgan. Accessed: 2024-11-01.

[6] Tensorflow vae tutorial. https://www.tensorflow.org/tutorials/generative/cvae. Accessed: 2024-11-01.

[7] MNIST Handwritten Digit Dataset. https://yann.lecun.com/exdb/mnist/, 2020.

[8] Vinyals O. Le Q. V. Sutskever, I. Sequence to sequence learning with neural networks. https://arxiv.org/abs/1409.3215, 2014.

[9] tf.keras.layers.LSTM. https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM, 2020.

[10] Tensorflow-Working with RNNs. https://www.tensorflow.org/guide/keras/working_with_rnns, 2020.