The dataset provided is about Heart sounds generated by the beating heart, In our code we are trying to analyse the audio files and classify them using LSTM, Vanilla RNN and GRU and deciding the best model while exploring the hyperparameters.

**1. Data Preprocessing:**

* Define data generators for the training, validation, and possibly test sets specific to audio data.
* Utilize techniques like Mel-frequency cepstral coefficients (MFCCs) extraction or other spectral features to convert raw audio data into a form suitable for training with recurrent networks.

**2. Model Architecture:**

* **LSTM Model**: Define an architecture using LSTM layers suitable for heart sound classification. Given the sequential nature of audio data, LSTM can capture long-term dependencies in the data.
* **Vanilla RNN Model**: Implement a model using SimpleRNN layers, which are the most basic form of recurrent units. This will serve as a baseline to compare against more complex models.
* **GRU Model**: Design a model using GRU layers, which offer a balance between the complexity of LSTM and the simplicity of Vanilla RNN.
* For all architectures, consider integrating the activations and Dropout layers to prevent overfitting.
* The models used, like **lstm\_model**, **rnn\_model**, and **gru\_model**.
* Additionally, grid search is used for different hyperparameters to find the best accuracy parameters.

**3. Compilation:**

* Compile each model using an optimizer like Adamax, RMSprop. Given that it's a classification problem, use 'categorical\_crossentropy' as there are three classes.
* Track metrics like accuracy, mean square error and mean absolute error to monitor the model's performance during training.

**4. Training:**

* Train each model—LSTM, Vanilla RNN, and GRU—using the training and validation data generators.
* Utilize callbacks like **EarlyStopping** and **ModelCheckpoint** to halt training when validation performance plateaus and save the best model respectively.
* Visualize the training and validation loss and accuracy over epochs to inspect convergence and potential overfitting.

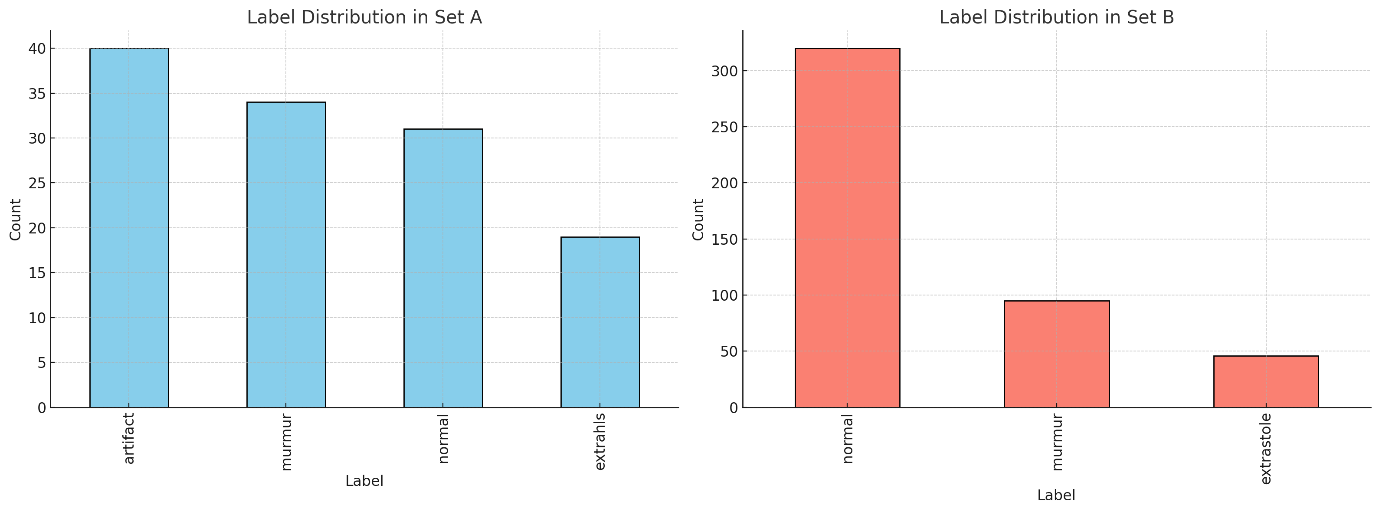
**5. Evaluation and Model Comparison:**

* Evaluate each model on a separate test set using the **evaluate** method.
* Compare the performance of the LSTM, Vanilla RNN, and GRU models to determine the best architecture for the task.

