Generative models and sequence modelling- Assignment 2

Ekansh Khanulia -s4336089
Marcell Nemeth - s4456394
Yanick Palmers - s4491890

December 8, 2025

Task 1: Generative Models

Task

The task is to learn how to train and use generative models for image-based applications. We are provided with an Jupyter notebook which serve as the foundation for understanding encoder/decoder structures, sampling-based generative mechanisms, and adversarial learning.In this task, we must use our own new image dataset, preprocess the images, and retrain both the VAE and the GAN . The goal is for the trained models to learn the underlying distribution of the data and eventually generate novel images that resemble the dataset.If necessary, the convolutional and deconvolutional networks can be adjusted to better suit the characteristics of the chosen dataset or to modify the model complexity.

After the models are trained ,as part of the required output, the goal includes creating a latent-space interpolation: a smooth visual transition between two randomly selected latent points, showing how the model morphs one generated image into another.

Dataset

We used the *CelebFaces Attributes (CelebA)* dataset obtained from Kaggle [1]. The full CelebA dataset contains 202,599 celebrity face images together with several other files. Since it is generative modeling, only the images were used.CelebA is suitable for the task because it has a diverse yet visually coherent collection of human faces. Although the dataset is uniform , it still contains natural variation like images differ in pose (frontal, left-facing, right-facing), illumination, background. This variability makes dataset an interesting test case for evaluating generative models .For this assignment, first 20,000 images were selected from the entire dataset and converted to an .npy file. Although the code of this conversion is provided in the submitted notebooks ,you can also download this file from here [2]) .

Generative Models

Before discussing the implementation and integration of GAN and VAE on the dataset we should understand how these architecture operate internally, and what distinguishes their working principles.

- Encoder & Decoder Architecture : The core of generative models is the encoder–decoder architecture. The encoder compresses the input image into a low-dimensional latent vector using convolutional layers that reduce resolution while retaining essential visual features. This vector serves as a compact summary of the image's structure. The **decoder** starting from this compact representation uses transposed convolutions to upsample and reconstruct the image.

- Variational Autoencoder : A Variational Autoencoder(VAE) extends the encoder–decoder idea by making the latent space *probabilistic* rather than deterministic. The encoder outputs two vectors instead of one: a mean μ and a variance σ^2. These define a Gaussian distribution for each input image. During training, a latent vector is sampled from this distribution using a trick:which draws random noise ,scales it by the learned variances and shifts it by learned mean. This enforces a smooth latent space where nearby points generate similar images.The decoder then receives this sampled latent vector z and does its work.The VAE is trained using a reconstruction loss (binary cross-entropy) to match the output to the input, and a KL-divergence term that pulls the latent distribution toward a standard normal prior. Balancing these two encourages coherent reconstructions, diversity in generated samples, and smooth interpolations across the latent space.

- Generative Adversarial Network : A Generative Adversarial Network reuses the same encoder–decoder architectures, but differently. The generator (decoder) starts from a low-dimensional latent vector and uses a stack of deconvolution layers to progressively upsample it into an image. The discriminator (encoder) applies standard convolution layers to an input image and compresses it into a single scalar representing the probability that the image is real. These two networks are trained in an adversarial game. The discriminator is optimized to distinguish real training images from fake ones produced by the generator, while the generator is trained to fool the discriminator into classifying its output as real. Formally, this is called minimax optimization. Through this , the generator learns to synthesize increasingly realistic images.

VAEs learn a probabilistic latent space and uses reconstruction loss and KL regularization to generate image, which results in smooth, interpretable latent spaces. GANs, however rely on the discriminator's feedback, which often produces sharper, more detailed images but a less structured latent space. VAEs emphazis continuity and representation learning; GANs emphasize realism through adversarial training.

Methodolgy

Processing CelebA dataset so that it can be used reliably across all generative models was the first step . Since the original images vary in size and quality,we convert each image to RGB, resize it to 64×64, and store it in a `.npy` file. This ensures uniformity and significantly reduces loading time during training, allowing the experiments to focus on the behaviour of the models rather than repeated data handling.

After this, the provided convolutional and deconvolutional network blocks were used as the backbone for all generative architectures.It was done to maintain consistency across the models and to isolate generative model behaviour when built on top of comparable feature extractors and decoders. The VAE introduces probabilistic encoding, while the GAN uses an adversarial discriminator–generator setup, but both rely on the same underlying convolutional structure.

