Assignment 1:

Introduction to Deep Convolutional Neural Networks

(Human Action Recognition)

Stanford 40 Actions Dataset

COSC2972 Deep Learning

(Undergraduate Level)

By Oisin Sol Emlyn Aeonn

Student ID: s3952320

1.0 Table of Contents

NOTE: Complementary Sections are denoted with *

By default we have hidden these Cells please click the header to see hidden sections.

1.1 Read Me

- This notebook was developed using AWS SageMaker, the industry standard at the Australian Bureau of Statistics where I work. While all code has been double-tested on Google Colab, please note that you may need to adjust directory paths and other attributes to run it in your environment. If you encounter any issues, please don't hesitate to reach out.

- All tasks have been meticulously completed in accordance with the Assignment 1 Brief.

- The Discussion Forum, Tutorials, and Lectures were thoroughly utilized to gain a comprehensive understanding of the assignment requirements. Recognizing some ambiguity in the phrasing, I approached this as a real-world problem with potential significant implications. For instance, if this system were used to classify various human actions (such as walking dogs, repairing cars, or blowing bubbles), a misclassification could lead to serious consequences. These might include:
  - Compromised safety in workplace environments (e.g., misclassifying a dangerous action as safe)
  - Incorrect allocation of resources in urban planning (e.g., misinterpreting common activities in public spaces)
  - Inaccurate content recommendation or moderation on social media platforms
  - Errors in surveillance systems for security purposes

  Such errors underscore the critical importance of developing robust and accurate classification models for human actions using ethical and explainable systems.

- Sections marked as Complementary (denoted with *) employ methods outside the scope of Assignment 1. These were included to further enhance the models, provide comparative analysis, and for personal learning. These sections are hidden by default to ensure easy navigation to the core parts of the code for assessment purposes.

- Predictions are dynamically generated within this Jupyter Notebook. Some cells are optional, and running them may yield different results. For optimal results, run all cells except those marked with "⚠".

- This project adheres to RMIT University's Academic Integrity Policy and was completed solely by Oisin Sol Emlyn Aeonn (s3952320) within the specified timeframe. All materials used are properly cited and referenced.

- The project was submitted on time to the COSC2972 (Undergraduate) Deep Learning Canvas before Friday 5:00 PM, August 30, 2024.

- As someone deeply passionate about Deep Learning and aspiring to build a career in this field, I welcome any constructive feedback to further enhance my learning and application of these concepts.

- Special acknowledgments to:
  - Current Course (Deep Learning COSC2972):
    - Professor Ruwan Tennakoon
    - Mr. Zeji Hui (Ph.D)
    - Ms. Kavindya Imbulgoda (Ph.D)
  - Previous Course (Machine Learning COSC2673):
    - Professor Azadeh Alavi
    - Dr. Pubudu Sanjeevani
    - Ms. Rumin Chu (Ph.D)
    - Ms. Hiqmat Nisa (Ph.D)

- I sincerely hope you find my work engaging and insightful!

1.2 Introduction

- **Problem Statement:**
  - Assessment 1 entails creating a supervised Deep Learning model that can Recognize Human Actions (HAR). The dataset we are using is the Stanford 40 Actions Dataset, which comprises 40 classes of actions, a high-level class, and a binary attribute indicating whether the image has one or more people. Our task is to create a single model that can predict both the action class and the presence of multiple people. Although not required, I decided to also predict the HighLevelCategory as there was little reason not to, although it was not a result I tried to maximize unlike the other two attributes. It's worth noting that the MoreThanOnePerson attribute, when true, can serve as a source of uncertainty as it tends to confuse models. To achieve this goal, we will conduct the following tasks:
    - Extensive EDA
    - Removal of Bad Data
    - Ingesting the Training Data
    - Prepare it for Training by Assigning Labels to the Dataset
    - Splitting the Data into Training, & Validation
    - Checking for Leaks in the Datasets
    - Training Models using different Algorithms, and Techniques
    - Evaluation using Classification Metrics
    - Hyperparameter Tuning, and Optimizing Model Performance
    - Collecting Independent Evaluation Data using the Scientific Method
    - Testing Generalizability of the Model on our Independently Sourced Evaluation Dataset
    - Writing a Report, and summarizing our key findings
  - We will start by exploring neural network-based models, focusing on a baseline Multi-Layer Perceptron (MLP). We will then utilize subsequent optimizations found in Convolutional Neural Networks (CNN). These optimizations will be pursued through advanced Machine Learning Techniques, including Hyperparameter Tuning such as applying Regularisation, Batch Tuning, and other methods. We will also focus on improving our Neural Network Layers. Finally, we will apply some Transfer Learning Techniques to potentially improve our model's performance and generalizability. It's worth noting that I initially explored simple, non-neural network models such as logistic regression and tree-based algorithms, but these were removed from the notebook as per the assignment requirements. However, these simpler models did not perform as well as even the baseline MLP, demonstrating the dominance of Neural Network based models for Computer Vision tasks.
  - For additional evaluation, I conducted a small amount of web scraping and took a few of my own photos to test on the model. This independently sourced evaluation dataset was used to assess the model's performance before testing on the actual test data split.
- **Dataset:**
  - The analysis is based on the Stanford 40 Actions Dataset, with a pre-made split. This dataset encompasses 40 actions ranging from blowing bubbles and walking dogs to repairing cars. The training and validation data is contained in 'train_data_2024.csv', while the unseen test data is in 'future_data_2024.csv'.
- **Objectives:**
  - Discuss and critically analyse a variety of neural network architectures; Evaluate and Compare approaches and algorithms on the basis of the nature of the problem/task being addressed.
  - Synthesise suitable solutions to address particular machine learning problems based on analysis of the problem and characteristics of the data involved.
  - Communicate effectively with a variety of audiences through a range of modes and media, in particular to: interpret abstract theoretical propositions, choose methodologies, justify conclusions and defend professional decisions to both IT and non-IT personnel via technical reports of professional standard and technical presentations.
  - Develop skills for further self-directed learning in the general context of neural networks and machine learning; Research, Discuss, and Use new and novel algorithms for solving problems; Adapt experience and knowledge to and from other computer sciences contexts such as artificial intelligence, machine learning, and software design.
- **Scope:**
  - Focus on predicting the correct Human Action and presence of multiple people, leveraging methods and insights from weeks 1-6 of the COSC2972 Deep Learning Course.
  - Adopt strategies such as regularization and normalization to refine the model, maintaining the integrity of the feature set.
  - Strive for the optimal model performance within the established constraints.
  - Undertake predictions on a separate, unseen dataset ('future_data_2024.csv') to validate the model's generalisability.

