**Backend R(2+1)D Training**

!pip install remotezip tqdm opencv-python einops

!pip install tensorflow==2.12.0 --user

!pip install tensorflow-docs

!pip install einops

import tqdm

import random

import pathlib

import itertools

import collections

import os

import cv2

import numpy as np

import remotezip as rz

import tensorflow as tf

# Some modules to display an animation using imageio.

import imageio

from IPython import display

from urllib import request

from tensorflow\_docs.vis import embed

import seaborn as sns

import matplotlib.pyplot as plt

import tensorflow as tf

import keras

from keras import layers

import einops

**Loading the DataSet**

dataset\_folder\_path = r"C:\Final Year Project\Real Life Violence Dataset"

import os

def list\_video\_files(dataset\_folder):

"""List the video files in each class of the dataset given a folder path.

Args:

dataset\_folder: Path to the dataset folder.

Returns:

List of video files in each of the classes.

"""

video\_files = []

for folder in ["fights", "nofights"]:

folder\_path = os.path.join(dataset\_folder, folder)

for root, dirs, filenames in os.walk(folder\_path):

for filename in filenames:

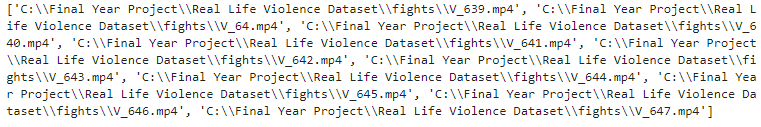
if filename.endswith('.mp4'):

video\_files.append(os.path.join(root, filename))

return video\_files

video\_files\_list = list\_video\_files(dataset\_folder\_path)

print(video\_files\_list[600:610]) # Printing the first 10 video files



def get\_class(fname):

"""Retrieve the name of the class given a filename.

Args:

fname: Name of the file in the Real Life Violence dataset.

Returns:

Class that the file belongs to.

"""

class\_name = os.path.basename(os.path.dirname(fname))

return class\_name

def get\_files\_per\_class(files):

"""Retrieve the files that belong to each class.

Args:

files: List of files in the dataset.

Returns:

Dictionary of class names (key) and files (values).

"""

files\_for\_class = collections.defaultdict(list)

for fname in files:

class\_name = get\_class(fname)

files\_for\_class[class\_name].append(fname)

return files\_for\_class

video\_files\_list = list\_video\_files(dataset\_folder\_path)

files\_for\_class = get\_files\_per\_class(video\_files\_list)

classes = list(files\_for\_class.keys())

print(classes)

print('Number of classes:', len(classes))

for class\_name in classes:

print('Number of videos for class', class\_name + ':', len(files\_for\_class[class\_name]))

**Splitting the Dataset into train , test and validation sets**

import shutil

import os

import random

def split\_dataset(files\_for\_class, train\_ratio=0.7, val\_ratio=0.15, test\_ratio=0.15):

"""

Split the dataset into train, validation, and test sets.

Args:

files\_for\_class: Dictionary of class names (key) and files (values).

train\_ratio: Ratio of the dataset to be used for training (default: 0.7).

val\_ratio: Ratio of the dataset to be used for validation (default: 0.15).

test\_ratio: Ratio of the dataset to be used for testing (default: 0.15).

Returns:

train\_set, val\_set, test\_set: Lists of video files for train, validation, and test sets.

"""

train\_set = []

val\_set = []

test\_set = []

for class\_name, files in files\_for\_class.items():

random.shuffle(files) # Shuffle files for each class

num\_files = len(files)

num\_train = int(train\_ratio \* num\_files)

num\_val = int(val\_ratio \* num\_files)

train\_set.extend(files[:num\_train])

val\_set.extend(files[num\_train:num\_train + num\_val])

test\_set.extend(files[num\_train + num\_val:])

return train\_set, val\_set, test\_set

# Define output folders relative to the dataset directory

train\_output\_folder = os.path.join(dataset\_folder\_path, "train")

val\_output\_folder = os.path.join(dataset\_folder\_path, "val")

test\_output\_folder = os.path.join(dataset\_folder\_path, "test")

# Function to save split files

def save\_split\_files(split\_set, output\_folder):

"""

Save the split files to the specified output folder.

Args:

split\_set: List of video files to be saved.

output\_folder: Path to the output folder where the files will be saved.

