Crazy Movie
Group number: 9

Group 9: Purbak Sengupta

Name: Purbak,Jiahui,Kanak,Shahinoor

1 Introduction

In today's world, movies have become an integral part of our everyday entertainment. The movie industry is growing, significant, and seizes the attention of individuals of all ages. There is a huge amount of data about movies, and taglines are usually short but filled with context. They have to connect in subtle which have meaningful ways to the film they represent. And people often check movie ratings because they want to have a quick way to judge if the film is worth watching. A rating gives a snapshot of how others felt about the movie if it's well received, they'll be more likely to watch. It also saves time and helps people avoid movies that might not be enjoyable.

2 Task

In our case, predicting movie titles from taglines or estimating a film's rating and quality category from metadata can serve as a kind of pre-training step. There's a lot of movie data available taglines, titles, user ratings, and popularity metrics and most of it can be collected automatically.

Moreover, this problem suits deep learning well. CNNs (both simple and complex), as well as architectures like ResNet or VGG16, can handle raw inputs directly, and we don't need to rely too heavily on crafted features. As humans aren't particularly good at mapping taglines to titles or guessing ratings from raw numbers, but a neural network can learn these patterns easily if given enough examples. So in this project, we will explore the changes in performance when we vary model complexity and how well the model scales as we pinch the amount of context it's given essentially testing how good a predictor we can build with different levels of complexity and data.

3 Data

We have worked with two datasets and our choice of datasets commonly referred to as the "Movie Identification Dataset [800 Movies]" which contains frames from 800 top-rated movies for building movie identification models and the "Full TMDB Movies Dataset 2024 (1M Movies)" which is a comprehensive movie database that provides information about movies, including details like titles, ratings, release dates, revenue, genres, and much more. The total size of the first dataset is about 12 GB as it includes frames from approximately 800 top-rated movies. Each movie has around 1000 frames, totaling up to 800,000 frames in the dataset.

4 Network Design

For the first part of the project, we did image classification by ResNet50 model. The architecture can be explained by five stages and one classification layer. The stage zero converts the input from 64x16x16 to 64x8x8 by MaxPool layer. Each bottleneck block contains convolutional layer, batch normalization layer and ReLU activated function. The stage one uses three bottleneck blocks to make the network deeper without making the computation too expensive. The stage two uses four bottleneck blocks, the stage three uses six bottleneck blocks and the stage four uses three bottleneck blocks. The model passes the learned features to a AdaptiveAvgpool layer to summary the learned features, then passes the results to two linear layers to make the final decision. Between these two linear layers, there's one dropout layer to avoid overfitting.

We built a model using a CNN-LSTM approach to classify movies into three categories: GOOD, AVG, and BAD, using features like popularity, revenue, and runtime. The CNN component, made up of three 1D convolutional layers (with kernel size 3, ReLU activation, batch normalization, and dropout), captures spatial patterns. The LSTM component, with two layers and a hidden size of 256, focuses on the temporal relationships in the data, also incorporating dropout for regularization. Two fully connected layers then reduce the LSTM's output to a more manageable size (128) before making the final predictions. We also built another CNN-LSTM model which is more advanced with deeper convolutional layers(128-¿256-¿512), batch normalization, dropout and two layers of LSTM with a hidden size of 256. This complex model was used for the dataset where we dropped all the NaN values instead of replacing with zero(more details about this data handling in Splits section). The data, a preprocessed CSV file with information such as titles, vote counts, runtime, revenue, popularity, and average votes, is scaled and reshaped to suit the CNN input format, ensuring effective classification.

5 Training approach

5.1 Data Preparation

For the first dataset,due to the huge size of image dataset, we first generated a small dataset which contains only 20 movies which each movie contains 800 images to test the effectiveness of our model and efficiency of our code. If the result is acceptable, we can then run it on the whole image dataset. Besides the second dataset is loaded and processed to remove inconsistencies, handle missing values, and standardize input features. Also, features are extracted

or transformed based on domain knowledge or requirements for the model. And data may be normalized or scaled for better model convergence, especially when using models sensitive to input scale.

5.2 Splits

For the image dataset, each graph was resized to (32, 32) pixels, converted to a tensor, and normalized by dividing the pixel values by 255, scaling them to the range [0, 1]. The image channels were also rearranged to match the format required by convolutional layers. We created a dictionary 'LabelNamesDict' that maps each movie title with an integer. The whole image dataset was split into training dataset, validation dataset and test dataset. The split ratio was 0.6:0.2:0.2.