1.3 Python Library Imports

- First, let's cover all of the required imports for this Jupyter Notebook.
- I collated them all here so that you can run most cells (especially EDA, but some Keras / Tensorflow, and Evaluation) out of order.
- This conforms to the coding standard DO NOT REPEAT YOURSELF (DRY).
- I have also divided the libraries into sections, and provided a commented description of each library for their use.

In [1]:
```
# Tensorflow
import tensorflow as tf
import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Input, BatchNormalization, GlobalAverageP
```

```
from keras.regularizers import l2
from keras.applications import VGG16, MobileNetV2
from keras.utils import load_img, img_to_array
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.applications.resnet import ResNet50

# Numpy
import numpy as np
np.object = np.object_

# Operating system interfaces
import os

# Data manipulation and analysis
import pandas as pd
from pandas.plotting import scatter_matrix
import math

# Model selection and evaluation
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_score, precision_score, recall_score, f1_scor
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.decomposition import PCA
from imblearn.over_sampling import RandomOverSampler

# Image processing
from PIL import Image
import imageio
from skimage import transform, io, color, filters, measure
from skimage.color import rgb2gray, gray2rgb
from scipy.spatial.distance import euclidean

# Plotting and visualization
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Debug
print(os.environ['PATH'])
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
print(tf.config.list_physical_devices('GPU'))

# Show TensorFlow version
print("TensorFlow Version:", tf.__version__)

# Plotting style
plt.style.use('dark_background')
%matplotlib inline

from sklearn.manifold import TSNE
import gc
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.manifold import TSNE
from scipy.spatial.distance import pdist, squareform
import gc
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.manifold import TSNE
from scipy.spatial.distance import pdist, squareform
import gc
```

```
/opt/anaconda/bin:/opt/anaconda/condabin:/usr/local/sbin:/usr/local/bin:/usr/bin:/var/lib/flatpak/exports/bin:/usr/lib/jvm/d
efault/bin:/usr/bin/site_perl:/usr/bin/vendor_perl:/usr/bin/core_perl:/var/lib/snapd/snap/bin
Num GPUs Available:  0
[]
TensorFlow Version: 2.12.0
```

2.0 Exploratory Data Analysis (EDA)

- Reason for performing EDA:
  - To gain insights into the characteristics and properties of the human action images.

    - To identify any patterns, trends, or anomalies in the dataset.
    - To understand the distribution and variability of different features extracted from the images.
    - To assess the quality and consistency of the image data.
    - To inform further steps in the analysis or modeling process.
    - To identify potential biases or limitations in the dataset that could affect model performance and generalizability.

- What we are looking for:

    - Image sizes and formats to ensure consistency and compatibility.
    - Image sharpness to identify any blurry or low-quality images.
    - Average pixel intensity to understand the overall brightness or darkness of the images.
    - Image entropy to measure the amount of information or complexity in the images.
    - Image similarities to identify duplicate or highly similar images.
    - Whether the dataset has been augmented already.
    - Image orientation (portrait or landscape) and whether only a small portion contains relevant information (person, action).
    - Accuracy of labeling in the training/validation split to ensure model training accuracy and reliability of validation metrics.

