Mirai1: Future Image Prediction in Simulator

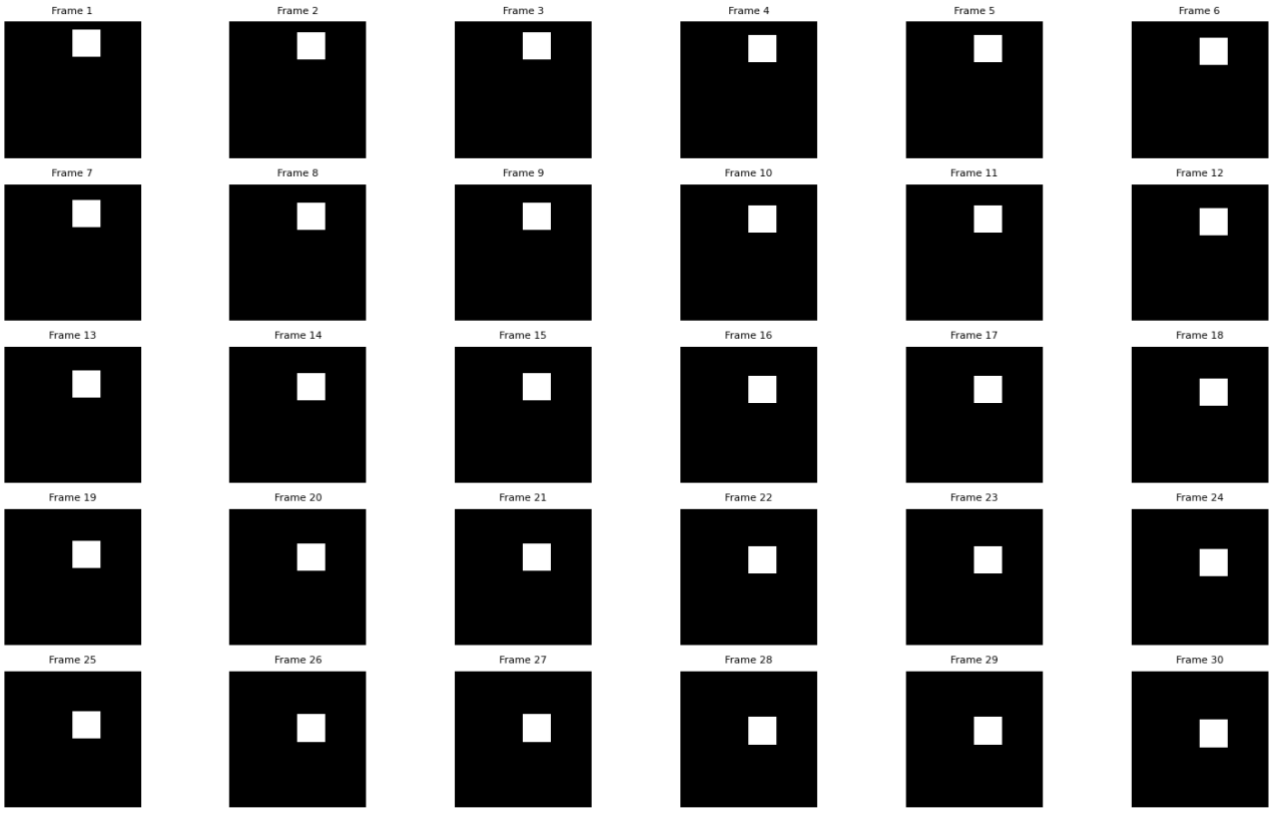
Intelligence:
Mirai1 is designed to **predict both spatial and temporal** information within the simulator.

Objective:
The primary aim is to create an AI agent, named Mirai 1.0, which can consistently **forecast future scenarios by leveraging temporal data** as a loss function. This model seeks to **master the inherent rules present** in a controlled virtual setting.

Significance of Temporal Data:
Human learning, to a large extent, is rooted in the assimilation of **temporal information**. This capability allows us to proficiently anticipate upcoming occurrences. For example, humans can intuitively predict where a falling marble will land. While contemporary computer vision models predominantly concentrate on spatial information, they often neglect the valuable insights temporal data can offer. I posit that a focus on temporal data can pave the way for more accurate and insightful predictive models.

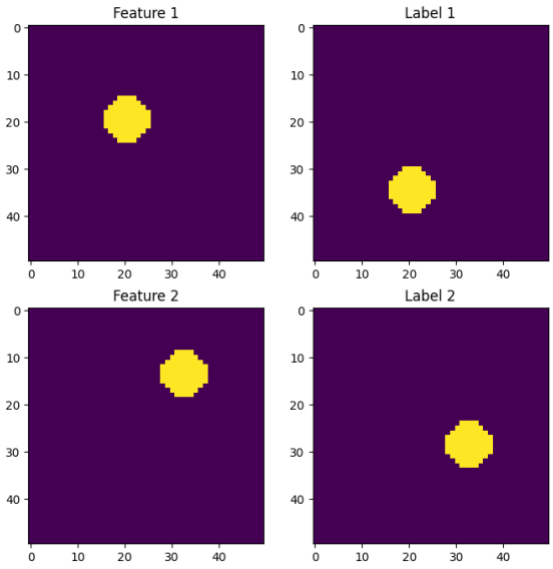
Data Collection and Preparation:
The dataset has been sourced from a 2D 50x50 virtual environment, featuring diverse objects such as squares and circles descending towards the base. This dataset encapsulates consecutive frames from the simulation, captured at a frequency of 30 frames per second.

For example:



Mirai1_0: Focusing on Falling Circles

Dataset Description:
In this variant, the feature image consists of generated circle patterns, while the label image represents the state of the simulator one second (or 30 frames per minute) after the feature image was generated.



1) **Benchmark Model**
    - Sample Size: 100
    - Batch Size: 32

Model Architecture:

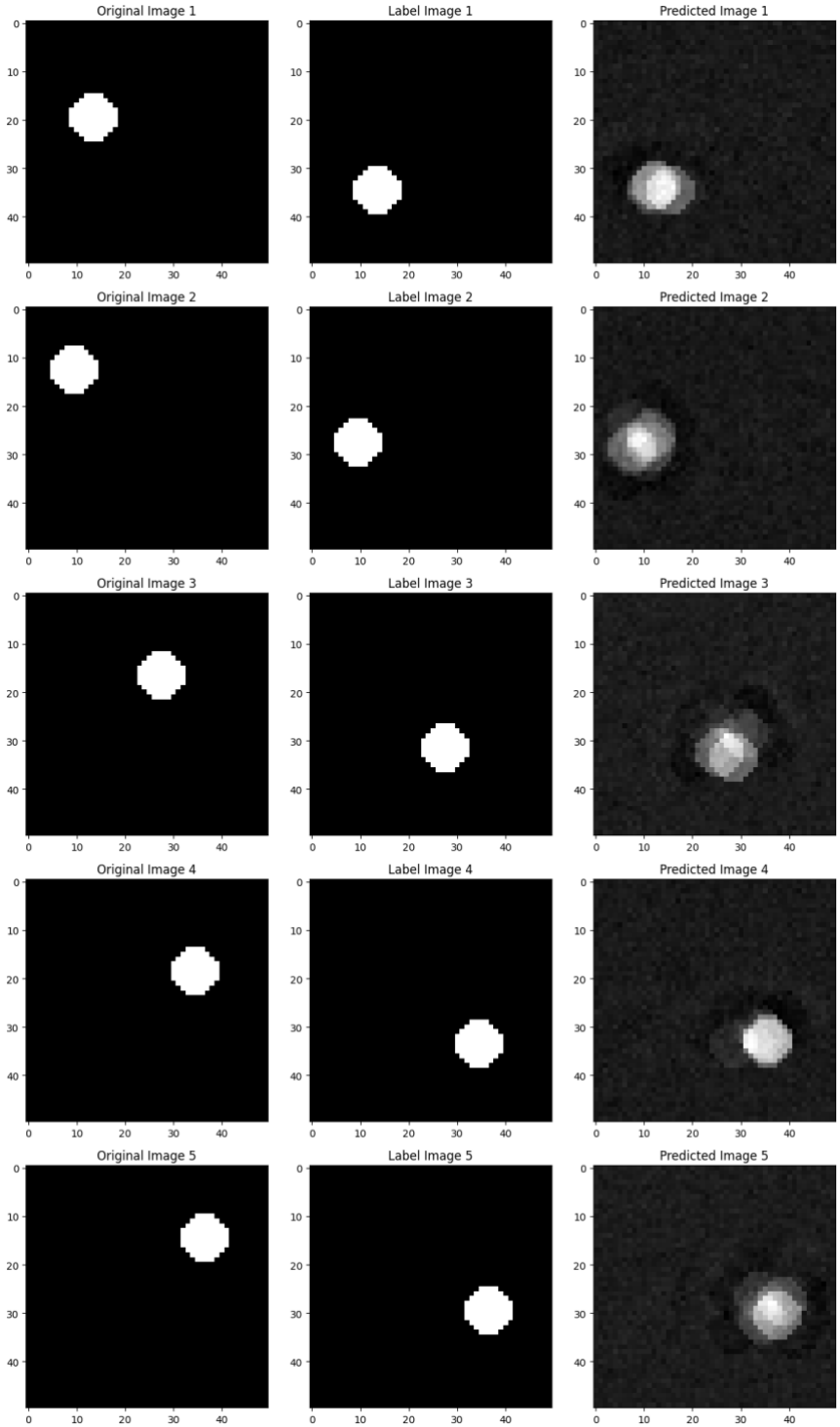
```
class BenchmarkModel(nn.Module):
    def __init__(self):
        super(BenchmarkModel, self).__init__()
        self.fc1 = nn.Linear(50*50, 1000)
        self.fc2 = nn.Linear(1000, 50*50)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x.view(x.size(0), 1, 50, 50)

criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=3)
```

