Kill-Death Classifier in a First-person Shooting Game

Presented To:

MD. JAFAR IQBAL

Managing Director & CEO,

Opus Technology Limited,

Dhaka, Bangladesh.

Presented By:

M. Mashrukh Zayed

MSc. Student is CSE, IUT-OIC

Home Address: Pabna Sadar, Pabna

Email: zayedmashrukh@ieee.org

Introduction

- This classification task is specifically dependent on a video dataset which includes video-clips of kill and death scores from the first-person shooting game “CS Go”.

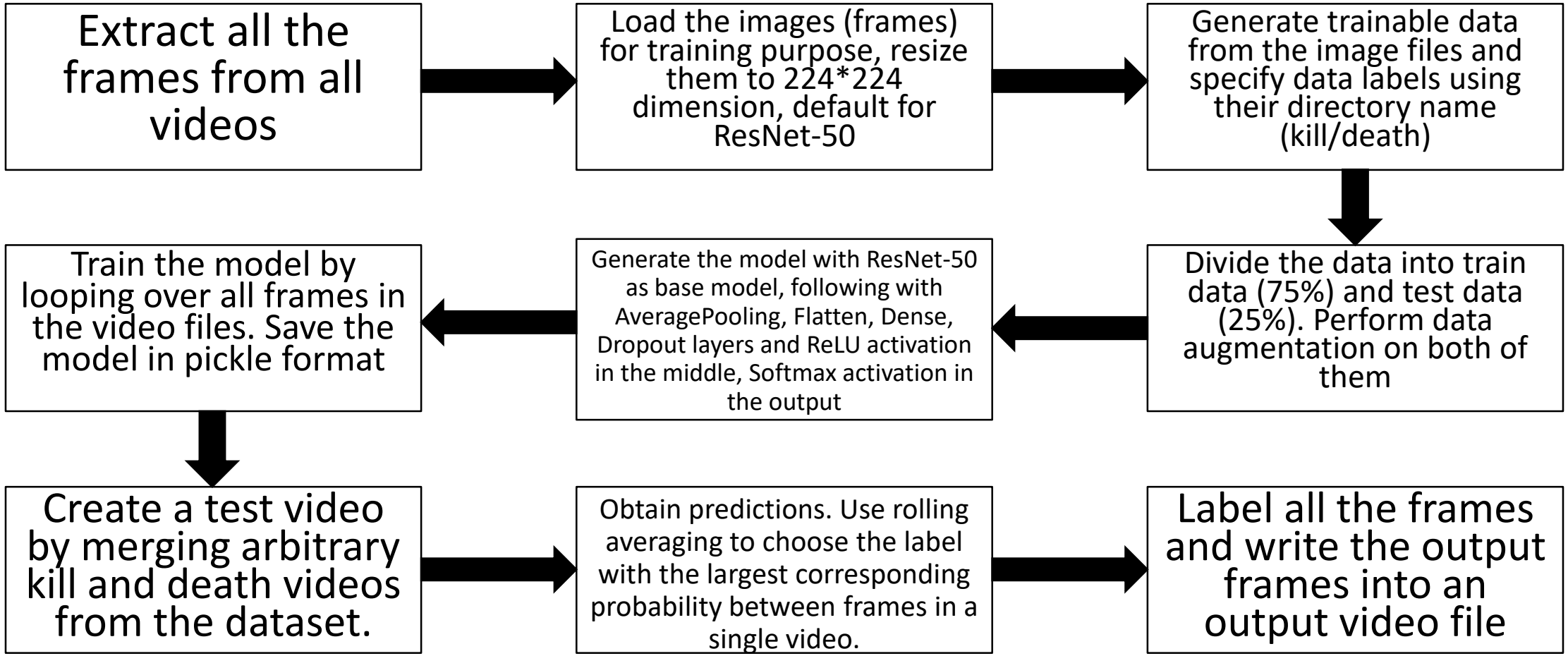
- Videos can be understood as a series of individual images. We can treat video classification as performing image classification a total of N times, where N is the total number of frames in a video.

- In this task, I have used ResNet-50 model for image classification and then turn it into a more accurate video classifier by employing rolling averaging method.

- ResNet-50 is a convolutional neural network that is 50 layers deep.

- I used rolling averaging method to avoid “**prediction flickering**”, which is a common problem for video classification. It is a prediction shifting problem between two predictions in a same video (one frame shows kill, other frame shows death, at the start of the video). It can easily be soled using rolling averaging method.

