National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics

Rohan Kanagal Sathyanarayana
Student ID: x19203829

School of Computing
National College of Ireland

Supervisor: Mayank Jain

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. Rohan Kanagal Sathyanarayana ……

Student ID: …… x19203829 ……

Programme: ……MSc Data Analytics…… **Year:** ……2023-24…….

Module: ……Research Project ……

Lecturer: …… Mayank Jain ……

Submission Due Date: …… 14 – December – 2023 ……

Project Title: CricNet: A Deep Learning Network to enhance the cricketing Analysis

Word Count: 501

Page Count: 16

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Rohan K S

Date: 14 – December - 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□✓
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□✓
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□✓

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Rohan Kanagal Sathyanarayana
X19203829@student.ncirl.ie

1. Introduction

The objective of this manual configuration handbook aims to provide an overview of the steps and configuration implemented during the present research project. This manual provides comprehensive details regarding both the hardware and software setups, and also about the libraries that are utilized in the project, it further discusses the programming approach and also the method that is used whereas executing the code.

2. Hardware/Software Requirements

Operating System	**Windows 10**
RAM	**16 GB**
Processor	**Core i7 Intel**
Graphics	**T4 Engine**
Platform	**Google Colab**

3. Code Running Steps

Log into the Google drive
Download the data and upload into the My Drive
Login to Google Colab
Connect the Google Drive to the Colab
Upload the code and run it

1. Data understanding

```python
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Importing necessary libraries
import cv2
import numpy as np
import os
from tqdm import tqdm
import matplotlib.pyplot as plt
import seaborn as sns
import random
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, BatchNormalization, Conv2D, MaxPooling2D
from tensorflow.keras.regularizers import l2
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import pickle
from tensorflow.keras.applications import VGG19, ResNet50, MobileNet, Xception, InceptionV3
from efficientnet.tfkeras import EfficientNetB0
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

2. Data preparation

```python
# Define dataset paths
pullshot = "/content/drive/MyDrive/Dataset/pullshot"
sweep = "/content/drive/MyDrive/Dataset/sweep"
drive = "/content/drive/MyDrive/Dataset/drive"
legglance = "/content/drive/MyDrive/Dataset/legglance-flick"

# Function to assign labels to images based on shot type
def assign_label(img, short_type):
    return short_type

# Function to add cricket shots to the training data
def add_shorts_to_train_data(short_type, DIR):
    for img in tqdm(os.listdir(DIR)):
        label = assign_label(img, short_type)
        path = os.path.join(DIR, img)
        img = cv2.imread(path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (150, 150))
        X.append(np.array(img))
        Z.append(str(label))

# Create empty lists for images (X) and labels (Z)
X = []
Z = []

# Add different cricket shots to the training data
add_shorts_to_train_data("pullshot", pullshot)
add_shorts_to_train_data("sweep", sweep)
add_shorts_to_train_data("drive", drive)
add_shorts_to_train_data("legglance-flick", legglance)

# Visualize a subset of the dataset
```

```python
# Visualize a subset of the dataset
f, ax = plt.subplots(4, 4)
f.set_size_inches(12, 12)
n = len(Z)

for i in range(4):
    for j in range(4):
        idx = random.randint(0, n)
        ax[i, j].imshow(X[idx])
        ax[i, j].set_title(Z[idx])

plt.tight_layout()

# Label Encoding for the dataset

le = LabelEncoder()
Y = le.fit_transform(Z)
Y = to_categorical(Y, 4)

# Split the data into training and testing sets

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=42, stratify=Y)
```

4. Modeling

3.1 Convolutional Neural Network (CNN):

```
# Defining the parameters
num_classes = 4
batch_size = 32

# Initialize the model
model = Sequential()

# Building the CNN Model (Hidden Output)
model = Sequential()
model.add(Conv2D(64, (3, 3), padding='same',
input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512,kernel_regularizer=l2(0.01)))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

```
# Display the model summary
model.summary()

# Compile the model with categorical crossentropy loss and Adam optimizer
model.compile(loss='categorical_crossentropy',optimizer='sgd',metrics=['accuracy'])

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalizing the input image
x_train /= 255
x_test /= 255
epochs=20

# Training the model
history = model.fit(x_train, y_train,
            batch_size=batch_size,epochs=epochs,
            validation_data=(x_test, y_test),
            shuffle=True)
```