"""

for file\_path in split\_set:

# Extract the filename from the file path

file\_name = os.path.basename(file\_path)

# Determine the class based on the filename prefix

class\_name = "fights" if file\_name.startswith("V\_") else "nofights"

# Create the output folder if it doesn't exist

class\_output\_folder = os.path.join(output\_folder, class\_name)

os.makedirs(class\_output\_folder, exist\_ok=True)

# Copy or move the file to the output folder

shutil.copy(file\_path, os.path.join(class\_output\_folder, file\_name))

# Split the dataset

train\_set, val\_set, test\_set = split\_dataset(files\_for\_class)

# Create the "Real Life Violence Dataset subset" folder within "F:/FYP"

subset\_folder\_path = os.path.join("C:/Final Year Project", "Real Life Violence Dataset subset")

os.makedirs(subset\_folder\_path, exist\_ok=True)

# Save the splits to the "Real Life Violence Dataset subset" folder

save\_split\_files(train\_set, os.path.join(subset\_folder\_path, "train"))

save\_split\_files(val\_set, os.path.join(subset\_folder\_path, "val"))

save\_split\_files(test\_set, os.path.join(subset\_folder\_path, "test"))

print("Split files saved to 'Real Life Violence Dataset subset' folder.")

**Creating Tensorflow DataSet**

import cv2

import random

import numpy as np

import tensorflow as tf

from pathlib import Path

def format\_frames(frame, output\_size):

"""

Pad and resize an image from a video.

Args:

frame: Image that needs to be resized and padded.

output\_size: Pixel size of the output frame image.

Return:

Formatted frame with padding of specified output size.

"""

frame = tf.image.convert\_image\_dtype(frame, tf.float32)

frame = tf.image.resize\_with\_pad(frame, \*output\_size)

return frame

def frames\_from\_video\_file(video\_path, n\_frames, output\_size=(224, 224), frame\_step=15):

"""

Creates frames from each video file present for each category.

Args:

video\_path: File path to the video.

n\_frames: Number of frames to be created per video file.

output\_size: Pixel size of the output frame image.

Return:

An NumPy array of frames in the shape of (n\_frames, height, width, channels).

"""

# Read each video frame by frame

result = []

src = cv2.VideoCapture(str(video\_path))

video\_length = src.get(cv2.CAP\_PROP\_FRAME\_COUNT)

need\_length = 1 + (n\_frames - 1) \* frame\_step

if need\_length > video\_length:

start = 0

else:

max\_start = video\_length - need\_length

start = random.randint(0, max\_start + 1)

src.set(cv2.CAP\_PROP\_POS\_FRAMES, start)

# ret is a boolean indicating whether read was successful, frame is the image itself

ret, frame = src.read()

result.append(format\_frames(frame, output\_size))

for \_ in range(n\_frames - 1):

for \_ in range(frame\_step):

ret, frame = src.read()

if ret:

frame = format\_frames(frame, output\_size)

result.append(frame)

else:

result.append(np.zeros\_like(result[0]))

src.release()

result = np.array(result)[..., [2, 1, 0]]

return result

class FrameGenerator:

def \_\_init\_\_(self, path, n\_frames, training=False):

""" Returns a set of frames with their associated label.

Args:

path: Video file paths.

n\_frames: Number of frames.

training: Boolean to determine if training dataset is being created.

"""

self.path = Path(path)

self.n\_frames = n\_frames

self.training = training

self.class\_names = sorted(set(p.name for p in self.path.iterdir() if p.is\_dir()))

self.class\_ids\_for\_name = dict((name, idx) for idx, name in enumerate(self.class\_names))

def get\_files\_and\_class\_names(self):

video\_paths = list(self.path.glob('\*/\*.mp4')) # Change the extension if your videos are in mp4 format

classes = [p.parent.name for p in video\_paths]

return video\_paths, classes

def \_\_call\_\_(self):

video\_paths, classes = self.get\_files\_and\_class\_names()

pairs = list(zip(video\_paths, classes))

if self.training:

random.shuffle(pairs)

for path, name in pairs:

video\_frames = frames\_from\_video\_file(path, self.n\_frames)

label = self.class\_ids\_for\_name[name] # Encode labels

yield video\_frames, label

subset\_paths = {

"train": "C:/Final Year Project/Real Life Violence Dataset subset/train",

"val": "C:/Final Year Project/Real Life Violence Dataset subset/val",

"test": "C:/Final Year Project/Real Life Violence Dataset subset/test"

