TechPart Vision: Final Project Report

Index:

1. Introduction

    1.1. Project overviews

    1.2. Objectives

2. Project Initialization and Planning Phase

    2.1. Define Problem Statement

    2.2. Project Proposal (Proposed Solution)

    2.3. Initial Project Planning

3. Data Collection and Preprocessing Phase

    3.1. Data Collection Plan and Raw Data Sources Identified

    3.2. Data Quality Report

    3.3. Data Preprocessing

4. Model Development Phase

    4.1. Model Selection Report

    4.2. Initial Model Training Code, Model Validation and Evaluation Report

5. Model Optimization and Tuning Phase

    5.1. Tuning Documentation

    5.2. Final Model Selection Justification

6. Results

    6.1. Output Screenshots

7. Advantages & Disadvantages

8. Conclusion

9. Future Scope

    10. Appendix

    10.1. Source Code

    10.2. GitHub & Project Demo Link

1. Introduction

1.1 Project Overviews

The project "TechPart Vision" aims to develop a robust image classification model to accurately classify and identify various personal computer(PC) parts. Utilizing the EfficientNet architecture and transfer learning techniques, this project leverages pre-trained models to enhance classification accuracy and efficiency.

1.2 Objectives

- To build a high-accuracy image classification model for PC parts.
- To aid in activities such as invemtory management, online shopping and technical support.

2. Project Initialization and Planning Phase

2.1 Define Problem Statement

The primary challenge addressed by this project is the automatic and accurate identification of personal computer parts from images, which is essential for inventory management, online marketplaces, and technical support.

2.2 Project Proposal (Proposed Solution)

The proposed solution involves using EfficientNet for transfer learning to develop a classification model that can identify different PC parts. The model will be trained on a labelled dataset of PC part images and will undergo several optimizations to achieve high accuracy.

2.3 Initial Project Planning

- Identify and collect a diverse dataset of PC part images.
- Data preprocessing to bring it up to standard
- Choosing the appropriate model for high accuracy and efficiency
- Evaluation/validation followed by deployment

3. Data Collection and Preprocessing Phase

3.1 Data Collection Plan and Raw Data Sources Identified

Dataset was collected from Kaggle, whose raw data was scraped from google images.

3.2 Data Quality Report

- Dataset had issues such as missing values and inconsistent image sizes.
- Solutions such as resizing images, and reducing the number to match the least represented group were implemented.

3.3 Data Preprocessing

- Resizing and normalizing images for consistent input to the model.
- Augmenting the dataset with transformations such as rotations, flips, and brightness adjustments to enhance model robustness.
- Splitting the dataset into training, validation, and test sets.

4. Model Development Phase

4.1 Model Selection Report

Initial Model: Custom CNN

Structure:

- Layers: Convolutional, MaxPooling, Dropout, Flatten, Dense
- Activation: ReLU (convolutional and dense), Softmax (final dense)
- Optimizer: Adam
- Loss Function: Categorical cross-entropy
- Training: 10 epochs, batch size 32, verbose 1

Performance:

- Result: Underfitting; low accuracy

Transition to EfficientNetV2B1

Rationale:

- Selected for its efficiency and performance, suitable for deployment with limited resources
- Utilizes pre-trained ImageNet weights for transfer learning
- Enhanced accuracy with fewer training samples

Custom Layers Added:

- GlobalAveragePooling for dimension reduction
- BatchNormalization for training stability
- Dense layer with ReLU and regularization to prevent overfitting
- Dropout layer for further overfitting reduction
- Softmax activation in output layer for classification into 14 categories

Performance:

- Significant improvement in training and validation accuracy and loss
- Outperformed VGG19

Comparison with VGG19

Structure:

- VGG19 without top classification layer (include_top=False)
- Added: GlobalAveragePooling2D, two Dense layers (1024 units with ReLU, final layer with softmax)

Performance:

- Initial improvements but validation accuracy plateaued, indicating overfitting

Transition Justification:

- EfficientNetV2B1 showed better training and validation accuracy, justifying its selection over VGG19.

