BAHRIA UNIVERSITY KARACHI CAMPUS

DEPARTMENT OF COMPUTER SCIENCE

(ARTIFICIAL INTELLIGENCE)

(AIC-401) Deep Learning

Assignment 01

Fall 2024

NAME	**ENROLMENT**
Muneeza Iftikhar	02-136212-012
Hafsa Hafeez Siddiqui	02-136212-026

SUBMITTED TO: REEMA QAISER KHAN

DL_Assignment_1

October 27, 2024

[13]:
```
import cv2
import numpy as np
import os
from tqdm import tqdm
from PIL import Image
from tensorflow.keras.preprocessing.image import ImageDataGenerator,␣
 ↪img_to_array, save_img
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, BatchNormalization, DepthwiseConv2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

# Paths and directory setup
Image.MAX_IMAGE_PIXELS = None
data_dir = '/content/drive/MyDrive/DL_Assignment1_Dataset'
preprocessed_dir = '/content/drive/MyDrive/Preprocessed_Img'
if not os.path.exists(preprocessed_dir):
    os.makedirs(preprocessed_dir)

# Custom Preprocessing Function
def custom_preprocess(img_array):
    hsv_img = cv2.cvtColor(img_array.astype(np.uint8), cv2.COLOR_RGB2HSV)
    mean_saturation = np.mean(hsv_img[:, :, 1])
    mean_value = np.mean(hsv_img[:, :, 2])

    is_black = (mean_value < 50) & (mean_saturation < 50)
    gray_img = cv2.cvtColor(img_array.astype(np.uint8), cv2.COLOR_RGB2GRAY)
    black_img = cv2.Canny(gray_img, threshold1=30, threshold2=100)
    black_img = cv2.cvtColor(black_img, cv2.COLOR_GRAY2RGB)

    hsv_img[:, :, 1] = hsv_img[:, :, 1] * (1.5 if mean_saturation > 100 else 1)
    hsv_img[:, :, 2] = hsv_img[:, :, 2] * (0.7 if (mean_value > 200) &␣
 ↪(mean_saturation < 50) else 1)

    final_img = black_img if is_black else cv2.cvtColor(hsv_img, cv2.
 ↪COLOR_HSV2RGB)
```

```
    return final_img

# Iterate through each class folder for preprocessing
for class_name in os.listdir(data_dir):
    class_dir = os.path.join(data_dir, class_name)
    new_class_dir = os.path.join(preprocessed_dir, class_name)
    os.makedirs(new_class_dir, exist_ok=True)

    for img_name in tqdm(os.listdir(class_dir), desc=f"Processing␣
 ↪{class_name}"):
        img_path = os.path.join(class_dir, img_name)

        if os.path.isfile(img_path):
            with Image.open(img_path) as img:
                img.thumbnail((224, 224))
                img_array = img_to_array(img)
                preprocessed_img = custom_preprocess(img_array)
                save_img(os.path.join(new_class_dir, img_name),␣
 ↪preprocessed_img)

# Custom ImageDataGenerator with updated paths and validation split
data_gen = ImageDataGenerator(
    rescale=1.0/255.0,
    validation_split=0.2
)

train_generator = data_gen.flow_from_directory(
    preprocessed_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

val_generator = data_gen.flow_from_directory(
    preprocessed_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)

# Model Definition
def create_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
        DepthwiseConv2D((3, 3), activation='relu'),
```

```
        MaxPooling2D(2, 2),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Conv2D(128, (3, 3), activation='relu'),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(3, activation='softmax')  # Output layer for 3 classes
    ])
    model.compile(optimizer=Adam(learning_rate=1e-4),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Instantiate and Train Model
model = create_model()
early_stopping = EarlyStopping(monitor='val_loss', patience=3,␣
 ↪restore_best_weights=True)

# Calculate steps per epoch
total_train_images = train_generator.samples
total_val_images = val_generator.samples
batch_size = 8

# Calculate steps per epoch based on the generator's total samples and batch␣
 ↪size
steps_per_epoch = train_generator.n // batch_size
validation_steps = val_generator.n // batch_size

# Train the model
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=val_generator,
    steps_per_epoch=steps_per_epoch,
    validation_steps=validation_steps,
    callbacks=[early_stopping]
)
```

```
Processing Transparent: 100%|      | 80/80 [00:06<00:00, 12.98it/s]
Processing Colorful: 100%|      | 80/80 [00:05<00:00, 15.71it/s]
Processing Black: 100%|      | 80/80 [00:16<00:00,  4.87it/s]

Found 192 images belonging to 3 classes.
Found 48 images belonging to 3 classes.


/usr/local/lib/python3.10/dist-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
```

```
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Epoch 1/10

/usr/local/lib/python3.10/dist-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()

 6/24             27s 2s/step -
accuracy: 0.3571 - loss: 1.0971

/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of data;
interrupting training. Make sure that your dataset or generator can generate at
least `steps_per_epoch * epochs` batches. You may need to use the `.repeat()`
function when building your dataset.
  self.gen.throw(typ, value, traceback)

24/24             16s 419ms/step -
accuracy: 0.3822 - loss: 1.0923 - val_accuracy: 0.4167 - val_loss: 1.0791
Epoch 2/10
24/24             22s 527ms/step -
accuracy: 0.5669 - loss: 1.0518 - val_accuracy: 0.5000 - val_loss: 1.0499
Epoch 3/10
24/24             18s 369ms/step -
accuracy: 0.5696 - loss: 0.9995 - val_accuracy: 0.5000 - val_loss: 1.0161
Epoch 4/10
24/24             12s 443ms/step -
accuracy: 0.6446 - loss: 0.9196 - val_accuracy: 0.5000 - val_loss: 0.9669
Epoch 5/10
24/24             12s 413ms/step -
accuracy: 0.6499 - loss: 0.8657 - val_accuracy: 0.5000 - val_loss: 0.9146
Epoch 6/10
24/24             10s 333ms/step -
accuracy: 0.7213 - loss: 0.7341 - val_accuracy: 0.5000 - val_loss: 0.8959
Epoch 7/10
24/24             12s 389ms/step -
accuracy: 0.7424 - loss: 0.6631 - val_accuracy: 0.5417 - val_loss: 0.9074
Epoch 8/10
24/24             20s 419ms/step -
accuracy: 0.7337 - loss: 0.6506 - val_accuracy: 0.4375 - val_loss: 0.9404
Epoch 9/10
24/24             10s 335ms/step -
accuracy: 0.6845 - loss: 0.6218 - val_accuracy: 0.6458 - val_loss: 0.7974
```

```
Epoch 10/10
24/24              13s 414ms/step -
accuracy: 0.7252 - loss: 0.5517 - val_accuracy: 0.5208 - val_loss: 0.9071
```

