Email Classification using RNN

This project implements a spam and ham email classification system using a Simple RNN model. The solution includes data preprocessing, model building, training with callbacks, and deployment using Streamlit Cloud.

Overview

The goal of this project is to classify emails as Ham (Not Spam) or Spam using a Recurrent Neural Network (RNN). The model is built using TensorFlow and Keras and deployed via a user-friendly web interface using Streamlit.

Project Workflow

Step 1: Data Import

- Load email data from an Excel file (spam.xlsx) using pandas.
- The dataset contains email texts and corresponding labels (Spam or Ham).

Step 2: Text Preprocessing I

The following preprocessing steps are applied to clean and prepare raw email text data:

1. Remove numbers and punctuation
2. Convert all text to lowercase
3. Tokenize the text into individual words
4. Remove stop words (e.g., "the", "is", "in")
5. Apply stemming using the Porter Stemmer
6. Reconstruct the processed words back into a cleaned sentence
7. Label Encoding
    - Ham → 0
    - Spam → 1

Step 3: Text Preprocessing II (Tokenization & Padding)

1. Tokenize the cleaned text using Tokenizer from Keras.
2. Set:
    - vocab_size = 10000
    - output_dim = 128
    - max_len = 500 (maximum length of each email)
3. Apply padding to ensure all input sequences are of equal length.
4. Save the tokenizer object for deployment to process real-time inputs.

Step 4: Train-Test Split

- Split the processed dataset into training and testing sets using train_test_split from Scikit-learn.

Step 5: Model Building – Simple RNN

Build a Sequential model using Keras with the following layers:

1. Embedding Layer
   Converts numeric tokens into dense vector embeddings.
2. SimpleRNN Layer
   Captures temporal dependencies in the sequence data.
3. Dense Output Layer
    - 1 neuron with sigmoid activation to output probabilities.

Step 6: Model Compilation & Training

- Loss Function: Binary Crossentropy
- Optimizer: Adam
- Metrics: Accuracy

Additional Configuration:

- TensorBoard Callback
  Enables training visualization.

- EarlyStopping Callback
  Stops training when the validation loss stops improving to avoid overfitting.
- Model Saving
  Save the trained model to .h5 format.

Step 7: Model Inference (Prediction Function)

1. Load the saved model and tokenizer.
2. Preprocess user-provided input using the same steps from Text Preprocessing I & II.
3. Predict using the model:
    - Model outputs a probability between 0 and 1.
    - Apply threshold:
        - $p < 0.5 \rightarrow$ Ham
        - $p \geq 0.5 \rightarrow$ Spam

Step 8: Streamlit Application

Streamlit UI Features:

- Text input for email content
- Button to classify email
- Real-time classification result display (Ham/Spam)

Integration Steps:

1. Combine all preprocessing and prediction functions into main.py.
2. Create a user interface with Streamlit widgets.
3. Ensure requirements.txt includes all necessary packages (e.g., TensorFlow, Streamlit, NLTK, pandas, scikit-learn).

Step 9: Deployment on Streamlit Cloud

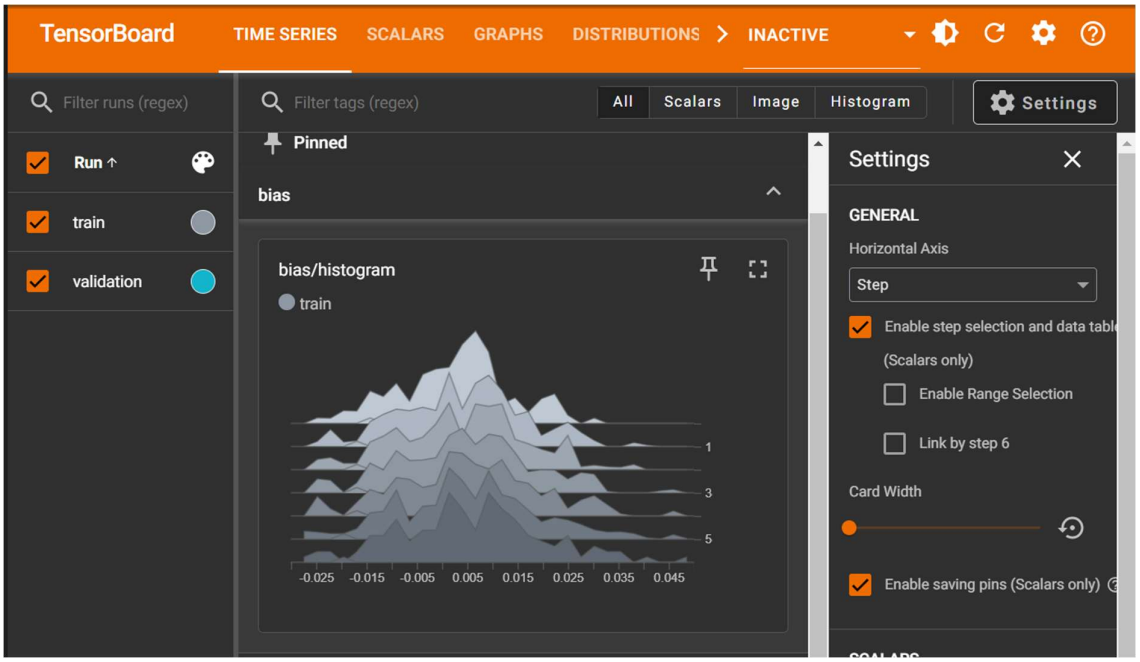
1. Push all project files to a GitHub repository.
2. Go to Streamlit Cloud and link your GitHub repository.
3. Configure the app to use:

- main.py as the entry point.
- requirements.txt for dependencies.

