Configuration Manual

MSc Research Project
Cloud Computing

Deepak Chandrashekar
Student ID: 23221178

School of Computing
National College of Ireland

Supervisor: Abubakr Siddig

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 placedinto the assignment box located outside the office.

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

1 Introduction

The implementation of the project is below, as we all know about the Parking Issues which has created Issue as the cities becoming urbanized, we are facing the traffic and parking issues. I have implemented the smart parking solutions in which the driver would not face many difficulties for the parking, as he can use the search engine for looking for the available parking slot. As I have implemented using the Deep Learning model which can give an accurate result and also I have configured Load Balancer and Autoscaling Group for the server High Availability and Scaling. Below are the details:

System Requirements

- **Software:** Windows 10/11, Ubuntu 20.04, or macOS; Python 3.7+.
- **Dependencies:** Streamlit, Ultralytics, EasyOCR, OpenCV, NumPy, Pandas, Pytesseract, Pillow.
- **Tools:** Tesseract-OCR, CUDA Toolkit (for GPU).

```
from google.colab import drive
drive.mount('/content/gdrive')
```

- `from google.colab import drive`: Imports the Google Colab `drive` module.

- `drive.mount('/content/gdrive')`: Mounts the user's Google Drive at the path `/content/gdrive` to access files stored on Google Drive.

```
import numpy as np
import pandas as pd
import os
import PIL
from PIL import Image
import matplotlib.pyplot as plt
from matplotlib.image import imread
import matplotlib.image as mpimg
import cv2
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
```

- Various libraries are imported for the project:
  - **NumPy & Pandas**: For numerical computations and data manipulation.
  - **PIL (Pillow)**: To handle image processing.
  - **Matplotlib & OpenCV**: For visualization and image manipulation.
  - **TensorFlow & Keras**: For deep learning tasks.
  - **Scikit-learn (train_test_split)**: To split data into training and testing sets.
- Multiple modules from Matplotlib (`pyplot`, `image`) and TensorFlow are explicitly imported for convenience.

```
data_path = "/content/gdrive/MyDrive/data/data"
Categories = ["Free", "Full"]
img_size = 224
```

- `data_path`: Specifies the path to the dataset stored in Google Drive.
- `Categories`: A list defining the class labels (`"Free"` and `"Full"`) in the dataset.
- `img_size`: Sets the image size (224x224 pixels) for preprocessing.

```python
data = []
def create_data():
    for category in Categories:
        path = os.path.join(data_path,category)
        class_num = Categories.index(category)
        # print(path)
        for img in os.listdir(path):
            # print(img)
            img_arr = cv2.imread(os.path.join(path,img))
            new_img_arr = cv2.resize(img_arr,(img_size,img_size))
            data.append([new_img_arr,class_num])
create_data()
```

- Initializes an empty list `data` to store preprocessed images and their labels.
- **`create_data()` function**:

  - Iterates over the categories defined earlier.
  - Joins the `data_path` with the category name to locate images for each category.
  - Reads each image using `cv2.imread()`, resizes it to `img_size x img_size`, and appends the resized image along with its label (`class_num`) to the `data` list.

- The function is called with `create_data()` to populate the `data` list.

```python
for features, labels in data:
    X.append(features)
    Y.append(labels)
```

- Iterates through the `data` list.
- Separates features (images) into `X` and corresponding labels into `Y`.

```python
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.20,random_state=42)
```

- Uses `train_test_split` from Scikit-learn to split `X` and `Y` into training and testing sets.
- `test_size=0.20` specifies 20% of the data is used for testing, and `random_state=42` ensures reproducibility.

```python
def process_images(image):
    # Normalize images to have a mean of 0 and standard deviation of 1
    image = tf.image.per_image_standardization(image)
    # Resize images from 32x32 to 277x277
    image = tf.image.resize(image, (224,224))
    return image
```

- Uses `train_test_split` from Scikit-learn to split `X` and `Y` into training and testing sets.
- `test_size=0.20` specifies 20% of the data is used for testing, and `random_state=42` ensures reproducibility.

```
x_train = process_images(x_train)
x_test = process_images(x_test)
```

`process_images(image)` **function**:

- Normalizes the image to have a mean of 0 and a standard deviation of 1 using TensorFlow's `per_image_standardization`.
- Resizes the image to 224x224 pixels (matches earlier preprocessing).
- Returns the processed image.

```
x_train = np.array(x_train).reshape(-1, 224,224, 3)
y_train = np.array(y_train)
x_test = np.array(x_test).reshape(-1, 224,224, 3)
y_test = np.array(y_test)
```

- Converts training and testing datasets (`x_train`, `x_test`, `y_train`, `y_test`) into NumPy arrays.
- Reshapes `x_train` and `x_test` into a 4D shape: `(-1, 224, 224, 3)`, where:

  - `-1` allows for flexibility in batch size.
  - `224x224` is the image dimension.
  - `3` represents RGB color channels.

```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu', input_shape=(224,224,3)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Flatten(),
    keras.layers.Dense(4096, activation='relu', input_shape=(224,224,3)),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4096, activation='relu', input_shape=(224,224,3)),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax', input_shape=(224,224,3))
])
```

Constructs a CNN model using Keras' `Sequential` API:

- Multiple convolutional layers with different filter sizes (e.g., `96`, `256`) and kernel sizes (`11x11`, `5x5`, etc.).
- **ReLU activation** is applied after each convolutional layer.
- **Batch Normalization** stabilizes training by normalizing layer outputs.

- **MaxPooling** layers reduce spatial dimensions.
- Fully connected (`Dense`) layers are added at the end with `4096` neurons each.
- Dropout layers are applied to prevent overfitting.
- The final output layer uses a `softmax` activation for multiclass classification (`10` classes).

```
model.compile(loss='sparse_categorical_crossentropy', optimizer=tf.optimizers.SGD(learning_rate=0.001), metrics=['accuracy'])
model.summary()
```

- Specifies loss function: `sparse_categorical_crossentropy` (for integer labels).
- Optimizer: Stochastic Gradient Descent (SGD) with a learning rate of `0.001`.
- Metric: Model evaluates accuracy during training.

```
from keras import callbacks
earlystopping = callbacks.EarlyStopping(monitor ="val_loss",mode ="min", patience = 5,restore_best_weights = True)
```

- Defines an early stopping callback to halt training when validation loss stops improving for `5` epochs.
- `restore_best_weights=True` ensures the best weights are retained.

```
history = model.fit(x_train,y_train, epochs=20, validation_data = (x_validate,y_validate),callbacks =[earlystopping])
```

Trains the model using `model.fit()`:

- Inputs: Training data (`x_train`, `y_train`).
- Validation data: `(x_validate, y_validate)` to evaluate performance.
- Epochs: 20.
- Early stopping is passed as a callback to stop training when conditions are met.

```
# Plotting the Model Accuracy & Model Loss vs Epochs (Hidden Input)
plt.figure(figsize=[20,8])
# summarize history for accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy', size=25, pad=20)
plt.ylabel('Accuracy', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
# summarize history for loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss', size=25, pad=20)
plt.ylabel('Loss', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

- Visualizes model training metrics:

  - Subplot 1: Plots training and validation accuracy over epochs.
  - Subplot 2: Plots training and validation loss over epochs.

- Customizes plots with titles, labels, legends, and figure size.
- The purpose is to analyze model performance trends during training.

```
i = 0
prop_class = []
mis_class = []

for i in range(len(y_test)):
    if(y_test[i] == pred_digits[i]):
        prop_class.append(i)
    else:
        mis_class.append(i)

print("Properly Predicted: "+str(len(prop_class)))
print("Misclassified: "+str(len(mis_class)))
```

- Compares `y_test` (true labels) with `pred_digits` (model predictions).
- Properly classified indices are stored in `prop_class`, and misclassified indices are stored in `mis_class`.
- Prints counts of properly classified and misclassified images.

```
count=0
fig,ax=plt.subplots(4,2)
fig.set_size_inches(15,15)
for i in range (4):
    for j in range (2):
        ax[i,j].imshow(x_test[prop_class[count]])
        ax[i,j].set_title("Predicted : "+str(pred_digits[prop_class[count]])+"\n"+"Actual : "+str(np.argmax([y_test[prop_class[count]]])))
        plt.tight_layout()
        count+=1
```

- Creates a grid of subplots to display properly classified images.
- Each subplot shows:

  - Predicted and actual labels of the image.

- Uses `imshow` to display images from `x_test`.