4.2. **Initial Model Training Code, Model Validation and Evaluation Report**

CNN Model

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten
from keras.layers import Convolution2D, MaxPooling2D
import tensorflow as tf
```

```
initializer = tf.keras.initializers.HeNormal()
```

```
model=Sequential()
model.add(Convolution2D(filters=32, kernel_size=3, padding='same', activation="relu",
                        input_shape=(255, 255, 3)))
model.add(MaxPooling2D(strides=2, pool_size=2, padding= "valid"))
model.add(Convolution2D(filters=32, kernel_size=2, padding='same', activation="relu"))
model.add(MaxPooling2D(strides=2, pool_size=2, padding="valid"))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_initializer=initializer))
model.add(Dropout(0.5))
model.add(Dense(14, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

EfficientNetV2B1 Model Initialisation

```
base_model=tf.keras.applications.EfficientNetV2B1(include_top=False, weights="imagenet",input_shape=(255,255,3), pooling='max')
print('Created EfficientNetV2 B1 model')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/efficientnet_v2/efficientnetv2-b1_notop.h5
28456008/28456008 [==============================] - 2s 0us/step
Created EfficientNetV2 B1 model
```

```
base_model.trainable=True
x=base_model.output
x=BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x)
x = Dense(256, kernel_regularizer = regularizers.l2(l = 0.016),activity_regularizer=regularizers.l1(0.006),
                bias_regularizer=regularizers.l1(0.006) ,activation='relu')(x)
x=Dropout(rate=.4, seed=123)(x)
output=Dense(14, activation='softmax')(x)
model=Model(inputs=base_model.input, outputs=output)
model.compile(Adamax(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
from tensorflow.keras.applications import VGG19
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(len(train_gen.class_indices), activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

for layer in base_model.layers:
    layer.trainable = False

model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

5. Model Optimization and Tuning Phase

5.1 Tuning Documentation

CNN Sequential model:

Convolutional Layers: Number of Filters: 32; Kernel Size: 3 (first layer), 2 (second layer); Activation Function: ReLU; Padding: Same

Pooling Layers: Pooling Size: 2

Regularization: Dropout Rate: 0.5

Dense Layers: Units: 128; Kernel Initializer: HeNormal

Output Layer: Units: 14

Training Parameters: Batch Size: 32; Number of Epochs: 10

EfficientnetV2B1:

Base Model

- Base Model: EfficientNetV2B1 (pretrained on ImageNet, used as a feature extractor)

Pooling

- Pooling: Max (global max pooling applied to the output of the base model)

Batch Normalization

- Axis: -1
- Momentum: 0.99
- Epsilon: 0.001

Dense Layer

- Units: 256

Regularization

- Kernel Regularization: l2(0.016)
- Activity Regularization: l1(0.006)
- Bias Regularization: l1(0.006)

Activation

- Activation Function: ReLU

Dropout

- Rate: 0.4

Output Layer

- Units: 14 (softmax activation for multi-class classification)

Optimizer

- Optimizer: Adamax
- Learning Rate: 0.001

Callbacks

- ReduceLROnPlateau:
  - Monitor: "val_loss"
  - Factor: 0.4
  - Patience: 2
  - Min LR: 0.0
- EarlyStopping:
  - Monitor: "val_loss"
  - Patience: 2
  - Restore Best Weights: True

Training Parameters

- Number of Epochs: 5

VGG19:

Base Model

- Base Model: VGG19 (pretrained on ImageNet, used as a feature extractor)
- Input Shape: (224, 224, 3)

Pooling

- Pooling: Global Average Pooling

Dense Layers

- Units: 1024
- Activation Function: ReLU

Output Layer

- Units: len(train_gen.class_indices)
- Activation Function: Softmax

Optimizer

- Optimizer: Adam
- Learning Rate: 0.001

Loss Function

- Loss Function: Categorical Crossentropy

Metrics

- Metrics: Accuracy

Freezing Layers

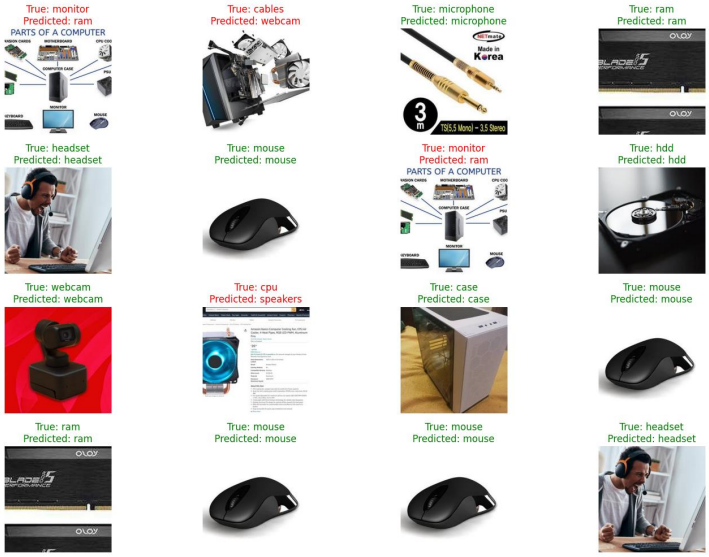
- Freezing Layers: Initially freeze all base model layers

5.2 Final Model Selection Justification

The comparison between VGG19 and EfficientNetV2B1 shows that EfficientNetV2B1 outperforms VGG19 in several key areas. EfficientNetV2B1 achieves higher training accuracy more quickly and maintains a relatively stable validation accuracy around 0.75, while VGG19's validation accuracy fluctuates and remains generally lower. Both models exhibit a significant decrease in training loss, but EfficientNetV2B1 starts with a higher value and decreases more sharply initially. Additionally, EfficientNetV2B1's validation loss shows a more consistent decrease, whereas VGG19's validation loss fluctuates after the initial decrease. Overall, EfficientNetV2B1 demonstrates better performance with higher and more stable validation accuracy and a consistent decrease in validation loss, indicating quicker and more reliable improvements compared to VGG19.

6. Results

This project was able to classify various PC parts such as keyboards, monitors, cables, RAM, HDD etc, and identify them with significant accuracy.



From website:

7. Advantages & Disadvantages

Advantages

- Significant accuracy in classifying PC parts.
- Efficient and fast training with EfficientNet.
- Transfer learning reduces the need for a large dataset.

Disadvantages

- Requires significant computational resources.
- Performance is dependent on the limited dataset.

8. Conclusion

The project successfully implemented EfficientNet for classifying PC parts with high accuracy. The model's performance demonstrated the effectiveness of transfer learning for image classification tasks.

9. Future Scope

- Extending the model to classify more types of PC parts.
- Integrating the model into real-time inventory or e-commerce systems.
- Exploring other advanced architectures and techniques for further improvement.

10. Appendix

4.1 Source Code

