



# PROJECT REPORT

**DOMAIN: Machine Learning, Deep Learning, Natural Language  
Processing, Computer Vision**

**By Harshitha P**

# **Assignment -1**

## **Comparison of CNN Architectures on Different Datasets**

### **Project Description and objective:**

- This project compares the performance of different CNN architectures on various datasets. Specifically, we will evaluate LeNet-5, Alex Net, Google Net, VGG Net, ResNet, XceptionNET, and Senet on MNIST, FMNIST, and CIFAR-10 datasets. The comparison will be based on metrics such as loss curves, accuracy, precision, recall, and F1-score.

### **Dataset understanding**

- The datasets used in this project are:
- - MNIST: Handwritten digits dataset consisting of 60,000 training images and 10,000 testing images. Each image is 28x28 pixels in grayscale.
- - FMNIST: Fashion MNIST dataset consisting of 60,000 training images and 10,000 testing images of fashion products. Each image is 28x28 pixels in grayscale.
- - CIFAR-10: Dataset consisting of 60,000 32x32 color images in 10 classes, with 50,000 training images and 10,000 testing images.

### **Data preprocessing steps:**

#### **1.Normalization**

- `astype('float32')`: This converts the data type of the images from uint8 (default for images loaded in CIFAR-10) to float32. This is done to prepare the images for floating-point calculations, which are more efficient and accurate during model training.
- `/ 255`: Image pixel values in CIFAR-10 range from 0 to 255 (because they are in 8-bit format). Dividing by 255 scales the pixel values to the range [0, 1] to ensure that all input values are within a similar scale, improving the stability and performance of the neural network during training such that pixel values in both `train_images` and `test_images` will be between 0 and 1 instead of the original range [0, 255].

#### **2.Convert Labels to Categorical (One Hot encoding)**

Convert the train\_labels and test\_labels to arrays where each label is now a 10-dimensional vector (one-hot encoded) corresponding to the class.

### **Model Training:**

- Models trained using Adam optimizer with default learning rate of 0.001 for training CNNs on large datasets.
- Categorical cross entropy as the loss function for multi class classification tasks to compare the predicted probability with true one hot encoded label to ensure model to learn correct classification.
- Trained model saved as cnnmodel\_dataset\_model.keras (e.g. lenet\_cifar10\_model.keras)

### **Model Evaluation metrics:**

- Test loss, test accuracy, precision, f1 score, recall saved to cnnmodel\_dataset\_results.json(e.g. lenet\_cifar10\_results.json) file to print the metrics and plot loss and accuracy curves
- Accuracy: The proportion of correct predictions out of the total predictions made.
- Precision: The proportion of true positive predictions out of all positive predictions made.
- Recall: The proportion of true positive predictions out of all actual positives.
- F1-score: The harmonic mean of precision and recall.
- Loss: The value of the loss function during training and testing.

### **Analysis and comparison of performance of the CNN models**

To compare the performance of LeNet-5, Alex Net, Google Net, VGG Net, ResNet, XceptionNET, and Senet on the MNIST, FMNIST, and CIFAR-10 datasets, we need to consider the following key aspects:

- a) **Model architecture:** Each network has unique characteristics in terms of depth, complexity, and design philosophy.
- b) **Performance metrics:** These networks are evaluated based on their classification accuracy, training time, and computational complexity on the respective datasets.
- c) **Dataset details:** The MNIST, FMNIST, and CIFAR-10 datasets differ in terms of image complexity, resolution, and the number of classes.

#### **a) Model Architecture**

## 1. LeNet-5

- **Architecture:** LeNet-5 is one of the earliest convolutional neural networks (CNNs) designed by Yann LeCun. It consists of 7 layers: two convolutional layers, two pooling layers, and fully connected layers.
- **Dataset performance:**
  - **MNIST:** LeNet-5 performs very well on MNIST with an accuracy of around 98-99%. It was specifically designed for digit classification.
  - **FMNIST:** It has achieved good performance (approximately 87%) but is less effective than more modern architectures due to the complexity of fashion images.
  - **CIFAR-10:** LeNet-5 is not suitable for CIFAR-10 as it has a simpler architecture and lacks the depth required to handle more complex data. Performance on CIFAR-10 is typically low (61%).

## 2. Alex Net

- **Architecture:** Alex Net is deeper and more complex than LeNet-5, consisting of 8 layers (5 convolutional and 3 fully connected layers). It introduced the use of ReLU activation functions and dropout to prevent overfitting.
- **Dataset performance:**
  - **MNIST:** Alex Net achieves high accuracy (98-99%) on MNIST but is overkill for such simple tasks.
  - **FMNIST:** Alex Net performs well (89%) but might be too complex for the fashion dataset.
  - **CIFAR-10:** Alex Net performs significantly better on CIFAR-10 (~67%) compared to LeNet, but still lags newer models like ResNet.

## 3. Google Net (Inception v1)

- **Architecture:** Google Net introduced the Inception module, which uses multiple filter sizes within a single layer. It has 22 layers and is designed to be computationally efficient while being deep.
- **Dataset performance:**

- **MNIST:** Google Net can achieve 99% accuracy on MNIST, but the architecture is more complex than necessary.
- **FMNIST:** Google Net performs well (85%) and is more effective than simpler models.
- **CIFAR-10:** Google Net is not effective on CIFAR-10, achieving around 69% accuracy.

#### 4. VGG Net

- **Architecture:** VGG Net is known for its simplicity, consisting of 16-19 layers, all of which use 3x3 convolutions and max-pooling layers. It is deep and computationally expensive.
- **Dataset performance:**
  - **MNIST:** VGGNet performs excellently (~99%) but is over-engineered for such simple datasets.
  - **FMNIST:** VGGNet performs well (93%) on FMNIST, thanks to its deeper architecture.
  - **CIFAR-10:** VGGNet achieves solid performance (83%) but is computationally more expensive than models like google Net or ResNet.

#### 5. ResNet (Residual Networks)

- **Architecture:** ResNet introduced residual connections, allowing very deep networks to be trained without vanishing gradients. It can be very deep (e.g., ResNet-50, ResNet-101).
- **Dataset performance:**
  - **MNIST:** ResNet can achieve near-perfect accuracy (~99%), though it is more complex than necessary.
  - **FMNIST:** ResNet performs well (90%) and is one of the better choices for more complex datasets like FMNIST.
  - **CIFAR-10:** ResNet on CIFAR-10 achieves less accuracy around 66% compared to other datasets

#### 6. XceptionNET

- **Architecture:** XceptionNET uses depth wise separable convolutions to improve model efficiency while maintaining high performance. It is based on the idea that Inception-like architectures can be improved by decoupling spatial and depth wise convolutions.
- **Dataset performance:**
  - **MNIST:** XceptionNET can perform with near-perfect accuracy (~98%) on MNIST.
  - **FMNIST:** XceptionNET performs well (~90%) and can capture the complex patterns in fashion images.
  - **CIFAR-10:** XceptionNET achieves excellent performance (~81%) on CIFAR-10 due to its efficient use of depth wise separable convolutions.

## 7. Senet (Squeeze-and-Excitation Networks)

- **Architecture:** Senet introduces a "squeeze-and-excitation" block that re-calibrates feature maps by modelling interdependencies between channels. This mechanism improves performance by emphasizing more informative features.
- **Dataset performance:**
  - **MNIST:** Senet achieves nearly perfect accuracy (~96%) but is more complex than required.
  - **FMNIST:** Senet performs excellently (~91%), as it is highly adaptive to different datasets.
  - **CIFAR-10:** Senet is one of the low performers on CIFAR-10, achieving up to 53% accuracy, making it one of the most ineffective models for this task.