These steps provide a structured approach to implementing and comparing LSTM, Vanilla RNN, and GRU for heart sound classification.

**Dataset Visualizing**

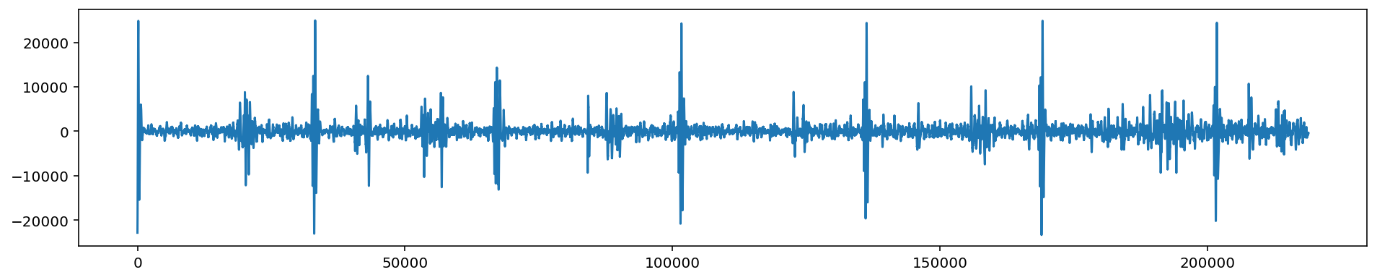
Lets look at the classes available in our dataset



The bar charts show the distribution of labels in both Set A and Set B:

* **Set A**: The majority of the labels are categorized as "artifact", "murmur", or "normal". There are some unlabeled instances which will require handling during pre-processing for any model development.
* **Set B**: This dataset contains a larger variety of labels, including "normal", "murmur", "extrastole", and others, also with a substantial number of unlabeled instances.

The below plot shows the waveform for a sample audio file and from the peaks we can understand the lub dub sounds.



**Architecture**

**1. LSTM Network:**

* **Layers**:
  + 3 LSTM layers with units of 100, 50, and 25 respectively.
  + 3 Dropout layers to prevent overfitting.
  + 2 Dense (fully connected) layers with 50 and 3 units respectively.
* **Parameters**:
  + Total parameters: 80,053
  + Trainable parameters: 80,053
  + Non-trainable parameters: 0
* The LSTM-based model is being used, given the recurrent nature of audio data.
* Dropout layers are added to prevent overfitting.
* The model is compiled using the 'categorical\_crossentropy' loss, suitable for classification tasks, and metrics like accuracy, mean squared error, and mean absolute error are tracked.
* The model's performance is evaluated on different datasets: training, testing, validation, and unlabeled.
* The LSTM model uses callbacks during training, specifically **EarlyStopping** and **ModelCheckpoint**, to prevent overfitting and save the best model.

**2. Vanilla RNN:**

* **Layers**:
  + 3 SimpleRNN layers with units of 100, 50, and 25 respectively.
  + 3 Dropout layers to prevent overfitting.
  + 2 Dense layers with 50 and 3 units respectively.
* **Parameters**:
  + Total parameters: 21,103
  + Trainable parameters: 21,103
  + Non-trainable parameters: 0
* After the initial LSTM model, there's another model implemented using a Vanilla RNN (**SimpleRNN** in Keras).
* Both the LSTM and Vanilla RNN models are compiled with similar loss functions and metrics.

**3. GRU Network:**

* **Layers**:
  + 3 GRU layers with units of 100, 50, and 25 respectively.
  + 3 Dropout layers to prevent overfitting.
  + 2 Dense layers with 50 and 3 units respectively.
* **Parameters**:
  + Total parameters: 60,928
  + Trainable parameters: 60,928
  + Non-trainable parameters: 0
* After the LSTM and Vanilla RNN models, a new model using Gated Recurrent Units (GRU) is being implemented. GRUs are another type of recurrent neural network.
* The GRU model has a similar structure to the previous models, with multiple GRU layers interspersed with Dropout layers.
* The performance of the Vanilla RNN model is evaluated on various datasets.
* The GRU model is compiled with similar loss functions and metrics as the previous models.
* The GRU model is then trained using a subset of the data for training and validation.