In the second dataset,we started by cleaning the data, dropping NaN values from the relevant columns to ensure completeness in the first approach. And the vote_average column was categorized into three parts like GOOD, AVG, and BAD, where we use pd.cut() to facilitate classification. For feature, movie titles were encoded as integers using LabelEncoder, while all numerical features were standardized with StandardScaler to normalize the data and enhance model performance. Furthermore, the dataset was then split into 80% for training and validation and 20% for testing and the training and validation set was further divided, with 60% used for training and 20% (equivalent to 20% of the original dataset) reserved for validation. Finally, all features and labels were converted into PyTorch tensors to prepare the data for model training and evaluation. On the other hand rest of the procedures are same for the second approach except the data splitting where we divided by 50% as testing and 50% training and further have drawn 5% of each dataset to make validation set so overall 45% training and testing and validation has rest 10% and instead of dropping all the NaN values we replaced it with 0 to retain the size of the dataset.

5.3 Workflow

The model training process was organized into a structured loop which ran over multiple epochs. The model performed forward propagation to generate predictions, calculated the loss, and updated the weights through backpropagation for each epoch. Both of the training loss and accuracy were closely monitored to track progress and the model was evaluated on the validation set to assess its generalization ability by tracking validation loss and accuracy. Early stopping was employed, which stopped the training process if the validation accuracy failed to improve within a specified number of epochs (patience). Once training was complete, the best-performing model, saved during training, was tested on the test set to calculate metrics such as test loss, test accuracy, and Mean Squared Error (MSE) between true labels and predictions. For error analysis, a graph was plotted to compare true labels with predicted labels, providing insights into the model's performance and highlighting the computed MSE. For image classification, during each epoch, we calculated both training and validation accuracy. After training, the final model was evaluated on the test dataset to get test accuracy. While the final model might not have the highest validation accuracy during training, its performance was generally close to the best model since it reached a stable state after 40 epochs.

5.4 Training

For image classification, We used cross-entropy as the loss function and the Adam optimizer with a learning rate of 1e-4. For the training dataset, the batch size was 512, while it was set to 256 for the test and validation datasets. We experimented with different CNN architectures, starting with a 27-layer model, then moving to VGG16 with 40 layers, and finally ResNet50 with 179 layers. The test accuracy improved progressively from 30.82% to 40.24%, and then to 40.47%. Due to time limitation, we only tested two cases of batch sizes with the ResNet50 model: 512 for training and 256 for test and validation datasets, versus 256 for all three datasets. The test accuracy dropped from 40.47% to 40.09% with the latter setup, indicating that the first case of batch sizes is more effective.

For working with the second TMDB dataset, we started by splitting the data so that 80% was used for training and validation, and 20% was held out for testing. Within that 80%, we allocated 60% for training and 20% for validation. We trained our CNN-LSTM model using the AdamW optimizer (learning rate: 0.0005, weight decay: 0.0001) and applied early stopping with 15 epochs of patience in validation accuracy. Using cross-entropy loss and a batch size of 64, we trained for up to 100 epochs. However, as the results were the same final output is shown with 25 epochs due to time limitations. This approach allowed the model to effectively learn both spatial and temporal features, achieving a peak validation accuracy of 38.17% in about 712 seconds. On the test set, it reached 37.91%. accuracy and a mean squared error (MSE) of 1.36.

In another approach where we handled the same dataset in a different way, and implemented a simpler version of CNN-LSTM model, we finally got testing accuracy of 83.49%.

6 Results

We correctly predicted 112897 out of 278965 samples. However, the performance of ResNet50 model showed that the training accuracy reached to 96.26%, while the test accuracy was only 40.47%, which suggested overfitting in our model. Both loss and accuracy on training and validation datasets arrived at a stable state when the epoch was close to 40.

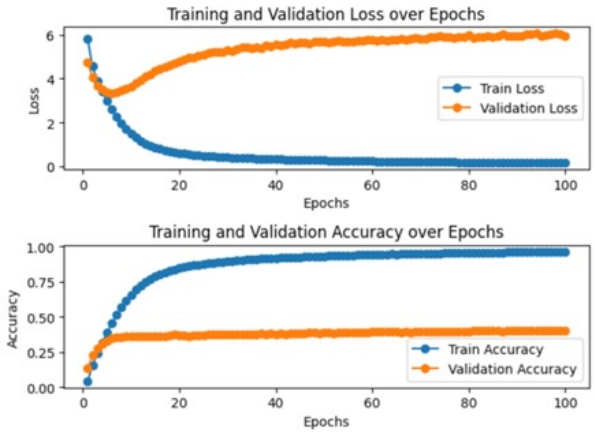

Figure:1

Throughout the training process of the CNN-LSTM model for the second approach(45-10-45 data-split), the training loss steadily decreased(fig 2), reflecting the model's growing ability to fit the training data. Meanwhile, the validation loss reached a stable point after about the 60th epoch.