```python
data = []

def create_data():
    for category in Categories:
        path = os.path.join(data_path, category)
        class_num = Categories.index(category)
        print(path)
        for img in os.listdir(path):
            print(img)
            img_path = os.path.join(path, img)
            img_arr = cv2.imread(img_path)

            # Check if the image is valid and not empty before resizing
            if img_arr is not None and not img_arr.size == 0:
                new_img_arr = cv2.resize(img_arr, (img_size, img_size))
                data.append([new_img_arr, class_num])
            else:
                print(f"Skipping {img_path} as it's not a valid image.")

create_data()
```

Adds error handling to the `create_data` function:

- Ensures images are valid (not `None`) and non-empty before resizing and appending.
- Prints a warning for skipped invalid images.

```python
test_data_path = "/content/gdrive/MyDrive/Kaggle/test"  # Update this to t
img_size = 227  # Set your desired image size

owu_img_data = []
owu_img_np = []

for owu_test_img in os.listdir(test_data_path):
    img_path = os.path.join(test_data_path, owu_test_img)

    # Try to load the image
    test_img_arr = cv2.imread(img_path)

    # Check if the image is valid and not empty before processing
    if test_img_arr is not None and not test_img_arr.size == 0:
        test_new_img_arr = cv2.resize(test_img_arr, (img_size, img_size))
        test_img = process_images(test_new_img_arr)
        test_img = tf.image.per_image_standardization(test_img)
        test_img = np.array(test_img).reshape(-1, 224, 224, 3)
        owu_img_data.append(img_path)
        owu_img_np.append(test_img)
    else:
        print(f"Skipping {img_path} as it's not a valid image.")
```

- Loads images from a test directory.
- Preprocesses images:
  - Ensures validity.
  - Resizes using OpenCV (`cv2`).
  - Standardizes images and reshapes them into a 4D tensor (`-1, 224, 224, 3`).
- Appends valid images to `owu_img_np` (processed data) and paths to `owu_img_data`.
- Prints warnings for invalid/skipped images.

```python
owu_predictions=[]
for test_img in owu_img_np:
        owu_pred=model.predict(test_img)
        owu_pred_digits=np.argmax(owu_pred,axis=1)
        owu_predictions.append(float(owu_pred_digits))
```

- Iterates through `owu_img_np` (processed test images).
- Uses `model.predict()` to generate predictions for each image.
- Retrieves the class with the highest probability using `np.argmax()` and appends it to `owu_predictions`.

```python
owu_results = []
for pred in owu_predictions:
    if(pred == 0):
        owu_results.append("free")
    else:
        owu_results.append("full")
```

- Maps numeric predictions (`0` or `1`) in `owu_predictions` to their respective class labels:
  - `0` -> "free"
  - `1` -> "full"
- Appends the corresponding labels to `owu_results`.

```python
fig = plt.figure(figsize=(10, 10))
k = 0
num_subplots = min(len(owu_img_data), 16)

for test_img in owu_img_data[:num_subplots]:
    k = k + 1
    ax = fig.add_subplot(4, 4, k)
    img = mpimg.imread(test_img)
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(owu_results[owu_img_data.index(test_img)])

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

Displays test images with predicted labels:

- Limits the number of images displayed to 16 or the total number of images, whichever is smaller.
- Adds each test image to a grid of subplots (4x4).
- Titles each image with the predicted label.

```python
# File path to the dataset directory
Full= "/content/gdrive/MyDrive/data/data/Full"
Free="/content/gdrive/MyDrive/data/data/Free"
```

```python
from tqdm import tqdm
# Ignore the warnings
import warnings
warnings.filterwarnings("always")
warnings.filterwarnings("ignore")
```

- Specifies file paths for the "Full" and "Free" datasets stored in Google Drive.
- Uses `tqdm` for progress bars during data processing and suppresses warnings using Python's `warnings` module.

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

- A helper function `assign_label(img, short_type)` assigns a label (`short_type`) to an image.
- Likely used to standardize label assignment based on predefined categories.

```python
# Process images of a cricket shot type and prepare data for training
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))
```

Function: `add_shorts_to_train_data(short_type, DIR)`:

- Processes images from the specified directory (`DIR`).
- Assigns labels using `assign_label(img, short_type)` for cricket shot types.
- Reads images with OpenCV, resizes them to `150x150`, and appends to `X` (features) as NumPy arrays.
- Corresponding labels are stored in `Z`.

```
# Add pullshots to the training data
add_shorts_to_train_data("Full",Full)
len(X)
```

```
# Add sweep shots to the training data
add_shorts_to_train_data("Free",Free)
len(X)
```

Adds "Full" shots and "Free" shots to the training dataset:

- Calls `add_shorts_to_train_data()` for each class directory.
- Displays the length of `X` after adding data for verification.

```
# Label Encoding for the dataset
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
Y=le.fit_transform(Z)
```

```
# Categorical Data
from keras.utils import to_categorical
Y=to_categorical(Y,4)
```

- Encodes class labels (`Z`) into integers using `LabelEncoder` from Scikit-learn.
- Converts encoded labels into one-hot categorical format using Keras' `to_categorical()`.
- Prepares `Y` for model training (e.g., "Full" and "Free" as `[1, 0]` and `[0, 1]`).

```python
import math
import pandas as pd
import numpy as np
import cv2
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
import tensorflow as tf
from tensorflow.keras import optimizers
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Conv2D, GlobalAveragePooling2D
from tensorflow.keras.layers import Dropout, BatchNormalization, Activation
from tensorflow.keras.callbacks import Callback, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
```

- Imports libraries for:
  - Data manipulation (`pandas`, `NumPy`).
  - Image processing (`cv2`, `ImageDataGenerator`).
  - Machine learning and deep learning (`Scikit-learn`, `TensorFlow/Keras`).
  - Model evaluation (`accuracy_score`, `f1_score`, etc.).

```python
# 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
```

Function: `build_model(bottom_model, classes)`:

- Takes a base model (`bottom_model`) and adds:
  - A `GlobalAveragePooling2D` layer to reduce dimensions.
  - A dense (`Dense`) output layer with `softmax` activation for multiclass classification.
- Returns the modified model, ready for training.

```python
"""# 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),]

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

- Loads the **VGG19** pre-trained model with ImageNet weights (`include_top=False`) and an input shape of `(150, 150, 3)`.
- Adds a custom head to the base model using `build_model`.
- Implements callbacks:
    - **EarlyStopping**: Stops training when validation accuracy doesn't improve for 11 epochs.
    - **ReduceLROnPlateau**: Reduces the learning rate when validation accuracy plateaus.
- Uses `ImageDataGenerator` for data augmentation (rotation, shifts, flips, etc.).
- Compiles the model with the Adam optimizer and categorical cross-entropy loss.

- Trains the model using `.fit` with augmented training data and validation data.

```python
# Plotting the Model Accuracy & Model Loss vs Epochs (Hidden Input)
plt.figure(figsize=[20,8])
# summarize history for accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy', size=25, pad=20)
plt.ylabel('Accuracy', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
# summarize history for loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss', size=25, pad=20)
plt.ylabel('Loss', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

- Visualizes training and validation accuracy/loss over epochs using Matplotlib.
- Plots are divided into subplots:
  - First subplot: Training vs. validation accuracy.
  - Second subplot: Training vs. validation loss.

```python
pred = model.predict(x_test)
pred_digits = np.argmax(pred, axis = 1)
i = 0
prop_class = []
mis_class = []

for i in range(len(y_test)):
    if np.argmax(y_test[i]) == np.argmax(pred[i]):
        prop_class.append(i)
    else:
        mis_class.append(i)
count=0
fig,ax=plt.subplots(4,2)
fig.set_size_inches(15,15)
for i in range (4):
    for j in range (2):
        ax[i,j].imshow(x_test[prop_class[count]])
        ax[i,j].set_title("Predicted : "+str(pred_digits[prop_class[count]])+"\n"+"Actual : "+str(np.argmax([y_test[prop_class[count]]])))
        plt.tight_layout()
        count+=1
```

- Predicts classes on `x_test` using `model.predict`.
- Compares predictions with actual labels (`y_test`) and identifies correctly and incorrectly classified samples.
- Displays correctly classified test samples with predicted and actual labels in a grid layout.

```python
"""# 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),]

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)
```

- Loads the **ResNet50** pre-trained model similarly to VGG19 but uses the ResNet50 architecture.
- Configures the same callbacks (`EarlyStopping` and `ReduceLROnPlateau`) and data augmentation.
- Compiles and trains the model with the same optimizer, loss function, and training setup.

```python
# Plotting the Model Accuracy & Model Loss vs Epochs (Hidden Input)
plt.figure(figsize=[20,8])
# summarize history for accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy', size=25, pad=20)
plt.ylabel('Accuracy', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
# summarize history for loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss', size=25, pad=20)
plt.ylabel('Loss', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

- Plots training/validation accuracy and loss for ResNet50 using the same visualization method.