**Analysis:**

* **Complexity**:
  + The LSTM network has the highest number of parameters, making it the most complex model among the three. This is expected as LSTM units inherently have more parameters than GRU or SimpleRNN units.
  + The Vanilla RNN has the fewest parameters, making it the least complex. SimpleRNNs are the most basic form of recurrent units.
  + The GRU network lies in between the LSTM and Vanilla RNN in terms of complexity.
* **Dropout Layers**:
  + All three models utilize dropout layers after each recurrent layer, which is a regularization technique to prevent overfitting.
* **Output Layers**:
  + All three models end with two dense layers. The final dense layer has 3 units, likely corresponding to 3 classes for classification, and uses a softmax activation function for probability distribution.
* **Recurrent Layers**:
  + LSTMs are generally more powerful than GRUs and SimpleRNNs, especially for longer sequences, because they can capture long-term dependencies using their gating mechanisms. However, they are also more computationally intensive.
  + GRUs are a compromise between LSTMs and SimpleRNNs. They have gating mechanisms like LSTMs but are less parameterized.
  + SimpleRNNs are the most basic form and can suffer from the vanishing gradient problem, especially with longer sequences. They might not perform as well on complex tasks compared to LSTMs or GRUs.
* **Size**:
  + In terms of memory, the LSTM model will consume the most space, followed by the GRU model, and then the Vanilla RNN.

**Q1: Hyperparameter Tuning on LSTM Algorithm**

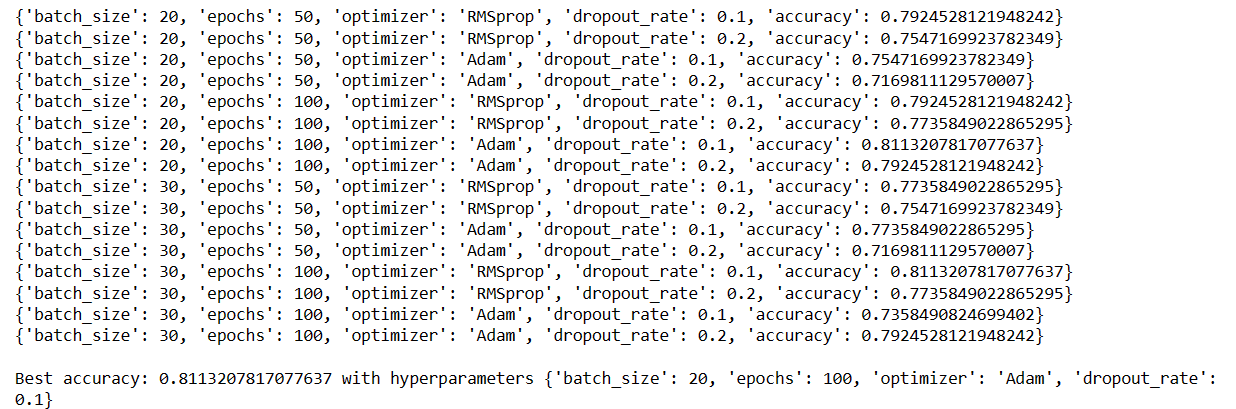
Based on the hyperparameter tuning results for the LSTM model, let's analyze the performance:

**Hyperparameters Explored:**

1. **Batch Size:** 20, 30
2. **Epochs:** 50, 100
3. **Optimizer:** RMSprop, Adam
4. **Dropout Rate:** 0.1, 0.2

**Best Model:**

* **Batch Size:** 20
* **Epochs:** 100
* **Optimizer:** Adam
* **Dropout Rate:** 0.1
* **Best Validation Accuracy:** 81.13%

****

**Observations:**

1. **Optimizer Preference:** The Adam optimizer generally provided better or comparable results when matched against RMSprop with similar other hyperparameters. This indicates that, for this dataset, Adam may be a more suitable optimization technique.
2. **Dropout Rate Impact:** A lower dropout rate of 0.1 seems to yield slightly better results compared to 0.2, especially when combined with the Adam optimizer and 100 epochs.
3. **Batch Size and Epochs:** The combination of a smaller batch size (20) and a larger number of epochs (100) provided the best accuracy. This suggests that the model benefits from more frequent weight updates (due to the smaller batch size) and longer training duration (more epochs).
4. **Consistency Across Runs:** Some configurations produce similar accuracies. For instance, using a batch size of 20, RMSprop optimizer, 50 epochs, and a dropout rate of 0.1 gives an accuracy close to that achieved with 100 epochs. This suggests that beyond a certain point, additional training might not result in significant accuracy improvements and could potentially lead to overfitting.