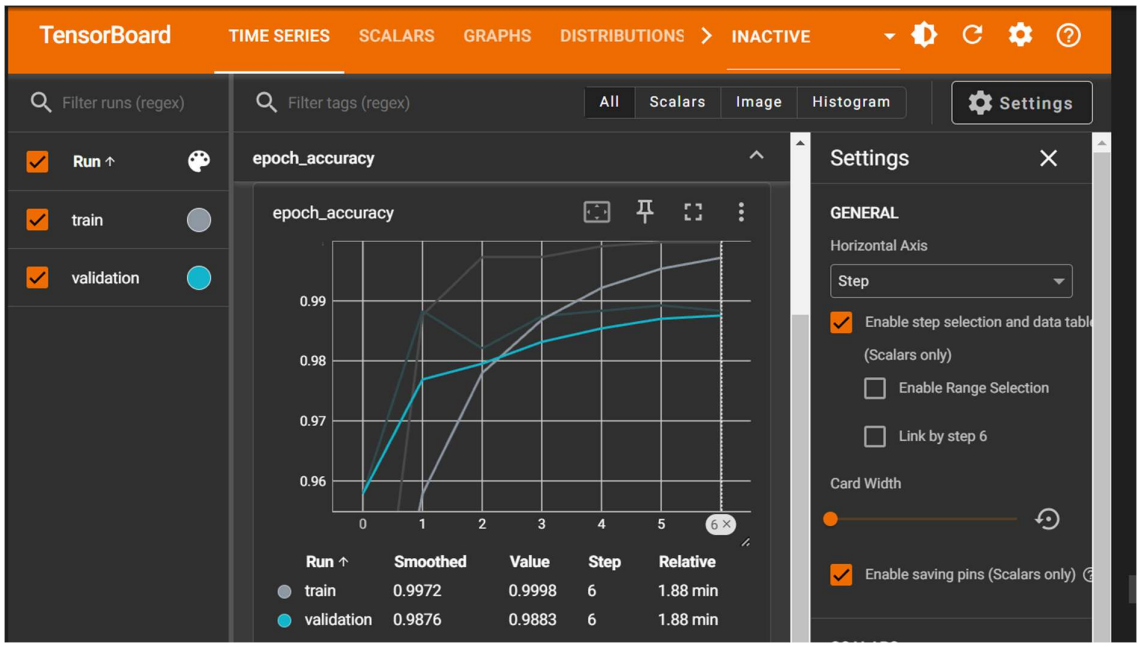
4. Launch and share the deployed URL.

Tensorboard Snaps:

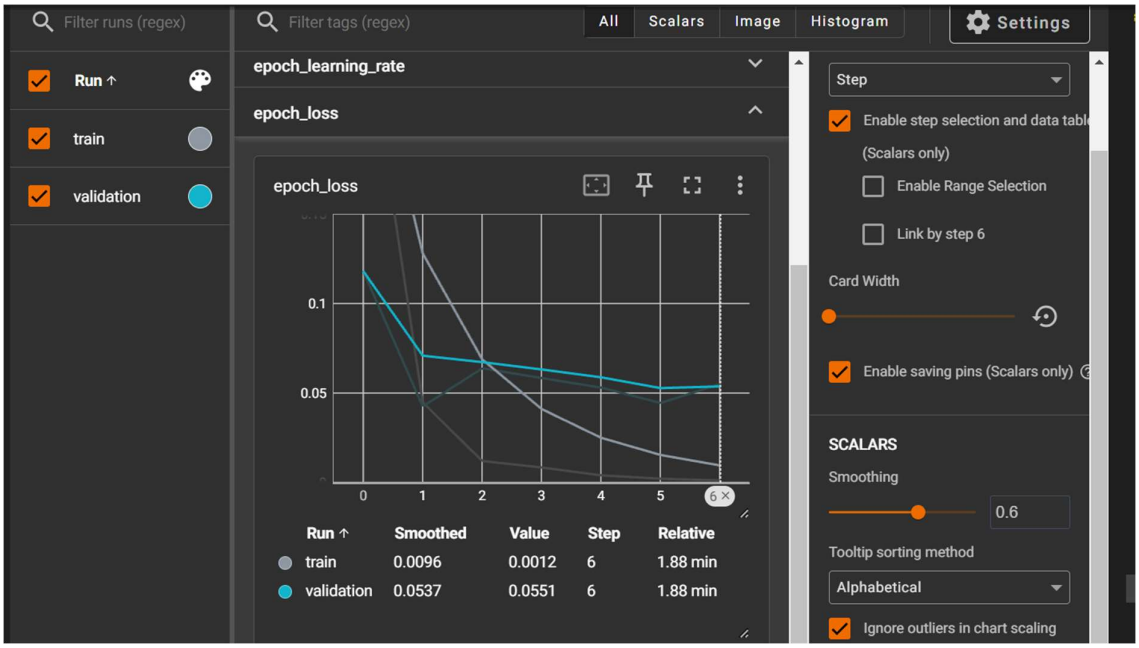
Bias Histogram



Epoch vs Accuracy



Epoch vs Loss



Gradient Descent