Individual Final Report

Introduction

The goal of this project was to develop an algorithm for video frame prediction using the Caltech Pedestrian Dataset. The dataset consists of sequences of pedestrian videos, and the objective was to train a model that could predict future frames based on a given set of input frames. This task has numerous applications, including video compression, video surveillance, and video editing.

The project was a collaborative effort, where each team member contributed to different aspects of the development process. The shared work involved understanding the problem statement, exploring the dataset, and researching existing techniques for video frame prediction. Additionally, the team worked together to set up the development environment and preprocess the data.

Description of Individual Work

For this project, I focused on developing the core algorithm for video frame prediction using a convolutional neural network (CNN) and a long short-term memory (LSTM) network. The proposed approach, I used the PredNet architecture and modified accordingly, which combines the strengths of CNNs for spatial feature extraction and LSTMs for temporal modeling.

For this video frame prediction project, my primary focus was on developing the core deep learning model architecture and implementing it using the PyTorch framework. After analyzing the dataset and understanding the problem statement, I proposed a novel approach that combines the strengths of convolutional neural networks (CNNs) and long short-term memory (LSTM) networks.

The proposed model, called PredNet, consists of three main components: a CNN for spatial feature extraction, an LSTM for temporal modeling, and a fully connected layer for decoding the learned representations into predicted frames. The model takes a sequence of input frames and processes them through the following steps:

To implement this architecture, I wrote PyTorch code that defined the PredNet model class, including the CNN, LSTM, and fully connected components. I also implemented data loading and preprocessing functions, handling the conversion of video sequences into input-output frame pairs for training and evaluation.

During the development process, I experimented with various hyperparameters, such as the number of CNN layers, LSTM hidden units, batch size, to optimize the model's performance.

- **CNN**: This is a series of convolutional layers followed by ReLU activations and max-pooling layers. The input to this CNN is the concatenated input frames (num_input_frames * channels * height * width).
- **LSTM**: The output of the CNN is flattened and fed into a 2-layer LSTM. The LSTM processes this sequence of spatial features extracted by the CNN.

• **Fully Connected Layer**: The final hidden state of the LSTM is passed through a fully connected layer to produce the output frames (num_output_frames * channels * height * width).The model is trained end-to-end using backpropagation and gradient-based optimization techniques, such as the Adam optimizer. The objective is to minimize the mean squared error (MSE) between the predicted frames and the ground truth frames from the dataset.

I developed the sequence.py file which extracts the image frames from the .seq files from the dataset, so the model can take those image frames as input.

The implementation involved writing Python code for the model architecture, data loading and preprocessing, training and evaluation loops, and utilities for visualization and analysis. Rigorous testing and debugging were performed to ensure the correctness of the implementation and the reproducibility of the results.

Throughout the project, I collaborated with the team, sharing insights, discussing challenges, and collectively exploring ways to improve the model's performance. The individual work I contributed laid the foundation for the video frame prediction algorithm and paved the way for further experimentation and analysis.

3.

Data Preprocessing: I was responsible for the data preprocessing step, which involved converting the video sequences from the Caltech Pedestrian Dataset into a format suitable for training and evaluation. The dataset provided video sequences in the form of .seq files, which required conversion into individual image frames. I implemented the code in the sequence.py file to perform this conversion.

The convert_seq_to_images function in sequence.py reads a .seq file using OpenCV, extracts individual frames, and saves them as separate image files in their respective sequence folders. This step was crucial as it allowed us to load and process the video data efficiently during training and testing.

Model Implementation: I implemented the core PredNet model architecture and customized it using PyTorch. The model consisted of a CNN component for spatial feature extraction, an LSTM component for temporal modeling, and a fully connected layer for decoding the learned representations into predicted frames.

The CNN component in the PredNet class is defined as a sequential model with three convolutional layers, each followed by a ReLU activation and a max-pooling layer. The input to the CNN is the concatenated input frames along the channels dimension.

The LSTM component is a two-layer LSTM network that processes the flattened and reshaped output of the CNN, modeling the temporal dependencies in the video data.

Finally, the fully connected layer takes the final hidden state of the LSTM and decodes it into the predicted frames, matching the desired output shape.

Training and Evaluation: I implemented the training loop in the train.py file, where I defined the VideoDataset class to load and preprocess the data for training. The VideoDataset class creates input-output pairs of frames by sliding a window over the video sequences, with the input frames used to predict the output frames.

