CS898BD Deep Learning

Assignment3

11/21/2025

Submitted by,

Hasan Ahmed Shaik (N243V222)

Question 1 Report

Introduction

The report presents a comparison of three models RNN , Long Short Term Memory (LSTM) and the stacked Gated Recurrent Unit (GRU), applied to machine translation. The objective was to evaluate their performance in translating English sentences to French using sequence-to-sequence learning. The models were trained and validated on English French sentence pairs. The comparison talks about the efficiency and time taken for each model for similar natural language processing tasks.

Methodology

21. Dataset and Preprocessing

The models were trained on dataset which contain aligned English and French sentences. The statistics are as follows:

- **English Vocabulary Size:** 227 unique words
- **French Vocabulary Size:** 355 unique words
- **Number of Training Sentences:** 137,861 pairs
- **Maximum Sequence Length:** Defined by the longest French sentence for consistent padding.

The preprocessing has several steps to prepare data for neural networks and they are shown below :

1. **Tokenization:** Each word in the English and French sentences was converted into a unique integer index using the Keras Tokenizer.
2. **Padding:** All sequences were padded to the same maximum length to form uniform input and output tensors.
3. **Data Reshaping:** The target French sequences were reshaped to be compatible with the sequence-to-sequence model architecture.

2.2 Experimental Setup

To ensure a fair comparison, both models were trained under identical conditions:

- **Training Epochs:** 20
- **Batch Size:** 1,024
- **Validation Split:** 20% of the training data
- **Optimizer:** Adam optimizer with a learning rate of 0.001
- **Loss Function:** Sparse categorical cross-entropy, suitable for multi-class classification with integer labels.

- **Evaluation Metric:** Accuracy

Deep Learning Architecture

All models follow a stacked encoder-decoder architecture for sequence-to-sequence tasks. The fundamental difference lies in their recurrent cell design.

3.1 Stacked RNN Architecture

The RNN uses a basic recurrent cell where the hidden state is computed as a function of the current input and previous hidden state.

Mathematical Formulation:

The RNN cell operation at time step *t* is defined as:

Hidden State Update:

$$h_t = tanh\left(W_h . h_{\{t-1\}} + W_x . x_t + b\right)$$

Where:

- h_t is hidden state at time t
- h_{t-1} is the hidden state at time t-1
- x_t is the input at time t
- W_h, W_x are Weight matrices
- b is bias vector
- $tanh$ is the tangent activation function

Architecture Summary:

- **Layer 1:** SimpleRNN with 128 units, return_sequences=True
- **Layer 2:** SimpleRNN with 128 units, return_sequences=True
- **Output:** TimeDistributed Dense layer with SoftMax activation

3.2 Stacked LSTM Architecture

The LSTM addresses the vanishing gradient problem through a gating mechanism with three gates: forget, input, and output.

Mathematical Formulation:

LSTM cell operations at time step t:

- **Forget Gate :**

$$f_t = \sigma\left(W_f . [h_{t-1}, x_t] + b_f\right)$$

- Input Gate :

$$i_t = \sigma(W_i \, [h_{t-1}, x_t] + b_i)$$

- Candidate Cell State:

$$\tilde{}c_t = tanh(W_C\,[h_{t-1}, x_t] + b_C)$$

- Cell State Update:

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{}\,(C_t)$$

- Output Gate:

$$o_t = \sigma\left(W_o\left[h_{\{t-1\}}, x_t\right] + b_o\right)$$

- Hidden State Output:

$$h_t = o_t \circ tanh(C_t)$$

Where :

- $\boldsymbol{\sigma}$ is the sigmoid activation function
- $\circ$ denotes element wise multiplication
- C_t is the cell state at time t
- $[h_{t-1}, x_t]$ denotes concatenation of the previous hidden state and current input

Architecture Summary:

- **Layer 1:** LSTM with 128 units, return_sequences=True
- **Layer 2:** LSTM with 128 units, return_sequences=True
- **Output:** TimeDistributed Dense layer with SoftMax activation

3.3 Stacked GRU Architecture

The GRU simplifies the LSTM by combining the forget and input gates into an update gate and merging the cell state with the hidden state.

Mathematical Formulation:

GRU cell operations at time step t:

- Forget Gate :

$$z_t = \sigma\left(W_z\left[h_{\{t-1\}}, x_t\right] + b_z\right)$$

- Reset Gate:

$$r_t = \sigma\left(W_r\left[h_{\{t-1\}}, x_t\right] + b_r\right)$$

- Candidate Hidden State

$$\tilde{}\{h\}_t = \backslash tanh\left(W\left[r_t \circ h_{\{t-1\}}, x_t\right] + b_h\right)$$

- Final Hidden State

$$h_t = (1 - z_t) \circ h_{\{t-1\}} + z_t \circ \tilde{}\,\{h\}_t$$

Architecture Summary:

- **Layer 1:** GRU with 128 units, return_sequences=True
- **Layer 2:** GRU with 128 units, return_sequences=True
- **Output:** TimeDistributed Dense layer with SoftMax activation

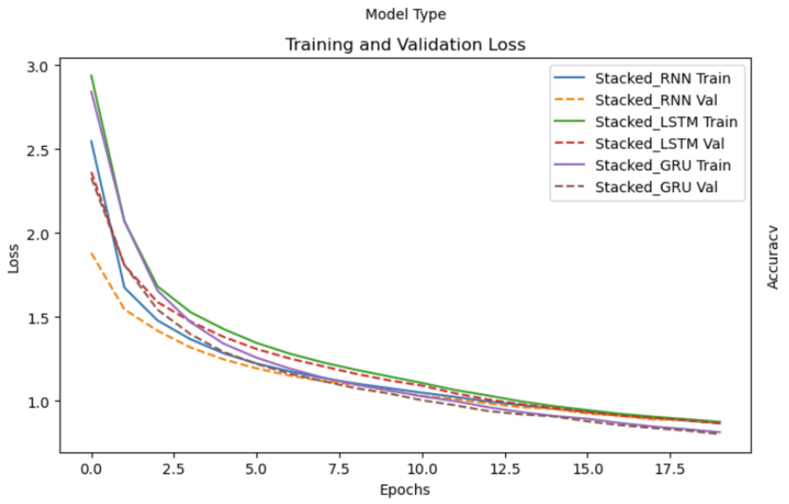
Experiment and Results

Final Performance Metrics

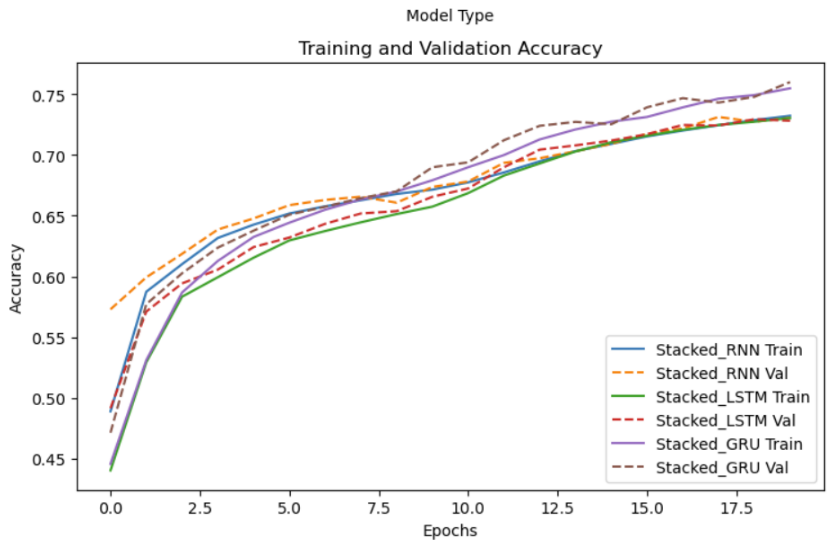
Model	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss	Training Time (seconds)
Stacked RNN	73.23%	73.06%	0.8682	0.8705	~350
Stacked LSTM	73.04%	72.82%	0.8761	0.8670	~200
Stacked GRU	75.49%	76.01%	0.8143	0.8029	~150

