EDGE INTELLIGENCE LAB 7
MACSE604

SUBMITTED BY
SHIBU P
25MML0042

VIDEO PROCESSING USING EFFICIENTNETB0:

Task 1:

```
[ ]
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input, decode_predictions
from tensorflow.keras.applications.efficientnet import preprocess_input, decode_predictions

# Load pretrained MobileNetV2
model = EfficientNetB0(weights='imagenet')
frames = []

import cv2
import numpy as np

def extract_frames(video_path, max_frames=20):
    cap = cv2.VideoCapture(video_path)

    if not cap.isOpened():
        print("Error: Cannot open video")
```

```
        print("Error: Cannot open video")
        return np.array([])


    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    # Handle videos where frame count is 0
    if total_frames > 0:
        step = max(1, total_frames // max_frames)
    else:
        step = 1

    count = 0

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        if count % step == 0:
            frame = cv2.resize(frame, (224, 224))
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB) # 🔴 Important fix
```

```
            frames.append(frame)

        count += 1

        if len(frames) >= max_frames:
            break

    cap.release()
    print("Extracted frames:", len(frames))
    return np.array(frames)



def classify_video(video_path):
    frames = extract_frames(video_path)

    if len(frames) == 0:
        print("No frames extracted!")
        return

    frames = preprocess_input(frames)
    predictions = model.predict(frames)
```

```
    # Average predictions across frames
    avg_pred = np.mean(predictions, axis=0)
    #top_pred = decode_predictions(np.expand_dims(avg_pred, axis=0), top=1)

    top_pred = decode_predictions(np.expand_dims(avg_pred, axis=0), top=5)

    for pred in top_pred[0]:
      print(pred[1], ":", round(pred[2]*100, 2), "%")


    print("Final Prediction:", top_pred[0][0][1])

# Replace with your video path
video_path = "/content/026c7465-309f6d33.mp4"
classify_video(video_path)
len(frames)
```

Output:

```
Extracted frames: 20
1/1 ━━━━━━━━━━━━━━━━━━━━ 5s 5s/step
car_mirror : 7.07 %
cab : 6.78 %
jinrikisha : 6.62 %
limousine : 6.55 %
traffic_light : 4.73 %
Final Prediction: car_mirror
20
```

```
frames[0][0]

array([[ 60,  67,  53],
       [ 57,  63,  50],
       [125, 131, 119],
       [103, 108, 101],
       [ 99, 103,  98],
       [157, 159, 153],
       [101,  99,  94],
       [ 88,  69,  71],
       [ 86,  60,  48],
       [136, 105,  91],
       [139, 102,  89],
       [136, 106,  96],
       [ 82,  61,  48],
       [ 72,  65,  46],
       [ 62,  63,  53],
       [ 62,  62,  57],
       [ 74,  72,  73],
       [ 48,  37,  36],
       [101,  70,  69],
       [111,  82,  76],
       [ 82,  60,  51]
```

```
frames[0]

ndarray (224, 224, 3) show data
```

Insights Learned:

- Shape of extracted frames are 20
- Shape of every frame is 224x224x3 where 224 are width ,224 height and 3 channel
- Here we are storing the frames in the list
- Cap is an object of the cv2 class
- Cap.read will return two value ret,frame,where ret contain the boolean value indicating the total cover of the frames after frames are over it return false,frame contain all collection of pixel values
- cap.read() is a function
- cv2.COLORRGB2BGR is a attribute

Why efficienetB0 is detecting car mirror,cab,jinriksha,limousine,traffic light?

The EfficientB0 is trained on imagenet dataset of 1000 classes,so if an object does not belong to the belong to the class it will choose the matching class,Averaging of every classes are taken for the final probability

```python
import numpy as np
import matplotlib.pyplot as plt
video_path="/content/026c7465-309f6d33.mp4"
# Extract frames
extracted_frames = extract_frames(video_path)

# Ensure numpy array
extracted_frames = np.array(extracted_frames)

print("Shape of frames:", extracted_frames.shape)

# Number of frames to show
num_frames_to_show = min(12, len(extracted_frames))

# Grid size
cols = 4
rows = int(np.ceil(num_frames_to_show / cols))

plt.figure(figsize=(12, 6))
```