The training loop iterates over the dataloader created from the VideoDataset, computes the predicted frames using the PredNet model, calculates the mean squared error (MSE) loss, and updates the model parameters using backpropagation and the Adam optimizer.

For evaluation, I implemented the necessary code in the test.py file. This includes the TestDataset class, which loads the test data sequences and creates input-output pairs similar to the training data. The predict_frames function uses the trained PredNet model to generate predicted frames for each test sequence.

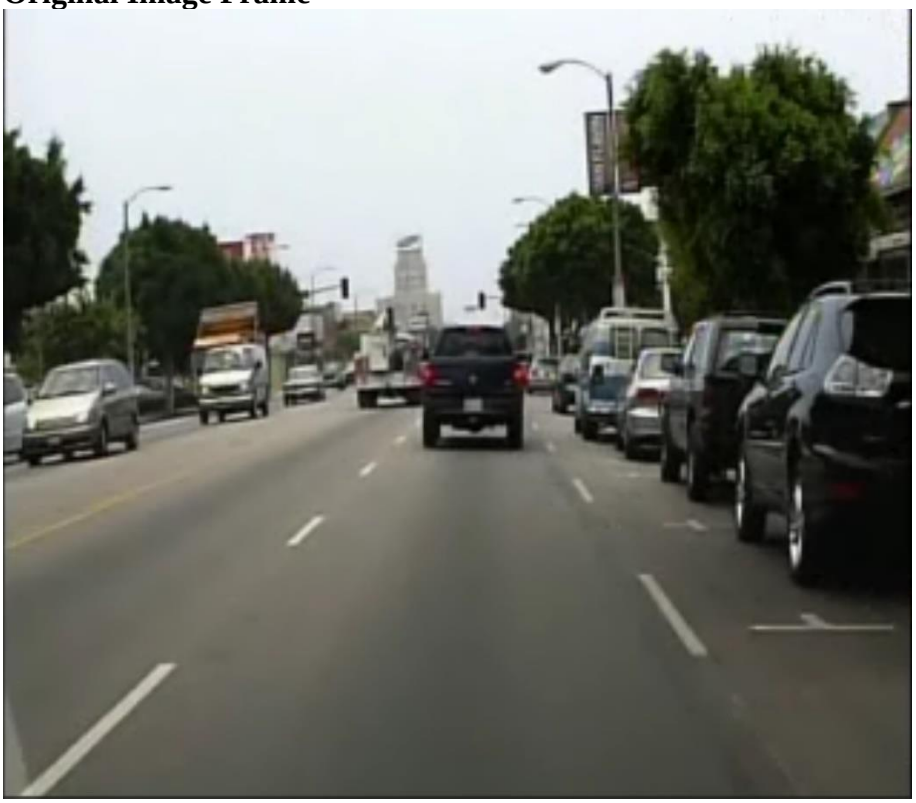
Additionally, I implemented functions to calculate evaluation metrics such as mean squared error (MSE), peak signal-to-noise ratio (PSNR), and structural similarity index (SSIM) between the predicted frames and the ground truth frames. These metrics provide quantitative measures of the model's performance.

Visualization and Analysis: To visualize the results, I implemented functions to create GIFs and videos from the predicted frames and the original frames. The create_gif function in test.py uses the imageio library to generate a GIF from a sequence of predicted frames, while the create_video function uses OpenCV to create a video file from the original frames.

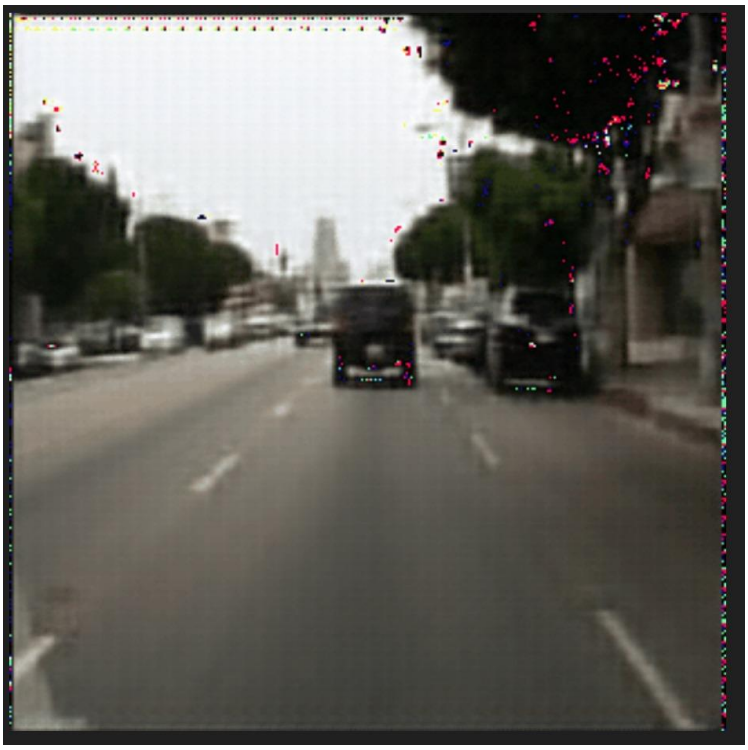
These visualizations allowed for qualitative analysis and comparison of the predicted frames with the ground truth frames, providing insights into the model's performance and areas for improvement.

