Configuration Manual

MSc Research Project
MSc in Data Analytics

Dhwani Dharmesh Hingu
Student ID: X19216742

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Dhwani Dharmesh Hingu
Student ID:	X19216742
Programme:	MSc in Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr. Catherine Mulwa
Submission Due Date:	31/01/2022
Project Title:	Configuration Manual
Word Count:	645
Page Count:	13

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Dhwani Hingu
Date:	31st January 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

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

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

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

Configuration Manual

Dhwani Dharmesh Hingu
X19216742

1 Introduction

This Configuration Manuals covers all steps how the research study was build, implemented, and executed with support of some hardware and software configurations.

2 System Configuration

This Research study have used images and deep learning models, so in order to successfully run all programs with an ease some hardware and software configurations needs to be taken into consideration before starting the project. The Hardware Setup Section 2.1 helps with hardware specification needs to have also on the other hand Software setup section 2.2 aids with programming language used along with all necessary libraries needed to be installed with their versions.

2.1 Hardware Setup

In order to make the models perform well, GPU setting was changed in NVIDIA controll panel and the steps are shown in the Figure 2, 3 and 4

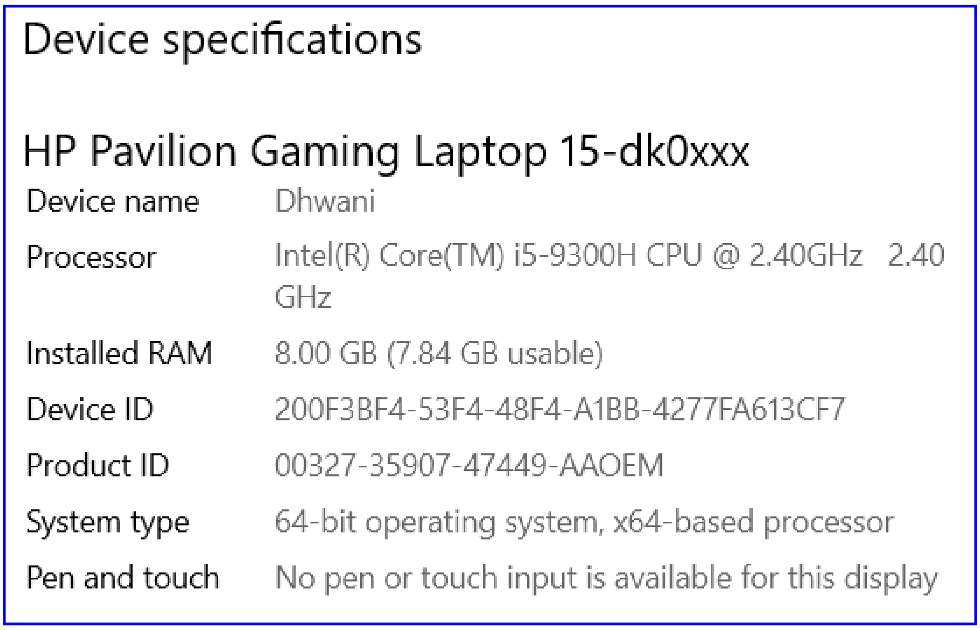

Figure 1: Computer Specifications

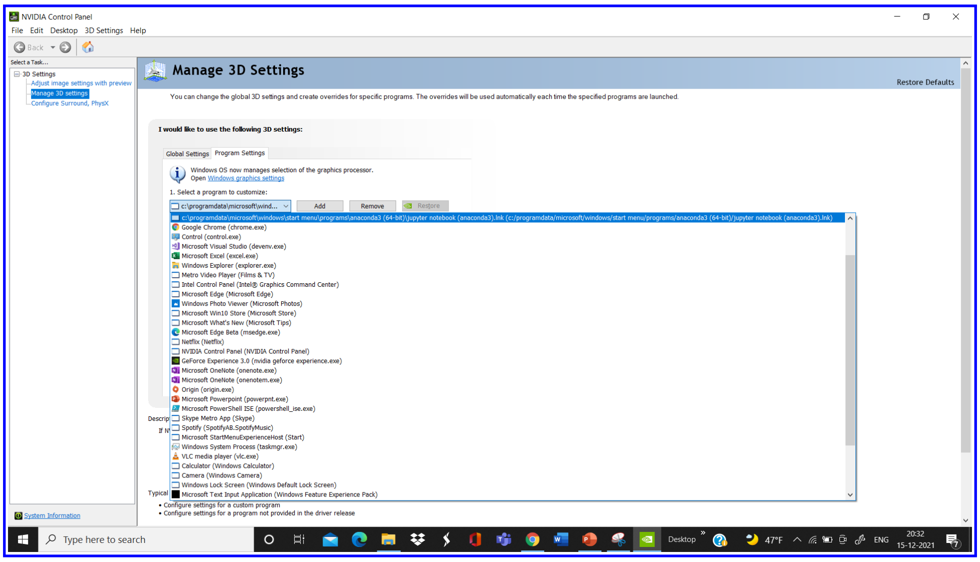

Figure 2: GPU Setting

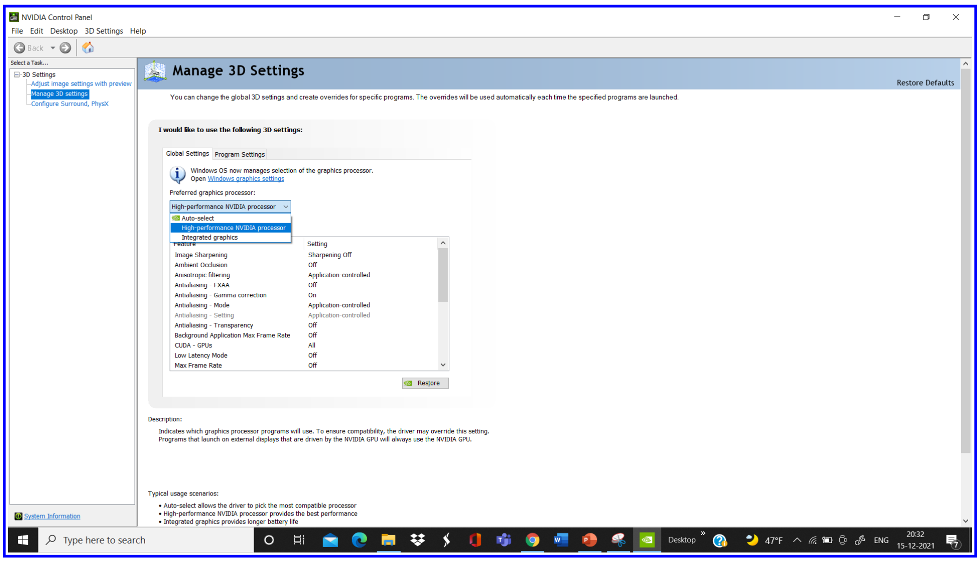

Figure 3: Setting GPU as High Performance

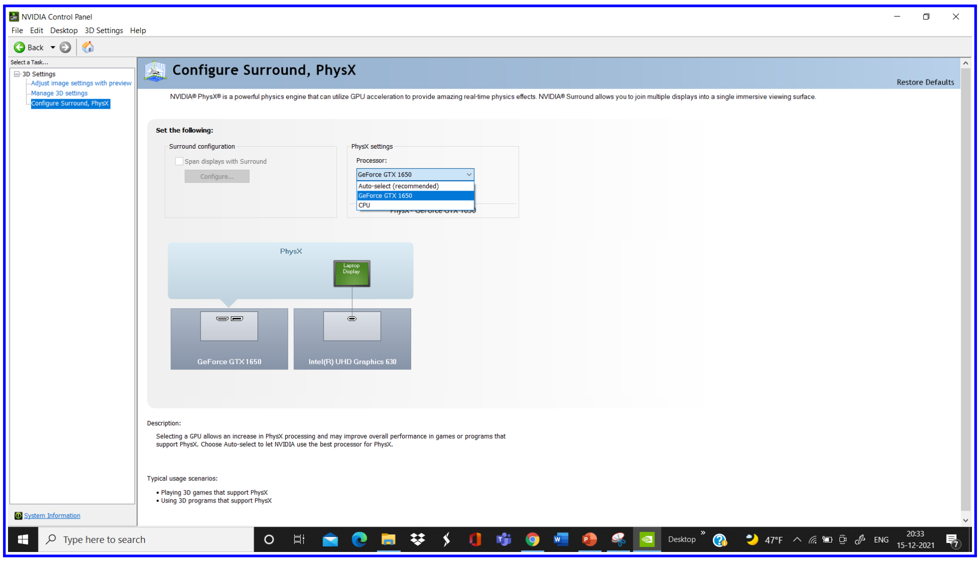

Figure 4: Assigning GPU Processor

2.2 Software Setup

Python as programming language was used and Jupyter notebook was used as shown in the 5 to carry out all code. Necessary python libraries along with their versions are mentioned in the Table 1 which needs to be installed in order to build and execute further experiments.

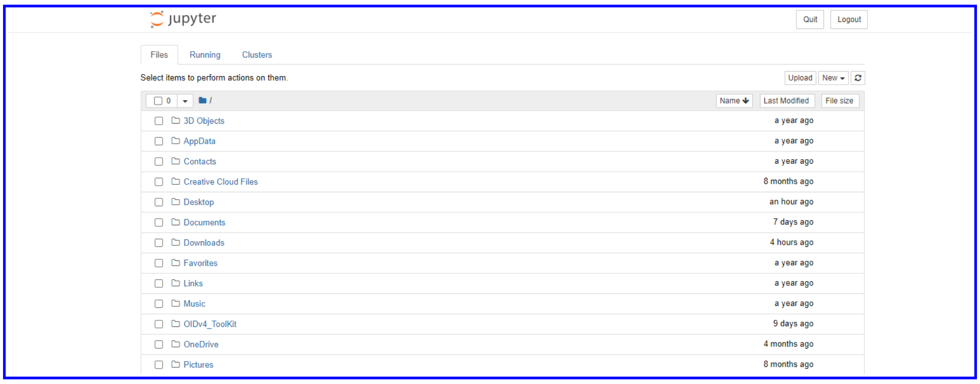

Figure 5: Snapshot of Jupyter

pandas	1.1.3
numpy	1.19.2
sklearn	0.24.1
matplotlib	3.3.2
seaborn	0.11.0
tensorflow	2.7.0
imblearn	0.8.0
keras	2.7.0
cv2	4.5.4

