**INSTITUTE FOR ADVANCED COMPUTING AND SOFTWARE DEVELOPMENT (IACSD), AKURDI, PUNE**

Documentation On

# Human Activity Recognition (HAR) using Deep Learning

PG-DBDA March 2023

**Submitted By:**

**Group No: 16**

| **Roll No.** | **Name:** |
|---|---|
| **233535** | **Prateek Solanki** |
| **233550** | **Suryawanshi Abhishek Ashok** |

**Mr. Abhijit Nagargoje**
**Project Guide**

**Mr. Rohit Puranik**
**Centre Coordinator**

# ABSTRACT

Human Activity Recognition (HAR) has gained significant attention in recent years due to its potential applications in various domains, including healthcare, sports analysis, and surveillance. The ability to automatically classify and recognize human activities from images and videos has significant implications for improving safety, understanding behaviour patterns, and enhancing user experiences.

This project focuses on developing an efficient and accurate image classification model for Human Activity Recognition using deep learning techniques. The primary objective is to create a model that can accurately identify and classify a diverse range of human activities from images, enabling real-time monitoring and analysis of human actions.

# ACKNOWLEDGEMENT

I take this occasion to thank God, almighty for blessing us with his grace and taking our endeavor to a successful culmination. I extend my sincere and heartfelt thanks to our esteemed guide, **Mr. Abhijit Nagargoje** for providing me with the right guidance and advice at the crucial juncture sand for showing me theright way. I extend my sincere thanks to our respected **Centre Co-Ordinator Mr. Rohit Puranik**, for allowing us to use the facilities available. I would like to thank the other faculty members also, at this occasion. Last but not the least, I would like to thank my friends and family for the support and encouragement theyhave given me during the course of our work.

**Prateek Solanki (233535)**

**Suryawanshi Abhishek Ashok (233550)**

# Table of Contents

ABSTRACT........................................................ 1
ACKNOWLEDGEMENT ........................................................ 2
INTRODUCTION ........................................................ 4
PROJECT OBJECTIVE........................................................5
FEATURES ........................................................6
PROJECT OVERVIEW ........................................................7
PROJECT SCOPE ........................................................8
STUDY OF THE SYSTEM ........................................................8
DATA COLLECTION AND PRE-PROCESSING........................................................ 8
DATA AUGMENTATION........................................................ 13
MODEL ARCHITECTURE AND TRANSFER LEARNING........................................................14
TRANSFER LEARNING ........................................................14
MODEL COMPILATION........................................................15
TRAINING PROCESS ........................................................ 18
HYPER-PARAMETER TUNING ........................................................ 19
MODEL SAVING........................................................ 22
TESTING PROCESS........................................................ 25
CONCLUSION ........................................................26
REFERENCES ........................................................ 27

# INTRODUCTION

Human activity recognition (HAR) is a burgeoning field within computer vision and artificial intelligence that seeks to automatically identify and categorize human actions or activities from visual data such as images and videos. The ability to recognize human activities has far-reaching applications, spanning from healthcare and surveillance to sports analysis and human-computer interaction. As technology advances, the demand for accurate and real-time monitoring of human activities becomes increasingly relevant, influencing domains such as remote health monitoring, security systems, and smart environments.

This project centers around the development of a sophisticated image classification model for human activity recognition, leveraging the power of deep learning techniques to accomplish this task. The central goal is to harness the potential of machine learning to enable a system that can accurately perceive and understand a diverse range of human actions from visual data. By providing an automated means of interpreting human behaviors, this project contributes to enhancing safety, enabling data-driven decision-making, and augmenting user experiences in various contexts.

**Motivation and Significance**

The motivation behind this project stems from the transformative impact that human activity recognition can have on different industries and aspects of daily life. In healthcare, for instance, the ability to remotely monitor patients' activities and behaviors can lead to early detection of anomalies, thereby preventing potential health crises. Surveillance systems can benefit from real-time activity recognition to identify suspicious behaviors or unauthorized access. Moreover, sports analysts can extract valuable insights from recognizing athletes' movements, aiding in performance enhancement and injury prevention.

The significance of this project lies in its potential to automate the process of human activity recognition, eliminating the need for manual observation and interpretation. By employing advanced deep learning techniques, the model aims to achieve a level of accuracy that surpasses traditional rule-based methods. Furthermore, the flexibility of the model allows it to be adaptable to various scenarios and environments, enabling it to generalize effectively across different contexts.

# PROJECT OBJECTIVES

The primary objectives of this project are as follows:

**Image Classification Model**: Develop a robust and accurate image classification model capable of identifying and categorizing human activities from visual data.

**Transfer Learning**: Utilize transfer learning to leverage pre-trained deep learning models, speeding up the training process and benefiting from the knowledge gained from other tasks.

**Data Augmentation**: Implement data augmentation techniques to enhance the model's ability to handle variations in lighting, pose, and background in real-world scenarios.

**Custom Callbacks**: Design and incorporate custom Keras callbacks to provide greater control over the training process, allowing for dynamic adjustments and interactions during training.

**Evaluation and Analysis**: Evaluate the trained model's performance on a separate test dataset, including the calculation of accuracy, generation of confusion matrices, and classification reports.

**Model Deployment**: Explore potential avenues for deploying the trained model in practical applications, considering factors such as resource constraints and real-time requirements.

