IMDB Sentiment Classification: RNN vs ANN

Objective

To implement and compare two different neural network architectures (RNN and ANN) for binary sentiment classification on the IMDB movie reviews dataset using TensorFlow and Keras.

Dataset and Preprocessing

- **Dataset**: IMDB reviews dataset (25,000 training and 25,000 testing samples).
- **Preprocessing**:
  - Used only the top 10,000 most frequent words.
  - Applied padding to ensure uniform input length.
  - Used maxlen = 467 (maximum review length) for padding.

```
vocab_size = 10_000
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words = vocab_size)
x_train_padded = pad_sequences(x_train, maxlen = maxlen, padding = "post", truncating = "post")
x_test_padded = pad_sequences(x_test, maxlen = maxlen, padding = "post", truncating = "post")
```

Model 1: Recurrent Neural Network (RNN)

Architecture:

- Embedding Layer
- LSTM Layer
- Dense Layer
- Output Layer

Hyperparameter Tuning with Keras Tuner

- Tuned: embedding_dim, lstm_units, dense_units, learning_rate

```python
def model_builder(hp):
  vocab_size = 10_000
  input_length = maxlen

  # Hyper parameters to tune
  embedding_dim = hp.Choice("embedding_dim", [64, 128, 256])
  lstm_units = hp.Int("lstm_units", min_value = 32, max_value = 128, step = 32)
  dense_units = hp.Int("dense_units", min_value = 32, max_value = 256, step = 32)
  learning_rate = hp.Choice("learning_rate", [1e-3, 1e-4, 2e-4, 5e-4])

  inputs = layers.Input(shape = (input_length, ))
  x = layers.Embedding(input_dim = vocab_size, output_dim = embedding_dim)(inputs)
  x = layers.LSTM(units = lstm_units)(x)
  x = layers.Dense(units = dense_units, activation = "relu")(x)
  outputs = layers.Dense(units = 1, activation = "sigmoid")(x)

  model = Model(inputs, outputs)

  model.compile(optimizer = optimizers.Adam(learning_rate = learning_rate),
                loss = "binary_crossentropy",
                metrics = ["accuracy"])

  return model
```

Training:

- Used early stopping
- Trained on 80% of training data, validated on 20%

Performance:

- Validation Accuracy: ~87%
- Observed some overfitting (validation loss increased mid-training)

Model 2: Artificial Neural Network (ANN)

Architecture:

- Embedding Layer
- GlobalAveragePooling1D

- Dense Layer
- Output Layer

```python
def create_ann_model(vocab_size = 10_000, input_length = maxlen):

    model = Sequential([
        Input(shape = (input_length,)),
        Embedding(input_dim = vocab_size, output_dim = 128),
        GlobalAveragePooling1D(),
        Dense(64, activation = "relu"),
        Dropout(0.3),
        Dense(1, activation = "sigmoid")
    ])

    model.compile(
        loss = "binary_crossentropy",
        optimizer = optimizers.Adam(learning_rate = 0.001),
        metrics = ["accuracy"]
    )
    return model
```

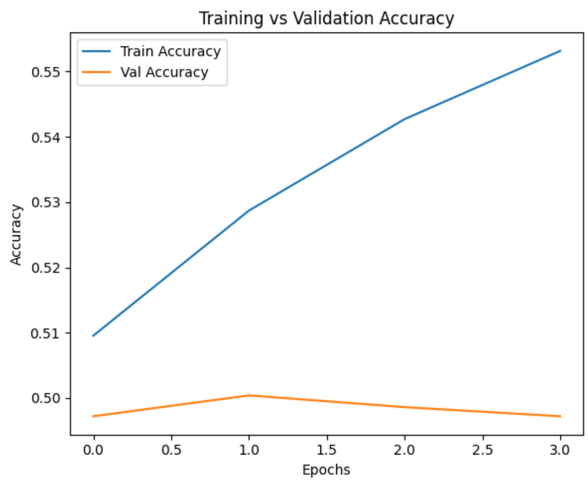
Performance:

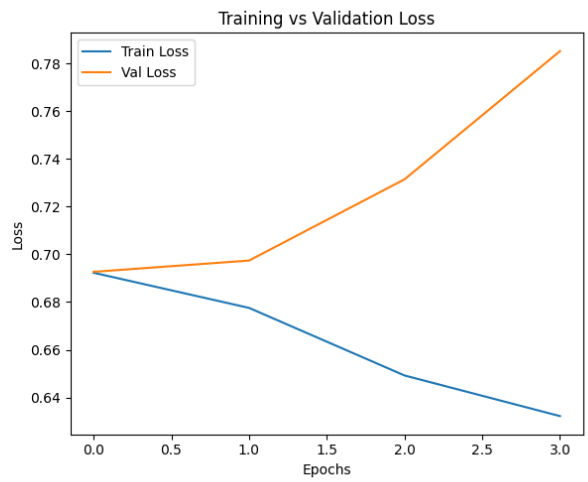
- Validation Accuracy: ~89%
- Validation loss decreased smoothly — indicating better generalization