[33]:
```python
# Conveyor Belt Prediction Function
conveyor_mapping = {0: 'A - Black Object', 1: 'C - Colorful Object', 2: 'B - Transparent Object'}

def predict_conveyor_belt(image_path):
    # Load the image
    img = cv2.imread(image_path)
    if img is None:
        print(f"Error: Could not load image at {image_path}. Please check the path.")
        return None

    # Resize to (128, 128) to match the model's expected input shape
    img = cv2.resize(img, (128, 128))
    img = img / 255.0  # Normalize pixel values
    img = np.expand_dims(img, axis=0)  # Add batch dimension

    # Make prediction
    prediction = model.predict(img)
    class_idx = np.argmax(prediction, axis=1)[0]
    conveyor_belt = conveyor_mapping[class_idx]  # Map class to conveyor belt

    return conveyor_belt

# Predict Conveyor Belt for Test Image
object_image_path = '/content/test_t.jpg'
conveyor_belt = predict_conveyor_belt(object_image_path)
print(f'The object should be sent to Conveyor Belt: {conveyor_belt}')
```

```
1/1              0s 64ms/step
The object should be sent to Conveyor Belt: B - Transparent Object
```

Introduction

This report details the implementation of a deep learning pipeline for image classification using TensorFlow and Keras. The primary objectives include preprocessing a dataset of images, generating augmented data for training and validation, and building a convolutional neural network (CNN) model for classification tasks. The following sections outline the methodologies and processes utilized in this implementation.