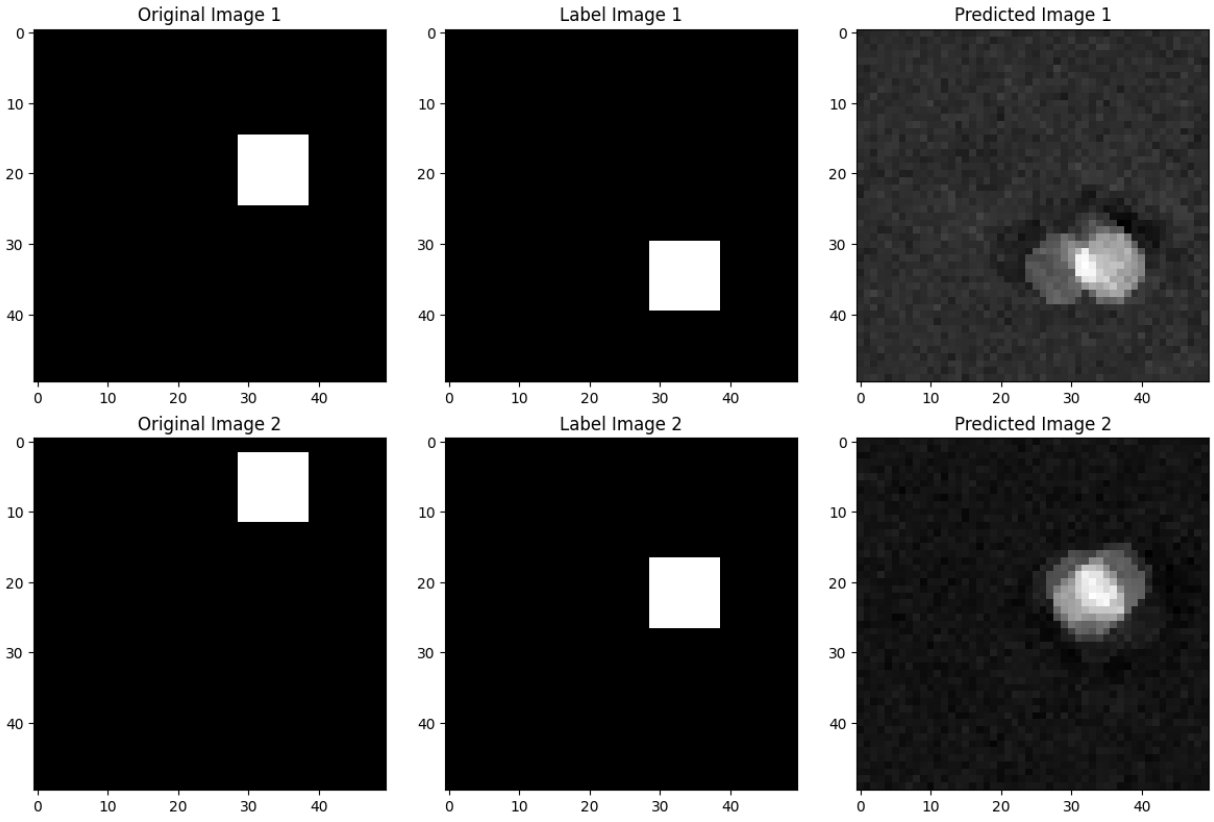
Results:

Epoch	Training Loss	Test Loss
1000	0.0018	0.0070
1500	0.0005	0.0065



Observation:
The model exhibits learning capabilities as evidenced by the decreasing loss values over the epochs. However, predictions appear to be **somewhat vague**. Introducing additional hidden layers may enhance the model's precision and capacity to capture intricate patterns within the data.

Observation on Unseen Data (Squares):
When tested on data featuring squares (a shape the model was not trained on), the model demonstrates the capability to accurately predict the location of the shape within the simulation. However, it struggles to recognize and replicate the specific shape, which in this case is a square.



2) Enhanced Model with Increased Training Data

- Sample Size: 200
- Batch Size: 8

Model Architecture: Unchanged from the previous Benchmark Model.

Results:

Epoch	Training Loss	Test Loss
1000	0.0002	0.0049

Observation:
With a larger dataset for training, the model showcases improved performance as evidenced by the reduced loss values. The decrease in both training and test loss suggests that the model benefits from additional data, enhancing its learning capacity and predictive accuracy.

3) Deeper Model with Additional Hidden Layer

- Sample Size: 100
- Batch Size: 32

Model Architecture:

```
class DeepBenchmarkModel(nn.Module):
    def __init__(self):
        super(DeepBenchmarkModel, self).__init__()
        self.fc1 = nn.Linear(50*50, 1000)
        self.fc2 = nn.Linear(1000, 1000)
        self.fc3 = nn.Linear(1000, 50*50)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x.view(x.size(0), 1, 50, 50
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=3)
```

Results:

Epoch	Training Loss	Test Loss
1000	0.0008	0.0066

Observation:
By introducing an additional hidden layer, the model has become deeper, aiming to capture more intricate patterns in the data. The results show a slightly higher training loss compared to the previous model with more data, but it's crucial to observe how the model behaves on different types of data or when further tuned.

4) **Deeper Model with Increased Training Data**
   - Sample Size: 200
   - Batch Size: 8

Model Architecture: Unchanged from the previous DeepBenchmarkModel.