Table 1: Libraries and their versions

3 Data Gathering

Google Open Image Dataset [1] as seen in the Figure 6. The google open image dataset consisted of 9M images with more than 600 classes. For this research study 10 classes images (ie.) were extracted. Three different dataset were created as bellow, for train in the Figure 7, for test in the Figure 8 and for validation as shown in the Figure 9

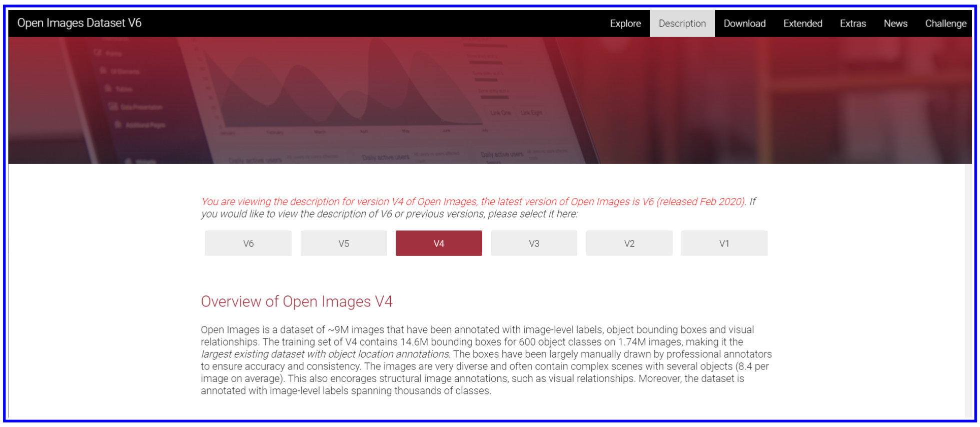

Figure 6: Dataset Used : Google Open Image Dataset V4

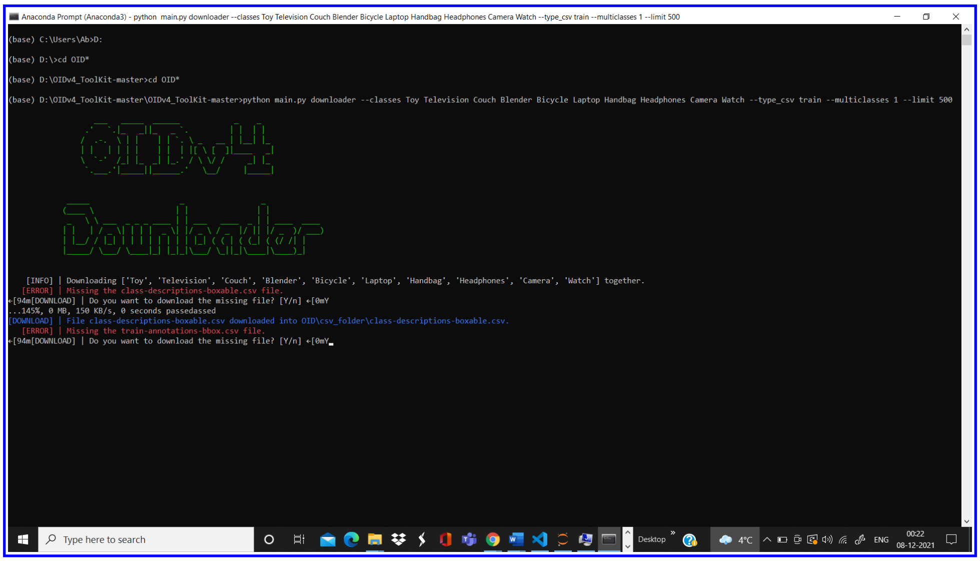

Figure 7: Extraction of Train Data

[1] https://storage.googleapis.com/openimages/web/factsfigures_v4.html

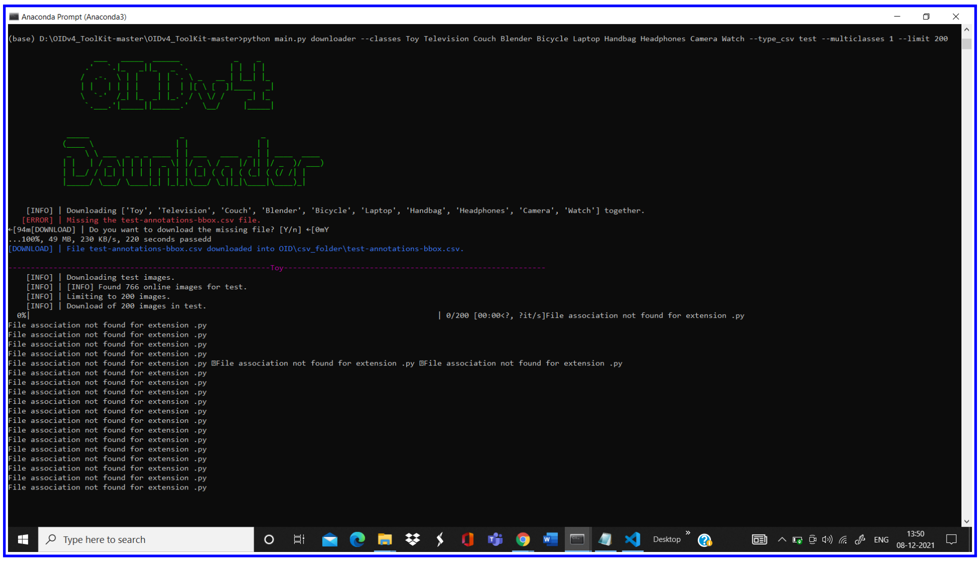

Figure 8: Extraction of Test Data

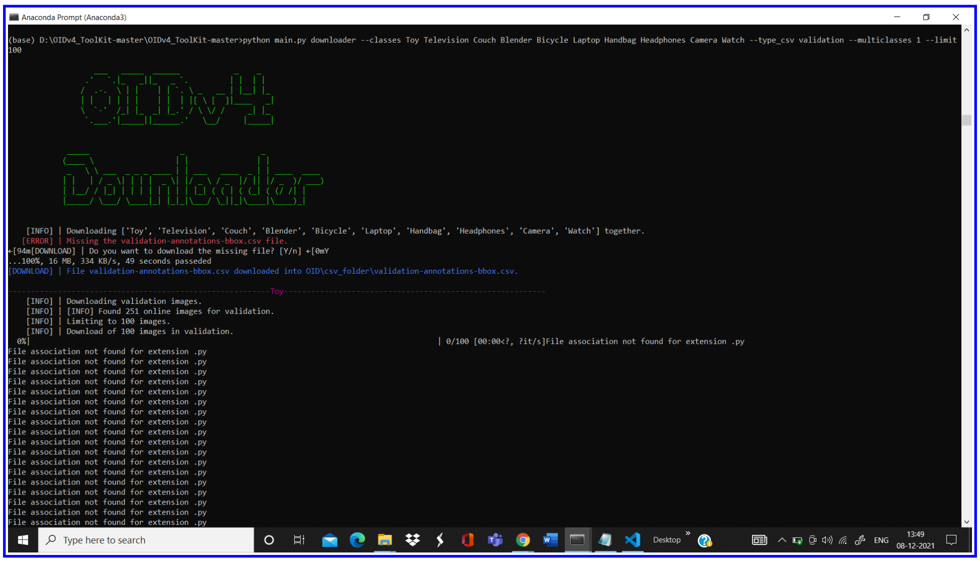

Figure 9: Extraction of Validation Data

4 Data Preparation and Transformation

The dataset had images of all different size, so scaling data to a fixed sized was carried out and data augmentation was done using ImageDataGenerator Class as shown in the Figure 11

```
# Directories
train_directory = "D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/train"
valid_directory = "D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/validation"
test_directory = "D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/test"
```

Figure 10: Setting Path for all Dataset Directory

```python
# Constants
img_rows = 224
img_cols = 224
input_shape = (img_rows,img_cols,3)
epochs = 10
batch_size = 64
num_of_classes = 10
num_of_train_samples = 4649
num_of_valid_samples = 537
num_of_test_samples = 1327

# Load Train Data as image generator with Keras, Normalization with (rescale=1./255)
train_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(directory=train_directory,
                                                                         class_mode='categorical',
                                                                         batch_size=batch_size,
                                                                         target_size=(img_rows, img_cols),
                                                                         color_mode="rgb",
                                                                         shuffle=True)

# Load Validation Data as image generator with Keras, Normalization with (rescale=1./255)
valid_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(directory=valid_directory,
                                                                         class_mode='categorical',
                                                                         batch_size=batch_size,
                                                                         target_size=(img_rows, img_cols),
                                                                         color_mode="rgb",
                                                                         shuffle=True)

# Load Test Data as image generator with Keras, Normalization with (rescale=1./255)
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(directory=test_directory,
                                                                        class_mode='categorical',
                                                                        batch_size=batch_size,
                                                                        target_size=(img_rows, img_cols),
                                                                        color_mode="rgb",
                                                                        shuffle=False)
```

Figure 11: Extraction of Validation Data

5 Experimental Setup

5.1 Experiment with CNN

CNN Model's parameter were changed in order to achieve good performing model. By tuning these parameters various combinations of experiments were conducted and further compared for evaluation. The adjusted parameters are highlighted in each figure.