3.2 DenseNet:

```
# Describing Epochs
epochs = 10

# Builds a new model by adding GlobalAveragePooling and a Dense output layer
def build_model(bottom_model, classes):
    model = bottom_model.layers[-2].output
    model = GlobalAveragePooling2D()(model)
    model = Dense(classes, activation = 'softmax', name = 'out_layer')(model)

    return model

"""# DenseNet"""

# Load DenseNet pre-trained weights with input shape (150, 150, 3)
res = tf.keras.applications.DenseNet121(weights = 'imagenet',
                                        include_top = False,
                                        input_shape = (150, 150, 3))
head = build_model(res, num_classes)

den_model = Model(inputs = res.input, outputs = head)

early_stopping = EarlyStopping(monitor = 'val_accuracy',
                               min_delta = 0.00005,
                               patience = 11,
                               verbose = 1,
                               restore_best_weights = True,)

lr_scheduler = ReduceLROnPlateau(monitor = 'val_accuracy',
                                 factor = 0.5,
                                 patience = 7,
                                 min_lr = 1e-7,
                                 verbose = 1,)
```

```
callbacks = [early_stopping,lr_scheduler,]

train_datagen = ImageDataGenerator(rotation_range = 15,
                                   width_shift_range = 0.15,
                                   height_shift_range = 0.15,
                                   shear_range = 0.15,
                                   zoom_range = 0.15,
                                   horizontal_flip = True,)
train_datagen.fit(x_train)

optims = [optimizers.Adam(learning_rate = 0.0001, beta_1 = 0.9, beta_2 = 0.999),]

# Compile the model
den_model.compile(loss = 'categorical_crossentropy',
                  optimizer = optims[0],
                  metrics = ['accuracy'])

history = den_model.fit(train_datagen.flow(x_train,
                                           y_train,
                                           batch_size = batch_size),
                                           validation_data = (x_test, y_test),
                                           steps_per_epoch = len(x_train) / batch_size,
                                           epochs = epochs,
                                           callbacks = callbacks,
                                           use_multiprocessing = True)

yhat_valid_den = np.argmax(den_model.predict(x_test), axis=1)
```

3.3 EfficientNet:

```
#Effiencientnet
res = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
head = build_model(res, num_classes)

model = Model(inputs = res.input, outputs = head)

early_stopping = EarlyStopping(monitor = 'val_accuracy',
                               min_delta = 0.00005,
                               patience = 11,
                               verbose = 1,
                               restore_best_weights = True,)

lr_scheduler = ReduceLROnPlateau(monitor = 'val_accuracy',
                                 factor = 0.5,
                                 patience = 7,
                                 min_lr = 1e-7,
                                 verbose = 1,)

callbacks = [early_stopping,lr_scheduler,]

train_datagen = ImageDataGenerator(rotation_range = 15,
                                   width_shift_range = 0.15,
                                   height_shift_range = 0.15,
                                   shear_range = 0.15,
                                   zoom_range = 0.15,
                                   horizontal_flip = True,)
train_datagen.fit(x_train)

optims = [optimizers.Adam(learning_rate = 0.0001, beta_1 = 0.9, beta_2 = 0.999),]
```

```
# Compile the model
model.compile(loss = 'categorical_crossentropy',
              optimizer = optims[0],
              metrics = ['accuracy'])

history = model.fit(train_datagen.flow(x_train,
                                       y_train,
                                       batch_size = batch_size),
                                       validation_data = (x_test, y_test),
                                       steps_per_epoch = len(x_train) / batch_size,
                                       epochs = epochs,
                                       callbacks = callbacks,
                                       use_multiprocessing = True)

yhat_valid_eff = np.argmax(model.predict(x_test), axis=1)
```