### c) Performance metrics

- **On MNIST:**
  - LeNet-5 and Alex Net perform exceptionally well with nearly 99% accuracy, but simpler architectures like LeNet-5 are more suitable.
  - VGGNet, Google Net, ResNet, XceptionNET, and Senet all achieve near-perfect accuracy but are overkill for this simple dataset.
- **MNIST results:**

Model	Accuracy	Precision	Recall	F1-score
Lenet-5	0.98	0.98	0.98	0.98
Alex Net	0.99	0.99	0.99	0.99
Google Net	0.99	0.99	0.99	0.99
VGGNet	0.99	0.99	0.99	0.99
XCeptionNET	0.98	0.98	0.98	0.98
ResNet	0.98	0.99	0.98	0.98
Senet	0.96	0.96	0.96	0.96

- **On FMNIST:**

- VGGNet leads in terms of performance (~93% accuracy), followed by Senet (~91%).
- ResNet, XCeptionNET performs well by 90%
- LeNet-5,AlexNet also perform well (~87-89%).
- Google Net, while still competitive, falls behind with accuracy around 85%.

- **Fashion MNIST results:**

Model	Accuracy	Precision	Recall	F1-score
Lenet-5	0.87	0.87	0.87	0.86
Alex Net	0.89	0.89	0.89	0.89
Google Net	0.85	0.87	0.85	0.85
VGGNet	0.93	0.93	0.93	0.93

<b>Xception Net</b>	0.90	0.90	0.90	0.90
<b>ResNet</b>	0.90	0.90	0.90	0.90
<b>Senet</b>	0.91	0.92	0.91	0.91

- **On CIFAR-10:**

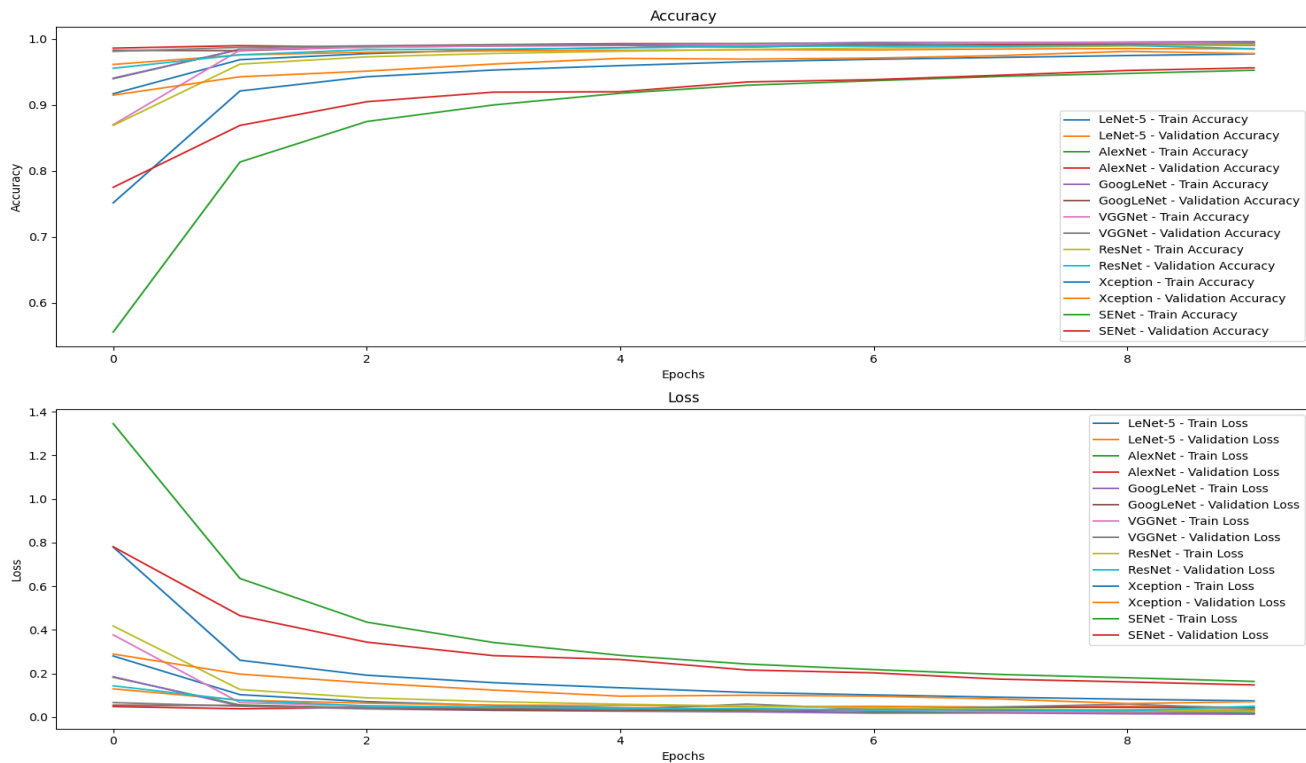
- VGGNet and XceptionNET perform best (~81-83%)
- ResNet,GoogLeNet and VGGNet perform low comparatively with accuracies around (~66-69%).
- Senet and LeNet-5 struggle on this more complex dataset, with accuracies in the 53-60% range.

- **CIFAR -10 results:**

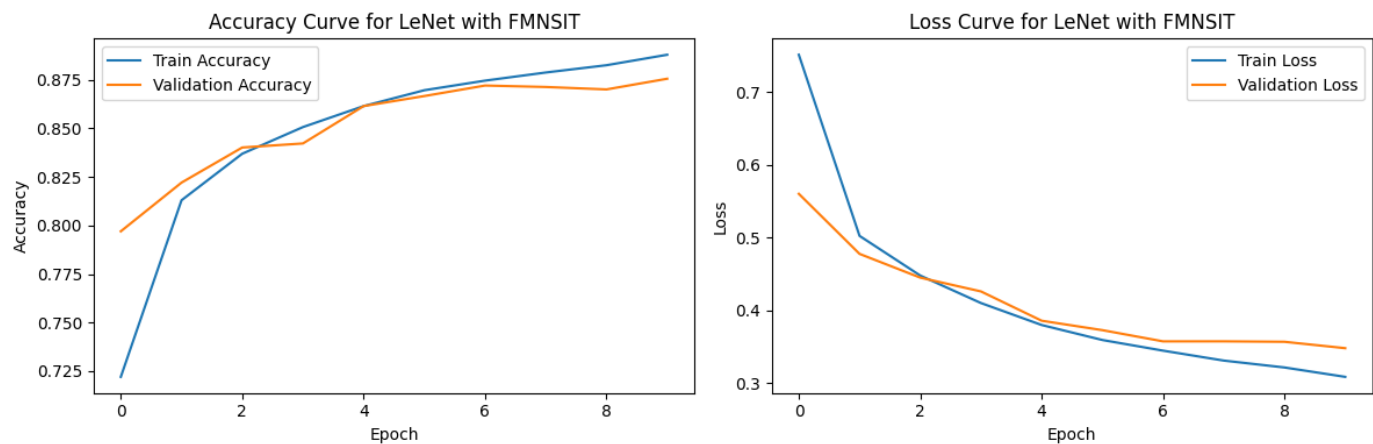
<b>Model</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
<b>Lenet-5</b>	0.61	0.62	0.61	0.61
<b>Alex Net</b>	0.67	0.73	0.67	0.67
<b>Google Net</b>	0.69	0.70	0.69	0.69
<b>VGGNet</b>	0.83	0.83	0.83	0.82
<b>XceptionNET</b>	0.81	0.82	0.81	0.81
<b>ResNet</b>	0.66	0.67	0.66	0.65
<b>Senet</b>	0.53	0.55	0.53	0.53