```python
pred = model.predict(x_test)
pred_digits = np.argmax(pred, axis = 1)
i = 0
prop_class = []
mis_class = []

for i in range(len(y_test)):
    if np.argmax(y_test[i]) == np.argmax(pred[i]):
        prop_class.append(i)
    else:
        mis_class.append(i)
count=0
fig,ax=plt.subplots(4,2)
fig.set_size_inches(15,15)
for i in range (4):
    for j in range (2):
        ax[i,j].imshow(x_test[prop_class[count]])
        ax[i,j].set_title("Predicted : "+str(pred_digits[prop_class[count]])+"\n"+"Actual : "+str(np.argmax([y_test[prop_class[count]]])))
        plt.tight_layout()
        count+=1
```

- Predicts classes for the test set using `model.predict`.
- Separates indices of properly classified samples (`prop_class`) and misclassified ones (`mis_class`).
- Visualizes a subset of correctly classified samples using `plt.subplots`:
    - Displays the predicted label and actual label on each subplot.

```
"""# 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),]

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)
```

- Uses **MobileNet** pre-trained on ImageNet as the base model (`include_top=False`) with a custom input shape `(150, 150, 3)`.
- Adds a custom head using `build_model`.
- Configures callbacks:
    - **EarlyStopping**: Stops training if validation accuracy doesn't improve for 11 epochs.
    - **ReduceLROnPlateau**: Adjusts learning rate upon validation accuracy stagnation.
- Employs `ImageDataGenerator` for augmentation (rotation, shift, zoom, etc.).
- Compiles and trains the model using Adam optimizer and categorical cross-entropy loss.

```
# Plotting the Model Accuracy & Model Loss vs Epochs (Hidden Input)
plt.figure(figsize=[20,8])
# summarize history for accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy', size=25, pad=20)
plt.ylabel('Accuracy', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
# summarize history for loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss', size=25, pad=20)
plt.ylabel('Loss', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Plots training and validation accuracy and loss over epochs using Matplotlib:

- Subplot 1: Accuracy trends.
- Subplot 2: Loss trends.

```
pred = model.predict(x_test)
pred_digits = np.argmax(pred, axis = 1)
i = 0
prop_class = []
mis_class = []

for i in range(len(y_test)):
    if np.argmax(y_test[i]) == np.argmax(pred[i]):
        prop_class.append(i)
    else:
        mis_class.append(i)
count=0
fig,ax=plt.subplots(4,2)
fig.set_size_inches(15,15)
for i in range (4):
    for j in range (2):
        ax[i,j].imshow(x_test[prop_class[count]])
        ax[i,j].set_title("Predicted : "+str(pred_digits[prop_class[count]])+"\n"+"Actual : "+str(np.argmax([y_test[prop_class[count]]])))
        plt.tight_layout()
        count+=1
```

This snippet predicts test set classes, identifies correctly classified samples (`prop_class`), and visualizes 8 of them in a 4x2 grid. Each subplot shows the test image with its predicted and actual labels.

```
!pip install efficientnet
from efficientnet.tfkeras import EfficientNetB0  # You can choose a different variant (B0, B1, ..., B7) based on your requirements
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

- Installs and imports **EfficientNet** (a highly efficient deep learning model).
- Sets up `EfficientNetB0` from the EfficientNet library for use in the project.
- Imports necessary layers and utilities from TensorFlow/Keras:

- `GlobalAveragePooling2D` and `Dense` for custom layers.
- `Model` for model assembly.
- `Adam` optimizer for compilation.

```python
#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),]

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)
```

- Utilizes **EfficientNetB0** pre-trained on ImageNet as the base model.
- Adds a custom classification head for fine-tuning.
- Implements data augmentation with `ImageDataGenerator` (rotation, shifts, flips, etc.).
- Uses **EarlyStopping** and **ReduceLROnPlateau** callbacks for training optimization.
- Compiles the model with Adam optimizer and categorical cross-entropy loss, trains it, and

evaluates predictions.

```
# Plotting the Model Accuracy & Model Loss vs Epochs (Hidden Input)
plt.figure(figsize=[20,8])
# summarize history for accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy', size=25, pad=20)
plt.ylabel('Accuracy', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
# summarize history for loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss', size=25, pad=20)
plt.ylabel('Loss', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

- Plots training and validation accuracy/loss trends over epochs using Matplotlib.
- Subplots show:
    - Accuracy trends in the first subplot.
    - Loss trends in the second subplot.

```
pred = model.predict(x_test)
pred_digits = np.argmax(pred, axis = 1)
i = 0
prop_class = []
mis_class = []

for i in range(len(y_test)):
    if np.argmax(y_test[i]) == np.argmax(pred[i]):
        prop_class.append(i)
    else:
        mis_class.append(i)
count=0
fig,ax=plt.subplots(4,2)
fig.set_size_inches(15,15)
for i in range (4):
    for j in range (2):
        ax[i,j].imshow(x_test[prop_class[count]])
        ax[i,j].set_title("Predicted : "+str(pred_digits[prop_class[count]])+"\n"+"Actual : "+str(np.argmax([y_test[prop_class[count]]])))
        plt.tight_layout()
        count+=1
```

- Predicts classes for the test set and identifies correctly/misclassified samples.
- Displays 8 correctly classified test images in a 4x2 grid with their predicted and actual labels.

```python
"""# DenseNet"""
import tensorflow as tf
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),]

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)
```

- Leverages **DenseNet121** pre-trained on ImageNet with a custom head.
- Trains the model with the same augmentation and callback setup as EfficientNet.
- Compiles the model with Adam optimizer and categorical cross-entropy loss, trains it, and evaluates predictions.

```python
# Plotting the Model Accuracy & Model Loss vs Epochs (Hidden Input)
plt.figure(figsize=[20,8])
# summarize history for accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy', size=25, pad=20)
plt.ylabel('Accuracy', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
# summarize history for loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss', size=25, pad=20)
plt.ylabel('Loss', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

- Similar to EfficientNet, it plots accuracy and loss trends over epochs for DenseNet121 training.

```python
from tensorflow.keras.models import Model, load_model
# Save the model
den_model.save("dense_model_final.h5")

# Load the model
loaded_model = load_model("dense_model_final.h5")
```

- Saves the trained model `den_model` to a file named `dense_model_final.h5`.
- Loads the saved model using `load_model` to reuse it without retraining.

```python
pred = loaded_model.predict(x_test)
pred_digits = np.argmax(pred, axis = 1)
i = 0
prop_class = []
mis_class = []

for i in range(len(y_test)):
    if np.argmax(y_test[i]) == np.argmax(pred[i]):
        prop_class.append(i)
    else:
        mis_class.append(i)
count=0
fig,ax=plt.subplots(4,2)
fig.set_size_inches(15,15)
for i in range (4):
    for j in range (2):
        ax[i,j].imshow(x_test[prop_class[count]])
        ax[i,j].set_title("Predicted : "+str(pred_digits[prop_class[count]])+"\n"+"Actual : "+str(np.argmax([y_test[prop_class[count]]])))
        plt.tight_layout()
        count+=1
```

- Uses the loaded model to predict on `x_test`.
- Separates correctly (`prop_class`) and incorrectly (`mis_class`) classified samples.
- Displays 8 correctly classified samples in a 4x2 grid with their predicted and actual labels.

