REPUBLIC OF TURKEY

YILDIZ TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

NEXT FRAME PREDICTION

16011057 — Büşra KÜDEN

17011001 — Elif ÖZCAN

SENIOR PROJECT

Advisor

Associate Prof. Mehmet Fatih AMASYALI

July, 2021

ACKNOWLEDGEMENTS

First of all we would like to express our gratitude and respect to all Yildiz Technical University Electrical and Electronics Faculty Computer Engineering family and our valuable teacher, who guided and supported us with his valuable knowledge and experience during our project work, Associate Prof. Mehmet Fatih AMASYALI.

We would like to thank our family, who have never left us alone with their financial and moral support throughout our work, and strive to provide us with a nice working environment.

Büşra KÜDEN
Elif ÖZCAN

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

NFP	Next Frame Prediction
GAN	Generative Adversarial Networks
CNN	Convolutional Neural Network
SSIM	Structural Similarity Index
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
CONV_LSTM	Convolutional Long Short Term Memory
RMSE	Root Mean Square Error
SSIM	Structural Similarity Index

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

NEXT FRAME PREDICTION

Büşra KÜDEN
Elif ÖZCAN

Department of Computer Engineering
Senior Project

Advisor: Associate Prof. Mehmet Fatih AMASYALI

Next frame prediction in videos is an artificial intelligence application that involves predicting the next few frames of a video given its previous frames. Next frame prediction in videos has a wide range of applications such as decision-making in robots and autonomous vehicles. Next frame prediction is one of the promising application areas in Computer Vision.
Two data sets were used within the scope of the project. One of them is our own dataset containing 1200 videos, each of which consists of 96 frames of 80*80 size. How we created this data set is explained in detail in the report. The other dataset we use is the Moving Mnist dataset, which is used in many next frame prediction research. Moving MNIST contains 10,000 sequences each of length 20 showing 2 digits moving in a 64 x 64 frame.
We trained three different models with these two datasets and compared their results. We used Conv-lstm, pix2pix, and Multiscale GAN models.

Keywords: Next frame prediction, video, pix2pix, Conv-lstm, GAN, dataset

1 INTRODUCTION

With Artificial Intelligence growing bigger day by day what we can do with it also expands. We can produce texts, predict weather, or detect objects in images and much more such as next frame predicting. Next Frame prediction is an application of AI which involves predicting the next few frames of a video given the previous frames. This prediction is about getting the history of images or understanding the sequence while the input being the previous few frames, and prediction are the next frames.

These predictions can be anything, for example a video frame predictor can be shown several movies of a specific genre, such as romance movies or action thrillers. The video frame predictor can learn the probability distribution of the frames from the second half of a movie given the frames from the first half[1]. But predicting the other half of the still far from what is achieved today, still many scenes in real life can be predicted since they satisfy physical laws, such as ball parabola prediction for a ping-pong robot.

In this project we will try different deep learning models for predicting next frames and continue our path with the best model, discuss ideas for predicting and performance metrics. Also try to understand which fields next frame prediction can be used.

In this document we will represent to you preliminary studies we do, feasibility, system analysis and system design.

2
PRELIMINARY EXAMINATION

In this chapter we will talk about our researches and preparations for the project.

2.1 Next Frame Prediction

Next Frame prediction is an application of AI which involves predicting the next few frames of a video given the previous frames. This prediction is about getting the history of images or understanding the sequence while the input being the previous few frames, and prediction are the next frames. With a next frame prediction model, we want to achieve unknown frames of a video. Predicting next frames can be used in many fields such as entertainment, robotics or autonomous vehicles.[2]

In entertainment field next frame predicting model can be used to generate infinite numbers of videos, for robotics and autonomous vehicles NFP can be used while making decision so we believe predicting next frames holds and big importance in the future.

In the history of next frame prediction there is many approaches like Generative Adversarial Networks (GAN) or pix2pix.

While predicting next frames general steps are like:

- Create/find video for training the model
- Train a model for predicting
- Give test frames for predicting next frames
- And for generating video use predicted frames to make a video[3]

2.2 Generative Adversarial Networks(GAN)

Deep learning is growing with the powerful hardware and the most striking successes in deep learning have involved discriminative models, usually those that map a high-dimensional, rich sensory input to a class label[4]. These striking successes have primarily been based on the backpropagation and dropout algorithms. Generative deep learning algorithms did not grow as much but now the success and many fields of GAN is very exciting.

In 2014, Ian Goodfellow and his colleagues at the University of Montreal published a paper introducing Generative Adversarial Networks or GANs[5]. Generative models create new data instances that resemble your training data. Generative modeling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset[6].

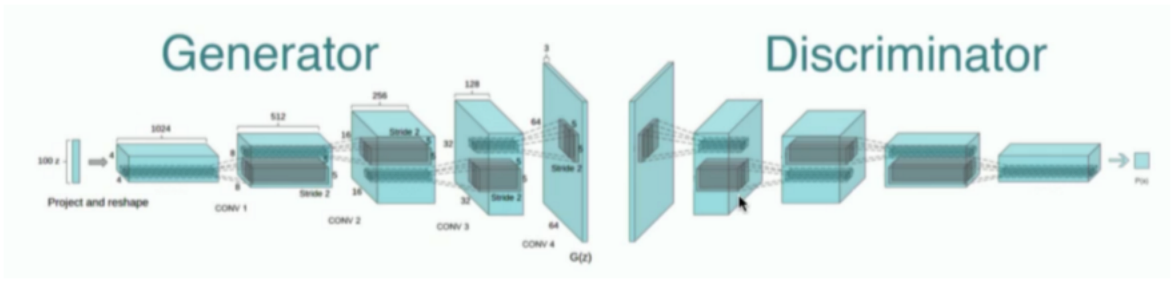


Figure 2.1 Generative Adversarial Networks [7]

Generative adversarial networks have two opposite neural networks, Generator and Discriminator. The generator tries to generate fake data that looks just like the real data and generated data becomes negative examples for discriminator. Discriminator learns to differentiate the real and fake data that generated. Through backpropagation, the discriminator's classification provides a signal that the generator uses to update its weights[8]. The generator tries to fool the discriminator, and the discriminator tries to keep from being fooled. In ideal case both discriminator and generator get better and better until generator gets too good and discriminator doesn't understand the data being generated.

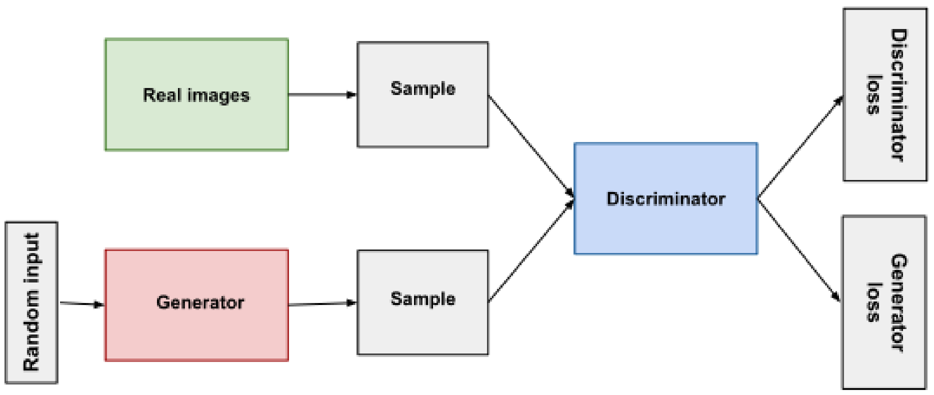


Figure 2.2 Architecture of GAN [8]

Goodfellow's metaphor was that generator was like a team of forgers trying to match real paintings with their output, while discriminator was the team of detectives trying to tell the difference[5].

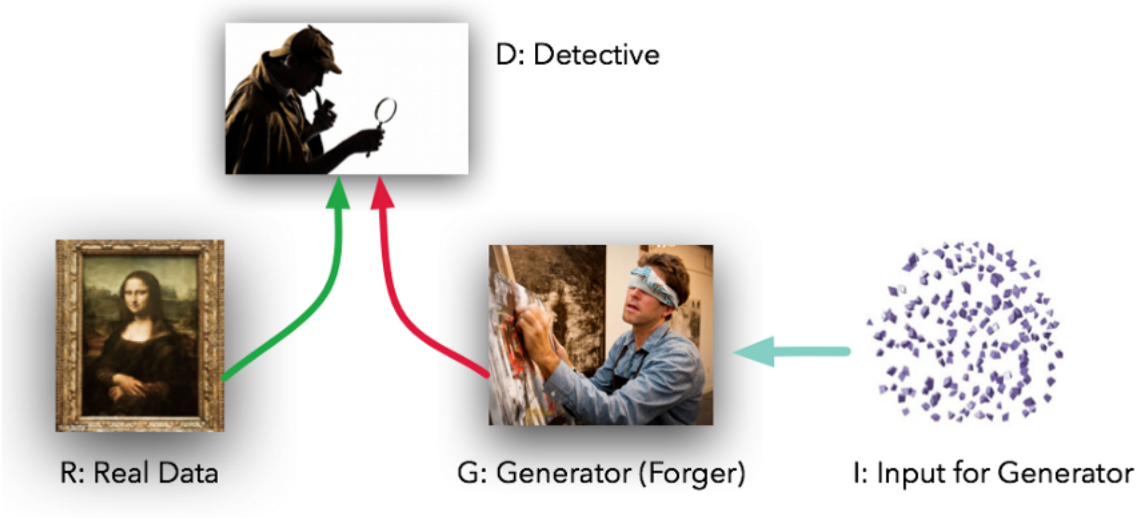


Figure 2.3 Generator and Discriminator

With many studies about GANs, level of realism achieved is incredible.

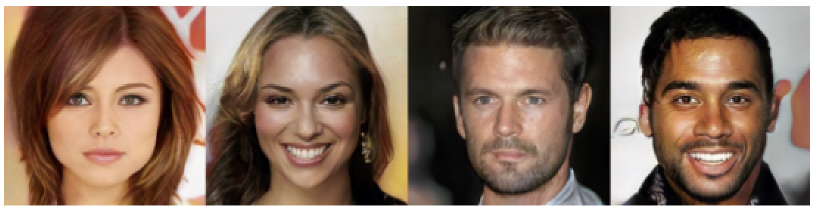


Figure 2.4 Images generated by a GAN created by NVIDIA [9]

2.2.1 Pix2Pix

Pix2Pix is a conditional image-to-image translation architecture that uses a conditional GAN which is presented in 2016 by researchers from Berkeley in their work "Image-to-Image Translation with Conditional Adversarial Networks"[10]. Compared to other GAN models image-to-image translation model can be used in many tasks such as converting maps to satellite photographs, black and white photographs to color, and sketches of products to product photographs and can work with big sized images too[11].

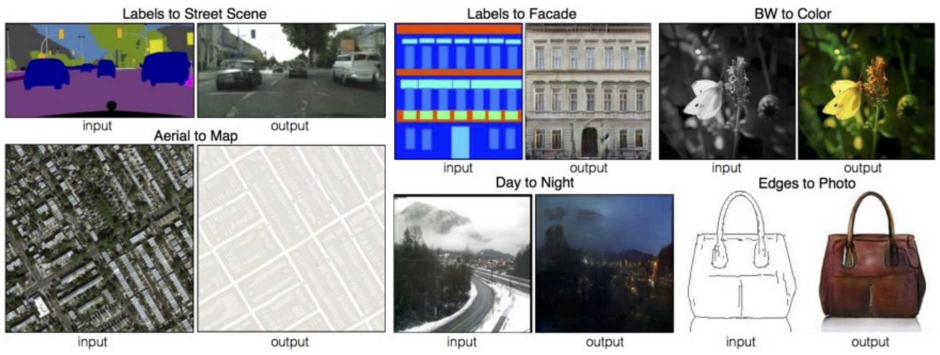


Figure 2.5 Examples of Pix2Pix Usage[12]

Both a source image and a target image are provided to the discriminator and discriminator must determine if the target is a reasonable transformation of the source image. Generator gets updated with both difference with target and output also discriminators decisions.[6]

2.3 ConvLSTM(Convolutional Long Short Term Memory) [13]

LSTM is a type of RNN, Recurrent Neural Network. LSTM has a Recurrent network structure like RNN, but instead of a single layer, it consists of four interactive layers. These layers are also called doors. It is a structure that receives information outside the normal flow. This information can be stored, written to the cell, and read. The cell decides what to store, when to read, write or delete, with the doors.

It is decided whether the information will pass through the door according to the weight values calculated during the learning of the recurrent network. With this structure, unnecessary and non-useful information is not sent to memory during learning. Therefore holding information on previous data the network has seen before and using it to make decisions. In other words, the data order is extremely important.

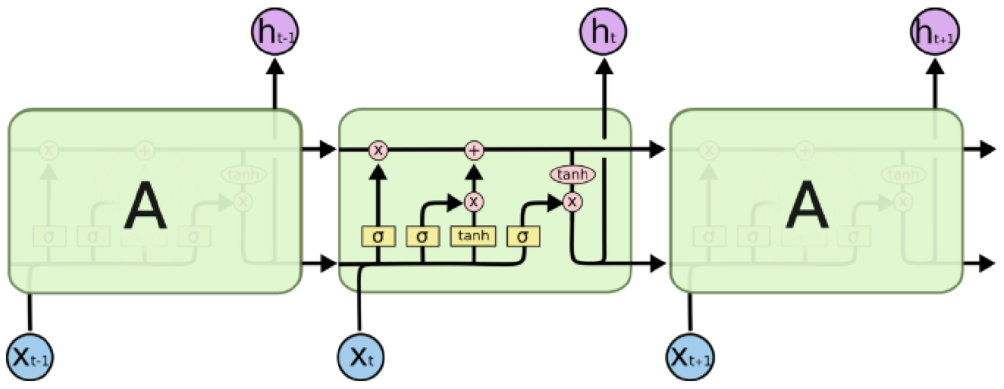


Figure 2.6 LSTM Cell [13]

Our dataset is consist of sequence of images, which is frames of videos. When working with images, the best approach is a CNN (Convolutional Neural Network) architecture. CNN tries to capture certain features by moving a specified filter step by step over the image. Usually these filters are 3x3 or 5x5 in size and more than one filter is used to detect multiple features. The parameters learned in CNN algorithms are the values in these filters. The model constantly updates these values and begins to detect features even better.

2.4 Measurement Technique

2.4.1 Structural Similarity Index (SSIM)

In this project our goal is predicting the next frames of given frames. In this task we have input images, ground truths and predicted images. While comparing models we calculate similarity of ground truth and output image for every model. For similarity we used Structural Similarity Index (SSIM) Algorithm.

SSIM models the perceived change between images and generates a similarity value. This method has been added to the Python scikit-image library. Using the compare_ssim() function generates a score and an image showing the difference between images. Score value is the similarity value between images and takes a value between -1 and 1[14]. The difference image, in other words diff, is a gray level image in which the different regions are shown darker. In Figure-2.7 is an example of calculating similarity with similarity score 0.982.