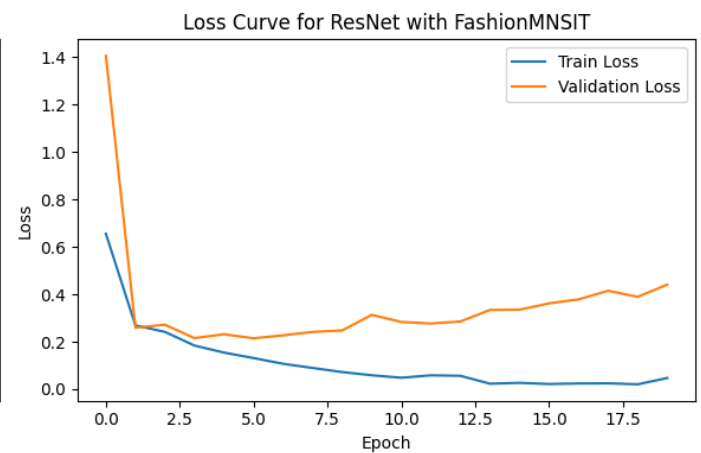
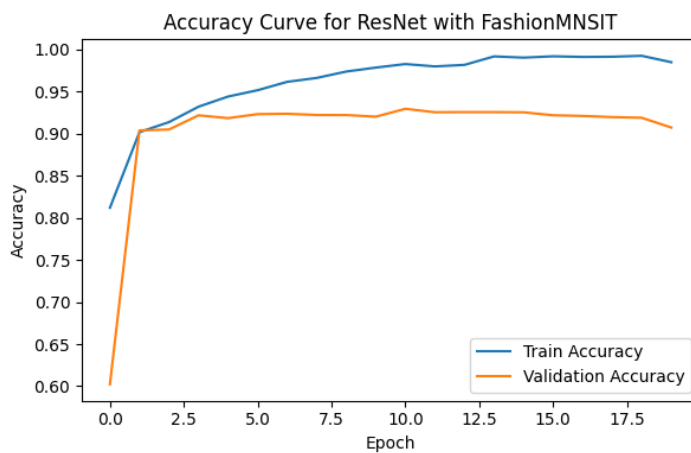
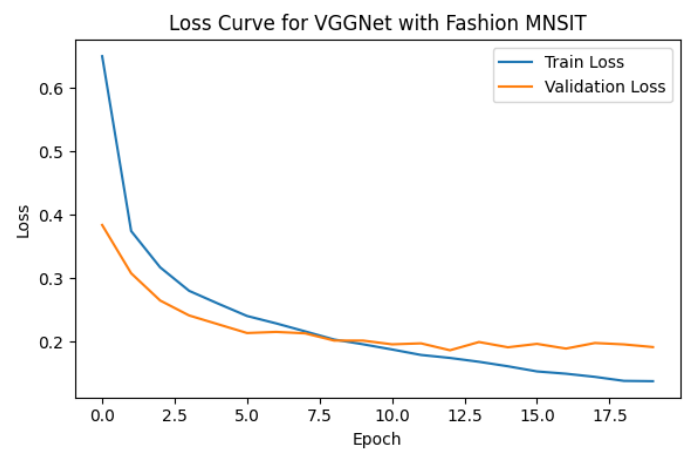
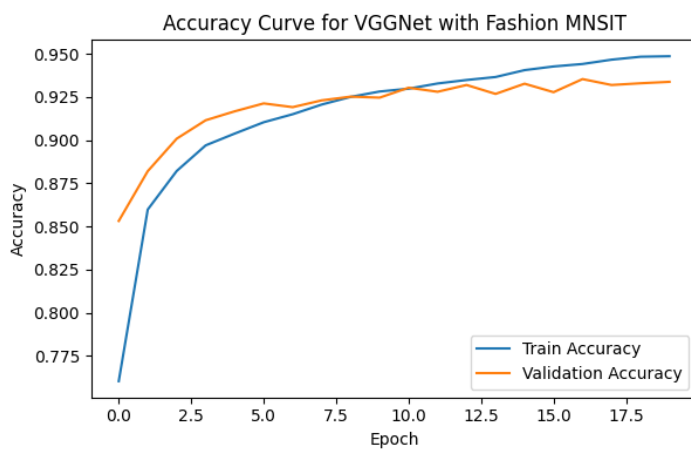
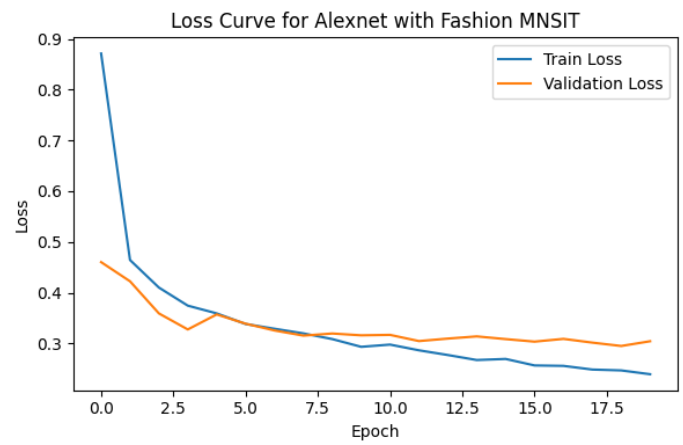
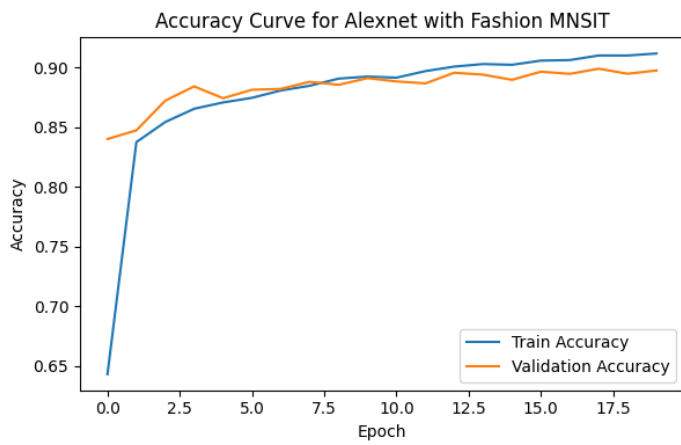


- **Loss curve and accuracy curve to analyse performance for all CNN models with MNIST dataset**