**Conclusion:**

* **Optimizer and Dropout:** The Adam optimizer combined with a dropout rate of 0.1 consistently gives good results. It appears that this dropout rate strikes a balance between providing regularization (to prevent overfitting) and allowing the model to learn from the data.
* **Batch Size and Epochs:** While the difference between batch sizes of 20 and 30 isn't profound, the smaller batch size of 20 combined with 100 epochs gives the best results.
* **Best Model Selection:** The hyperparameter tuning process is crucial in identifying the best combination of parameters for the dataset. The best model achieves an accuracy of 81.13% on the validation set, which is commendable. This model utilizes a batch size of 20, 100 epochs, the Adam optimizer, and a dropout rate of 0.1.

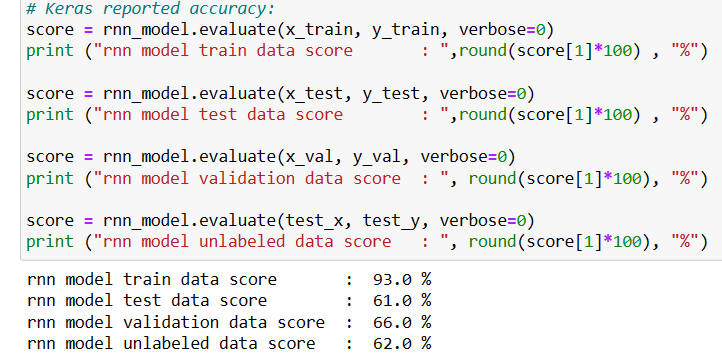
The goal of hyperparameter tuning is not just to achieve the highest accuracy, but to find a model that generalizes well to new, unseen data.

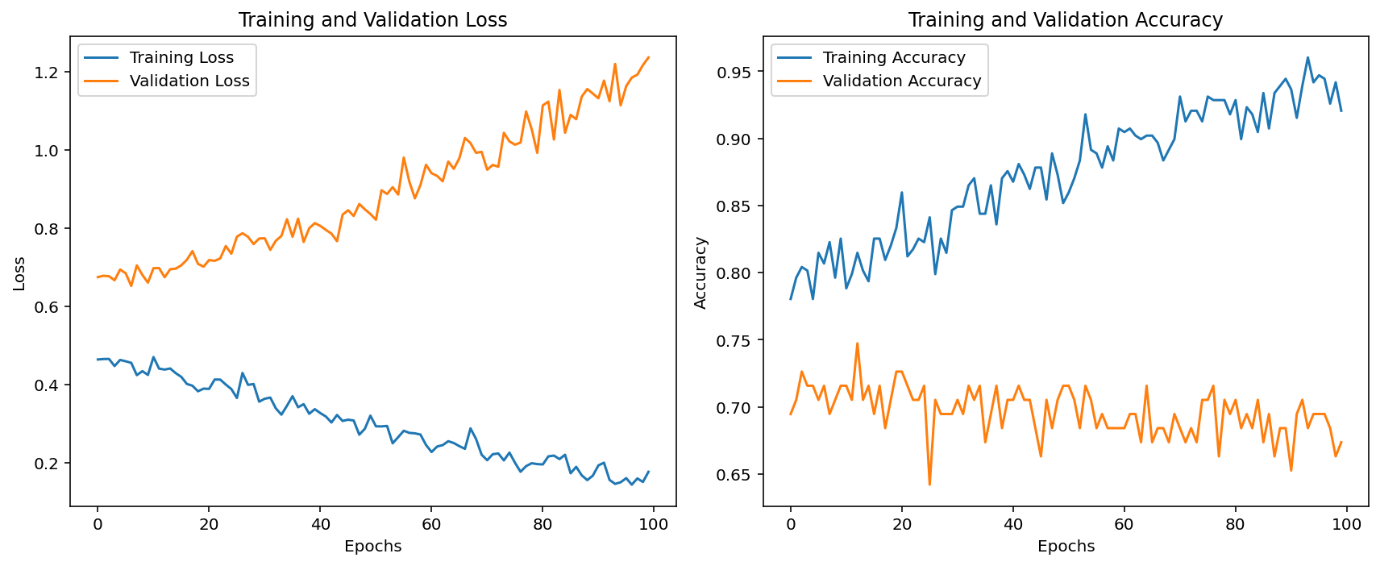
**Q 2: Comparing the Performance of Vanilla RNN vs LSTM vs GRU network**