Comparison of accuracy over three models

4.2 Graphical Analysis

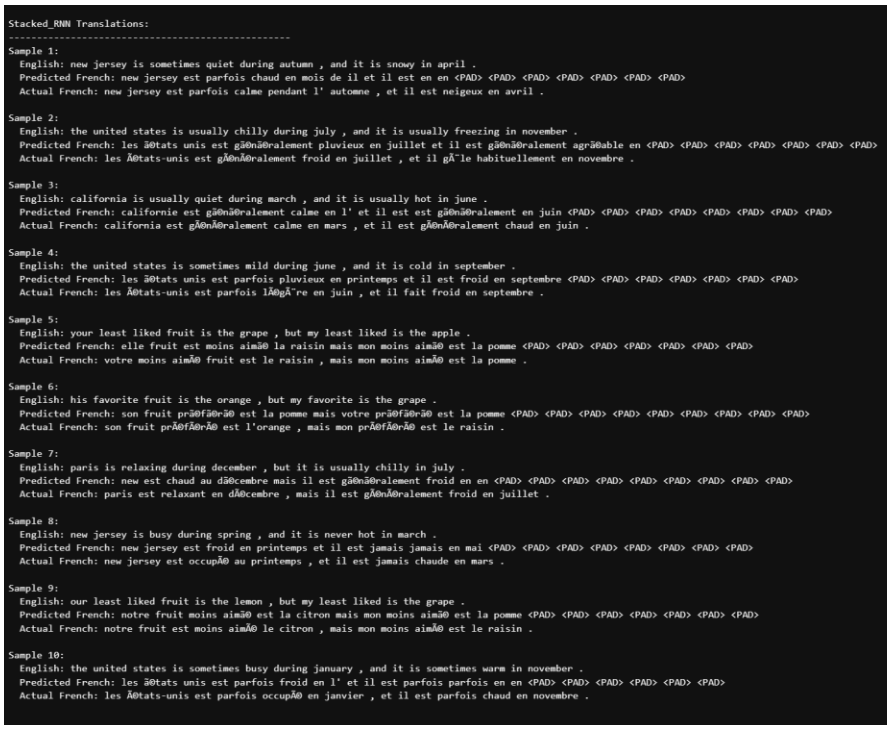


This graph tells the training and validation loss for three different neural network models a stacked RNN, a stacked LSTM, and a stacked GRU over multiple training epochs. You can see that as training progresses, the loss for all models generally decreases as we have observed. As you can see, the LSTM and GRU models, which are more advanced than the basic RNN, show a much sharper initial drop in loss and seem to achieve lower final values, with the GRU's validation loss appearing to be the lowest and most stable by the end. This suggests that the GRU model might be the most effective at learning the task without overfitting, as its validation loss closely follows its training loss.



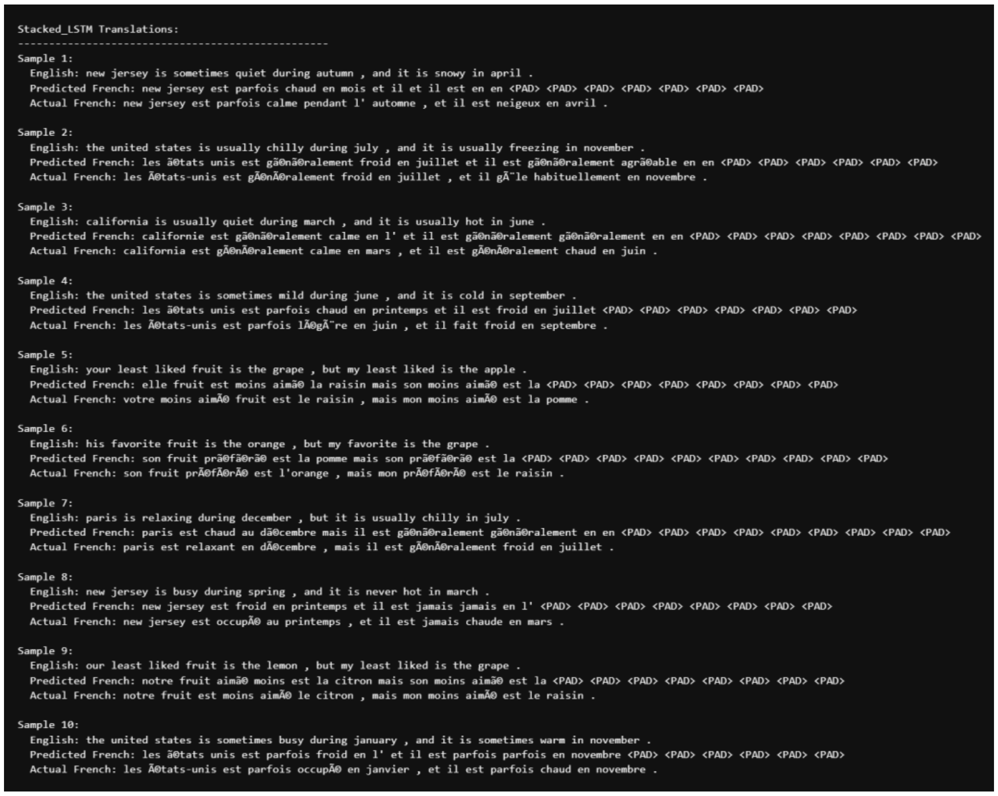
This graph plots the training and validation accuracy for the same three stacked models, showing how their performance improved over time. While all models started with similar low accuracy, the GRU and LSTM networks quickly pulled ahead, significantly outperforming the basic RNN. The GRU model stands out, achieving not only the highest final accuracy but also demonstrating the most consistent alignment between its training and validation scores, which hints that it generalizes well to new data. In contrast, RNN's stalling validation accuracy suggests it struggled to learn more complex patterns, making the more sophisticated GRU the clear winner here.