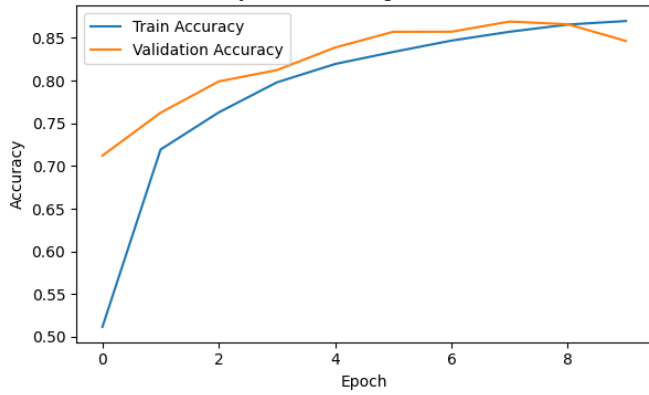


- **Loss and accuracy curve with Fashion MNSIT dataset**

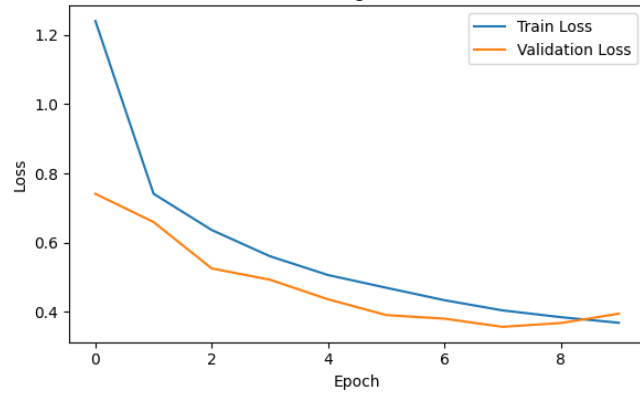




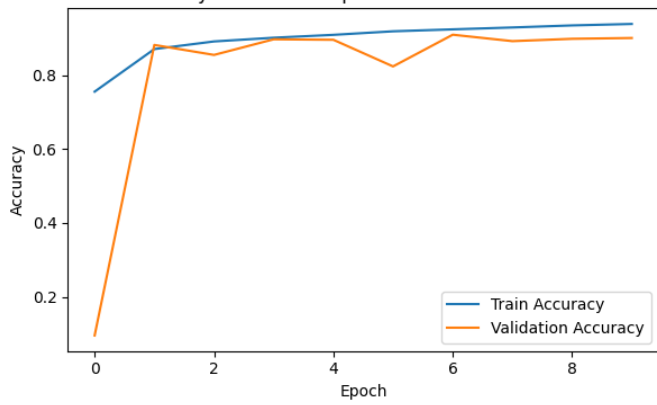
Accuracy Curve for GoogleNet with FMNSIT



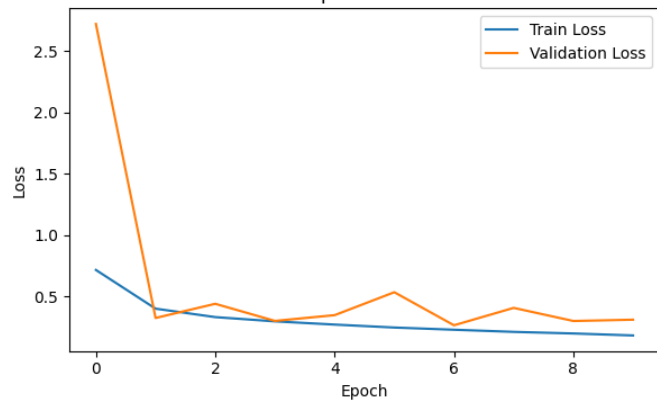
Loss Curve for GoogleNet with FMNSIT



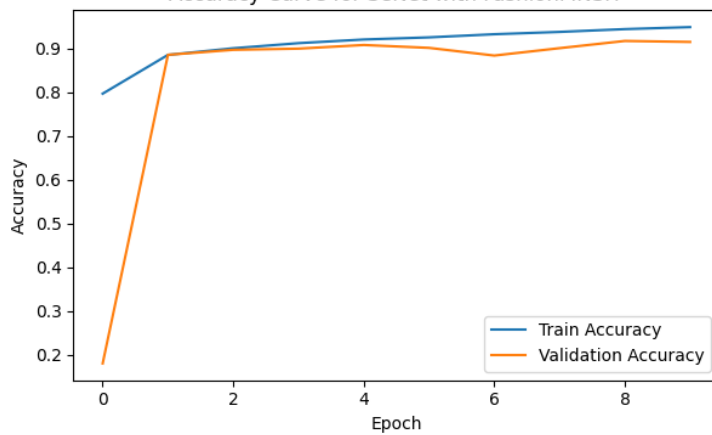
Accuracy Curve for XceptionNet with FashionMNSIT



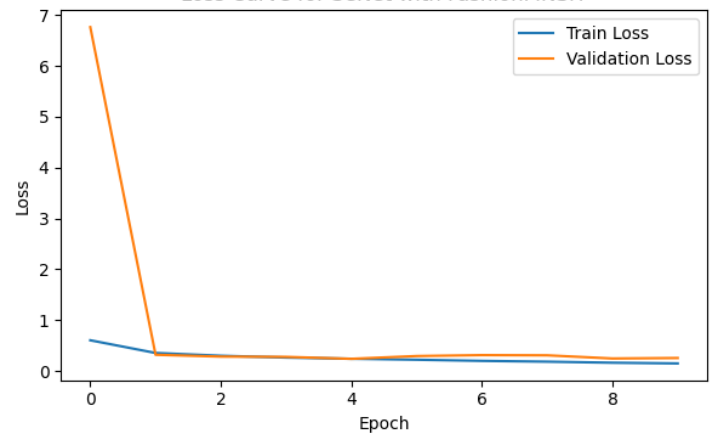
Loss Curve for XceptionNet with FashionMNSIT



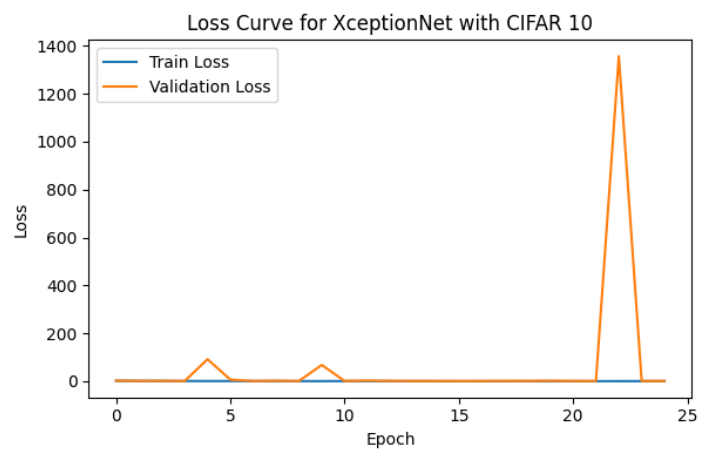
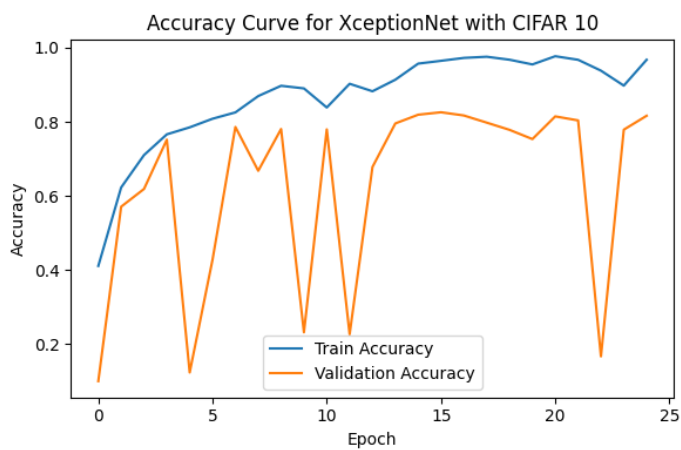
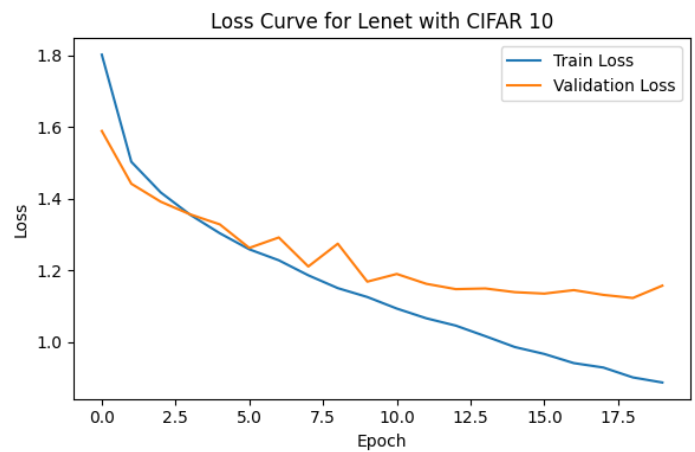
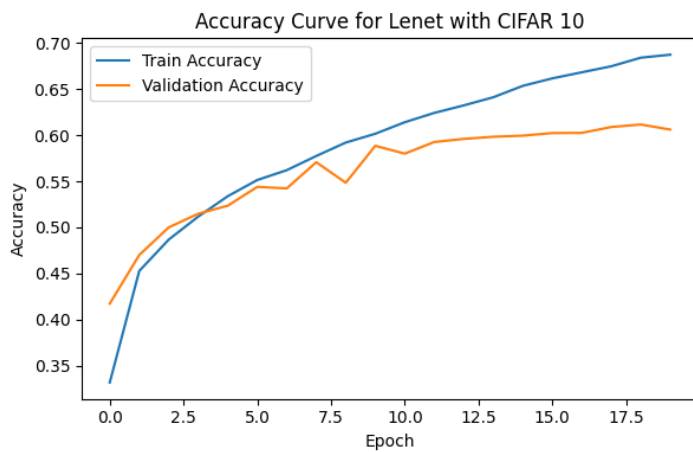
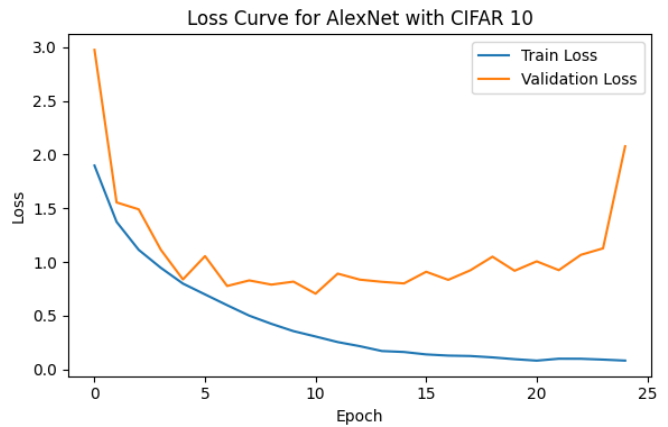
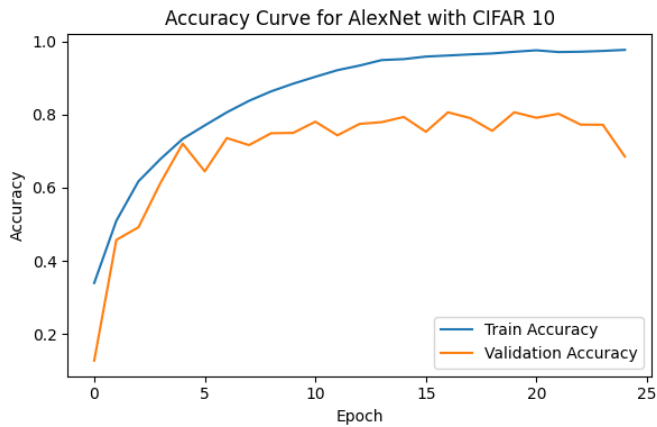
Accuracy Curve for SeNet with FashionMNSIT



