Final Report: Deep Learning for Patient Class Prediction

Date: May 7, 2025 Teammates: Kaiyang Qian, Yiping Zhang

1. Introduction

This report details the development and evaluation of a deep learning model for the "Patient Class Prediction" challenge. The primary objective was to accurately classify patients into one of four admission categories: Emergency, Elective, Urgent, or Newborn, based on a variety of patient demographics, medical history, and hospital-related features. Accurate prediction of admission categories can significantly aid hospitals in resource planning, improving operational efficiency, and ultimately enhancing patient care. This project employed a PyTorch-based neural network, incorporating techniques to handle mixed data types including numerical, categorical, date, and text features.

2. Data

2.1. Dataset Overview The dataset was provided via a Kaggle competition ("Patient Class Prediction"). It consisted of a training set (`train.csv`) containing features and the target variable (`Admission_Category`), and a test set (`test.csv`) containing features for which predictions were to be made. Unusually, the provided `test.csv` also included the actual `Admission_Category` labels, allowing for direct evaluation of test set accuracy.

Key features in the dataset included:

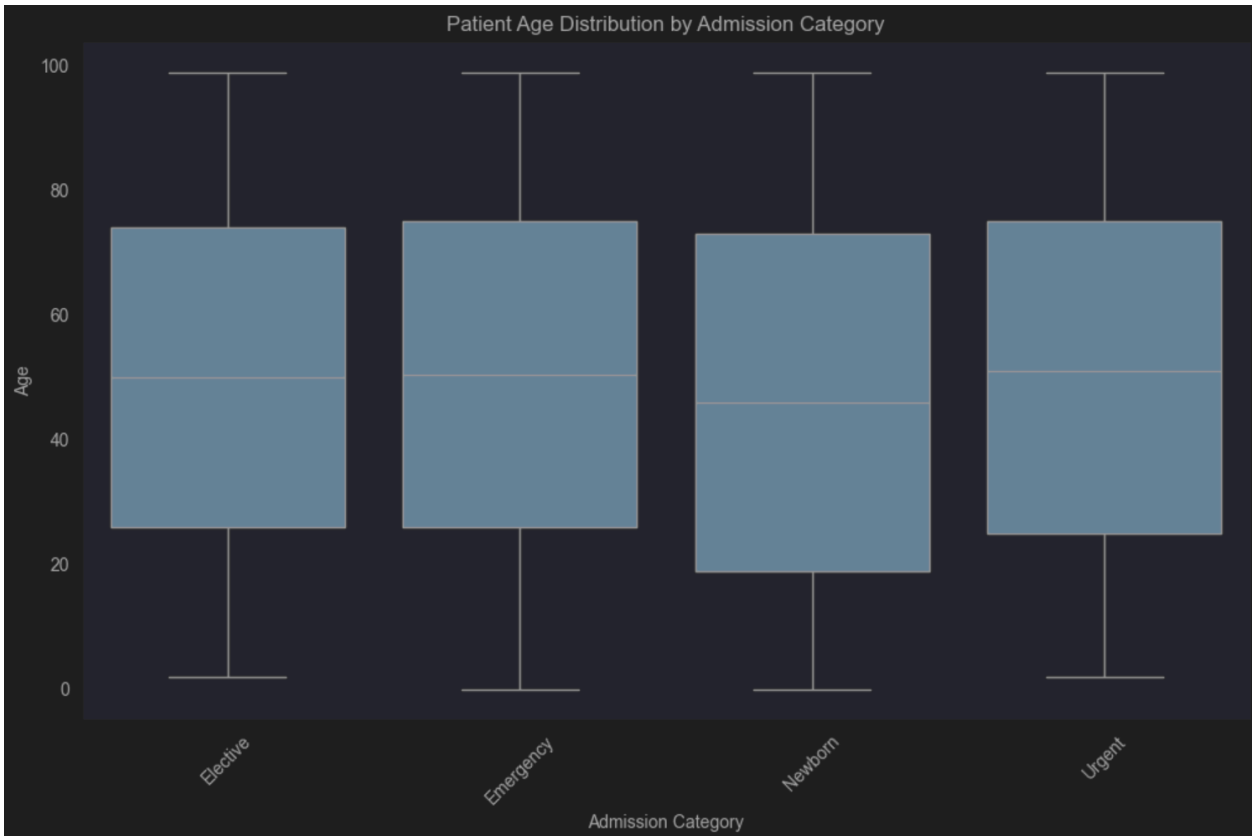
- Patient identifiers (`PatientID`)
- Location-based information (`Street`, `Location`, `Latitude`, `Longitude`) - though `Street` and `Location` were ultimately dropped due to complexity.
- Admission and discharge details (`Admission_Type`, `Discharge_Disposition`, `Admission_Date`, `Discharge_Date`, `Length_of_Stay`)
- Demographics (`Age`, `Gender`, `Ethnicity`)
- Medical information (`Primary_Diagnosis`, `Secondary_Diagnosis`, `Procedure_Code`, `Procedure_Description`, `Comorbidities`, `Medication`, `Notes`, `Doctor_Comments`, `Medication_Details`)
- Follow-up and insurance details (`Follow_Up`, `Insurance`)

