**1. Hyperparameter Tuning**

**Objective**

The initial step in improving the model’s performance involved fine-tuning the hyperparameters. The aim was to optimize the training process and achieve better results by adjusting key parameters.

**Initial Configuration**

The model was initially configured with the following hyperparameters:

* **Learning Rate**: 3e-4
* **Weight Decay**: 3e-6
* **Batch Size**: 2
* **Proprioception Dimension**: 8
* **Action Dimension**: 7
* **Visual Embedding Dimension**: 224

These hyperparameters were based on standard values that typically work well for similar tasks but had room for optimization to better suit the specific requirements of the robotic task.

**Tuning Process**

The hyperparameter tuning process involved systematically altering the learning rate, weight decay, and batch size to observe their impact on the model's performance. Small, incremental adjustments were made to each parameter, followed by retraining the model to measure the effects on loss and accuracy.

**Outcomes**

* **Learning Rate**: Fine-tuning this parameter had a noticeable impact on the model's convergence speed and stability. Slight reductions or increases led to variations in how quickly the model learned from the data.
* **Weight Decay**: Adjustments in weight decay helped in controlling overfitting, particularly when the model started showing signs of high variance.

**2. Optuna Hyperparameter Optimization**

**Objective**

After the initial manual hyperparameter tuning yielded moderate improvements, the next step was to employ Optuna, an automatic hyperparameter optimization framework. The goal was to systematically explore a larger hyperparameter space to identify the optimal settings for the model, thereby improving its performance more effectively.

**Optuna Setup**

Optuna was chosen for its efficiency in exploring hyperparameter spaces using advanced algorithms like Tree-structured Parzen Estimator (TPE). This allowed for a more focused search compared to traditional grid search methods.

* **Number of Trials**: 50
* **Hyperparameters Tuned**:
  + **Learning Rate**: [Explored a range of values from 1e-5 to 1e-3]
  + **Weight Decay**: [Explored values from 1e-7 to 1e-4]

**Tuning Process**

Optuna's optimization process involved running multiple trials, with each trial testing a different combination of hyperparameters. The framework evaluated the performance of each trial based on a predefined objective function, which in this case, was the model’s validation loss. The trial with the best result provided the optimal set of hyperparameters.

The process involved the following steps:

1. **Objective Function Definition**: The objective function was defined to minimize the validation loss during training.
2. **Parameter Sampling**: Optuna automatically sampled hyperparameters within the defined ranges, using its TPE sampler to prioritize promising regions in the hyperparameter space.
3. **Model Training and Evaluation**: For each trial, the model was trained using the sampled hyperparameters, and its performance was evaluated based on the validation loss.
4. **Optimal Hyperparameters Identification**: After 50 trials, Optuna identified the best set of hyperparameters that minimized the validation loss.

**Outcomes**

The Optuna optimization process significantly enhanced the model's performance by identifying a set of hyperparameters that were better suited to the specific task. The optimal hyperparameters discovered were:

* **Learning Rate**: [Optuna-identified optimal value]
* **Weight Decay**: [Optuna-identified optimal value]

These optimized settings resulted in better convergence and overall model accuracy compared to the manually tuned hyperparameters. The use of Optuna streamlined the tuning process, making it more efficient and effective than manual methods.

**Performance Improvement**

The model showed a noticeable improvement in both training and validation performance. The reduction in validation loss indicated better generalization, and the accuracy on test sets also improved. This step demonstrated the value of automated hyperparameter tuning in complex models.

With these optimal hyperparameters, the foundation was set for exploring further enhancements, including changes to the model architecture, to push the performance even further.

**3. Model Architecture Adjustments**

**Original Model Architecture**

The initial model architecture, as provided by the client, consisted of a convolutional neural network (CNN) followed by an LSTM network. The CNN was used to process visual inputs, while the LSTM network was designed to manage temporal dependencies. The architecture was relatively simple, with fewer convolutional layers and a higher dimensionality in the LSTM network.

* **Convolutional Layers**: The model had five convolutional layers with an increasing number of filters (8, 16, 32, 64, 64). The kernel size was consistently set to 5x5, with padding and stride values adjusted to reduce the spatial dimensions gradually.
* **LSTM Layer**: The LSTM layer had a high dimension of 4096 + proprioceptive dimension, which was designed to capture the temporal dynamics of the task.
* **Dropout**: A dropout layer was applied with a probability of 0.4 to prevent overfitting.
* **Output Layer**: A fully connected layer was used to map the LSTM outputs to the action space.
* **Optimizer**: The Adam optimizer was used with a learning rate and weight decay specified in the configuration.