    - Domain-specific features relevant to human action recognition:

        - Background context (indoor vs. outdoor, urban vs. rural, etc.)
        - Body position and orientation of the person(s) in the image
        - Weather conditions in outdoor images
        - Clothing styles and their correlation with specific actions
        - Age distribution of people in the images
        - Racial and ethnic diversity representation

    - Class-specific characteristics:

        - Whether certain classes are predominantly indoor or outdoor
        - Consistent clothing or accessories associated with specific actions (e.g., overalls for car repair, aprons for dishwashing)

    - Potential biases in the dataset:

        - Underrepresentation of certain racial or ethnic groups
        - Age or gender imbalances in specific action categories
        - Cultural or geographic biases in action representations

- Importance of thorough EDA:

    - Ensures transparency about the dataset's limitations and potential biases.
    - Helps identify if the model would be suitable for deployment in diverse communities.
    - Informs decisions on using techniques like transfer learning, fine-tuning, or oversampling to improve model generalization.
    - Allows for better communication with stakeholders about the model's capabilities and limitations.
    - Guides the development of a more robust and fair model for human action recognition.
    - Validates the integrity of the training and validation data splits, ensuring reliable model evaluation and performance metrics.

2.1 Data Ingestion

In [2]:
```
# Load the labels from the CSV file
labels_df = pd.read_csv('train_data_2024.csv')

# Specify the folder containing images
image_folder = 'Images'

# Add a column to the DataFrame with the full path to each image
labels_df['FileName'] = 'Images/' + labels_df['FileName']

# Display the dataframe to ensure it loaded correctly
labels_df.head()
```

Out[2]:

	FileName	Class	MoreThanOnePerson	HighLevelCategory
0	Images/Img_460.jpg	blowing_bubbles	YES	Social_LeisureActivities
1	Images/Img_8152.jpg	blowing_bubbles	YES	Social_LeisureActivities
2	Images/Img_9056.jpg	jumping	YES	Sports_Recreation
3	Images/Img_3880.jpg	pushing_a_cart	YES	OutdoorActivities
4	Images/Img_3168.jpg	writing_on_a_book	NO	Artistic_MusicalActivities

- Above we just amended the 'FileName' column in the dataframe to include the 'Images/' prefix, ensuring proper ingestion of image files located in the 'Images' folder.

In [3]:
```
# Function to get image size
def get_image_size(filename):
    with Image.open(filename) as img:
        return img.size

# Function to get image format
def get_image_format(filename):
    with Image.open(filename) as img:
        return img.format

# Function to get image sharpness
def get_image_sharpness(filename):
```

```
    with Image.open(filename) as img:
        img_array = np.array(img.convert('L'))
        return img_array.std()

# Function to get pixel intensities
def get_pixel_intensity(filename):
    with Image.open(filename) as img:
        img_array = np.array(img.convert('L'))
        return img_array.mean()

# Function to get entropy
def get_image_entropy(filename):
    with Image.open(filename) as img:
        img_array = np.array(img.convert('L')).flatten()
        hist, _ = np.histogram(img_array, bins=256, range=(0, 256), density=True)
        hist = hist[hist > 0]
        return -np.sum(hist * np.log2(hist))

# Function to get number of channels
def get_image_channels(filename):
    try:
        with Image.open(filename) as img:
            return len(img.getbands())
    except Exception as e:
        print(f"Error processing {filename}: {str(e)}")
        return None

# Add image metadata to the DataFrame
labels_df['Size'] = labels_df['FileName'].apply(get_image_size)
labels_df['Format'] = labels_df['FileName'].apply(get_image_format)
labels_df['Sharpness'] = labels_df['FileName'].apply(get_image_sharpness)
labels_df['PixelIntensity'] = labels_df['FileName'].apply(get_pixel_intensity)
labels_df['Entropy'] = labels_df['FileName'].apply(get_image_entropy)
labels_df['Channels'] = labels_df['FileName'].apply(get_image_channels)

# Extract width and height from Size column
labels_df['Width'] = labels_df['Size'].apply(lambda x: x[0])
labels_df['Height'] = labels_df['Size'].apply(lambda x: x[1])

# Display summary statistics
labels_df.describe()
```