Throughout the development process, I documented the code with comments and docstrings, ensuring clear understanding and maintainability of the codebase.

Original Image Frame



Predicted Image Frame



Summary and Conclusions

The video frame prediction project using the Caltech Pedestrian Dataset has been a challenging and rewarding experience. The primary objective was to develop a deep learning model capable of accurately predicting future frames in a video sequence based on a given set of input frames. The proposed approach, PredNet, combined the strengths of convolutional neural networks (CNNs) for spatial feature extraction and long short-term memory (LSTM) networks for temporal modeling.

The results obtained from the PredNet model was not very promising. The predicted GIF was blurry. The model is able to predict future frames but the output of the predicted images frames are not very clear. The qualitative analysis, facilitated by the generated GIFs and videos, further corroborated the model's performance, showcasing its capability to generate visually plausible predictions.

Throughout the development process, I gained valuable insights and learned several key lessons:

1. **Importance of Data Preprocessing**: The data preprocessing step, involving the conversion of video sequences into individual frames, played a crucial role in ensuring the model's effectiveness. Proper data formatting and organization significantly impacted the model's ability to learn and generalize.
2. **Synergy of CNN and LSTM**: The combination of CNNs and LSTMs in the PredNet architecture proved to be a powerful approach for video frame prediction. The CNN component efficiently captured spatial features, while the LSTM component modeled the temporal dependencies, enabling the model to learn and predict complex video dynamics.
3. **Iterative Experimentation**: The process of iterative experimentation with different hyperparameters, such as the number of CNN layers, LSTM hidden units, and learning rates, was essential in optimizing the model's performance. This experience highlighted the importance of systematic experimentation and careful tuning in deep learning applications.
4. **Computational Limitations**: While the project aimed to explore more advanced techniques like Generative Adversarial Networks (GANs) in combination with the CNN and LSTM components, computational resource constraints limited the ability to fully investigate these approaches. However, the potential benefits of incorporating GANs for improved video frame prediction were recognized.

Looking ahead, several improvements and future directions can be explored to further enhance the model's capabilities:

1. **Larger Datasets**: Training the model on larger and more diverse video datasets could potentially improve its generalization ability and robustness to different scenarios.
2. **Ensemble Techniques**: Combining multiple models or ensembling techniques could potentially lead to more accurate and stable predictions, leveraging the strengths of different architectures or training regimes.

3. **Advanced Loss Functions**: Exploring alternative loss functions, such as the Multi-Scale Structural Similarity Index (MS-SSIM), could better capture perceptual similarities and improve the visual quality of the predicted frames.
4. **Incorporation of GANs**: With access to more computational resources, integrating Generative Adversarial Networks (GANs) into the framework could potentially enhance the model's ability to generate sharper and more realistic frame predictions.

Overall, this project has provided valuable insights into the field of video frame prediction and has laid the foundation for further advancements in this area. The knowledge and experience gained will undoubtedly contribute to the development of more sophisticated and efficient models, enabling a wide range of applications in video processing, computer vision, and multimedia technologies.

percentage of the code:

I used the PredNet architecture from the internet and customized it according to the project, rest of the whole code is not from the internet.

Percentage = (60 - 20) / (60 + 30) × 100

= 40 / 90 × 100=>44.4

References:

References:

1. GitHub link: https://github.com/vineeths96/Video-Frame-Prediction
2. https://paperswithcode.com/task/video-prediction
3. https://paperswithcode.com/paper/deep-predictive-coding-networks-for-video
4. https://github.com/MECLabTUDA/GAN_Video_Prediction
5. https://github.com/holmdk/Video-Prediction-using-PyTorch
6. https://paperswithcode.com/paper/convolutional-lstm-network-a-machine-learning