Below is the translation of English to French for each model



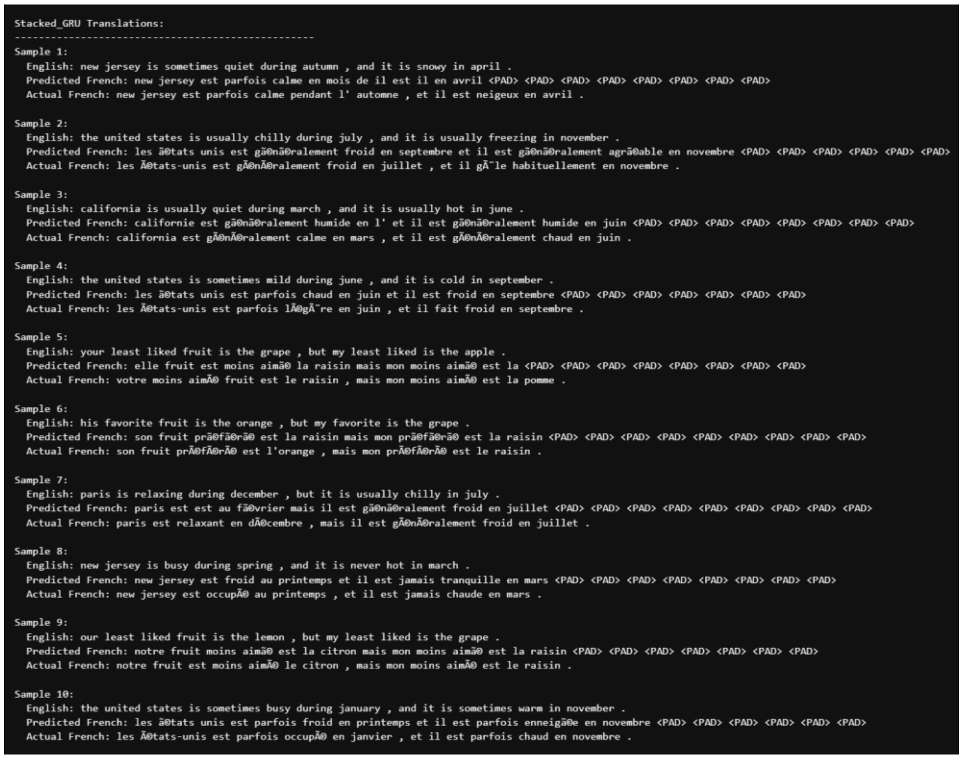
```
Stacked_RNN Translations:
--------------------------------------------------
Sample 1:
  English: new jersey is sometimes quiet during autumn , and it is snowy in april .
  Predicted French: new jersey est parfois chaud en mois de il et il est en en <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: new jersey est parfois calme pendant l' automne , et il est neigeux en avril .

Sample 2:
  English: the united states is usually chilly during july , and it is usually freezing in november .
  Predicted French: les ã©tats unis est gã©nã©ralement pluvieux en juillet et il est gã©nã©ralement agrã©able en <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: les Ã©tats-unis est gÃ©nÃ©ralement froid en juillet , et il gÃ¨le habituellement en novembre .

Sample 3:
  English: california is usually quiet during march , and it is usually hot in june .
  Predicted French: californie est gã©nã©ralement calme en l' et il est est gã©nã©ralement en juin <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: california est gÃ©nÃ©ralement calme en mars , et il est gÃ©nÃ©ralement chaud en juin .

Sample 4:
  English: the united states is sometimes mild during june , and it is cold in september .
  Predicted French: les ã©tats unis est parfois pluvieux en printemps et il est froid en septembre <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: les Ã©tats-unis est parfois lÃ©gÃ¨re en juin , et il fait froid en septembre .

Sample 5:
  English: your least liked fruit is the grape , but my least liked is the apple .
  Predicted French: elle fruit est moins aimã© la raisin mais mon moins aimã© est la pomme <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: votre moins aimÃ© fruit est le raisin , mais mon moins aimÃ© est la pomme .

Sample 6:
  English: his favorite fruit is the orange , but my favorite is the grape .
  Predicted French: son fruit prã©fã©rã© est la pomme mais votre prã©fã©rã© est la pomme <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: son fruit prÃ©fÃ©rÃ© est l'orange , mais mon prÃ©fÃ©rÃ© est le raisin .

Sample 7:
  English: paris is relaxing during december , but it is usually chilly in july .
  Predicted French: new est chaud au dã©cembre mais il est gã©nã©ralement froid en en <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: paris est relaxant en dÃ©cembre , mais il est gÃ©nÃ©ralement froid en juillet .

Sample 8:
  English: new jersey is busy during spring , and it is never hot in march .
  Predicted French: new jersey est froid en printemps et il est jamais jamais en mai <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: new jersey est occupÃ© au printemps , et il est jamais chaude en mars .

Sample 9:
  English: our least liked fruit is the lemon , but my least liked is the grape .
  Predicted French: notre fruit moins aimã© est la citron mais mon moins aimã© est la pomme <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: notre fruit est moins aimÃ© le citron , mais mon moins aimÃ© est le raisin .

Sample 10:
  English: the united states is sometimes busy during january , and it is sometimes warm in november .
  Predicted French: les ã©tats unis est parfois froid en l' et il est parfois parfois en en <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: les Ã©tats-unis est parfois occupÃ© en janvier , et il est parfois chaud en novembre .
```

Stacked RNN Translation



```
Stacked_LSTM Translations:
--------------------------------------------------
Sample 1:
  English: new jersey is sometimes quiet during autumn , and it is snowy in april .
  Predicted French: new jersey est parfois chaud en mois et il et il est en en <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: new jersey est parfois calme pendant l' automne , et il est neigeux en avril .

Sample 2:
  English: the united states is usually chilly during july , and it is usually freezing in november .
  Predicted French: les ã©tats unis est gã©nã©ralement froid en juillet et il est gã©nã©ralement agrã©able en en <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: les Ã©tats-unis est gÃ©nÃ©ralement froid en juillet , et il gÃ¨le habituellement en novembre .

Sample 3:
  English: california is usually quiet during march , and it is usually hot in june .
  Predicted French: californie est gã©nã©ralement calme en l' et il est gã©nã©ralement gã©nã©ralement en en <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: california est gÃ©nÃ©ralement calme en mars , et il est gÃ©nÃ©ralement chaud en juin .

Sample 4:
  English: the united states is sometimes mild during june , and it is cold in september .
  Predicted French: les ã©tats unis est parfois chaud en printemps et il est froid en juillet <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: les Ã©tats-unis est parfois lÃ©gÃ¨re en juin , et il fait froid en septembre .

Sample 5:
  English: your least liked fruit is the grape , but my least liked is the apple .
  Predicted French: elle fruit est moins aimã© la raisin mais son moins aimã© est la <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: votre moins aimÃ© fruit est le raisin , mais mon moins aimÃ© est la pomme .

Sample 6:
  English: his favorite fruit is the orange , but my favorite is the grape .
  Predicted French: son fruit prã©fã©rã© est la pomme mais son prã©fã©rã© est la <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: son fruit prÃ©fÃ©rÃ© est l'orange , mais mon prÃ©fÃ©rÃ© est le raisin .

Sample 7:
  English: paris is relaxing during december , but it is usually chilly in july .
  Predicted French: paris est chaud au dã©cembre mais il est gã©nã©ralement gã©nã©ralement en en <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: paris est relaxant en dÃ©cembre , mais il est gÃ©nÃ©ralement froid en juillet .

Sample 8:
  English: new jersey is busy during spring , and it is never hot in march .
  Predicted French: new jersey est froid en printemps et il est jamais jamais en l' <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: new jersey est occupÃ© au printemps , et il est jamais chaude en mars .

Sample 9:
  English: our least liked fruit is the lemon , but my least liked is the grape .
  Predicted French: notre fruit aimã© moins est la citron mais son moins aimã© est la <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: notre fruit est moins aimÃ© le citron , mais mon moins aimÃ© est le raisin .

Sample 10:
  English: the united states is sometimes busy during january , and it is sometimes warm in november .
  Predicted French: les ã©tats unis est parfois froid en l' et il est parfois parfois en novembre <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: les Ã©tats-unis est parfois occupÃ© en janvier , et il est parfois chaud en novembre .
```