CNN Model

```python
# Create a model for CNN
def getCNNModel(num_of_layers=2, num_of_filters=32, filter_size=(3, 3), initializer='glorot_uniform',
                activation_function='relu', dropout=0.2, opt='adam'):
  # Create a Sequential
  model = Sequential()

  # Create Convolution Layers
  for i in range(0, num_of_layers):
    model.add(Conv2D(num_of_filters, filter_size, kernel_initializer=initializer, activation=activation_function,
                     input_shape=input_shape))
    model.add(MaxPooling2D(2, 2))
    model.add(Dropout(dropout))

  model.add(Flatten())

  model.add(Dense(512, activation=activation_function, kernel_initializer=initializer))
  model.add(Dense(num_of_classes, activation='softmax'))

  # Compile model
  model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
  model.summary()

  return model
```

Figure 12: CNN Model Building Code

```python
# Combination 1
model = getCNNModel(num_of_layers=2, num_of_filters=32, filter_size=(3, 3), initializer='glorot_normal',
                    activation_function='relu', dropout=0.2, opt='adam')
model = trainModelAndGetConfusionMatrix(model)
model.save('D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/New_model_comb1.h5')
```

Figure 13: CNN Combination1

```
# Combination 2
model = getCNNModel(num_of_layers=2, num_of_filters=32, filter_size=(5, 5), initializer='glorot_normal',
                    activation_function='relu', dropout=0.2, opt='adam')
model = trainModelAndGetConfusionMatrix(model)
model.save('D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/model_comb2.h5')
```

Figure 14: CNN Combination2

```
# Combination 3
model = getCNNModel(num_of_layers=2, num_of_filters=32, filter_size=(3, 3), initializer='glorot_normal',
                    activation_function='relu', dropout=0.7, opt='adam')
model = trainModelAndGetConfusionMatrix(model)
model.save('D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/model_comb3.h5')
```

Figure 15: CNN Combination3

```
# Combination 4
model = getCNNModel(num_of_layers=3, num_of_filters=32, filter_size=(3, 3), initializer='glorot_uniform',
                    activation_function='relu', dropout=0.2, opt='adam')
model = trainModelAndGetConfusionMatrix(model)
model.save('D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/model_comb4.h5')
```

Figure 16: CNN Combination4

```
# Combination 5
model = getCNNModel(num_of_layers=3, num_of_filters=32, filter_size=(5, 5), initializer='glorot_uniform',
                    activation_function='relu', dropout=0.2, opt='adam')
model = trainModelAndGetConfusionMatrix(model)
model.save('D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/model_comb5.h5')
```

Figure 17: CNN Combination5

```
# Combination 6
model = getCNNModel(num_of_layers=3, num_of_filters=32, filter_size=(3, 3), initializer='glorot_uniform',
                    activation_function='relu', dropout=0.7, opt='adam')
model = trainModelAndGetConfusionMatrix(model)
model.save('D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/model_comb6.h5')
```

Figure 18: CNN Combination6

Once all the experiments with adjusting different parameters was done. A function was written to compare all CNN models as shown in the Figure 19

```
combinations = list(data.keys())
accuracies = list(data.values())

fig = plt.figure(figsize = (20, 10))

# creating the bar plot
plt.bar(combinations, accuracies, color='#057D9F', width=0.5)

plt.xlabel("Combination of CNN")
plt.ylabel("Test Accuracy")
plt.title("Test Accuracies of Some Combinations of CNN Model")
plt.show()
```

Figure 19: CNN Models Comparison code

5.2 Experiment with Transfer Learning Models

The experiment with CNN models were not reliable for carrying out image retrieval. Thus, transfer learning technique was taken into consideration. So, VGG16 and ResNet50 models were developed as shown in the Figure 20 and 21

```
# Get VGG-16 Model
def getVGG16Model(lastFourTrainable=False):
  vgg_model = VGG16(weights='imagenet', input_shape=input_shape, include_top=True)

  # Make all layers untrainable
  for layer in vgg_model.layers[:]:
      layer.trainable = False

  # Add fully connected layer which have 1024 neuron to VGG-16 model
  output = vgg_model.get_layer('fc2').output
  output = Flatten(name='new_flatten')(output)
  output = Dense(units=1024, activation='relu', name='new_fc')(output)
  output = Dense(units=10, activation='softmax')(output)
  vgg_model = Model(vgg_model.input, output)

  # Make last 4 layers trainable if lastFourTrainable == True
  if lastFourTrainable == True:
    vgg_model.get_layer('block5_conv3').trainable = True
    vgg_model.get_layer('fc1').trainable = True
    vgg_model.get_layer('fc2').trainable = True
    vgg_model.get_layer('new_fc').trainable = True

  # Compile VGG-16 model
  vgg_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
  vgg_model.summary()

  return vgg_model
```