---

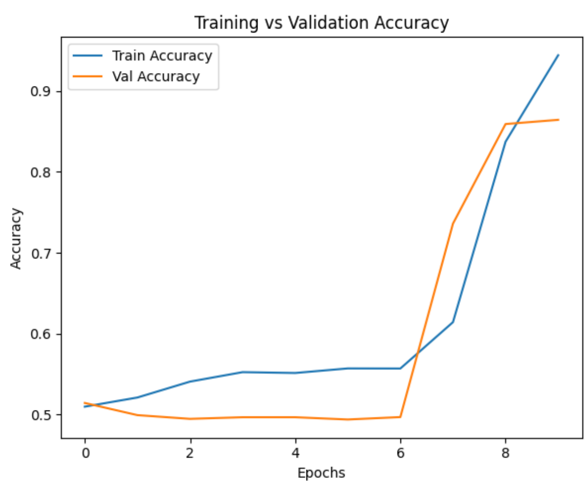
Metrics Visualization:

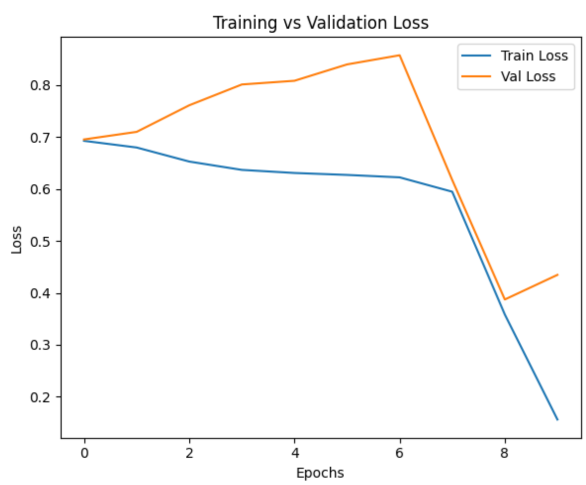
Base Model’s Visualization



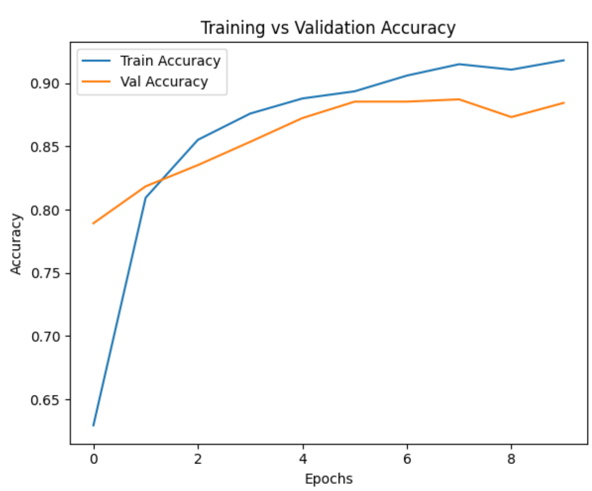


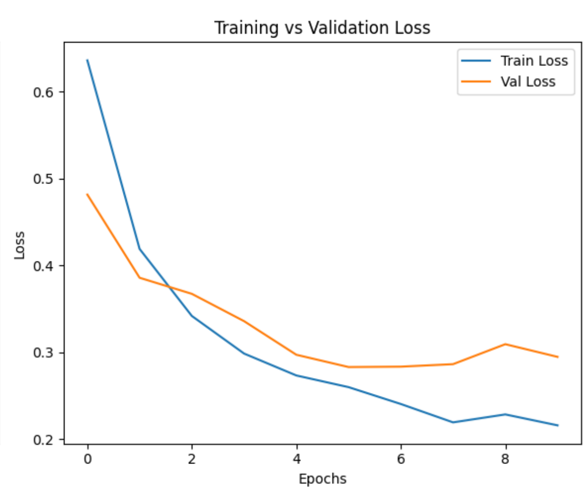
Fined-tuned model Visualization





ANN model Visualization





Model Comparison

Metric	RNN Model	ANN Model
Validation Accuracy	~87%	~89%
Training Speed	Slower	Faster
Generalization	Moderate	Strong

Analysis & Insights

- Despite being simpler, the ANN model outperformed the RNN.
- The IMDB dataset is relatively small and binary-labeled — sequence information might not be essential.
- RNNs are powerful but sensitive to overfitting and hyperparameter tuning.
- For basic sentiment classification tasks, FFNNs can be surprisingly strong baselines.

Conclusion

- RNNs may not always be the best choice for all NLP tasks, especially when the dataset is small and task is simple.
- ANN provided better generalization and stability with faster training.
- Hyperparameter tuning, early stopping, and model simplicity play a significant role in practical performance.

This exercise provided practical insight into model selection, tuning, and the importance of testing assumptions even in deep learning.