**Modifications to the Model Architecture**

To improve the model's performance, several key adjustments were made to the architecture:

1. **Reduced LSTM Dimension**: The LSTM dimension was reduced from 4096 to 512 to decrease the model's complexity and prevent potential overfitting. This also aligned better with the reduced feature map size after the CNN layers.
2. **Convolutional Network Adjustments**:
   * **Increased Depth**: The number of convolutional layers was increased from five to six, adding an additional layer to capture more complex visual features.
   * **Adjusted Channels**: The number of channels in each layer was modified to provide a smoother transition between layers. Specifically, the channels were set to 8, 16, 32, 64, 128, and 256, progressively increasing the depth of the network.
   * **Smaller Kernel and Stride Adjustments**: The kernel size in the last two convolutional layers was reduced to 3x3 to retain more spatial information, and the stride was adjusted to prevent excessive reduction in spatial dimensions.
3. **MaxPooling Layer**: The max-pooling operation was retained but with a smaller kernel size, ensuring that the feature map size was reduced without losing critical information.
4. **Scheduler for Learning Rate Adjustment**: A learning rate scheduler (ReduceLROnPlateau) was introduced to adjust the learning rate dynamically based on the validation loss, helping to improve convergence.
5. **Metric Calculation**: An additional metric calculation step was added to monitor the accuracy of the model during training, providing better insight into the model's performance.

**Impact on Performance**

These architectural changes led to a noticeable improvement in the model's performance, particularly in terms of the accuracy and stability of the training process. However, the overall improvement was not sufficient to meet the desired performance levels, indicating that further optimizations or a different architectural approach might be necessary

**4. Updated Model Architecture: Enhanced Convolutional and LSTM Network**

**Overview of the New Architecture**

In this iteration, the model architecture was significantly enhanced by incorporating deeper convolutional layers, batch normalization, and strategic use of max-pooling. These modifications aimed to improve the model's ability to capture and process complex visual features while maintaining the temporal dynamics essential for robotic manipulation tasks.

**Key Components of the Updated Architecture**

1. **Convolutional Network with Batch Normalization**:
   * **Depth and Channels**: The network was deepened to six convolutional layers, with the number of channels progressively increasing from 8 to 256. This setup allowed the model to extract hierarchical features, capturing both low-level details and high-level abstractions.
   * **Batch Normalization**: Batch normalization layers were introduced after each convolutional layer to stabilize the learning process, improve convergence speed, and reduce sensitivity to initialization.
   * **Selective Max-Pooling**: Max-pooling was applied selectively after certain layers (conv1, conv3, and conv5) to downsample the feature maps while retaining important spatial information. This approach helped in controlling the dimensionality and computational complexity.
2. **Adjusted LSTM Network**:
   * **Reduced LSTM Dimension**: The dimension of the LSTM network was reduced to 512 plus the proprioceptive dimension, aligning with the reduced feature map size from the convolutional layers. This adjustment helped in preventing overfitting and made the model more efficient.
   * **Integrated Linear Layer for Compatibility**: A linear layer was added after the convolutional network to ensure the output dimensions matched the LSTM input requirements, enhancing the integration between the CNN and LSTM.
3. **Optimizer with Parameter Groups**:
   * **Learning Rate Scheduling**: The optimizer was configured with different learning rates for the LSTM and the convolutional layers. A scheduler (ReduceLROnPlateau) was also introduced to dynamically adjust the learning rate based on the validation loss, improving the model's ability to converge.
   * **Gradient Descent Optimization**: The Adam optimizer was retained, with specific adjustments to learning rates for different parts of the network, helping in fine-tuning the learning process.
4. **Regularization Techniques**:
   * **Dropout**: A dropout layer with a probability of 0.4 was used to prevent overfitting by randomly deactivating a fraction of the neurons during training.
   * **Normalization and Pooling**: The combination of batch normalization and max-pooling not only stabilized the training but also helped in generalizing the model to unseen data.
5. **Metric Calculation**:
   * **Accuracy Measurement**: An additional metric calculation function was introduced to monitor the accuracy of predictions against the actual actions, providing a more comprehensive understanding of the model's performance during training and validation.

**Impact on Performance**

The introduction of batch normalization, selective max-pooling, and a deeper convolutional network resulted in a model that was more robust and capable of learning complex visual features. The reduced LSTM dimension and the inclusion of a learning rate scheduler further refined the model, making it more efficient and stable during training. However, while these improvements led to better performance metrics, the task's complexity still posed challenges, indicating that additional strategies or architectural changes might be needed to achieve optimal results.

**5. Third Architecture Iteration: Optimized Convolutional and LSTM Network**

**Overview of the New Architecture**

In this third iteration, the architecture was further refined by optimizing the convolutional network depth, adjusting the LSTM input size, and incorporating more robust regularization techniques. Additionally, the introduction of new evaluation metrics provided a more comprehensive understanding of the model's performance, particularly in terms of precision, recall, and F1 score.