During training, the VAE was monitored by decoding random latent vectors after each epoch to observe how well its latent space captured the overall distribution of faces. The GAN was evaluated in a similar way by generating new images from random noise, allowing visual inspection of whether the generator gradually learns to produce more realistic samples.

Finally, both models will be examined through latent-space interpolation, where two randomly chosen points are connected through a sequence of intermediate vectors. This step is included to evaluate whether the learned latent spaces are smooth and meaningful: a well-trained VAE or GAN should produce gradual, coherent transformations between the two endpoint images.

Experiments & Results

For VAE , the baseline configuration used were 64 filters , a latent dimension of 64, 50 epochs, and a batch size of 8 . We first increased the number of filters to 128 to improve reconstruction quality, but observed no meaningful change, so we focused on varying the latent dimension (32, 64, 128) (Figures 1 , 2, 3).Also the loss each latent dimensions is shown in Figure 5

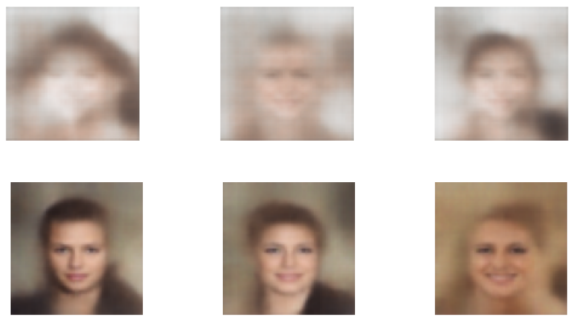
Figure 1: VAE outputs for latent dimension 32. Top: Epoch 0. Bottom: Epoch 49.

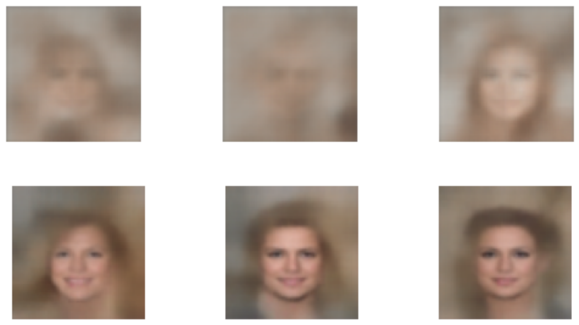
Figure 2: VAE outputs for latent dimension 64. Top: Epoch 0. Bottom: Epoch 49.

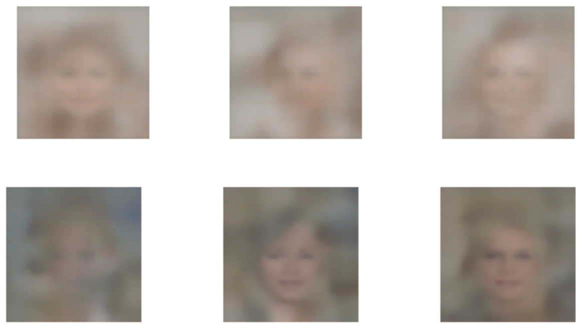
Figure 3: VAE outputs for latent dimension 128. Top: Epoch 0. Bottom: Epoch 49.

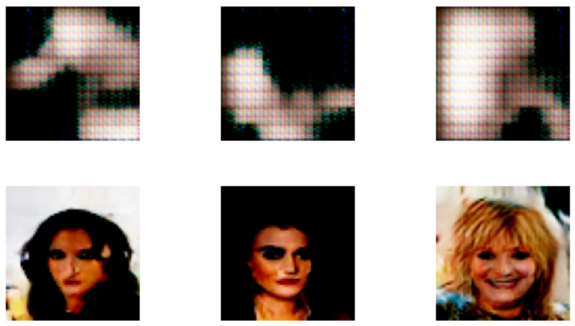
Figure 4: GAN outputs for latent dimension 256. Top: Epoch 0. Bottom: Epoch 49.