```python
# 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')


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

    # 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)
```

- Calculates accuracy, precision, recall, and F1 score for multiple models (`VGGNet`, `ResNet`, `MobileNet`, `EfficientNet`, and `DenseNet`) using Scikit-learn metrics.
- Organizes results into a pandas DataFrame for easy comparison and sorts by accuracy.

```python
# Create a confusion matrix
import itertools
from sklearn.metrics import confusion_matrix

class_names = ['Free','Full']
def make_confusion_matrix(y_true, y_pred, classes= None,figsize=(10,10),text_size=10):

  # Create the confusion matrix
  cm=confusion_matrix(y_true,y_pred)
  cm_norm = cm.astype('float')/ cm.sum(axis=1)[:,np.newaxis] #normalize the confuion matrix
  n_classes = cm.shape[0]

  # Let's prettify it
  fig, ax= plt.subplots(figsize=figsize)

  # Create a matrix plot
  cax = ax.matshow(cm,cmap=plt.cm.Blues)
  fig.colorbar(cax)

  # Set labels to be classes
  if classes:
    labels= classes
  else:
    labels = np.arange(cm.shape[0])

  # Label the axes
  ax.set(title="Confusion Matrix",
        xlabel='Predicted Label',
        ylabel='True label',
        xticks=np.arange(n_classes),
        yticks=np.arange(n_classes),
        xticklabels=labels,
        yticklabels=labels)

  # Set x-axis labels to the bottom
  ax.xaxis.set_label_position('bottom')
  ax.xaxis.tick_bottom()

  # Adjust label size
  ax.yaxis.label.set_size(text_size)
  ax.xaxis.label.set_size(text_size)
  ax.title.set_size(text_size)

  # Set threshold for different colors
  threshold = (cm.max()+cm.min()) / 2

  # Plot the text on each cell
  for i,j in itertools.product(range(cm.shape[0]),range(cm.shape[1])):
    plt.text(j,i,f"{cm[i,j]} ({cm_norm[i,j]*100:.1f}%)",
            horizontalalignment='center',
            color='white' if cm[i,j]>threshold else 'black',
            size=text_size)
```

Defines a function `make_confusion_matrix` to:

- Create and normalize a confusion matrix.
- Visualize the matrix using Matplotlib.
- Annotate matrix cells with percentages and counts.

```python
confusion_matrix(y_true,yhat_valid_vgg)
make_confusion_matrix(y_true,yhat_valid_vgg,classes=class_names,figsize=(15,15),text_size=10)
```

```python
confusion_matrix(y_true, yhat_valid_res)
make_confusion_matrix(y_true,yhat_valid_res,classes=class_names,figsize=(15,15),text_size=10)
```

```python
confusion_mob = confusion_matrix(y_true, yhat_valid_mob)
make_confusion_matrix(y_true,yhat_valid_mob,classes=class_names,figsize=(15,15),text_size=10)
```

```python
confusion_eff = confusion_matrix(y_true, yhat_valid_eff)
make_confusion_matrix(y_true,yhat_valid_eff,classes=class_names,figsize=(15,15),text_size=10)
```

```python
confusion_den = confusion_matrix(y_true, yhat_valid_den)
make_confusion_matrix(y_true,yhat_valid_den,classes=class_names,figsize=(15,15),text_size=10)
```

Creates and visualizes confusion matrices for each model (`VGGNet`, `ResNet`, `MobileNet`, `EfficientNet`, and `DenseNet`) using the defined function.

Number_plate_detection

```python
import cv2
import numpy as np
from ultralytics import YOLO
import easyocr

def detect_and_read_plates(input_path, output_path=None, confidence_threshold=0.5):

    model = YOLO('yolov11n.pt')
    reader = easyocr.Reader(['en'])

    if input_path.lower().endswith(('.jpg', '.jpeg', '.png', '.bmp')):

        image = cv2.imread(input_path)
        process_frame(image, model, reader, confidence_threshold)
        if output_path:
            cv2.imwrite(output_path, image)
        cv2.imshow("License Plate Detection", image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    elif input_path.lower().endswith(('.mp4', '.avi', '.mov', '.mkv')):
        cap = cv2.VideoCapture(input_path)

        # Get original video properties
        frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = int(cap.get(cv2.CAP_PROP_FPS))
        if output_path:
            fourcc = cv2.VideoWriter_fourcc(*'mp4v')
            out = cv2.VideoWriter(output_path, fourcc, fps, (frame_width, frame_height))

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

            # Process each frame
            process_frame(frame, model, reader, confidence_threshold)

            if output_path:
                out.write(frame)

            cv2.imshow("License Plate Detection", frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
```

```python
        cap.release()
        if output_path:
            out.release()
        cv2.destroyAllWindows()

    else:
        raise ValueError("Unsupported file format. Please provide an image or video file.")

def process_frame(frame, model, reader, confidence_threshold):
    # Detect license plates using YOLO
    results = model(frame)

    for result in results:
        boxes = result.boxes.xyxy.cpu().numpy().astype(int)
        confidences = result.boxes.conf.cpu().numpy()

        for box, confidence in zip(boxes, confidences):
            if confidence > confidence_threshold:
                x1, y1, x2, y2 = box
                plate_img = frame[y1:y2, x1:x2]

                # Perform OCR on the plate image
                try:
                    ocr_result = reader.readtext(plate_img)
                    if ocr_result:
                        plate_text = ocr_result[0][1]
                    else:
                        plate_text = ""
                except Exception as e:
                    print(f"OCR error: {e}")
                    plate_text = ""

                # Draw bounding box and OCR result on the frame
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
                cv2.rectangle(frame, (x1, y1 - 30), (x2, y1), (0, 255, 0), -1)
                cv2.putText(frame, f"{plate_text}", (x1 + 5, y1 - 10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2)
                cv2.putText(frame, f"Conf: {confidence:.2f}", (x1 + 5, y2 + 15),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

if __name__ == "__main__":
    # Input and output paths
    input_path = "input.jpg"
    output_path = "output_detected.jpg"

    detect_and_read_plates(input_path, output_path, confidence_threshold=0.5)
```

1. Functionality:

The code detects and reads license plates from images or videos using a YOLO model for object detection and EasyOCR for optical character recognition (OCR).

2. Main Components:

- **`detect_and_read_plates(input_path, output_path, confidence_threshold)`**:
    - Loads a YOLO model (`yolov11n.pt`) and an OCR reader for English (`easyocr.Reader(['en'])`).
    - Handles input files (images or videos):
        - **Images**:
            - Reads the image with `cv2.imread`.
            - Processes the frame using `process_frame` to detect plates and perform OCR.

        - Saves the processed image to `output_path` (if provided).
    - **Videos**:
        - Uses `cv2.VideoCapture` to read frames from the video.
        - Processes each frame to detect plates and perform OCR.
        - Saves processed frames to an output video file using `cv2.VideoWriter`.
- **`process_frame(frame, model, reader, confidence_threshold)`**:
    - Detects license plates in the frame using YOLO.
    - Extracts bounding boxes and confidence scores from YOLO's results.
    - For each detection with confidence above the threshold:
        - Crops the detected license plate area.
        - Uses EasyOCR to extract text from the cropped region.
        - Draws bounding boxes and annotates the frame with:
            - Extracted license plate text.
            - Confidence score of the detection.
- **Bounding Box and Annotation:**
    - Draws rectangles around detected license plates (`cv2.rectangle`).
    - Adds license plate text and confidence score as labels using `cv2.putText`.

Pklot[2]

```
!pip install ultralytics
```

- `!pip install ultralytics`: Installs the YOLOv8 library for object detection.

- `!pip install roboflow`: Installs the Roboflow Python package for dataset management.

```
!pip install roboflow
```

- Imports `Roboflow`.

```
from roboflow import Roboflow
rf = Roboflow(api_key="aSgxZMjrNAbRblOdLfH2")
project = rf.workspace("brad-dwyer").project("pklot-1tros")
version = project.version(4)
dataset = version.download("yolov8")
```

- Uses an API key to authenticate and download a specific project dataset (`"pklot-1tros"`) in the YOLOv8 format.

```python
import shutil

shutil.move('/content/PKLot-4/test',
            '/content/PKLot-4/PKLot-4/test')

shutil.move('/content/PKLot-4/valid',
            '/content/PKLot-4/PKLot-4/valid')

shutil.move('/content/PKLot-4/train',
            '/content/PKLot-4/PKLot-4/train')
```

Uses `shutil.move` to rearrange dataset folders (`test`, `valid`, and `train`) into their correct locations.