3.4 InceptionV3:

```
'''# InceptionV3'''
res = InceptionV3(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
head = build_model(res, num_classes)

model = Model(inputs = res.input, outputs = head)

early_stopping = EarlyStopping(monitor = 'val_accuracy',
                               min_delta = 0.00005,
                               patience = 11,
                               verbose = 1,
                               restore_best_weights = True,)

lr_scheduler = ReduceLROnPlateau(monitor = 'val_accuracy',
                                 factor = 0.5,
                                 patience = 7,
                                 min_lr = 1e-7,
                                 verbose = 1,)

callbacks = [early_stopping,lr_scheduler,]

train_datagen = ImageDataGenerator(rotation_range = 15,
                                   width_shift_range = 0.15,
                                   height_shift_range = 0.15,
                                   shear_range = 0.15,
                                   zoom_range = 0.15,
                                   horizontal_flip = True,)
train_datagen.fit(x_train)

optims = [optimizers.Adam(learning_rate = 0.0001, beta_1 = 0.9, beta_2 = 0.999),]
```

```
# Compile the model
model.compile(loss = 'categorical_crossentropy',
              optimizer = optims[0],
              metrics = ['accuracy'])

history = model.fit(train_datagen.flow(x_train,
                                       y_train,
                                       batch_size = batch_size),
                                       validation_data = (x_test, y_test),
                                       steps_per_epoch = len(x_train) / batch_size,
                                       epochs = epochs,
                                       callbacks = callbacks,
                                       use_multiprocessing = True)

yhat_valid_in = np.argmax(model.predict(x_test), axis=1)
```

3.5. ResNet:

```
"""# ResNet"""
res = tf.keras.applications.ResNet50(weights = 'imagenet',
                                     include_top = False,
                                     input_shape = (150, 150, 3))

head = build_model(res, num_classes)

model = Model(inputs = res.input, outputs = head)

early_stopping = EarlyStopping(monitor = 'val_accuracy',
                               min_delta = 0.00005,
                               patience = 11,
                               verbose = 1,
                               restore_best_weights = True,)

lr_scheduler = ReduceLROnPlateau(monitor = 'val_accuracy',
                                 factor = 0.5,
                                 patience = 7,
                                 min_lr = 1e-7,
                                 verbose = 1,)

callbacks = [early_stopping,lr_scheduler,]
train_datagen = ImageDataGenerator(rotation_range = 15,
                                   width_shift_range = 0.15,
                                   height_shift_range = 0.15,
                                   shear_range = 0.15,
                                   zoom_range = 0.15,
                                   horizontal_flip = True,)
train_datagen.fit(x_train)

optims = [optimizers.Adam(learning_rate = 0.0001, beta_1 = 0.9, beta_2 = 0.999),]
```

```
# Compile the model
model.compile(loss = 'categorical_crossentropy',
              optimizer = optims[0],
              metrics = ['accuracy'])

history = model.fit(train_datagen.flow(x_train,
                                       y_train,
                                       batch_size = batch_size),
                    validation_data = (x_test, y_test),
                    steps_per_epoch = len(x_train) / batch_size,
                    epochs = epochs,
                    callbacks = callbacks,
                    use_multiprocessing = True)

yhat_valid_res = np.argmax(model.predict(x_test), axis=1)
```

3.5 Xception:

```
'''# Xception'''
res = Xception(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
head = build_model(res, num_classes)

model = Model(inputs = res.input, outputs = head)

early_stopping = EarlyStopping(monitor = 'val_accuracy',
                               min_delta = 0.00005,
                               patience = 11,
                               verbose = 1,
                               restore_best_weights = True,)

lr_scheduler = ReduceLROnPlateau(monitor = 'val_accuracy',
                                 factor = 0.5,
                                 patience = 7,
                                 min_lr = 1e-7,
                                 verbose = 1,)

callbacks = [early_stopping,lr_scheduler,]

train_datagen = ImageDataGenerator(rotation_range = 15,
                                   width_shift_range = 0.15,
                                   height_shift_range = 0.15,
                                   shear_range = 0.15,
                                   zoom_range = 0.15,
                                   horizontal_flip = True,)
train_datagen.fit(x_train)

optims = [optimizers.Adam(learning_rate = 0.0001, beta_1 = 0.9, beta_2 = 0.999),]
```

```
# Compile the model
model.compile(loss = 'categorical_crossentropy',
              optimizer = optims[0],
              metrics = ['accuracy'])

history = model.fit(train_datagen.flow(x_train,
                                       y_train,
                                       batch_size = batch_size),
                                       validation_data = (x_test, y_test),
                                       steps_per_epoch = len(x_train) / batch_size,
                                       epochs = epochs,
                                       callbacks = callbacks,
                                       use_multiprocessing = True)

yhat_valid_exe = np.argmax(model.predict(x_test), axis=1)
```

