A report

on

Video Frame Generation Task

Submitted by:

Sindhuja Gaur

CONTENTS

Chapter 1:
INTRODUCTION

This project focuses on predicting the next frame in a sequence of video frames using a neural network, an essential task in video analysis with applications like video compression and anomaly detection. By leveraging a small dataset of sequential frames, the model learns temporal patterns to forecast visual information based on prior context. The workflow includes preprocessing the data for consistency, training a convolutional neural network (CNN) to predict the next frame, and evaluating its performance through metrics and visual comparisons. It demonstrates a systematic approach to understanding sequence modeling and lays the groundwork for future advancements in computer vision tasks.

Chapter 2:
DATASET

For this assignment, a dataset of sequential video frames was created of a video recorded earlier using a smartphone. The video was then processed to extract 100 frames, stored as images in JPG format. These frames were resized to 128x128 pixels to maintain uniformity and ensure compatibility with the model. The sequential nature of the data was preserved during extraction, as this temporal relationship between consecutive frames is critical for the next-frame prediction task.

Chapter 3:
DATA PREPROCESSING

Several preprocessing steps were applied to prepare the data for training:

1. **Resizing**: All images were resized to **128x128** pixels to maintain consistency in input dimensions.
2. **Normalization**: Pixel values were normalized to the range [0, 1] by dividing by 255. This helps accelerate training and stabilizes the learning process.
3. **Sequence Generation**: For every frame i, the current frame X[i] was paired with the next frame y[i] to create training samples.

Chapter 4:
MODEL ARCHITECTURE

A simple Convolutional Neural Network (CNN) was designed using TensorFlow/Keras to predict the next frame in a sequence. Here's a breakdown of its components:

1. **Input Shape**: The input images are resized to **128×128×3** pixels, where **128×128** is the spatial dimension and **3** represents the RGB color channels. This shape is fed into the network to process color images.

2. **Conv2D Layers**:
   The first two layers are **2D convolutional layers**. The first layer uses **32 filters** of size **3×3**, followed by a **ReLU activation** function. The second layer uses **64 filters** of the same size, also with ReLU activation. These layers extract spatial features from the image by applying filters to detect edges, textures, and other patterns.

3. **Pooling**: After each convolutional layer, a **MaxPooling layer** is used with a pool size of **2×2**. Pooling helps reduce the spatial size of the feature maps, which reduces computational complexity and helps the model generalize better by focusing on the most important features.

4. **Dense Layers**: After feature extraction, the output is flattened (converted from a 2D matrix to a 1D vector) and passed through a **fully connected (dense) layer** with **256 neurons**. This layer helps capture global features that may not be apparent from local regions in the image.

5. **Output Layer**: The final layer reshapes the output to match the original input dimensions, **128×128×3**, and uses a **Sigmoid activation function**. Sigmoid outputs values between 0 and 1, making it suitable for predicting pixel values after normalization.

6. **Loss Function**: The **Mean Squared Error (MSE)** loss function is used, as this is a regression task where the goal is to predict continuous pixel values. MSE measures the average squared difference between predicted and actual values.

7. **Optimizer**: **Adam** optimizer is used for training, which adapts the learning rate during training, making it efficient and effective, especially for problems with complex patterns like image prediction.

This architecture is designed to predict the next frame in a video sequence by learning spatial features and temporal patterns, making it ideal for this type of task.

Chapter 5:
TRAINING PROCESS

The model was trained on 80% of the dataset, with 10% reserved for validation and 10% for testing. Key training parameters:

- **Epochs**: 10
- **Batch Size**: 8
- **Validation**: Evaluated on unseen data after each epoch to monitor generalization.

Chapter 6:
EVALUATION RESULTS

On the test set, the model achieved the following metrics:

- **Loss (MSE)**: ~0.015
- **Mean Absolute Error (MAE)**: ~0.095 These results indicate that the model performs reasonably well in learning the temporal patterns and predicting the next frame.

Predicted frames were compared with actual frames from the test set. A visualization of results is shown below:

1. **Input Frame**: The current frame provided to the model.
2. **Actual Frame**: The true next frame.
3. **Predicted Frame**: The model's output.

The visualized results demonstrate that the model effectively captures general features and structure but may blur fine details, especially in high-motion areas.