```
!yolo task=detect mode=train model=yolov5lu.pt data=/content/PKLot-4/data.yaml epochs=20 imgsz=640
```

The `yolo` command specifies:

- Task: `detect` (object detection task).
- Mode: `train` (train the model).
- Model: `yolov5lu.pt` (pretrained YOLOv5 model).
- Data: Path to dataset YAML file.
- Epochs: `20` (number of training iterations).
- Image size: `640` (resolution for training images).

```
!yolo task=detect mode=train model=yolov8l.pt data=/content/PKLot-4/data.yaml epochs=10 imgsz=640
```

- This command trains a YOLOv8 large model (`yolov8l.pt`) for object detection using the dataset specified in `/content/PKLot-4/data.yaml` for 10 epochs with a 640x640 image size.

```
!yolo task=detect mode=train model=yolov3.pt data=/content/PKLot-4/data.yaml epochs=10 imgsz=640
```

- This command trains a YOLOv3 model (`yolov3.pt`) for object detection using the dataset defined in `/content/PKLot-4/data.yaml`, with 10 epochs and an image size of 640x640.

```
import streamlit as st
from PIL import Image
import numpy as np
from ultralytics import YOLO
from collections import Counter
import pandas as pd
import easyocr
import cv2
import pytesseract

# Set the path to the Tesseract executable
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

# Initialize EasyOCR reader (this will download the model files on first run)
def load_ocr():
    return easyocr.Reader(['en'])

# Function to perform OCR on a number plate image
def read_plate(plate_img, reader):
    preprocessed_img = preprocess_image(plate_img)
    results = reader.readtext(preprocessed_img)

    # Extract and combine text
    if results:
        text = ' '.join([result[1] for result in results])
        # Clean the text (remove spaces and unwanted characters)
        text = ''.join(e for e in text if e.isalnum())
        return text
    return ''

# Image preprocessing function
def preprocess_image(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, binary_image = cv2.threshold(gray_image, 128, 255, cv2.THRESH_BINARY)
    return binary_image

# Function to perform YOLO inference and OCR
def detect_objects(image, model, reader, confidence=0.5):
    img_array = np.array(image)
    results = model(img_array)  # Run inference
    annotated_image = results[0].plot()  # Annotate image

    # Extract bounding boxes and class IDs
    detections = results[0].boxes.data.cpu().numpy()  # Get bounding boxes
    filtered_detections = [
        det for det in detections if det[4] > confidence  # Filter by confidence
    ]
```

```python
    # Extract class IDs
    class_ids = [int(det[5]) for det in filtered_detections]  # Index 5 contains class ID

    # Process number plates if any detected
    plate_texts = []
    if filtered_detections:
        for det in filtered_detections:
            x1, y1, x2, y2 = map(int, det[:4])  # Get coordinates
            # Extract plate (variable) img_array: Any
            plate_region = img_array[y1:y2, x1:x2]
            if plate_region.size > 0:  # Check if region is valid
                plate_text = read_plate(plate_region, reader)
                if plate_text:  # Only add if text was detected
                    plate_texts.append(plate_text)

    return Image.fromarray(annotated_image), filtered_detections, class_ids, plate_texts

# Streamlit UI
st.title("Parking Lot System")
st.write("Choose a detection task and upload an image.")

# Initialize EasyOCR
reader = load_ocr()

# Task Selection
task = st.radio("Select Task", ["Parking Lot Detection", "Number Plate Detection"])

if task == "Parking Lot Detection":
    st.header("Upload Image of Parking Lot")

    # File upload
    uploaded_file = st.file_uploader("Choose an image of a parking lot...", type=["jpg", "jpeg", "png"])

    if uploaded_file is not None:
        # Load the image
        image = Image.open(uploaded_file)
        st.image(image, caption="Uploaded Image", use_column_width=True)

        st.write("Detecting parking spaces...")

        # Load the parking lot detection model
        model_path = "parking_yolov9_model.pt"
        model = YOLO(model_path)

        # Perform detection
        confidence_threshold = st.slider("Confidence Threshold", 0.1, 1.0, 0.5)
        output_image, detections, class_ids, _ = detect_objects(image, model, reader, confidence=confidence_threshold)

        # Display results
        st.image(output_image, caption="Detection Results", use_column_width=True)
        st.write("Detections:")
        st.write(detections)

        # Count class occurrences
        class_counts = Counter(class_ids)
        df = pd.DataFrame(class_counts.items(), columns=["Class ID", "Count"])
        st.table(df)
```

```python
elif task == "Number Plate Detection":
    st.header("Upload Image of Car")

    # File upload
    uploaded_file = st.file_uploader("Choose an image of a car...", type=["jpg", "jpeg", "png"])

    if uploaded_file is not None:
        # Load the image
        image = Image.open(uploaded_file)
        st.image(image, caption="Uploaded Image", use_column_width=True)

        st.write("Detecting number plates...")

        # Load the number plate detection model
        model_path = "anpr_model.pt"
        model = YOLO(model_path)

        # Perform detection
        confidence_threshold = st.slider("Confidence Threshold", 0.1, 1.0, 0.5)
        output_image, detections, class_ids, plate_texts = detect_objects(image, model, reader, confidence=confidence_threshold)

        # Display results
        st.image(output_image, caption="Detection Results", use_column_width=True)

        # Display detected number plates
        if plate_texts:
            st.subheader("Detected Number Plates:")
            for idx, plate in enumerate(plate_texts, 1):
                st.write(f"Plate {idx}: {plate}")
        else:
            st.write("No readable number plates detected")

        st.write("Detections:")
        st.write(detections)

        # Count class occurrences
        class_counts = Counter(class_ids)
        df = pd.DataFrame(class_counts.items(), columns=["Class ID", "Count"])
        st.table(df)
```

Setting Up Dependencies and Helper Functions

1. **Dependencies:** Includes modules such as `streamlit`, `PIL`, `YOLO`, `easyocr`, and `pytesseract`.
2. **OCR Initialization:** Initializes EasyOCR and sets up Tesseract for OCR tasks.
3. **Helper Functions:**
    - `read_plate`: Preprocesses images and performs OCR on detected license plates.
    - `preprocess_image`: Converts images to grayscale and applies binary thresholding.
    - `detect_objects`: Uses YOLO for object detection and extracts bounding boxes, IDs, and plate texts.

Snippet 2: Parking Lot Detection

1. **Task Selection:** Implements a `st.radio` widget for task selection.
2. **File Upload:** Uses `st.file_uploader` to upload parking lot images.
3. **YOLO Detection:** Loads a parking lot detection model (`parking_yolov9_model.pt`) and performs inference.
4. **Confidence Slider:** Allows the user to set a confidence threshold for detections.
5. **Result Display:** Outputs the detected objects, visualizations, and detection counts in a table.

Snippet 3: Number Plate Detection

1. **File Upload:** Accepts images of cars for number plate detection.
2. **YOLO Detection:** Uses a number plate detection model (`anpr_model.pt`) for inference.

3. **Plate Extraction:** Extracts number plates from bounding boxes and performs OCR to display readable text.
4. **Result Display:** Outputs the detected number plates and visualizations, with a fallback message if no plates are detected.

Streamlit application Config file for Number Plate Detection and Parking Lot System

```python
import streamlit as st
from PIL import Image
import numpy as np
from ultralytics import YOLO
from collections import Counter
import pandas as pd
import easyocr
import cv2
import pytesseract

# Set the path to the Tesseract executable
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

# Initialize EasyOCR reader (this will download the model files on first run)
def load_ocr():
    return easyocr.Reader(['en'])

# Function to perform OCR on a number plate image
def read_plate(plate_img, reader):
    preprocessed_img = preprocess_image(plate_img)
    results = reader.readtext(preprocessed_img)

    # Extract and combine text
    if results:
        text = ' '.join([result[1] for result in results])
        # Clean the text (remove spaces and unwanted characters)
        text = ''.join(e for e in text if e.isalnum())
        return text
    return ''

# Image preprocessing function
def preprocess_image(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, binary_image = cv2.threshold(gray_image, 128, 255, cv2.THRESH_BINARY)
    return binary_image