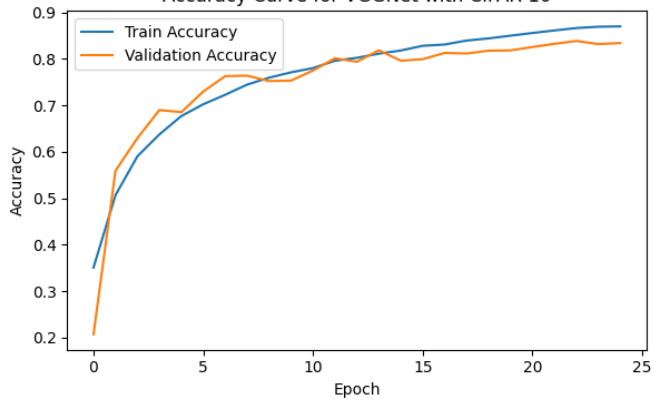
Loss Curve for SeNet with FashionMNSIT



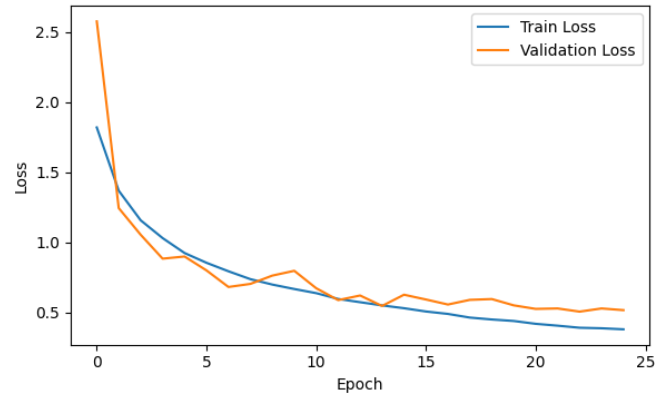
- **Loss and accuracy curve for all CNN models with CIFAR -10 dataset**



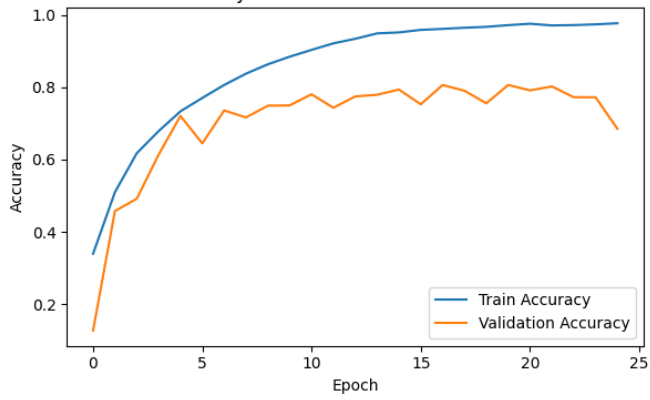
Accuracy Curve for VGGNet with CIFAR 10



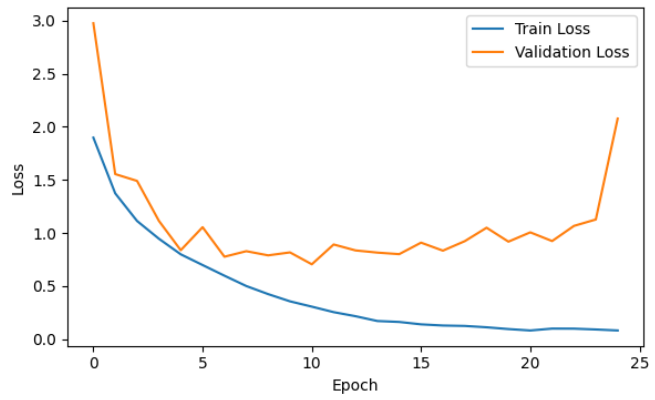
Loss Curve for VGGNet with CIFAR 10



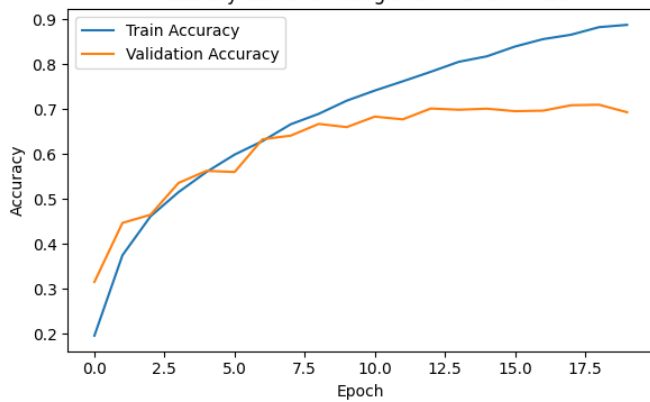
Accuracy Curve for AlexNet with CIFAR 10