System Architecture



(1) Extract all the frames from all videos

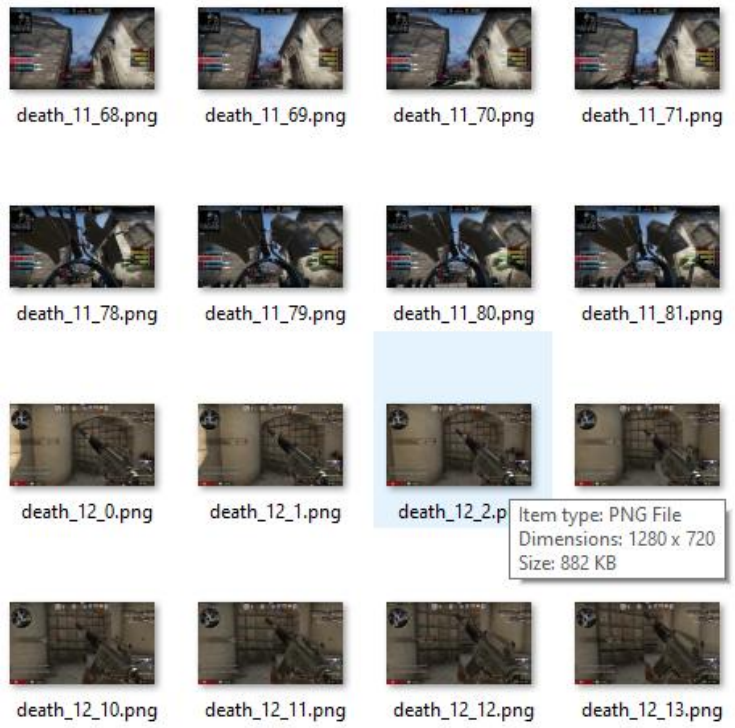
- Extracted all the frames from all the videos using a single code and saved them as frames.
- Used the below portion to name the frames according to their video name and frame number.
- Also kept option to extract the frames using any intervals **(gap)** I want.

```
while True:
    ret, frame = cap.read()

    if ret == False:
        cap.release()
        break

    if idx == 0:
        filename = save_path + "_" + str(idx) + ".png"
        cv2.imwrite(filename, frame)
    else:
        if idx % gap == 0:
            filename = save_path + "_" + str(idx) + ".png"
            cv2.imwrite(filename, frame)

    idx += 1
```



(2) Load Images, Resize (3) Define Label

- Loaded frames from the “kill” and “directory”.
- Defined the labels using the directory name.
- Resized the images into 224×224 dimension.
- Transformed the labels into binarized form (0,1).

```
KD_labels = set(['kill', 'death'])

print('Images are being loaded...')

path_to_images = list(paths.list_images(data_path))
data = []
labels = []

for images in path_to_images:
    label = images.split(os.path.sep)[-2]
    if label not in KD_labels:
        continue
    image = cv2.imread(images)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (224,224))

    data.append(image)
    labels.append(label)
```

```
data = np.array(data)
labels = np.array(labels)

lb = LabelBinarizer()
labels = lb.fit_transform(labels)
```

(4) Train-Test Split, Augmentation

- 25% data is used for testing, the rest is for training.
- Performed data augmentation on both portion of the data.
- Defined the mean subtraction (in RGB order) and set the mean subtraction value for each of the data augmentation.

```
(X_train, X_test, Y_train, Y_test) = train_test_split(data, labels, test_size=0.25, stratify=labels, random_state=42)
```

```
train_Aug = ImageDataGenerator(
    rotation_range=30,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest"
)

valid_Aug = ImageDataGenerator()
mean = np.array([123.68, 116.779, 103.939], dtype="float32")

train_Aug.mean = mean
valid_Aug.mean = mean
```

(5) Generate the model

- ResNet50 architecture has 50 layers of CNN. I have added several other layers such as: AveragePooling2D with 77 pool-size, Flatten, Dense Layer with 512 neurons and ReLU activation, 50% Dropout
- Used Softmax activation in the output layer, with number of classes defining the number of output neurons
- Used categorical_crossentropy as loss function

```
# Model

base_model = ResNet50(weights="imagenet", include_top=False, input_tensor=Input(shape=(224,224,3)))

head_model = base_model.output
head_model = AveragePooling2D(pool_size=(7,7))(head_model)
head_model = Flatten(name="flatten")(head_model)
head_model = Dense(512, activation='relu')(head_model)
head_model = Dropout(0.5)(head_model)
head_model = Dense(len(lb.classes_), activation='softmax')(head_model)

model = Model(inputs=base_model.input, outputs=head_model)
epochs = 5
```

```
from keras.optimizers import SGD

opt = SGD(lr=0.0001, momentum=0.9, decay=1e-4/epochs)
```

```
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
```