```
import pandas as pd
import numpy as np
from numpy import random
import os

from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Activation,Dropout,Conv2D, MaxPooling2D,BatchNormalization
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras import regularizers
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K
import albumentations as A
from sklearn.metrics import f1_score
import cv2
```

Loading Image set into colab

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

```
Mounted at /content/drive
```

```
!unzip /content/drive/MyDrive/archive.zip
```

```
directory="/content/pc_parts"
```

```
labels = os.listdir(directory)
labels
```

Creating Dataframe

```
def read_data(folder):
    data, label, paths = [], [], []
    for l in labels:
        path = f"{folder}/{l}/"
        folder_data = os.listdir(path)
        for image_path in folder_data:
            img = cv2.imread(path + image_path)
            data.append(img)
            label.append(l)
            paths.append(os.path.join(directory, l, image_path))

    return data, label, paths
```

```
import cv2
import matplotlib.pyplot as plt

def edge_detection(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray_image, 100, 200)
    return edges

image_path = 'path_to_image.jpg'
image = cv2.imread(image_path)
edges = edge_detection(image)
```

```
all_data, all_labels, all_paths = read_data(directory)
```

```
df = pd.DataFrame({
    'image':all_data,
    'path': all_paths,
    'label': all_labels
})
```

```
train_df, dummy_df=train_test_split(df, train_size=.8, random_state=123, shuffle=True, stratify=df['label'])
valid_df, test_df=train_test_split(dummy_df, train_size=.5, random_state=123, shuffle=True, stratify=dummy_df['label'])
print("Train dataset : ",len(train_df),"Test dataset : ",len(test_df),"Validation dataset : ",len(valid_df))
train_balance=train_df['label'].value_counts()
print('Train dataset value count: \n',train_df['label'].value_counts())
```

```
Train dataset :  2623 Test dataset :  328 Validation dataset :  328
Train dataset value count:
 label
cables         238
speakers       237
case           225
keyboard       214
headset        211
hdd            210
monitor        205
motherboard    193
ram            181
microphone     171
mouse          168
webcam         131
gpu            125
cpu            114
Name: count, dtype: int64
```

```
import plotly.express as px
```

```
px.histogram(train_df, x='label', barmode='group')
```

Data Augmentation

```
import cv2
import numpy as np
```

```
def apply_transform(image):

    # Rotate (random angle between -40 and 40 degrees)
    angle = np.random.uniform(-40, 40)
    rows, cols = image.shape[:2]
    M = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)
    image = cv2.warpAffine(image, M, (cols, rows))

    # Horizontal Flip
    if np.random.rand() < 0.5:
        image = cv2.flip(image, 1)

    # Vertical Flip
    if np.random.rand() < 0.5:
        image = cv2.flip(image, 0)

    # Random Brightness and Contrast
    alpha = 1.0 + np.random.uniform(-0.2, 0.2)  # Brightness
    beta = 0.0 + np.random.uniform(-0.2, 0.2)   # Contrast
    image = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