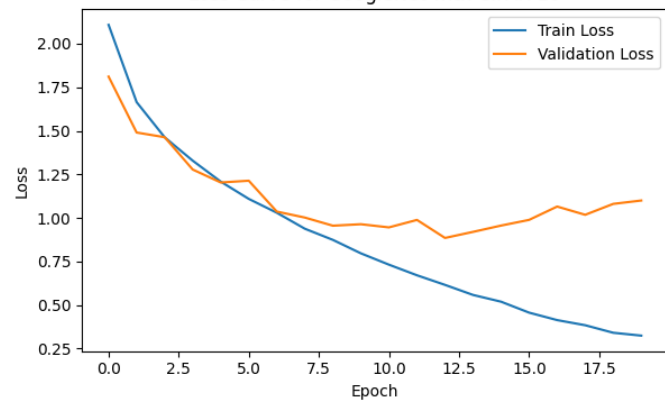
Loss Curve for AlexNet with CIFAR 10

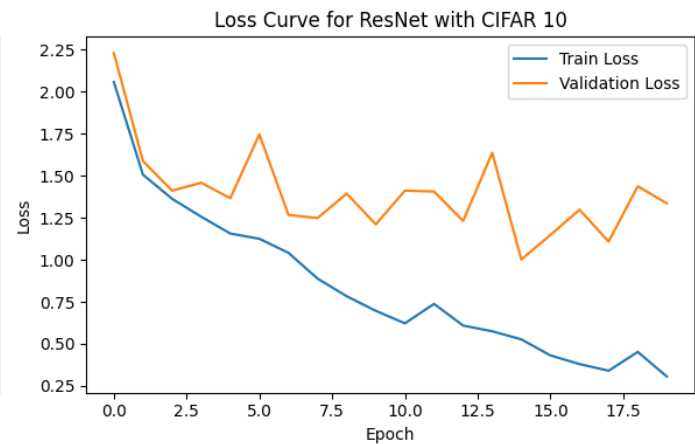
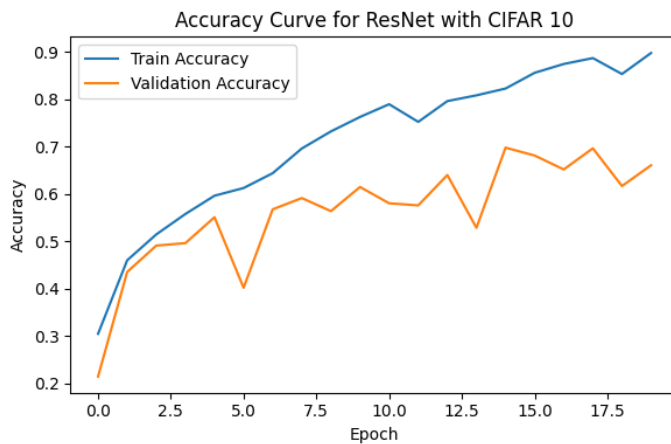
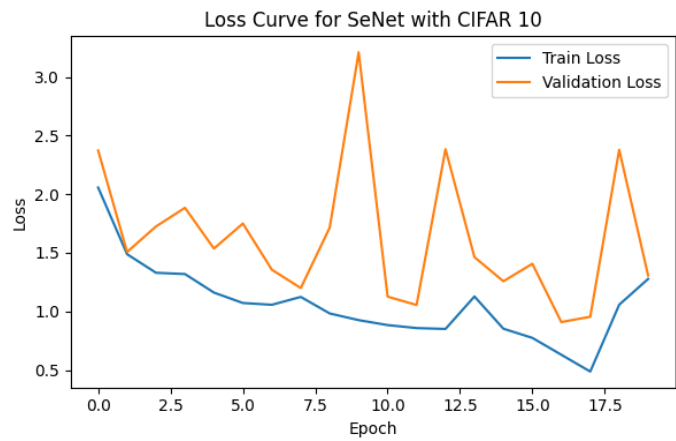
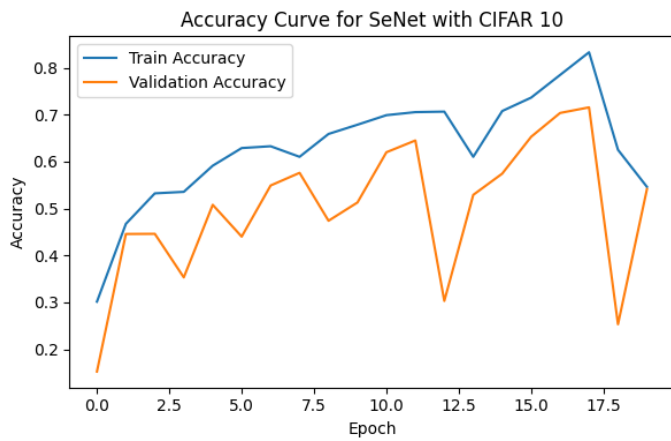


Accuracy Curve for GoogleNet with CIFAR 10



Loss Curve for GoogleNet with CIFAR 10





## Conclusion

- For simpler datasets like **MNIST**, lighter models like **LeNet-5** and **Alex Net** are sufficient, with near-perfect performance.
- For more complex datasets like **FMNIST** and **CIFAR-10**, deeper and more advanced architectures like **ResNet**, **XceptionNET** offer better performance. **VGGNet** stands out as the best performer overall due to its adaptive feature recalibration.

# Assignment -2

## Sequence-to-Sequence Modeling with Attention Mechanism

### Project Description

This project implements a sequence-to-sequence (Seq2Seq) model with attention using PyTorch. The model is designed to learn a task where an input sequence (a list of integers) is transformed into an output sequence (its reversed version). The core of the model consists of an encoder-decoder architecture with attention mechanisms. In addition to the model architecture, the code also handles data generation, training, evaluation, and visualization of the attention mechanism.

### 1. Data Generation and Preprocessing

- **generate\_data Function:** This function generates a synthetic dataset of sequences. Each sequence is composed of random integers, and the corresponding target is the reversed version of the source sequence. The default parameters generate 10,000 samples, each of length 10, with a vocabulary size of 10.
- **pad\_sequences Function:** This helper function ensures that all sequences in a batch have the same length by padding shorter sequences with a specified value (default 0).
- **prepare\_batch Function:** This function prepares data for batching, where it shuffles the data and pads sequences to ensure they are of equal length. It yields batches of source-target pairs, ready for processing by the model.

### 2. Model Architecture

- The Seq2Seq model consists of three key components: the **Encoder**, the **Attention Mechanism**, and the **Decoder**.
- **Encoder:** The encoder takes in a sequence of tokens, embeds them into a dense representation, and processes them through an LSTM (Long Short-Term Memory) network. The encoder returns the output sequence along with the final hidden and cell states of the LSTM.
- **Attention Mechanism:** The attention mechanism calculates a weighted sum of the encoder outputs, where the weights are based on how relevant each encoder output is to the current decoder state. The attention mechanism uses a simple feedforward network to compute these weights.