The performance results show a clear comparison between the three types of recurrent neural networks (RNNs): Vanilla RNN, LSTM, and GRU. Let's analyze these results:

**Vanilla RNN:**

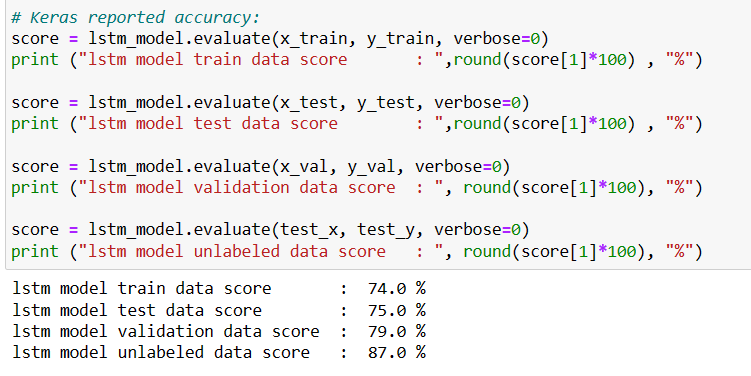
* **Training Accuracy:** 93.0%
* **Test Accuracy:** 61.0%
* **Validation Accuracy:** 66.0%
* **Unlabeled Data Accuracy:** 62.0%