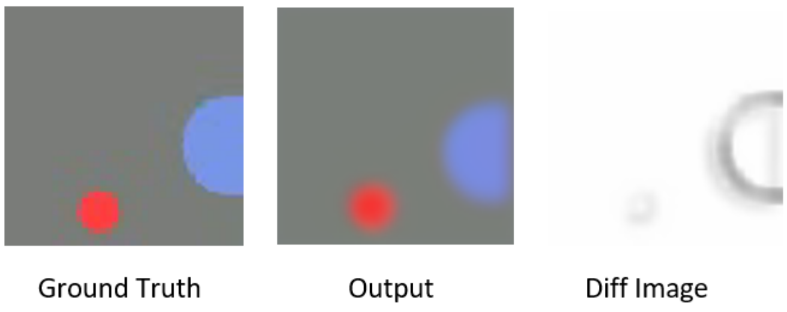


Figure 2.7 Similarity Result

In addition, as shown in the formula below, SSIM uses windows instead of the whole image when comparing.

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Figure 2.8 SSIM [14]

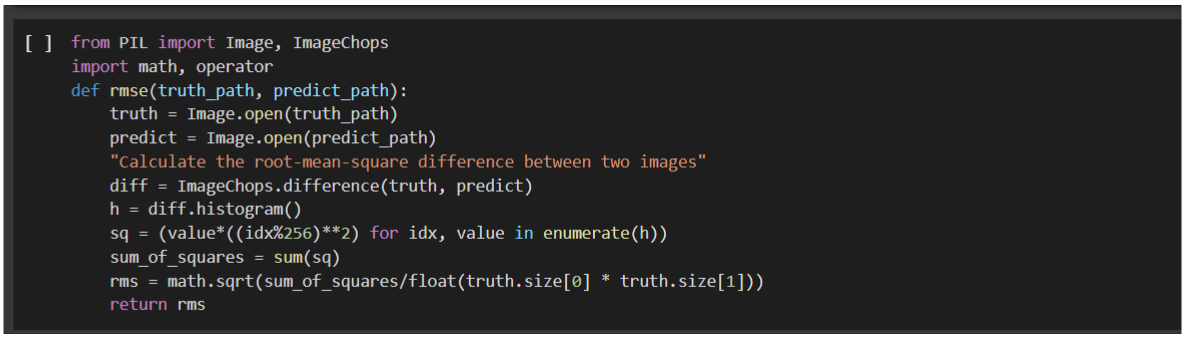
2.4.2 Root Mean Square Error (RMSE)

In the many of papers related to our project for the purpose of measuring accuracy Root Mean Square Error (RMSE) is used so we decided to use RMSE, beside our previous measurement metric.

Root Mean Square Error is often used for machine learning projects for calculating the error. RMSE is standard deviation of errors of in other words RMSE is the root of mean square error.

$$RMSE = \sqrt{\frac{\sum_{j=1}^{n} e_j^2}{n}} \tag{2.1}$$

In this method while calculating errors we used "difference" function of ImageChops from PIL.



```
[ ]  from PIL import Image, ImageChops
     import math, operator
     def rmse(truth_path, predict_path):
         truth = Image.open(truth_path)
         predict = Image.open(predict_path)
         "Calculate the root-mean-square difference between two images"
         diff = ImageChops.difference(truth, predict)
         h = diff.histogram()
         sq = (value*((idx%256)**2) for idx, value in enumerate(h))
         sum_of_squares = sum(sq)
         rms = math.sqrt(sum_of_squares/float(truth.size[0] * truth.size[1]))
         return rms
```

Figure 2.9 Root Mean Square Error Code

3
FEASIBILITY

The purpose of feasibility is to examine the possibility of the project. In this chapter we look into feasibility of project in aspect of economic, legal, technique and time.

3.1 Technique Feasibility

We examined the software and hardware features to be used for the project under this section.

3.1.1 Software Feasibility

The software requirements for the project were determined.

- Operation System: Windows 10
- Programming Language: As Python has many standard libraries for deep learning we used Python.
- Development Environment: We used Google Colab because we needed a GPU for the deep learning algorithms we use.
- Libraries:
  - Numpy : For array operations
  - OpenCV : For read and write image
  - Os : For handle files or folders within a directory
  - Chainer : Chainer is a powerful, flexible, and intuitive deep learning framework.
  - CuPy : CuPy provides GPU accelerated computing with Python.
  - Scipy : For resize image

- Keras : For deep learning algorithms
- Glob : For find all path names matching a specified pattern
- Argparse : For command-line parsing

3.1.2 Hardware Feasibility

In our project, we use deep learning algorithms which requires a lot of computational power to run on. So while working with deep learning algorithms we either need GPU or a lot of time to run on our laptops. We solved this problem with Google Colaboratory that offers 12GB GPU for 12 hours in one session for free.

3.2 Organization and Schedule Feasibility

The organization diagram is shown at Figure-3.1

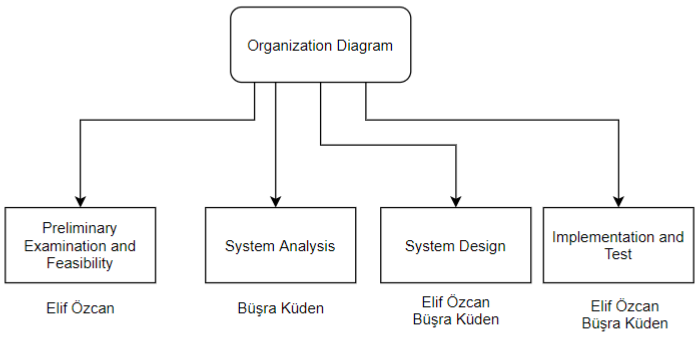


Figure 3.1 Organization Diagram

3.3 Economic Feasibility

If a software house wants to realize this project, the Table-3.1 shows how much this project will cost.

Hardware Requirement	**Piece**	**Total Cost**
Computer	2	2 * 7000 = 14000
Job Definition	**Duration (Man-Days)**	**Total Cost**
Planing Project	3	3 * 200 = 600
Feasibility	1	1 * 200 = 200
Researching about NFP and Models	20	20 * 200 = 4000
Trying/Training Models	10	10 * 200 = 2000
Creating Dataset	1	1 * 200 = 200
Training Models with Custom Dataset	5	5 * 200 = 1000
Deciding Measurement Technique and Coding	2	2 * 200 = 400
Sum	**42**	**22400**

Table 3.1 Table of Economic Feasibility for Next Frame Prediction Project

3.4 Legal Feasibility

The project is within the scope of scientific and literary work according to the 2nd article of the "Law on Intellectual and Artistic Works" numbered 5846.

4 SYSTEM ANALYSIS

In this section, how we will proceed while developing the project, the purpose and outputs of our project are explained.

4.1 Requirement Analysis

The processes determined as a result of the preliminary examination were detailed with a draft diagram. Within the scope of the project, creating and/or expanding the data set to be used for training and testing (video generation), creating/training the model to make predictions with different methods, and selecting the best model by comparing the test results.

4.2 Project Purpose and Outcomes

Next frame prediction in videos is an artificial intelligence application that involves predicting the next few frames of a video given previous frames. Next frame prediction in videos has a wide range of applications such as decision making in robots, autonomous vehicles. While predicting the next frames in videos, Computer Vision will be used to interpret each frame of the video and Sequence Modeling to understand the sequence of frames in the video. The goal of the project is to predict the next frame(s) for the given video and train the model to predict.

4.3 Draft Diagram

It is a diagram determined as a result of the preliminary examination. It is used to represent the relationship between the system and entities. Here, the system is designated as "Next Frame Prediction". The entity was identified as "programmer" and "model (the next frame prediction model we chose)". The relationship between entity and system is as in the Figure-4.1.

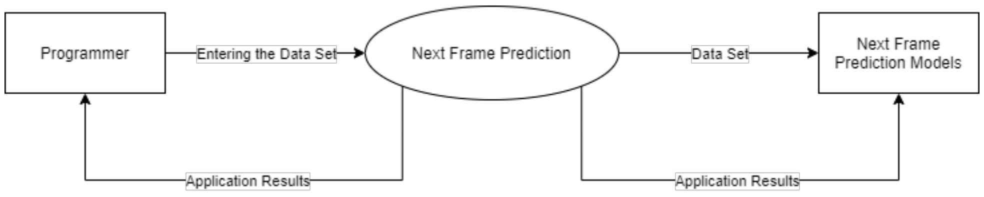


Figure 4.1 Draft Diagram

4.4 1. Level Data Flow Diagram

Here, we detailed the relationship between the entities and the system identified in the draft diagram with the processes and the database used. Our aim here is to detail the relationship of all processes of the system with each other and with external sources. The 1st level data flow diagram of our project is as shown in the Figure-4.2.

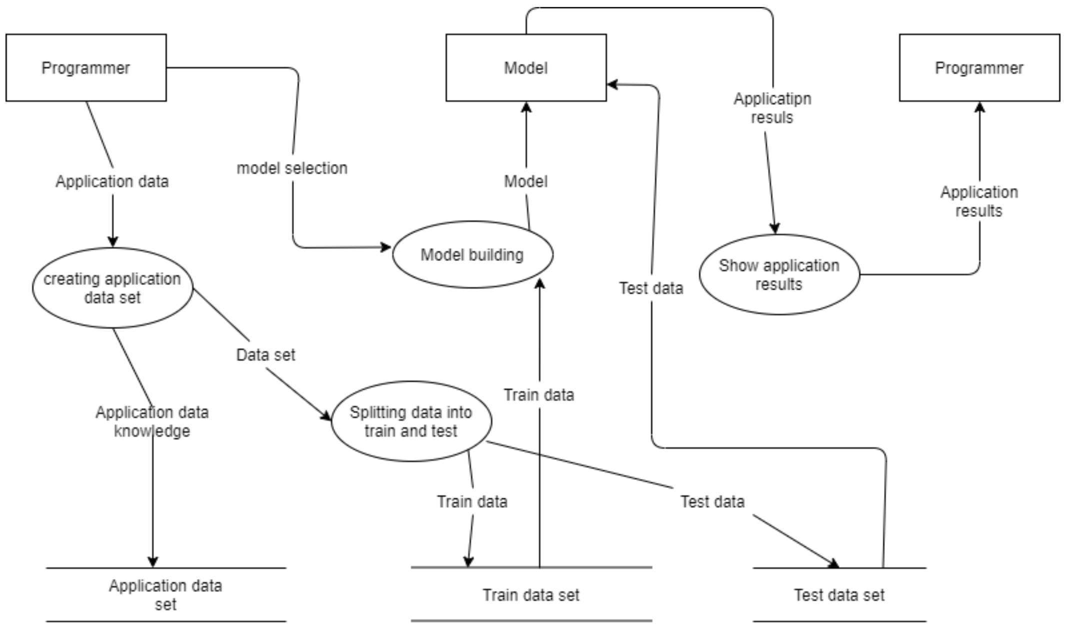


Figure 4.2 1. Level Data Flow Diagram

4.5 Use Case Diagram

Use case diagram of our project is as in the Figure-4.3.

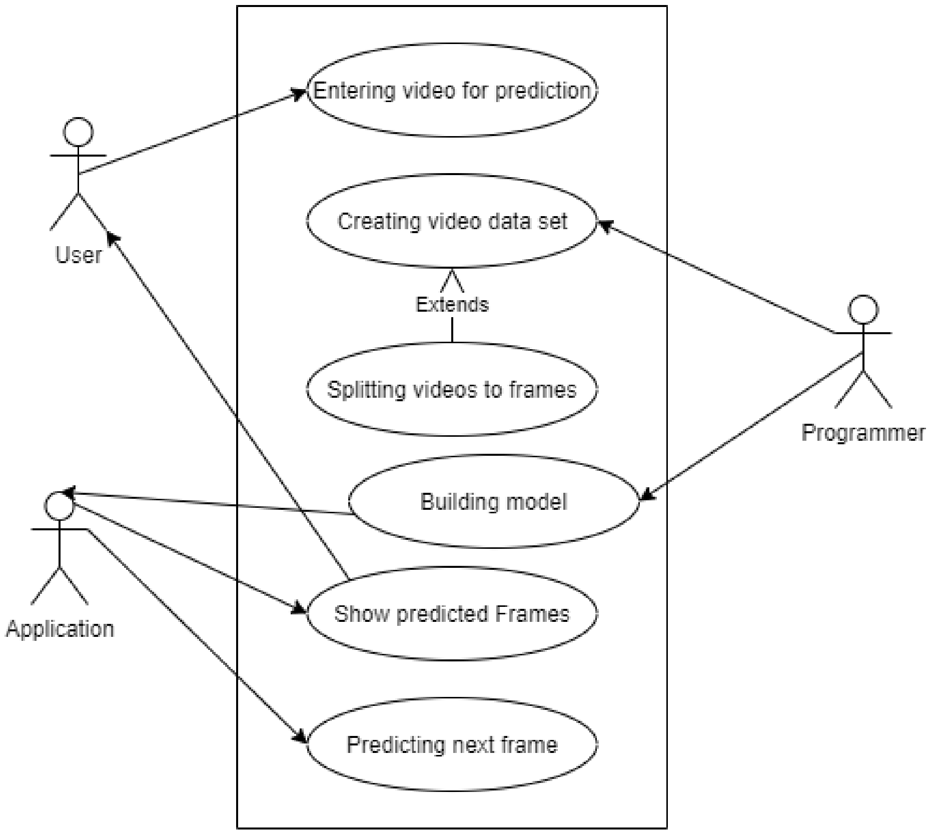


Figure 4.3 Use Case Diagram

5
SYSTEM DESIGN

In our previous study, the requirements analysis in our next frame prediction project was performed and the system requirements were determined. During the system design phase, the main processes in our project will be determined and detailed. The software, database, and input-output designs of the system were made to make the necessary detailing during the design phase.

5.1 Software Design

In this section, the main operations of the system are determined and shown in the diagram in the Figure-5.1. Later, these processes will be explained in detail.