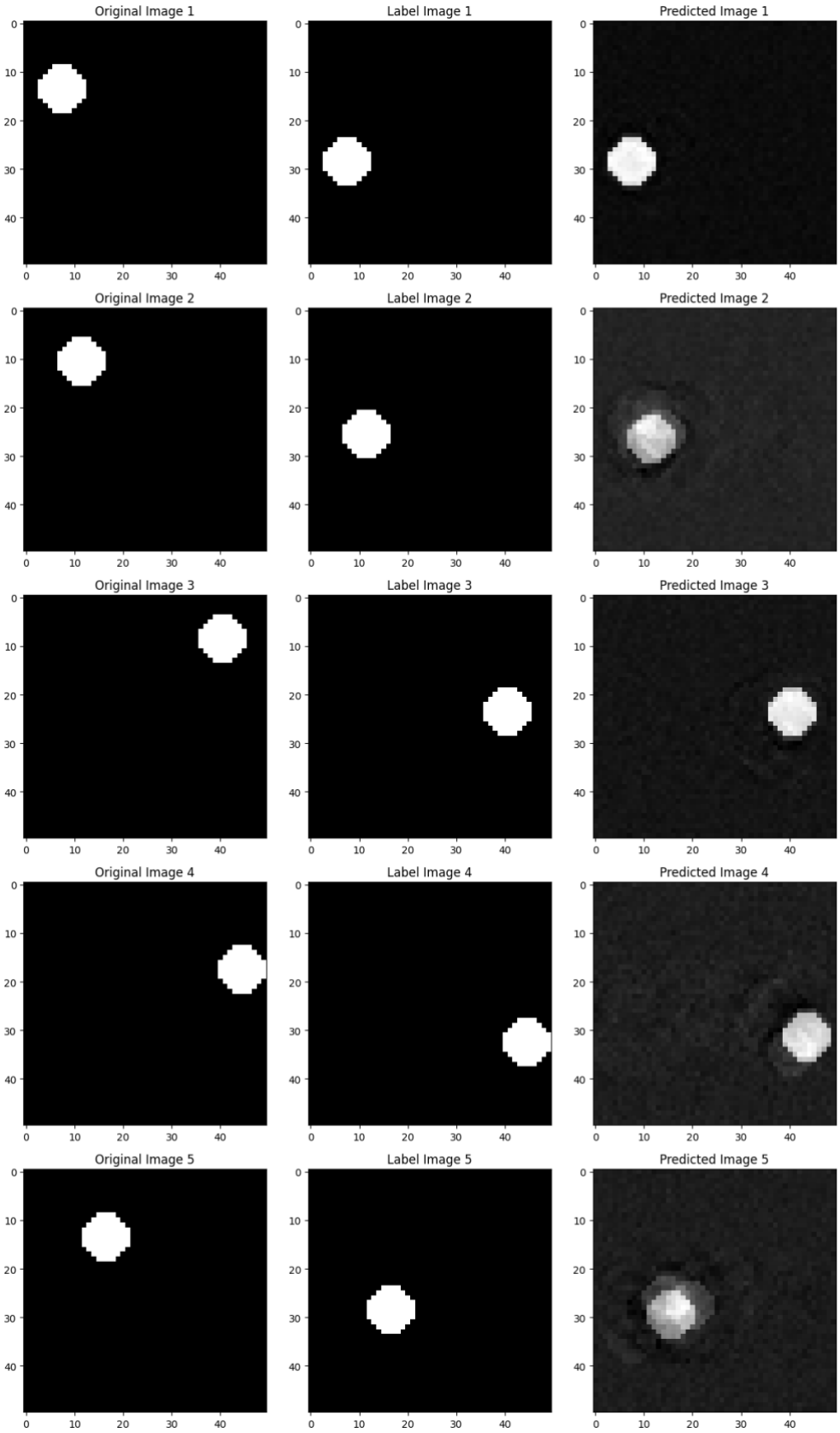
Results:

Epoch	Training Loss	Test Loss
1000	0.0005	0.0047

Training Duration: 194.19 seconds

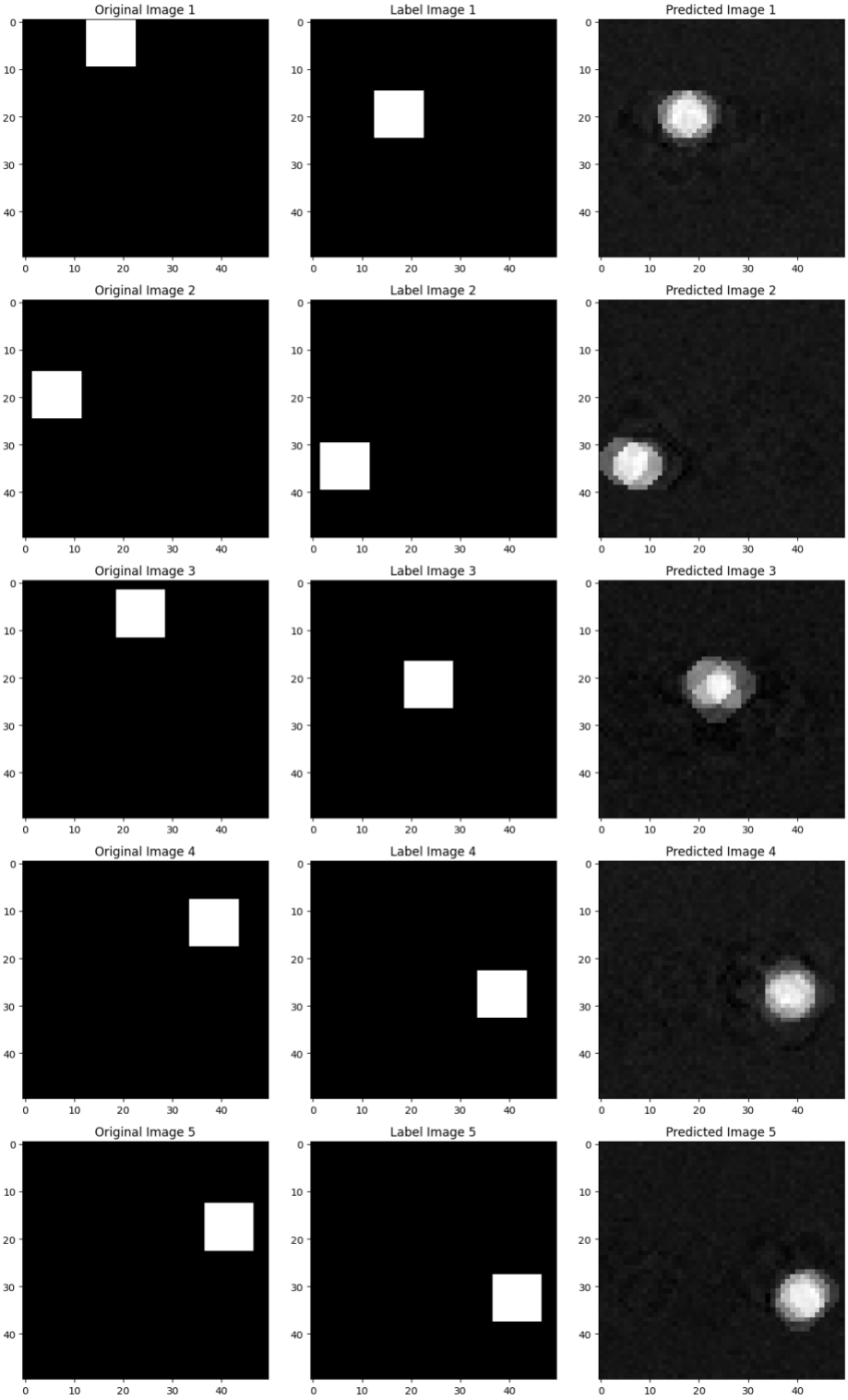
Observation:
By providing the deeper model with a larger dataset, the performance further improved, reaching a minimum test loss of 0.0047. The decrease in both training and test loss suggests that the deeper architecture, when coupled with more training data, is capable of better generalization and prediction accuracy.

Predictions:



Observation on Unseen Data (Squares):

The enhanced model's capability to accurately predict the location of shapes, even for square images which it wasn't specifically trained on, indicates robust generalization. The accurate location prediction for both circles and squares underlines the model's versatility and its potential to be applied to a broader range of scenarios or shapes in the future.



5) **Enhanced Model with Regularization Techniques**
   - Sample Size: 200
   - Batch Size: 8

Model Architecture:

```
class EnhancedBenchmarkModel(nn.Module):
    def __init__(self):
        super(EnhancedBenchmarkModel, self).__init__()
        self.fc1 = nn.Linear(50*50, 1000)
        self.dropout1 = nn.Dropout(0.2)  # 20% dropout
        self.fc2 = nn.Linear(1000, 1000)
        self.dropout2 = nn.Dropout(0.2)  # 20% dropout
        self.fc3 = nn.Linear(1000, 50*50)

        # He Initialization
        nn.init.kaiming_normal_(self.fc1.weight, nonlinearity='relu')
        nn.init.kaiming_normal_(self.fc2.weight, nonlinearity='relu')
        nn.init.kaiming_normal_(self.fc3.weight, nonlinearity='relu')

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.dropout1(x)
        x = F.relu(self.fc2(x))
        x = self.dropout2(x)
        x = self.fc3(x)
        return x.view(x.size(0), 1, 50, 50)
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=3, weight_decay=1e-4,
momentum=0.9)
```

Results:

Epoch	Training Loss	Test Loss
1000	0.0046	0.0064

Training Duration: 256 seconds

Observation:
The results indicate a balance between the training and test loss, suggesting the model is learning without being overly specific to the training data. However, the test loss is higher compared to previous results.