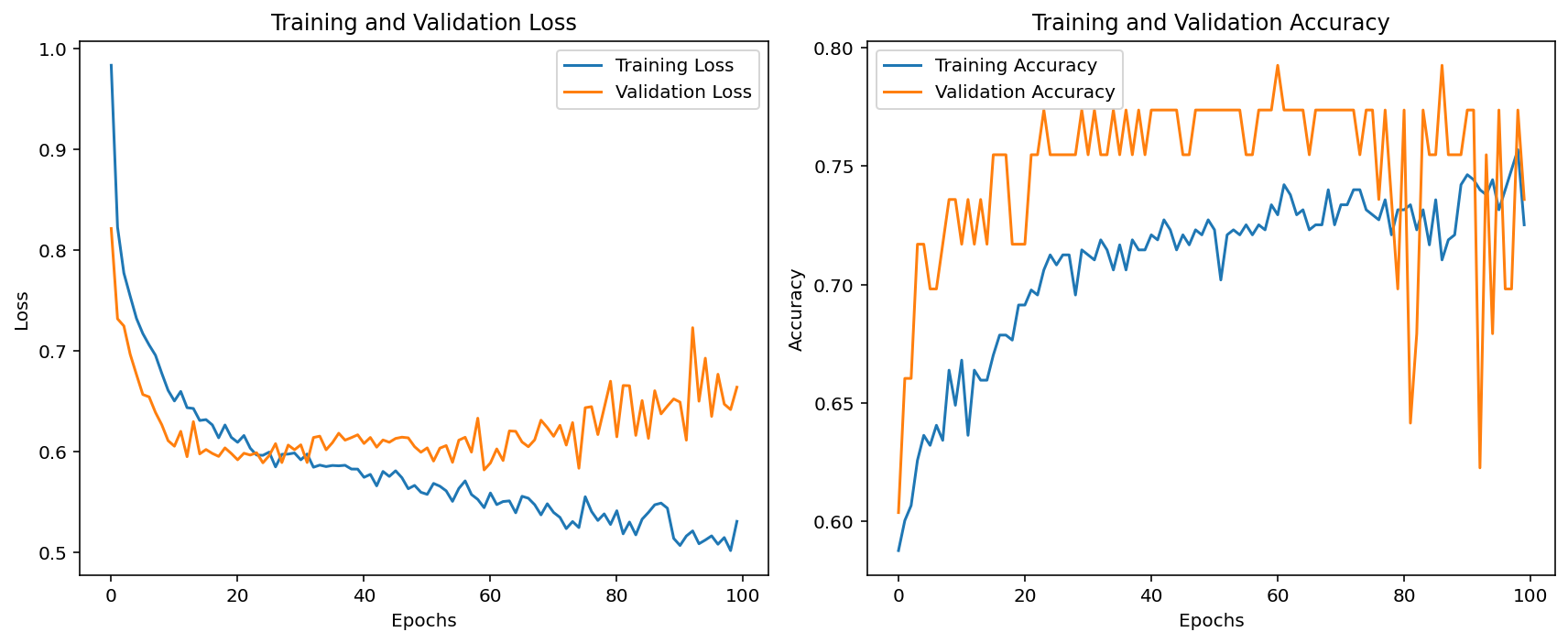
****

****

**LSTM:**

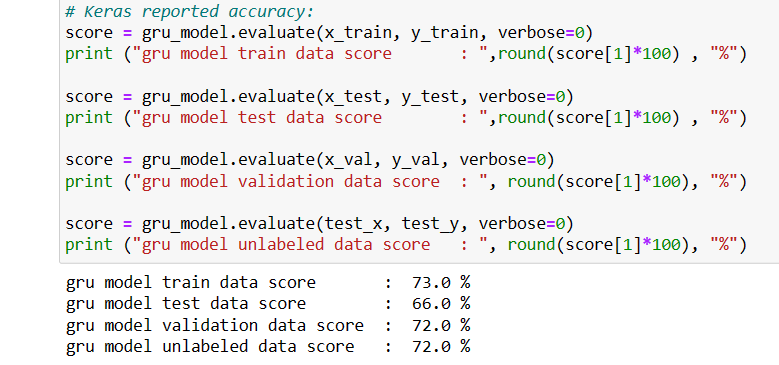
* **Training Accuracy:** 74.0%
* **Test Accuracy:** 75.0%
* **Validation Accuracy:** 79.0%
* **Unlabeled Data Accuracy:** 87.0%