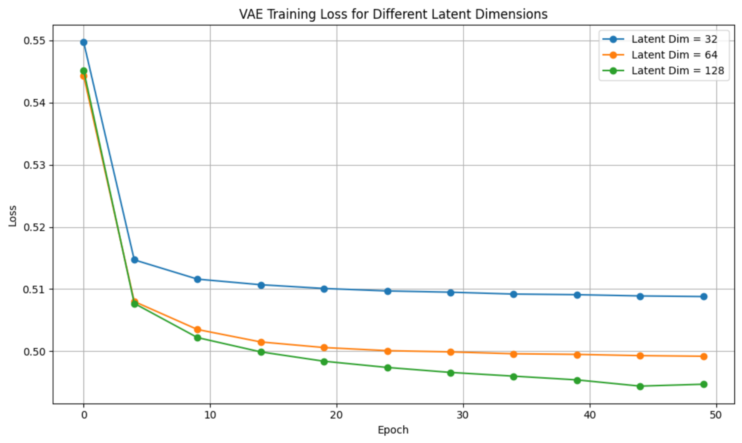

Figure 5: VAE loss curves for Latent Dimensions 32, 64, 128

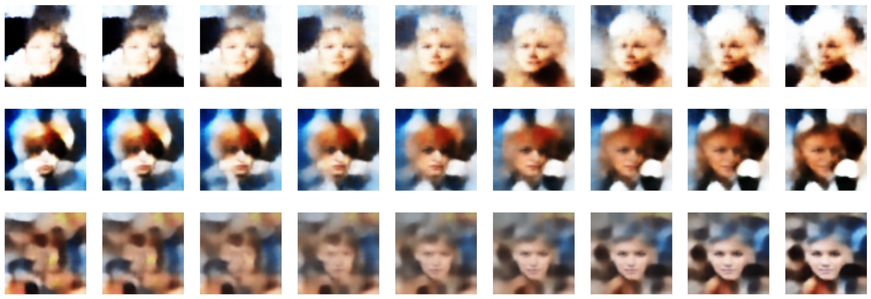

Figure 6: Latent-space interpolation for latent dimensions :32 (row 1), 64 (row 2), 128 (row 3).

For GAN, the baseline setup used was latent dimension of 256, 128 filters, 50 training epochs, and a batch size of 64. To study how latent space size affects generation stability and diversity, we tested our model onlatent dimensions 64 (Figure 7), 128 (Figure 8), and 256 (Figure 4) .

Additionally we produced latent interpolations for both the models and there respective configuration to compare smoothness and realism across different latent sizes (Figure 9).

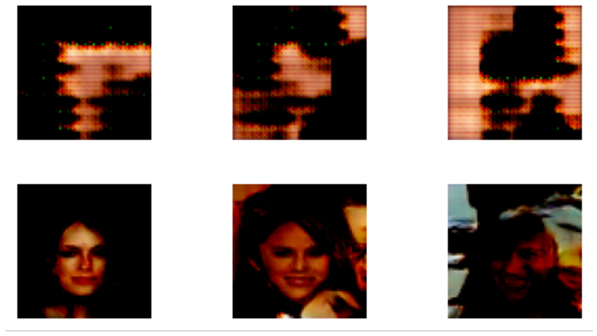

Figure 7: GAN :latent dimension 64. Top: Epoch 0. Bottom: Epoch 49.

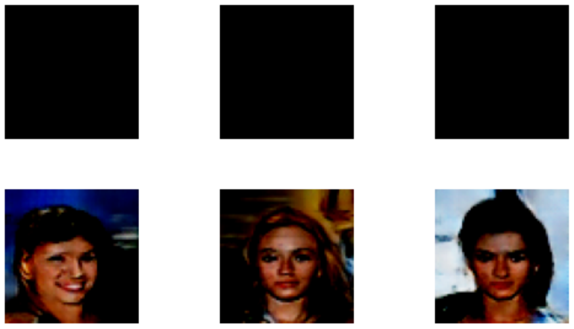

Figure 8: GAN : latent dimension 128. Top: Epoch 0. Bottom: Epoch 49.

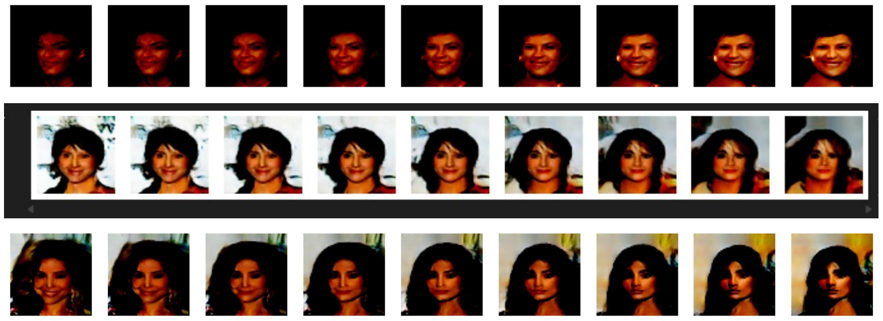

Figure 9: Latent-space interpolation for latent dimensions : 64 (row 1), 128 (row 2), 256 (row 3).