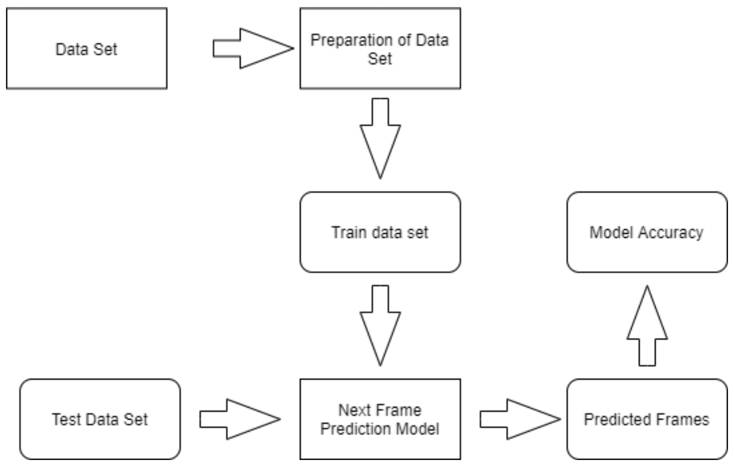


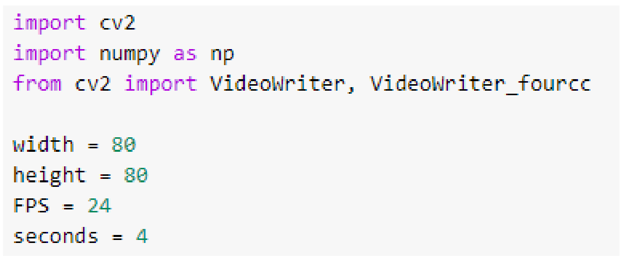
Figure 5.1 Software Design Diagram

5.2 Creation of Data Set

5.2.1 Circle Dataset

For the next frame prediction application, we created our video data set. This data set contains 1000 videos for training and 200 for testing. Each video is 4 seconds and has

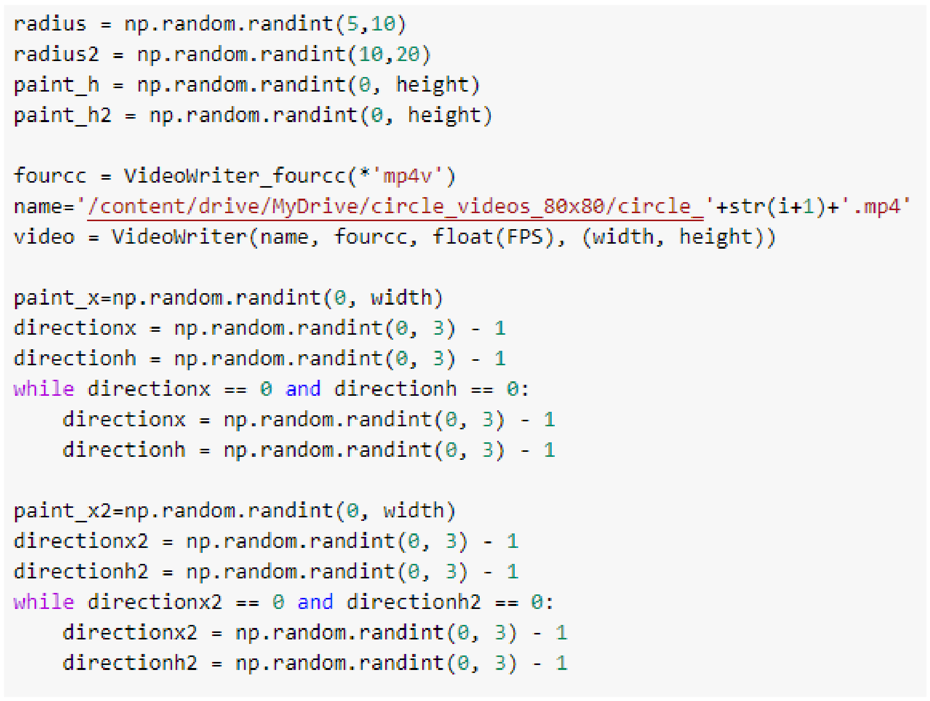
24 frames per second. Each frame is 80x80 in size. There are 2 circles in the videos, one big and one small, and these circles move linearly in the randomly determined direction.



```python
import cv2
import numpy as np
from cv2 import VideoWriter, VideoWriter_fourcc

width = 80
height = 80
FPS = 24
seconds = 4
```

Figure 5.2 Importing necessary packages and determining video size

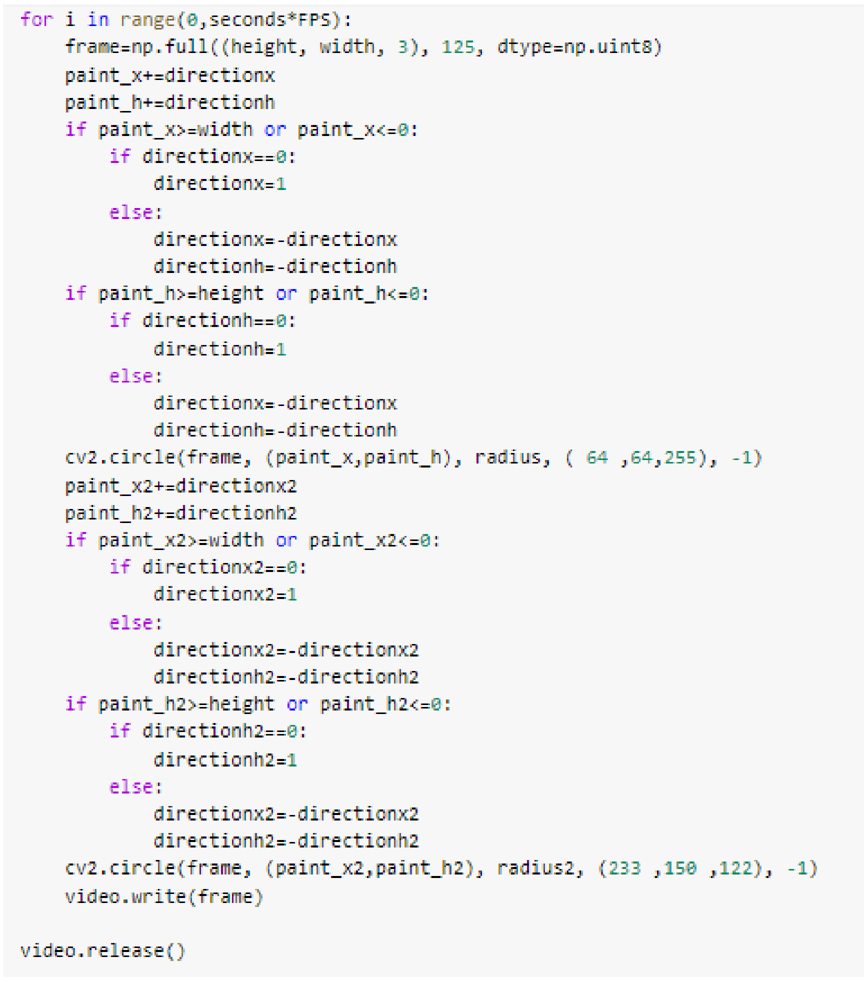
With for loop in range 0-1200, we are started to generate our videos. First of all, we randomly selected the radius of the circles in the frame within a certain range. Then, we assigned a random starting point for both circles. With the VideoWriter function in the cv2 library, we created a video in the dimensions we determined. Then we determined the directions to go for both circles.



```python
radius = np.random.randint(5,10)
radius2 = np.random.randint(10,20)
paint_h = np.random.randint(0, height)
paint_h2 = np.random.randint(0, height)

fourcc = VideoWriter_fourcc(*'mp4v')
name='/content/drive/MyDrive/circle_videos_80x80/circle_'+str(i+1)+'.mp4'
video = VideoWriter(name, fourcc, float(FPS), (width, height))

paint_x=np.random.randint(0, width)
directionx = np.random.randint(0, 3) - 1
directionh = np.random.randint(0, 3) - 1
while directionx == 0 and directionh == 0:
    directionx = np.random.randint(0, 3) - 1
    directionh = np.random.randint(0, 3) - 1

paint_x2=np.random.randint(0, width)
directionx2 = np.random.randint(0, 3) - 1
directionh2 = np.random.randint(0, 3) - 1
while directionx2 == 0 and directionh2 == 0:
    directionx2 = np.random.randint(0, 3) - 1
    directionh2 = np.random.randint(0, 3) - 1
```

Figure 5.3 Determining circle size and which way to go

We formed circles in each frame to move linearly in the direction of movement. If the circle moves out of the frame during the movement, it moves in the opposite direction of the moving direction. Then we wrote the created frame to the video.



```
for i in range(0,seconds*FPS):
    frame=np.full((height, width, 3), 125, dtype=np.uint8)
    paint_x+=directionx
    paint_h+=directionh
    if paint_x>=width or paint_x<=0:
        if directionx==0:
            directionx=1
        else:
            directionx=-directionx
            directionh=-directionh
    if paint_h>=height or paint_h<=0:
        if directionh==0:
            directionh=1
        else:
            directionx=-directionx
            directionh=-directionh
    cv2.circle(frame, (paint_x,paint_h), radius, ( 64 ,64,255), -1)
    paint_x2+=directionx2
    paint_h2+=directionh2
    if paint_x2>=width or paint_x2<=0:
        if directionx2==0:
            directionx2=1
        else:
            directionx2=-directionx2
            directionh2=-directionh2
    if paint_h2>=height or paint_h2<=0:
        if directionh2==0:
            directionh2=1
        else:
            directionx2=-directionx2
            directionh2=-directionh2
    cv2.circle(frame, (paint_x2,paint_h2), radius2, (233 ,150 ,122), -1)
    video.write(frame)

video.release()
```

Figure 5.4 Determining circle size and which way to go

Thus, we created our video data set. There is one of the frames created in the Figure-5.5.

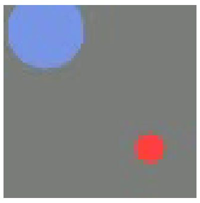


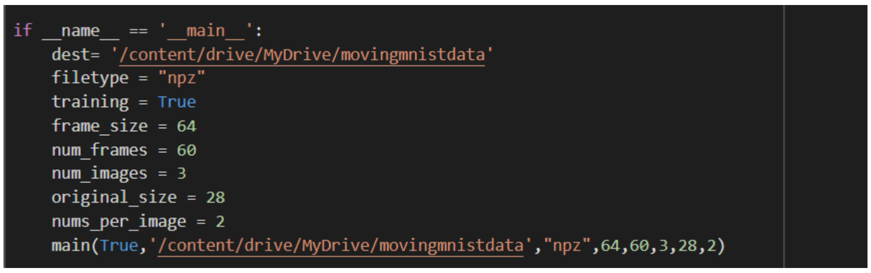
Figure 5.5 Frame example

5.2.2 Moving Mnsit

While training the models we also used Moving Mnist dataset beside circle dataset and the reason for this is to compare our results with other studies that usually uses the Moving Mnsit dataset.

In the pix2pix model only one video is used to train the model and normally Moving Mnist Dataset has 10.000 videos each containing 20 frames so we have to use 10 frames for training and 10 frames for testing. After getting the results we thought 10 frames can be so small for training and we create a video with 60 frames of Mnist data.

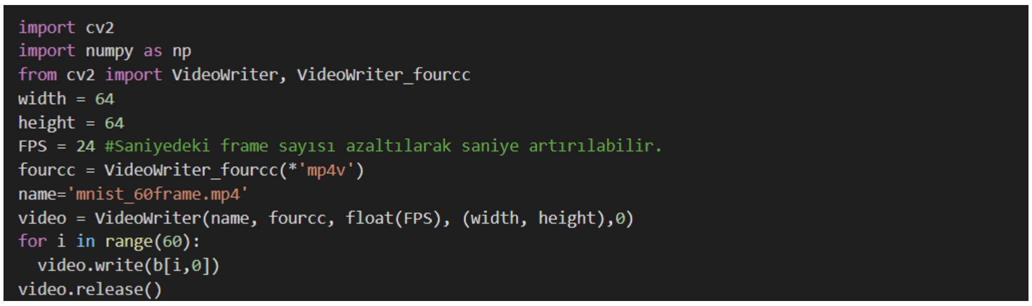
While creating new data we used a Github repository[15] with the slightest changes.



```
if __name__ == '__main__':
    dest= '/content/drive/MyDrive/movingmnistdata'
    filetype = "npz"
    training = True
    frame_size = 64
    num_frames = 60
    num_images = 3
    original_size = 28
    nums_per_image = 2
    main(True,'/content/drive/MyDrive/movingmnistdata',"npz",64,60,3,28,2)
```

Figure 5.6 Creating Moving Mnist 60 Frame

After creating the ".npz" file we used VideoWriter from OpenCv and created a video with 60 frames and 2 digits moving.

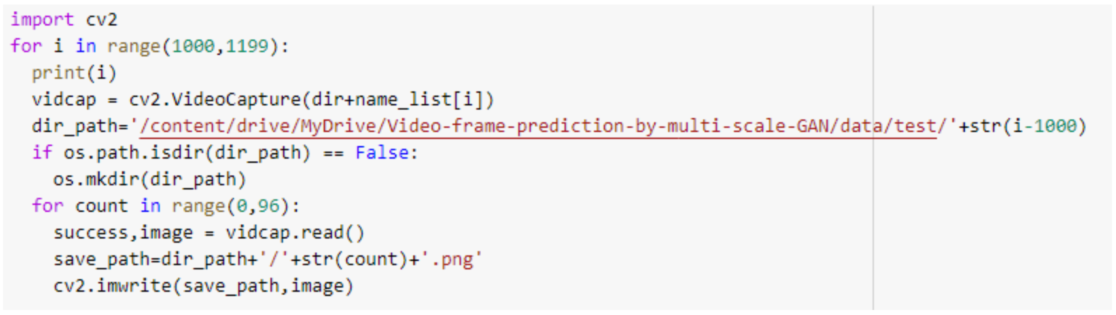


```
import cv2
import numpy as np
from cv2 import VideoWriter, VideoWriter_fourcc
width = 64
height = 64
FPS = 24 #Saniyedeki frame sayısı azaltılarak saniye artırılabilir.
fourcc = VideoWriter_fourcc(*'mp4v')
name='mnist_60frame.mp4'
video = VideoWriter(name, fourcc, float(FPS), (width, height),0)
for i in range(60):
  video.write(b[i,0])
video.release()
```

Figure 5.7 Creating Moving Mnist 60 Frame (2)

5.3 Pre-processing the data set

We split the data set into test and training set. While splitting the data set into test and training sets, we also split the videos into frames. The creation of the test set is given in the Figure-5.8.



```
import cv2
for i in range(1000,1199):
  print(i)
  vidcap = cv2.VideoCapture(dir+name_list[i])
  dir_path='/content/drive/MyDrive/Video-frame-prediction-by-multi-scale-GAN/data/test/'+str(i-1000)
  if os.path.isdir(dir_path) == False:
    os.mkdir(dir_path)
  for count in range(0,96):
    success,image = vidcap.read()
    save_path=dir_path+'/'+str(count)+'.png'
    cv2.imwrite(save_path,image)
```

Figure 5.8 The creation of the test set

6
APPLICATION

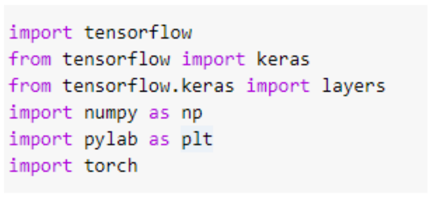
In this section, it is explained how video frame prediction by multi-scale GAN[16], conv_lstm[13], and pix2pix[17] models.