# Function to perform YOLO inference and OCR
def detect_objects(image, model, reader, confidence=0.5):
    img_array = np.array(image)
    results = model(img_array)  # Run inference
    annotated_image = results[0].plot()  # Annotate image

    # Extract bounding boxes and class IDs
    detections = results[0].boxes.data.cpu().numpy()  # Get bounding boxes
    filtered_detections = [
        det for det in detections if det[4] > confidence  # Filter by confidence
    ]

    # Extract class IDs
    class_ids = [int(det[5]) for det in filtered_detections]  # Index 5 contains class ID
```

This code sets up a framework for a **Parking Lot and Number Plate Detection System** using YOLO for object detection and EasyOCR/Tesseract for text recognition. Key functionalities include:

1. **Image Preprocessing**: Converts images to grayscale and applies binary thresholding to enhance OCR performance.

2. **YOLO Object Detection**: Detects objects in the image, filters them by confidence, and extracts bounding boxes and class IDs.
3. **OCR Setup**: Initializes EasyOCR and Tesseract for reading license plate text.
4. **License Plate Text Extraction**: Crops detected plates, applies OCR, cleans text, and returns alphanumeric content only.

This setup integrates computer vision and text recognition efficiently, enabling detection and extraction of relevant data from images.

```python
    # Process number plates if any detected
    plate_texts = []
    if filtered_detections:
        for det in filtered_detections:
            x1, y1, x2, y2 = map(int, det[:4])  # Get coordinates
            # Extract plate region from original image
            plate_region = img_array[y1:y2, x1:x2]
            if plate_region.size > 0:  # Check if region is valid
                plate_text = read_plate(plate_region, reader)
                if plate_text:  # Only add if text was detected
                    plate_texts.append(plate_text)

    return Image.fromarray(annotated_image), filtered_detections, class_ids, plate_texts

# Streamlit UI
st.title("Parking Lot System")
st.write("Choose a detection task and upload an image.")

# Initialize EasyOCR
reader = load_ocr()

# Task Selection
task = st.radio("Select Task", ["Parking Lot Detection", "Number Plate Detection"])

if task == "Parking Lot Detection":
    st.header("Upload Image of Parking Lot")

    # File upload
    uploaded_file = st.file_uploader("Choose an image of a parking lot...", type=["jpg", "jpeg", "png"])

    if uploaded_file is not None:
        # Load the image
        image = Image.open(uploaded_file)
        st.image(image, caption="Uploaded Image", use_column_width=True)

        st.write("Detecting parking spaces...")

        # Load the parking lot detection model
        model_path = "parking_yolov9_model.pt"
        model = YOLO(model_path)

        # Perform detection
        confidence_threshold = st.slider("Confidence Threshold", 0.1, 1.0, 0.5)
        output_image, detections, class_ids, _ = detect_objects(image, model, reader, confidence=confidence_threshold)

        # Display results
        st.image(output_image, caption="Detection Results", use_column_width=True)
        st.write("Detections:")
        st.write(detections)

        # Count class occurrences
        class_counts = Counter(class_ids)
        df = pd.DataFrame(class_counts.items(), columns=["Class ID", "Count"])
        st.table(df)

elif task == "Number Plate Detection":
    st.header("Upload Image of Car")

```

This code extends the functionality of the **Parking Lot and Number Plate Detection**

System by integrating the Streamlit user interface and logic for task execution. Here's a concise explanation:

Core Components

1. **Number Plate Processing**:
    - Iterates over detected objects.
    - Extracts bounding box coordinates and crops the plate region.
    - Runs OCR on the cropped region using `read_plate()` and appends detected text to a list.
2. **Streamlit UI**:
    - Displays a title and allows users to select between two tasks: "Parking Lot Detection" or "Number Plate Detection."
    - Includes a file uploader for users to upload images.
3. **Parking Lot Detection**:
    - Loads a YOLO model (`parking_yolov5_model.pt`) for detecting parking spaces.
    - Users can adjust a confidence threshold via a slider.
    - Displays:
        - Annotated image of parking lot with detections.
        - Class counts (e.g., cars, bikes) in a tabular format.
4. **Number Plate Detection**:
    - Loads a YOLO model (`anpr_model.pt`) to detect vehicles and plates.
    - Extracts license plate text using OCR for detected regions.
    - Outputs:
        - Annotated image with vehicle and plate bounding boxes.
        - Recognized number plates.

```python
# File upload
uploaded_file = st.file_uploader("Choose an image of a car...", type=["jpg", "jpeg", "png"])

if uploaded_file is not None:
    # Load the image
    image = Image.open(uploaded_file)
    st.image(image, caption="Uploaded Image", use_column_width=True)

    st.write("Detecting number plates...")

    # Load the number plate detection model
    model_path = "anpr_model.pt"
    model = YOLO(model_path)

    # Perform detection
    confidence_threshold = st.slider("Confidence Threshold", 0.1, 1.0, 0.5)
    output_image, detections, class_ids, plate_texts = detect_objects(image, model, reader, confidence=confidence_th

    # Display results
    st.image(output_image, caption="Detection Results", use_column_width=True)

    # Display detected number plates
    if plate_texts:
        st.subheader("Detected Number Plates:")
        for idx, plate in enumerate(plate_texts, 1):
            st.write(f"Plate {idx}: {plate}")
    else:
        st.write("No readable number plates detected")

    st.write("Detections:")
    st.write(detections)