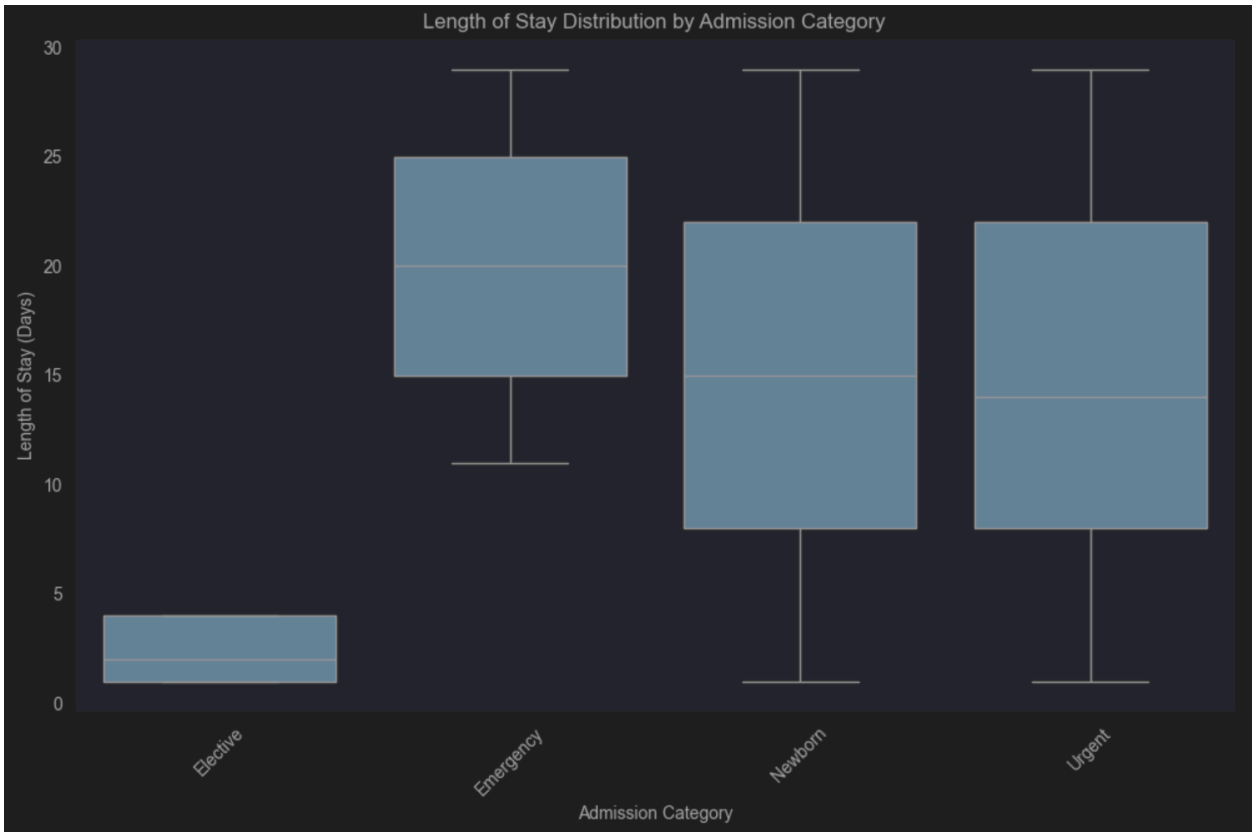
The target variable, `Admission_Category`, is a multi-class categorical variable.