(6) Train the model & save

- Trained the model with a batch size of 32, for steps dependent on the batch size also.
- Used only 5 epochs considering the 2 days deadline for the task. A considerable number of more epochs surely can increase the performance.
- Saved the trained model in pickle format.

```
history = model.fit_generator(
    train_Aug.flow(X_train, Y_train, batch_size=32),
    steps_per_epoch=len(X_train)//32,
    validation_data=valid_Aug.flow(X_test, Y_test),
    validation_steps=len(X_test)//32,
    epochs=epochs
)
```

```
# Save Model

import pickle

model.save(output_model)
lbinarizer = open(r"KD_classification_model\KD_classification_binarizer.pickle", "wb")
lbinarizer.write(pickle.dumps(lb))
lbinarizer.close()
```

(7) Creating Test Video &

- Chose 5 death and 5 kill videos for creating the test video.
- The right-bottom figure shows the process of creating the merged video using "**MoviePy**" library.

```
from moviepy.editor import *
from moviepy.editor import VideoFileClip
from moviepy.video.io.VideoFileClip import VideoFileClip


clip1 = VideoFileClip("death_57.mp4",audio=True)
clip2 = VideoFileClip("death_58.mp4",audio=True)
clip3 = VideoFileClip("death_60.mp4",audio=True)
clip4 = VideoFileClip("death_62.mp4",audio=True)
clip5 = VideoFileClip("death_63.mp4",audio=True)

clip6 = VideoFileClip("kill_13.mp4",audio=True)
clip7 = VideoFileClip("kill_28.mp4",audio=True)
clip8 = VideoFileClip("kill_51.mp4",audio=True)
clip9 = VideoFileClip("kill_112.mp4",audio=True)
clip10 = VideoFileClip("kill_143.mp4",audio=True)


final_clip = concatenate_videoclips([clip1,clip2, clip3, clip4, clip5,
                                     clip6, clip7, clip8, clip9, clip10], method="compose")
final_clip.write_videofile("merged.mp4")
```

```
In [4]: runfile('D:/Opus Project/video_data/VideoMerge.py',
wdir='D:/Opus Project/video_data')
chunk:   0%|          | 0/654 [00:00<?, ?it/s, now=None]Moviepy -
Building video merged.mp4.
MoviePy - Writing audio in mergedTEMP_MPY_wvf_snd.mp3
t:   0%|          | 0/890 [00:00<?, ?it/s, now=None]
MoviePy - Done.
Moviepy - Writing video merged.mp4

t:  49%|████▉     | 434/890 [00:22<00:28, 16.10it/s, now=None]
```

(8) Obtaining Prediction, Applying rolling averaging

- Captured the video using cv2, read the frames and obtained label predictions on them individually.
- Used rolling averaging to choose the label with the largest corresponding probability between frames in a single video.
- The variable queue (bottom-right figure) keeps track of the prediction results and helps with choosing the label with highest probability.

```
capture_video = cv2.VideoCapture(r"KD_classification_model\5D_5K.mp4")
writer = None
(width, height) = (None, None)

while True:
    (taken, frame) = capture_video.read()
    if not taken:
        break
    if width is None or height is None:
        (width, height) = frame.shape[:2]

    output = frame.copy()
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame = cv2.resize(frame, (224,224)).astype("float32")
    frame -= mean

    pred = model.predict(np.expand_dims(frame, axis=0))[0]
```

```
queue.append(pred)
result = np.array(queue).mean(axis=0)
i = np.argmax(result)
label = lb.classes_[i]
```

(9) Label the frames & generate an output video

- Labeled each frame with the predicted action at top-left (blue text).
- Displayed the progress frame by frame while processing.
- Used all the output frames to generate an output video.

```
text = "Score : {}".format(label)
cv2.putText(output, text, (45,60), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,0,0), 4)

if writer is None:
    fourcc = cv2.VideoWriter_fourcc(*"MJPG")
    writer = cv2.VideoWriter(video_output, fourcc, 30, (width, height), True)

writer.write(output)
cv2.imshow("In progress frame by frame", output)
```

Thank you