    # Random Gamma Correction
    gamma = np.random.uniform(0.8, 1.2)
    image = np.clip((image / 255.0) ** gamma, 0, 1) * 255.0

    return image
```

```
def apply_augmentation(image_path, label):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    augmented_image = apply_transform(image=image)
    return augmented_image, label
```

```
augmented_data = []
for index, row in train_df.iterrows():
    image_path = row['path']
    label = row['label']
    augmented_image, augmented_label = apply_augmentation(image_path, label)
    augmented_data.append((augmented_image, augmented_label, image_path))
```

```
augmented_df = pd.DataFrame(augmented_data, columns=['object', 'label', 'path'])
```

```
merged_df = pd.concat([train_df, augmented_df], ignore_index=True)
```

```
import cv2
import matplotlib.pyplot as plt

def denoise_image(image):
    denoised_image = cv2.fastNlMeansDenoisingColored(image, None, 10, 10, 7, 21)
    return denoised_image


image_path = 'path_to_image.jpg'
image = cv2.imread(image_path)
denoised_image = denoise_image(image)
```

```
from collections import Counter
```

```
class_counts = Counter(merged_df['label'])
min_class_count = min(class_counts.values())

balanced_data = []
for label, count in class_counts.items():
    class_samples = merged_df[merged_df['label'] == label].sample(min_class_count)
    balanced_data.append(class_samples)

balanced_df = pd.concat(balanced_data)
```

```
balanced_df['label'].value_counts()
```

```
label
mouse          228
keyboard       228
motherboard    228
cpu            228
microphone     228
webcam         228
hdd            228
monitor        228
cables         228
ram            228
gpu            228
case           228
headset        228
speakers       228
Name: count, dtype: int64
```

```
gen=ImageDataGenerator()
```

```
train_gen=gen.flow_from_dataframe(balanced_df, x_col='path', y_col='label', target_size=(255,255),seed=123,
                                  class_mode='categorical', color_mode='rgb', shuffle=True, batch_size=32)
```

```
Found 3192 validated image filenames belonging to 14 classes.
```

```
def resize_image(image, size=(255, 255)):
    resized_image = cv2.resize(image, size)
    return resized_image
```

```
val_gen=gen.flow_from_dataframe(valid_df, x_col='path', y_col='label', target_size=(255,255),seed=123,
                                class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=32)
```

```
Found 328 validated image filenames belonging to 14 classes.
```

```
test_gen=gen.flow_from_dataframe(test_df, x_col='path', y_col='label', target_size=(255,255),seed=123,
                                class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=32)
```

```
Found 328 validated image filenames belonging to 14 classes.
```

CNN Model

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten
from keras.layers import Convolution2D, MaxPooling2D
import tensorflow as tf
```

```
initializer = tf.keras.initializers.HeNormal()
```

```
model=Sequential()
model.add(Convolution2D(filters=32, kernel_size=3, padding='same', activation="relu",
                        input_shape=(255, 255, 3)))
model.add(MaxPooling2D(strides=2, pool_size=2, padding= "valid"))
model.add(Convolution2D(filters=32, kernel_size=2, padding='same', activation="relu"))
model.add(MaxPooling2D(strides=2, pool_size=2, padding="valid"))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_initializer=initializer))
model.add(Dropout(0.5))
model.add(Dense(14, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model.fit(train_gen, validation_data= val_gen, batch_size= 32, epochs = 10, verbose = 1)
```

```
Epoch 1/10
100/100 [==============================] - 18s 106ms/step - loss: 97.1222 - accuracy: 0.0702 - val_loss: 2.6387 - val_accuracy: 0.0823
Epoch 2/10
100/100 [==============================] - 10s 97ms/step - loss: 2.6393 - accuracy: 0.0667 - val_loss: 2.6387 - val_accuracy: 0.0823
Epoch 3/10
100/100 [==============================] - 10s 101ms/step - loss: 2.6402 - accuracy: 0.0674 - val_loss: 2.6388 - val_accuracy: 0.0823
Epoch 4/10
100/100 [==============================] - 10s 100ms/step - loss: 2.6393 - accuracy: 0.0708 - val_loss: 2.6388 - val_accuracy: 0.0823
Epoch 5/10
100/100 [==============================] - 9s 90ms/step - loss: 2.6393 - accuracy: 0.0620 - val_loss: 2.6389 - val_accuracy: 0.0823
Epoch 6/10
100/100 [==============================] - 11s 107ms/step - loss: 2.6393 - accuracy: 0.0689 - val_loss: 2.6390 - val_accuracy: 0.0427
Epoch 7/10
100/100 [==============================] - 10s 95ms/step - loss: 2.6393 - accuracy: 0.0680 - val_loss: 2.6390 - val_accuracy: 0.0823
Epoch 8/10
100/100 [==============================] - 10s 97ms/step - loss: 2.6394 - accuracy: 0.0636 - val_loss: 2.6389 - val_accuracy: 0.0915
Epoch 9/10
100/100 [==============================] - 10s 100ms/step - loss: 2.6393 - accuracy: 0.0711 - val_loss: 2.6390 - val_accuracy: 0.0823
Epoch 10/10
100/100 [==============================] - 10s 99ms/step - loss: 2.6393 - accuracy: 0.0623 - val_loss: 2.6390 - val_accuracy: 0.0823
```