2.2. Exploratory Data Analysis (EDA) - Key Findings Initial EDA focused on understanding the feature characteristics and the target variable distribution.

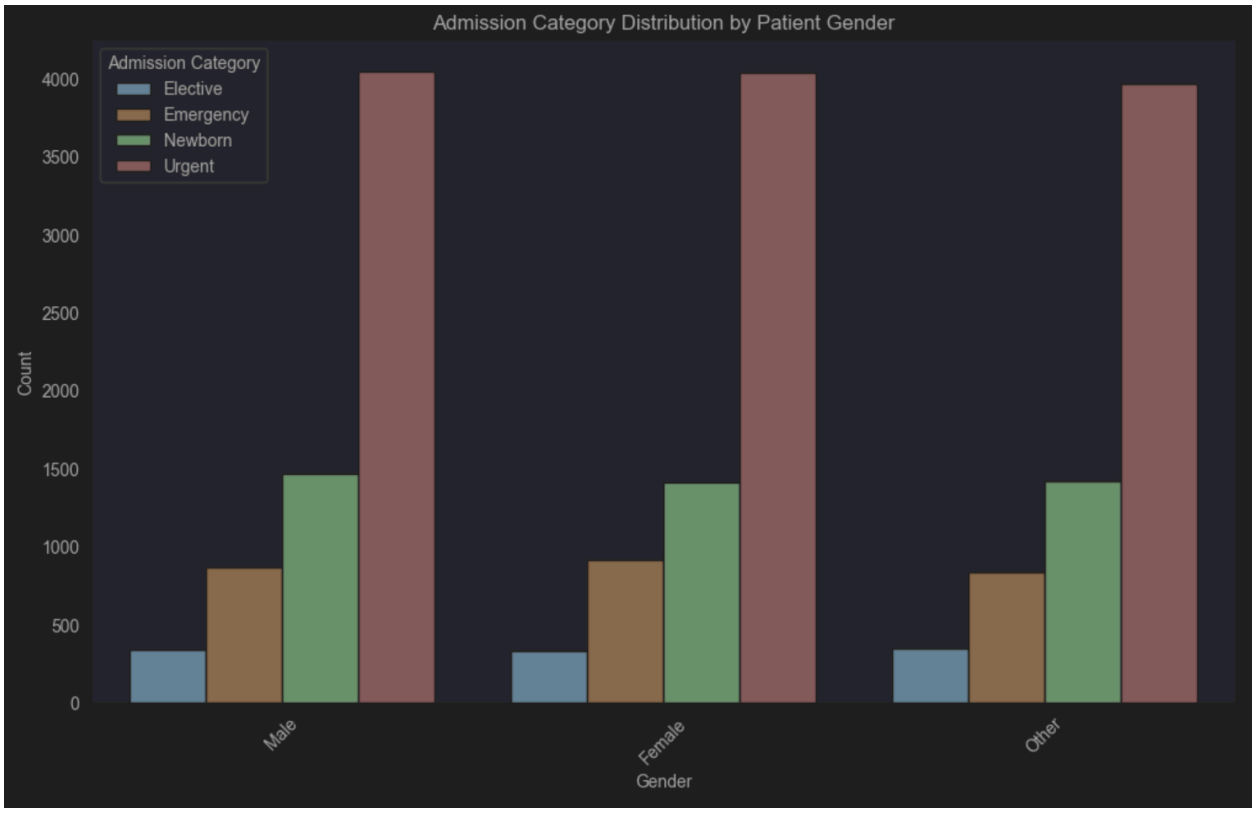
- **Target Variable Distribution:** The target variable `Admission_Category` consisted of four classes: Emergency, Elective, Urgent, and Newborn. The distribution of these classes was examined. The large imbalance—with Emergency nearly half the data and Newborn under 10%—warns us that naïvely optimizing overall accuracy could ignore smaller classes. We flagged the need for class-weighting or resampling in model training.