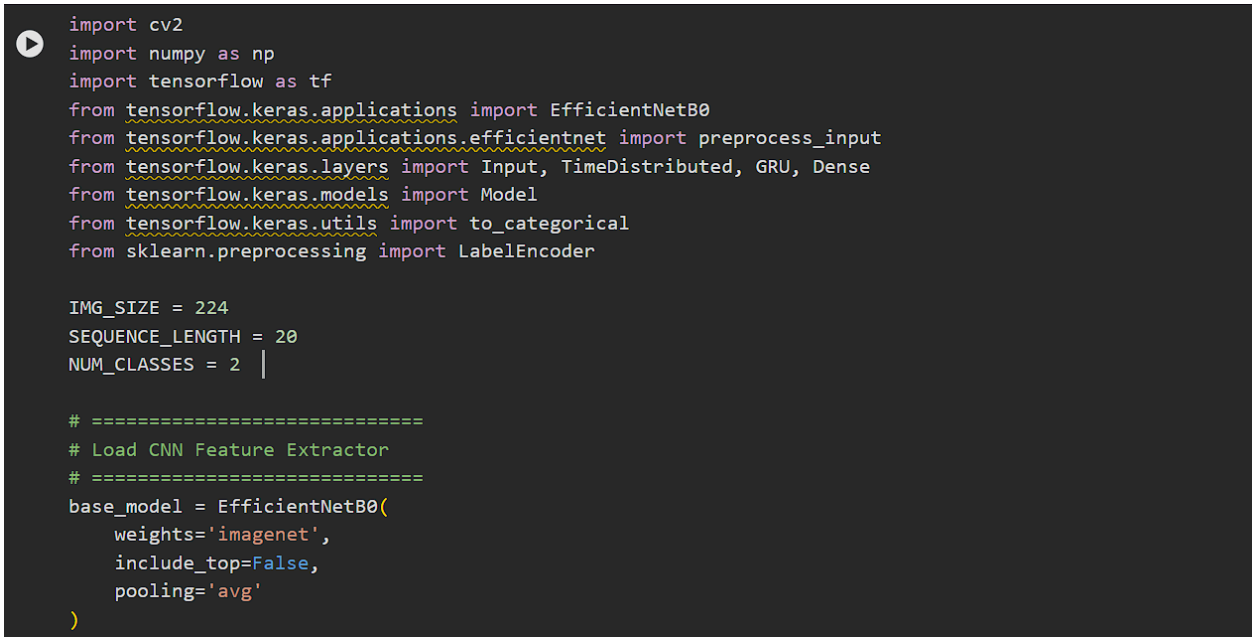
```
    plt.figure(figsize=(12, 6))

    for i in range(num_frames_to_show):
      plt.subplot(rows, cols, i + 1)
      plt.imshow(extracted_frames[i])
      plt.title(f"Frame {i}")
      plt.axis("off")

      plt.tight_layout ()
      plt.show()
```

```
Extracted frames: 21
Shape of frames: (21, 224, 224, 3)
```

Existing code uses **EfficientNetB0** as a pretrained image classification model to classify a video by extracting frames and predicting each frame independently. It then averages the predictions of all frames to produce a final class label. This approach treats the video as a collection of separate images and does not consider the order of frames or motion between them. Therefore, it mainly detects objects present in the frames (like dog, car, person) but does not understand actions or temporal changes. In contrast, a CNN + **Gated Recurrent Unit** model first extracts spatial features from each frame using the CNN and then passes the sequence of feature vectors to the GRU. The GRU processes the frames sequentially, learning temporal dependencies and motion patterns across time. This allows the model to understand actions such as walking or running because it analyzes how the content changes from one frame to another. In summary, your current code performs frame-level object classification with averaging, while the CNN + GRU model performs spatiotemporal video classification by learning both spatial and temporal information.



```python
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.applications.efficientnet import preprocess_input
from tensorflow.keras.layers import Input, TimeDistributed, GRU, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

IMG_SIZE = 224
SEQUENCE_LENGTH = 20
NUM_CLASSES = 2

# ============================
# Load CNN Feature Extractor
# ============================
base_model = EfficientNetB0(
    weights='imagenet',
    include_top=False,
    pooling='avg'
)
```

```python
base_model.trainable = False

# ============================
# Extract All Frames
# ============================
def extract_all_frames(video_path):
    cap = cv2.VideoCapture(video_path)
    frames = []

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        frame = cv2.resize(frame, (IMG_SIZE, IMG_SIZE))
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        frames.append(frame)

    cap.release()
    return np.array(frames)
```

```python
# ============================
# Create Sequences
# ============================
def create_sequences(frames, sequence_length):
    sequences = []
    for i in range(0, len(frames) - sequence_length, sequence_length):
        seq = frames[i:i+sequence_length]
        sequences.append(seq)
    return np.array(sequences)

# ============================
# MAIN
# ============================
video_path = "/content/026c7465-309f6d33.mp4"

print("Extracting frames...")
frames = extract_all_frames(video_path)

print("Creating sequences...")
sequences = create_sequences(frames, SEQUENCE_LENGTH)

print("Total sequences:", len(sequences))
```

```
# Preprocess
sequences = preprocess_input(sequences)

# Extract CNN features
print("Extracting CNN features...")
num_samples = sequences.shape[0]
features = []

for i in range(num_samples):
    feat = base_model.predict(sequences[i])
    features.append(feat)

features = np.array(features)  # (samples, 20, 1280)

# ==============================
# Artificial Labels
# ==============================
# First half -> class0
# Second half -> class1
labels = np.zeros(num_samples)
```

```
labels[num_samples//2:] = 1

labels = to_categorical(labels, NUM_CLASSES)

# ==============================
# Build GRU Model
# ==============================
input_layer = Input(shape=(SEQUENCE_LENGTH, 1280))
x = GRU(128)(input_layer)
output = Dense(NUM_CLASSES, activation='softmax')(x)

model = Model(input_layer, output)

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

model.summary()
```

```
# ============================
# Train Model
# ============================
model.fit(features, labels, epochs=5, batch_size=2)

# ============================
# Predict
# ============================
prediction = model.predict(features[0:1])

class_names = ["Car", "No car"]

predicted_class = class_names[np.argmax(prediction)]

print("Predicted Class:", predicted_class)
```