EfficientNetV2B1 Model Initialisation

```
base_model=tf.keras.applications.EfficientNetV2B1(include_top=False, weights="imagenet",input_shape=(255,255,3), pooling='max')
print('Created EfficientNetV2 B1 model')
```

Python

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/efficientnet_v2/efficientnetv2-b1_notop.h5
28456008/28456008 [==============================] - 2s 0us/step
Created EfficientNetV2 B1 model
```

```
base_model.trainable=True
x=base_model.output
x=BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x)
x = Dense(256, kernel_regularizer = regularizers.l2(l = 0.016),activity_regularizer=regularizers.l1(0.006),
                bias_regularizer=regularizers.l1(0.006) ,activation='relu')(x)
x=Dropout(rate=.4, seed=123)(x)
output=Dense(14, activation='softmax')(x)
model=Model(inputs=base_model.input, outputs=output)
model.compile(Adamax(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

Python

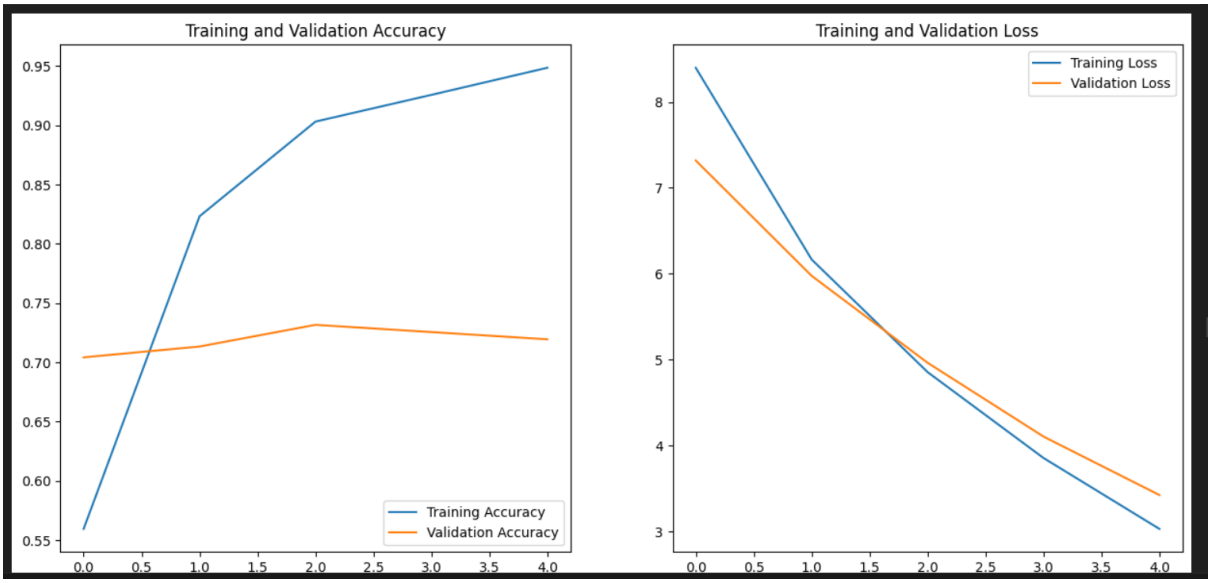
```
rlronp=keras.callbacks.ReduceLROnPlateau( monitor="val_loss", factor=0.4, patience=2, verbose=1, mode="auto",  min_delta=0.00001, cooldown=0, min_lr=0.0
estop=keras.callbacks.EarlyStopping( monitor="val_loss", min_delta=0,  patience=2, verbose=1, mode="auto", baseline=None, restore_best_weights=True)