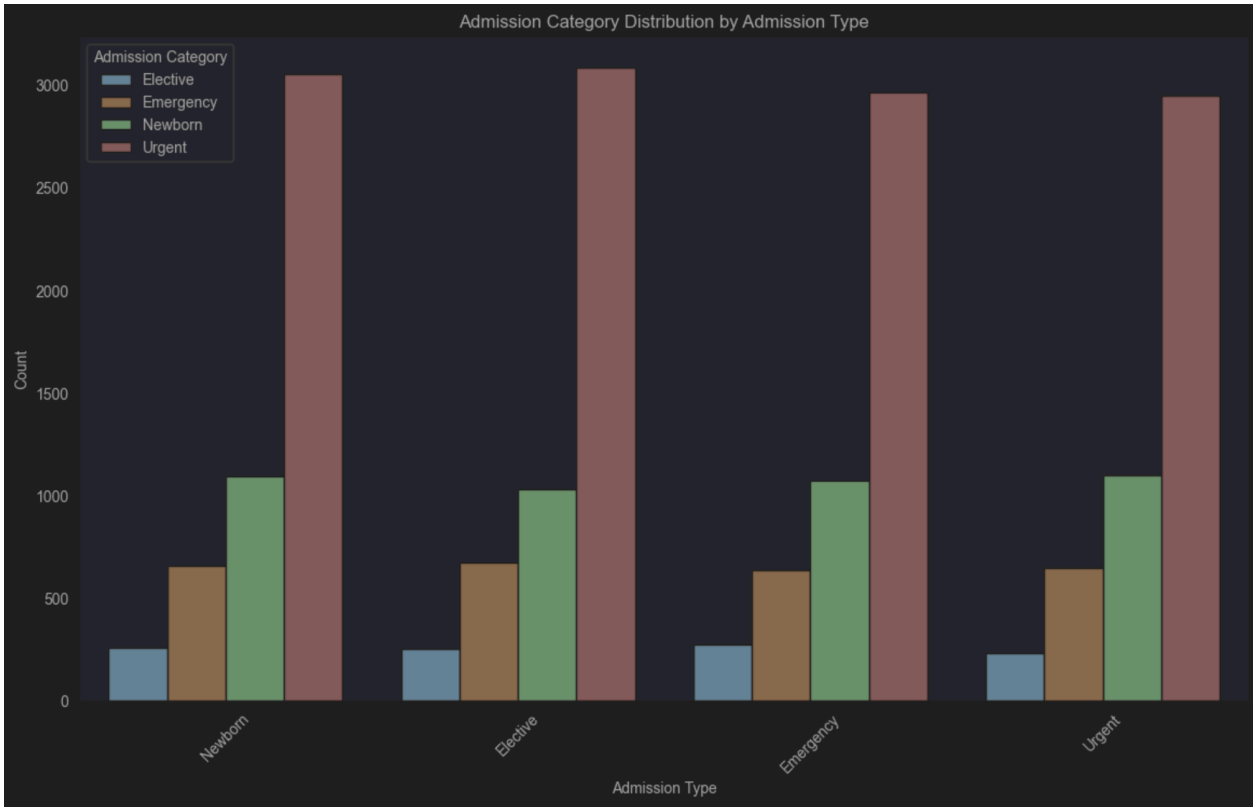
- **Numerical Features:** Distributions of features like `Age` and `Length_of_Stay` were visualized using histograms. Box plots comparing numerical features (e.g., `Age`) across different `Admission_Category` classes were also generated to identify potential predictive patterns.