Output:

```
Extracting frames...
Creating sequences...
Total sequences: 119
Extracting CNN features...
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
```

```
Model: "functional_2"
```

Layer (type)	Output Shape	Param #
input_layer_6 (InputLayer)	(None, 20, 1280)	0
gru_2 (GRU)	(None, 128)	541,440
dense (Dense)	(None, 2)	258

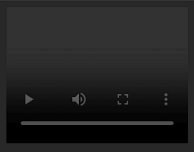
```
 Total params: 541,698 (2.07 MB)
 Trainable params: 541,698 (2.07 MB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/5
60/60 ━━━━━━━━━━━━━━━━━━━━ 4s 30ms/step - accuracy: 0.8410 - loss: 0.6536
Epoch 2/5
60/60 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 1.0000 - loss: 0.0163
Epoch 3/5
60/60 ━━━━━━━━━━━━━━━━━━━━ 3s 48ms/step - accuracy: 1.0000 - loss: 0.0056
Epoch 4/5
60/60 ━━━━━━━━━━━━━━━━━━━━ 3s 47ms/step - accuracy: 1.0000 - loss: 0.0026
Epoch 5/5
60/60 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 1.0000 - loss: 0.0010
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 215ms/step
Predicted Class: Car
```

Video and frame analysis:
Task 2:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from glob import glob
import IPython.display as ipd
from tqdm.notebook import tqdm
```

```
#import subprocess
```

```
ipd.Video("026c7465-309f6d33.mp4",width = 200)
```



```
Start coding or generate with AI.
```

Open the Video and Read Metadata

```
!pip install opencv-python
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.12/dist-packages (4.13.0.92)
Requirement already satisfied: numpy>=2 in /usr/local/lib/python3.12/dist-packages (from opencv-python) (2.0.2)
```

```
# Load in video capture
import cv2
cap = cv2.VideoCapture('026c7465-309f6d33.mp4')
```

```
# Total number of frames in video
cap.get(cv2.CAP_PROP_FRAME_COUNT)
```

```
2398.0
```

```
# Video height and width
height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
width = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
print(f'Height {height}, Width {width}')
```

```
Height 720.0, Width 1280.0
```

```
# Get frames per second
fps = cap.get(cv2.CAP_PROP_FPS)
print(f'FPS : {fps:0.2f}')
```

```
FPS : 59.94
```

```
cap.release()
```

Pull the image from the first frame

```
cap = cv2.VideoCapture('026c7465-309f6d33.mp4')
ret, img = cap.read()                          #will return TRUE everytime we call this function till we reach the end of the video
print(f'Returned {ret} and img of shape {img.shape}')
```

```
Returned True and img of shape (720, 1280, 3)
```

Loading the First Image

```
plt.imshow(img)  #opencv loads color in BGR format, matplot lib expects RGB format
```

```
<matplotlib.image.AxesImage at 0x7faad2e85be0>
```


```
## Helper function for plotting opencv images in notebook
def display_cv2_img(img, figsize=(10, 10)):
    img_ = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    fig, ax = plt.subplots(figsize=figsize)
    ax.imshow(img_)
    ax.axis("off")
```

```
display_cv2_img(img)
```

```
cap.release()
```

Matplotlib follows the standard **RGB (Red, Green, Blue)** channel order because RGB is the widely accepted color format in most visualization libraries and deep learning frameworks. When you use plt.imshow(), Matplotlib assumes the image array is arranged as RGB. If the image is not in RGB format, the displayed colors will appear incorrect.

On the other hand, OpenCV uses **BGR (Blue, Green, Red)** channel ordering by default. This design choice comes from historical reasons. OpenCV was originally built on older Windows bitmap (BMP) image formats and Intel image processing libraries, which stored pixel data in BGR format. To maintain compatibility and efficiency with those systems, OpenCV adopted BGR as its default color representation.

Video Resolution:

1.SD:

- **480p** → 854 × 480 pixels
- Older TVs and DVDs
- Low clarity
- Small file size
- Fast processing in ML models

2.HD:

- **720p** → 1280 × 720 pixels
- Good clarity
- Common in YouTube videos
- Moderate computational cost

3.FULL HD

- **1080p** → 1920 × 1080 pixels
- Very clear and detailed
- Common in modern smartphones
- Higher GPU memory required

4.4K

- **3840 × 2160 pixels**
- Extremely high detail
- Very large file size
- Heavy GPU requirement

Annotations:

Annotation is the process of labeling data (images, videos, or audio) by adding meaningful information such as object names, actions, or regions of interest.

In surveillance the annotations can be used where user gives images of the input the model should predict which frames contains the particular image