Discussion

Before discussing the results computed by these Generative models, it is important to consider the variability of the CelebA dataset used. The faces are not consistently aligned: many images show side profiles, tilted angles, or different distances from the camera. Therefore making reconstruction harder, since the model must learn both facial structure and large pose variation. As a result, part of the blur and instability in the generated outputs reflects dataset complexity rather than limitations of the model itself.

In case of VAE across latent sizes, 32-dimensional latent vectors produced most visually coherent outputs, even though they had the highest loss. Conversely, 128-dimensional models achieved the lowest loss but produced blurrier, less stable images. This mismatch occurs because the VAE loss mainly rewards distribution matching (via the KL term), not perceptual quality. A large latent space makes it easier for encoder to satisfy the KL constraint, reducing the total loss, but it also spreads information too thinly, causing over-smoothed reconstructions. A smaller latent space forces the model to compress only the most essential features, which improves visual sharpness despite a higher loss. Thus, in VAEs, lower loss does not necessarily correspond to better-looking samples, and compact latent spaces can yield superior visual quality. Interestingly, interpolations did not show strong differences across dimensions. All three latent sizes produced smooth transitions between the endpoint . A probable reasoning can be images sharpness varies with latent dimensionality, the global structure of the latent space remains continuous enough in all cases to support meaningful interpolation.VAE interpolation is performed by encoding two real images into latent vectors and interpolating between them.

On other hand, across latent dimensions (64, 128, and 256), GAN produced very visually similar outputs by the last training epoch. All configurations generated blurry, low-detail face-like structures, indicating that maybe 50 epochs were insufficient for the network to meaningfully exploit between latent-space sizes. The only clear distinction appeared at the beginning of training: the 128-dimensional model produced an almost completely black image at Epoch 0, reflecting an initial instability. By Epoch 49, however, this difference had disappeared. The GAN interpolation results show smooth transitions between the two latent vector across all latent dimensions. Although the generated images contain very limited detail, the intermediate frames change gradually rather than abruptly, indicating that the generator has learned a continuous latent space. GAN interpolation is done by interpolating between two randomly sampled noise vectors since GANs do not have an encoder.

Task 2: Sequence Modelling with Recurrent Networks

This task focuses on applying encoder-decoder neural architectures to sequence-to-sequence learning problems. The goal is to train recurrent models that can understand the underlying rules behind simple arithmetic operations and correctly map input sequences ,being it textual or image ,to their output. We have to work around three subtasks under this section.

Text-to-Text :

In the text-to-text scenario, the model receives a five-character arithmetic query such as `"81+24"` or `"41-89"` and must output the correct result, e.g., `"105"` or `"-48"`. Both inputs and outputs are represented using one-hot encodings. The purpose of this subtask is not to memorize all possible expressions, but to infer and generalize the principles of addition and subtraction. Different training-testing splits are explored to assess the model's generalisation capability on unseen queries.

Methods

The model's parameters are described in Table 1. To prepare the data to be fed into the LSTM model, each character in the input sequence was one-hot encoded into a thirteen-dimensional vector. Each entry in this vector represents either a number from zero through nine, a null space for padding, or a positive or negative sign. The null space is used for padding; for instance, an equation like `"1+17"` uses a null value in front of the `1` to maintain a fixed input length of five.

- Data Dimensions: The dataset contains a total of **20,000** arithmetic expressions. Each input sequence is stored in a tensor of shape **(20000, 5, 13)**, where **20,000** denotes the number of samples, **5** is the fixed length of the input query, and **13** is the one-hot encoding dimension covering digits, operators, and padding.