Dataset Creation

We gathered a dataset comprising 80 images per category, representing three types of objects: transparent, black, and colorful. To ensure diversity and robustness in the model's training, each object was photographed under various lighting conditions and against different backgrounds, specifically white and brown. This setup was designed to enhance the model's ability to recognize and classify objects accurately across a range of real-world scenarios.

The diversity in lighting and background helps to simulate different environmental conditions that the model might encounter in practical applications, thus improving its generalization capabilities.

Libraries and Modules Import

The implementation begins with the import of essential libraries and modules. The libraries included are:

- **OpenCV**: For image processing tasks such as resizing and color space conversion.
- **NumPy**: For numerical operations and handling of image data.
- **OS**: For directory management and file operations.
- **TQDM**: For displaying progress bars during iterations.
- **TensorFlow/Keras**: For building and training the neural network model, as well as for data augmentation through the ImageDataGenerator class.

Directory Setup

Paths are established for the original dataset and for the storage of preprocessed images. The following actions are performed:

- **data_dir**: This variable points to the directory containing the original images.
- **preprocessed_dir**: This variable is designated for the output of preprocessed images.
- The code checks for the existence of the preprocessed directory and creates it if it does not exist.

Custom Preprocessing Function

A custom preprocessing function, custom_preprocess, is defined to enhance the quality of the images. The key features of this function are:

- **Color Space Conversion**: Each image is converted from RGB to HSV (Hue, Saturation, Value) to analyze color characteristics.
- **Mean Computation**: The mean saturation and brightness values of the image are calculated to determine specific image properties.

- **Black Image Detection**: If the mean brightness and saturation indicate a predominantly black image, edge detection is applied using Canny edge detection.
- **Image Adjustment**: For non-black images, the saturation and brightness are adjusted based on predefined thresholds to enhance image quality.
- The final output of the function is either a processed black image or an adjusted color image.

Image Preprocessing Pipeline

The preprocessing pipeline involves the following steps:

1. **Class Directory Iteration**: The code iterates through each class folder within the dataset.
2. **Directory Creation**: For each class, a corresponding directory is created in the preprocessed folder.
3. **Image Processing**: Each image is opened, resized to a maximum dimension of 224x224 pixels, and converted to a compatible array format. The custom preprocessing function is then applied, and the processed image is saved in the respective class directory.

Image Data Augmentation

To enhance model performance and generalization, an ImageDataGenerator instance is created with the following features:

- **Rescaling**: Image pixel values are rescaled to a range between 0 and 1.
- **Validation Split**: A split of 20% is allocated for validation purposes.

The training and validation datasets are generated from the preprocessed images. The images are resized to 128x128 pixels, and a batch size of 32 is specified. This results in two generators: train_generator for the training data and val_generator for the validation data.

Model Definition

A function, create_model, is defined to construct the convolutional neural network architecture, which includes:

- **Convolutional Layers**: Multiple Conv2D layers for feature extraction, including a DepthwiseConv2D layer.
- **Max Pooling Layers**: MaxPooling2D layers that reduce the dimensionality of the feature maps.
- **Dense Layer**: A fully connected Dense layer with dropout applied for regularization.
- **Output Layer**: A softmax output layer suitable for multi-class classification (three classes).

The model is compiled with the Adam optimizer, utilizing categorical cross-entropy as the loss function and accuracy as the evaluation metric.