6.1 Conv_lstm

6.1.1 Model 1

6.1.1.1 Circle Dataset

Firstly we imported the necessary packages.

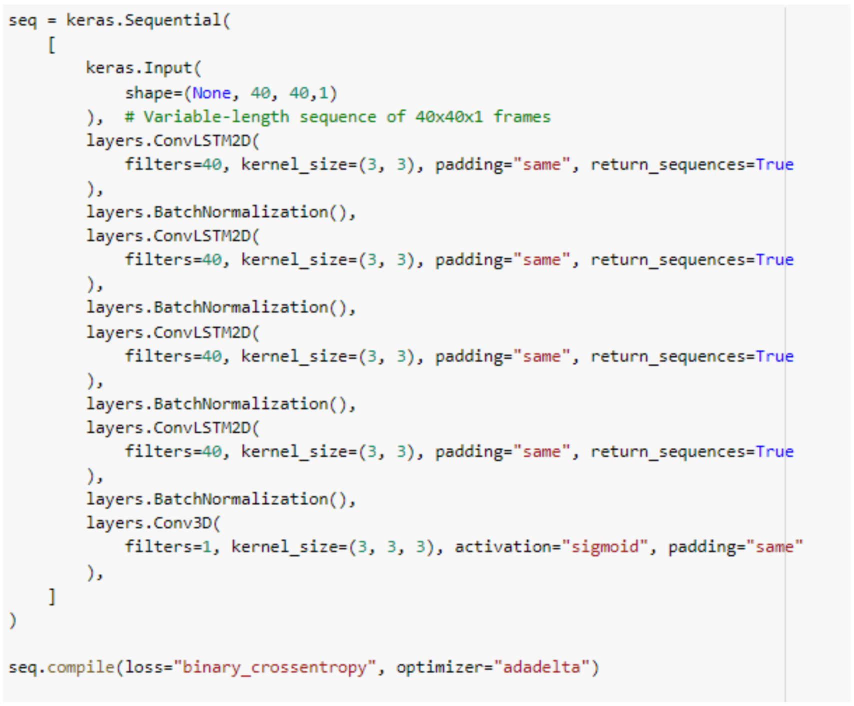


```
import tensorflow
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import pylab as plt
import torch
```

Figure 6.1 Importing the necessary packages

We create a model which take as input movies of shape (n_frames, width, height, channels) and returns a movie of identical shape. We used one input layer, four ConvLSTM2D layers, and one Conv3D layer.

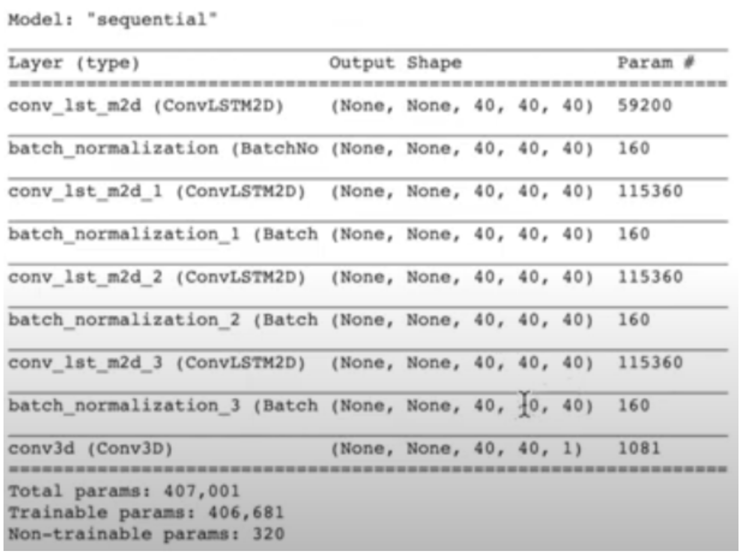
Batch normalization which is used to stabilize training and is one of the most important things to know about deep learning generally tends to dramatically stabilize and improve training deep neural networks.



```
seq = keras.Sequential(
    [
        keras.Input(
            shape=(None, 40, 40,1)
        ),  # Variable-length sequence of 40x40x1 frames
        layers.ConvLSTM2D(
            filters=40, kernel_size=(3, 3), padding="same", return_sequences=True
        ),
        layers.BatchNormalization(),
        layers.ConvLSTM2D(
            filters=40, kernel_size=(3, 3), padding="same", return_sequences=True
        ),
        layers.BatchNormalization(),
        layers.ConvLSTM2D(
            filters=40, kernel_size=(3, 3), padding="same", return_sequences=True
        ),
        layers.BatchNormalization(),
        layers.ConvLSTM2D(
            filters=40, kernel_size=(3, 3), padding="same", return_sequences=True
        ),
        layers.BatchNormalization(),
        layers.Conv3D(
            filters=1, kernel_size=(3, 3, 3), activation="sigmoid", padding="same"
        ),
    ]
)

seq.compile(loss="binary_crossentropy", optimizer="adadelta")
```

Figure 6.2 Building a model

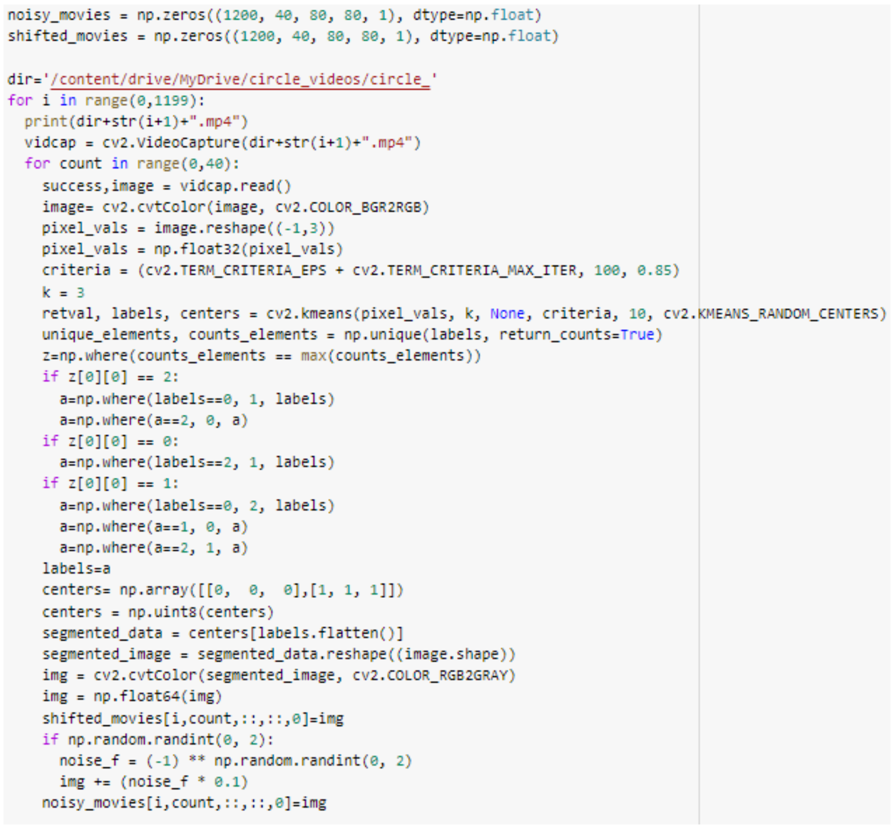
Putting these together our neural network contains 400 000 parameters, and this is for very low resolution 40 by 40 images and only gray-scale images.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_lst_m2d (ConvLSTM2D)    (None, None, 40, 40, 40)  59200
_________________________________________________________________
batch_normalization (BatchNo (None, None, 40, 40, 40)  160
_________________________________________________________________
conv_lst_m2d_1 (ConvLSTM2D)  (None, None, 40, 40, 40)  115360
_________________________________________________________________
batch_normalization_1 (Batch (None, None, 40, 40, 40)  160
_________________________________________________________________
conv_lst_m2d_2 (ConvLSTM2D)  (None, None, 40, 40, 40)  115360
_________________________________________________________________
batch_normalization_2 (Batch (None, None, 40, 40, 40)  160
_________________________________________________________________
conv_lst_m2d_3 (ConvLSTM2D)  (None, None, 40, 40, 40)  115360
_________________________________________________________________
batch_normalization_3 (Batch (None, None, 40, 40, 40)  160
_________________________________________________________________
conv3d (Conv3D)              (None, None, 40, 40, 1)   1081
=================================================================
Total params: 407,001
Trainable params: 406,681
Non-trainable params: 320
```

Figure 6.3 Model summary

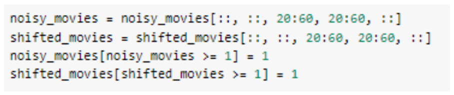
Then, we preprocessed the circle dataset to train the model. After reading the videos from the Drive, we set the background and two circles by clustering so that the data is not corrupted while the image is converted to black and white. We have obtained both noisy and normal image data. This is used to help the model make sense of predictions that aren't just 1s for the squares and 0s for the background.



```
noisy_movies = np.zeros((1200, 40, 80, 80, 1), dtype=np.float)
shifted_movies = np.zeros((1200, 40, 80, 80, 1), dtype=np.float)

dir='/content/drive/MyDrive/circle_videos/circle_'
for i in range(0,1199):
  print(dir+str(i+1)+".mp4")
  vidcap = cv2.VideoCapture(dir+str(i+1)+".mp4")
  for count in range(0,40):
    success,image = vidcap.read()
    image= cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    pixel_vals = image.reshape((-1,3))
    pixel_vals = np.float32(pixel_vals)
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.85)
    k = 3
    retval, labels, centers = cv2.kmeans(pixel_vals, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
    unique_elements, counts_elements = np.unique(labels, return_counts=True)
    z=np.where(counts_elements == max(counts_elements))
    if z[0][0] == 2:
      a=np.where(labels==0, 1, labels)
      a=np.where(a==2, 0, a)
    if z[0][0] == 0:
      a=np.where(labels==2, 1, labels)
    if z[0][0] == 1:
      a=np.where(labels==0, 2, labels)
      a=np.where(a==1, 0, a)
      a=np.where(a==2, 1, a)
    labels=a
    centers= np.array([[0,  0,  0],[1, 1, 1]])
    centers = np.uint8(centers)
    segmented_data = centers[labels.flatten()]
    segmented_image = segmented_data.reshape((image.shape))
    img = cv2.cvtColor(segmented_image, cv2.COLOR_RGB2GRAY)
    img = np.float64(img)
    shifted_movies[i,count,::,::,0]=img
    if np.random.randint(0, 2):
      noise_f = (-1) ** np.random.randint(0, 2)
      img += (noise_f * 0.1)
    noisy_movies[i,count,::,::,0]=img
```

Figure 6.4 Preprocessing of the circle dataset

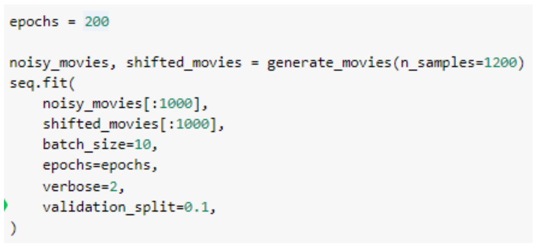
Since the frame has to be 40x40 size, we cut 40x40 size regions from each frame.



```
noisy_movies = noisy_movies[::, ::, 20:60, 20:60, ::]
shifted_movies = shifted_movies[::, ::, 20:60, 20:60, ::]
noisy_movies[noisy_movies >= 1] = 1
shifted_movies[shifted_movies >= 1] = 1
```

Figure 6.5 Cutting the images to the desired size

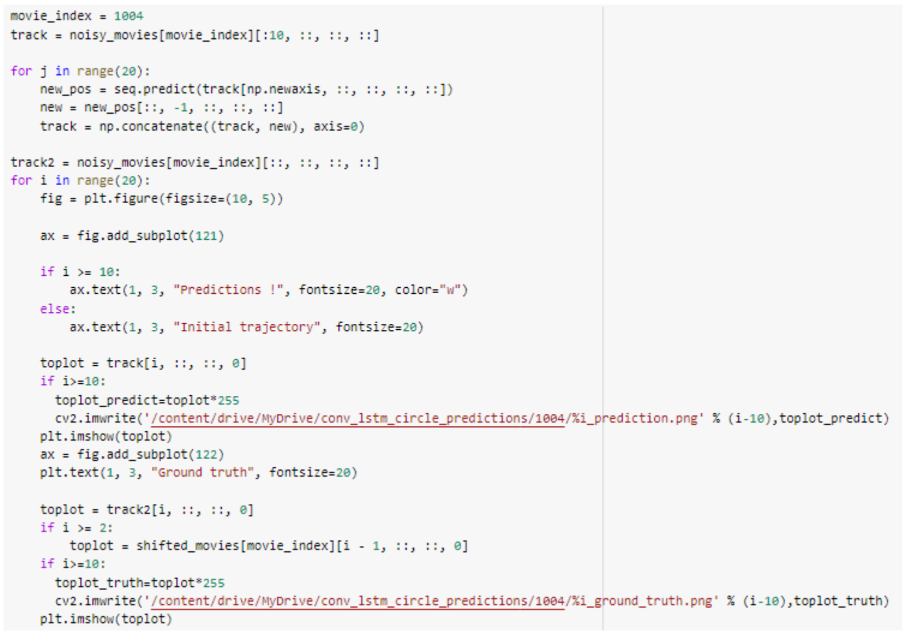
And we trained our model with 200 epochs.



```
epochs = 200

noisy_movies, shifted_movies = generate_movies(n_samples=1200)
seq.fit(
    noisy_movies[:1000],
    shifted_movies[:1000],
    batch_size=10,
    epochs=epochs,
    verbose=2,
    validation_split=0.1,
)
```

Figure 6.6 Training the model

We predicted the 11th frame using the first 10 frames in the video. For the estimation of the 12th frame, the 11th frame predicted was also used. As a result, as the predicted number of frames increased, the prediction results started to be worse.