}

fg = FrameGenerator(subset\_paths['train'], 10, training=True)

frames, label = next(fg())

print(f"Shape: {frames.shape}")

print(f"Label: {label}")

n\_frames = 10

batch\_size = 8

output\_signature = (tf.TensorSpec(shape = (None, None, None, 3), dtype = tf.float32),

tf.TensorSpec(shape = (), dtype = tf.int16))

train\_ds = tf.data.Dataset.from\_generator(FrameGenerator(subset\_paths['train'], n\_frames, training=True),

output\_signature = output\_signature)

# Batch the data

train\_ds = train\_ds.batch(batch\_size)

val\_ds = tf.data.Dataset.from\_generator(FrameGenerator(subset\_paths['val'], n\_frames),

output\_signature = output\_signature)

val\_ds = val\_ds.batch(batch\_size)

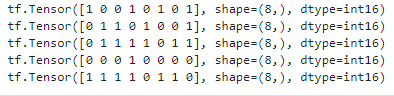
test\_ds = tf.data.Dataset.from\_generator(FrameGenerator(subset\_paths['test'], n\_frames),

output\_signature = output\_signature)

test\_ds = test\_ds.batch(batch\_size)

for frames, labels in train\_ds.take(5):

print(labels)



# Print the shapes of the data

train\_frames, train\_labels = next(iter(train\_ds))

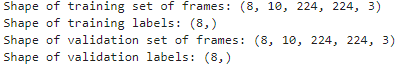
print(f'Shape of training set of frames: {train\_frames.shape}')

print(f'Shape of training labels: {train\_labels.shape}')

val\_frames, val\_labels = next(iter(val\_ds))

print(f'Shape of validation set of frames: {val\_frames.shape}')

print(f'Shape of validation labels: {val\_labels.shape}')



**Model Designing**

# Define the dimensions of one frame in the set of frames created

HEIGHT = 224

WIDTH = 224

class Conv2Plus1D(keras.layers.Layer):

def \_\_init\_\_(self, filters, kernel\_size, padding):

"""

A sequence of convolutional layers that first apply the convolution operation over the

spatial dimensions, and then the temporal dimension.

"""

super().\_\_init\_\_()

self.seq = keras.Sequential([

# Spatial decomposition

layers.Conv3D(filters=filters,

kernel\_size=(1, kernel\_size[1], kernel\_size[2]),

padding=padding),

# Temporal decomposition

layers.Conv3D(filters=filters,

kernel\_size=(kernel\_size[0], 1, 1),

padding=padding)

])

def call(self, x):

return self.seq(x)

class ResidualMain(keras.layers.Layer):

"""

Residual block of the model with convolution, layer normalization, and the

activation function, ReLU.

"""

def \_\_init\_\_(self, filters, kernel\_size):

super().\_\_init\_\_()

self.seq = keras.Sequential([

Conv2Plus1D(filters=filters,

kernel\_size=kernel\_size,

padding='same'),

layers.LayerNormalization(),

layers.ReLU(),

Conv2Plus1D(filters=filters,

kernel\_size=kernel\_size,

padding='same'),

layers.LayerNormalization()

])

def call(self, x):

return self.seq(x)

class Project(keras.layers.Layer):

"""

Project certain dimensions of the tensor as the data is passed through different

sized filters and downsampled.

"""

def \_\_init\_\_(self, units):

super().\_\_init\_\_()

self.seq = keras.Sequential([

layers.Dense(units),

layers.LayerNormalization()

])

def call(self, x):

return self.seq(x)

def add\_residual\_block(input, filters, kernel\_size):

"""

Add residual blocks to the model. If the last dimensions of the input data

and filter size does not match, project it such that last dimension matches.

"""

out = ResidualMain(filters,

kernel\_size)(input)

res = input

# Using the Keras functional APIs, project the last dimension of the tensor to

# match the new filter size

if out.shape[-1] != input.shape[-1]:

res = Project(out.shape[-1])(res)

return layers.add([res, out])

class ResizeVideo(keras.layers.Layer):

def \_\_init\_\_(self, height, width):

super().\_\_init\_\_()

self.height = height

self.width = width

self.resizing\_layer = layers.Resizing(self.height, self.width)

def call(self, video):

"""

Use the einops library to resize the tensor.

Args:

video: Tensor representation of the video, in the form of a set of frames.

Return:

A downsampled size of the video according to the new height and width it should be resized to.