3.6 VGG19:

```
"""# VGG19"""
vgg = tf.keras.applications.VGG19(weights = 'imagenet',
                                  include_top = False,
                                  input_shape = (150, 150, 3))

head = build_model(vgg, num_classes)
model = Model(inputs = vgg.input, outputs = head)
early_stopping = EarlyStopping(monitor = 'val_accuracy',
                               min_delta = 0.00005,
                               patience = 11,
                               verbose = 1,
                               restore_best_weights = True,)
lr_scheduler = ReduceLROnPlateau(monitor = 'val_accuracy',
                                 factor = 0.5,
                                 patience = 7,
                                 min_lr = 1e-7,
                                 verbose = 1,)

callbacks = [early_stopping,lr_scheduler,]

train_datagen = ImageDataGenerator(rotation_range = 15,
                                   width_shift_range = 0.15,
                                   height_shift_range = 0.15,
                                   shear_range = 0.15,
                                   zoom_range = 0.15,
                                   horizontal_flip = True,)
train_datagen.fit(x_train)

optims = [optimizers.Adam(learning_rate = 0.0001, beta_1 = 0.9, beta_2 = 0.999),]
```

```
# Compile the model
model.compile(loss = 'categorical_crossentropy',
              optimizer = optims[0],
              metrics = ['accuracy'])

history = model.fit(train_datagen.flow(x_train,
                                       y_train,
                                       batch_size = batch_size),
                    validation_data = (x_test, y_test),
                    steps_per_epoch = len(x_train) / batch_size,
                    epochs = epochs,
                    callbacks = callbacks,
                    use_multiprocessing = True)
```

3.7 MobileNet:

```
"""# MobileNet"""

res = tf.keras.applications.MobileNet(weights = 'imagenet',
                                       include_top = False,
                                       input_shape = (150, 150, 3))

head = build_model(res, num_classes)
model = Model(inputs = res.input, outputs = head)

early_stopping = EarlyStopping(monitor = 'val_accuracy',
                               min_delta = 0.00005,
                               patience = 11,
                               verbose = 1,
                               restore_best_weights = True,)

lr_scheduler = ReduceLROnPlateau(monitor = 'val_accuracy',
                                 factor = 0.5,
                                 patience = 7,
                                 min_lr = 1e-7,
                                 verbose = 1,)

callbacks = [early_stopping,lr_scheduler]

train_datagen = ImageDataGenerator(rotation_range = 15,
                                   width_shift_range = 0.15,
                                   height_shift_range = 0.15,
                                   shear_range = 0.15,
                                   zoom_range = 0.15,
                                   horizontal_flip = True,)
train_datagen.fit(x_train)
optims = [optimizers.Adam(learning_rate = 0.0001, beta_1 = 0.9, beta_2 = 0.999),]
```

```
# Compile the model
model.compile(loss = 'categorical_crossentropy',
              optimizer = optims[0],
              metrics = ['accuracy'])

history = model.fit(train_datagen.flow(x_train,
                                       y_train,
                                       batch_size = batch_size),
                                       validation_data = (x_test, y_test),
                                       steps_per_epoch = len(x_train) / batch_size,
                                       epochs = epochs,
                                       callbacks = callbacks,
                                       use_multiprocessing = True)

yhat_valid_mob = np.argmax(model.predict(x_test), axis=1)
```