Figure 20: VGG16 Model code

```
# Get ResNet-50 Model
def getResNet50Model(lastFourTrainable=False):
  resnet_model = ResNet50(weights='imagenet', input_shape=input_shape, include_top=True)

  # Make all layers non-trainable
  for layer in resnet_model.layers[:]:
      layer.trainable = False

  # Add fully connected layer which have 1024 neuron to ResNet-50 model
  output = resnet_model.get_layer('avg_pool').output
  output = Flatten(name='new_flatten')(output)
  output = Dense(units=1024, activation='relu', name='new_fc')(output)
  output = Dense(units=10, activation='softmax')(output)
  resnet_model = Model(resnet_model.input, output)

  # Make last 4 layers trainable if lastFourTrainable == True
  if lastFourTrainable == True:
    resnet_model.get_layer('conv5_block3_2_bn').trainable = True
    resnet_model.get_layer('conv5_block3_3_conv').trainable = True
    resnet_model.get_layer('conv5_block3_3_bn').trainable = True
    resnet_model.get_layer('new_fc').trainable = True

  # Compile ResNet-50 model
  resnet_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
  resnet_model.summary()

  return resnet_model
```

Figure 21: ResNet50 Model code

Further, these model's last four layer trainable parameter was adjusted either true or false and results were recorded as well as compared with evaluation methods.

```
# Get ResNet-50 Model with LastFourTrainable=False
resnet_model_a = getResNet50Model(lastFourTrainable=False)
# Train ResNet-50 Model and get Confusion Matrix
resnet_model_a = trainModelAndGetConfusionMatrix(resnet_model_a)
resnet_model_a.save_weights('D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/model_resnet_nontrainable.h5')
```

Figure 22: ResNet50 Model code
with last four layer trainable as False

```
# Get ResNet-50 Model with LastFourTrainable=True
resnet_model_b = getResNet50Model(lastFourTrainable=True)
# Train ResNet-50 Model and get Confusion Matrix
resnet_model_b = trainModelAndGetConfusionMatrix(resnet_model_b)
resnet_model_b.save_weights('D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/model_resnet_trainable.h5')
```

Figure 23: ResNet50 Model code
with last four layer trainable as True

```
# Get VGG-16 Model with LastFourTrainable=False
vgg_model_a = getVGG16Model(lastFourTrainable=False)
# Train VGG-16 Model and get Confusion Matrix
vgg_model_a = trainModelAndGetConfusionMatrix(vgg_model_a)
vgg_model_a.save_weights('D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/model_vgg_nontrainable.h5')
```

Figure 24: VGG16 Model code
with last four layer trainable as False

```
# Get VGG-16 Model with LastFourTrainable=True
vgg_model_b = getVGG16Model(lastFourTrainable=True)
# Train VGG-16 Model and get Confusion Matrix
vgg_model_b = trainModelAndGetConfusionMatrix(vgg_model_b)
vgg_model_b.save_weights('D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/model_vgg_trainable.h5')
```

Figure 25: VGG16 Model code
with last four layer trainable as True

Once, all the models were build and experimented with adjusting the last layer trainable. A bar chart was plotted and the results were compared as shown in the Figure 26

```
combinations = list(data.keys())
accuracies = list(data.values())

fig = plt.figure(figsize = (15, 10))

# creating the bar plot
plt.bar(combinations, accuracies, color='#FFCA00', width=0.5)

plt.xlabel("Transfer Learning Model")
plt.ylabel("Test Accuracy")
plt.title("Test Accuracies of Transfer Learning Models")
plt.show()
```

Figure 26: Comparison Code for VGG16 and Resnet50

5.3 Experiment with Feature Extraction and Image Retrieval

Based the experiments carried in the above sections, VGG16 gave best performing results. Using pre-trained VGG16 model, image retrieval experiment was carried out. Features of the image were extracted as well as similarity between images were checked as shown in the Figure 27

```
# Get feature vector of an image by given model and img_path
def getFeatureVector(model, img_path):
  img = cv2.imread(img_path)
  img = cv2.resize(img, (224, 224))
  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
  feature_vector = model.predict(img.reshape(1, 224, 224, 3))

  print(img_path + " is added.")
  return feature_vector

# Get cosine similarity between feature vectors A and B using cosine similarity
def getCosineSimilarity(A, B):
  cos_similarity = np.dot(A,B.T) / (np.linalg.norm(A)*np.linalg.norm(B)) # Get cosine similarity
  return cos_similarity[0][0]
```

Figure 27: Function for extracting features and getting similarity

```
# Function for get dataframe which contains the output features of given model
def getFeatureDataFrame(model):
  df = pd.DataFrame(columns=['file', 'features'])
  train_files = train_generator.filepaths
  valid_files = valid_generator.filepaths

  files = train_files + valid_files
  print(len(files))

  df['file'] = files
  df['features'] = df.apply(lambda row: getFeatureVector(model, row['file']), axis=1)

  print("All files added.")
  return df
```