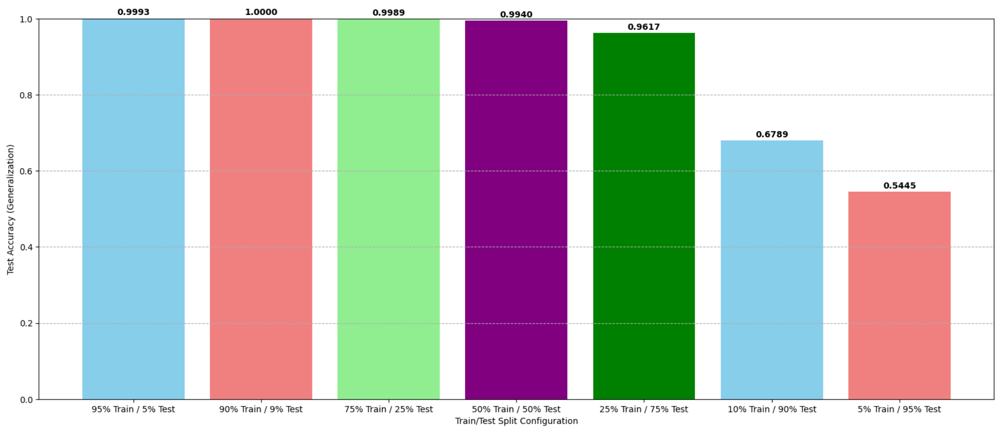

Figure 10: Accuracy of the text2text model with diffrent train/test splits

The corresponding output sequences have shape **(20000, 3, 13)**, reflecting the maximum **3**-character length of the results, encoded with the same **13**-dimensional representation.

- Model Architecture: The text–to–text model follows a standard encoder–decoder LSTM design. The encoder LSTM processes the 5-character input sequence and compresses it into a single 256-dimensional hidden representation. This latent vector is then repeated across three time steps using a `RepeatVector`, corresponding to the maximum answer length. The decoder LSTM generates the output sequence one character at a time, and a final `TimeDistributed(Dense(13, softmax))` layer predicts a probability distribution over the 13 possible symbols at each output time step

- Training Procedure: The model was trained with seven different train/test split ratios to assess performance with varying amounts of training data. Each setup was trained for one-hundred epochs, with a batch size of 32.

Results and Analysis

Across all 7 train-test splits, the model converges rapidly, reaching above 0.95 accuracy for splits with at least 25% training data. The accuracy curves in Figure 10 show that larger training split produce smoother,and stable learning , whereas smaller splits (10% and 5%) show slower convergence and higher variance.

Discussion

- Generalization Capability : The results clearly indicate that even when the model only sees 25 percent of the available data, the accuracy is still incredibly high (~ 0.96). This suggests that the model is not simply memorizing each equation, but is actually learning the underlying semantics and able to adapt to unseen data. Even when the model sees only 5 percent of the equations (only 1,000 out of 20,000 examples), it still achieves a ~ 0.54 accuracy.

- Observation of Overfitting : We observe in Figure 11 that for models with lower percent training data (specifically 10 percent and below), the validation accuracy plateaus while the train accuracy climbs to 1. This indicates overfitting when not enough training data is available, which can be explained by the model almost perfectly remembering what it saw but failing to generalize to new data.

Image-to-Text :

In this task, the input arithmetic expression is represented as a sequence of MNIST digit images . For example, the query `"89+56"` is converted into a sequence of five digit images using randomly selected MNIST