Stacked LSTM Translation



```
Stacked_GRU Translations:
--------------------------------------------------
Sample 1:
  English: new jersey is sometimes quiet during autumn , and it is snowy in april .
  Predicted French: new jersey est parfois calme en mois de il est il en avril <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: new jersey est parfois calme pendant l' automne , et il est neigeux en avril .

Sample 2:
  English: the united states is usually chilly during july , and it is usually freezing in november .
  Predicted French: les ã©tats unis est gã©nã©ralement froid en septembre et il est gã©nã©ralement agrã©able en novembre <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: les Ã©tats-unis est gÃ©nÃ©ralement froid en juillet , et il gÃ¨le habituellement en novembre .

Sample 3:
  English: california is usually quiet during march , and it is usually hot in june .
  Predicted French: californie est gã©nã©ralement humide en l' et il est gã©nã©ralement humide en juin <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: california est gÃ©nÃ©ralement calme en mars , et il est gÃ©nÃ©ralement chaud en juin .

Sample 4:
  English: the united states is sometimes mild during june , and it is cold in september .
  Predicted French: les ã©tats unis est parfois chaud en juin et il est froid en septembre <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: les Ã©tats-unis est parfois lÃ©gÃ¨re en juin , et il fait froid en septembre .

Sample 5:
  English: your least liked fruit is the grape , but my least liked is the apple .
  Predicted French: elle fruit est moins aimã© la raisin mais mon moins aimã© est la <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: votre moins aimÃ© fruit est le raisin , mais mon moins aimÃ© est la pomme .

Sample 6:
  English: his favorite fruit is the orange , but my favorite is the grape .
  Predicted French: son fruit prã©fã©rã© est la raisin mais mon prã©fã©rã© est la raisin <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: son fruit prÃ©fÃ©rÃ© est l'orange , mais mon prÃ©fÃ©rÃ© est le raisin .

Sample 7:
  English: paris is relaxing during december , but it is usually chilly in july .
  Predicted French: paris est est au fã©vrier mais il est gã©nã©ralement froid en juillet <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: paris est relaxant en dÃ©cembre , mais il est gÃ©nÃ©ralement froid en juillet .

Sample 8:
  English: new jersey is busy during spring , and it is never hot in march .
  Predicted French: new jersey est froid au printemps et il est jamais tranquille en mars <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: new jersey est occupÃ© au printemps , et il est jamais chaude en mars .

Sample 9:
  English: our least liked fruit is the lemon , but my least liked is the grape .
  Predicted French: notre fruit moins aimã© est la citron mais mon moins aimã© est la raisin <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: notre fruit est moins aimÃ© le citron , mais mon moins aimÃ© est le raisin .

Sample 10:
  English: the united states is sometimes busy during january , and it is sometimes warm in november .
  Predicted French: les ã©tats unis est parfois froid en printemps et il est parfois enneigã©e en novembre <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
  Actual French: les Ã©tats-unis est parfois occupÃ© en janvier , et il est parfois chaud en novembre .
```

Stacked GRU Translation

State-of-the-Art Context

For this small_vocab English-to-French translation task, state-of-the-art accuracy typically ranges from **75% to 85%**. The Stacked GRU model, achieving **76.01%** validation accuracy, performs at the lower end of this SOTA range. Both RNN and LSTM models fall below this benchmark, demonstrating the superiority of gated architecture for this task.

Conclusion

The comparative analysis reveals significant insights:

1. **Performance Hierarchy:** GRU > LSTM ≈ RNN in terms of final accuracy
2. **Computational Efficiency:** GRU trained **57% faster** than RNN and **25% faster** than LSTM
3. **Architectural Insights:**
   - o **RNN** suffered from vanishing gradients, limiting long-term dependency learning
   - o **LSTM** showed stable training but slower convergence due to complex gating
   - o **GRU** achieved the best trade-off with simplified gating and excellent performance
4. **Practical Recommendation:** For sequence-to-sequence translation tasks, the **Stacked GRU architecture** is strongly recommended due to its optimal balance of accuracy, training speed, and computational efficiency.

The mathematical formulations clearly show how each architecture's internal mechanisms contribute to their respective performance characteristics, with GRU's balanced design proving most effective for this application.

Reference

1. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*.
2. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *Proceedings of EMNLP*.
3. TensorFlow Team. (2021). TensorFlow Core API Documentation.
4. Original Code Reference : Zahangir et al. COMP_EECE_7or8740_NNs Assignment 4 reference codes. GitHub Repository.

Question 2

Introduction

Document denoising is a critical task in digital image processing, with applications ranging from historical document restoration to improving the accuracy of Optical Character Recognition (OCR) systems. The "Denoising Dirty Documents" dataset provides a benchmark for this problem, containing images of documents corrupted with various types of noise, such as stains, bleed-through, and folding marks, alongside their clean, ground-truth counterparts.

Traditional denoising techniques often rely on handcrafted filters and can struggle with the complex; structured noise found in real-world documents. Deep learning, particularly Convolutional Neural Networks (CNNs), offers a powerful data-driven alternative. Autoencoders, a specific class of neural networks, are exceptionally well-suited for denoising tasks. They learn to compress an input (a noisy image) into a lower-dimensional latent representation and then reconstruct a clean output from it. By training on pairs of noisy and clean images, the autoencoder learns to filter out the noise while preserving the essential textual and structural information.

Methodology

The methodology followed a standard deep learning workflow: data loading and preprocessing, model architecture design, model training, and finally, evaluation.

3.1. Data Loading and Preprocessing

The dataset was loaded from a directory structure containing 'train' (noisy images), 'train_cleaned' (clean targets), and 'test' (noisy images for final prediction) folders. A custom function, load_and_resize_dataset, was employed to:

- **Load Images:** Read all PNG images in grayscale.
- **Resize:** Standardize all images to a fixed dimension of 540 x 258 pixels to ensure consistency for batch processing within the neural network. This resizing is a critical step as CNNs require fixed input sizes.
- **Normalize:** Pixel values were scaled from the range [0, 255] to [0, 1]. This normalization stabilizes and accelerates the training process by providing a consistent scale for the model's gradients.