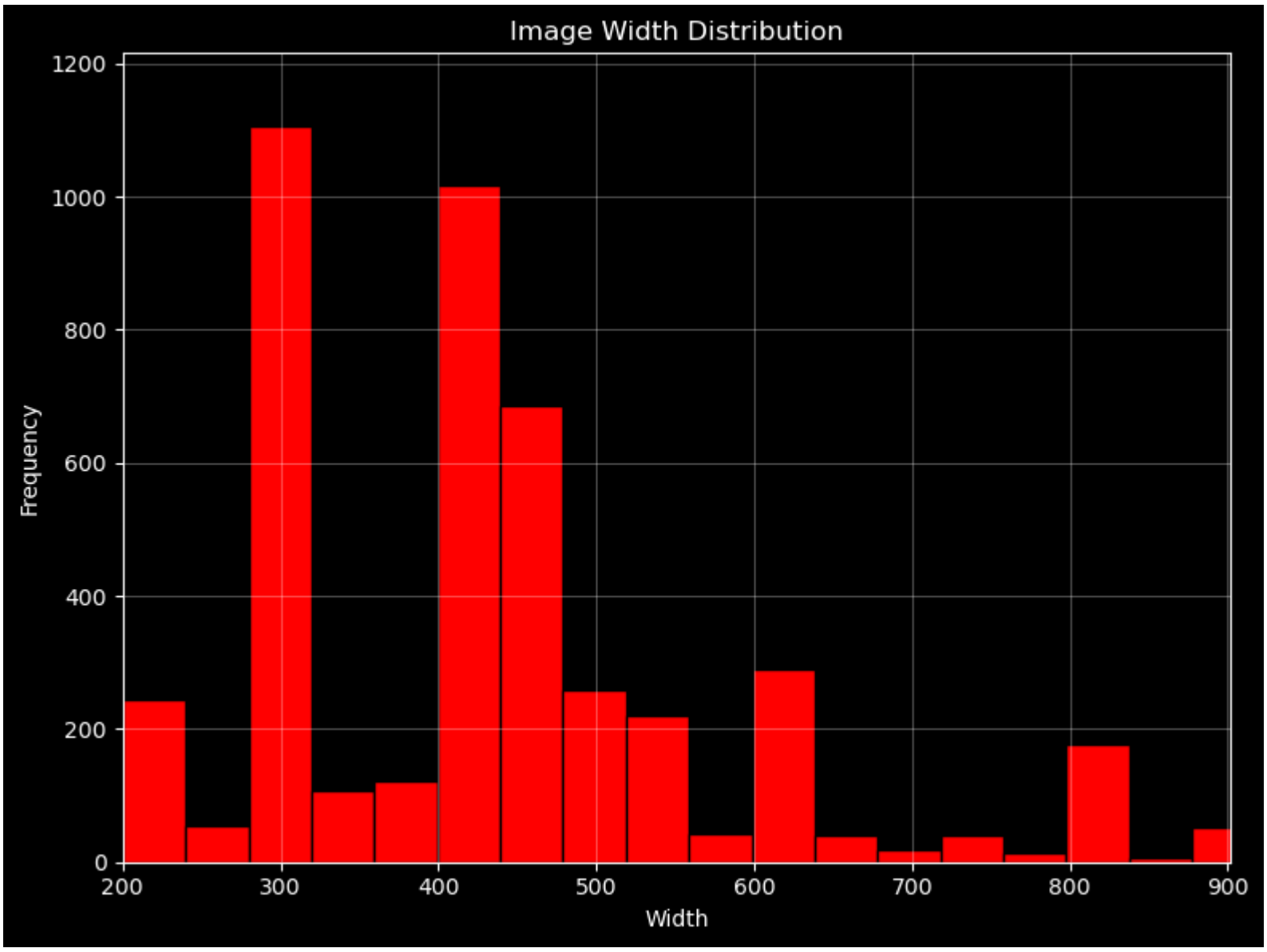
Out[3]:

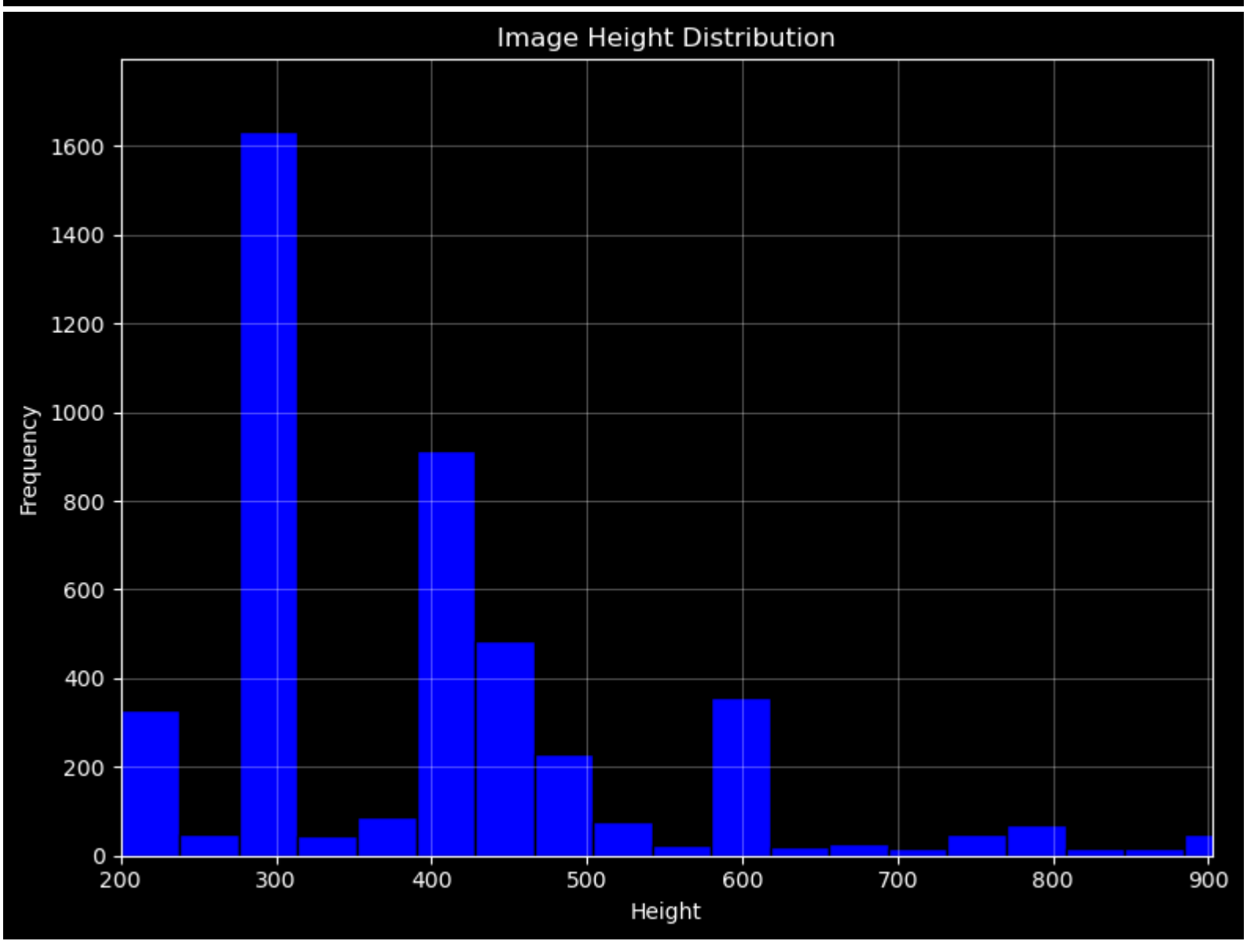
	Sharpness	PixelIntensity	Entropy	Channels	Width	Height
count	4500.000000	4500.000000	4500.000000	4500.0	4500.000000	4500.000000
mean	60.268131	112.845794	7.435299	3.0	427.937111	399.291556
std	12.484330	28.995262	0.390291	0.0	146.506065	143.577378
min	19.613698	21.012003	3.776446	3.0	200.000000	200.000000
25%	52.002366	94.713968	7.289804	3.0	300.000000	300.000000
50%	60.296694	112.730599	7.530868	3.0	400.000000	400.000000
75%	68.423923	131.255759	7.698810	3.0	488.750000	451.000000
max	101.867240	220.903686	7.962339	3.0	997.000000	960.000000

In [5]:

```
# Function to plot histogram with consistent settings
def plot_histogram(data, column, color, title):
    plt.figure(figsize=(8, 6))
    plt.hist(data[column], bins=20, color=color, edgecolor='black')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.title(f'{title} Distribution')
    plt.grid(True, alpha=0.3)

    # Set x-axis limits to cover 99% of the data
    x_min, x_max = np.percentile(data[column], [0.5, 99.5])
    plt.xlim(x_min, x_max)

    # Set y-axis limits
    counts, _ = np.histogram(data[column], bins=20)
    plt.ylim(0, max(counts) * 1.1)

    plt.tight_layout()
    plt.show()
    plt.close()
    gc.collect()

# Plot histograms
plot_histogram(labels_df, 'Width', 'red', 'Image Width')
plot_histogram(labels_df, 'Height', 'blue', 'Image Height')
plot_histogram(labels_df, 'Sharpness', 'purple', 'Image Sharpness')
plot_histogram(labels_df, 'PixelIntensity', 'green', 'Pixel Intensity')
plot_histogram(labels_df, 'Entropy', 'orange', 'Entropy')

# Plot image format distribution separately
plt.figure(figsize=(8, 6))
labels_df['Format'].value_counts().plot(kind='bar', color='purple')
plt.xlabel('Image Format')
plt.ylabel('Frequency')
```

```
plt.title('Image Format Distribution')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
plt.close()
gc.collect()

# Plot channels distribution
plt.figure(figsize=(10, 6))
labels_df['Channels'].value_counts().sort_index().plot(kind='bar', color='teal')
plt.xlabel('Number of Channels')
plt.ylabel('Frequency')
plt.title('Image Channels Distribution')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
plt.close()
gc.collect()
```