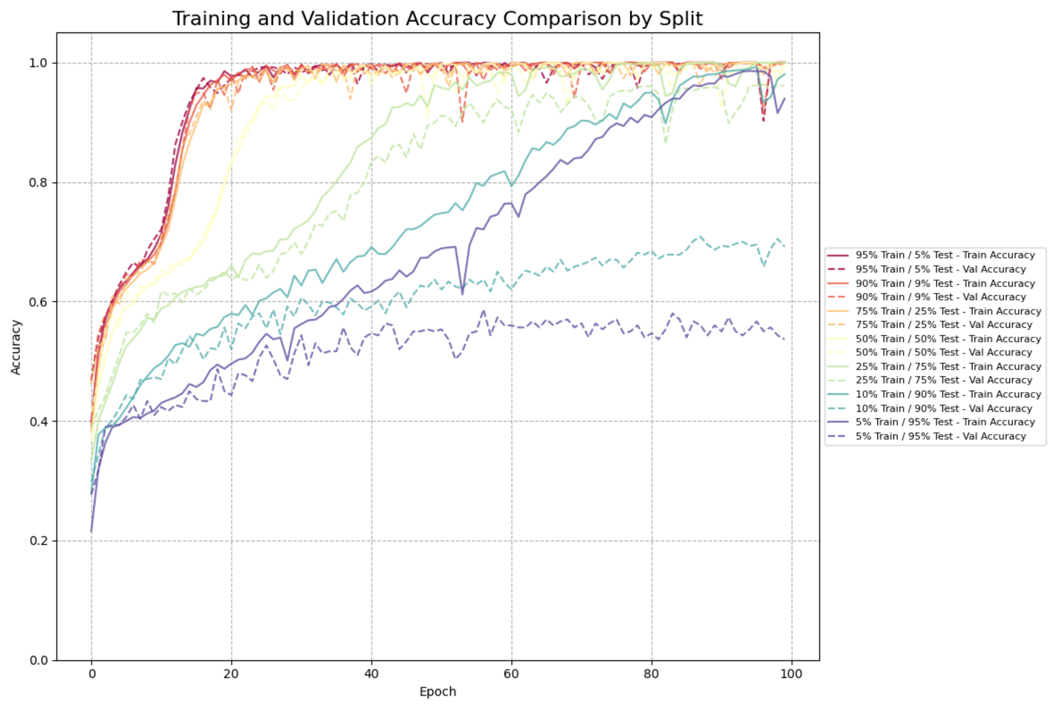

Figure 11: Accuracy of the text2text model with diffrent train/test splits

Table 1: Text-to-Text Model Architecture Summary

Layer (Type)	**Output Shape**	**Param #**
`lstm_2 (LSTM)`	`(None, 256)`	276,480
`repeat_vector_1 (RepeatVector)`	`(None, 3, 256)`	0
`lstm_3 (LSTM)`	`(None, 3, 256)`	525,312
`time_distributed_1 (TimeDistributed)`	`(None, 3, 13)`	3,341
Total params:		**805,133**

samples. The recurrent model must interpret this visual sequence, recognize the digits and operator, perform the arithmetic operation, and output the answer in text format. This evaluates the model's ability to combine visual recognition with symbolic reasoning.

Methods

For the input data in image to text model we again have a sequential input of length five , where each vector is size (28, 28, 1). So in simple terms, there are five grayscale images as input. Since the models we will use on this data have way more trainable parameters we want to make use of data augmentation to ensure there is no overfitting on the limited training data. So the first couple of layers of the networks are data augmentation layers. This makes the model more robust, by never seeing the same image sequence twice. The specific data augmentation layers are random rotation, random zoom, and random translation. The output of the model is the same as the text2text model, a (3, 13) vector where each vector is a one-hot encoded number.

There are atleast two ways to build the model, one where we flatten the images to (5, 784) and one where we make use of ConvLSTM2D layers to extract features out of the images. The parameter count in these models are very different, the model where we flatten the images has about 1.5 million trainable parameters, as seen in 2. The model with the convolutional layers is described in 3, here we can see differences in trainable parameters is about five million.

We train both models for one hundred epochs, with a batch size of 64.

Table 2: Flatten model layer configuration

Layer	Shape	Params
TD (Flatten)	(N, 5, 784)	0
LSTM Encoder	(N, 256)	1.07M
Dropout	(N, 256)	0
RepeatVector	(N, 3, 256)	0
LSTM Decoder	(N, 3, 256)	525,312
TD (Dense)	(N, 3, 13)	3,341

Table 3: Convolution model layer configuration

Layer	Shape	Params
ConvLSTM2D	(N, 28, 28, 32)	38,144
Flatten	(N, 25088)	0
Dense	(N, 256)	6.42M
Dropout	(N, 256)	0
RepeatVector	(N, 3, 256)	0
LSTM Decoder	(N, 3, 256)	525,312
TD (Dense)	(N, 3, 13)	3,341

Experiments & Results

The results of this training can be found in figure 12. Also, the importance of the data augmentation step can be seen in figure 13, where the training accuracy shoots up to 1, and the validation accuracy stays well below 0.5. While with the data augmentation added, the train- and validation accuracy stay close to each other and show consistent growth. After these initial experiments, we can clearly see that the convolutional layer outperforms the flatten image strategy. To explore the full potential of the convolution model we trained it for an additional one hundred epochs, since the training showed the accuracy was still increasing. After this final training of a total of two hundred epochs, we observed a .55 validation accuracy, and an accuracy that is no longer improving.