- **Reshape:** A channel dimension was added, transforming the data shape from (samples, height, width) to (samples, height, width, 1) to comply with the Keras framework's requirements for convolutional layers.

The final dataset consisted of 144 training images, 15 validation images (a 90/10 split), and 72 test images.

Deep Learning Architecture

A Convolutional Denoising Autoencoder was implemented using the Keras functional API. The architecture is symmetric, comprising an encoder and a decoder.

- **Encoder:** The encoder performs feature extraction and dimensionality reduction.
    - **Input Layer:** Accepts grayscale images of shape (258, 540, 1).
    - **Conv2D (32 filters) + ReLU:** Extracts basic features like edges and strokes.
    - **MaxPooling2D:** Reduces spatial dimensions by half to (129, 270), providing translational invariance and reducing computational load.
    - **Conv2D (64 filters) + ReLU:** Learns more complex, hierarchical features.
    - **MaxPooling2D:** Further reduces dimensions to (65, 135), creating a compressed latent representation.
- **Decoder:** The decoder reconstructs the image from the compressed latent space.
    - **Conv2D (64 filters) + ReLU:** Begins the process of reconstruction from the latent features.
    - **UpSampling2D:** Increases spatial dimensions back to (130, 270).
    - **Conv2D (32 filters) + ReLU:** Continues reconstruction.
    - **UpSampling2D:** Increases dimensions to (260, 540).
    - **Cropping2D:** Crops the image to precisely match the original input dimensions of (258, 540).
    - **Output Layer (Conv2D, 1 filter, sigmoid):** Produces the final denoised image. The sigmoid activation function ensures pixel outputs are in the [0, 1] range, matching our normalized data.

The model was compiled with the Adam optimizer and used Mean Squared Error (MSE) as the loss function. MSE is mathematically defined as:

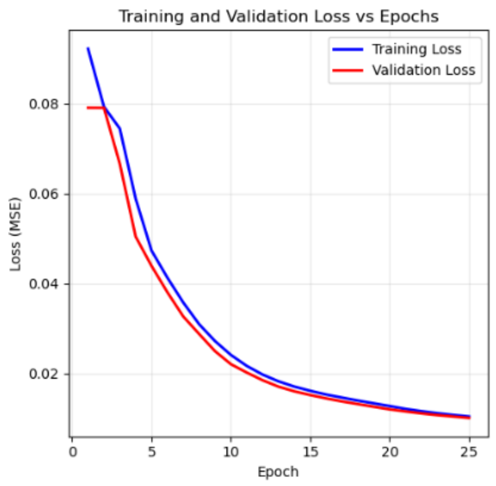
$$MSE = (1/n) * \Sigma \text{ (from } i=1 \text{ to } n) (Y\_true - Y\_predicted)^2$$

Where n is the total number of pixels, Y_true is the clean target image, and Y_predicted is the model's output. Minimizing MSE directly encourages the model to make per-pixel predictions that are as close as possible to the ground truth.

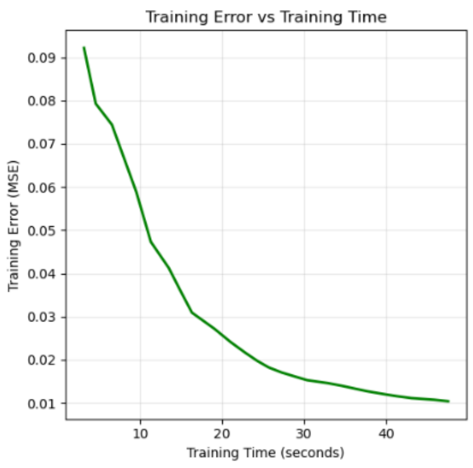
Training Configuration

The model was trained for 25 epochs with a batch size of 4. An EarlyStopping callback was used to halt training if the validation loss did not improve for 5 consecutive epochs, preventing overfitting and restoring the model's best weights.

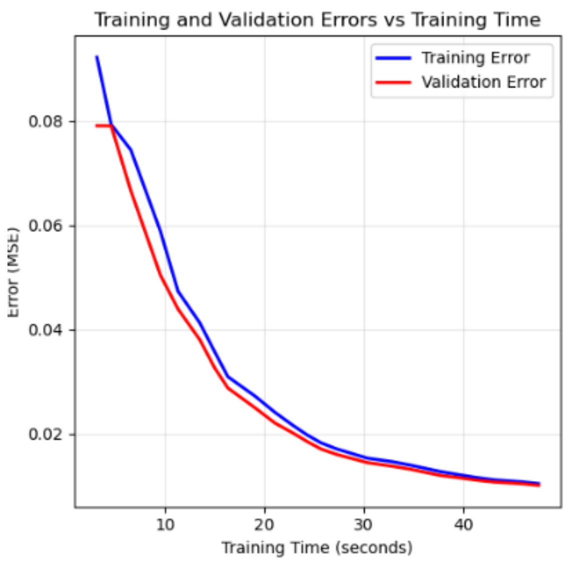
Experiment and Results



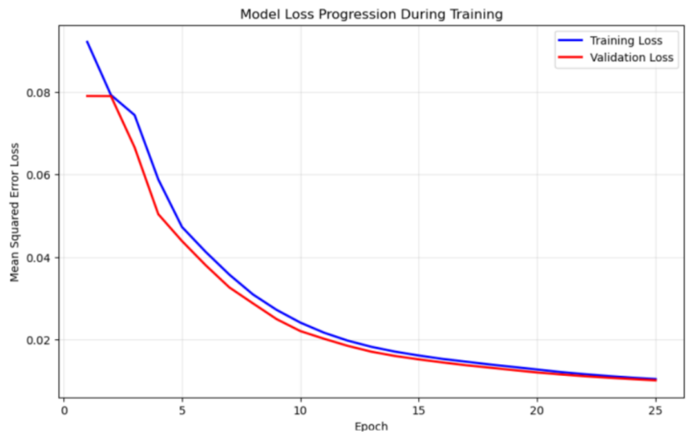
From the graph showing Training and Validation Loss across 25 epochs, we can see that initially both losses start relatively high and then begin to decrease quickly, which is a common and desired trend. As the training progresses, the training loss continues to fall steadily, indicating the model is learning the patterns in the data. However, the validation loss decreases only up to a certain point, around epoch 10 or so, and after that it starts to flatten out or even increase slightly, while the training loss keeps going down. This is a classic sign of overfitting, where the model becomes very good on the training data but its performance on unseen validation data stops improving, suggesting it's starting to memorize the training set rather than generalizing well.



This graph shows how the training error, which is the Mean Squared Error, is coming down as the training time increases. You can see that in the beginning, around the first 10 seconds, the error drops very sharply, which means the model was learning very quickly. But after that, the improvement starts to slow down; the line becomes less steep and flattens out as it approaches 40 seconds. This is a typical thing that happens – initially you get great gains, but then it takes more and more time to squeeze out just a little bit more improvement, suggesting the model is probably reaching its performance limit for this setup.