6) **Model Without Dropouts**
   - Sample Size: 200
   - Batch Size: 8

Model: Same as the EnhancedBenchmarkModel but without the dropout layers.

Results:

Epoch	Training Loss	Test Loss
1000	0.0036	0.0061

Training Duration: 240 seconds

7) **Model Without Weight Decay**
   - Sample Size: 200
   - Batch Size: 8

Model: Same as the EnhancedBenchmarkModel but without weight decay in the optimizer.

Results:

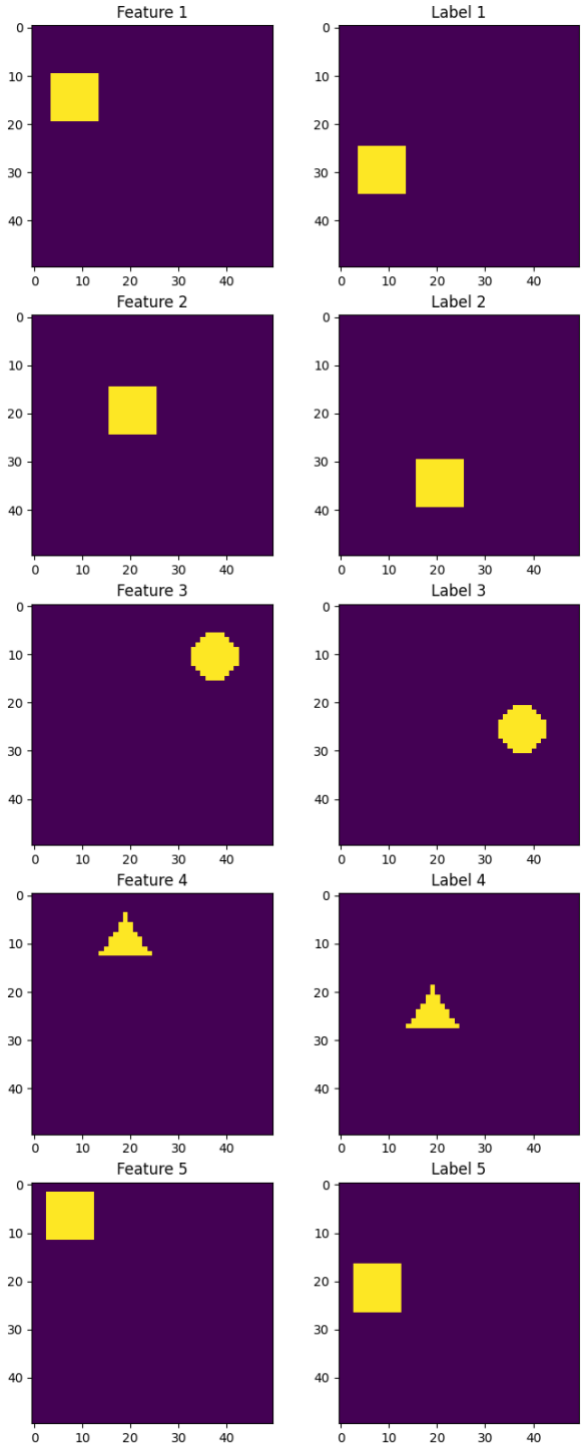
Epoch	Training Loss	Test Loss
1000	0.0001	0.0059

Training Duration: 220 seconds

Observation:
From the results, it's evident that removing regularization techniques, such as dropouts and weight decay, has led to a slight improvement in the test loss. This indicates that, for this specific dataset and model architecture, these regularization methods may not be necessary and might even be counterproductive.

Mirai1_1: Multi-Shape Prediction

Dataset Description:
The simulator now features three shapes: circles, squares, and triangles. Mirai1_1's task is to accurately predict their future positions and remember the shape present in the original image. The introduction of multiple shapes amplifies the dataset's complexity, necessitating more sophisticated model training and optimization techniques.



1) **Model Adjusted for Additional Data Complexity**
   - Sample Size: 200
   - Batch Size: 8

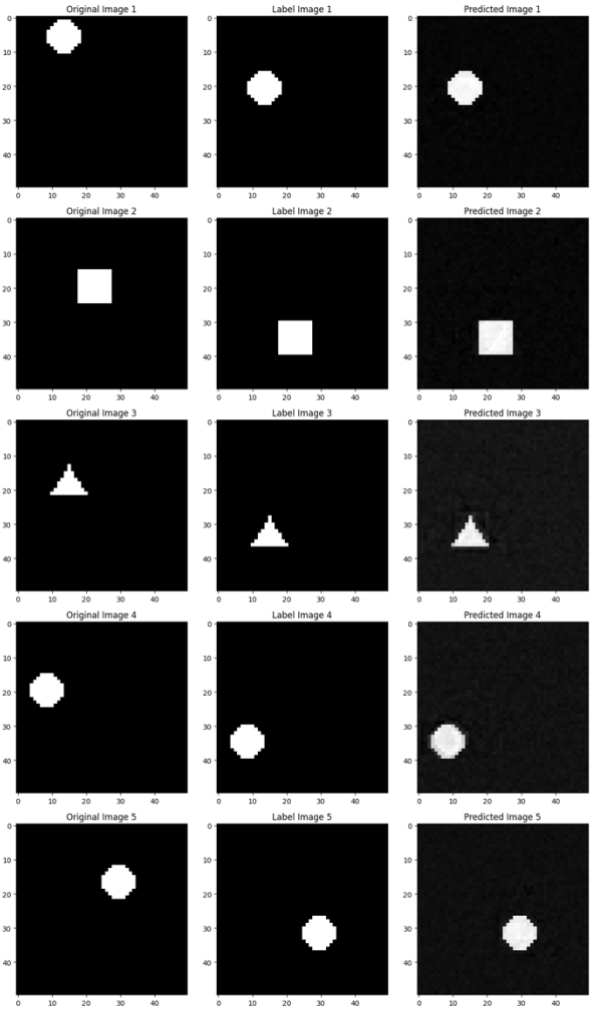
Model Architecture: Unchanged from the previous DeepBenchmarkModel.

Results:

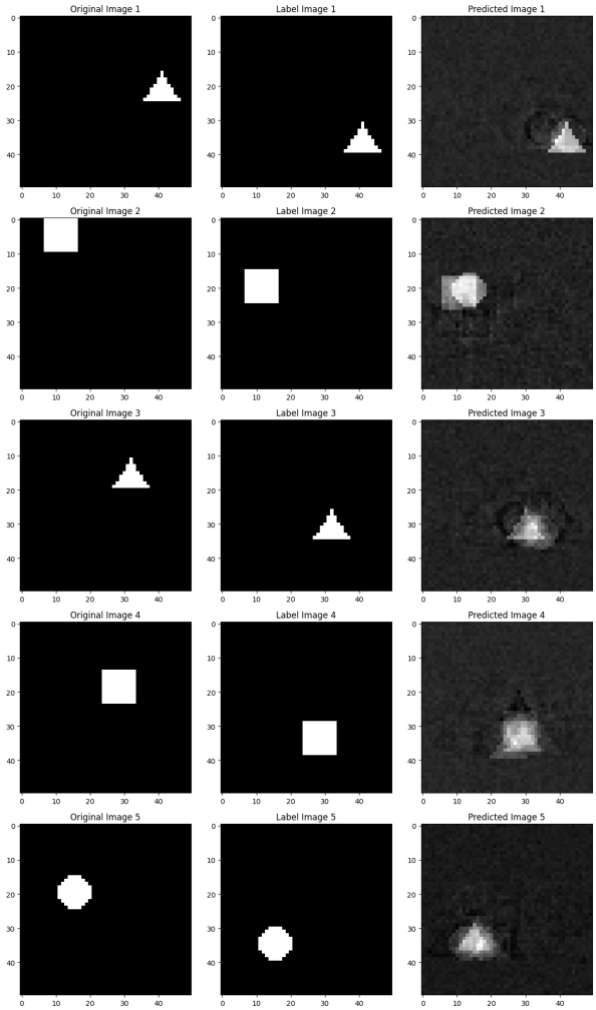
Epoch	Training Loss	Test Loss
1000	0.0004	0.0057

Observation:
Considering the augmented data complexity with the introduction of multiple shapes, the DeepBenchmarkModel demonstrates good performance. The low training and test loss values suggest that the model is effectively learning to predict the future positions of circles, squares, and triangles within the simulator.