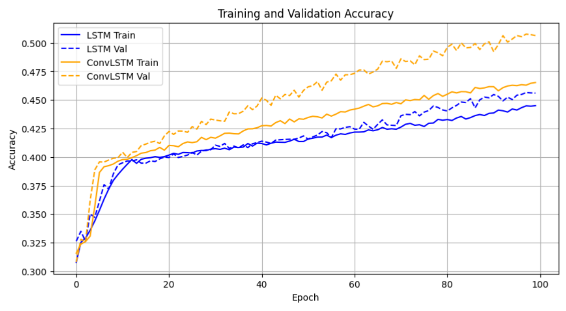

Figure 12: Accuracy of the image to text models with data augmentation

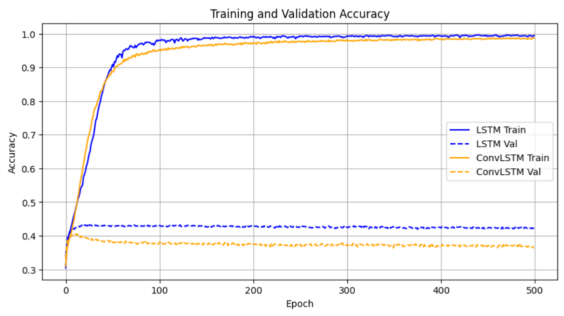

Figure 13: Models with no data augmentation

Discussion

In comparison, the image to text model is considerably less accurate than the text to text model, even tough there were many more trainable parameters, and longer training. where the text to text model got about .99 percent accuracy after only forty epochs and 800,000 parameters, and only seeing a couple of thousand examples. The convolutional image to text model was way bigger with about seven million trainable parameters and trained for two hundred epochs, the accuracy was still nowhere near .99. This is to be expected since each image can be mislabeled by the model, which gives it a hard time to predict the answer. And the input size differs quite a bit too (5, 13) for only 65 dimensions, in case of the text model, while the image model has to handle (28 x 28) x 5 = 3920 dimensions.

It is still impressive that a handwritten equation can be solved about fifty percent of the time, but if you want to take this to the next level you might want to first classify each image to a number, and then feed that to a deterministic calculator. That method might yield better results.

Text-to-Image :

Here, the model receives a text-based arithmetic query and must generate a sequence of MNIST-style images representing the correct answer. For instance, from the query `"89+56"`, the model should generate digit images forming the sequence `"145"`. Since numerical accuracy is less meaningful in this generative setting, evaluation is based primarily on qualitative inspection of the produced images. This task highlights the generative capabilities of recurrent encoder–decoder networks. A multitude of models were tested with a variety of hyperparameters (see **Effect of Additonal LSTM Layers**. Furthermore, Early stopping and Learning Rate adjustment were implemented as well.

Methods

- Dataset Construction: The dataset for the `text2image` model was constructed in the following way. As with the `text2text` model, first the arithmetic expressions were generated randomly. Then the expression is evaluated arithmetically, to get the ground-truth labels. For every digit in the ground-truth label an image is sampled from the MNIST dataset, which is then flattened to fit the output dimensions of the model. This procedure makes it possible to evaluate results on a pixel-to-pixel basis, measuring loss. For the `text2image` model, an 80% test and 20% validation split was used.

- Model Architecture: The `text2image` model follows an encoder–decoder structure using recurrent layers. The encoder consists of one or more stacked LSTM layers that compress the five-character one-hot query into a latent vector of size 256–512. The decoder repeats this vector three times and applies one or more LSTM layers before a TimeDistributed Dense layer outputs 28×28 pixel values for each timestep. The final output is reshaped into three MNIST-style digit images. Different architectural variants were explored by changing the number of encoder layers, decoder layers, and LSTM units.

Experiments & Results

- Error Analysis and Visualisation: As stated in the assignment, for the text-to-image model it was difficult to create a reliable evaluation metric. Therefore, random samples were taken from the training set to visualize some of the generated results. Figures 14 shows a sample output by the model, and Figure 15 presents the model's Training and Validation Loss/MAE curves over 12 epochs (with early stopping applied).