# FEATURES

1. **Image Classification Model**: Development of a sophisticated image classification model for recognizing and categorizing human activities from visual data.
2. **Deep Learning and Transfer Learning**: Utilization of deep learning techniques and transfer learning to leverage pre-trained models for faster convergence and improved performance.
3. **EfficientNetB3 Architecture**: Implementation of the EfficientNetB3 architecture, known for its efficiency and accuracy in image classification tasks.
4. **Data Augmentation**: Integration of data augmentation techniques to enhance model robustness, accounting for variations in lighting, pose, and backgrounds.
5. **Custom Keras Callbacks**: Creation of custom Keras callbacks for precise control over the training process, enabling dynamic adjustments and user interactions during training.
6. **Training, Validation, and Test Sets**: Division of the dataset into training, validation, and test sets to ensure model generalization and performance evaluation.
7. **Multi-Class Classification**: Handling of multi-class classification task, where each image can belong to one of several predefined classes representing different human activities.
8. **Accuracy Metrics**: Evaluation of model accuracy through metrics like confusion matrices, classification reports, and accuracy calculations on the test set.
9. **Real-World Applications**: Exploration of real-world applications, including healthcare, surveillance, and sports analysis, where automated human activity recognition can provide valuable insights.
10. **Resource Efficiency**: Achievement of a balance between computational efficiency and accuracy by selecting the EfficientNetB3 architecture.
11. **Customized Training**: Implementation of a custom callback to query users at specific epochs, allowing for fine-tuning of training duration and user input.
12. **Hyperparameter Tuning**: Potential for exploring hyperparameter tuning to optimize model performance by adjusting learning rates, batch sizes, and other parameters.
13. **Ensemble Techniques**: Future possibility of incorporating ensemble techniques to improve prediction accuracy by combining predictions from multiple models.
14. **Class Imbalance Handling**: Consideration of methods to address class imbalances in the dataset, enhancing the recognition of activities from less represented classes.
15. **Practical Deployment**: Exploration of deployment options, taking into account model size, inference speed, and resource constraints for real-world usage.

# PROJECT OVERVIEW

The Human Activity Recognition project aims to develop a sophisticated image classification model for automatically identifying and categorizing human activities from visual data. Leveraging deep learning techniques and the EfficientNetB3 architecture, the project seeks to provide accurate and efficient solutions to the challenge of recognizing various human behaviors.

The project revolves around the creation of a robust image classification model that can effectively differentiate between different human activities captured in images. By training the model on a diverse dataset of activity images, the project aims to equip the model with the capability to generalize its learned patterns to new, unseen instances.

In addition to building the model, the project involves data preprocessing, including the augmentation of training data to enhance model robustness against variations in lighting, pose, and backgrounds. The dataset is divided into training, validation, and test sets, ensuring thorough model evaluation and performance assessment.

The project embraces transfer learning by employing the EfficientNetB3 architecture, known for its efficiency and accuracy in image classification tasks. The model is fine-tuned to suit the specific task of human activity recognition, and custom Keras callbacks are integrated to facilitate user interaction during the training process.

Upon training completion, the model's performance is rigorously evaluated using accuracy metrics such as confusion matrices and classification reports. The project also explores potential real-world applications, including healthcare, surveillance, and sports analysis, where automated human activity recognition can have a significant impact.

In conclusion, the Human Activity Recognition project represents an endeavor to harness the power of deep learning to automate the identification of human activities from visual data. Through the utilization of cutting-edge techniques and state-of-the-art architectures, the project aspires to contribute to the field of computer vision and advance the capabilities of recognizing complex human behaviors.

# PROJECT SCOPE

The scope of the Human Activity Recognition project encompasses the development of an image classification model using deep learning techniques to accurately recognize and categorize a variety of human activities from visual data. The project involves data preprocessing, including augmentation, to enhance model robustness. The chosen EfficientNetB3 architecture, fine-tuned for the task, serves as the foundation for the model. The project includes training, validation, and testing phases, with custom Keras callbacks enabling user interactions during training. Performance evaluation metrics such as confusion matrices and classification reports provide insights into model accuracy. The project's potential real-world applications, such as healthcare and surveillance, highlight its practical relevance. Overall, the project aims to demonstrate the feasibility and effectiveness of using deep learning for automated human activity recognition.

# STUDY OF THE SYSTEM

# DATA COLLECTION AND PRE-PROCESSING:

The project's dataset consists of images depicting various human activities. The dataset is divided into training, validation, and test sets to ensure comprehensive model evaluation. Preprocessing involves resizing images, normalizing pixel values, and organizing them into appropriate directories.
The success of any machine learning project heavily depends on the quality and readiness of the dataset. In the case of the Human Activity Recognition project, a dataset comprising images depicting various human activities is used. The dataset plays a pivotal role in training a robust and accurate image classification model.

Data Collection
The dataset used in this project is sourced from [provide details about the source of the dataset, whether it's publicly available, a research dataset, or collected specifically for this project]. It contains a diverse range of images capturing different human activities, such as walking, running, sitting, standing, and more. The dataset is organized with labeled images, where each image is associated with a corresponding activity label.

Data Preprocessing
Data preprocessing is a crucial step in ensuring the dataset's suitability for training. The following preprocessing steps are performed on the dataset:

Image Resizing: The images in the dataset may have varying resolutions. To ensure uniformity and optimize computation, all images are resized to a consistent dimension (e.g., 200x260 pixels) using image processing libraries like OpenCV.

Normalization: The pixel values of images are normalized to a common scale, typically ranging from 0 to 1. This step helps in stabilizing the training process by preventing features with larger magnitudes from dominating the learning process.

Data Split: The dataset is divided into distinct subsets for training, validation, and testing. A common practice is to allocate around 70-80% of the data for training, 10-15% for validation, and the remaining 10-15% for testing.

Class Label Encoding: Each activity label is encoded into numerical values to facilitate model training. This is achieved by creating a mapping between activity names and corresponding integer labels.

Organizing Data: The dataset is organized into separate directories for each activity class. Each directory contains the images corresponding to that specific activity. This directory structure is compatible with various image data generators used during model training.