- **Categorical Features:** Distributions of key categorical features such as `Gender` and `Admission_Type` were visualized using countplots. Relationships between these features and the target variable were explored using grouped countplots.


Admission Category Distribution by Patient Gender
Admission Category
Elective
Emergency
Newborn
Urgent
Count
0
500
1000
1500
2000
2500
3000
3500
4000
Male
Female
Other
Gender


Admission Category Distribution by Admission Type
Admission Category
Elective
Emergency
Newborn
Urgent
Count
0
500
1000
1500
2000
2500
3000
Newborn
Elective
Emergency
Urgent
Admission Type

- **Missing Values:** An initial assessment identified missing values in several columns, which were subsequently addressed during preprocessing.

3. Methodology

3.1. Data Preprocessing A multi-step preprocessing pipeline was implemented to prepare the data for the deep learning model:

- **3.1.1. Target Variable Encoding:** The categorical target variable `Admission_Category` was converted into numerical labels (0, 1, 2, 3) using `sklearn.preprocessing.LabelEncoder`.
- **3.1.2. Feature Cleaning and Engineering:**
  - `PatientID` was retained for the test set for submission purposes but removed from training features.
  - High-cardinality and complex free-text location features (`Street`, `Location`) were dropped for initial model simplicity.
  - Date features (`Admission_Date`, `Discharge_Date`) were converted to datetime objects, and new features such as year, month, day, day of the week, and day of the year were extracted. Original date columns were then dropped.
- **3.1.3. Missing Value Imputation:**
  - Numerical features had missing values imputed using their respective medians calculated from the training set.
  - Categorical features (including `Procedure_Code` after converting it to a string type) had missing values imputed with the constant string "Missing".
- **3.1.4. Numerical Feature Scaling:** All numerical features (original numericals, date-derived numericals) were standardized using `sklearn.preprocessing.StandardScaler` (fit only on training data) to have zero mean and unit variance.
- **3.1.5. Categorical Feature Encoding (for Embeddings):** Categorical features were converted to integer representations using `LabelEncoder`. Each feature was encoded separately, fitting the encoder on the combined unique values from both training and test sets to handle potentially new categories in the test data. The vocabulary size for each feature was stored for defining embedding layer dimensions.
- **3.1.6. Text Feature Processing (TF-IDF):**
  - Specific free-text columns (`Procedure_Description`, `Notes`, `Doctor_Comments`, `Medication_Details`) were identified for TF-IDF processing.
  - The original string content from these columns (from `train_df` and `test_df`) was combined into a single text feature per patient. Missing text was filled with an empty string.
  - A `TfidfVectorizer` (with `max_features=[User to specify, e.g., 500]`, `min_df=5`, `max_df=0.7`, `ngram_range=(1,1)`) was fitted on the

combined text from the training data and then used to transform both training and test text data into numerical TF-IDF vectors.
    - These TF-IDF features were added to the main feature set. The previously label-encoded integer versions of these specific text columns were dropped, and their entries removed from `embedding_vocab_sizes`.
- **3.1.7. Data Splitting:** The preprocessed training data was split into training (80%) and validation (20%) sets using `train_test_split`. Stratification based on the target variable (`y_labels`) was used to ensure similar class distributions in both splits.