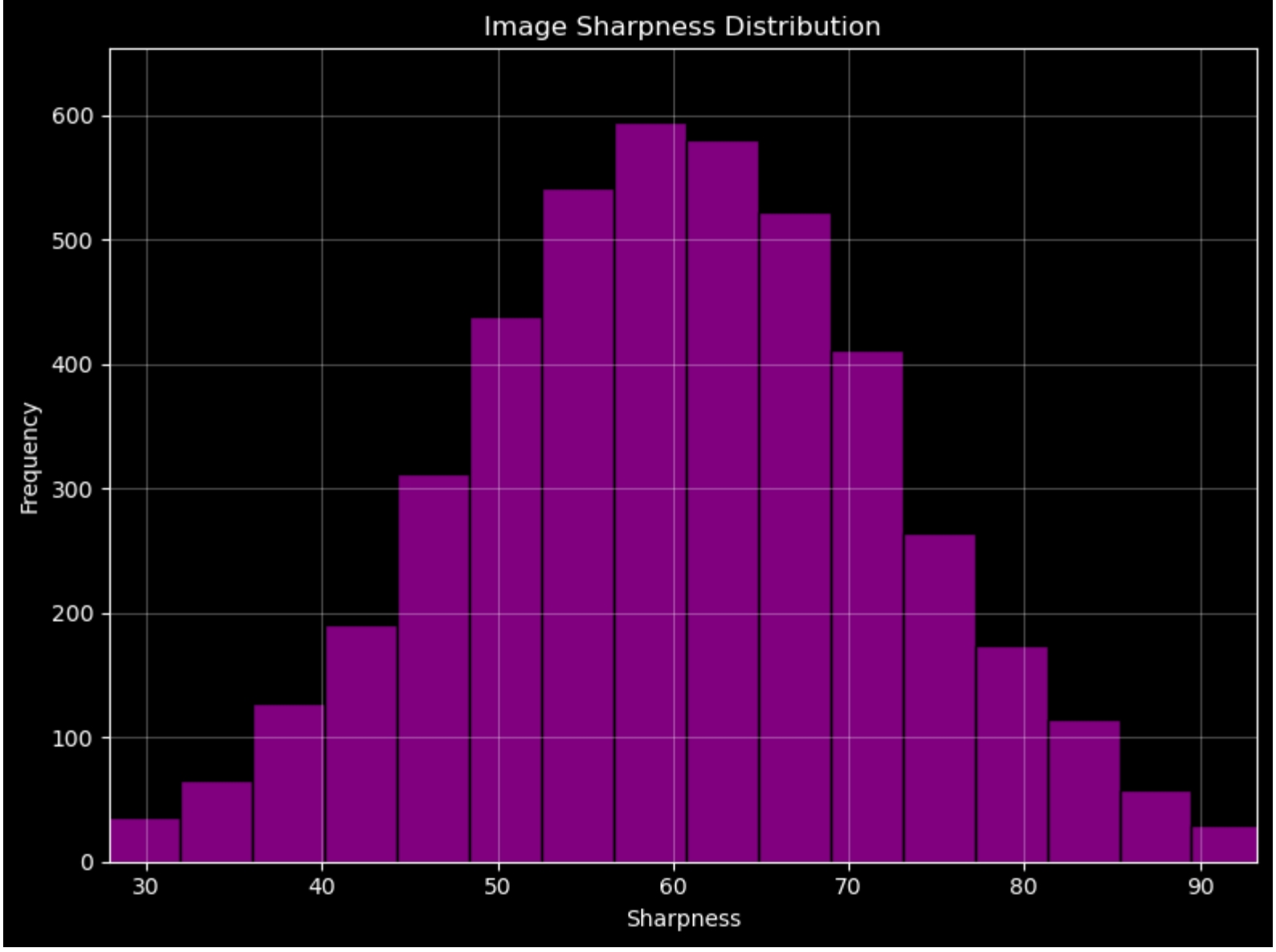
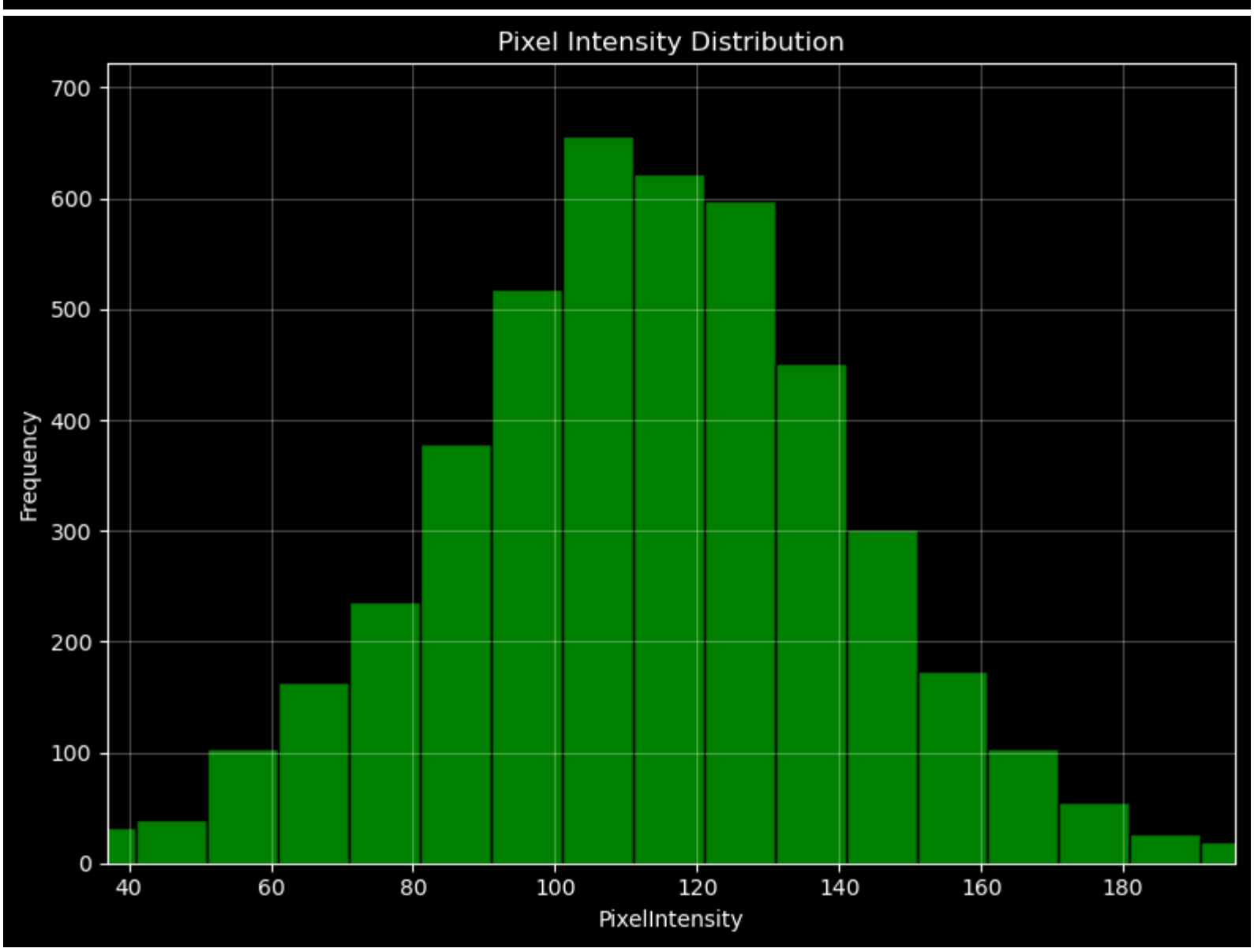