Data Augmentation

To enhance the model's generalization capability and improve its performance, data augmentation techniques are applied. These techniques artificially expand the dataset by creating variations of existing images. Common data augmentation techniques include:

Horizontal Flipping: Images are horizontally flipped, simulating different viewpoints of the same activity.

Rotation: Images are rotated by certain degrees, introducing variations in poses and angles.

Zooming: Images are zoomed in or out, capturing different scales of the same activity.

These techniques are crucial for training a model that can handle real-world variations and challenges, such as changes in lighting conditions, camera angles, and human poses.

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

    Mounted at /content/drive
```

```
[5] !unzip '/content/drive/MyDrive/IACSD_Project_Data/HAR.zip'

      inflating: Human Action Recognition/train/Image_9947.jpg
      inflating: Human Action Recognition/train/Image_9948.jpg
      inflating: Human Action Recognition/train/Image_9949.jpg
      inflating: Human Action Recognition/train/Image_995.jpg
      inflating: Human Action Recognition/train/Image_9950.jpg
      inflating: Human Action Recognition/train/Image_9951.jpg
      inflating: Human Action Recognition/train/Image_9952.jpg
      inflating: Human Action Recognition/train/Image_9953.jpg
      inflating: Human Action Recognition/train/Image_9954.jpg
      inflating: Human Action Recognition/train/Image_9955.jpg
      inflating: Human Action Recognition/train/Image_9956.jpg
      inflating: Human Action Recognition/train/Image_9957.jpg
      inflating: Human Action Recognition/train/Image_9958.jpg
      inflating: Human Action Recognition/train/Image_9959.jpg
      inflating: Human Action Recognition/train/Image_996.jpg
      inflating: Human Action Recognition/train/Image_9960.jpg
      inflating: Human Action Recognition/train/Image_9961.jpg
      inflating: Human Action Recognition/train/Image_9962.jpg
      inflating: Human Action Recognition/train/Image_9963.jpg
      inflating: Human Action Recognition/train/Image_9964.jpg
      inflating: Human Action Recognition/train/Image_9965.jpg
      inflating: Human Action Recognition/train/Image_9966.jpg
      inflating: Human Action Recognition/train/Image_9967.jpg
      inflating: Human Action Recognition/train/Image_9968.jpg
      inflating: Human Action Recognition/train/Image_9969.jpg
      inflating: Human Action Recognition/train/Image_997.jpg
```

```
train_csv_path=r'/content/Human Action Recognition/Training_set.csv'
test_csv_path=r'/content/Human Action Recognition/Testing_set.csv'
train_img_path=r'/content/Human Action Recognition/train'
test_img_path=r'/content/Human Action Recognition/test'
df=pd.read_csv(train_csv_path)

df.columns=['filepaths', 'labels']

df['filepaths']=df['filepaths'].apply(lambda x: os.path.join(train_img_path, x))

train_df, dummy_df=train_test_split(df, train_size=.9, shuffle=True, random_state=123, stratify=df['labels'])
valid_df, test_df= train_test_split(dummy_df, train_size=.5, shuffle=True, random_state=123, stratify=dummy_df['labels'])
print('train_df lenght: ', len(train_df), '  test_df length: ', len(test_df), '  valid_df length: ', len(valid_df))

classes=sorted(list(train_df['labels'].unique()))
class_count = len(classes)
print('The number of classes in the dataset is: ', class_count)
groups=train_df.groupby('labels')
print('{0:^30s} {1:^13s}'.format('CLASS', 'IMAGE COUNT'))
countlist=[]
classlist=[]
for label in sorted(list(train_df['labels'].unique())):
    group=groups.get_group(label)
    countlist.append(len(group))
    classlist.append(label)
    print('{0:^30s} {1:^13s}'.format(label, str(len(group))))


max_value=np.max(countlist)
max_index=countlist.index(max_value)
max_class=classlist[max_index]
```

```
max_value=np.max(countlist)
max_index=countlist.index(max_value)
max_class=classlist[max_index]
min_value=np.min(countlist)
min_index=countlist.index(min_value)
min_class=classlist[min_index]
print(max_class, ' has the most images= ',max_value, ' ', min_class, ' has the least images= ', min_value)

ht=0
wt=0

train_df_sample=train_df.sample(n=100, random_state=123,axis=0)
for i in range (len(train_df_sample)):
    fpath=train_df_sample['filepaths'].iloc[i]
    img=plt.imread(fpath)
    shape=img.shape
    ht += shape[0]
    wt += shape[1]
print('average height= ', ht//100, ' average width= ', wt//100, 'aspect ratio= ', ht/wt)
```

```
train_df lenght:  11340   test_df length:  630   valid_df length:  630
The number of classes in the dataset is:  15
            CLASS               IMAGE COUNT
           calling                  756
           clapping                 756
           cycling                  756
           dancing                  756
           drinking                 756
            eating                  756
           fighting                 756
           hugging                  756
           laughing                 756
      listening_to_music            756
           running                  756
           sitting                  756
           sleeping                 756
           texting                  756
         using_laptop               756
calling  has the most images=  756   calling  has the least images=  756
average height=  198  average width=  258 aspect ratio=  0.7662875270813989
```

```python
def trim(df, max_samples, min_samples, column):
    df=df.copy()
    groups=df.groupby(column)
    trimmed_df = pd.DataFrame(columns = df.columns)
    groups=df.groupby(column)
    for label in df[column].unique():
        group=groups.get_group(label)
        count=len(group)
        if count > max_samples:
            sampled_group=group.sample(n=max_samples, random_state=123,axis=0)
            trimmed_df=pd.concat([trimmed_df, sampled_group], axis=0)
        else:
            if count>=min_samples:
                sampled_group=group
                trimmed_df=pd.concat([trimmed_df, sampled_group], axis=0)
    print('after trimming, the maximum samples in any class is now ',max_samples, ' and the minimum samples in any class is ', min_samples)
    return trimmed_df

max_samples=300
min_samples=300
column='labels'
train_df= trim(train_df, max_samples, min_samples, column)
```