Training Data



Testing Data



2) **Model with Increased Data**
   - Sample Size: 400
   - Batch Size: 8

Model Architecture: Unchanged from the previous DeepBenchmarkModel.

Results:

Epoch	Training Loss	Test Loss
1000	0.0006	0.0041

Observation:
By doubling the sample size, the model achieves a significant reduction in the test loss, indicating better generalization on unseen data.

3) **Further Data Augmentation**
   - Sample Size: 800
   - Batch Size: 8

Model Architecture: Unchanged from the previous DeepBenchmarkModel.

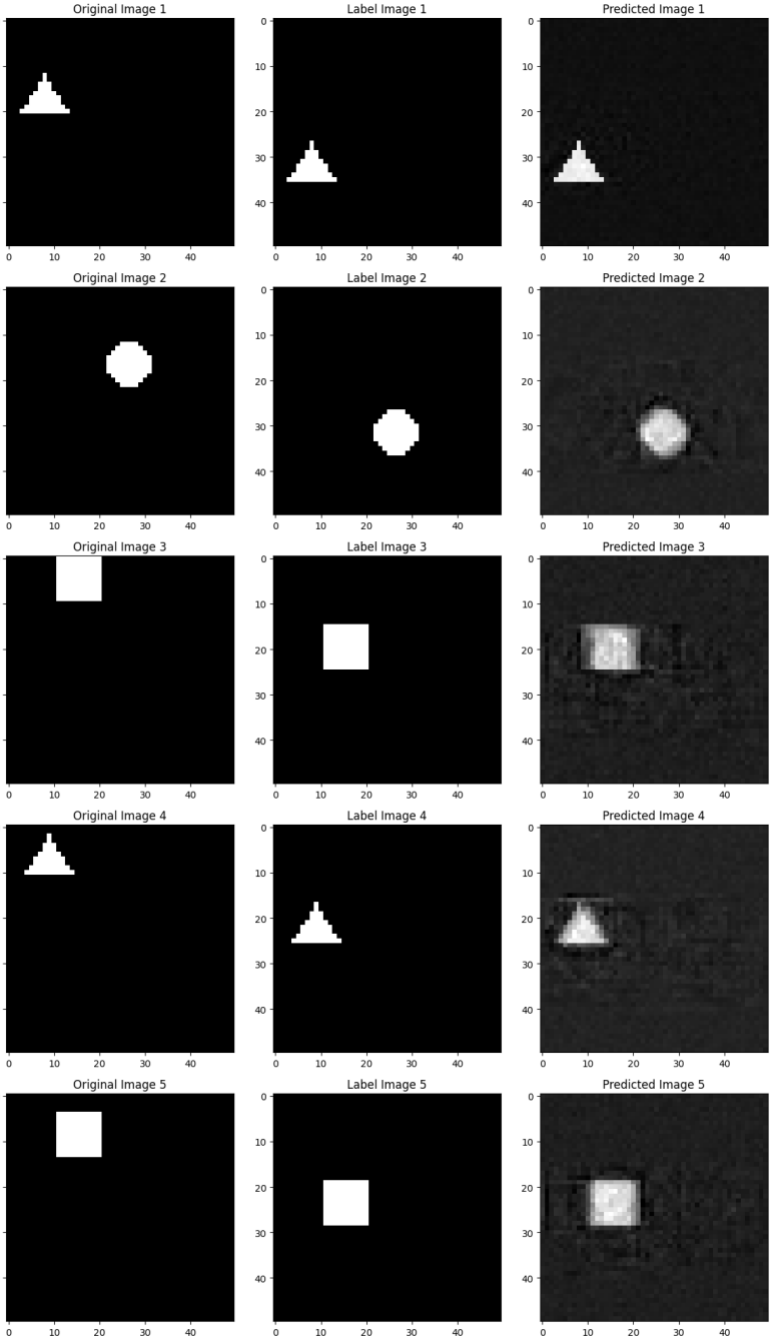
Results:

Epoch	Training Loss	Test Loss
1000	0.0006	0.0026

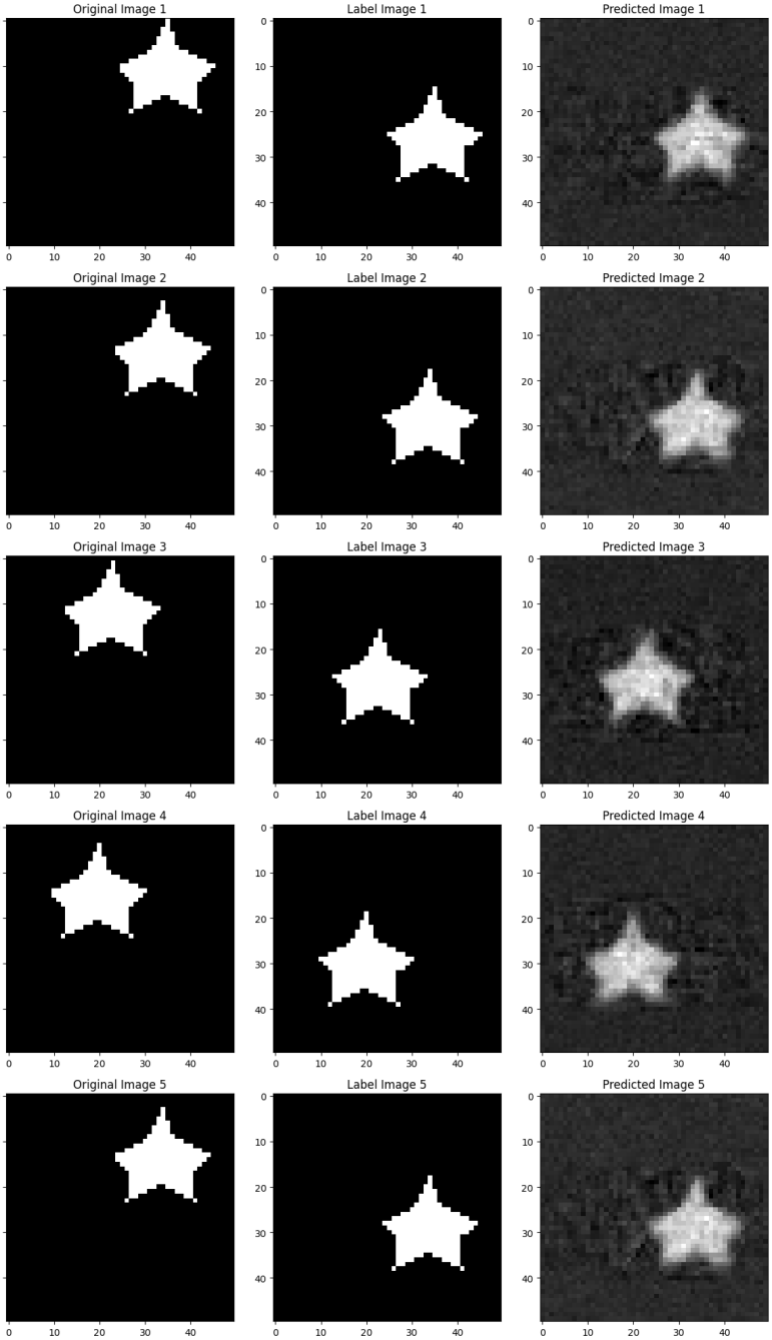
Training Duration: 783.39 seconds

Observation:
With the sample size further increased to 800, the test loss sees an even more substantial reduction. This suggests that the model greatly benefits from additional training data, achieving better accuracy and predictive capabilities. However, the trade-off is an increase in training time, as observed from the reported duration.