**Key Components of the Updated Architecture**

1. **Enhanced Convolutional Network**:
   * **Deeper Network**: The convolutional network was deepened to six layers, with the number of channels ranging from 16 to 512. This increase in depth allowed for the extraction of even more detailed features from the visual input, crucial for tasks requiring fine-grained perception.
   * **Batch Normalization**: Batch normalization was consistently applied after each convolutional layer, continuing to stabilize training and reduce the sensitivity to parameter initialization.
   * **Max-Pooling**: Max-pooling layers were strategically placed after specific convolutional layers (conv1, conv2, conv4) to downsample the feature maps, reducing the computational load while preserving important spatial information.
2. **LSTM Network Optimization**:
   * **Input Dimension Calculation**: The LSTM input size was carefully calculated based on the output dimensions of the convolutional network, ensuring compatibility and efficient processing. This adjustment was critical for maintaining the balance between computational efficiency and model capacity.
   * **Dropout Regularization**: An increased dropout rate (0.4) was applied after the LSTM layer to prevent overfitting, particularly given the model's complexity and the risk of memorizing training data.
3. **Evaluation Metrics**:
   * **Introduction of Precision, Recall, and F1 Score**: In addition to accuracy, precision, recall, and F1 score were introduced as key performance metrics. These metrics provided deeper insights into the model's ability to correctly predict positive actions, especially in scenarios where the distribution of action classes was imbalanced.
   * **Handling Edge Cases**: Care was taken to handle cases where there were no positive predictions or actions, preventing division by zero errors and ensuring the robustness of the metric calculations.
4. **Optimization Strategy**:
   * **Adam Optimizer**: The Adam optimizer was retained for its effectiveness in handling sparse gradients and its adaptive learning rate capabilities. The learning rate and weight decay parameters were fine-tuned based on the specific needs of this architecture.
   * **Loss Function**: The loss was calculated using the log-probability of actions weighted by feedback, which was then averaged across the trajectory. This approach ensured that the model was learning not just to predict actions, but to do so in a way that maximized the task performance.
5. **Metric Calculation During Training**:
   * **Real-time Evaluation**: Metrics were calculated after each parameter update, providing real-time feedback on the model's performance. This enabled more informed decisions during training, allowing for early stopping or further tuning based on the observed trends in accuracy, precision, recall, and F1 score.

**Impact on Performance**

This iteration of the architecture demonstrated improved precision, recall, and overall F1 score, indicating that the model was becoming better at distinguishing between different actions and making correct predictions. The deeper convolutional network, combined with careful LSTM input sizing and enhanced regularization, contributed to a more robust and generalizable model. The introduction of additional evaluation metrics provided a more nuanced understanding of the model's strengths and areas for further improvement, particularly in handling edge cases and imbalanced action distributions.

Here's a brief comparison of the architectures based on the given accuracies and losses:

1. **Architecture 1**:
   * **Accuracy**: 67.91%
   * **Loss**: 5.704
   * **Summary**: This architecture had a moderately high accuracy but a relatively high loss compared to the third architecture.
2. **Architecture 2**:
   * **Accuracy**: 66.52%
   * **Loss**: 8.66862
   * **Summary**: This architecture had the lowest accuracy and the highest loss, indicating it was less effective compared to the others.
3. **Architecture 3**:
   * **Accuracy**: 69.57%
   * **Loss**: 1.34518
   * **Summary**: This architecture achieved the highest accuracy and the lowest loss, suggesting it was the most successful in training.

**6 : Integrating Object Detection Model with CNN Policy**

**Objective:**  
The goal of this experiment is to enhance the robot's ability to detect and manipulate the stator by integrating an object detection model with the existing CNN policy model. This experiment is ongoing and not a confirmed improvement.

**Process Overview:**

1. **Frame Extraction and Labeling:**
   * Video episodes were used to extract frames that contain images of the stator.
   * These frames were manually labeled to create a dataset for training the object detection model.
2. **Detection Model Development:**
   * A detection model was developed to identify and locate the stator within the extracted frames.
   * The model was trained using the manually labeled dataset to accurately detect the stator.
3. **Integration with CNN Policy Model:**
   * The detection model is integrated into the existing CNN policy model.
   * The strategy involves using the detection model to provide the location of the stator as input to the policy model.
   * The CNN policy model then uses this information to guide the robot towards the stator, enabling precise manipulation.

**Current Status:**  
This experiment is in progress, with the detection model being evaluated and refined to ensure accurate stator detection before fully integrating and testing with the CNN policy model.