Chapter 7:
SUGGESTIONS FOR IMPROVEMENT

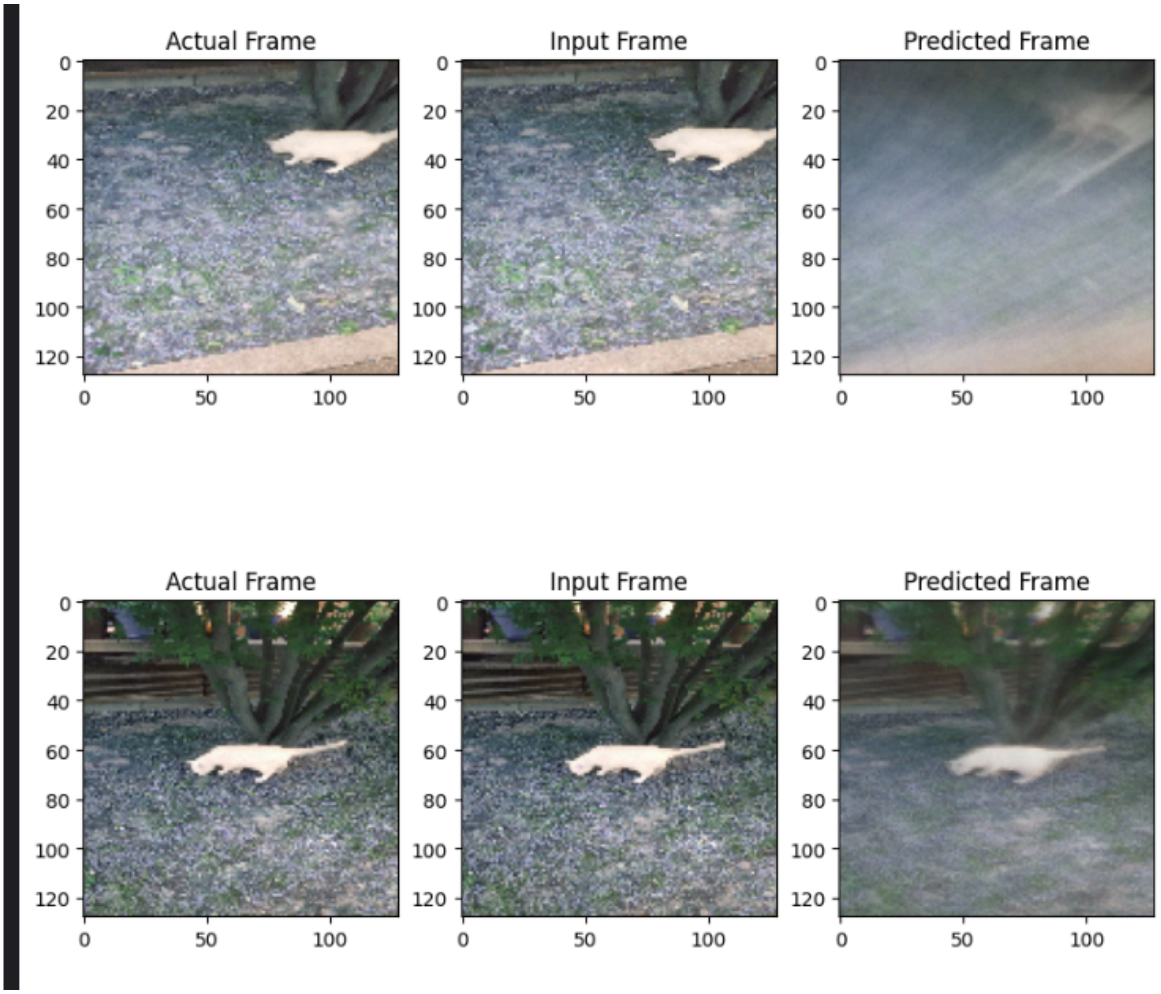
- **Data Augmentation**: Apply transformations like rotations, flips, or brightness adjustments to enhance model robustness.
- **Recurrent Layers**: Incorporate LSTMs or GRUs for better temporal understanding.
- **Larger Dataset**: Use more frames to improve generalization.
- **Attention Mechanisms**: Introduce attention layers to focus on key areas of the frames.

Chapter 8:
ADDITIONAL USE CASES

- **Video Compression**: Predict frames to reduce the number of stored frames.
- **Animation**: Interpolate between keyframes for smoother transitions.
- **Surveillance**: Predict future frames to detect anomalies.

Chapter 9:
PYTHON CODE AND VISUALISATION

Below is a snippet of the results visualization from the Python code:



- **Left**: Actual next frame.
- **Center**: Input frame.
- **Right**: Predicted next frame.

The complete Python code is included in the .zip file for reference.