"""

# b stands for batch size, t stands for time, h stands for height,

# w stands for width, and c stands for the number of channels.

old\_shape = einops.parse\_shape(video, 'b t h w c')

images = einops.rearrange(video, 'b t h w c -> (b t) h w c')

images = self.resizing\_layer(images)

videos = einops.rearrange(

images, '(b t) h w c -> b t h w c',

t = old\_shape['t'])

return videos

**Model**

input\_shape = (None, 10, HEIGHT, WIDTH, 3)

input = layers.Input(shape=(input\_shape[1:]))

x = input

x = Conv2Plus1D(filters=16, kernel\_size=(3, 7, 7), padding='same')(x)

x = layers.BatchNormalization()(x)

x = layers.ReLU()(x)

x = ResizeVideo(HEIGHT // 2, WIDTH // 2)(x)

# Block 1

x = add\_residual\_block(x, 16, (3, 3, 3))

x = ResizeVideo(HEIGHT // 4, WIDTH // 4)(x)

# Block 2

x = add\_residual\_block(x, 32, (3, 3, 3))

x = ResizeVideo(HEIGHT // 8, WIDTH // 8)(x)

# Block 3

x = add\_residual\_block(x, 64, (3, 3, 3))

x = ResizeVideo(HEIGHT // 16, WIDTH // 16)(x)

# Block 4

x = add\_residual\_block(x, 128, (3, 3, 3))

x = layers.GlobalAveragePooling3D()(x)

x = layers.Flatten()(x)

x = layers.Dense(10)(x)

model = keras.Model(input, x)

model.compile(loss = keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

optimizer = keras.optimizers.Adam(learning\_rate = 0.0001),

metrics = ['accuracy'])

import os

from tensorflow.keras.callbacks import ModelCheckpoint

# Define the directory where you want to save the weights

checkpoint\_dir = r"C:\Final Year Project\weights"

# Define the file path where you want to save the weights

checkpoint\_path = os.path.join(checkpoint\_dir, "cp.ckpt")

# Create the directory if it doesn't exist

os.makedirs(checkpoint\_dir, exist\_ok=True)

# Create a callback that saves the model's weights

cp\_callback = ModelCheckpoint(filepath=checkpoint\_path,

save\_weights\_only=True,

verbose=1)

**Training**

# Train the model without any callback

history = model.fit(

x=train\_ds,

epochs=50,

validation\_data=val\_ds,

callbacks=[cp\_callback]) # Pass callback to training

**Training Result**

Epoch 1/50

171/Unknown - 925s 5s/step - loss: 0.7195 - accuracy: 0.5641

Epoch 1: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 991s 6s/step - loss: 0.7195 - accuracy: 0.5641 - val\_loss: 0.7025 - val\_accuracy: 0.5171

Epoch 2/50

171/171 [==============================] - ETA: 0s - loss: 0.5893 - accuracy: 0.6916

Epoch 2: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.5893 - accuracy: 0.6916 - val\_loss: 0.5010 - val\_accuracy: 0.7740

Epoch 3/50

171/171 [==============================] - ETA: 0s - loss: 0.5087 - accuracy: 0.7473

Epoch 3: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 982s 6s/step - loss: 0.5087 - accuracy: 0.7473 - val\_loss: 0.4429 - val\_accuracy: 0.7877

Epoch 4/50

171/171 [==============================] - ETA: 0s - loss: 0.4553 - accuracy: 0.7751

Epoch 4: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.4553 - accuracy: 0.7751 - val\_loss: 0.3769 - val\_accuracy: 0.8219

Epoch 5/50

171/171 [==============================] - ETA: 0s - loss: 0.4260 - accuracy: 0.8088

Epoch 5: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 982s 6s/step - loss: 0.4260 - accuracy: 0.8088 - val\_loss: 0.3814 - val\_accuracy: 0.8185

Epoch 6/50

171/171 [==============================] - ETA: 0s - loss: 0.4024 - accuracy: 0.8132

Epoch 6: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 981s 6s/step - loss: 0.4024 - accuracy: 0.8132 - val\_loss: 0.3717 - val\_accuracy: 0.7945

Epoch 7/50

171/171 [==============================] - ETA: 0s - loss: 0.3991 - accuracy: 0.8278

Epoch 7: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.3991 - accuracy: 0.8278 - val\_loss: 0.4457 - val\_accuracy: 0.7877

Epoch 8/50

171/171 [==============================] - ETA: 0s - loss: 0.3882 - accuracy: 0.8205