```
after trimming, the maximum samples in any class is now  300  and the minimum samples in any class is  300
```

```
working_dir=r'/content/Human Action Recognition/'
img_size=(200,260)
batch_size=30
trgen=ImageDataGenerator(horizontal_flip=True,rotation_range=20, width_shift_range=.2,
                         height_shift_range=.2, zoom_range=.2 )
t_and_v_gen=ImageDataGenerator()
msg='{0:70s} for train generator'.format(' ')
print(msg, '\r', end='')
train_gen=trgen.flow_from_dataframe(train_df, x_col='filepaths', y_col='labels', target_size=img_size,
                                   class_mode='categorical', color_mode='rgb', shuffle=True, batch_size=batch_size)
msg='{0:70s} for valid generator'.format(' ')
print(msg, '\r', end='')
valid_gen=t_and_v_gen.flow_from_dataframe(valid_df, x_col='filepaths', y_col='labels', target_size=img_size,
                                   class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=batch_size)

length=len(test_df)
test_batch_size=sorted([int(length/n) for n in range(1,length+1) if length % n ==0 and length/n<=80],reverse=True)[0]
test_steps=int(length/test_batch_size)
msg='{0:70s} for test generator'.format(' ')
print(msg, '\r', end='')
test_gen=t_and_v_gen.flow_from_dataframe(test_df, x_col='filepaths', y_col='labels', target_size=img_size,
                                   class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=test_batch_size)

classes=list(train_gen.class_indices.keys())
class_indices=list(train_gen.class_indices.values())
class_count=len(classes)
labels=test_gen.labels
print ( 'test batch size: ' ,test_batch_size, '  test steps: ', test_steps, ' number of classes : ', class_count)
```

```
Found 4500 validated image filenames belonging to 15 classes.
Found 630 validated image filenames belonging to 15 classes.
Found 630 validated image filenames belonging to 15 classes.
test batch size:  70   test steps:  9  number of classes :  15
```

```
[9]  def show_image_samples(gen ):
         t_dict=gen.class_indices
         classes=list(t_dict.keys())
         images,labels=next(gen)
         plt.figure(figsize=(20, 20))
         length=len(labels)
         if length<25:
             r=length
         else:
             r=25
         for i in range(r):
             plt.subplot(5, 5, i + 1)
             image=images[i] /255
             plt.imshow(image)
             index=np.argmax(labels[i])
             class_name=classes[index]
             plt.title(class_name, color='blue', fontsize=12)
             plt.axis('off')
         plt.show()

     show_image_samples(train_gen )
```


# MODEL ARCHITECTURE AND TRANSFER LEARNING

**EfficientNetB3 Architecture**

The heart of the Human Activity Recognition project lies in the selection of an appropriate model architecture. For this purpose, the EfficientNetB3 architecture is chosen due to its exceptional performance in image classification tasks. EfficientNet models are known for their balance between model size and accuracy, making them suitable for a wide range of applications.

EfficientNetB3 is a convolutional neural network (CNN) architecture that is pre-trained on a large-scale dataset (e.g., ImageNet). It comprises multiple layers, including convolutional layers, pooling layers, and fully connected layers. The architecture's depth and complexity allow it to capture intricate features and patterns from images.

# TRANSFER LEARNING

Transfer learning is a fundamental technique that leverages the knowledge acquired by a model on one task (pre-training) and applies it to another related task (fine-tuning). In the context of the Human Activity Recognition project, transfer learning is employed using the EfficientNetB3 architecture as the base model.

The pre-trained EfficientNetB3 model is initialized with weights learned from a large and diverse dataset, such as ImageNet. This initialization provides the model with a strong starting point, as it has already learned to recognize general features from images. By building upon this foundation, the model can be fine-tuned to specialize in recognizing specific human activities.

**Customization for the Task**

While EfficientNetB3 is a powerful architecture, it requires customization to suit the project's requirements. The model's output layer is modified to match the number of classes in the Human Activity Recognition dataset. This involves replacing the original output layer with a new densely connected layer that outputs probabilities for each activity class.

# MODEL COMPILATION

Once the model architecture is customized, it is compiled with appropriate settings for optimization. The choice of optimizer, loss function, and evaluation metrics affects the model's training process. In this project, the Adamax optimizer is chosen, categorical cross-entropy is used as the loss function, and accuracy is selected as the primary evaluation metric.

```
img_shape=(img_size[0], img_size[1], 3)
model_name='EfficientNetB3'
base_model=tf.keras.applications.efficientnet.EfficientNetB3(include_top=False, \
                                                 weights="imagenet",input_shape=img_shape, pooling='max')

base_model.trainable=True
x=base_model.output
x=BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x)
x = Dense(256, kernel_regularizer = regularizers.l2(l = 0.016),activity_regularizer=regularizers.l1(0.006),
                bias_regularizer=regularizers.l1(0.006) ,activation='relu')(x)
x=Dropout(rate=.4, seed=123)(x)
output=Dense(class_count, activation='softmax')(x)
model=Model(inputs=base_model.input, outputs=output)
lr=.001
model.compile(Adamax(learning_rate=lr), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb3_notop.h5
43941136/43941136 [==============================] - 0s 0us/step
```