```
movie_index = 1004
track = noisy_movies[movie_index][:10, ::, ::, ::]

for j in range(20):
    new_pos = seq.predict(track[np.newaxis, ::, ::, ::, ::])
    new = new_pos[::, -1, ::, ::, ::]
    track = np.concatenate((track, new), axis=0)

track2 = noisy_movies[movie_index][::, ::, ::, ::]
for i in range(20):
    fig = plt.figure(figsize=(10, 5))

    ax = fig.add_subplot(121)

    if i >= 10:
        ax.text(1, 3, "Predictions !", fontsize=20, color="w")
    else:
        ax.text(1, 3, "Initial trajectory", fontsize=20)

    toplot = track[i, ::, ::, 0]
    if i>=10:
      toplot_predict=toplot*255
      cv2.imwrite('/content/drive/MyDrive/conv_lstm_circle_predictions/1004/%i_prediction.png' % (i-10),toplot_predict)
    plt.imshow(toplot)
    ax = fig.add_subplot(122)
    plt.text(1, 3, "Ground truth", fontsize=20)

    toplot = track2[i, ::, ::, 0]
    if i >= 2:
        toplot = shifted_movies[movie_index][i - 1, ::, ::, 0]
    if i>=10:
      toplot_truth=toplot*255
      cv2.imwrite('/content/drive/MyDrive/conv_lstm_circle_predictions/1004/%i_ground_truth.png' % (i-10),toplot_truth)
    plt.imshow(toplot)
```

Figure 6.7 Testing the model on one movie

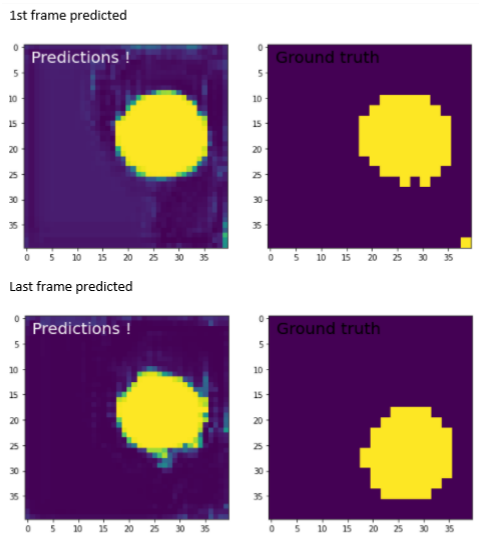
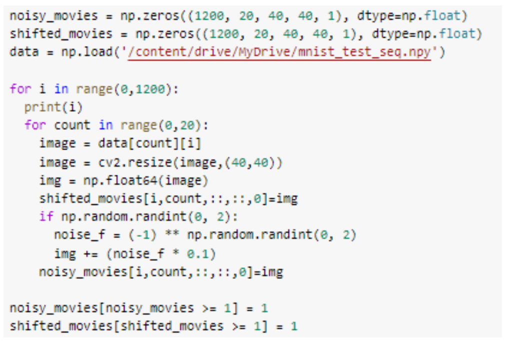


Figure 6.8 Example of predicted frames

6.1.1.2 Moving Mnist Dataset

To train the model with the Moving Mnist dataset, we first imported the necessary libraries as shown in Figure-6.1. Then we built the model as shown in the Figure-6.2. After reading the data from the npy file, we resized the images to use the dataset in the model.

And we trained our model with 200 epochs as shown in Figure-6.6. Then we predicted the next 10 frames using the first 10 frames as shown in the Figure-6.22.



```
noisy_movies = np.zeros((1200, 20, 40, 40, 1), dtype=np.float)
shifted_movies = np.zeros((1200, 20, 40, 40, 1), dtype=np.float)
data = np.load('/content/drive/MyDrive/mnist_test_seq.npy')

for i in range(0,1200):
  print(i)
  for count in range(0,20):
    image = data[count][i]
    image = cv2.resize(image,(40,40))
    img = np.float64(image)
    shifted_movies[i,count,::,::,0]=img
    if np.random.randint(0, 2):
      noise_f = (-1) ** np.random.randint(0, 2)
      img += (noise_f * 0.1)
    noisy_movies[i,count,::,::,0]=img

noisy_movies[noisy_movies >= 1] = 1
shifted_movies[shifted_movies >= 1] = 1
```

Figure 6.9 Preprocessing of the moving mnist dataset

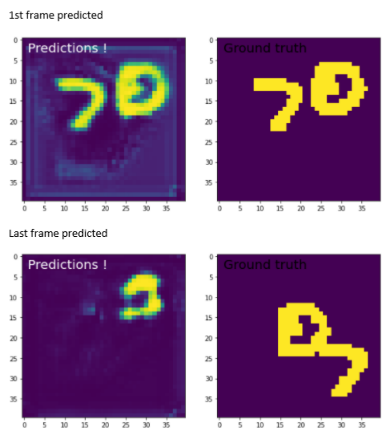


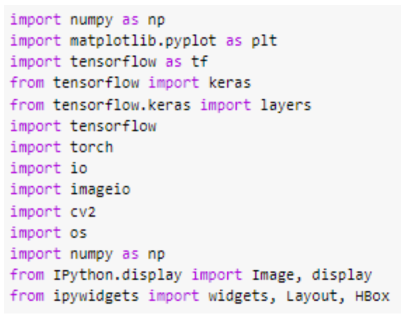
Figure 6.10 Example of predicted frames

6.1.2 Model 2

As a second model of conv_lstm we followed the tutorial in keras website[18].

6.1.2.1 Circle Dataset

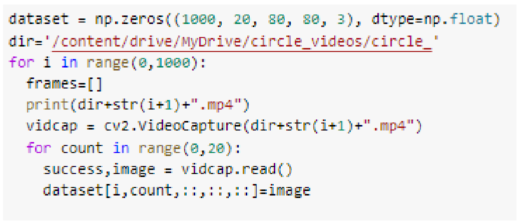
Firstly we imported the necessary packages.



```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow
import torch
import io
import imageio
import cv2
import os
import numpy as np
from IPython.display import Image, display
from ipywidgets import widgets, Layout, HBox
```

Figure 6.11 Importing the necessary packages

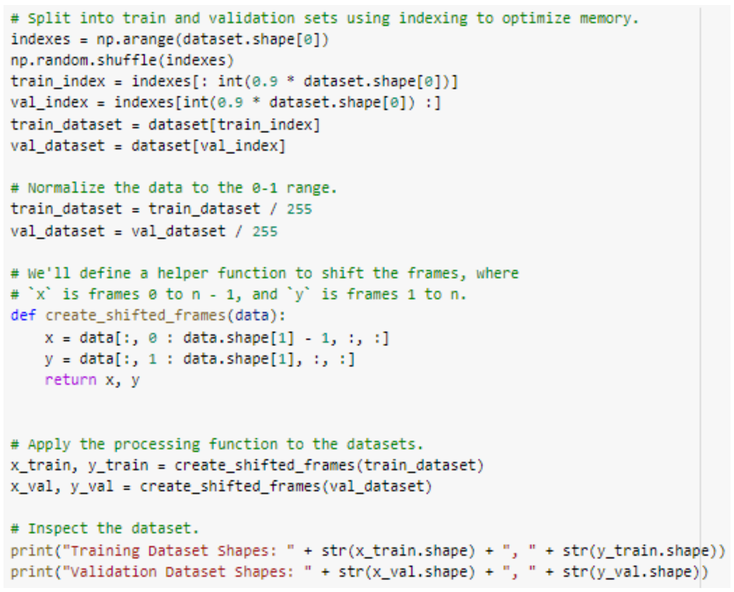
Then we created an array from the circle dataset.



```
dataset = np.zeros((1000, 20, 80, 80, 3), dtype=np.float)
dir='/content/drive/MyDrive/circle_videos/circle_'
for i in range(0,1000):
  frames=[]
  print(dir+str(i+1)+".mp4")
  vidcap = cv2.VideoCapture(dir+str(i+1)+".mp4")
  for count in range(0,20):
    success,image = vidcap.read()
    dataset[i,count,::,::,::]=image
```

Figure 6.12 Creating an array from dataset

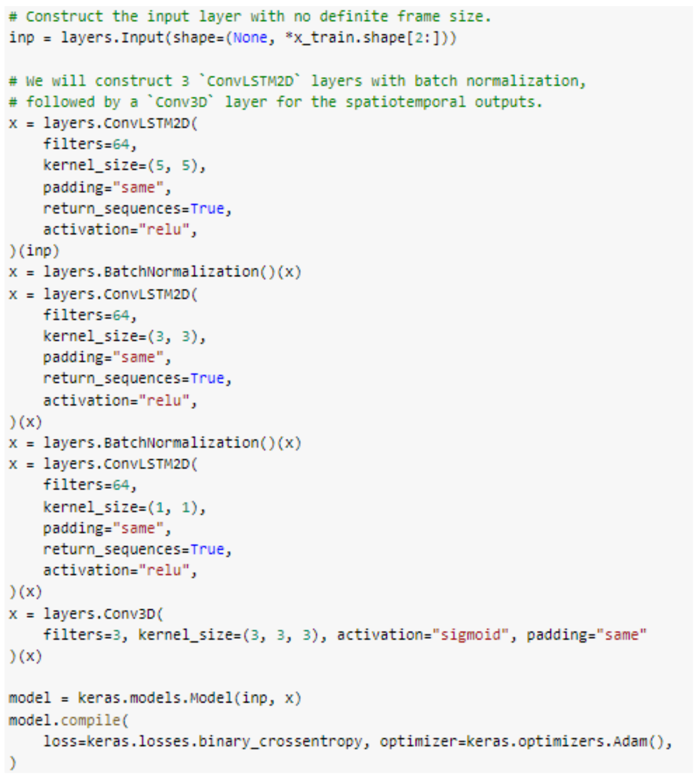
We split the dataset into training and validation data by a ratio of 0,9. While doing this, we normalized frames pixels to 0-1 values.



```
# Split into train and validation sets using indexing to optimize memory.
indexes = np.arange(dataset.shape[0])
np.random.shuffle(indexes)
train_index = indexes[: int(0.9 * dataset.shape[0])]
val_index = indexes[int(0.9 * dataset.shape[0]) :]
train_dataset = dataset[train_index]
val_dataset = dataset[val_index]

# Normalize the data to the 0-1 range.
train_dataset = train_dataset / 255
val_dataset = val_dataset / 255

# We'll define a helper function to shift the frames, where
# `x` is frames 0 to n - 1, and `y` is frames 1 to n.
def create_shifted_frames(data):
    x = data[:, 0 : data.shape[1] - 1, :, :]
    y = data[:, 1 : data.shape[1], :, :]
    return x, y


# Apply the processing function to the datasets.
x_train, y_train = create_shifted_frames(train_dataset)
x_val, y_val = create_shifted_frames(val_dataset)

# Inspect the dataset.
print("Training Dataset Shapes: " + str(x_train.shape) + ", " + str(y_train.shape))
print("Validation Dataset Shapes: " + str(x_val.shape) + ", " + str(y_val.shape))
```

Figure 6.13 Split data into training and validation

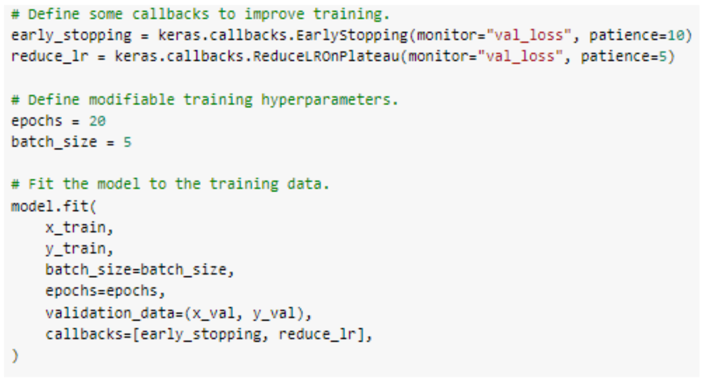
We create a model which take as input movies of shape (n_frames, width, height, channels) and returns a movie of identical shape. We used one input layer, three ConvLSTM2D layers, and one Conv3D layer.



```
# Construct the input layer with no definite frame size.
inp = layers.Input(shape=(None, *x_train.shape[2:]))

# We will construct 3 `ConvLSTM2D` layers with batch normalization,
# followed by a `Conv3D` layer for the spatiotemporal outputs.
x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(5, 5),
    padding="same",
    return_sequences=True,
    activation="relu",
)(inp)
x = layers.BatchNormalization()(x)
x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(3, 3),
    padding="same",
    return_sequences=True,
    activation="relu",
)(x)
x = layers.BatchNormalization()(x)
x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(1, 1),
    padding="same",
    return_sequences=True,
    activation="relu",
)(x)
x = layers.Conv3D(
    filters=3, kernel_size=(3, 3, 3), activation="sigmoid", padding="same"
)(x)

model = keras.models.Model(inp, x)
model.compile(
    loss=keras.losses.binary_crossentropy, optimizer=keras.optimizers.Adam(),
)
```

Figure 6.14 Building a model

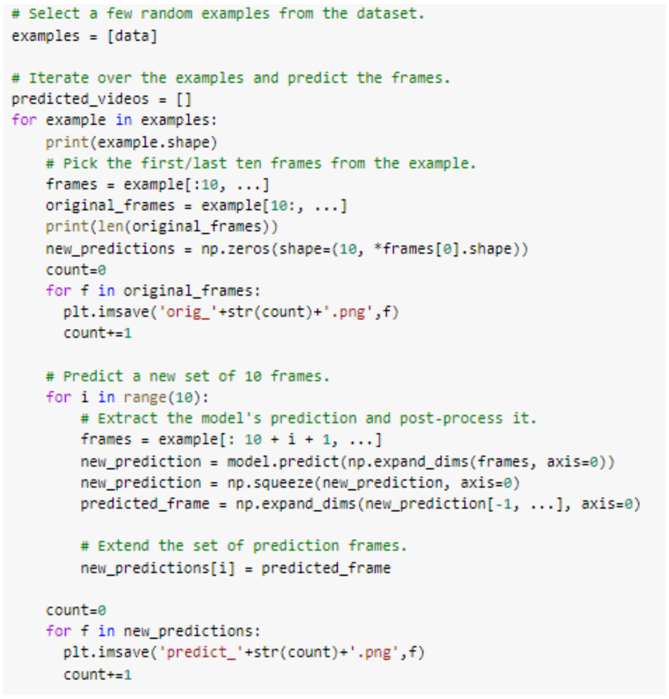
And we trained our model with 20 epochs.



```
# Define some callbacks to improve training.
early_stopping = keras.callbacks.EarlyStopping(monitor="val_loss", patience=10)
reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor="val_loss", patience=5)

# Define modifiable training hyperparameters.
epochs = 20
batch_size = 5

# Fit the model to the training data.
model.fit(
    x_train,
    y_train,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=(x_val, y_val),
    callbacks=[early_stopping, reduce_lr],
)
```

Figure 6.15 Training the model

Then we predicted the next 10 frames using the first 10 frames as shown in the Figure-6.16.