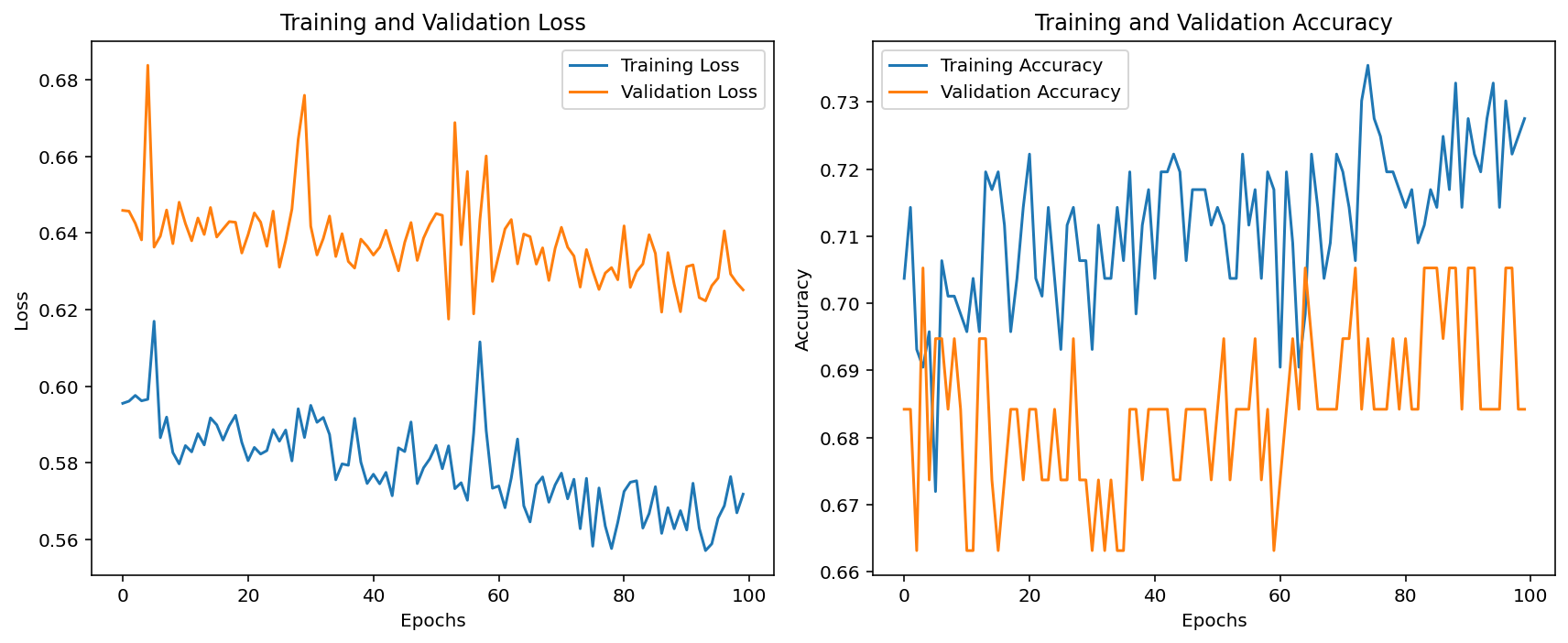
****

****

**GRU:**

* **Training Accuracy:** 73.0%
* **Test Accuracy:** 66.0%
* **Validation Accuracy:** 72.0%
* **Unlabeled Data Accuracy:** 72.0%





**Observations:**

1. **Overfitting in Vanilla RNN:** The Vanilla RNN has the highest training accuracy of 93.0%, but its test, validation, and unlabeled data accuracies are significantly lower. This indicates that the Vanilla RNN model may have overfitted to the training data and lacks generalization capability on unseen data.
2. **LSTM Performance:** LSTM seems to have the best generalization performance among the three. It has balanced training and validation accuracies, indicating that it's neither overfitting nor underfitting. Moreover, the LSTM performs exceptionally well on unlabeled data with an accuracy of 87.0%, suggesting that the model generalizes well to new, unseen data.
3. **GRU Performance:** The GRU model's performance is consistent across training, test, validation, and unlabeled data, with accuracies hovering around the 70% mark. This indicates a balanced fit without significant overfitting or underfitting.

**Conclusion:**

* **Vanilla RNNs** are simpler models, but they are more prone to problems like vanishing and exploding gradients, leading to overfitting or underperformance, especially on complex datasets.
* **LSTMs** are designed to capture long-term dependencies and avoid the vanishing gradient problem. In this dataset, the LSTM seems to offer the best generalization capability, making it the most suitable model for this task.
* **GRUs** are a variation of LSTMs and are designed to be more computationally efficient. They perform comparably to LSTMs in many tasks but, in this case, they slightly underperformed compared to the LSTM.

The LSTM appears to be the best choice for this specific dataset and task.

**Q 3: Explore Regularization Technique for LSTM**

Comparing the performance of the LSTM model with regularization techniques (early stopping and dropout) versus the normal LSTM model.

**LSTM with Regularization:**

* **Training Accuracy:** 74.0%
* **Test Accuracy:** 69.0%
* **Validation Accuracy:** 74.0%
* **Unlabeled Data Accuracy:** 77.0%

**Normal LSTM:**

* **Training Accuracy:** 74.0%
* **Test Accuracy:** 75.0%
* **Validation Accuracy:** 79.0%
* **Unlabeled Data Accuracy:** 87.0%

**Observations:**

1. **Training Accuracy**: Both models have the same training accuracy (74.0%), which means the regularization did not affect the model's ability to fit the training data.
2. **Validation & Test Accuracy**: The regular LSTM model has higher validation and test accuracies compared to the LSTM with regularization. This suggests that, for the current dataset and model architecture, the early stopping and dropout did not enhance the generalization capability.
3. **Unlabeled Data Accuracy**: The regular LSTM performs significantly better on the unlabeled data (87.0% vs 77.0%). This is an important metric, especially if the unlabeled data represents new, unseen data that the model might encounter in a real-world setting.
4. **Early Stopping**: The addition of early stopping is intended to prevent overfitting by halting the training process if the model's performance on the validation set does not improve for a defined number of epochs (patience). In this case, the early stopping might have halted training prematurely, potentially before the model reached its optimal performance.
5. **Dropout**: Dropout is also a regularization technique designed to prevent overfitting by randomly setting a fraction of input units to 0 during training. The rates used (0.2) are common, but sometimes it may be necessary to adjust these rates or experiment with other forms of regularization.

**Conclusion:**

The regular LSTM model outperforms the LSTM with added regularization on this dataset, especially in terms of validation, test, and unlabeled data accuracies.

* The early stopping might have been too aggressive, stopping training before the model reached its peak performance.
* While dropout is a commonly used technique to prevent overfitting, in this case, it might not have been necessary or could have been implemented with a different rate.

For the current dataset and task, the regular LSTM appears to be the better choice.