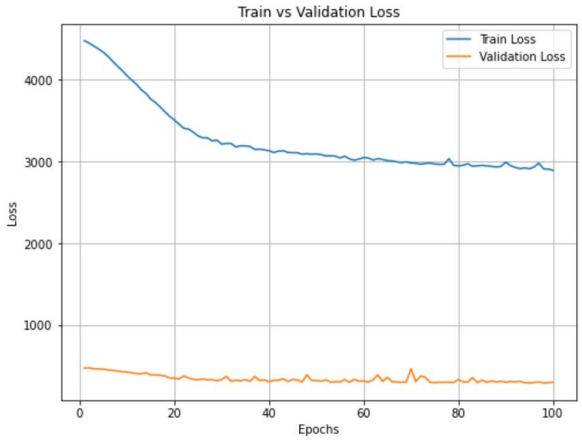

Figure:2

Both the training and validation accuracy improved slowly with each epoch where the validation accuracy tracked closely with the training accuracy. This close alignment suggests that the model was not only learning effectively from the training data but also generalizing well to unseen data.

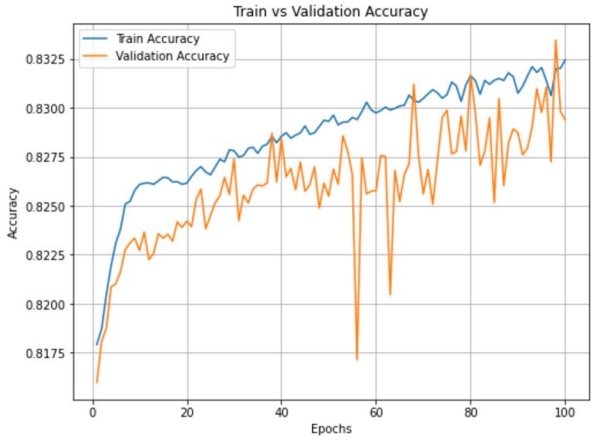

Figure: 3

The bar graph shows that the model's accuracy on the test set closely matches its accuracy on the training set, suggesting that the model is neither overfitting nor underfitting.

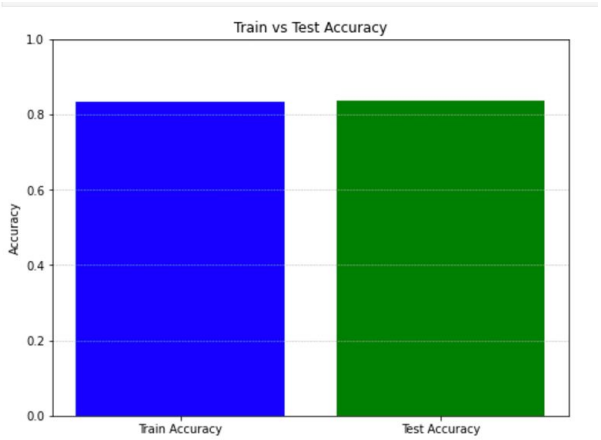

Figure: 4

The confusion matrix shows that the model does a good job of telling the three classes apart, with most predictions landing correctly. The BAD class stands out with high accuracy and almost no confusion with AVG or GOOD. The AVG class, on the other hand, can be mistaken for either BAD or GOOD, probably because the "average" category is a bit fuzzy and harder to define. The GOOD class occasionally gets mixed up with AVG, which might be due to having fewer examples of truly top-quality movies. Overall, though, the model still manages to classify most movies correctly.

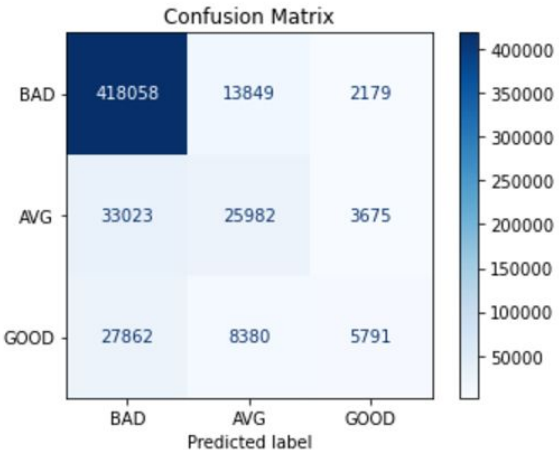

Figure:5