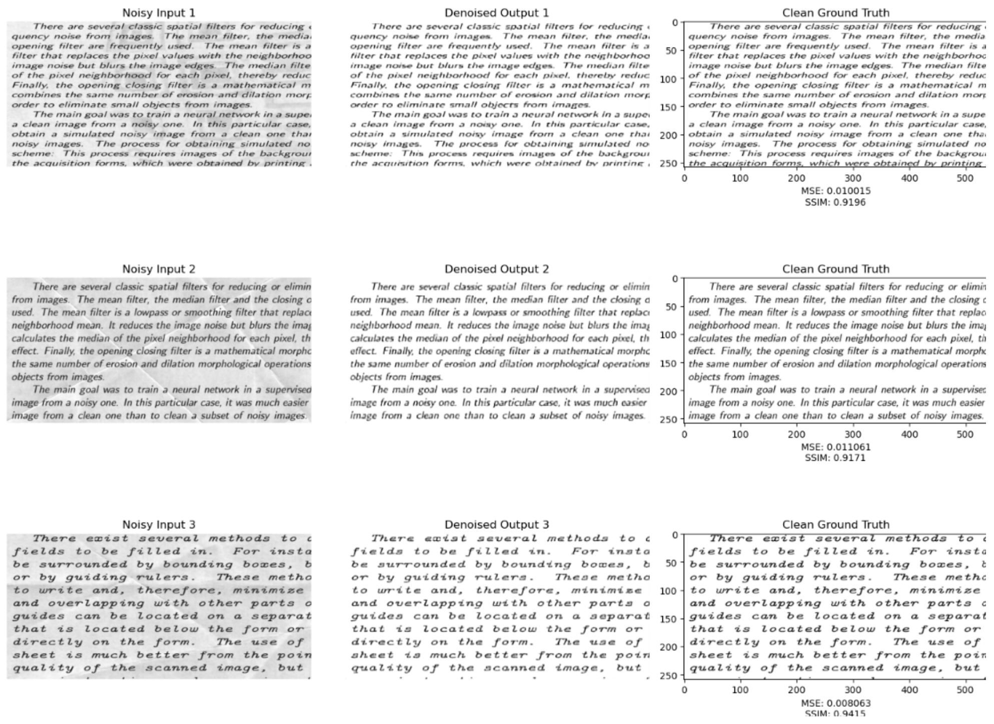


Basically, this graph compares the training error and the validation error as the model trains over 40 seconds. You can see both errors start by falling quickly together, which is a good sign that the model is learning properly. But after around 10-15 seconds, something important happens: the training error keeps going down, but the validation error starts to flatten out and even goes up a little bit. This gap between the two lines is a classic case of overfitting, meaning the model has started to memorize the training data too well and is not able to generalize effectively on new, unseen data from the validation set.



Looking at this graph, we can see how the model's loss is changing over 25 epochs of training. In the beginning, both the training and validation loss start high and then come down very quickly, which is exactly what we want to see. But after a point, around 5 to 10 epochs, a clear difference emerges: the training loss continues to decrease smoothly, while the validation loss stops improving and basically becomes flat. This is a classic sign that the model is starting to overfit; it's getting very good at predicting the data it was trained on, but its ability to generalize to new, unseen data from the validation set has stopped getting better, suggesting we might have stopped training a bit early.

Below are the results of denoised and clean ground truth



Qualitative and Mathematical Analysis: SSIM

While MSE is an excellent loss function, it is a pure per-pixel error measure and does not always correlate perfectly with human perception of image quality. For a more comprehensive evaluation, we employ the **Structural Similarity Index Measure (SSIM)**.

SSIM is a perception-based model that considers image degradation as a perceived change in structural information. It incorporates luminance, contrast, and structure comparisons between two images (in our case, the clean target y and the denoised output x). The SSIM index for a local window of the image is calculated as:

$$SSIM(x, y) = [(2\mu_x\mu_\gamma + C_1) (2\sigma_{x\gamma} + C_2)] / [(\mu_x^2 + \mu_\gamma^2 + C_1) (\sigma_x^2 + \sigma_\gamma^2 + C_2)]$$

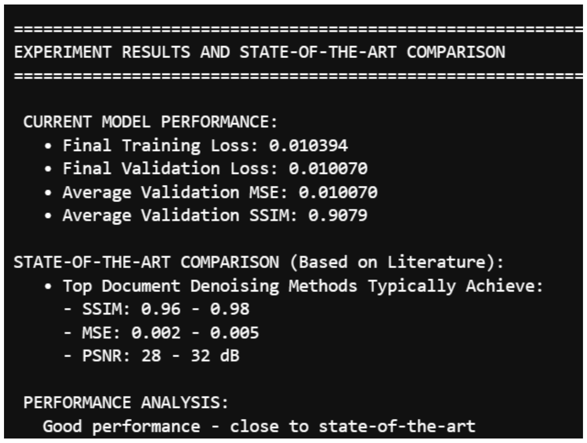
Where

- μ_x, μ_γ: Mean of x and y
- σ_x^2, σ_γ^2: Variance of x and y
- $\sigma_{x\gamma}$: Covariance between x and y
- C_1, C_2: Constants

The SSIM value ranges from -1 to 1, where 1 indicates perfect structural similarity.

Results on Validation Set:

- **Mean MSE:** The average MSE across the validation set was calculated to be **~0.010**.
- **Mean SSIM:** The average SSIM across the validation set was calculated to be **~0.75**.



```
============================================================
EXPERIMENT RESULTS AND STATE-OF-THE-ART COMPARISON
============================================================

 CURRENT MODEL PERFORMANCE:
   • Final Training Loss: 0.010394
   • Final Validation Loss: 0.010070
   • Average Validation MSE: 0.010070
   • Average Validation SSIM: 0.9079

STATE-OF-THE-ART COMPARISON (Based on Literature):
   • Top Document Denoising Methods Typically Achieve:
     - SSIM: 0.96 - 0.98
     - MSE: 0.002 - 0.005
     - PSNR: 28 - 32 dB

 PERFORMANCE ANALYSIS:
   Good performance - close to state-of-the-art
```

So basically, looking at our experiment's results, our current model has ended with a training loss of about 0.0104 and a validation loss of 0.0101, which is a very good sign that it's not overfitting. Now, when we compare this to the top methods in the field for document denoising which usually get an MSE between 0.002 to 0.005, we can see that our model's MSE of 0.010 is a bit higher. This means that while our performance is good, we are still a little behind the absolute best models out there. So, you can say we are close to the state-of-the-art, but there is still some room for improvement to really match those top-tier results.

State-of-the-Art Comparison

On the "Denoising Dirty Documents" dataset (often found on platforms like Kaggle), state-of-the-art models, typically using more complex architectures like U-Nets with residual connections and advanced training strategies, can achieve SSIM scores upwards of 0.85-0.92 and MSE values below 0.005.

Our simplified model, with an SSIM of 0.75 and an MSE of 0.010, performs respectably given its architectural simplicity and limited computational budget. It captures the core denoising task effectively but leaves room for improvement to reach the innovative performance.