callbacks=[rlronp, estop]
```

```
history=model.fit(x=train_gen,   epochs=5, verbose=1, callbacks=callbacks,  validation_data=val_gen,
                validation_steps=None,  shuffle=True)
```

```
Epoch 1/5
100/100 [==============================] - 94s 423ms/step - loss: 8.3989 - accuracy: 0.5595 - val_loss: 7.3195 - val_accuracy: 0.7043 - lr: 0.0010
Epoch 2/5
100/100 [==============================] - 33s 330ms/step - loss: 6.1665 - accuracy: 0.8233 - val_loss: 5.9759 - val_accuracy: 0.7134 - lr: 0.0010
Epoch 3/5
100/100 [==============================] - 33s 334ms/step - loss: 4.8552 - accuracy: 0.9032 - val_loss: 4.9639 - val_accuracy: 0.7317 - lr: 0.0010
Epoch 4/5
100/100 [==============================] - 34s 335ms/step - loss: 3.8550 - accuracy: 0.9258 - val_loss: 4.1046 - val_accuracy: 0.7256 - lr: 0.0010
Epoch 5/5
100/100 [==============================] - 34s 337ms/step - loss: 3.0294 - accuracy: 0.9486 - val_loss: 3.4238 - val_accuracy: 0.7195 - lr: 0.0010
```

```
import matplotlib.pyplot as plt
```

```python
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(5)

plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

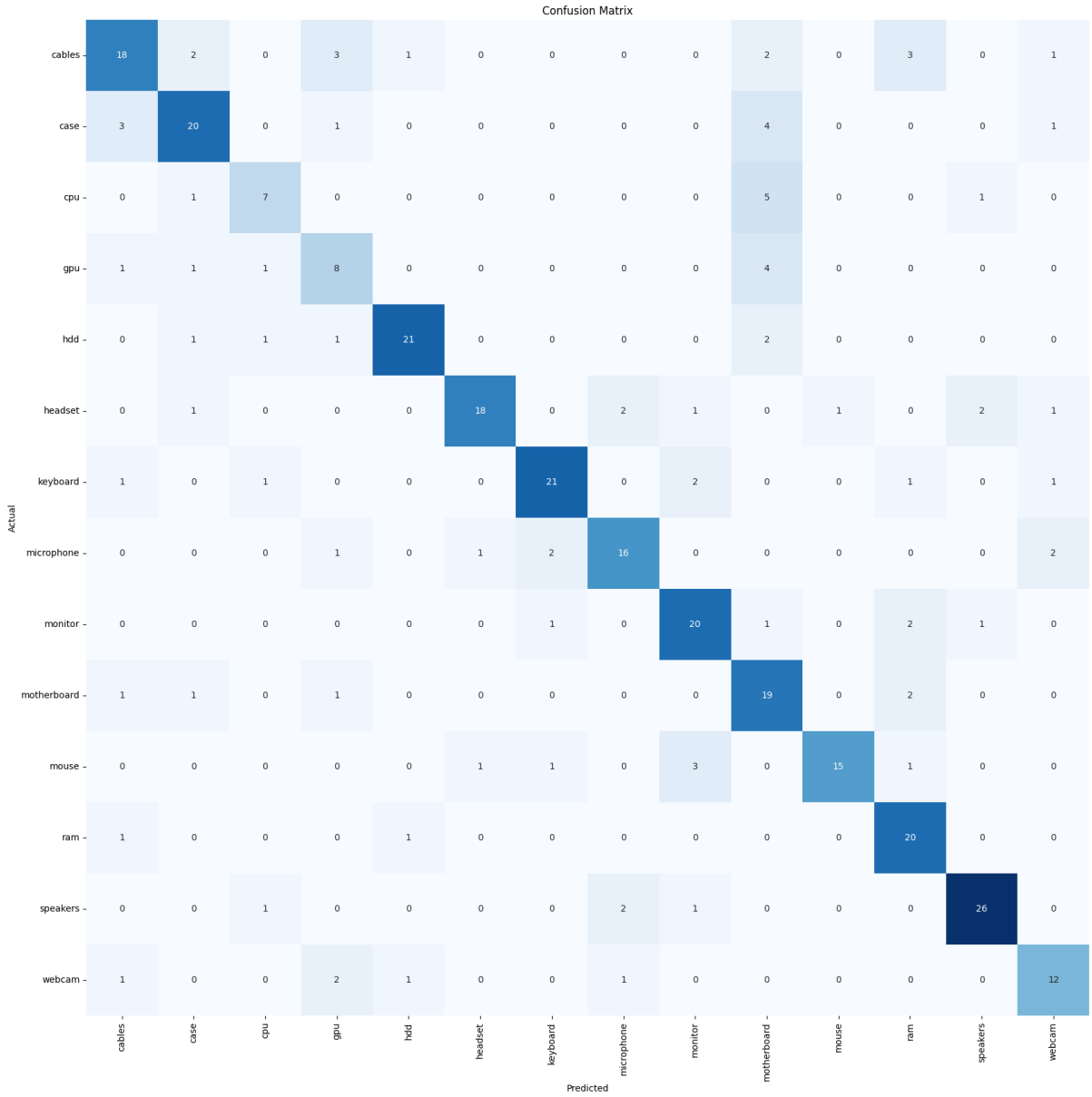