Epoch 8: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 981s 6s/step - loss: 0.3882 - accuracy: 0.8205 - val\_loss: 0.3275 - val\_accuracy: 0.8356

Epoch 9/50

171/171 [==============================] - ETA: 0s - loss: 0.3612 - accuracy: 0.8388

Epoch 9: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.3612 - accuracy: 0.8388 - val\_loss: 0.3405 - val\_accuracy: 0.8322

Epoch 10/50

171/171 [==============================] - ETA: 0s - loss: 0.3493 - accuracy: 0.8410

Epoch 10: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.3493 - accuracy: 0.8410 - val\_loss: 0.3351 - val\_accuracy: 0.8288

Epoch 11/50

171/171 [==============================] - ETA: 0s - loss: 0.3347 - accuracy: 0.8579

Epoch 11: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.3347 - accuracy: 0.8579 - val\_loss: 0.3240 - val\_accuracy: 0.8459

Epoch 12/50

171/171 [==============================] - ETA: 0s - loss: 0.3470 - accuracy: 0.8520

Epoch 12: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 981s 6s/step - loss: 0.3470 - accuracy: 0.8520 - val\_loss: 0.3177 - val\_accuracy: 0.8596

Epoch 13/50

171/171 [==============================] - ETA: 0s - loss: 0.3298 - accuracy: 0.8469

Epoch 13: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 981s 6s/step - loss: 0.3298 - accuracy: 0.8469 - val\_loss: 0.3202 - val\_accuracy: 0.8596

Epoch 14/50

171/171 [==============================] - ETA: 0s - loss: 0.3374 - accuracy: 0.8564

Epoch 14: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.3374 - accuracy: 0.8564 - val\_loss: 0.3304 - val\_accuracy: 0.8493

Epoch 15/50

171/171 [==============================] - ETA: 0s - loss: 0.3149 - accuracy: 0.8659

Epoch 15: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.3149 - accuracy: 0.8659 - val\_loss: 0.3350 - val\_accuracy: 0.8425

Epoch 16/50

171/171 [==============================] - ETA: 0s - loss: 0.2863 - accuracy: 0.8813

Epoch 16: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 981s 6s/step - loss: 0.2863 - accuracy: 0.8813 - val\_loss: 0.2686 - val\_accuracy: 0.8767

Epoch 17/50

171/171 [==============================] - ETA: 0s - loss: 0.2807 - accuracy: 0.8842

Epoch 17: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.2807 - accuracy: 0.8842 - val\_loss: 0.3650 - val\_accuracy: 0.8493

Epoch 18/50

171/171 [==============================] - ETA: 0s - loss: 0.2876 - accuracy: 0.8864

Epoch 18: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.2876 - accuracy: 0.8864 - val\_loss: 0.2433 - val\_accuracy: 0.8870

Epoch 19/50

171/171 [==============================] - ETA: 0s - loss: 0.2790 - accuracy: 0.8908

Epoch 19: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.2790 - accuracy: 0.8908 - val\_loss: 0.2660 - val\_accuracy: 0.8801

Epoch 20/50

171/171 [==============================] - ETA: 0s - loss: 0.2600 - accuracy: 0.8908

Epoch 20: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 981s 6s/step - loss: 0.2600 - accuracy: 0.8908 - val\_loss: 0.2411 - val\_accuracy: 0.8973

Epoch 21/50

171/171 [==============================] - ETA: 0s - loss: 0.2669 - accuracy: 0.8938

Epoch 21: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 981s 6s/step - loss: 0.2669 - accuracy: 0.8938 - val\_loss: 0.3522 - val\_accuracy: 0.8390

Epoch 22/50

171/171 [==============================] - ETA: 0s - loss: 0.2831 - accuracy: 0.8755

Epoch 22: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.2831 - accuracy: 0.8755 - val\_loss: 0.2439 - val\_accuracy: 0.9041

Epoch 23/50

171/171 [==============================] - ETA: 0s - loss: 0.2697 - accuracy: 0.8967

Epoch 23: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.2697 - accuracy: 0.8967 - val\_loss: 0.2573 - val\_accuracy: 0.8801

Epoch 24/50

171/171 [==============================] - ETA: 0s - loss: 0.2331 - accuracy: 0.9062

Epoch 24: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.2331 - accuracy: 0.9062 - val\_loss: 0.2487 - val\_accuracy: 0.8801

Epoch 25/50

171/171 [==============================] - ETA: 0s - loss: 0.2275 - accuracy: 0.9114

