Introduction

The Convolutional LSTM architectures bring together time series processing and computer vision by introducing a convolutional recurrent cell in a LSTM layer. In this example, we will explore the Convolutional LSTM model in an application to next-frame prediction, the process of predicting what video frames come next given a series of past frames.

1. ssh -L 8080:localhost:8080 ENEE439D_GIS@10.229.55.68
2. enter password
3. bash
4. conda activate cloudEnv
5. jupyter notebook --port=8080

Setup

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import numpy as np
import gc
import matplotlib.pyplot as plt

import tensorflow as tf
import keras
from keras import layers

import io
import imageio
from IPython.display import Image, display
from ipywidgets import widgets, Layout, HBox
```

Dataset Construction

For next-frame prediction, our model will be using a previous frame, which we'll call `f_n`, to predict a new frame, called `f_(n + 1)`. To allow the model to create these predictions, we'll need to process the data such that we have "shifted" inputs and outputs, where the input data is frame `x_n`, being used to predict frame `y_(n + 1)`.

```
import numpy as np
import os
import cv2
from sklearn.model_selection import train_test_split

# def ensure_dir(file_path):
#     directory = os.path.dirname(file_path)
#     if not os.path.exists(directory):
#         os.makedirs(directory)

# Define the directory to save processed data
data_dir = '/content/drive/MyDrive/ENEE439D Group Project/Cloud Predictions/LSTM Model'
# ensure_dir(data_dir)  # Ensure the directory exists

# Function to save processed data
def save_data(data, filename):
    np.save(os.path.join(data_dir, filename), data)

# Function to load processed data
def load_data(filename):
    return np.load(os.path.join(data_dir, filename))

# # Processing images if not already saved
# images_file = os.path.join(data_dir, 'images.npy')
# if not os.path.exists(images_file):
#     images = []
#     pngs_dir = "/content/drive/MyDrive/ENEE439D Group Project/Cloud Data /PNGs"
#     for filename in sorted(os.listdir(pngs_dir)):
#         if filename.find("Masked") >= 0:
#             img_path = os.path.join(pngs_dir, filename)
#             img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
#             img = cv2.resize(img, (320, 160))
#             images.append(img)
#     images = np.array(images, dtype=np.float32) / 255.0
#     images = np.expand_dims(images, axis=-1)
#     save_data(images, 'images.npy')
# else:
images = load_data('images.npy')

# Function to create sequences from the images
# def create_shifted_frames(images, sequence_length=10):
#     x, y = [], []
#     for i in range(len(images) - sequence_length):
#         x.append(images[i:i + sequence_length])
#         y.append(images[i + 1:i + sequence_length + 1])
#     return np.array(x), np.array(y)

# Split and save data
#x, y = create_shifted_frames(images)
# x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.1, random_state=42)
# save_data(x_train, 'x_train.npy')
# save_data(x_val, 'x_val.npy')
# save_data(y_train, 'y_train.npy')
# save_data(y_val, 'y_val.npy')
#save_data(x, 'x.npy')
#save_data(y, 'y.npy')
# Load directly in future uses
x_train = load_data('x_train.npy')
x_val = load_data('x_val.npy')
y_train = load_data('y_train.npy')
y_val = load_data('y_val.npy')
x = load_data('x.npy')
train_dataset = x_train
val_dataset = x_val
print("Training Dataset Shapes:", x_train.shape, y_train.shape)
print("Validation Dataset Shapes:", x_val.shape, y_val.shape)
```

```
Training Dataset Shapes: (1791, 10, 160, 320, 1) (1791, 10, 160, 320, 1)
Validation Dataset Shapes: (200, 10, 160, 320, 1) (200, 10, 160, 320, 1)
```

```
# # Function to create sequences from the images
# def create_shifted_frames(images, sequence_length=10):
#     x, y = [], []
#     for i in range(len(images) - sequence_length):
#         x.append(images[i:i + sequence_length])
#         y.append(images[i + 1:i + sequence_length + 1])
#     return np.array(x), np.array(y)

# # Split data into train and validation sets
# x, y = create_shifted_frames(images)
# x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.1, random_state=42)
# train_dataset = x_train
# val_dataset = x_val
# print("Training Dataset Shapes: " + str(x_train.shape) + ", " + str(y_train.shape))
# print("Validation Dataset Shapes: " + str(x_val.shape) + ", " + str(y_val.shape))

# # Clear variables
# del images, x, y
# gc.collect()
```

Data Visualization

Our data consists of sequences of frames, each of which are used to predict the upcoming frame. Let's take a look at some of these sequential frames.