```python
import seaborn as sns
```

```python
def predictor(model,test_gen):
    classes=list(test_gen.class_indices.keys())
    class_count=len(classes)
    preds=model.predict(test_gen, verbose=1)
    errors=0
    pred_indices=[]
    test_count =len(preds)
    for i, p in enumerate (preds):
        pred_index=np.argmax(p)
        pred_indices.append(pred_index)
        true_index= test_gen.labels[i]
        if  pred_index != true_index:
            errors +=1
    accuracy = (test_count-errors)*100/test_count
    ytrue=np.array(test_gen.labels, dtype='int')
    ypred=np.array(pred_indices, dtype='int')
    msg=f'There were {errors} errors in {test_count} tests for an accuracy of {accuracy:6.2f}'
    print (msg)
    cm = confusion_matrix(ytrue, ypred )
    # plot the confusion matrix
    plt.figure(figsize=(20, 20))
    sns.heatmap(cm, annot=True, vmin=0, fmt='g', cmap='Blues', cbar=False)
    plt.xticks(np.arange(class_count)+.5, classes, rotation=90)
    plt.yticks(np.arange(class_count)+.5, classes, rotation=0)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix")
    plt.show()
    clr = classification_report(ytrue, ypred, target_names=classes, digits= 4) # create classification report
    print("Classification Report:\n----------------------\n", clr)
    return
```

```
predictor(model,test_gen) # make predictions on test set
```

```
11/11 [==============================] - 1s 86ms/step
There were 87 errors in 328 tests for an accuracy of  73.48
```



```
Classification Report:
----------------------
              precision    recall  f1-score   support

      cables     0.6923    0.6000    0.6429        30
        case     0.7407    0.6897    0.7143        29
         cpu     0.6364    0.5000    0.5600        14
         gpu     0.4706    0.5333    0.5000        15
         hdd     0.8750    0.8077    0.8400        26
     headset     0.9000    0.6923    0.7826        26
    keyboard     0.8400    0.7778    0.8077        27
  microphone     0.7619    0.7273    0.7442        22
     monitor     0.7407    0.8000    0.7692        25
 motherboard     0.5135    0.7917    0.6230        24
       mouse     0.9375    0.7143    0.8108        21
         ram     0.6897    0.9091    0.7843        22
    speakers     0.8667    0.8667    0.8667        30
      webcam     0.6667    0.7059    0.6857        17

    accuracy                         0.7348       328
   macro avg     0.7380    0.7225    0.7237       328
weighted avg     0.7526    0.7348    0.7373       328
```

```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

```
test_set = ['/content/pc_parts/webcam/109.jpg','/content/pc_parts/cables/105.jpg','/content/pc_parts/case/103.jpg',
            '/content/pc_parts/cpu/110.jpg', '/content/pc_parts/gpu/127.jpg', '/content/pc_parts/hdd/11.jpg',
            '/content/pc_parts/headset/107.jpg', '/content/pc_parts/keyboard/108.jpg', '/content/pc_parts/microphone/103.jpg',
            '/content/pc_parts/monitor/104.jpg', '/content/pc_parts/motherboard/105.jpg', '/content/pc_parts/mouse/102.jpg',
            '/content/pc_parts/ram/101.jpg','/content/pc_parts/speakers/10.jpg'
            ]
```

```
labels.sort()
```

```
labels
```

```
['cables',
 'case',
 'cpu',
 'gpu',
 'hdd',
 'headset',
 'keyboard',
 'microphone',
 'monitor',
 'motherboard',
 'mouse',
 'ram',
 'speakers',
 'webcam']
```

```
def get_model_prediction(image_path):
    img = load_img(image_path, target_size=(255, 255))
    x = img_to_array(img)
    x = np.expand_dims(x, axis=0)
    predictions = model.predict(x, verbose=0)
    return labels[predictions.argmax()]
```