Prediction on Test Data

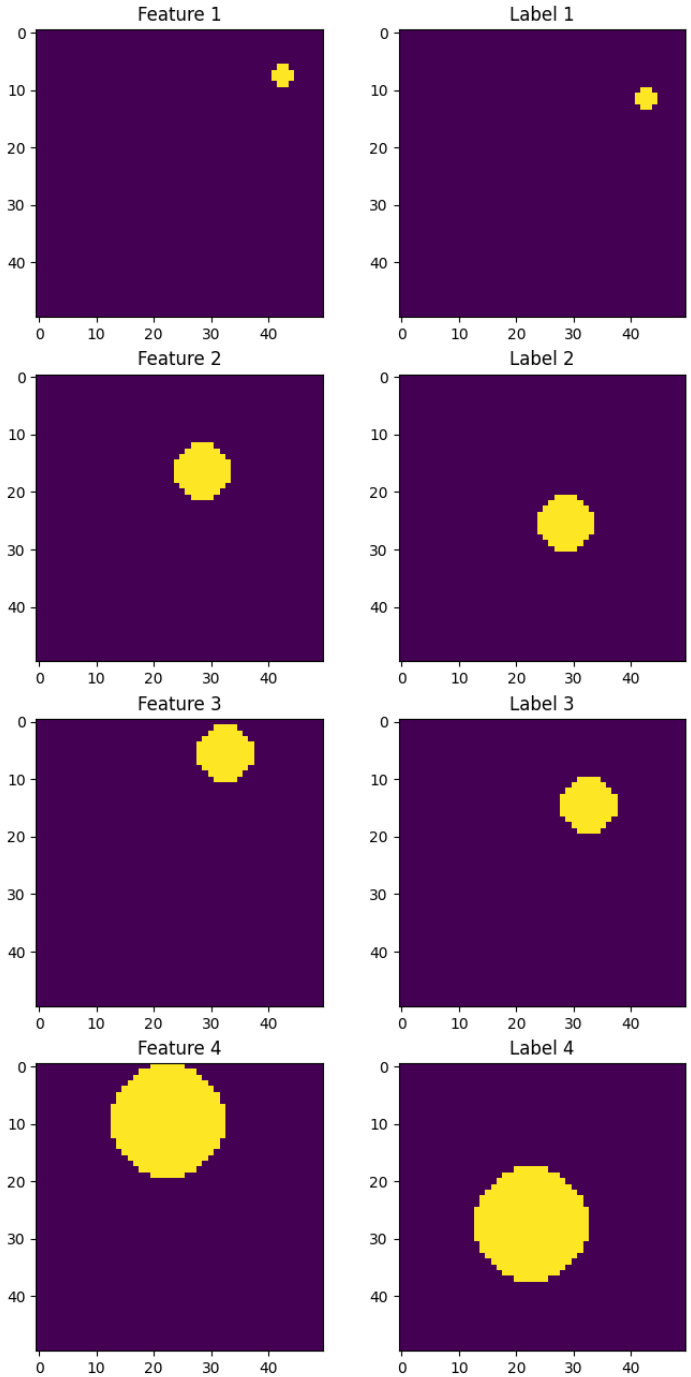


Prediction on Unseen Data



Mirai1_2: Velocity-Based Circle Prediction

Dataset:
Mirai1_2 predicts the positions of circles with varying sizes and velocities in the simulator: smaller circles fall faster, while larger ones descend slowly. The model must understand the inverse size-velocity relationship and adjust predictions, accordingly, adding a layer of complexity to training**.**



1) **Model with Increased Data for Velocity-Based Prediction**
   - Sample Size: 800
   - Batch Size: 8

Model Architecture: Unchanged from the previous DeepBenchmarkModel.

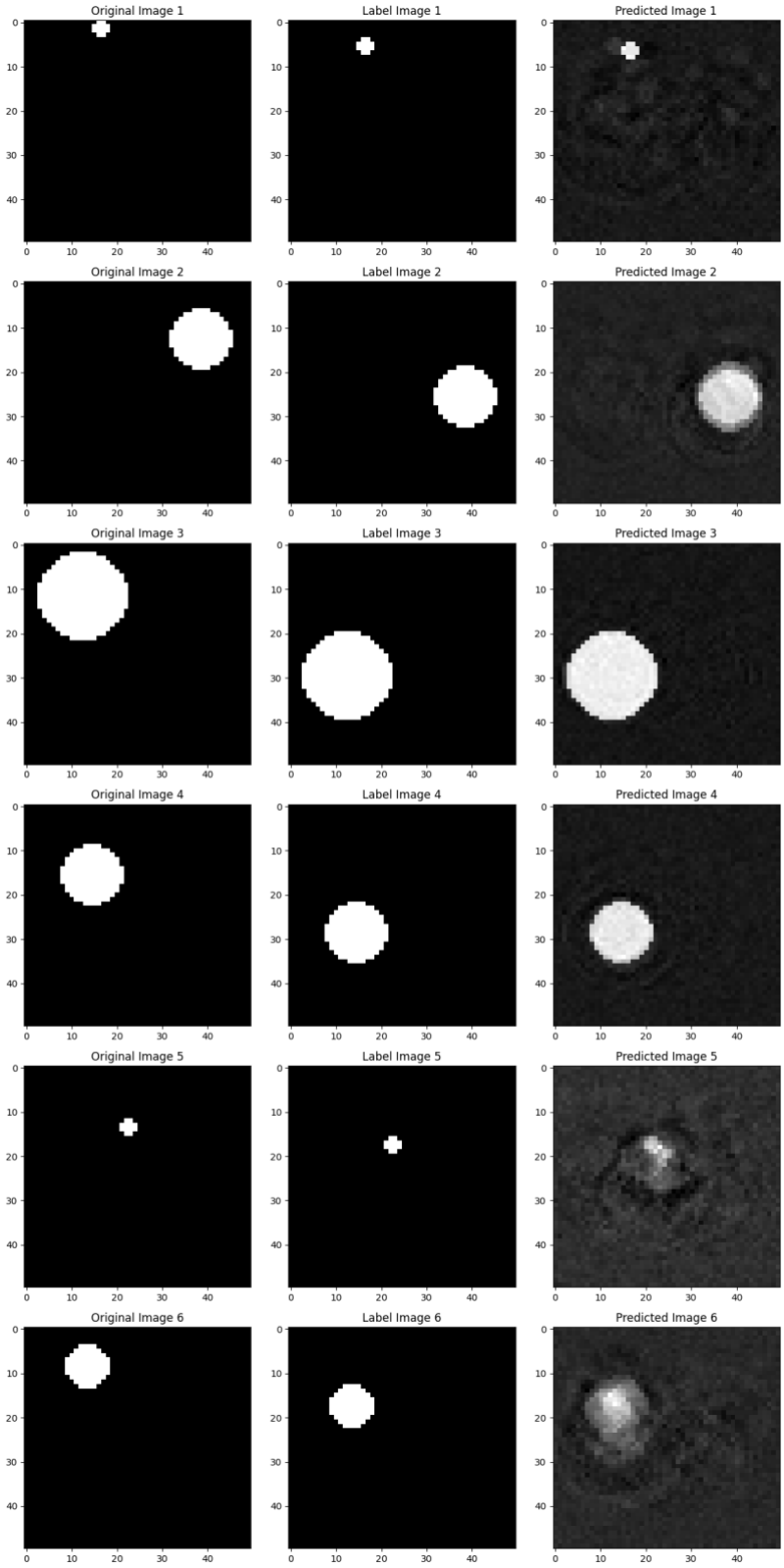
Results:

Epoch	Training Loss	Test Loss
1000	0.0007	0.0051

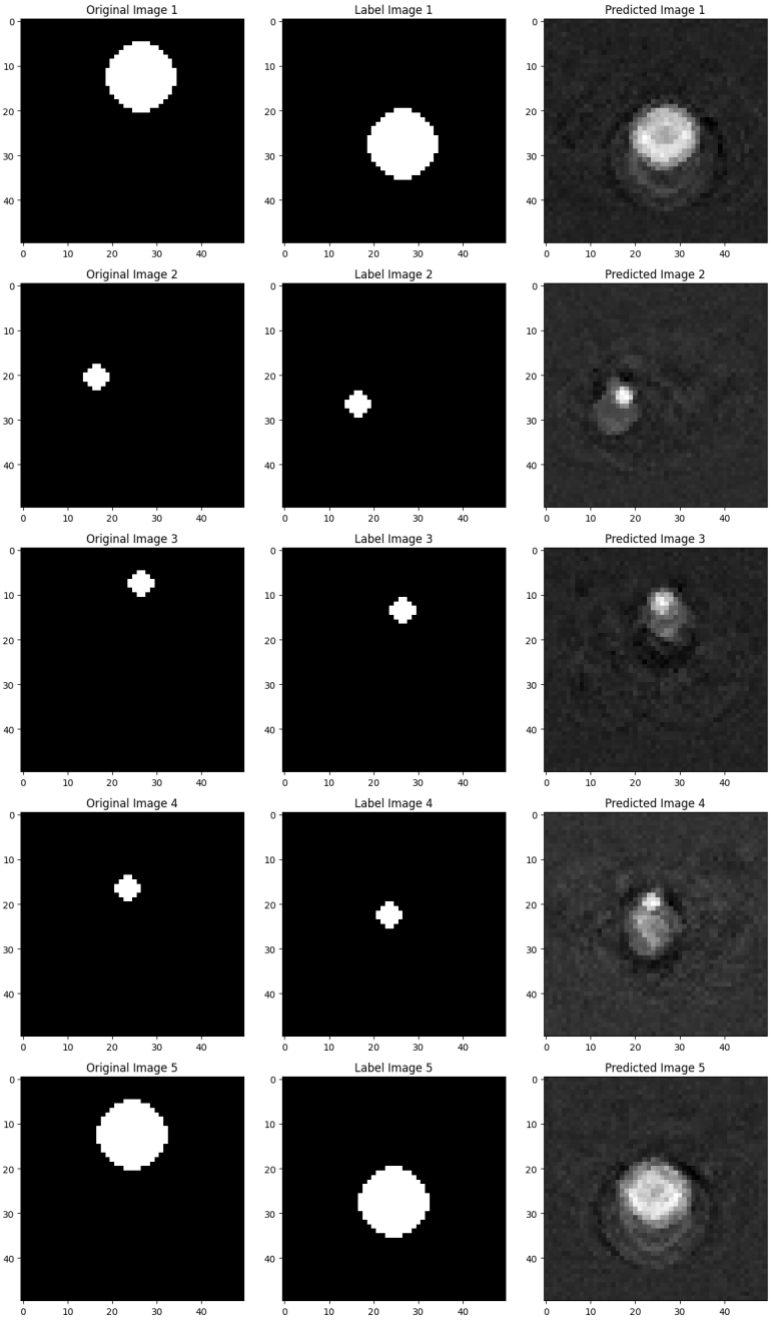
Observation:
With an increased sample size tailored for velocity-based circle predictions, the DeepBenchmarkModel displays promising performance. The relatively low training and test loss values indicate that the model is effectively learning the inverse relationship between circle size and falling velocity, successfully predicting future positions. The model demonstrates proficient accuracy in predicting future positions of medium to large circles. However, its predictions for smaller circles are notably less precise.

Unseen Circle Sizes: When tested on circle sizes that the model has never seen during training, it struggles to predict their locations accurately. The model likely defaults to treating them as the closest known sizes from the training data, leading to off-mark predictions.

Prediction on Test Data



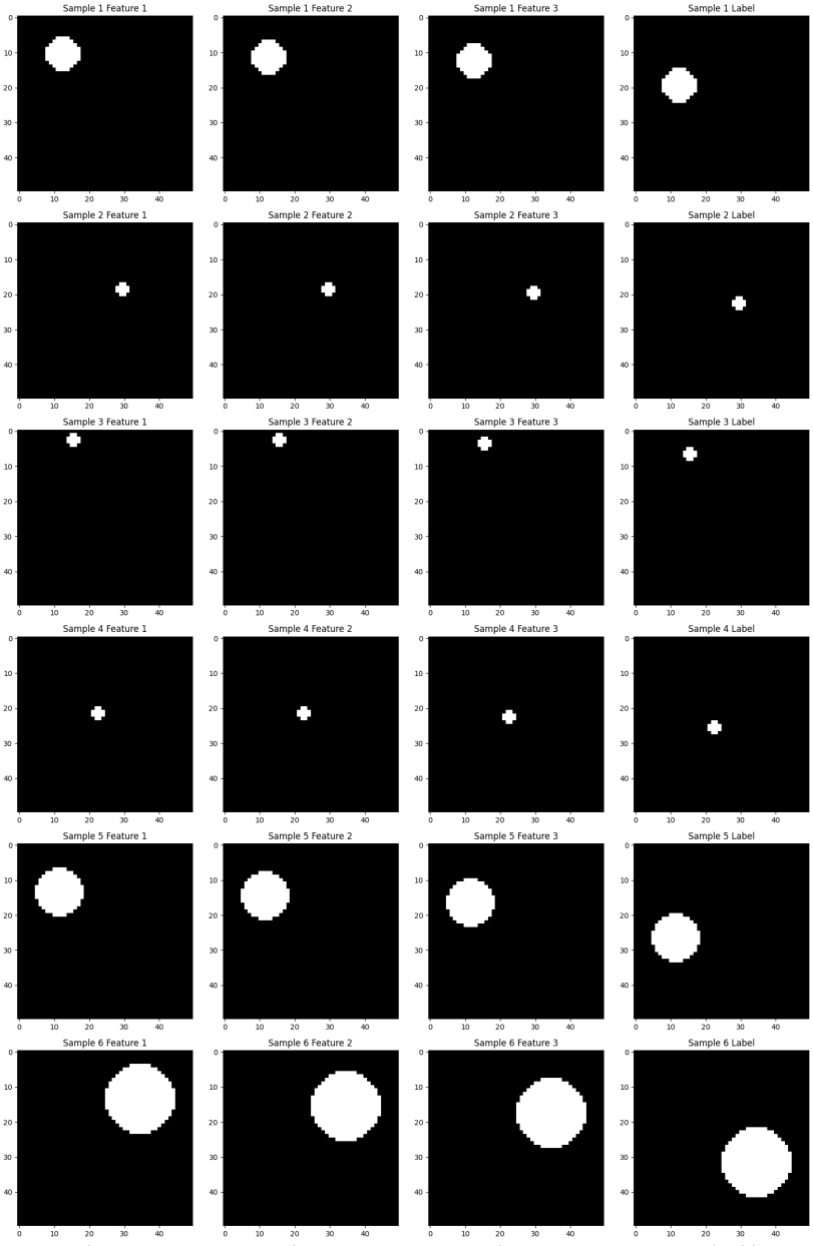
Prediction on Unseen Data



Mirai1_3: Sequential Information-Based Circle Prediction

Dataset Description:
For Mirai1_3, the task remains the prediction of future positions of four different-sized circles. However, the input has been augmented to include not just a single initial image but a set of three sequential images. This change allows the model to grasp the movement pattern over a short duration, which could potentially enhance its predictive accuracy.

Challenges:
Incorporating sequential data means the model now has to process temporal patterns in addition to spatial ones. This added dimension of time can introduce complexities in training, but it also provides the model with richer information to make predictions.