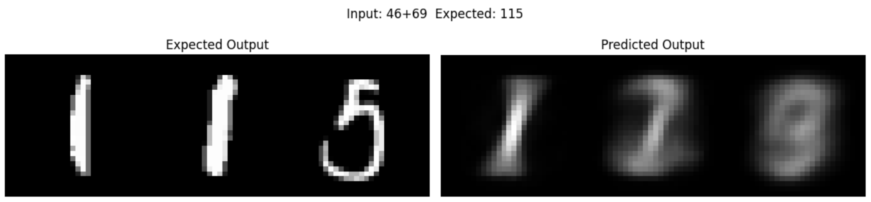

Figure 14: Sample output of `text2image` model, with input "46+69" and expected output "115"

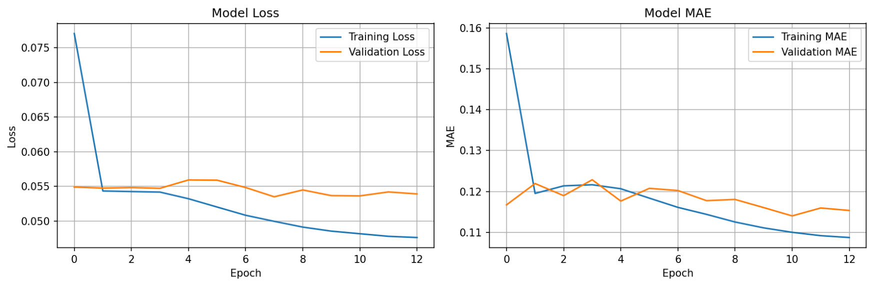

Figure 15: Model Training and Validation Loss/MAE over 12 epochs (early stopping).

- Effect of Additional LSTM Layers: As the results for the text-to-image model did not significantly improve over epochs (see the Training MAE/Loss curve vs. Validation MAE/Loss curve), further hyper-parameter tuning was performed. The parameters tested were the number of encoder layers, the number of decoder layers, and the number of units in the layers. The tested parameter ranges are summarized in Table 4 with their respective Validation Loss/MAE. The table shows that the best configuration was 1 encoder layer, 1 decoder layer, and 512 units, and no improvement was achieved by adding additional layers or changing the number of units.

Encoders	**Decoders**	**Units**	**Epochs**	**Params**	**Best Val Loss**	**Best Val MAE**
1	1	512	13	3,578,640	0.0535	0.1140
2	1	512	14	5,677,840	0.0546	0.1184
1	2	512	18	5,677,840	0.0545	0.1169
2	2	256	15	2,053,904	0.0546	0.1202
3	2	256	13	2,579,216	0.0546	0.1212

Table 4: `text2image` model parameters and results

Discussion & Conclusion

The performance of the text2image model, based on validation Loss/MAE scores and visual inspection, could not be improved in the scope of this task.Validation Loss/MAE did not significantly improved after 1 training epoch with any of the tried model configurations. This could be due to multiple factors. Based on visual inspection, it seems that the model is trying to predict amorphous "digits" that simply minimise pixel-to-pixel MAE rather than produce clear numerical structure. Compressing the original input to a 512-dimensional vector and then expecting a correct output in the shape of images can also lead to information loss during training. Furthermore, the experiments with additional LSTM layers showed no improvement, indicating that deeper architectures do not resolve this limitation and that the difficulty lies in the generative nature of the task itself.

Conclusion

Across all tasks, the effectiveness of sequence-to-sequence learning strongly depends the data modality and the model architecture. In Task 1, the VAE and GAN highlighted the challenges of generative modeling : VAEs produced smoother latent spaces but often blurry reconstructions influenced by different pose , while GANs generated similarly low-detail outputs across latent dimensions, suggesting that 50 epochs were not enough. In Task 2, the text-to-text model demonstrated excellent generalization, learning the underlying arithmetic structure even with very limited training data, though overfitting appeared for very small splits. The image-to-text model performed worse due to the high input dimensionality , while the text-to-image model struggled to generate clear outputs, with deeper LSTM architectures providing no improvement.

Contributions

Ekansh: Task 1 code, Task 1 report.
Yanick: Task 2 (text-to-text) code, Task 2 (text-to-text) report, Task 2(image to text) code, Task 2 (image to text) report.
Marcell Nemeth: Task 2 (text-to-image) code, Task 2 (text-to-image) report.

References

[1] Kaggle Contributor. Celebfaces attributes (celeba) dataset. `https://www.kaggle.com/datasets/jessicali9530/celeba-dataset`. Accessed: 2025-12-08.

[2] Ekansh. Preprocessed celeba subset (NPY file) used for task 1. `https://drive.google.com/file/d/14OlCaDKywA26I_ukngXSHWvmT8-O43EB/view?usp=drive_link`. Accessed: 2025-12-08.