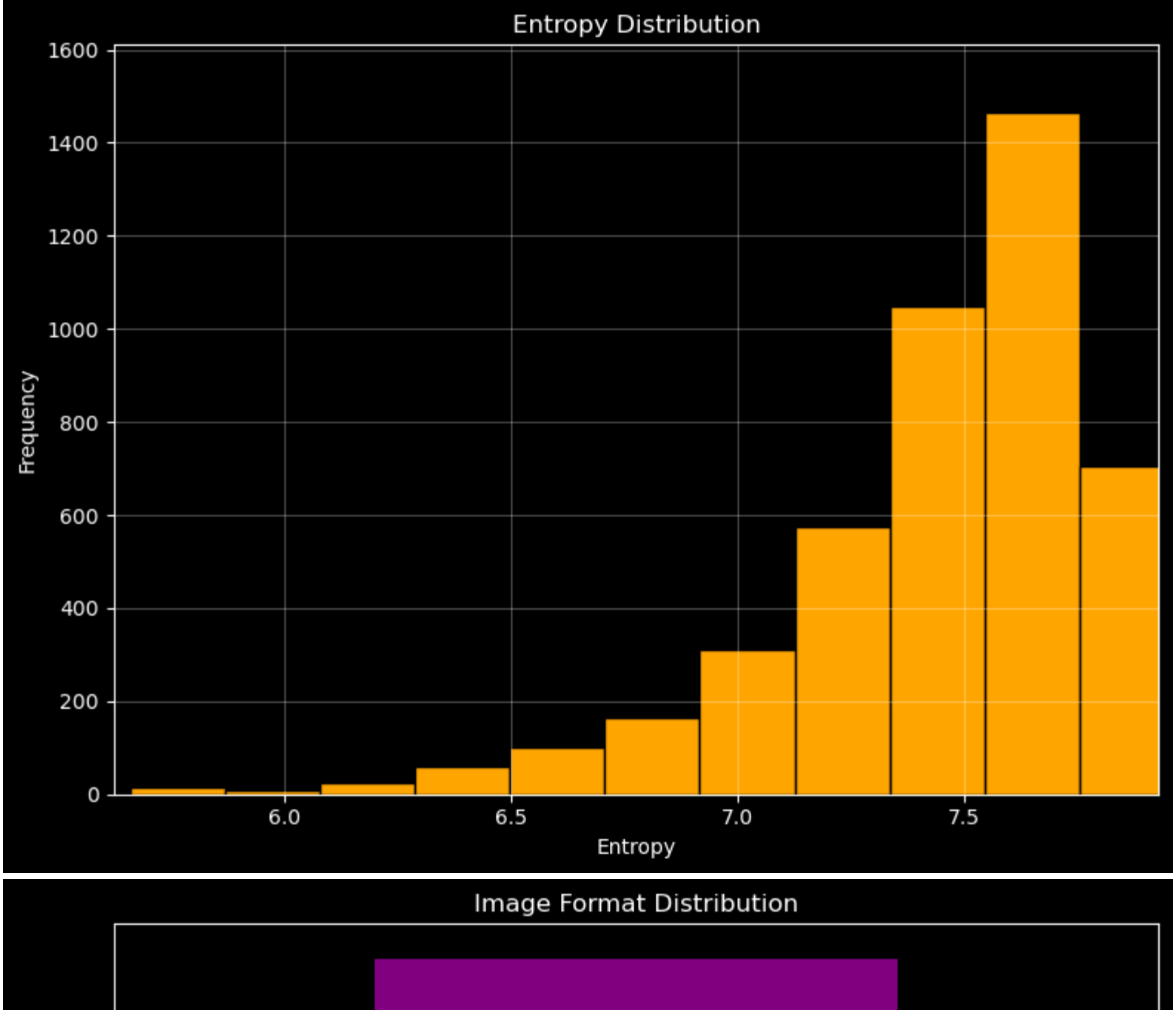