5. Evaluation

```
# Generate predictions for the test set
yhat_valid_mob = np.argmax(model.predict(x_test), axis=1)

# Generate predictions for the test set
yhat_valid_vgg = np.argmax(model.predict(x_test), axis=1)

# Generate predictions for the test set
yhat_valid_res = np.argmax(model.predict(x_test), axis=1)

# Generate predictions for the test set
yhat_valid_eff = np.argmax(model.predict(x_test), axis=1)

# Generate predictions for the test set
yhat_valid_den = np.argmax(model.predict(x_test), axis=1)

# Generate predictions for the test set
yhat_valid_exe = np.argmax(model.predict(x_test), axis=1)

# Generate predictions for the test set
yhat_valid_in = np.argmax(model.predict(x_test), axis=1)
```

```
# Calculates and organizes evaluation metrics for different models
y_true = np.argmax(y_test, axis=1)
accuracy_vgg = accuracy_score(y_true, yhat_valid_vgg)
precision_vgg = precision_score(y_true, yhat_valid_vgg, average='weighted')
recall_vgg = recall_score(y_true, yhat_valid_vgg, average='weighted')
f1_vgg = f1_score(y_true, yhat_valid_vgg, average='weighted')

accuracy_res = accuracy_score(y_true, yhat_valid_res)
precision_res = precision_score(y_true, yhat_valid_res, average='weighted')
recall_res = recall_score(y_true, yhat_valid_res, average='weighted')
f1_res = f1_score(y_true, yhat_valid_res, average='weighted')

accuracy_mob = accuracy_score(y_true, yhat_valid_mob)
precision_mob = precision_score(y_true, yhat_valid_mob, average='weighted')
recall_mob = recall_score(y_true, yhat_valid_mob, average='weighted')
f1_mob = f1_score(y_true, yhat_valid_mob, average='weighted')

accuracy_eff = accuracy_score(y_true, yhat_valid_eff)
precision_eff = precision_score(y_true, yhat_valid_eff, average='weighted')
recall_eff = recall_score(y_true, yhat_valid_eff, average='weighted')
f1_eff = f1_score(y_true, yhat_valid_eff, average='weighted')

accuracy_den = accuracy_score(y_true, yhat_valid_den)
precision_den = precision_score(y_true, yhat_valid_den, average='weighted')
recall_den = recall_score(y_true, yhat_valid_den, average='weighted')
f1_den = f1_score(y_true, yhat_valid_den, average='weighted')

accuracy_exe = accuracy_score(y_true, yhat_valid_exe)
precision_exe = precision_score(y_true, yhat_valid_exe, average='weighted')
recall_exe = recall_score(y_true, yhat_valid_exe, average='weighted')
f1_exe = f1_score(y_true, yhat_valid_exe, average='weighted')
```

```
accuracy_in = accuracy_score(y_true, yhat_valid_in)
precision_in = precision_score(y_true, yhat_valid_in, average='weighted')
recall_in = recall_score(y_true, yhat_valid_in, average='weighted')
f1_in = f1_score(y_true, yhat_valid_in, average='weighted')


data = {
    'Model': ['VGGNet', 'ResNet', 'MobileNet', 'EfficientNet', 'DenseNet','Xception','InceptionV3'],
    'Accuracy': [accuracy_vgg, accuracy_res, accuracy_mob, accuracy_eff, accuracy_den,accuracy_exe,accuracy_in],
    'Precision': [precision_vgg, precision_res, precision_mob, precision_eff, precision_den,precision_exe,precision_in],
    'Recall': [recall_vgg, recall_res, recall_mob, recall_eff, recall_den,recall_exe,recall_in],
    'F1 Score': [f1_vgg, f1_res, f1_mob, f1_eff, f1_den,f1_exe,f1_in]
    }

    # Create a DataFrame
df = pd.DataFrame(data)

    # Set 'Model' as the index
df.set_index('Model', inplace=True)

df = df.sort_values(by='Accuracy', ascending=False)
```