Epoch 25: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.2275 - accuracy: 0.9114 - val\_loss: 0.2590 - val\_accuracy: 0.8870

Epoch 26/50

171/171 [==============================] - ETA: 0s - loss: 0.2539 - accuracy: 0.8923

Epoch 26: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 983s 6s/step - loss: 0.2539 - accuracy: 0.8923 - val\_loss: 0.3305 - val\_accuracy: 0.8288

Epoch 27/50

171/171 [==============================] - ETA: 0s - loss: 0.2157 - accuracy: 0.9106

Epoch 27: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 981s 6s/step - loss: 0.2157 - accuracy: 0.9106 - val\_loss: 0.2189 - val\_accuracy: 0.9075

Epoch 28/50

171/171 [==============================] - ETA: 0s - loss: 0.2091 - accuracy: 0.9084

Epoch 28: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 981s 6s/step - loss: 0.2091 - accuracy: 0.9084 - val\_loss: 0.2628 - val\_accuracy: 0.8767

Epoch 29/50

171/171 [==============================] - ETA: 0s - loss: 0.2035 - accuracy: 0.9260

Epoch 29: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.2035 - accuracy: 0.9260 - val\_loss: 0.2273 - val\_accuracy: 0.8973

Epoch 30/50

171/171 [==============================] - ETA: 0s - loss: 0.2085 - accuracy: 0.9275

Epoch 30: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 981s 6s/step - loss: 0.2085 - accuracy: 0.9275 - val\_loss: 0.2616 - val\_accuracy: 0.8836

Epoch 31/50

171/171 [==============================] - ETA: 0s - loss: 0.2088 - accuracy: 0.9121

Epoch 31: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 981s 6s/step - loss: 0.2088 - accuracy: 0.9121 - val\_loss: 0.2030 - val\_accuracy: 0.9007

Epoch 32/50

171/171 [==============================] - ETA: 0s - loss: 0.1965 - accuracy: 0.9260

Epoch 32: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.1965 - accuracy: 0.9260 - val\_loss: 0.3356 - val\_accuracy: 0.8630

Epoch 33/50

171/171 [==============================] - ETA: 0s - loss: 0.2066 - accuracy: 0.9187

Epoch 33: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.2066 - accuracy: 0.9187 - val\_loss: 0.2271 - val\_accuracy: 0.9075

Epoch 34/50

171/171 [==============================] - ETA: 0s - loss: 0.1769 - accuracy: 0.9282

Epoch 34: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.1769 - accuracy: 0.9282 - val\_loss: 0.2421 - val\_accuracy: 0.8836

Epoch 35/50

171/171 [==============================] - ETA: 0s - loss: 0.1927 - accuracy: 0.9289

Epoch 35: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.1927 - accuracy: 0.9289 - val\_loss: 0.2550 - val\_accuracy: 0.8973

Epoch 36/50

171/171 [==============================] - ETA: 0s - loss: 0.1901 - accuracy: 0.9179

Epoch 36: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 979s 6s/step - loss: 0.1901 - accuracy: 0.9179 - val\_loss: 0.2068 - val\_accuracy: 0.9075

Epoch 37/50

171/171 [==============================] - ETA: 0s - loss: 0.1780 - accuracy: 0.9348

Epoch 37: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.1780 - accuracy: 0.9348 - val\_loss: 0.3137 - val\_accuracy: 0.8767

Epoch 38/50

171/171 [==============================] - ETA: 0s - loss: 0.1724 - accuracy: 0.9319

Epoch 38: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 982s 6s/step - loss: 0.1724 - accuracy: 0.9319 - val\_loss: 0.2461 - val\_accuracy: 0.8870

Epoch 39/50

171/171 [==============================] - ETA: 0s - loss: 0.1654 - accuracy: 0.9370

Epoch 39: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.1654 - accuracy: 0.9370 - val\_loss: 0.4270 - val\_accuracy: 0.7945

Epoch 40/50

171/171 [==============================] - ETA: 0s - loss: 0.1541 - accuracy: 0.9407

Epoch 40: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 979s 6s/step - loss: 0.1541 - accuracy: 0.9407 - val\_loss: 0.2006 - val\_accuracy: 0.9212

Epoch 41/50

171/171 [==============================] - ETA: 0s - loss: 0.1467 - accuracy: 0.9436