Model Training Setup

The model training setup includes:

- **Early Stopping Callback**: This callback halts training if the validation loss does not improve over three consecutive epochs, restoring the best weights of the model.

- **Step Calculation**: The total number of training and validation images is divided by the batch size to calculate the steps per epoch and validation steps.
- **Model Training**: The model is trained using the train_generator, with validation provided by the val_generator. The training process is set for a maximum of 10 epochs or until early stopping conditions are met.

Conveyor Belt Prediction Function

To facilitate the classification of objects, a mapping dictionary, conveyor_mapping, is defined, associating numerical class indices with corresponding conveyor belt labels:

- **0**: 'A - Black Object'
- **1**: 'C - Colorful Object'
- **2**: 'B - Transparent Object'

This mapping allows for easy translation from model output to actionable instructions for conveyor belt sorting.

predict_conveyor_belt(image_path)

This function is responsible for loading an image from a specified path, preprocessing it, and predicting the corresponding conveyor belt. The key components of the function are as follows:

1. **Image Loading**:

- The function attempts to load the image using OpenCV's cv2.imread() method.
- If the image cannot be loaded (e.g., due to an incorrect path), an error message is printed, and the function returns None.

2. **Image Preprocessing**:

- The loaded image is resized to the dimensions (128, 128) to match the input shape expected by the model.
- Pixel values are normalized by dividing by 255.0 to scale them to the range [0, 1].
- A batch dimension is added to the image using np.expand_dims(), transforming the shape from (128, 128, 3) to (1, 128, 128, 3). This step is necessary as the model expects input in batches.

3. **Prediction**:

- The preprocessed image is fed into the model to make predictions using the model.predict() method.
- The output prediction is a probability distribution across the three classes. The np.argmax() function is employed to retrieve the index of the class with the highest probability, effectively determining the predicted class.

4. **Conveyor Belt Mapping**:

- The predicted class index is used to look up the corresponding conveyor belt label from the conveyor_mapping dictionary.

5. **Return Value**:

- The function returns the string indicating which conveyor belt the object should be sent to based on the model's prediction.

Example Usage

To illustrate the function's application, an example is provided where a test image of an object is predicted:

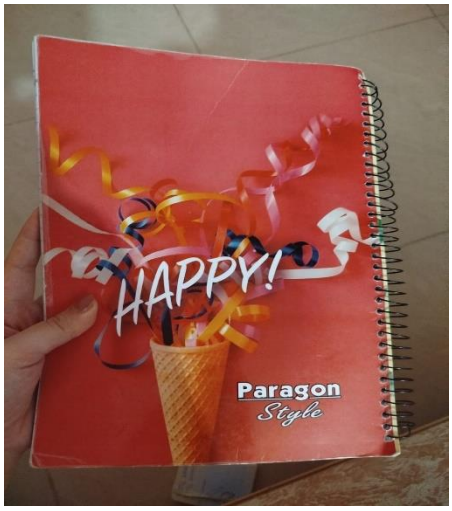
- **Object Image Path**: The image path is defined as /content/test_t.jpg.
- **Prediction Call**: The function predict_conveyor_belt() is called with the specified image path.
- **Output**: The result is printed, displaying the conveyor belt to which the object should be directed.