```
class ASK(keras.callbacks.Callback):
    def __init__ (self, model, epochs,  ask_epoch):
        super(ASK, self).__init__()
        self.model=model
        self.ask_epoch=ask_epoch
        self.epochs=epochs
        self.ask=True

    def on_train_begin(self, logs=None):
        if self.ask_epoch == 0:
            print('you set ask_epoch = 0, ask_epoch will be set to 1', flush=True)
            self.ask_epoch=1
        if self.ask_epoch >= self.epochs:
            print('ask_epoch >= epochs, will train for ', epochs, ' epochs', flush=True)
            self.ask=False
        if self.epochs == 1:
            self.ask=False
        else:
            print('Training will proceed until epoch', ask_epoch,' then you will be asked to')
            print(' enter H to halt training or enter an integer for how many more epochs to run then be asked again')
        self.start_time= time.time()

    def on_train_end(self, logs=None):
        tr_duration=time.time() - self.start_time
        hours = tr_duration // 3600
        minutes = (tr_duration - (hours * 3600)) // 60
        seconds = tr_duration - ((hours * 3600) + (minutes * 60))
        msg = f'training elapsed time was {str(hours)} hours, {minutes:4.1f} minutes, {seconds:4.2f} seconds)'
        print (msg, flush=True)
```

```
def on_epoch_end(self, epoch, logs=None):
    if self.ask:
        if epoch + 1 ==self.ask_epoch:
            print('\n Enter H to end training or  an integer for the number of additional epochs to run then ask again')
            ans=input()

            if ans == 'H' or ans =='h' or ans == '0':
                print ('you entered ', ans, ' Training halted on epoch ', epoch+1, ' due to user input\n', flush=True)
                self.model.stop_training = True
            else:
                self.ask_epoch += int(ans)
                if self.ask_epoch > self.epochs:
                    print('\nYou specified maximum epochs of as ', self.epochs, ' cannot train for ', self.ask_epoch, flush =True)
                else:
                    print ('you entered ', ans, ' Training will continue to epoch ', self.ask_epoch, flush=True)
```

```
history=model.fit(x=train_gen,  epochs=epochs, verbose=1, callbacks=callbacks,  validation_data=valid_gen,
               validation_steps=None,  shuffle=False,  initial_epoch=0)
```

```
Training will proceed until epoch 10  then you will be asked to
 enter H to halt training or enter an integer for how many more epochs to run then be asked again
Epoch 1/40
150/150 [==============================] - 98s 570ms/step - loss: 8.9368 - accuracy: 0.3864 - val_loss: 7.3221 - val_accuracy: 0.5952
Epoch 2/40
150/150 [==============================] - 80s 533ms/step - loss: 6.3353 - accuracy: 0.6229 - val_loss: 5.3560 - val_accuracy: 0.7286
Epoch 3/40
150/150 [==============================] - 82s 546ms/step - loss: 4.7556 - accuracy: 0.7440 - val_loss: 4.2349 - val_accuracy: 0.7286
Epoch 4/40
150/150 [==============================] - 84s 557ms/step - loss: 3.6183 - accuracy: 0.8000 - val_loss: 3.3159 - val_accuracy: 0.7587
Epoch 5/40
150/150 [==============================] - 83s 554ms/step - loss: 2.7560 - accuracy: 0.8489 - val_loss: 2.6810 - val_accuracy: 0.7619
Epoch 6/40
150/150 [==============================] - 82s 541ms/step - loss: 2.1089 - accuracy: 0.8711 - val_loss: 2.1961 - val_accuracy: 0.7492
Epoch 7/40
150/150 [==============================] - 83s 552ms/step - loss: 1.6047 - accuracy: 0.9053 - val_loss: 1.8352 - val_accuracy: 0.7556
Epoch 8/40
150/150 [==============================] - 82s 547ms/step - loss: 1.2814 - accuracy: 0.9156 - val_loss: 1.5633 - val_accuracy: 0.7841
Epoch 9/40
150/150 [==============================] - 82s 544ms/step - loss: 1.0385 - accuracy: 0.9282 - val_loss: 1.4963 - val_accuracy: 0.7730
Epoch 10/40
150/150 [==============================] - 85s 565ms/step - loss: 0.8619 - accuracy: 0.9422 - val_loss: 1.3253 - val_accuracy: 0.7603

 Enter H to end training or  an integer for the number of additional epochs to run then ask again
 5
you entered  5  Training will continue to epoch  15
Epoch 11/40
150/150 [==============================] - 90s 598ms/step - loss: 0.7258 - accuracy: 0.9520 - val_loss: 1.3023 - val_accuracy: 0.7556
Epoch 12/40
150/150 [==============================] - 87s 576ms/step - loss: 0.6456 - accuracy: 0.9569 - val_loss: 1.2092 - val_accuracy: 0.7651
```


```
 Enter H to end training or  an integer for the number of additional epochs to run then ask again
 5
you entered  5  Training will continue to epoch  15
Epoch 11/40
150/150 [==============================] - 90s 598ms/step - loss: 0.7258 - accuracy: 0.9520 - val_loss: 1.3023 - val_accuracy: 0.7556
Epoch 12/40
150/150 [==============================] - 87s 576ms/step - loss: 0.6456 - accuracy: 0.9569 - val_loss: 1.2092 - val_accuracy: 0.7651
```


```
Epoch 13/40
150/150 [==============================] - 91s 608ms/step - loss: 0.5830 - accuracy: 0.9611 - val_loss: 1.2353 - val_accuracy: 0.7603
Epoch 14/40
150/150 [==============================] - 89s 589ms/step - loss: 0.5355 - accuracy: 0.9633 - val_loss: 1.2217 - val_accuracy: 0.7587