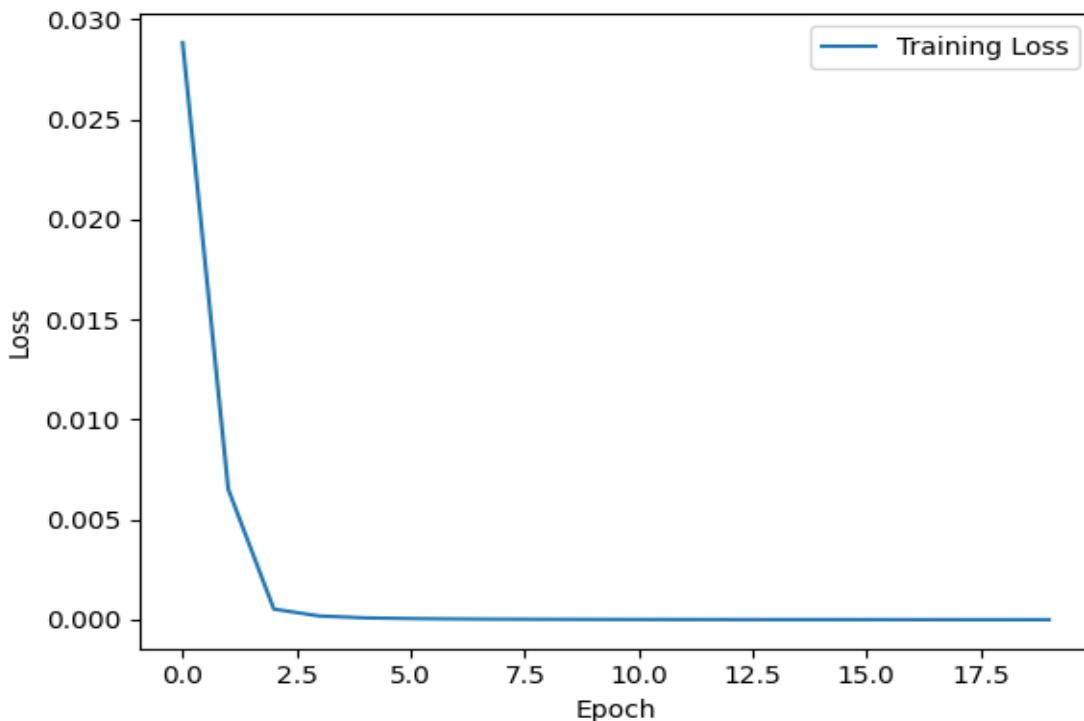
- **Decoder:** The decoder generates the output sequence based on the encoded input sequence and the attention context. It first takes an embedding of the previous token, computes attention weights for the encoder outputs, and then concatenates the attention context with the embedded token to produce the next token in the sequence.
- **Seq2Seq Model:** The Seq2Seq model integrates the encoder and decoder into a complete pipeline. During training, the model uses teacher forcing, where the true previous token is fed into the decoder at each time step.

### 3. Training

- The training process involves iterating over batches of source-target pairs and updating the model's parameters using backpropagation and gradient descent. The model is trained using the **CrossEntropyLoss** function, and the optimizer is Adam.

### 4. Evaluation

- **Accuracy:** Accuracy is computed by comparing the predicted output with the actual target sequence, ignoring padding tokens. Accuracy =1.0.



### 5. Analysis of the Effectiveness of the Attention Mechanism in Improving Seq2Seq Model Performance

The attention mechanism allows the decoder to dynamically focus on different parts of the input sequence at each time step, rather than relying on a single, fixed context vector. The key points of the attention mechanism in the provided code are as follows:



- **Dynamic Focus:** At each time step of decoding, the model computes a set of attention weights that determine how much attention should be given to each token in the input sequence. These attention weights are used to create a weighted sum of the encoder's outputs, which serves as a dynamic context vector for generating the next token in the output sequence.

In the code, the attention weights are computed in the Attention class, which takes the encoder's outputs and the decoder's hidden state to calculate the attention scores.

- **Contextual Representation:** The decoder then uses this attention-derived context vector, which is a weighted combination of the encoder's outputs, at each decoding step. This allows the decoder to focus on different parts of the input sequence that are most relevant for generating each token in the output sequence.

The context vector is computed using the attention weights, and the decoder then concatenates this context vector with the embedding of the current token before passing it through the LSTM.

### . 3. Key Benefits of the Attention Mechanism

The attention mechanism has several key advantages that contribute to the improvement of the Seq2Seq model's performance:

- **Handling Long Sequences:** In the case of reversing the input sequence (as in the synthetic dataset), attention allows the decoder to dynamically select parts of the source sequence that are most relevant for generating each token in the output sequence.
- **Improved Handling of Complex Dependencies:** For tasks where the output at each time step depends on multiple tokens from the input sequence, attention enables the model to learn and focus on these dependencies. This dynamic alignment between input and output sequences helps the model better handle complex patterns and relationships.
- **Interpretability:** The attention weights provide an interpretable insight into the model's behaviour. By visualizing the attention matrix, we can observe which parts of the input sequence the decoder focuses on at each decoding step. This can help in understanding the model's decision-making process.

**Conclusion:** This project demonstrates a complete pipeline for training a Seq2Seq model with attention. The attention mechanism allows the model to focus on different parts of the input sequence at each step of the output sequence generation. The data generation and batching functions provide synthetic data for training, while the visualization tools allow users to inspect the attention weights and the model's predictions. The model is evaluated on accuracy, and training progress is visualized through loss curves.

# Assignment-3

## Multifunctional NLP and Image Generation Tool using Hugging Face Models

### Project Description:

This project is a **Streamlit-based web application** that provides an interactive platform for performing a range of natural language processing (NLP) and image generation tasks using pre-trained models from the **Hugging Face Transformers** and **Diffusers** libraries. The application allows users to engage with advanced machine learning models for tasks such as text summarization, sentiment analysis, question answering, chatbot interaction, story generation, next word prediction, and image generation. All tasks leverage popular NLP models and the **Stable Diffusion** model for generating images.

### **Key Features**

1. **Text Summarization:** Users can input long text, and the application will summarize it into a shorter, more digestible form using the **BART** model (Facebook's BART-large-CNN).
2. **Next Word Prediction:** Given a prompt, the application predicts the next word in the sequence using a **DialoGPT** or **GPT-2** model. This feature helps users extend or complete sentences.
3. **Story Prediction:** Users can generate a short story from a given prompt. The model (based on **GPT-2**) will generate a creative, coherent narrative based on the user's input.
4. **Chatbot Interaction:** The application enables users to chat with a machine learning-based chatbot powered by **GPT2**, which responds contextually to user input, maintaining conversation history.
5. **Sentiment Analysis:** Users can input text for sentiment analysis. The application uses a pre-trained sentiment model to classify the text as positive, negative, or neutral, along with a confidence score.
6. **Question Answering:** Given a context and a question, the model (based on **DistilBERT**) will find and return the most relevant answer from the provided text.
7. **Image Generation:** Using the **Stable Diffusion** model, users can generate images from text prompts. The model creates realistic images based on descriptive input, ideal for visualizing concepts, objects, or scenes described in text.

## Technology Stack

- **Streamlit:** For building the interactive user interface, which allows users to easily input data and see results in real time.
- **Hugging Face Transformers:** For providing access to a variety of NLP models such as **BART** (for summarization), **DistilBERT** (for question answering), **GPT-2** and **DialogPT** (for text generation and chatbot), and **BART-large-CNN**.
- **Diffusers:** For handling image generation with the **Stable Diffusion** model.
- **PyTorch:** Backend framework for running the models efficiently, especially for GPU or CPU computations.

## How It Works

### 1. Model Loading:

- Pre-trained models are loaded using the `load_*` helper functions. These models are cached for performance optimization, ensuring they are loaded only once per session.
- Tasks like text summarization, sentiment analysis, story generation, etc., are linked to their respective pipelines that provide simplified access to complex models.