6. Gesture Recognition using Detectron2.

The following code needs to be run for the Gesture detection,

5.1 Data Preparation:

```
# Load images and labels from the specified path
path = '/content/drive/MyDrive/Dataset'
folders = os.listdir(path)

images = []
labels = []

for folder in folders:
    names = os.listdir(os.path.join(path, folder))

    for name in names:
        try:
            img = cv2.imread(os.path.join(path, folder, name))

            if img is not None:
                images.append(img)
                labels.append(folder)
            else:
                print(f"Error loading image: {os.path.join(path, folder, name)}")
        except Exception as e:
            print(f"Error processing image: {os.path.join(path, folder, name)} - {e}")
```

5.2 Gesture Recognition with Detectron2:

```
# Import necessary libraries and set up Detectron2
!pip3 install torch torchvision torchaudio
!git clone https://github.com/facebookresearch/detectron2.git
!pip install -e detectron2

from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg

# Set up configuration for the Keypoint RCNN model
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Keypoints/keypoint_rcnn_R_101_FPN_3x.yaml"))
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Keypoints/keypoint_rcnn_R_101_FPN_3x.yaml")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.8

# Create a predictor
predictor = DefaultPredictor(cfg)
```

5.3 Gesture Visualization and Analysis:

```python
import numpy as np

# Function to calculate the angle between three points
def calculate_angle(point1, point2, point3):
    vector1 = np.array(point1) - np.array(point2)
    vector2 = np.array(point3) - np.array(point2)
    cosine_angle = np.dot(vector1, vector2) / (np.linalg.norm(vector1) * np.linalg.norm(vector2))
    angle = np.arccos(cosine_angle)
    angle_deg = np.degrees(angle)
    return angle_deg

# for drawing predictions on images
from detectron2.utils.visualizer import Visualizer
# to obtain metadata
from detectron2.data import MetadataCatalog
# to display an image
from google.colab.patches import cv2_imshow
# randomly select images
for img in random.sample(images, 5):
    # make predictions
    outputs = predictor(img)

    # Extract kinematics instances from predictions
    kinematics_instances = outputs["instances"].to("cpu")

    # Check if "pred_keypoints" is present in the instance fields
    if "pred_keypoints" in kinematics_instances.get_fields():
        # Check if the list of keypoints is not empty
        if len(kinematics_instances.pred_keypoints) > 0:
            # Create a copy of the original image
            original_img = img.copy()
```

```python
            # use `Visualizer` to draw the original image
            v = Visualizer(original_img[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1)
            v = v.draw_instance_predictions(outputs["instances"].to("cpu"))
            original_image_with_predictions = v.get_image()[:, :, ::-1]

            # use `Visualizer` to draw only kinematics on a separate image
            v_kinematics = Visualizer(np.zeros_like(original_img), MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1)
            v_kinematics = v_kinematics.draw_instance_predictions(kinematics_instances)
            kinematics_image = v_kinematics.get_image()[:, :, ::-1]

            # Get keypoints from kinematics_instances
            keypoints = kinematics_instances.pred_keypoints[0].numpy()

            # Calculate and display angles between keypoint pairs
            angle_elbow_hand_wrist = calculate_angle(keypoints[7], keypoints[9], keypoints[11])
            angle_shoulder_elbow_hand = calculate_angle(keypoints[5], keypoints[7], keypoints[9])

            print(f"Angle between elbow, hand, and wrist: {angle_elbow_hand_wrist:.2f} degrees")
            print(f"Angle between shoulder, elbow, and hand: {angle_shoulder_elbow_hand:.2f} degrees")

            # Display original image with predictions
            cv2_imshow(original_image_with_predictions)

            # Display kinematics image
            cv2_imshow(kinematics_image)
        else:
            print("No keypoints detected in this instance.")
    else:
        print("No keypoint information in this instance.")
```

3. Results to be evaluated and compared. [Please visit the results section of the dissertation]

References and Links

[1] Google Drive: https://drive.google.com/drive/my-drive
[2] Dataset Link: https://www.kaggle.com/datasets/aneesh10/cricket-shot-dataset
[3]GoogleColab:https://colab.research.google.com/drive/1E7J30sJcJPv6kuGtRZyQzOfoObhNbh3h
https://colab.research.google.com/drive/1DaXq84rOhb8V_PLuZVpUMzVoV-Mqfcmu