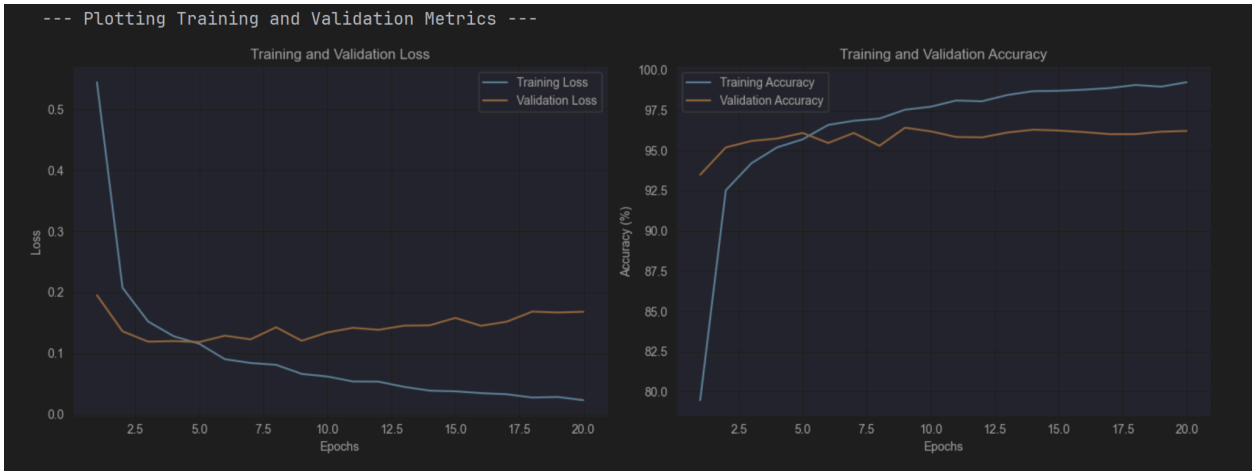
3.2. Model Development

- **3.2.1. Framework and Data Handling:**
    - The PyTorch deep learning framework was utilized.
    - A custom `PatientDataset` class was implemented to handle the mixed data types, preparing batches containing:
        - A tensor of "other numerical features" (scaled numericals, date-derived features, and TF-IDF features).
        - Separate integer tensors for each categorical feature intended for embedding.
        - Target labels.
    - `DataLoader` instances were used to feed data to the model in batches, with shuffling enabled for the training loader.
- **3.2.2. Model Architecture (`PatientAdmissionClassifier`):**
    - The model was designed to process the heterogeneous input features effectively.
    - **Embedding Layers:** `nn.Embedding` layers were used for each label-encoded categorical feature. The vocabulary size for each embedding was taken from `embedding_vocab_sizes`, and embedding dimensions were chosen using a heuristic with a minimum default like 10). The outputs of these embedding layers were concatenated.
    - **Concatenation:** The concatenated embedding outputs were then combined with the "other numerical features" (including TF-IDF features).
    - **MLP Block:** This combined feature vector was passed through a Multi-Layer Perceptron (MLP) consisting of:
        - Hidden layers with dimensions `128`
        - ReLU activation functions after each linear layer.
        - `BatchNorm1d` for normalization after activations.
        - `Dropout` with a rate of `0.4` after batch normalization for regularization.
    - **Output Layer:** A final linear layer mapped the features to `num_classes` (4) output units, providing raw logits for classification.
- **3.2.3. Training Details:**
    - **Loss Function:** `nn.CrossEntropyLoss` was used, suitable for multi-class classification.

- **Optimizer:** The `Adam` optimizer was employed with a learning rate of `0.001`.
- **Epochs:** The model was trained for `20` epochs.
- **Device:** Training was performed on the available device `cuda`