Epoch 41: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.1467 - accuracy: 0.9436 - val\_loss: 0.1818 - val\_accuracy: 0.9212

Epoch 42/50

171/171 [==============================] - ETA: 0s - loss: 0.1498 - accuracy: 0.9458

Epoch 42: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.1498 - accuracy: 0.9458 - val\_loss: 0.2372 - val\_accuracy: 0.8973

Epoch 43/50

171/171 [==============================] - ETA: 0s - loss: 0.1720 - accuracy: 0.9304

Epoch 43: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 979s 6s/step - loss: 0.1720 - accuracy: 0.9304 - val\_loss: 0.1799 - val\_accuracy: 0.9349

Epoch 44/50

171/171 [==============================] - ETA: 0s - loss: 0.1322 - accuracy: 0.9509

Epoch 44: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 979s 6s/step - loss: 0.1322 - accuracy: 0.9509 - val\_loss: 0.2308 - val\_accuracy: 0.9007

Epoch 45/50

171/171 [==============================] - ETA: 0s - loss: 0.1129 - accuracy: 0.9560

Epoch 45: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 979s 6s/step - loss: 0.1129 - accuracy: 0.9560 - val\_loss: 0.2321 - val\_accuracy: 0.8938

Epoch 46/50

171/171 [==============================] - ETA: 0s - loss: 0.1347 - accuracy: 0.9473

Epoch 46: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 981s 6s/step - loss: 0.1347 - accuracy: 0.9473 - val\_loss: 0.2095 - val\_accuracy: 0.9041

Epoch 47/50

171/171 [==============================] - ETA: 0s - loss: 0.1359 - accuracy: 0.9436

Epoch 47: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 979s 6s/step - loss: 0.1359 - accuracy: 0.9436 - val\_loss: 0.2058 - val\_accuracy: 0.9281

Epoch 48/50

171/171 [==============================] - ETA: 0s - loss: 0.1252 - accuracy: 0.9531

Epoch 48: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.1252 - accuracy: 0.9531 - val\_loss: 0.2817 - val\_accuracy: 0.8801

Epoch 49/50

171/171 [==============================] - ETA: 0s - loss: 0.1253 - accuracy: 0.9575

Epoch 49: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 979s 6s/step - loss: 0.1253 - accuracy: 0.9575 - val\_loss: 0.2253 - val\_accuracy: 0.9007

Epoch 50/50

171/171 [==============================] - ETA: 0s - loss: 0.1191 - accuracy: 0.9597

Epoch 50: saving model to C:\Final Year Project\weights\cp.ckpt

171/171 [==============================] - 980s 6s/step - loss: 0.1191 - accuracy: 0.9597 - val\_loss: 0.2420 - val\_accuracy: 0.9007

**Plotting Accuracy**

def plot\_history(history):

"""

Plotting training and validation learning curves.

Args:

history: model history with all the metric measures

"""

fig, (ax1, ax2) = plt.subplots(2)

fig.set\_size\_inches(18.5, 10.5)

# Plot loss

ax1.set\_title('Loss')

ax1.plot(history.history['loss'], label = 'train')

ax1.plot(history.history['val\_loss'], label = 'test')

ax1.set\_ylabel('Loss')

# Determine upper bound of y-axis

max\_loss = max(history.history['loss'] + history.history['val\_loss'])

ax1.set\_ylim([0, np.ceil(max\_loss)])

ax1.set\_xlabel('Epoch')

ax1.legend(['Train', 'Validation'])

# Plot accuracy

ax2.set\_title('Accuracy')

ax2.plot(history.history['accuracy'], label = 'train')

ax2.plot(history.history['val\_accuracy'], label = 'test')

ax2.set\_ylabel('Accuracy')