    # Count class occurrences
    class_counts = Counter(class_ids)
    df = pd.DataFrame(class_counts.items(), columns=["Class ID", "Count"])
    st.table(df)
```

- **File Upload**:
  - Allows users to upload an image (e.g., a car with a number plate) in `.jpg`, `.jpeg`, or `.png` formats using `st.file_uploader()`.
- **Image Display**:
  - Displays the uploaded image on the Streamlit interface for user confirmation.
- **Model Loading**:
  - Loads a pre-trained YOLO model (`anpr_model.pt`) specifically for number plate detection.
- **Object Detection and OCR**:
  - Performs object detection using YOLO to identify number plate regions.
  - Passes detected regions to an OCR reader to extract plate numbers.
  - Confidence threshold for YOLO detection is adjustable using a slider.
- **Display Results**:
  - Annotated image with detection boxes is shown.

- Detected number plates are displayed, or a fallback message is shown if no plates are detected.

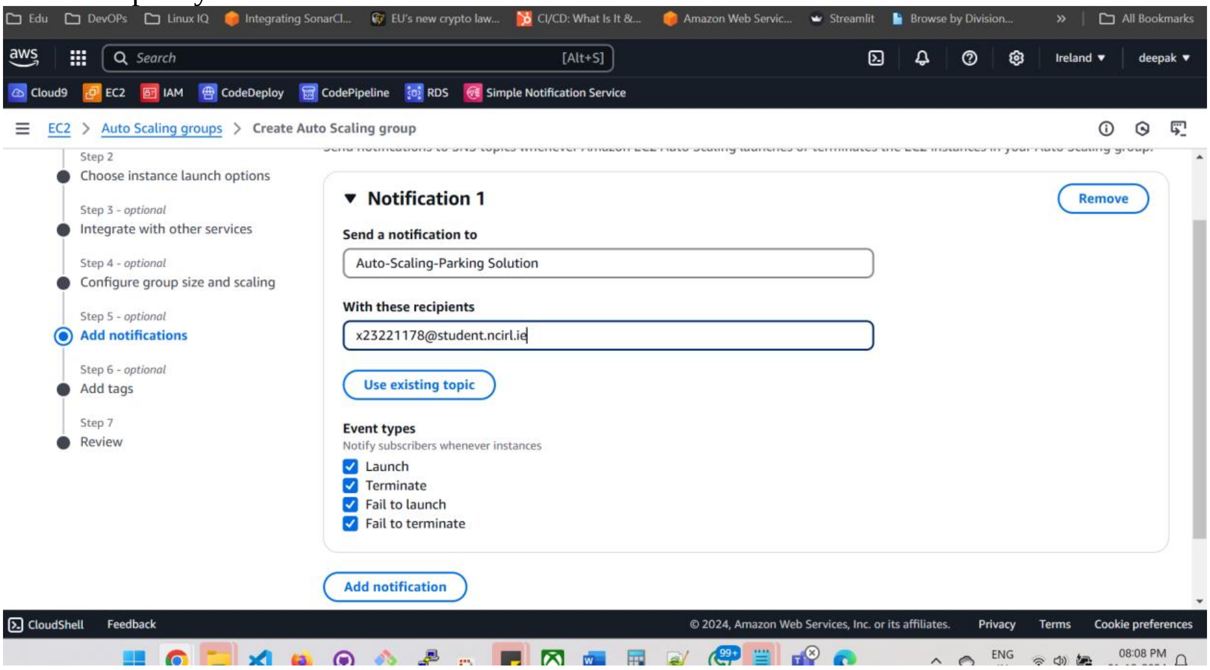
- **Detection Summary**:

  - Shows raw detection results for debugging or review.
  - Counts detected object classes (e.g., cars) and displays the counts in a tabular format.

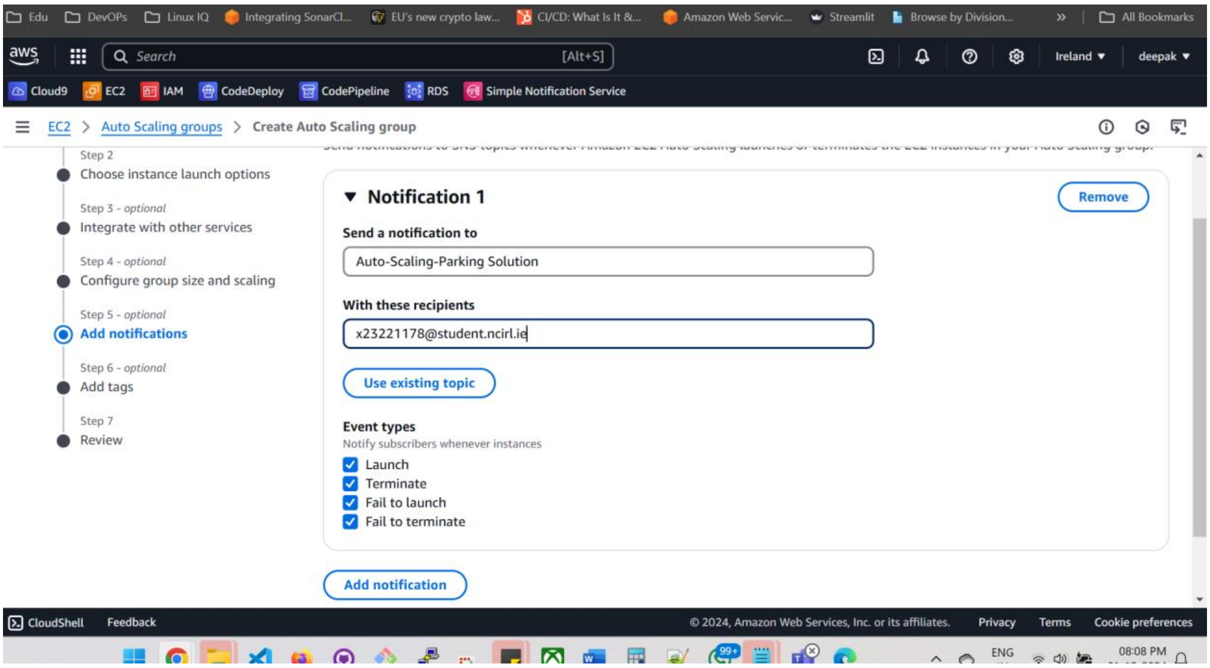
We have connected to Flask API as an end module for enabling seamless interaction

Configuring the Load Balancer and Auto Scaling Group

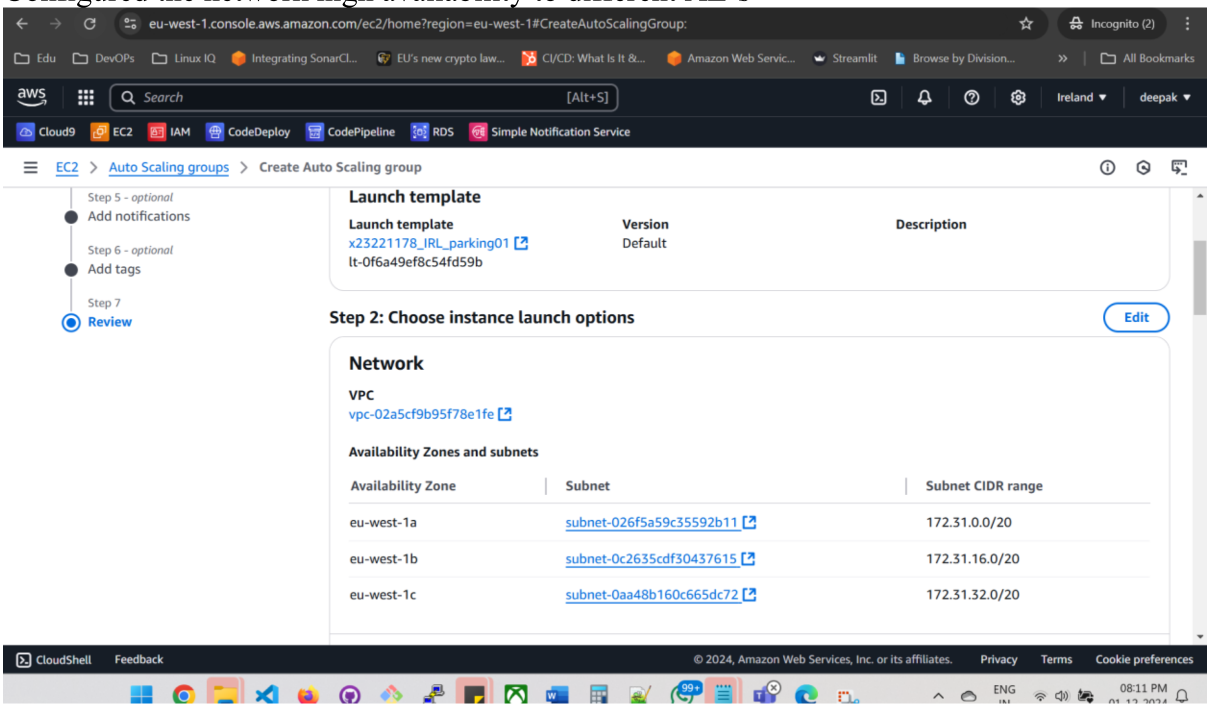
Here I am trying to make the instance High Available and Scalable, The Scale minimum desired capacity I have made it as 1 and Maximum as 2



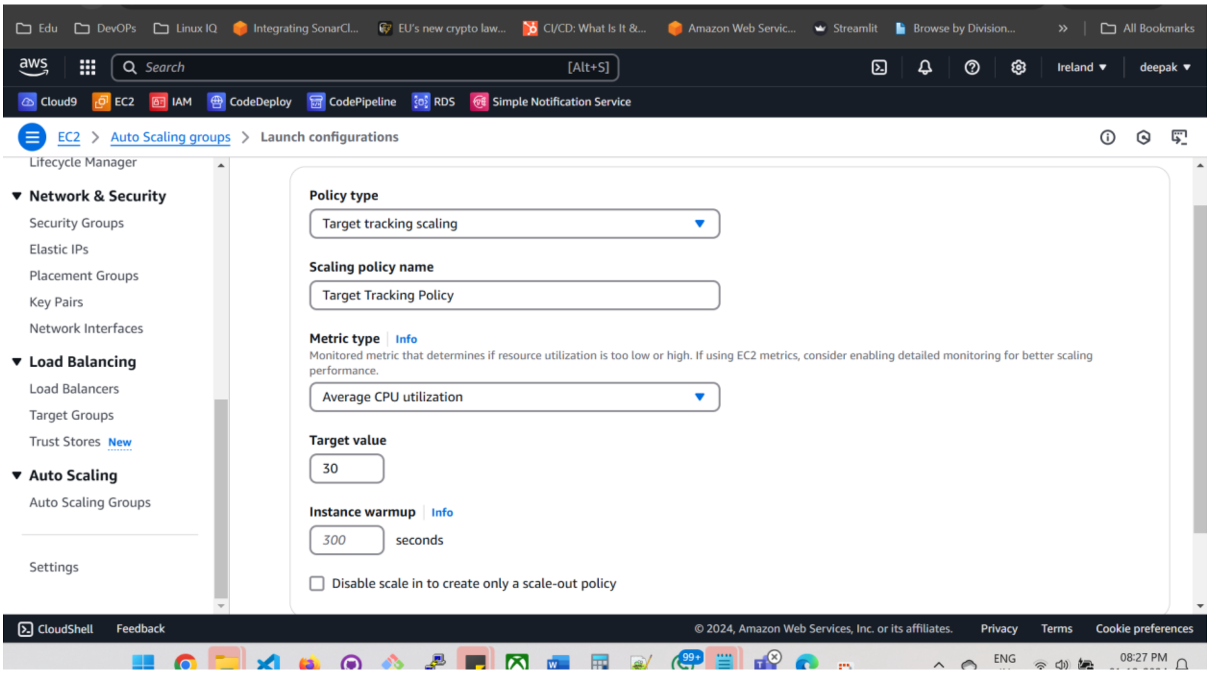