```
# Select a few random examples from the dataset.
examples = [data]

# Iterate over the examples and predict the frames.
predicted_videos = []
for example in examples:
    print(example.shape)
    # Pick the first/last ten frames from the example.
    frames = example[:10, ...]
    original_frames = example[10:, ...]
    print(len(original_frames))
    new_predictions = np.zeros(shape=(10, *frames[0].shape))
    count=0
    for f in original_frames:
      plt.imsave('orig_'+str(count)+'.png',f)
      count+=1

    # Predict a new set of 10 frames.
    for i in range(10):
        # Extract the model's prediction and post-process it.
        frames = example[: 10 + i + 1, ...]
        new_prediction = model.predict(np.expand_dims(frames, axis=0))
        new_prediction = np.squeeze(new_prediction, axis=0)
        predicted_frame = np.expand_dims(new_prediction[-1, ...], axis=0)

        # Extend the set of prediction frames.
        new_predictions[i] = predicted_frame

    count=0
    for f in new_predictions:
      plt.imsave('predict_'+str(count)+'.png',f)
      count+=1
```

Figure 6.16 Testing the model on one movie

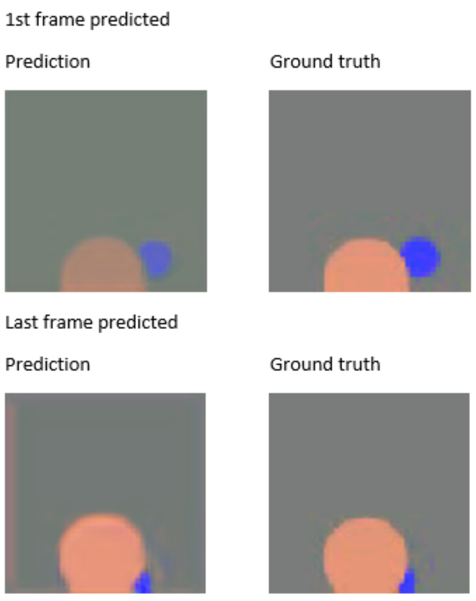
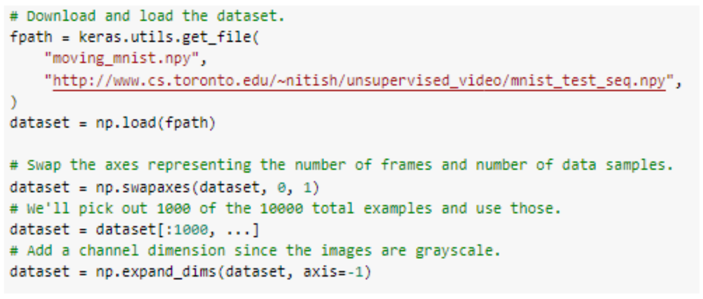


Figure 6.17 Example of predicted frames

6.1.2.2 Moving Mnist Dataset

Firstly, we imported the necessary libraries as shown in Figure-6.11.We downloaded and loaded the moving mnist dataset into an array.



```
# Download and load the dataset.
fpath = keras.utils.get_file(
    "moving_mnist.npy",
    "http://www.cs.toronto.edu/~nitish/unsupervised_video/mnist_test_seq.npy",
)
dataset = np.load(fpath)

# Swap the axes representing the number of frames and number of data samples.
dataset = np.swapaxes(dataset, 0, 1)
# We'll pick out 1000 of the 10000 total examples and use those.
dataset = dataset[:1000, ...]
# Add a channel dimension since the images are grayscale.
dataset = np.expand_dims(dataset, axis=-1)
```

Figure 6.18 Download and load moving mnist dataset

Then for training the model with moving mnist dataset and predicting, we used the same steps in Figure-6.13, Figure-6.14, Figure-6.15, and Figure-6.16.

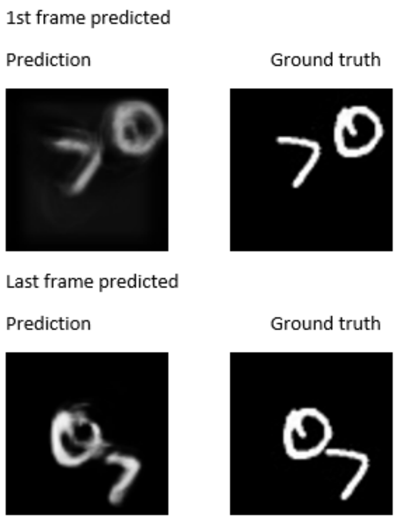


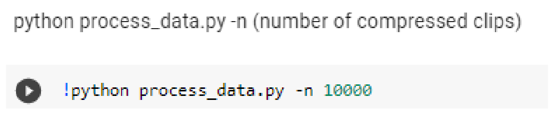
Figure 6.19 Example of predicted frames

6.2 Next frame prediction by multi-scale GAN

A multiscale approach is taken. We train a combination of Generators and Discriminators at different scales so that they learn internal representations at various scales. The estimate at a lower scale is used as an input for the network at a higher scale.

We downloaded the repository, but we could not run it directly. Since the project dates back 3 years, there were some library differences. First of all, we fixed these problems. Then, we preprocessed the data set to train the model with the data set we created.

The network trains on random 32x32 pixel crops of the input images, filtered to make sure that most clips have some movement in them. To process our input data into this form, we run the script python process_data from the directory. We built around 10000 compressed clips.

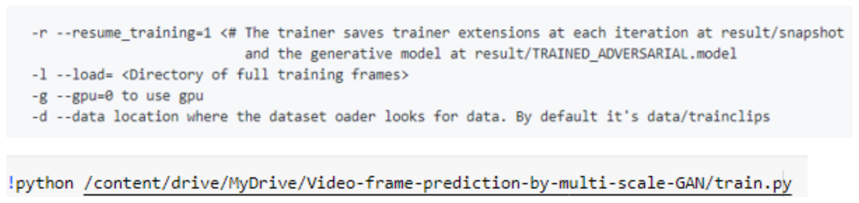


python process_data.py -n (number of compressed clips)

```
!python process_data.py -n 10000
```

Figure 6.20 Process data

To train with the default values simple you can run train.py with the following optional arguments. We used default values.

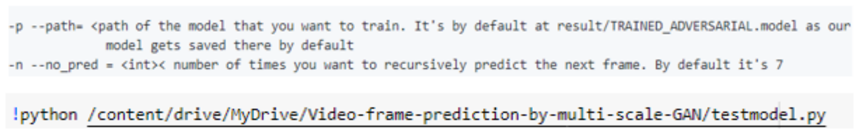


```
-r --resume_training=1 <# The trainer saves trainer extensions at each iteration at result/snapshot
                          and the generative model at result/TRAINED_ADVERSARIAL.model
-l --load= <Directory of full training frames>
-g --gpu=0 to use gpu
-d --data location where the dataset oader looks for data. By default it's data/trainclips
```

```
!python /content/drive/MyDrive/Video-frame-prediction-by-multi-scale-GAN/train.py
```

Figure 6.21 Train data

To see how our network performs, we run testmodel.py. It saves the result of how our model behaves in a new inference folder. It takes in two optional arguments.



```
-p --path= <path of the model that you want to train. It's by default at result/TRAINED_ADVERSARIAL.model as our
            model gets saved there by default
-n --no_pred = <int>< number of times you want to recursively predict the next frame. By default it's 7
```

```
!python /content/drive/MyDrive/Video-frame-prediction-by-multi-scale-GAN/testmodel.py
```

Figure 6.22 Test model

Prediction results and ground truth images are given in the Figure-6.23.

We also trained the model using the moving mnist dataset, but the pixels of the predicted frames always took the value 0.

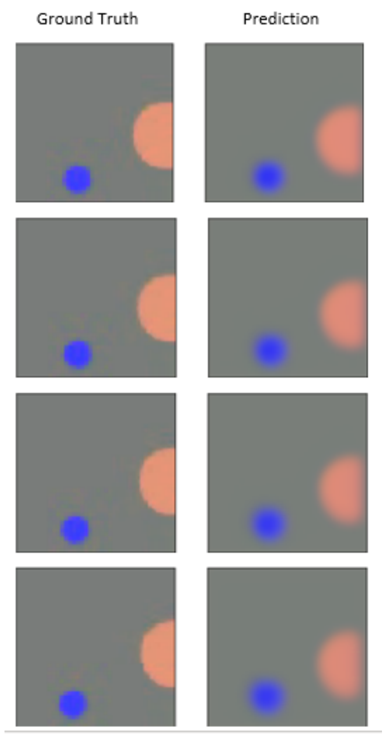


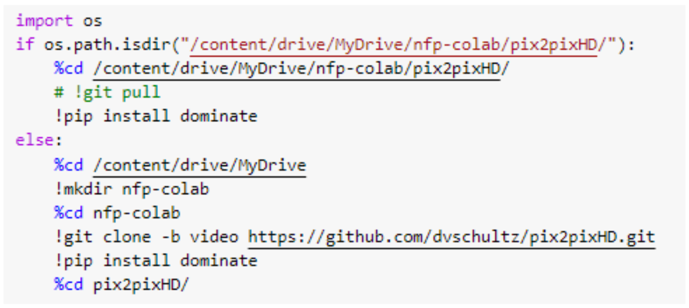
Figure 6.23 Prediction results

6.3 Pix2pix

6.3.1 Circle Dataset

The model generates video by recursively generating images with pix2pix.

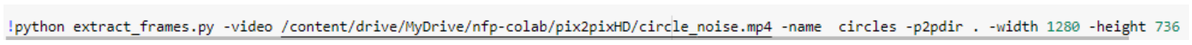
Firstly we cloned the pix2pix repository[19].



```
import os
if os.path.isdir("/content/drive/MyDrive/nfp-colab/pix2pixHD/"):
    %cd /content/drive/MyDrive/nfp-colab/pix2pixHD/
    # !git pull
    !pip install dominate
else:
    %cd /content/drive/MyDrive
    !mkdir nfp-colab
    %cd nfp-colab
    !git clone -b video https://github.com/dvschultz/pix2pixHD.git
    !pip install dominate
    %cd pix2pixHD/
```

Figure 6.24 Clone pix2pix

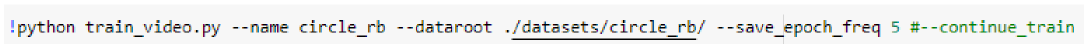
Then we used the code block in the Figure-6.25 to extract frames from videos.



```
!python extract_frames.py -video /content/drive/MyDrive/nfp-colab/pix2pixHD/circle_noise.mp4 -name  circles -p2pdir . -width 1280 -height 736
```

Figure 6.25 Extract frames

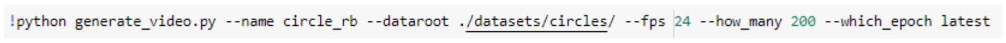
After extracting frames we trained our model with 50, 100 and 200 epochs.



```
!python train_video.py --name circle_rb --dataroot ./datasets/circle_rb/ --save_epoch_freq 5 #--continue_train
```

Figure 6.26 Training the model

Using the trained model, we ran the generate_video.py file to predict the next frame.



```
!python generate_video.py --name circle_rb --dataroot ./datasets/circles/ --fps 24 --how_many 200 --which_epoch latest
```

Figure 6.27 Testing the model

Prediction results and ground truth images are given in the Figure-6.28.

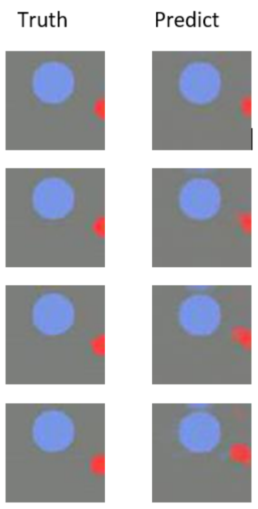
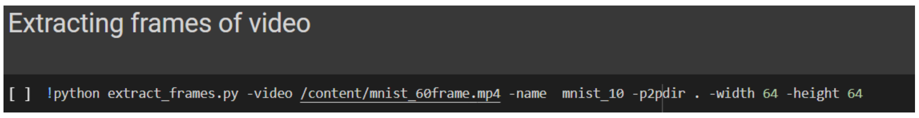


Figure 6.28 Prediction results

6.3.2 Moving Mnist Dataset

For Moving Mnist Dataset after connecting to drive and going to right directory we extract frames of the video we used for the model. While extracting the video we give values of frames, which is 64*64 for Moving Mnsit Dataset as shown in the Figure-6.29.

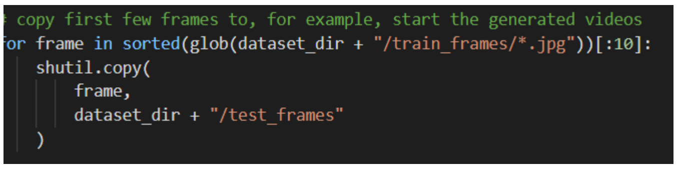


Extracting frames of video

```
[ ]  !python extract_frames.py -video /content/mnist_60frame.mp4 -name  mnist_10 -p2pdir . -width 64 -height 64
```

Figure 6.29 Pix2pix Extracting Frames of Moving Mnsit

While extracting frames we also made a change in "extract_frames.py" , normally pix2pix model uses 60 frames for test set but because of every video of this dataset has 20 frames each we changed size of test set to 10 frames.