Figure 28: Function for feature vector dataframe

The model which was trained earlier with last four layer = false was used in order to load weights and extract features as shown in the Figure 29

```
# Get feature extractor model from last layer of vgg_model_a
vgg_model_a = getVGG16Model(lastFourTrainable=False)
vgg_model_a.load_weights('D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/model_vgg_nontrainable.h5')
feature_model_vgg_a = Model(inputs=vgg_model_a.input, outputs=vgg_model_a.get_layer('new_fc').output)

df = getFeatureDataFrame(feature_model_vgg_a)
df.to_pickle("D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/features_vgg_a.pickle")
```

Figure 29: Feature extractor VGG Model(a)

The model which was trained earlier with last four layer = true was used in order to load weights and extract features as shown in the Figure 30

```
# Get feature extractor model from last layer of vgg_model_b
vgg_model_b = getVGG16Model(lastFourTrainable=True)
vgg_model_b.load_weights('D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/model_vgg_trainable.h5')
feature_model_vgg_b = Model(inputs=vgg_model_b.input, outputs=vgg_model_b.get_layer('new_fc').output)

df = getFeatureDataFrame(feature_model_vgg_b)
df.to_pickle("D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/features_vgg_b.pickle")
```

Figure 30: Feature extractor VGG Model(b)

```
# Plot similar 5 images with given image and similar images dataframe
def plotSimilarImages(img_file, similar_df, model_name):
  img = cv2.imread(img_file)
  img = cv2.resize(img, (224, 224))
  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

  split_list = img_file.split('/')
  split_list.reverse()
  img_class = split_list[1]
  fig, axarr = plt.subplots(2,3)
  axarr[0,0].imshow(img)
  axarr[0,0].set_title("TEST IMAGE - " + model_name + "\nClass: " + img_class)
  axarr[0,0].axis('off')

  j, k, m = 0, 0, 1
  for index, sim in similar_df.iterrows():
    filepath = sim['file']
    similarity = sim['similarity']
    split_list = filepath.split('/')
    split_list.reverse()
    sim_class = split_list[1]

    similar = cv2.imread(filepath)
    similar = cv2.resize(similar, (224, 224))
    similar = cv2.cvtColor(similar, cv2.COLOR_BGR2RGB)
    axarr[k,m].imshow(similar)
    axarr[k,m].set_title("Similarity: %.3f" % similarity + "\nClass: " + sim_class)
    axarr[k,m].axis('off')

    m += 1
    if m == 3 and k != 1:
      k += 1
      m = 0

    j += 1
    if j == 5:
      break

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

Figure 31: Function for plotting similar images

```
# Get and plot 5 similar images for given image path and features dataframe
def getSimilarImages(img_file, features_df, model, model_name):
  img_features = getFeatureVector(model, img_file)
  features_df['similarity'] = features_df.apply(lambda row: getCosineSimilarity(img_features, np.asarray(row['features'])),
                                                axis=1)
  sorted_df = features_df.sort_values(by='similarity', ascending=False)

  plotSimilarImages(img_file, sorted_df.head(5), model_name)
```

Figure 32: Function for getting Similarity between images

```
# Test images path
feature_test_path = 'D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/test'
feature_test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(directory=feature_test_path,
                                                                                 class_mode='categorical',
                                                                                 batch_size=batch_size,
                                                                                 target_size=(img_rows, img_cols),
                                                                                 color_mode="rgb",
                                                                                 shuffle=False)
feature_test_files = feature_test_generator.filepaths

Found 1327 images belonging to 10 classes.
```

Figure 33: Giving Test Images Path

```python
# Get similar images of test images for VGG-16 (a)
vgg_model_a = getVGG16Model(lastFourTrainable=False)
vgg_model_a.load_weights('D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/model_vgg_nontrainable.h5')
feature_model_vgg_a = Model(inputs=vgg_model_a.input, outputs=vgg_model_a.get_layer('new_fc').output)

df = pd.read_pickle('D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/features_vgg_a.pickle')
for file in feature_test_files:
  getSimilarImages(file, df, feature_model_vgg_a, 'VGG-16 (a)')
```

Figure 34: Getting Similar Images and Similarity with VGG16(a) Model

```python
# Get similar images of test images for VGG-16 (b)
vgg_model_b = getVGG16Model(lastFourTrainable=True)
vgg_model_b.load_weights('D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/model_vgg_trainable.h5')
feature_model_vgg_b = Model(inputs=vgg_model_b.input, outputs=vgg_model_b.get_layer('new_fc').output)