Configured the Cloudwatch by add my college email address so that I would receive an email for high resource utilization.



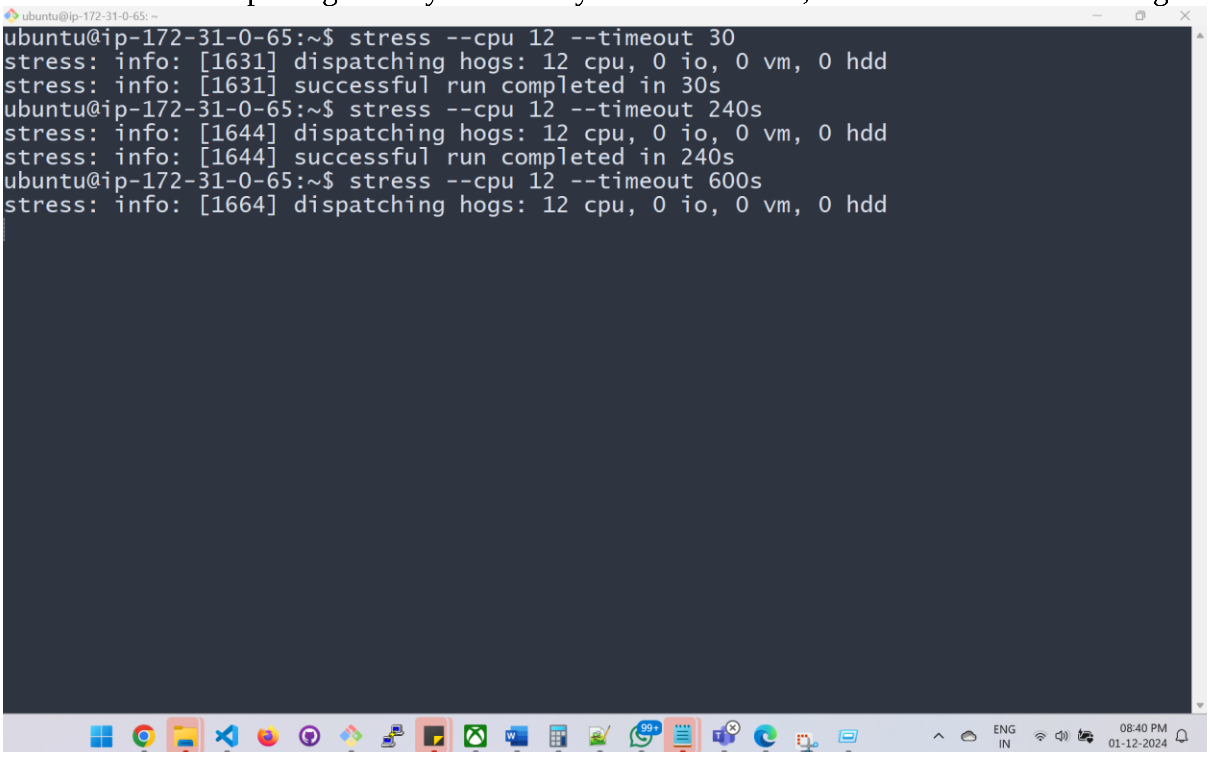
Configured the network high availability to different AZ’s

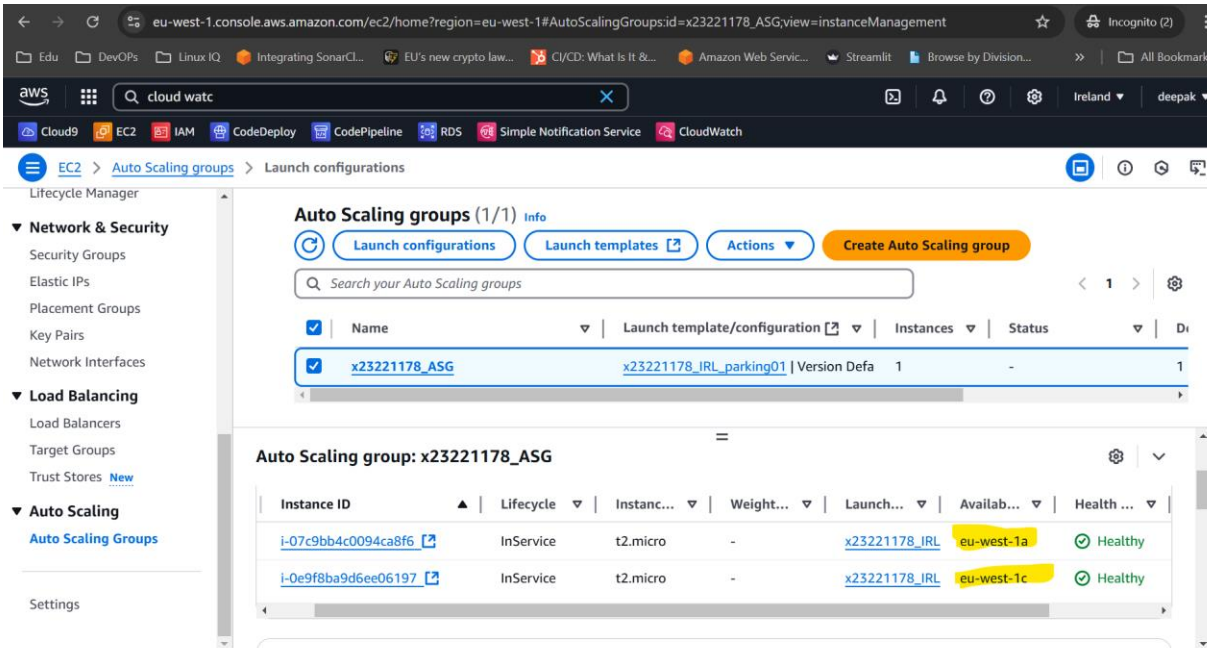


Target tracking has been configured if the instance CPU goes high the instance gets created automatically resulting in scaling



Installed the stress package on my ec2 and try to stress the ec2, which resulted Autoscaling.




eu-west-1.console.aws.amazon.com/ec2/home?region=eu-west-1#AutoScalingGroups:id=x23221178_ASG;view=instanceManagement
Incognito (2)
Edu DevOPs Linux IQ Integrating SonarCl... EU's new crypto law... CI/CD: What Is It &... Amazon Web Servic... Streamlit Browse by Division... All Bookmarks
cloud watc
Ireland
deepak
Cloud9 EC2 IAM CodeDeploy CodePipeline RDS Simple Notification Service CloudWatch
EC2 > Auto Scaling groups > Launch configurations
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
Load Balancing
Load Balancers
Target Groups
Trust Stores New
Auto Scaling
Auto Scaling Groups
Settings
Auto Scaling groups (1/1) Info
Launch configurations
Launch templates
Actions
Create Auto Scaling group
Search your Auto Scaling groups
Name
Launch template/configuration
Instances
Status
x23221178_ASG
x23221178_IRL_parking01 | Version Defa
1
-
1
Auto Scaling group: x23221178_ASG
Instance ID
Lifecycle
Instanc...
Weight...
Launch...
Availab...
Health ...
i-07c9bb4c0094ca8f6
InService
t2.micro
-
x23221178_IRL
eu-west-1a
Healthy
i-0e9f8ba9d6ee06197
InService
t2.micro
-
x23221178_IRL
eu-west-1c
Healthy