**Backend Model\_Loader**

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

import einops

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

HEIGHT = 224

WIDTH = 224

class Conv2Plus1D(keras.layers.Layer):

"""

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

spatial dimensions, and then the temporal dimension.

"""

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

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):

"""

Resizes the video tensor using the einops library.

"""

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):

"""

Resizes the tensor representation of the video, in the form of a set of frames.

Args:

video: Tensor representation of the video.

Returns:

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

"""

# Parse shape of the video tensor

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

# Rearrange the tensor to make it compatible with resizing

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

# Resize the images

images = self.resizing\_layer(images)

# Rearrange the images back to the original shape

videos = einops.rearrange(

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

t=old\_shape['t'])

return videos

# Function to load the pre-trained model

def load\_custom\_model():

"""

Loads the pre-trained custom model with custom layers.

Returns:

tf.keras.Model: Loaded pre-trained model.

"""

custom\_objects = {

'Conv2Plus1D': Conv2Plus1D,

'ResizeVideo': ResizeVideo,

'ResidualMain': ResidualMain,

'Project': Project

}

# Load the model with custom objects

return tf.keras.models.load\_model(r"E:\Final Year Project\r\_2\_1\_d\_3d\_cnn\_model\_final.h5", custom\_objects=custom\_objects)