1) **Model Incorporating Sequential Data Using CNN + RNN**
   - Sample Size: 800
   - Batch Size: 8

Model Architecture: CompactModel (CNN + LSTM)
The model harnesses the power of Convolutional Neural Networks (CNN) to capture spatial features and Long Short-Term Memory networks (LSTM) to handle temporal or sequential information.

```
class CompactModel(nn.Module):
    def __init__(self):
        super(CompactModel, self).__init__()

        # CNN Layers
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        # LSTM Layer
        self.lstm = nn.LSTM(25*25*32, 128, 1, batch_first=True)

        # Fully Connected Layer
        self.fc = nn.Linear(128, 50*50)

    def forward(self, x):
        batch_size, sequence_length, channels, height, width = x.size()
        x = x.view(batch_size * sequence_length, channels, height, width)

        # CNN Layers
        x = F.relu(self.conv1(x))
        x = self.pool(x)

        # Prepare for LSTM
        x = x.view(batch_size, sequence_length, -1)

        # LSTM
        lstm_out, _ = self.lstm(x)

        # Use the last LSTM output for the fully connected layer
        x = lstm_out[:, -1, :]
        x = self.fc(x)

        return x.view(batch_size, 1, 50, 50)
```

Results:

Epoch	Training Loss	Test Loss
1000	0.0041	0.0067

Training Duration: 2057.15 seconds (approx. 34 minutes)

Observation:
The model, by leveraging both CNN and LSTM layers, seeks to effectively handle the spatial features of the circles and their sequential movement. The training time has significantly increased due to the combined complexities of CNN and RNN layers. While the losses indicate the model's learning capability, further refinements might be needed to optimize its performance, especially given the nuanced challenge of predicting based on sequential data.

2) **Model Incorporating Sequential Data Using FC + RNN**
   - Sample Size: 800
   - Batch Size: 8

Model Architecture: CompactModel (FC + LSTM)
The model combines Fully Connected (FC) layers to capture spatial features and Long Short-Term Memory Networks (LSTM, a type of RNN) to handle the temporal or sequential information.

Results:

Epoch	Training Loss	Test Loss
1000	0.0036	0.0061

Observation:
By using a combination of FC layers and LSTM, the model is designed to handle both the spatial representations of the circles and their sequential movement patterns. The observed losses indicate that the model has learned well from the sequential data, showcasing the potential of combining traditional neural networks with recurrent layers for such tasks.

```
class CompactModel(nn.Module):
    def __init__(self):
        super(CompactModel, self).__init__()

        # Fully Connected Layers
        self.fc1 = nn.Linear(50*50, 1000)
        self.fc2 = nn.Linear(1000, 1000)

        # LSTM Layer
        self.lstm = nn.LSTM(1000, 128, 1, batch_first=True)

        # Fully Connected Layer for output
        self.fc3 = nn.Linear(128, 50*50)

    def forward(self, x):
        batch_size, sequence_length, channels, height, width = x.size()
        x = x.view(batch_size * sequence_length, channels * height *
width)

        # Fully Connected Layers
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))

        # Prepare for LSTM
        x = x.view(batch_size, sequence_length, -1)

        # LSTM
        lstm_out, _ = self.lstm(x)

        # Use the last LSTM output for the fully connected layer
        x = lstm_out[:, -1, :]
        x = self.fc3(x)

        return x.view(batch_size, 1, 50, 50)
```

3) Model with Increased Data Using FC + RNN

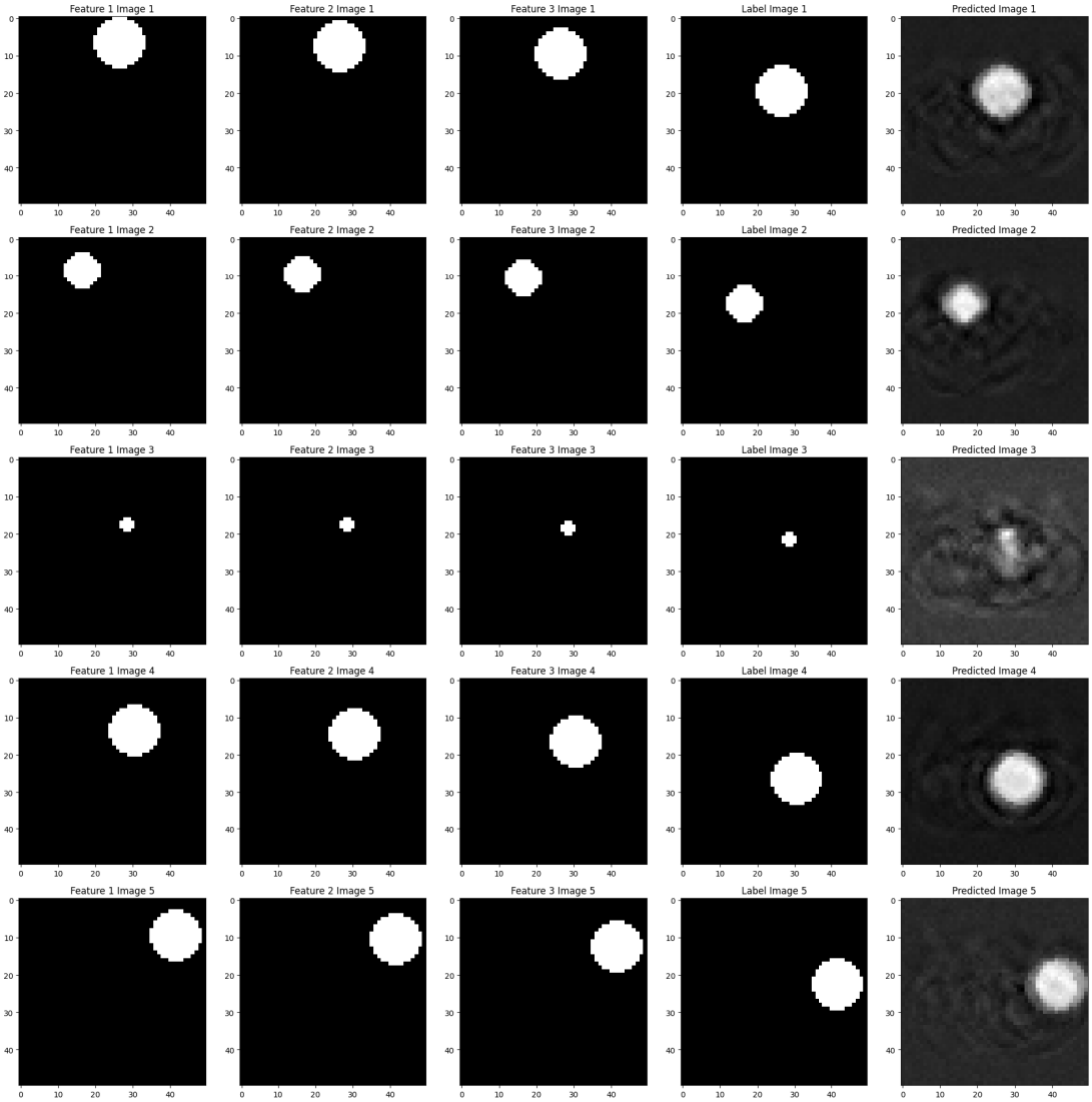
- Sample Size: 1600
- Batch Size: 8

Model Architecture: Unchanged from the previous CompactModel (FC + LSTM)
Results:

Epoch	Training Loss	Test Loss
1000	0.0034	0.0047

Training Duration: 2673.57 seconds (approx. 44 minutes)

Observation:
By doubling the sample size, the model achieves a notable improvement in test loss, indicating better generalization on unseen data. This suggests that the model benefits from having more data, particularly when handling the challenges of predicting based on sequential information. The increased training time is a trade-off for the enhanced predictive performance achieved with the larger dataset.



Benefits of the Study:

1. **Improved Accuracy with More Data:**
   As the dataset size was increased across iterations, the model's accuracy showed consistent improvement. This underscores the importance of having a robust dataset for training deep learning models.

2. **Generalization Capabilities:**
   Training on three distinct shapes (circles, squares, and triangles) enabled the model to generalize well on unseen shapes. This highlights the model's ability to not just memorize but also extrapolate from the training data.

3. **Architecture Evaluation:**
   Throughout the study, different model architectures were compared, such as CNN + RNN and FC + RNN combinations. This evaluation was crucial in identifying the most effective architecture for predicting future states in the simulator.

4. **Data Importance Analysis:**
   The study showcased the significance of various types of data, such as shape, size, and sequential information, in making accurate predictions. This analysis aids in understanding which data facets are most influential in the prediction task.

5. **Feasibility of Iterative Modeling:**
   The simulator's controlled environment, with relatively low complexity compared to real-world scenarios, made it feasible to iteratively experiment with and refine different models. This iterative approach is essential for achieving optimal performance.

The study offers valuable insights into the intricacies of model training, architecture selection, and data importance, providing a blueprint for similar predictive tasks in more complex environments.