df = pd.read_pickle('D:/OIDv4_ToolKit-master/OIDv4_ToolKit-master/OID/Dataset/features_vgg_b.pickle')
for file in feature_test_files:
  getSimilarImages(file, df, feature_model_vgg_b, 'VGG-16 (b)')
```

Figure 35: Getting Similar Images and Similarity with VGG16(b) Model

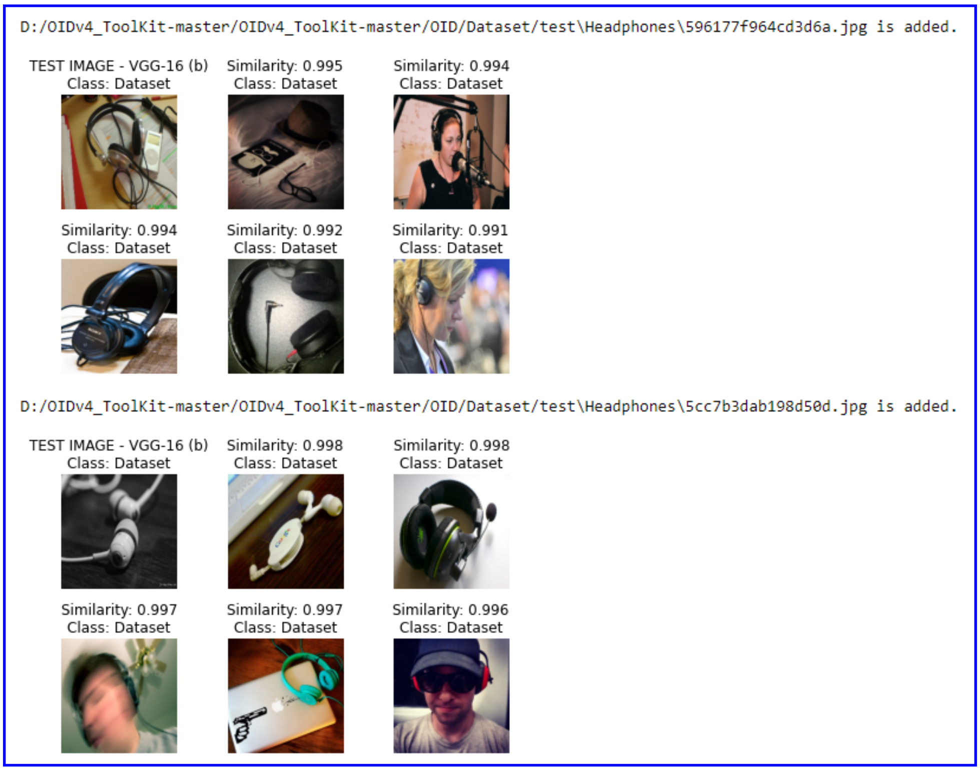

Figure 36: Example of result with VGG16(b) Model with Headphones Class

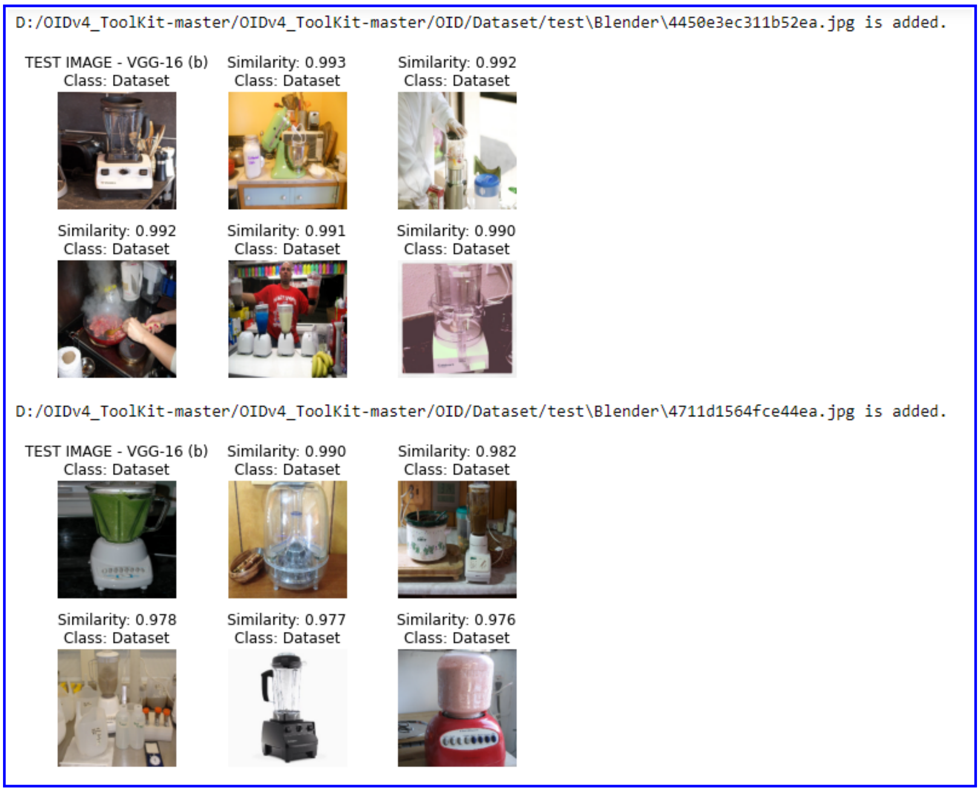

Figure 37: Example of result with VGG16(b) Model with Blender class