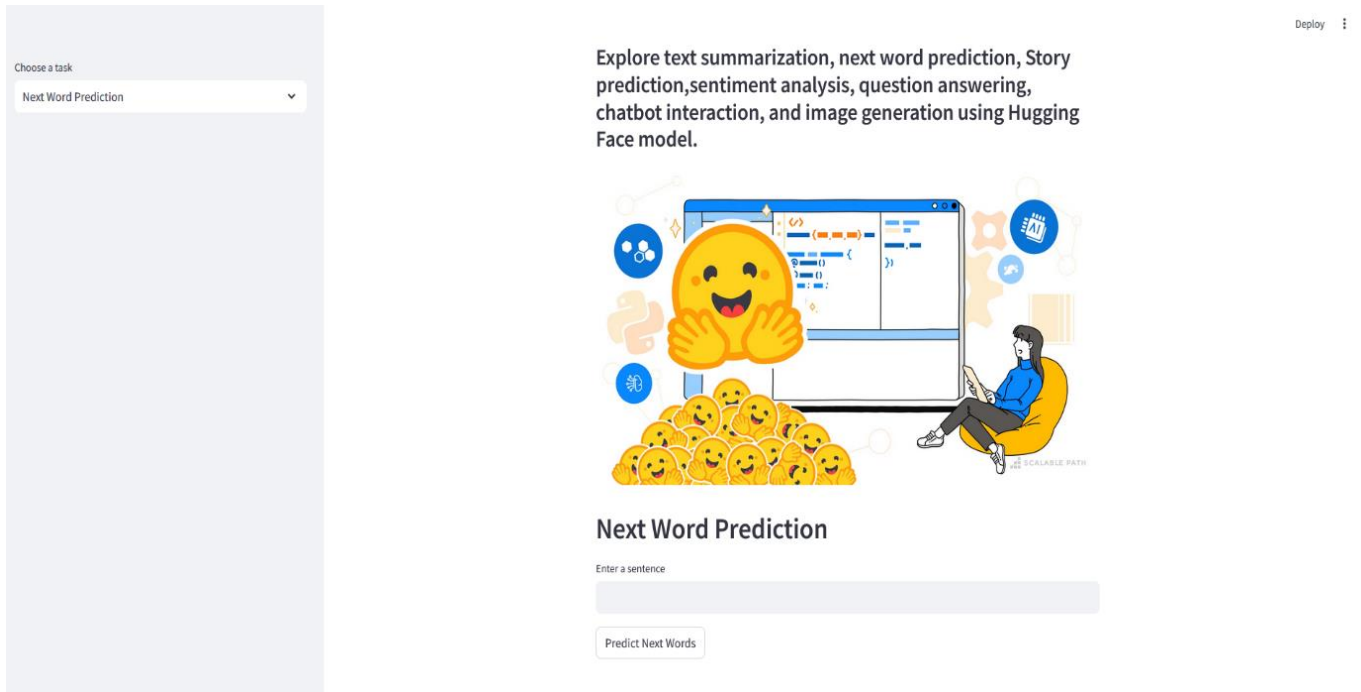
### 2. Task-specific Pipelines:

- For each selected task, the app activates the corresponding pipeline function that processes the user input and returns a result:
  - **Summarization:** facebook/bart-large-cnn
  - **Sentiment Analysis:** Uses the default sentiment pipeline from Hugging Face.
  - **Question Answering:** distilbert-base-cased-distilled-squad
  - **Text Generation:** gpt2 (for story prediction and next word generation).
  - **Image Generation:** Uses **Stable Diffusion** with specific configurations to generate images based on text input.

### 3. Interactive Web Interface:

- **Sidebar:** Users can select the task they want to perform from the sidebar menu.
- **Main Area:** Displays the relevant input fields and output areas (such as text boxes, buttons, or image displays).

- **Real-time Updates:** The interface shows a spinner while the model processes the request, followed by a result or warning messages as needed (e.g., if input fields are left empty).



#### 4. Caching:

- The application uses `st.cache_resource` and `st.cache_data` to store model states and processed data, ensuring the models are not repeatedly loaded and that results from previous tasks are retained to speed up future queries.

### Tasks Walkthrough

#### 1. Text Summarization:

- Users input a paragraph or more, and the system summarizes the content, keeping the main ideas intact in a concise manner.

#### 2. Next Word Prediction:

- Given an incomplete sentence or prompt, the system predicts the next word based on contextual knowledge, simulating the completion of user input.

#### 3. Story Prediction:

- A creative narrative is generated based on a user-provided prompt. This feature is particularly useful for writers or those looking for inspiration.

#### 4. **Chatbot:**

- A conversational AI model responds to users' inputs, carrying on a conversation. It can handle multiple exchanges, maintaining context between user inputs.

#### 5. **Sentiment Analysis:**

- Text entered by the user is analyzed for sentiment, providing both the sentiment label (positive, negative, or neutral) and the confidence score.

#### 6. **Question Answering:**

- By providing a context (such as an article) and a question, users can extract an answer from the provided text, making this tool useful for educational or research purposes.

#### 7. **Image Generation:**

- Users describe a scene or concept in words, and the model generates an image that visually represents the description, offering a unique way to create images from text.

### **Installation and Setup**

#### 1. **Dependencies:** To run this project locally, you need to install the following libraries:

- Streamlit
- transformers
- diffusers
- torch
- pillow

Install them via pip:

```
pip install Streamlit transformers diffusers accelerate pillow
```

#### 2. **Running the App:** Once the dependencies are installed, save the script and run the app with:

#### 3. Streamlit run app.py


This will open the app in your default web browser, where you can interact with the tool.

- Sentiment Analysis example

Choose a task  
Sentiment Analysis

### Multifunctional NLP & Image Generation App

Explore summarization, text prediction, story prediction, sentiment analysis, question answering, chatbot interaction, and image generation using Hugging Face models.



### Sentiment Analysis

Enter text(s) for sentiment analysis (separate multiple texts with newline)

"I absolutely love the new feature update! It's made my work so much easier."

Analyze Sentiment

### Sentiment Analysis Results

Text: "I absolutely love the new feature update! It's made my work so much easier."

Sentiment: POSITIVE, Score: 0.998

Select a rating based on your experience

0 1

You rated: 1

Please share your experience with us. Thank you!

- Image generation example data


Choose a task  
Image Generation

### Image Generation

Enter a description for the image

"Generate an image of a futuristic city with flying cars and glowing skyscrapers at sunset."

Generate Image



Generated image

Deploy

- Chatbot example

### Multifunctional NLP & Image Generation App

Explore summarization, text prediction, Story prediction, sentiment analysis, question answering, chatbot interaction, and image generation using Hugging Face model.



#### Chatbot

You:

Hi, How are you?

Chat

Bot:

I'm good, how are you?

Select a rating based on your experience

0 5

You rated: 3

Please share your experience with us. Thank you!

- Text summarization example

### Multifunctional NLP & Image Generation App

Explore summarization, text prediction, Story prediction, sentiment analysis, question answering, chatbot interaction, and image generation using Hugging Face model.



#### Text Summarization

Enter text to summarize:

"Artificial intelligence (AI) is a branch of computer science that aims to create machines that can perform tasks that would normally require human intelligence. Some examples of these tasks include learning, reasoning, problem-solving, perception, and language understanding. Over the past decade, AI has made significant strides in fields such as natural language processing, computer vision, and autonomous vehicles."

Summarize

Summary

Artificial intelligence (AI) aims to create machines that can perform tasks that would normally require human intelligence. Over the past decade, AI has made significant strides in fields such as natural language processing, computer vision, and autonomous vehicles.

Select a rating based on your experience

0 5

You rated: 3

## Use Cases

- **Content Creators:** Writers, bloggers, and journalists can use text summarization, sentiment analysis, and story generation to speed up their content creation.
- **Research:** Academics can leverage the question-answering and summarization features to analyze and extract information from research papers or articles.

- **Education:** Teachers and students can use the chatbot and sentiment analysis for educational exercises or discussions.
- **Designers:** Image generation can assist visual artists or graphic designers in conceptualizing designs based on text descriptions.

## Conclusion

This project provides a versatile tool that combines state-of-the-art NLP models and image generation capabilities, offering an easy-to-use interface to interact with these models. By leveraging the power of **Hugging Face** and **Stable Diffusion**, this tool democratizes access to advanced machine learning models, making them available to a broad range of users for personal, educational, and professional use cases.