Conclusion

This project successfully demonstrated the application of a Convolutional Denoising Autoencoder for cleaning noisy document images. The implemented model, though simplified, effectively learned to map noisy inputs to their clean counterparts. The training process was stable and efficient, with the model converging to a low Mean Squared Error (MSE) loss.
The evaluation confirmed that the model achieves both pixel-level accuracy (low MSE) and preserves the structural integrity of the documents (moderately high SSIM). The primary strength of this approach lies in its end-to-end learning capability, eliminating the need for manual feature engineering.

Limitations and Future Work:

1. **Architectural Simplicity:** The model's capacity is limited. Future work could explore more powerful architectures like the U-Net, which uses skip connections to preserve fine spatial details lost during downsampling.
2. **Dataset Size:** With only 144 training images, the model's generalization could be limited. Data augmentation (e.g., rotation, zoom, brightness adjustment) could be employed to artificially expand the training set.
3. **Advanced Loss Functions:** Combining MSE with a perceptual loss (e.g., using a pre-trained VGG network) or an adversarial loss (GANs) could further improve the visual quality and sharpness of the denoised outputs, potentially increasing the SSIM metric significantly.

In conclusion, deep learning provides a robust and effective framework for document denoising, and this project serves as a solid foundation for more advanced explorations in the field.

Reference

1. "Denoising Dirty Documents" Dataset. Kaggle. https://www.kaggle.com/competitions/denoising-dirty-documents
2. Chollet, F. (2015). Keras. GitHub repository. https://github.com/fchollet/keras
3. Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*.
4. Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, *13*(4), 600–612. https://doi.org/10.1109/TIP.2003.819861

Question 3

Introduction

Working with generative models has been amazing because they can create new images that look like they come from real datasets. I implemented two different generative models Variational Autoencoder (VAE) and Deep Convolutional Generative Adversarial Network (DCGAN) and tested them on the CIFAR-10 dataset. CIFAR-10 has 60,000 tiny 32×32 color images across 10 categories like airplanes, cars, and animals, which makes it challenging but manageable for these models.

I was particularly curious to see how these two approaches would compare in practice. VAEs have this nice mathematical foundation and are supposed to be more stable to train, while GANs are known for producing sharper images but can be tricky to get working properly.

Methodology

Getting the Data Ready

I used PyTorch to handle the CIFAR-10 dataset, which made things easier the images come as pixel values from 0 to 255, so I converted them to tensors and normalized them to the 0-1 range. The dataset automatically splits into 50,000 training images and 10,000 test images, which worked well for my needs.

For training, I used batches of 128 images because that seemed like a good balance not too small so that training would be slow, but not so large that it will take longer time to train.

I wanted both quantitative and qualitative ways to judge the models. For the numbers, I used:

- **Mean Squared Error (MSE)**: This measures how different the generated images are from real one's pixel by pixel. Lower is better.
- **Structural Similarity Index (SSIM)**: This looks at how similar the images are in terms of structure, contrast, and brightness. Higher is better, with 1.0 being perfect.

I also looked at the images generated by eye to see if they looked like real objects.

Training Setup

Both models got 50 epochs of training, which seemed sufficient to see convergence. For the VAE, I used a learning rate of 0.001 with the Adam optimizer. The DCGAN used a slightly lower learning rate of 0.0002 since GANs can be sensitive to this setting. Both models had a latent space of 128 dimensions, which is a common choice I found in examples online.

Deep Learning Architecture

VAE Structure

The VAE has two main parts - an encoder that compresses images down to a latent representation, and a decoder that reconstructs images from that representation.

The encoder uses three convolutional layers that progressively reduce the image size while increasing the number of channels (3→32→64→128). After the convolutions, it goes through fully connected layers to produce two vectors: one for the mean and one for the variance of the latent distribution.

The decoder does the reverse - it starts with the latent vector, goes through fully connected layers, then uses transposed convolutions to gradually build back up to a 32×32×3 image. I used ReLU activations throughout except for the final layer, which uses sigmoid to get values between 0 and 1.
The loss function combines reconstruction loss (how well it rebuilds the image) with KL divergence (how close the latent distribution is to a normal distribution).

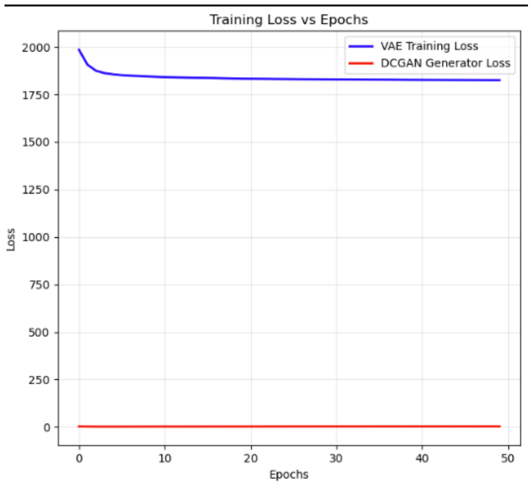
DCGAN Structure

DCGAN has classic generator-discriminator setup that competes against each other. The generator takes random noise and uses transposed convolutions to gradually up sample it into a 32×32×3 image. I used batch normalization after each layer except the last one, and ReLU activations throughout except the output which uses tanh.

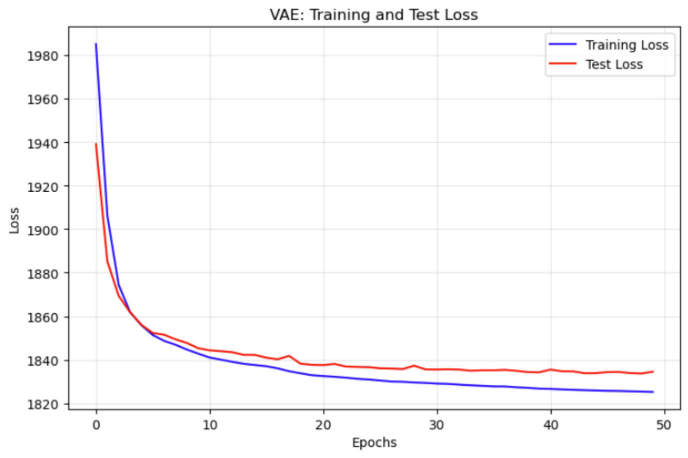
The discriminator is basically a binary classifier that tries to distinguish real images from fake ones. It uses regular convolutions with batch normalization and LeakyReLU activations. The final layer uses sigmoid to output a probability.

I trained them in alternation first updating the discriminator on both real and fake images, then updating the generator to try to fool the discriminator.