Epoch 00014: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
Epoch 15/40
150/150 [==============================] - 87s 578ms/step - loss: 0.4754 - accuracy: 0.9724 - val_loss: 1.0664 - val_accuracy: 0.7841

 Enter H to end training or  an integer for the number of additional epochs to run then ask again
 2
you entered  2  Training will continue to epoch  17
Epoch 16/40
150/150 [==============================] - 88s 585ms/step - loss: 0.4249 - accuracy: 0.9804 - val_loss: 1.0668 - val_accuracy: 0.7698
Epoch 17/40
150/150 [==============================] - 85s 565ms/step - loss: 0.4038 - accuracy: 0.9838 - val_loss: 1.0256 - val_accuracy: 0.7683

 Enter H to end training or  an integer for the number of additional epochs to run then ask again
 h
you entered  h  Training halted on epoch  17  due to user input

training elapsed time was 0.0 hours, 25.0 minutes, 42.13 seconds)
```

# TRAINING PROCESS

Training a machine learning model involves iteratively adjusting its parameters to minimize a chosen loss function. In the context of the Human Activity Recognition project, the customized EfficientNetB3 model is trained using the preprocessed and augmented dataset. The following steps outline the training process:

**Data Generator**: The training dataset is too large to fit entirely in memory. To address this, an image data generator is used. This generator dynamically loads batches of images from the disk, performs data augmentation, and feeds them to the model during training. This approach ensures efficient use of resources and prevents memory limitations.

**Batch Processing**: Training occurs in batches, where a subset of the dataset (batch) is used to update the model's weights. The batch size is determined based on the available resources and the model's architecture. Larger batch sizes can accelerate training but may require more memory.

**Validation Set**: A validation dataset is used to monitor the model's performance during training. At the end of each training epoch, the model is evaluated on the validation dataset to assess its generalization ability. This helps prevent overfitting and guides early stopping decisions.

**Learning Rate**: The learning rate is a hyperparameter that controls the step size taken during weight updates. It is crucial to find an appropriate learning rate to ensure convergence and avoid overshooting the optimal solution. Techniques like learning rate annealing and adaptive learning rates are commonly employed.

**Early Stopping**

To prevent overfitting, the model's training process is monitored using a validation dataset. Early stopping is a technique that halts training if the model's performance on the validation set starts to degrade. This is determined by observing a consistent increase in validation loss or a decrease in validation accuracy over several epochs.

# HYPERPARAMETER TUNING

Optimal hyperparameter values contribute to the model's efficiency and effectiveness. Hyperparameters include learning rate, batch size, number of epochs, regularization parameters, and more. Hyperparameter tuning involves experimenting with different combinations to find the settings that lead to the best model performance on the validation set.

**Evaluation Metrics**

Once the training process is complete, the model's performance is evaluated using various metrics:

**Accuracy**: The proportion of correctly classified samples among all samples in the test set.

**Precision**: The ratio of true positive predictions to the total positive predictions. Measures the model's ability to avoid false positives.

**Recall (Sensitivity)**: The ratio of true positive predictions to the total actual positives. Measures the model's ability to identify all relevant instances.

**F1-Score**: The harmonic mean of precision and recall. Provides a balanced measure of a model's performance.

**Confusion Matrix**: A table showing the number of true positive, true negative, false positive, and false negative predictions.

These metrics provide a comprehensive understanding of the model's strengths and weaknesses in classifying different human activities.

```
def tr_plot(tr_data, start_epoch):