```
# copy first few frames to, for example, start the generated videos
for frame in sorted(glob(dataset_dir + "/train_frames/*.jpg"))[:10]:
    shutil.copy(
        frame,
        dataset_dir + "/test_frames"
    )
```

Figure 6.30 Changing Test Set Size

After creating train and test frames we train model as shown in the Figure-6.28. We train our model for 50, 100, 200 epochs in one go and saved checkpoints and generate videos with using those checkpoints like shown in the Figure-6.27 with changing "which_epoch" value to number of epochs.

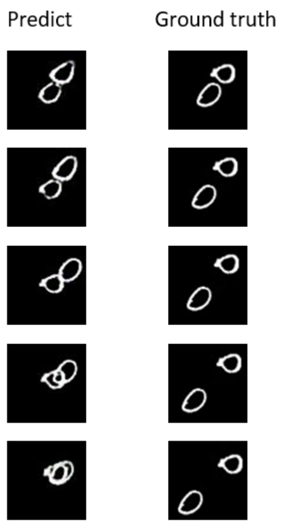


Figure 6.31 Example of predicted frames

7
Experimental Results

In this section, the results of the models trained with the circle and moving mnist datasets are mentioned.

7.1 Conv_lstm

For conv_lstm we trained two different model with circle and moving mnist dataset and the results follow as:

7.1.1 Circle Dataset

We tested the model we mentioned earlier in the Section-6.1.1.1 with the circle dataset. We used 10 frames as input and predicted 10 frames.

There is a graph that shows the similarity and RMSE values between ground truth and predicted frame for the 10 predicted frames in the Figure-7.1.

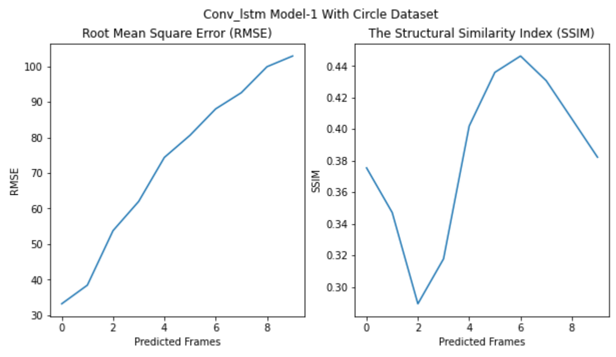


Figure 7.1 Conv_lstm model1 application results for circle data

For conv_lstm model-2 that we mentioned earlier in the Section-6.1.2.1 we used 10 frames as input and predicted 10 frames.

The graph in the Figure-7.1 shows the similarity and RMSE values between ground truth and predicted frame for the 10 predicted frames.

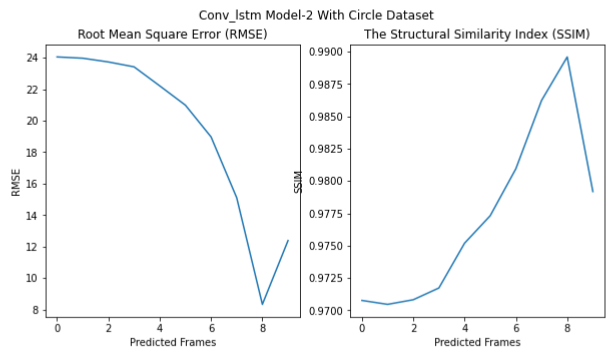


Figure 7.2 Conv_lstm model2 application results for circle data

7.1.2 Moving Mnist Dataset

For the model in the Section-6.1.1.2 we predicted 10 frames using moving mnist data. The similarity and RMSE values between ground truth and predicted frame for the 10 predicted frames are shown in the Figure-7.3.

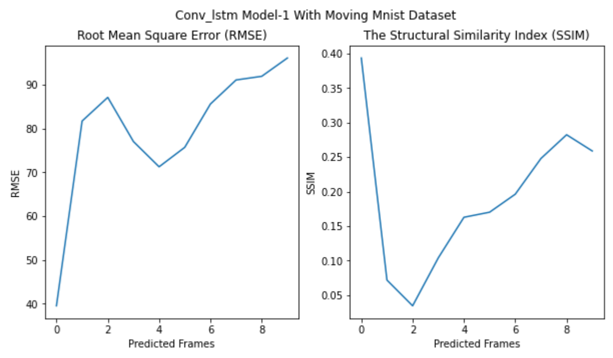


Figure 7.3 Conv_lstm model1 application results for moving mnist data

For conv_lstm model-2 in the Section-6.1.2.2 we used 10 frames as input and predicted 10 frames with using moving mnist data.

The graph in the Figure-7.4 shows the similarity and RMSE values between ground truth and predicted frame for the 10 predicted frames.

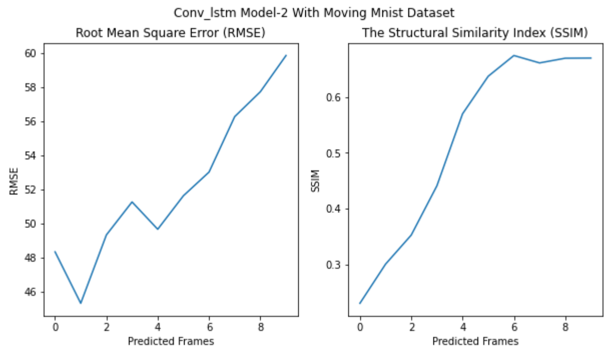


Figure 7.4 Conv_lstm model2 application results for moving mnist data

7.2 Multiscale GAN

For the multiscale gan model, we made frame predicts for both the circle dataset and the moving mnist dataset. However, when we used the moving mnist dataset for prediction, pixels of the predicted frames took the value 0 so we only included the circle dataset predicted results.

Since 4 frames are given as input with the Multiscale GAN model, we estimated 10 frames using 4 frames.

There is a graph that shows the similarity and RMSE values between ground truth and predicted frame for the 10 predicted frames in the Figure-7.5.

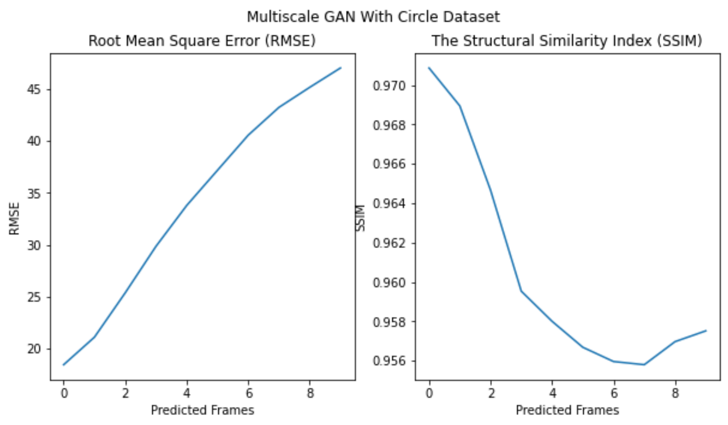


Figure 7.5 Multiscale GAN application results for circle data

7.3 Pix2Pix

For pix2pix we trained the model for 50, 100 and 200 epochs and the results follow as:

7.3.1 Circle Dataset

The every video in circle dataset has 96 frames and their size is 80*80 pixels. For using in the pix2pix model we extract 1 video and used 60 frames for training, 60 frames for starter for test set and 10 frames for testing.
The results for 50 epoch is shown in the Figure- 7.6.

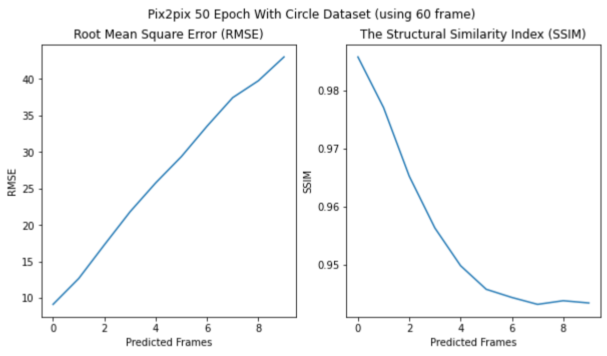


Figure 7.6 Pix2pix application results for circle data and 50 epoch

The results for 100 epoch is shown in the Figure- 7.7.

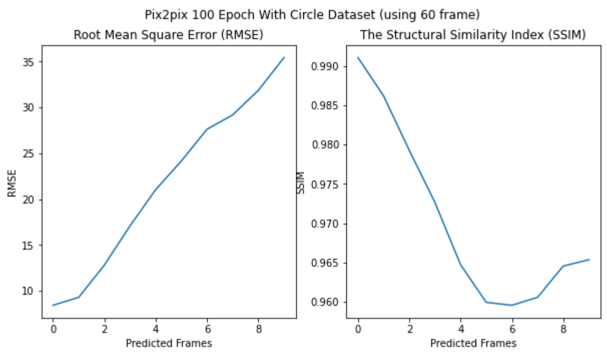


Figure 7.7 Pix2pix application results for circle data and 100 epoch

The results for 200 epoch is shown in the Figure- 7.8.

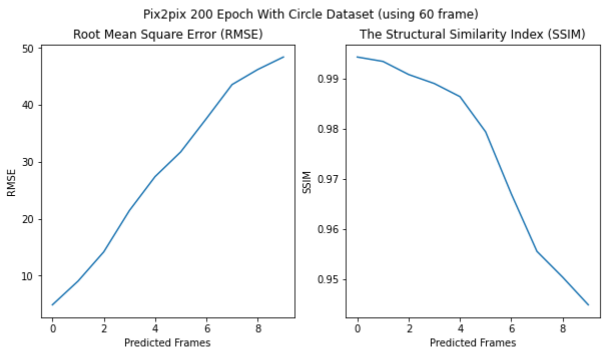


Figure 7.8 Pix2pix application results for circle data and 200 epoch

There is a graph that shows the similarity and RMSE values between ground truth and predicted frame for the first 10 predicted frames in the Figure-7.9.

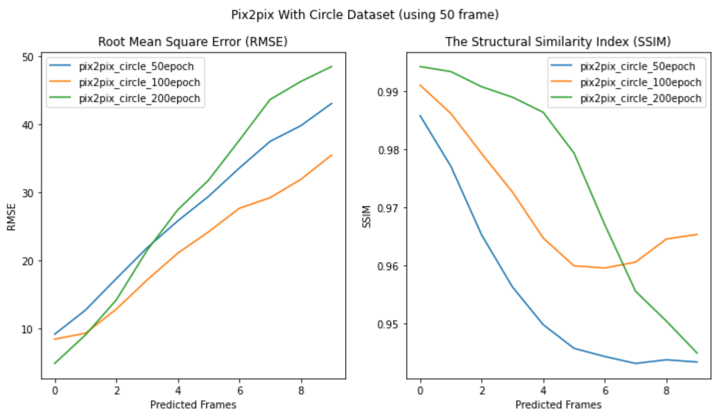


Figure 7.9 Pix2pix application results for circle data

7.3.2 Moving Mnist Dataset

Every video in moving mnist dataset has 20 frames so as mentioned earlier we used both 20 frames version and 60 frames version we created. For 20 frames version we used 10 frames for train, 10 frames for starter of test set and 10 frames for testing while for 60 frames version we used 50 frames for training, 50 frames for starter of test set and 10 frames for testing.

7.3.3 Moving Mnist Dataset-20 Frames

The results for 50 epoch in 20 frame version is shown in the Figure- 7.10.

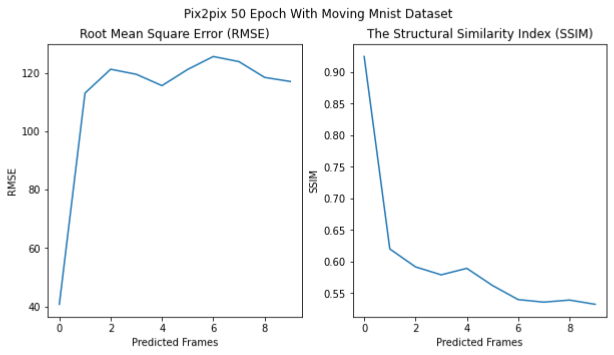


Figure 7.10 Pix2pix application results for mnist data and 50 epoch

The results for 100 epoch in 20 frame version is shown in the Figure- 7.11.

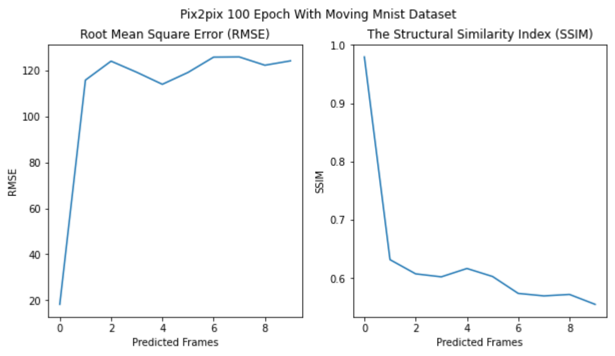


Figure 7.11 Pix2pix application results for mnist data and 100 epoch

The results for 200 epoch in 20 frame version is shown in the Figure- 7.12.

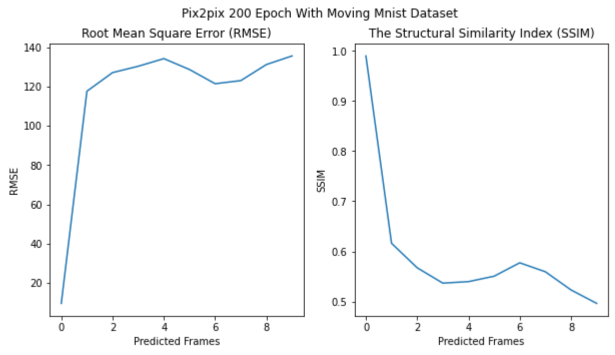


Figure 7.12 Pix2pix application results for mnist data and 200 epoch

There is a graph that shows the similarity and RMSE values between ground truth and predicted frame for the first 10 predicted frames in the Figure-7.13.

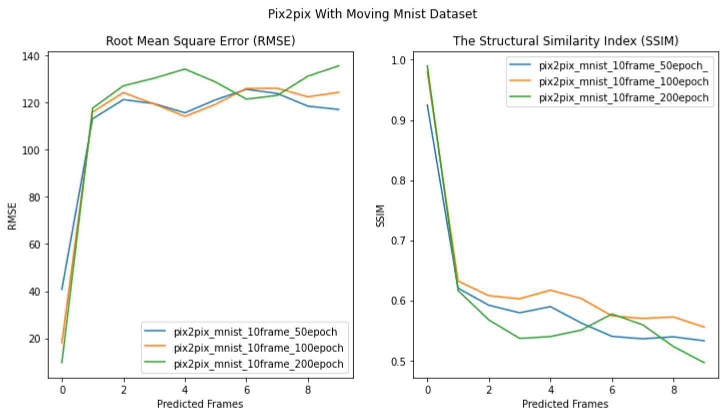


Figure 7.13 Pix2pix application results for moving mnsit data-20 frames

7.3.3.1 Moving Mnist Dataset-60 Frames

The results for 50 epoch in 60 frames version is shown in the Figure- 7.14.

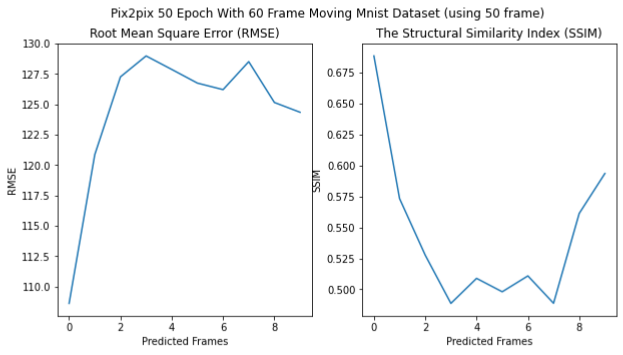


Figure 7.14 Pix2pix application results for mnist data and 50 epoch

The results for 100 epoch in 60 frame version is shown in the Figure- 7.15.

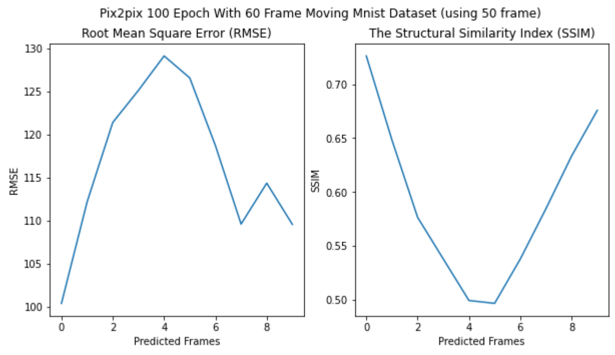


Figure 7.15 Pix2pix application results for mnist data and 100 epoch

The results for 200 epoch in 60 frame version is shown in the Figure- 7.16.

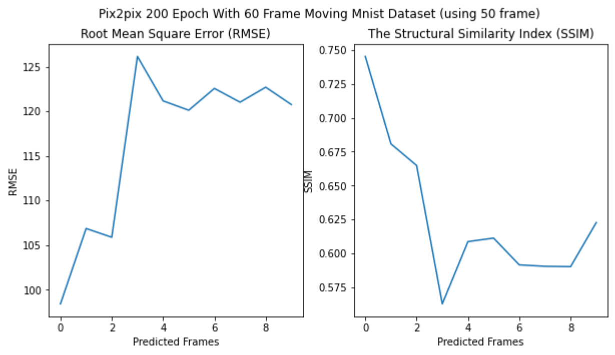


Figure 7.16 Pix2pix application results for mnist data and 200 epoch

There is a graph that shows the similarity and RMSE values between ground truth and predicted frame for the first 10 predicted frames in the Figure-7.17.

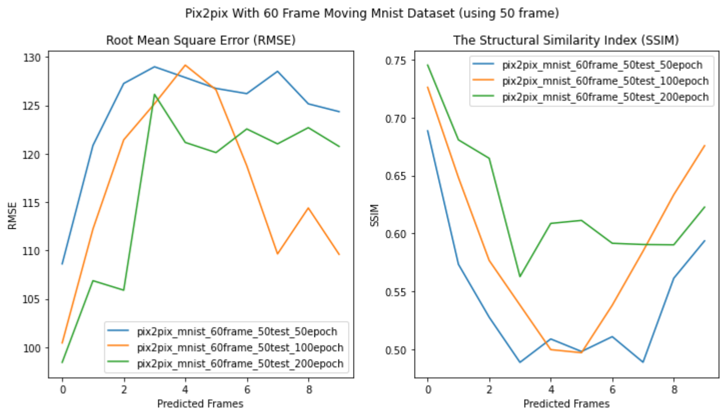


Figure 7.17 Pix2pix application results for moving mnsit data-60 frames

8 Performance Analysis

In this chapter, the performances of every model we trained in both of performance metrics we used for our project and our comments of the results are mentioned.

8.1 Moving Mnist Dataset

For Moving Mnsit dataset we trained 2 conv_lstm model and pix2pix model with different size of sets and epochs. In the Table-8.1 RMSE values of models we trained with Moving Mnist are given.
If we look at results for the first frames pix2pix model that trained with 10 frames and for 200 epoch is the most successful but if we look at overall success conv_lstm model 2 is the best one.
While looking at result we can say worst ones are pix2pix that trained with 50 frames but normally we expected them to be better than pix2pix models that are trained with 10 frames, the purpose of training the model with more frames was to overcome overfitting if there is an overfitting but we see that training with more frames didn't work for this model.

Table 8.1 Moving Mnist RMSE Values

Model	Frame 1	Frame 2	Frame 4	Frame 6	Frame 8	Frame 10
Conv_lstm model1	39.53	81.72	77.04	75.71	91.09	96.11
Conv_lstm model2	48.32	45.30	51.26	51.62	56.27	59.87
Pix2Pix 20 Frame 50 epoch	40.79	113.04	119.47	121.13	123.81	117.02
Pix2Pix 20 Frame 100 epoch	18.20	115.91	119.33	119.24	126.00	124.30
Pix2Pix 20 Frame 200 epoch	9.71	117.58	130.27	128.62	122.97	135.52
Pix2Pix 60 Frame 50 epoch	108.61	120.86	128.99	126.75	128.52	124.35
Pix2Pix 60 Frame 100 epoch	100.43	112.21	125.16	126.61	109.64	109.60
Pix2Pix 60 Frame 200 epoch	98.42	106.87	126.14	120.12	121.02	120.75

In the Table-8.4 SSIM values of models we trained with Moving Mnist are given.
If we look at the results unlike RMSE values pix2pix model trained with 10 frames are better than any other model. Pix2pix model are visibly way more better than

conv_lstm models. The reason might be because of SSIM is a similarity metric that compare the images in human way of looking.

Table 8.2 Moving Mnist SSIM Values

Model	Frame 1	Frame 2	Frame 4	Frame 6	Frame 8	Frame 10
Conv_lstm model1	0.39	0.07	0.10	0.17	0.24	0.25
Conv_lstm model2	0.23	0.30	0.44	0.63	0.66	0.66
Pix2Pix 20 Frame 50 epoch	0.92	0.62	0.57	0.56	0.53	0.53
Pix2Pix 20 Frame 100 epoch	0.97	0.63	0.60	0.60	0.56	0.55
Pix2Pix 20 Frame 200 epoch	0 .98	0.61	0.53	0.55	0.55	0.49
Pix2Pix 60 Frame 50 epoch	0.68	0.57	0.48	0.49	0.48	0.59