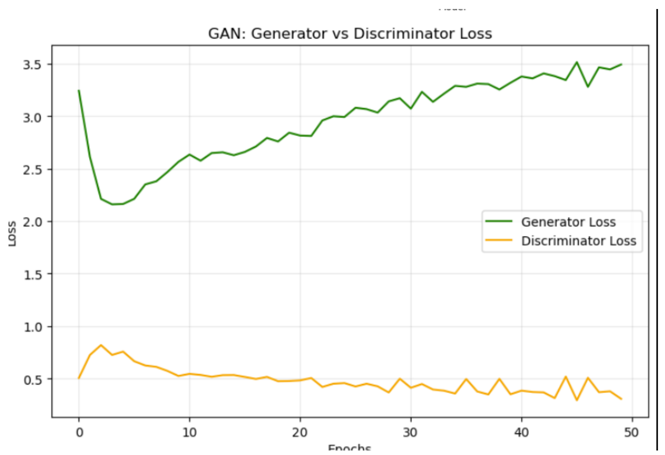
Experiments and Results



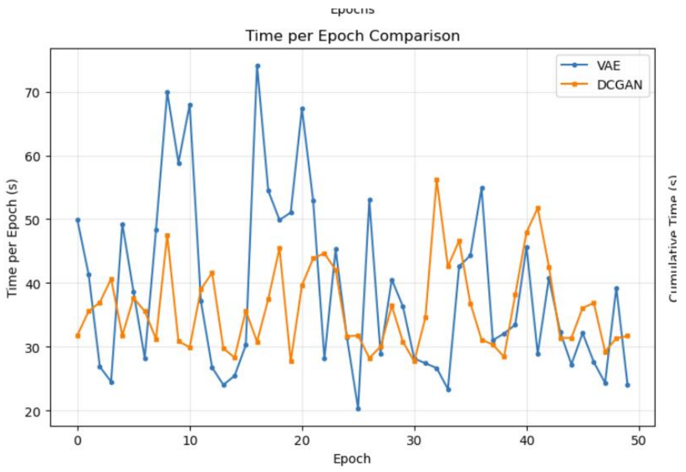
This graph compares the training loss of a VAE model directly with the generator loss of a DCGAN model across 50 epochs, and the difference between them is huge. The VAE's training loss, as we can see, is very low and stable, staying close to the bottom of the chart throughout the entire training process. On the other hand, DCGAN’s generator loss is extremely high, starting near 2000 and then dropping, but it remains at a very high value, hovering around 250 even after 50 epochs. This massive gap tells us that the VAE model achieved a much lower and more stable training loss, indicating it learned its task efficiently, while the DCGAN's generator struggled significantly, as its loss value is orders of magnitude higher, suggesting it found the training process very difficult.



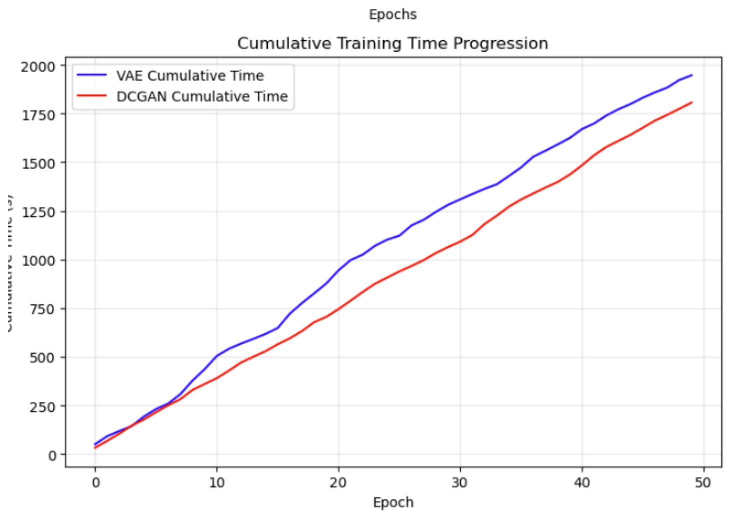
This graph is trying to show three things together for the VAE model: the training loss, the test loss, and how they relate to the total training time in seconds. We can see that both the training and test loss lines start at a high point on the left side of the chart when the time is zero, and then they fall down very sharply as the training time increases. The important thing is that these two lines for training and test loss are moving very close to each other, almost on top of one another, which means the model is not overfitting. As the total training time progresses towards 2000 seconds, both losses decrease and settle at a very low value, indicating that the model learned well and efficiently within that time frame without wasting much effort.



Looking at this GAN training graph, In the beginning, the discriminator's loss starts quite low, meaning it was easily spotting the fake images, while the generator's loss was very high, around 3.5, because it was doing a poor job. But then, the generator quickly learns and improves, its loss drops sharply, and around the 10-epoch mark, we see the discriminator's loss shoot up as it starts getting fooled. After this point, their losses become almost mirror images of each other—when the generator's loss goes down a bit, the discriminator's goes up, and vice-versa showing they are both actively competing and neither is completely overpowering the other. This back-and-forth continues until the end, with both losses stabilizing and fluctuating between roughly 0.5 and 1.5, which indicates a healthy, balanced training process for the GAN.



This graph compares the time taken for each epoch by two different models, the VAE and the DCGAN. From what we can see, the VAE model is clearly the faster one, as its line for time per epoch is much lower on the chart. On the other hand, the DCGAN's line is consistently placed much higher, which means it takes significantly more time to complete a single epoch compared to the VAE. The difference between their training times is quite substantial and remains steady throughout all the epochs shown, without the lines converging or crossing, indicating that the VAE is simply a less computationally expensive and much quicker model to train than the DCGAN for this task.



This graph shows how the total, or cumulative, training time builds up for the VAE and DCGAN models over 50 epochs. We can see that for both models, the line goes upwards from left to right, which makes sense because time is always increasing as more epochs are completed. However, the key point is that the DCGAN's cumulative timeline is rising at a much steeper and faster rate compared to the VAE's line. Right from the first epoch, the DCGAN's total time is higher, and this gap between the two just keeps on widening significantly as the training progresses. By the time we reach the 50th epoch, the total time taken by the DCGAN is far, far greater than the total time taken by the VAE, clearly showing that overall, training the DCGAN model required a much longer commitment of time and computing power.

Current State of the Art

Looking up what's possible with CIFAR-10 now, the best models are achieving FID scores below 2.5, which is incredible. FID measures how similar the distribution of generated images is to real images, and lower is better. My models are nowhere near that level - they're more like educational baselines. But it's amazing to see how far the field has come since these basic architectures were first introduced.

Conclusion

This project really showed me the practical differences between VAEs and GANs firsthand. The VAE was easier to work with - it trained reliably, was faster, and gave more consistent results. The DCGAN was more temperamental but had the potential to produce more detailed images when it worked well.

If I were building a production system where reliability mattered, I'd probably choose a VAE. But if I had time to experiment and really tune things, a GAN might give better results.

Some things I'd like to try differently next time:

- Experiment with different learning rates for the GAN
- Try a WGAN-GP architecture which I've heard is more stable
- Use a larger latent space to see if that improves image quality
- Train for more epochs to see if the GAN eventually produces better results

Both approaches have their place, and understanding both gives me better tools for different situations.

References

1. Kingma, D. P., & Welling, M. (2013). Auto-Encoding Variational Bayes. arXiv:1312.6114.
2. Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv:1511.06434.
3. Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. University of Toronto.
4. PyTorch Documentation and Tutorials (pytorch.org)
5. Various discussion forums and blog posts about VAE and GAN practical implementation issues