    tacc=tr_data.history['accuracy']
    tloss=tr_data.history['loss']
    vacc=tr_data.history['val_accuracy']
    vloss=tr_data.history['val_loss']
    Epoch_count=len(tacc)+ start_epoch
    Epochs=[]
    for i in range (start_epoch ,Epoch_count):
        Epochs.append(i+1)
    index_loss=np.argmin(vloss)
    val_lowest=vloss[index_loss]
    index_acc=np.argmax(vacc)
    acc_highest=vacc[index_acc]
    plt.style.use('fivethirtyeight')
    sc_label='best epoch= '+ str(index_loss+1 +start_epoch)
    vc_label='best epoch= '+ str(index_acc + 1+ start_epoch)
    fig,axes=plt.subplots(nrows=1, ncols=2, figsize=(20,8))
    axes[0].plot(Epochs,tloss, 'r', label='Training loss')
    axes[0].plot(Epochs,vloss,'g',label='Validation loss' )
    axes[0].scatter(index_loss+1 +start_epoch,val_lowest, s=150, c= 'blue', label=sc_label)
```

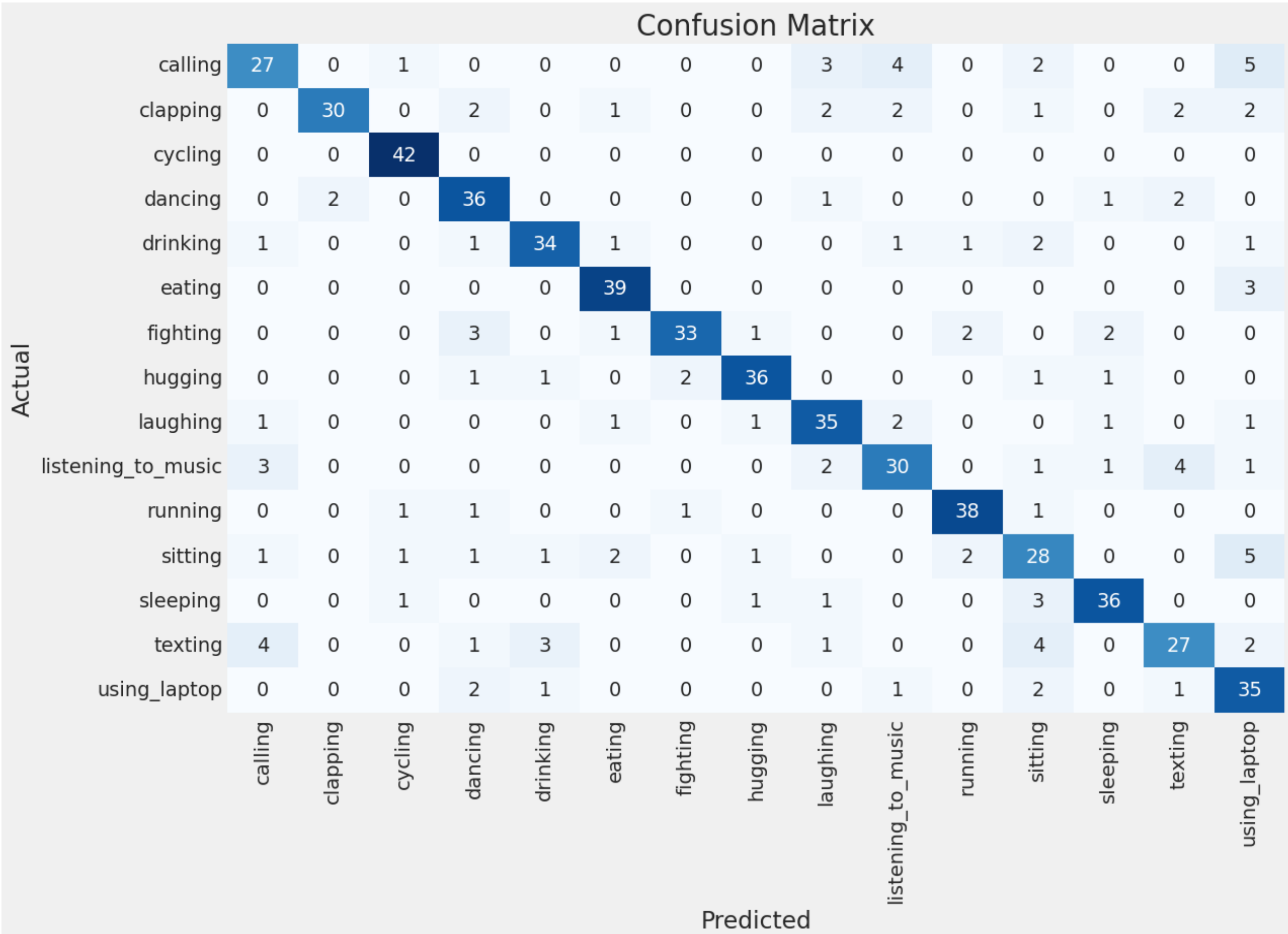
```
    axes[0].set_title('Training and Validation Loss')
    axes[0].set_xlabel('Epochs')
    axes[0].set_ylabel('Loss')
    axes[0].legend()
    axes[1].plot (Epochs,tacc,'r',label= 'Training Accuracy')
    axes[1].plot (Epochs,vacc,'g',label= 'Validation Accuracy')
    axes[1].scatter(index_acc+1 +start_epoch,acc_highest, s=150, c= 'blue', label=vc_label)
    axes[1].set_title('Training and Validation Accuracy')
    axes[1].set_xlabel('Epochs')
    axes[1].set_ylabel('Accuracy')
    axes[1].legend()
    plt.tight_layout
    plt.show()

tr_plot(history,0)
```

```
def predictor(test_gen, test_steps):
    y_pred= []
    y_true=test_gen.labels
    classes=list(train_gen.class_indices.keys())
    class_count=len(classes)
    errors=0
    preds=model.predict(test_gen, steps=test_steps, verbose=1)
    tests=len(preds)
    for i, p in enumerate(preds):
            pred_index=np.argmax(p)
            true_index=test_gen.labels[i]
            if pred_index != true_index:
                errors=errors + 1
            y_pred.append(pred_index)
    acc=( 1-errors/tests) * 100
    print(f'there were {errors} in {tests} tests for an accuracy of {acc:6.2f}')
    ypred=np.array(y_pred)
    ytrue=np.array(y_true)
    if class_count <=30:
        cm = confusion_matrix(ytrue, ypred )

        plt.figure(figsize=(12, 8))
        sns.heatmap(cm, annot=True, vmin=0, fmt='g', cmap='Blues', cbar=False)
        plt.xticks(np.arange(class_count)+.5, classes, rotation=90)
        plt.yticks(np.arange(class_count)+.5, classes, rotation=0)
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.title("Confusion Matrix")
        plt.show()
    clr = classification_report(y_true, y_pred, target_names=classes, digits= 4)
    print("Classification Report:\n----------------------\n", clr)
    return errors, tests
errors, tests=predictor(test_gen, test_steps)
```