Saving Model

```
model.save("model.h5")
print("Saved model to disk")
```

Python

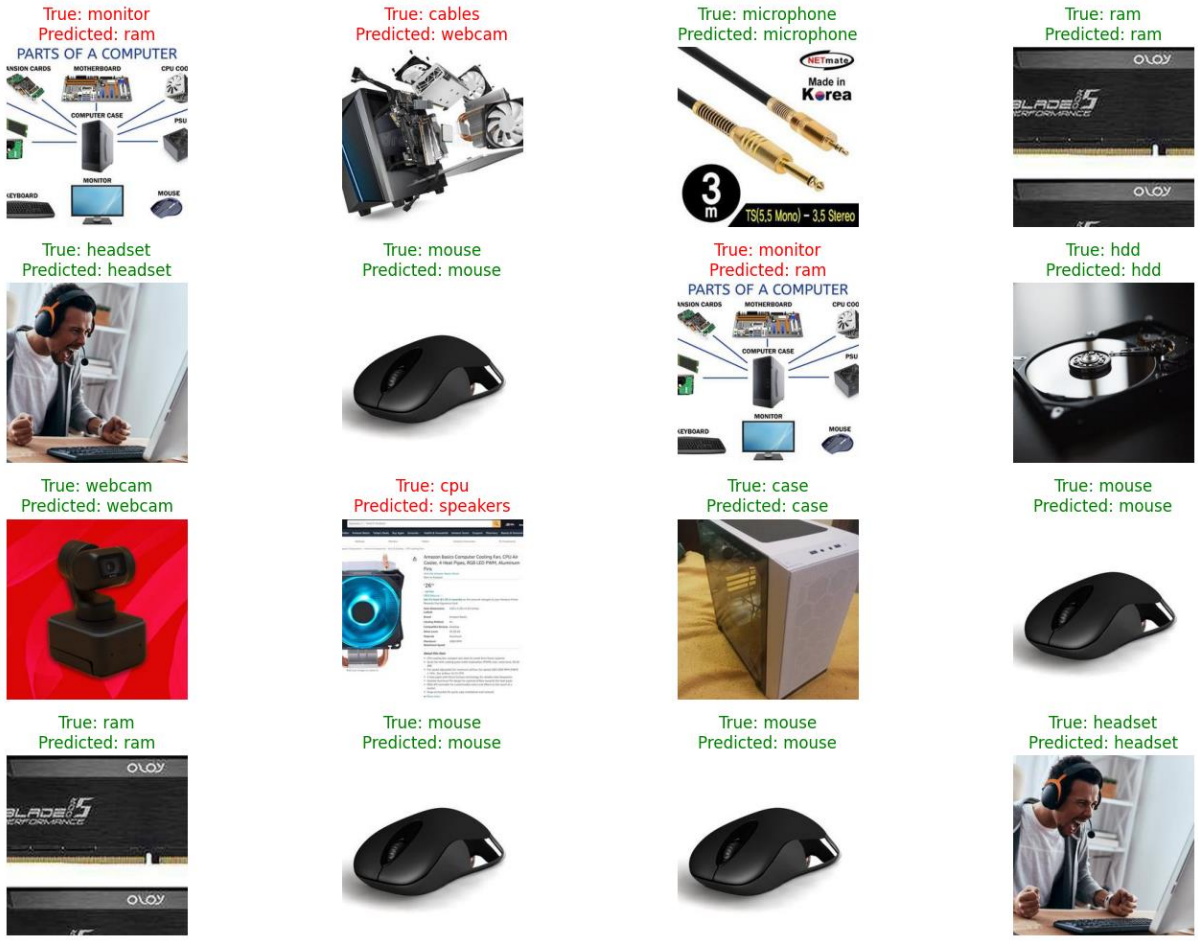
```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning:

You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g

Saved model to disk
```

Manual Testing

```
pred = []
for file in test_df['path'].values:
    pred.append(get_model_prediction(file))
```

Python

```
fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(15, 10))
random_index = np.random.randint(0, len(test_gen), 16)

for i, ax in enumerate(axes.ravel()):
    img_path = test_df['path'].iloc[random_index[i]]

    ax.imshow( load_img(img_path))
    ax.axis('off')

    if test_df['label'].iloc[random_index[i]] == pred[random_index[i]]:
        color = "green"
    else:
        color = "red"

    ax.set_title(f"True: {test_df['label'].iloc[random_index[i]]}\nPredicted: {pred[random_index[i]]}", color=color)

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



4.2 Github and Project demo link

Project Demo link:

https://vitapacin-my.sharepoint.com/:v:/g/personal/kowshik_22bce9556_vitapstudent_ac_in/Ef3AmwMXiQZNu2fK-iNm1c4BcC77mJ2tRjCON_-WhX1GKw?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJTdHJlYW1XZWJBcHAiLCJyZWZlcnJhbFZpZXciOiJTaGFyZURpYWxvZy1MaW5rIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXcifX0%3D&e=vevLJx

Github:

https://github.com/kowshikdontu/TechPart-Vision