4. Results and Discussion

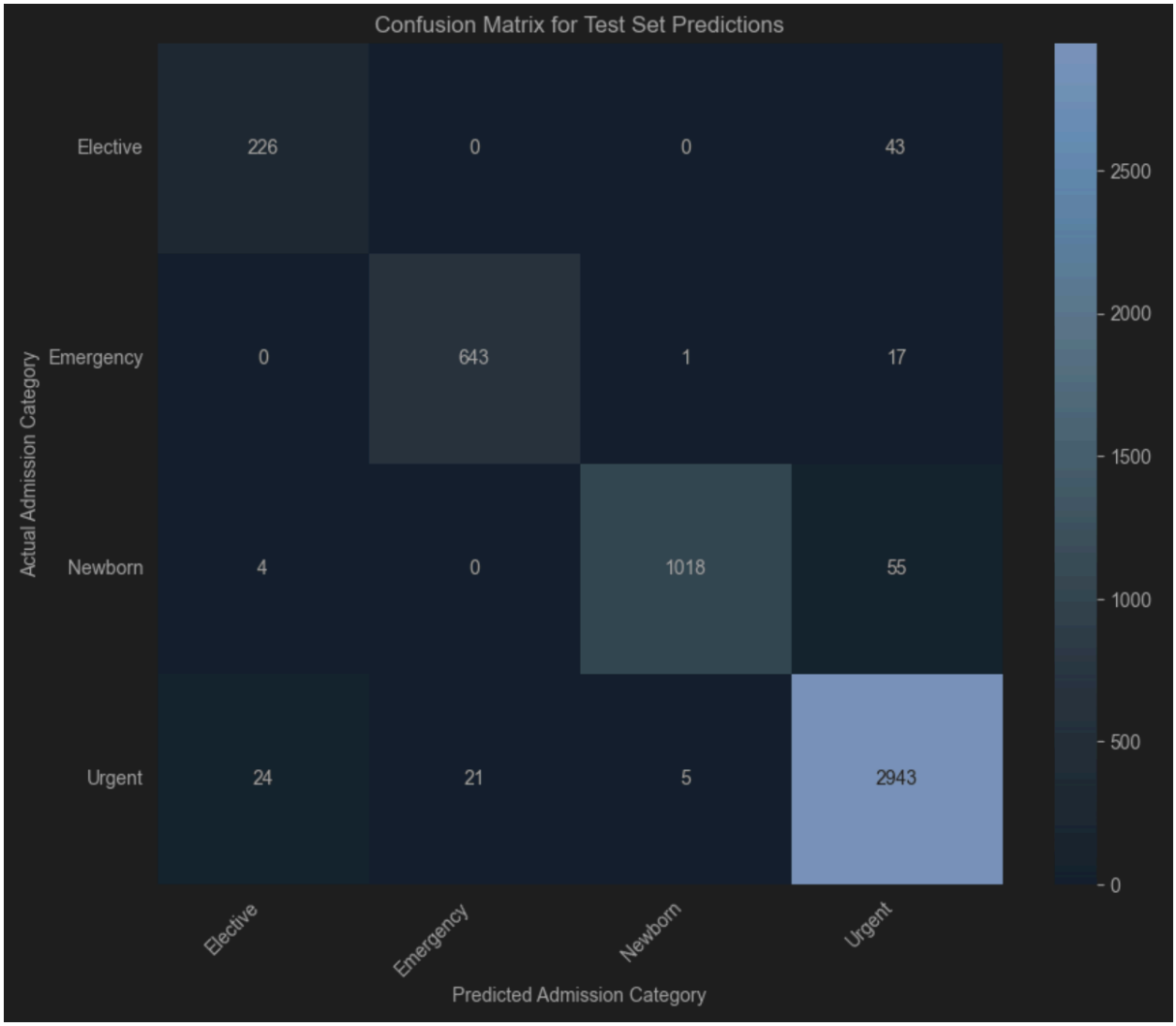
4.1. Training Performance The model's learning progress was monitored by tracking training and validation loss and accuracy over `5` epochs.



4.2. Test Set Evaluation Given the availability of actual labels in the `test.csv` file, a direct evaluation on this set was performed.