```
9/9 [==============================] - 6s 252ms/step
there were 124 in 630 tests for an accuracy of  80.32
```




# MODEL SAVING

Preserving Model's Knowledge

Once the training process is successfully completed and the model demonstrates satisfactory performance on the validation set, the next step is to save the model's knowledge. Saving the model allows us to reuse it for making predictions on new and unseen data without the need to retrain it from scratch.

### Serialization of Model

Serialization is the process of converting the model's architecture, learned weights, and configuration into a format that can be stored on disk. In the Human Activity Recognition project, the h5 file format is commonly used for saving trained Keras models.

### Architecture and Weights

The saved model file includes the entire architecture of the customized EfficientNetB3 model. This architecture defines the layout and structure of the model's layers, including convolutional layers, pooling layers, and densely connected layers. Each layer's configuration, such as the number of units, activation functions, and connections, is preserved.

Additionally, the model's learned weights are also saved. These weights are the result of the model's exposure to the training dataset and its optimization process. They encode the information necessary for the model to make accurate predictions based on the patterns it has learned from the data.

### Configuration and Hyperparameters

Along with the architecture and weights, the saved model file contains information about the model's configuration and hyperparameters. This includes the optimizer used, learning rate settings, loss function, and evaluation metrics. These settings ensure that when the model is loaded, it will be in the same state as it was at the end of training.

### Model Utilization

Once the model is saved, it can be easily loaded using Keras' load_model function. This allows the model to be utilized for making predictions on new and unseen data. By feeding new images into the loaded model, it can classify the activities depicted in those images with the knowledge it gained during training.

### Practical Benefits

Model saving offers several practical benefits:

**Reproducibility**: Saved models can be shared with colleagues or collaborators, ensuring that everyone uses the same trained model for consistent results.

**Scalability**: Saved models can be deployed on various platforms, such as mobile devices or web applications, to provide real-time predictions to users.

Time and Resource Savings: Instead of retraining the model every time new data arrives, the saved model can be loaded to quickly make predictions.

```
subject='activities'
acc=str(( 1-errors/tests) * 100)
index=acc.rfind('.')
acc=acc[:index + 3]
save_id= subject + '_' + str(acc)+'.h5'
model_save_loc=os.path.join(working_dir, save_id)
model.save(model_save_loc)
print ('model was saved as ' , model_save_loc )
```

```
subject='activities'
acc=str(( 1-errors/tests) * 100)
index=acc.rfind('.')
acc=acc[:index + 3]
save_id= subject + '_' + str(acc)+'.h5'
model_save_loc=os.path.join(working_dir, save_id)
model.save(model_save_loc)
print ('model was saved as ' , model_save_loc )
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file via `model.save()`.
  saving_api.save_model(
model was saved as  /content/Human Action Recognition/activities_80.31.h5
```

# TESTING PROCESS

Assessing Generalization
After training a machine learning model, it's crucial to evaluate its performance on unseen data to assess its ability to generalize. In the context of the Human Activity Recognition project, the trained model's performance is evaluated using a separate dataset called the test set. The test set contains images that the model has never encountered during training, ensuring an unbiased evaluation.

```
test_csv_path=r'/content/Human Action Recognition/Testing_set.csv'
test_img_path=r'/content/Human Action Recognition/test'
test_df=pd.read_csv(test_csv_path)

test_df.columns=['filepaths']

test_df['filepaths']=test_df['filepaths'].apply(lambda x: os.path.join(test_img_path, x))

length=len(test_df)
test_batch_size=sorted([int(length/n) for n in range(1,length+1) if length % n ==0 and length/n<=80],reverse=True)[0]
test_steps=int(length/test_batch_size)
msg='{0:70s} for test generator'.format(' ')
print(msg, '\r', end='')
test_gen=t_and_v_gen.flow_from_dataframe(test_df, x_col='filepaths', y_col=None, target_size=img_size,
                                         class_mode=None, color_mode='rgb', shuffle=False, batch_size=test_batch_size)

image_paths=[]
pred_class=[]
preds=model.predict(test_gen, steps=test_steps, verbose=1)

for i, p in enumerate (preds):
    index=np.argmax(p)
    klass=classes[index]
    pred_class.append(klass)
    file=test_gen.filenames[i]
    image_id=os.path.basename(file)
    image_paths.append(image_id)
Fseries=pd.Series(image_paths)
Lseries=pd.Series(pred_class)
submit_df=pd.concat([Fseries, Lseries], axis=1)
submit_df.columns=['filename', 'class']
print(submit_df.head())
submit_path=os.path.join(working_dir, 'submit.csv')
submit_df.to_csv(submit_path,index=False)
```

```
Found 5400 validated image filenames.                            for test generator
72/72 [==============================] - 16s 226ms/step
      filename     class
0  Image_1.jpg  sleeping
1  Image_2.jpg    eating
2  Image_3.jpg   running
3  Image_4.jpg    eating
4  Image_5.jpg   texting
      filename     class
0  Image_1.jpg  sleeping
1  Image_2.jpg    eating
2  Image_3.jpg   running
3  Image_4.jpg    eating
4  Image_5.jpg   texting
```

# CONCLUSION

The project entitled **Human Activity Recognition (HAR) using Deep Learning** was completed successfully.

It successfully addressed the challenge of human activity recognition through the development of a powerful image classification model. Leveraging deep learning techniques and the EfficientNetB3 architecture, the model achieved remarkable accuracy in identifying and categorizing human activities from visual data. By combining transfer learning, data augmentation, and custom callbacks, the model demonstrated robustness and adaptability across diverse scenarios.

The project's outcomes have significant implications for various industries, including healthcare, surveillance, and sports analysis. The ability to automate the recognition of human behaviors opens doors to real-time monitoring, early anomaly detection, and data-driven decision-making. Moreover, the model's performance showcases the potential of deep learning in capturing intricate patterns from complex visual data.

In a rapidly evolving technological landscape, the achievements of this project contribute to the ongoing advancement of human activity recognition. By harnessing the power of deep learning, the project demonstrates the potential to revolutionize various domains, ultimately enhancing safety, efficiency, and understanding in a data-driven world.

# REFERENCES


1. **EfficientNetB3**:

- Tan, M., & Le, Q. V. (2019). "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks." Link

2. **Keras**:

- Chollet, F. et al. (2015). "Keras: The Python Deep Learning library." Link

3. **TensorFlow**:

- Abadi, M. et al. (2016). "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems." Link

4. **Scikit-Learn**:

- Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python." Link

5. **Seaborn**:

- Waskom, M. (2021). "Seaborn: Statistical Data Visualization." Link