Results:

```
Processing Transparent: 100%|          | 80/80 [00:06<00:00, 12.98it/s]
Processing Colorful: 100%|          | 80/80 [00:05<00:00, 15.71it/s]
Processing Black: 100%|          | 80/80 [00:16<00:00,  4.87it/s]Found 192 images belonging to 3 classes.
Found 48 images belonging to 3 classes.

/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `inp
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `Py
  self._warn_if_super_not_called()
 6/24 ━━━━━━━━━━━━━━━━━━━━ 27s 2s/step - accuracy: 0.3571 - loss: 1.0971/usr/lib/python3.10/contextlib.py:153: UserWarnin
  self.gen.throw(typ, value, traceback)
24/24 ━━━━━━━━━━━━━━━━━━━━ 16s 419ms/step - accuracy: 0.3822 - loss: 1.0923 - val_accuracy: 0.4167 - val_loss: 1.0791
Epoch 2/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 22s 527ms/step - accuracy: 0.5669 - loss: 1.0518 - val_accuracy: 0.5000 - val_loss: 1.0499
Epoch 3/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 18s 369ms/step - accuracy: 0.5696 - loss: 0.9995 - val_accuracy: 0.5000 - val_loss: 1.0161
Epoch 4/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 12s 443ms/step - accuracy: 0.6446 - loss: 0.9196 - val_accuracy: 0.5000 - val_loss: 0.9669
Epoch 5/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 12s 413ms/step - accuracy: 0.6499 - loss: 0.8657 - val_accuracy: 0.5000 - val_loss: 0.9146
Epoch 6/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 10s 333ms/step - accuracy: 0.7213 - loss: 0.7341 - val_accuracy: 0.5000 - val_loss: 0.8959
Epoch 7/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 12s 389ms/step - accuracy: 0.7424 - loss: 0.6631 - val_accuracy: 0.5417 - val_loss: 0.9074
Epoch 8/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 20s 419ms/step - accuracy: 0.7337 - loss: 0.6506 - val_accuracy: 0.4375 - val_loss: 0.9404
Epoch 9/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 10s 335ms/step - accuracy: 0.6845 - loss: 0.6218 - val_accuracy: 0.6458 - val_loss: 0.7974
Epoch 10/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 13s 414ms/step - accuracy: 0.7252 - loss: 0.5517 - val_accuracy: 0.5208 - val_loss: 0.9071
```

Figure 1 - Model Accuracy achieved 72%




```
# Predict Conveyor Belt for Test Image
object_image_path = '/content/test2.jpg'
conveyor_belt = predict_conveyor_belt(object_image_path)
print(f'The object should be sent to Conveyor Belt: {conveyor_belt}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
The object should be sent to Conveyor Belt: C - Colorful Object
```

Figure 2 - Correct Colorful object detection


```
# Predict Conveyor Belt for Test Image
object_image_path = '/content/test3.jpg'
conveyor_belt = predict_conveyor_belt(object_image_path)
print(f'The object should be sent to Conveyor Belt: {conveyor_belt}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
The object should be sent to Conveyor Belt: A - Black Object
```

Figure 3 - Correct Black object detection

```
# Predict Conveyor Belt for Test Image
object_image_path = '/content/test4.jpg'
conveyor_belt = predict_conveyor_belt(object_image_path)
print(f'The object should be sent to Conveyor Belt: {conveyor_belt}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step
The object should be sent to Conveyor Belt: B - Transparent Object
```

Figure 4 - Correct Transparent object detection

```
# Predict Conveyor Belt for Test Image
object_image_path = '/content/test_t.jpg'
conveyor_belt = predict_conveyor_belt(object_image_path)
print(f'The object should be sent to Conveyor Belt: {conveyor_belt}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
The object should be sent to Conveyor Belt: B - Transparent Object
```

Figure 5 - Correct Transparent object detection

```
# Predict Conveyor Belt for Test Image
object_image_path = '/content/test_c.jpg'
conveyor_belt = predict_conveyor_belt(object_image_path)
print(f'The object should be sent to Conveyor Belt: {conveyor_belt}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
The object should be sent to Conveyor Belt: C - Colorful Object
```

Figure 6 - Correct Colorful object detection

```
# Predict Conveyor Belt for Test Image
object_image_path = '/content/test_b.jpg'
conveyor_belt = predict_conveyor_belt(object_image_path)
print(f'The object should be sent to Conveyor Belt: {conveyor_belt}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
The object should be sent to Conveyor Belt: C - Colorful Object
```

*Figure 7 - **Incorrect** Black object detection*