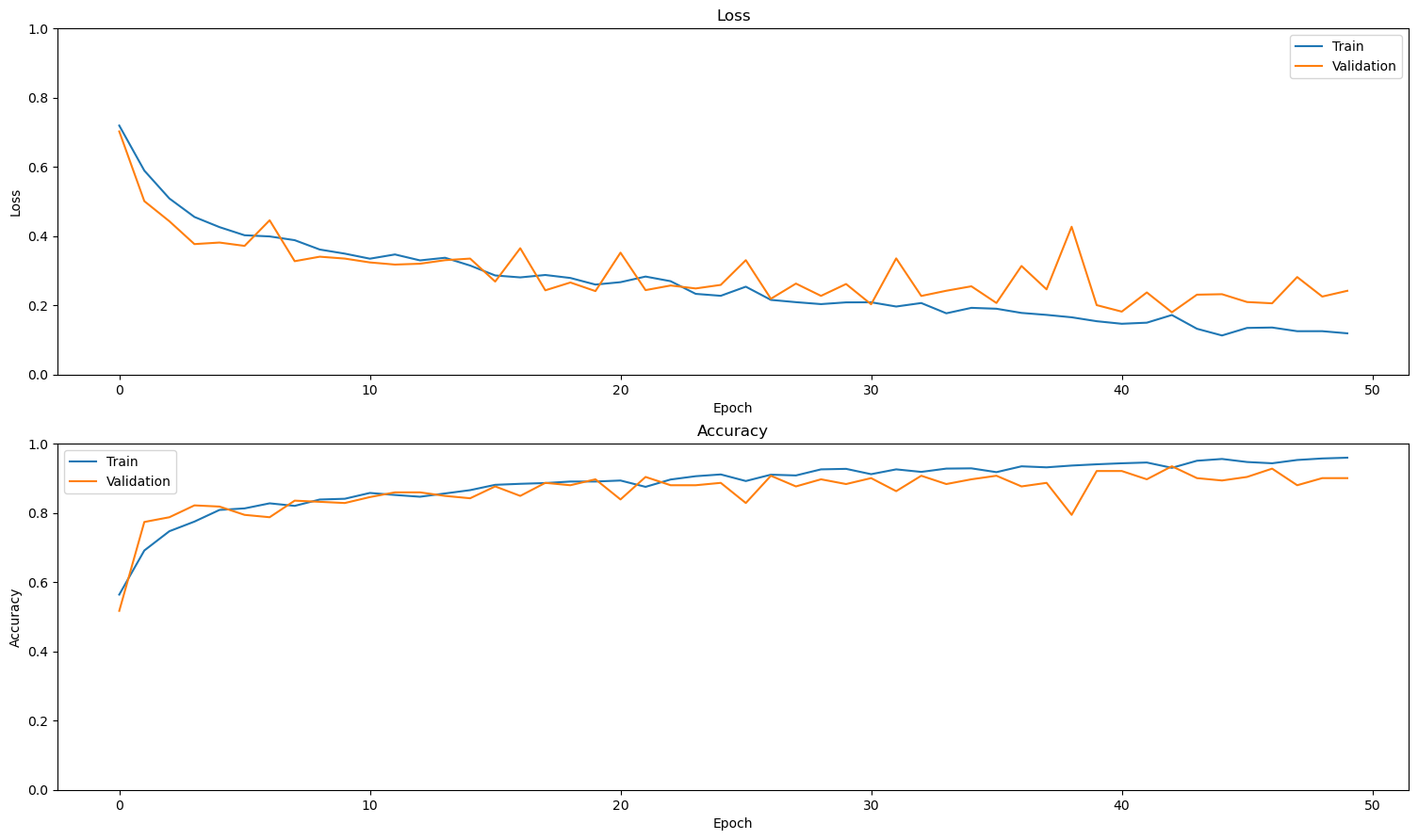
ax2.set\_ylim([0, 1])

ax2.set\_xlabel('Epoch')

ax2.legend(['Train', 'Validation'])

plt.show()

plot\_history(history)



def calculate\_classification\_metrics(y\_actual, y\_pred, labels):

"""

Calculate the precision and recall of a classification model using the ground truth and

predicted values.

Args:

y\_actual: Ground truth labels.

y\_pred: Predicted labels.

labels: List of classification labels.

Return:

Precision and recall measures.

"""

cm = tf.math.confusion\_matrix(y\_actual, y\_pred)

tp = np.diag(cm) # Diagonal represents true positives

precision = dict()

recall = dict()

for i in range(len(labels)):

col = cm[:, i]

fp = np.sum(col) - tp[i] # Sum of column minus true positive is false negative

row = cm[i, :]

fn = np.sum(row) - tp[i] # Sum of row minus true positive, is false negative

precision[labels[i]] = tp[i] / (tp[i] + fp) # Precision

recall[labels[i]] = tp[i] / (tp[i] + fn) # Recall

return precision, recall

precision, recall = calculate\_classification\_metrics(actual, predicted, labels) # Test dataset

print(precision)

{'fights': 0.8974358974358975, 'nofights': 0.927536231884058}

print(recall)

{'fights': 0.9333333333333333, 'nofights': 0.8888888888888888}