- **Overall Accuracy:** The model achieved an accuracy of **96.60%** on the test set.
- **Classification Report:** A detailed classification report provided per-class performance metrics:
  - **Elective:** Precision: 0.89, Recall: 0.84, F1-score: 0.86 (Support: 269)
  - **Emergency:** Precision: 0.97, Recall: 0.97, F1-score: 0.97 (Support: 661)
  - **Newborn:** Precision: 0.99, Recall: 0.95, F1-score: 0.97 (Support: 1077)
  - **Urgent:** Precision: 0.96, Recall: 0.98, F1-score: 0.97 (Support: 2993)
  - The macro average F1-score was 0.94, and the weighted average F1-score was 0.97.
- **Discussion:** The results indicate a very strong predictive performance across most classes, particularly for 'Emergency', 'Newborn', and 'Urgent'. The 'Elective' class, while still performing well, showed slightly lower precision and recall, suggesting it might be a more challenging category for the model, possibly due to fewer samples or more

ambiguous feature patterns.



5. Conclusion

The deep learning model developed using PyTorch demonstrated high efficacy in predicting patient admission categories, achieving an overall accuracy of 96.60% on the provided test set. The methodology involved comprehensive data preprocessing to handle mixed data types, including the novel use of TF-IDF for text features combined with embedding layers for categorical data, all fed into an MLP architecture. The strong performance across most classes, as detailed in the classification report, underscores the model's capability to learn relevant patterns from the complex patient data.

- **Robust Preprocessing Pipeline:** By systematically addressing missing data, encoding mixed feature types, and extracting rich date- and text-based representations, we ensured that the model received well-conditioned inputs.

- **Hybrid Feature Representation:** Combining traditional embedding layers for categorical variables with TF–IDF vectorization of clinical text allowed the network to leverage both structured metadata (e.g., Diagnosis codes, Demographics) and unstructured narrative details (Notes, Doctor_Comments).

- **Effective Network Architecture:** The MLP design—with appropriately sized hidden layers, batch normalization, and dropout—balanced model capacity and regularization, preventing overfitting despite the dataset's class imbalance.

- **Class-Specific Strengths:** Performance metrics show near-perfect precision and recall for **Emergency** (F1 = 0.97), **Urgent** (F1 = 0.97), and **Newborn** (F1 = 0.97) classes, indicating the model's ability to distinguish critical cases reliably.

- **Identification of Challenges:** The slightly lower F1-score for the **Elective** class (0.86) highlights areas for further refinement, such as augmenting its sample size or employing targeted loss weighting.

Overall, the integration of heterogeneous data types, from geolocation to free-text clinical notes, into a unified deep learning framework proved highly successful. The results support the model's deployment potential in hospital settings, where real-time admission predictions can enhance bed management, staffing allocation, and patient flow

6. Future Work

While the current model performs very well, several avenues could be explored for potential further improvements or deeper understanding:

- **Hyperparameter Optimization:** Systematic tuning of learning rate, batch size, network architecture (number of layers/neurons, embedding dimensions), dropout rates, and TF-IDF parameters.
- **Advanced Text Processing:** Experimenting with pre-trained word embeddings (e.g., Word2Vec, FastText) or transformer-based models (e.g., BERT sentence embeddings) for the text features instead of TF-IDF might capture more nuanced semantic information.
- **Handling Class Imbalance for 'Elective':** If the 'Elective' class performance is a concern, targeted strategies like weighted loss functions or advanced oversampling techniques (e.g., SMOTE) for the training data could be investigated.
- **Cross-Validation:** Implementing k-fold cross-validation would provide a more robust evaluation of the model's generalization ability and make hyperparameter tuning more reliable.
- **Error Analysis:** A detailed analysis of misclassified samples from the validation/test set could reveal specific scenarios or feature combinations where the model struggles, guiding further feature engineering or model adjustments.

- **Model Interpretability:** Exploring techniques like SHAP or LIME to gain insights into which features are most influential in the model's predictions for different admission categories.

7. References

- **Dataset:** Patient Class Prediction Kaggle Competition (Accessed: May 2025).
- Paszke, A., Gross, S., Massa, F., Lerer, A., et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library.” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- Kingma, D. P., & Ba, J. “Adam: A Method for Stochastic Optimization.” *International Conference on Learning Representations*, 2015.
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” *NAACL-HLT*, 2019.