```
# # Construct a figure on which we will visualize the images.
# sequence_length = 10  # Define the sequence length if not already defined
# fig, axes = plt.subplots(4, 5, figsize=(10,8))  # Adjust the subplot layout

# # Plot each of the sequential images for one random data example.
# data_choice = np.random.choice(range(len(train_dataset)), size=1)[0]
# for idx, ax in enumerate(axes.flat):
#     if idx < sequence_length:  # Ensure we do not exceed the sequence length
#         ax.imshow(np.squeeze(train_dataset[data_choice][idx]), cmap="gray")
#         ax.set_title(f"Frame {idx + 1}")
#     ax.axis("off")

# # Print information and display the figure.
# print(f"Displaying frames for example {data_choice}.")
# plt.show()
```

Model Construction

To build a Convolutional LSTM model, we will use the `ConvLSTM2D` layer, which will accept inputs of shape `(batch_size, num_frames, width, height, channels)`, and return a prediction movie of the same shape.

```
# Add extra batch normalization in next training to perhaps

inp = layers.Input(shape=(None, *x_train.shape[2:]))

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.BatchNormalization()(x)
x = layers.Conv3D(
    filters=1, 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(),
)
```

Model Training

With our model and data constructed, we can now train the model.

```
checkpoint_path = '/content/drive/MyDrive/ENEE439D Group Project/Cloud Predictions/LSTM Model'
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create a callback that saves the model's weights
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                 save_weights_only=True,
                                                 verbose=1)


model.load_weights(checkpoint_path)
model.summary()
```

Show hidden output

```
# #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 = 10
# 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],
# )



# Save the weights
# model.save_weights('/content/drive/MyDrive/ENEE439D Group Project/Cloud Predictions/LSTM Model')

# checkpoint_path = '/content/drive/MyDrive/ENEE439D Group Project/Cloud Predictions/LSTM Model'
# checkpoint_dir = os.path.dirname(checkpoint_path)

# # Create a callback that saves the model's weights
# cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
#                                                  save_weights_only=True,
#                                                  verbose=1)
```

Frame Prediction Visualizations

With our model now constructed and trained, we can generate some example frame predictions based on a new video.

We'll pick a random example from the validation set and then choose the first ten frames from them. From there, we can allow the model to predict 10 new frames, which we can compare to the ground truth frame predictions.

```
# 12
example = val_dataset[48]
initial_frames = example[:5]
frames = np.copy(initial_frames)

print("Length of pre-process predicted frames:", len(frames))

for _ in range(5):

    input_frames = np.expand_dims(frames, axis=0)
    new_prediction = model.predict(input_frames)
    new_prediction = np.squeeze(new_prediction, axis=0)

    predicted_frame = np.expand_dims(new_prediction[-1], axis=0)
    frames = np.concatenate((frames[1:], predicted_frame), axis=0)  # Skip the first frame and add the new frame

    print("Length of predicted frames after appending new frame:", len(frames))

input_frames = example[0:5]
original_next_frames = example[5:10]
fig, axes = plt.subplots(3, 5, figsize=(25, 10))

# input 5 fraems
for idx, ax in enumerate(axes[0]):
    ax.imshow(np.squeeze(input_frames[idx]), cmap="gray")
    ax.set_title(f"Input Frame {idx + 1}")
    ax.axis("off")

# Original 5 frames
for idx, ax in enumerate(axes[1]):
    ax.imshow(np.squeeze(original_next_frames[idx]), cmap="gray")
    ax.set_title(f"Original Frame {idx + 6}")
    ax.axis("off")

# New 5 frames
predicted_frames = frames[-5:]
for idx, ax in enumerate(axes[2]):
    ax.imshow(np.squeeze(predicted_frames[idx]), cmap="gray")
    ax.set_title(f"Predicted Frame {idx + 6}")
    ax.axis("off")
plt.show()
```

```
Length of pre-process predicted frames: 5
1/1 [==============================] - 0s 29ms/step
Length of predicted frames after appending new frame: 5
1/1 [==============================] - 0s 27ms/step
```

```
# Select a random example from the validation dataset
example = val_dataset[np.random.choice(range(len(val_dataset)))]

# Initialize with the first 5 frames for prediction
```