Pix2Pix 60 Frame 100 epoch	0.72	0.64	0.53	0.49	0.58	0.67
Pix2Pix 60 Frame 200 epoch	0.74	0.68	0.56	0.61	0.59	0.62

In the Figure-8.1 examples of predicted frames are given.

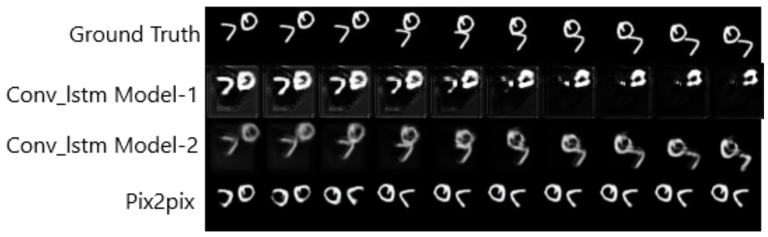


Figure 8.1 Predicted frames of moving mnist dataset

8.2 Circle Dataset

For circle dataset we trained 2 conv_lstm models, multiscale gan and pix2pix model with different epochs. In the Table - 8.3 RMSE values of models we trained with circle dataset that we created are given.
If we look at results for the first frames pix2pix model that trained with 60 frames and for 200 epoch is the most successful but if we look at overall success conv_lstm model 2 is the best one.
While in pix2pix model in every frame the accuracy is getting lower but in the conv_lstm model2 accuracy getting higher and when we look at the predicted images we see that it is because of in the fist frames color is more blurry and it gets better with time but the movements are true so if we consider that we can say conv_lstm model2 is better than other models.
In general pix2pix is good at predicting the first frames.

Table 8.3 Circle Dataset RMSE Values

Model	Frame 1	Frame 2	Frame 4	Frame 6	Frame 8	Frame 10
Conv_lstm model1	33.24	38.47	62.00	80.59	92.57	102.87
Conv_lstm model2	24.04	23.96	23.41	20.98	15.10	12.38
Multiscale GAN	18.44	21.09	29.84	37.17	43.23	47.03
Pix2pix 50 epoch	9.15	12.69	21.82	29.38	37.44	43.01
Pix2pix 100 epoch	8.43	9.30	17.10	24.17	29.19	35.44
Pix2pix 200 epoch	4.87	9.05	21.44	31.76	43.59	48.43

In the Table-8.4 SSIM values of the models we trained with Circle dataset are given. when we look at results other than conv_lstm model1 other models have similarity values that are higher than 0,95 which is quite good.
For conv_lstm model2 the RMSE values was getting better but because of SSIM is calculated in gray level it is not the case in here.

Table 8.4 Circle Dataset SSIM Values

Model	Frame 1	Frame 2	Frame 4	Frame 6	Frame 8	Frame 10
Conv_lstm model1	0.37	0.34	0.31	0.43	0.43	0.38
Conv_lstm model2	0.97	0.97	0.97	0.97	0.98	0.97
Multiscale GAN	0.97	0.96	0.95	0.95	0.95	0.95
Pix2pix 50 epoch	0.98	0.97	0.95	0.94	0.94	0.94
Pix2pix 100 epoch	0.99	0.98	0.97	0.95	0.96	0.96
Pix2pix 200 epoch	0.99	0.99	0.98	0.97	0.95	0.94

In the Figure-8.2 and Figure-8.3 examples of predicted frames of circle dataset are given.

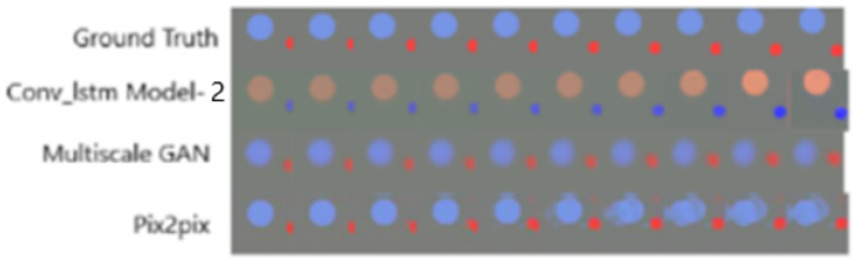


Figure 8.2 Predicted frames of circle dataset

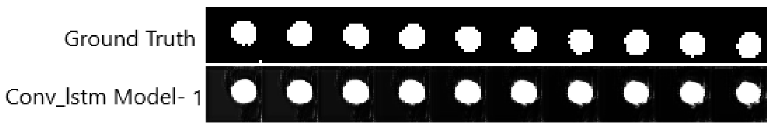


Figure 8.3 Conv_lstm model-1 predicted frames of circle dataset

9 Result

The aim of our project was to observe the success of the models by testing different models and different datasets for next frame prediction. We trained and tested the conv_lstm, multiscale gan and pix2pix models with the moving mnist dataset and the circle dataset we created.

In our study, we observed that the pix2pix model was more successful in the circle data set than the moving mnist data set.

When we look at the results of conv_lstm model1 and conv_lstm model2 for the circle data set, there is an increase in success in conv_lstm model2 contrary to expectations. However, when the two models are compared, conv_lstm model2 seems to be more successful. When we compare these two models for the moving mnist dataset, it is seen that although conv_lstm model1 predicts the first predicted frame more successfully, conv_lstm model2 is more successful when looking at the success until the last predicted frame.

We cannot make a comparison for the multiscale gan model as we do with the other models because the prediction results for the moving mnist dataset were not correct. However, it can be said that it is a successful model for the circle data set.

Considering the success of all models tried for the Moving mnist dataset, it can be said that the most successful one is conv_lstm model2.

Considering the success of all models tried for the Circle dataset, it can be said that the pix2pix model is quite successful in predicting the first frames, while the conv_lstm model2 is more successful in predicting the final frames.

When the studies[20][21] in the literature with the Moving mnist dataset were examined, it was observed that they gave more successful results than the models we tried. Since we do not have enough hardware resources to train with big data, we were able to train with 1000 videos while other studies were training with 10000 data.

We think that this is the biggest reason why the success is lower than other studies.

References

[1] T. Paparaju. (2018). “Video frame prediction with keras,” [Online]. Available: https://medium.com/machine-learning-basics/video-frame-prediction-with-keras-f74dd4743a1f.

[2] Y. Zhou, H. Dong, and A. El Saddik, “Deep learning in next-frame prediction: A benchmark review,” *IEEE Access*, vol. 8, pp. 69 273–69 283, 2020. DOI: 10.1109/ACCESS.2020.2987281.

[3] J. Testud. (2018). “Video generation with pix2pix,” [Online]. Available: https://medium.com/@jctestud/video-generation-with-pix2pix-aed5b1b69f57.

[4] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial networks*, 2014. arXiv: 1406.2661 [stat.ML].

[5] D. Nag. (2017). “Generative adversarial networks (gans) in 50 lines of code (pytorch),” [Online]. Available: https://medium.com/@devnag/generative-adversarial-networks-gans-in-50-lines-of-code-pytorch-e81b79659e3f.

[6] J. Brownlee. (2021). “How to develop a pix2pix gan for image-to-image translation,” [Online]. Available: https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/.

[7] M. Buyukkinaci. (2017). “Generative adversarial networks — gan nedir ? (türkçe),” [Online]. Available: https://medium.com/@muhammedbuyukkinaci/generative-adversarial-networks-gan-nedir.

[8] G. Developers. (2019). “Overview of gan structure,” [Online]. Available: https://developers.google.com/machine-learning/gan/gan_structure.

[9] T. Karras, T. Aila, S. Laine, and J. Lehtinen, “Progressive growing of gans for improved quality, stability, and variation,” *CoRR*, vol. abs/1710.10196, 2017. arXiv: 1710.10196. [Online]. Available: http://arxiv.org/abs/1710.10196.

[10] M. ul Hassan. (2018). “Pix2pix – image-to-image translation neural network,” [Online]. Available: https://neurohive.io/en/popular-networks/pix2pix-image-to-image-translation/.

[11] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” *CoRR*, vol. abs/1611.07004, 2016. arXiv: 1611.07004. [Online]. Available: http://arxiv.org/abs/1611.07004.

[12] N. Sinanoğlu. (2020). "Pix2pix gan for image-to-image translation," [Online]. Available: https://medium.com/devcist/pix2pix-gan-for-image-to-image-translation-7f650345fa3f#:~:text=Pix2Pix.

[13] A. Xavier. (2019). "An introduction to convlstm," [Online]. Available: https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7.

[14] A. Rosebrock. (2014). "How-to: Python compare two images," [Online]. Available: https://www.pyimagesearch.com/2014/09/15/python-compare-two-images/.

[15] T. Lee. (2021). "Moving mnist," [Online]. Available: https://gist.github.com/tencia/afb129122a64bde3bd0c.

[16] A. K. Bishoyi. (2018). "Video frame prediction by multi scale gan," [Online]. Available: https://github.com/alokwhitewolf/Video-frame-prediction-by-multi-scale-GAN.

[17] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, "High-resolution image synthesis and semantic manipulation with conditional gans," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[18] A. Joshi. (2021). "Next-frame video prediction with convolutional lstms," [Online]. Available: https://keras.io/examples/vision/conv_lstm/.

[19] D. Schultz. (2018). "Pix2pixhd," [Online]. Available: https://github.com/dvschultz/pix2pixHD.

[20] Z. Xu, Y. Wang, M. Long, and J. Wang, "Predcnn: Predictive learning with cascade convolutions," Jul. 2018, pp. 2940–2947. DOI: 10.24963/ijcai.2018/408.

[21] S. Oprea, P. Martinez-Gonzalez, A. Garcia-Garcia, J. Castro-Vargas, S. Orts-Escolano, J. Garcia-Rodriguez, and A. Argyros, "A review on deep learning techniques for video prediction," *IEEE Transactions on Pattern Analysis Machine Intelligence*, no. 01, pp. 1–1, Dec. 5555, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2020.3045007.

Curriculum Vitae

FIRST MEMBER

Name-Surname: Büşra KÜDEN
Birthdate and Place of Birth: 11.08.1998, Porto
E-mail: busrakuden@gmail.com
Phone: 0545 905 43 35
Practical Training: TUBITAK BTE
Oredata Yazılım Limited Şirketi

SECOND MEMBER

Name-Surname: Elif ÖZCAN
Birthdate and Place of Birth: 12.08.1999, İstanbul
E-mail: elfozcn08@gmail.com
Phone: 0538 018 24 90
Practical Training: TUBITAK BTE B3LAB

Project System Informations

System and Software: Windows Operating, Python
Required RAM: 8GB
Required Disk: 256GB