Image Sharpness Distribution
Frequency
0
100
200
300
400
500
600
30
40
50
60
70
80
90
Sharpness


Pixel Intensity Distribution
Frequency
0
100
200
300
400
500
600
700
40
60
80
100
120
140
160
180
PixelIntensity


Entropy Distribution
Frequency
1600
1400
1200
1000
800
600
400
200
0
6.0
6.5
7.0
7.5
Entropy

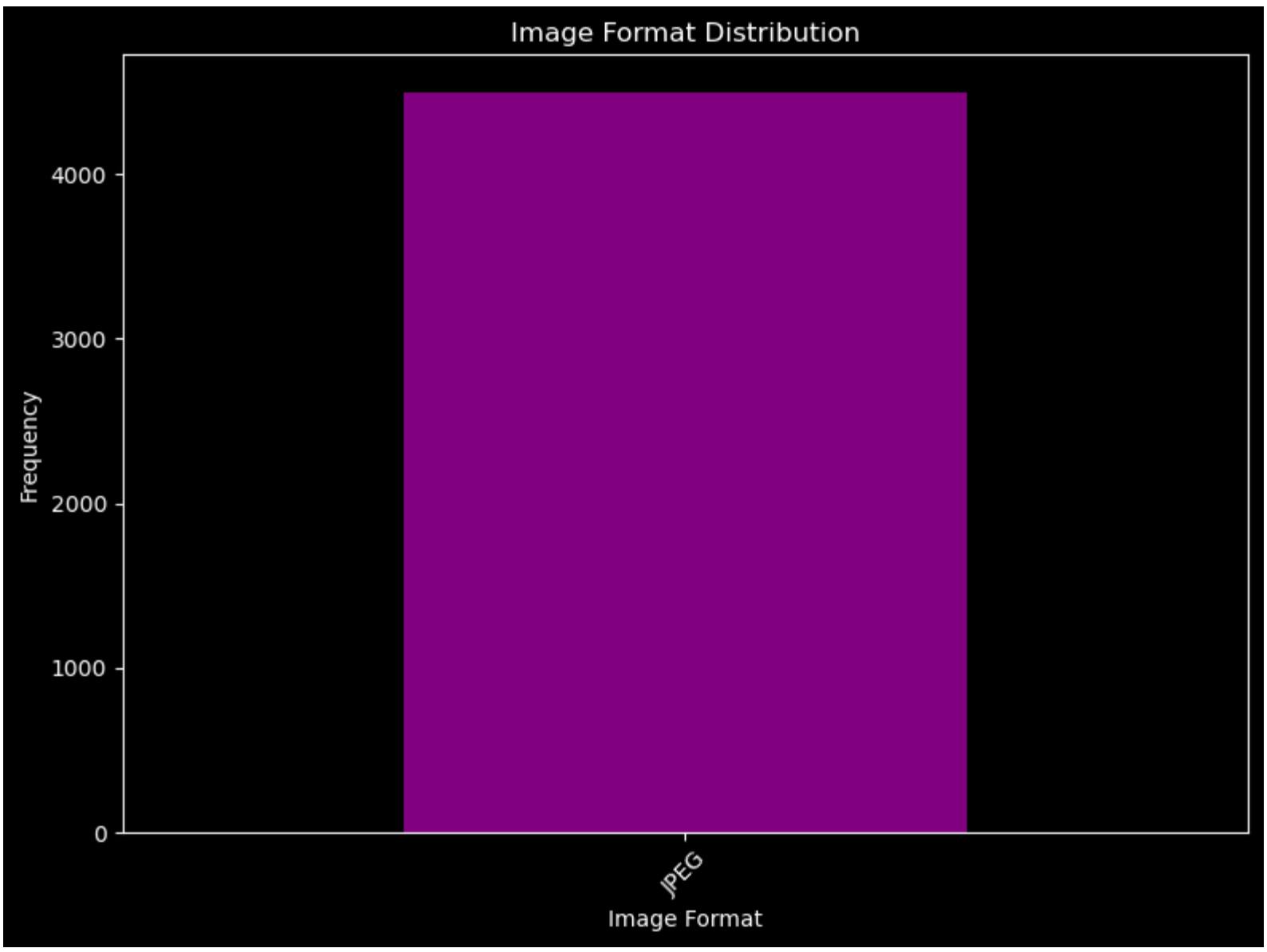
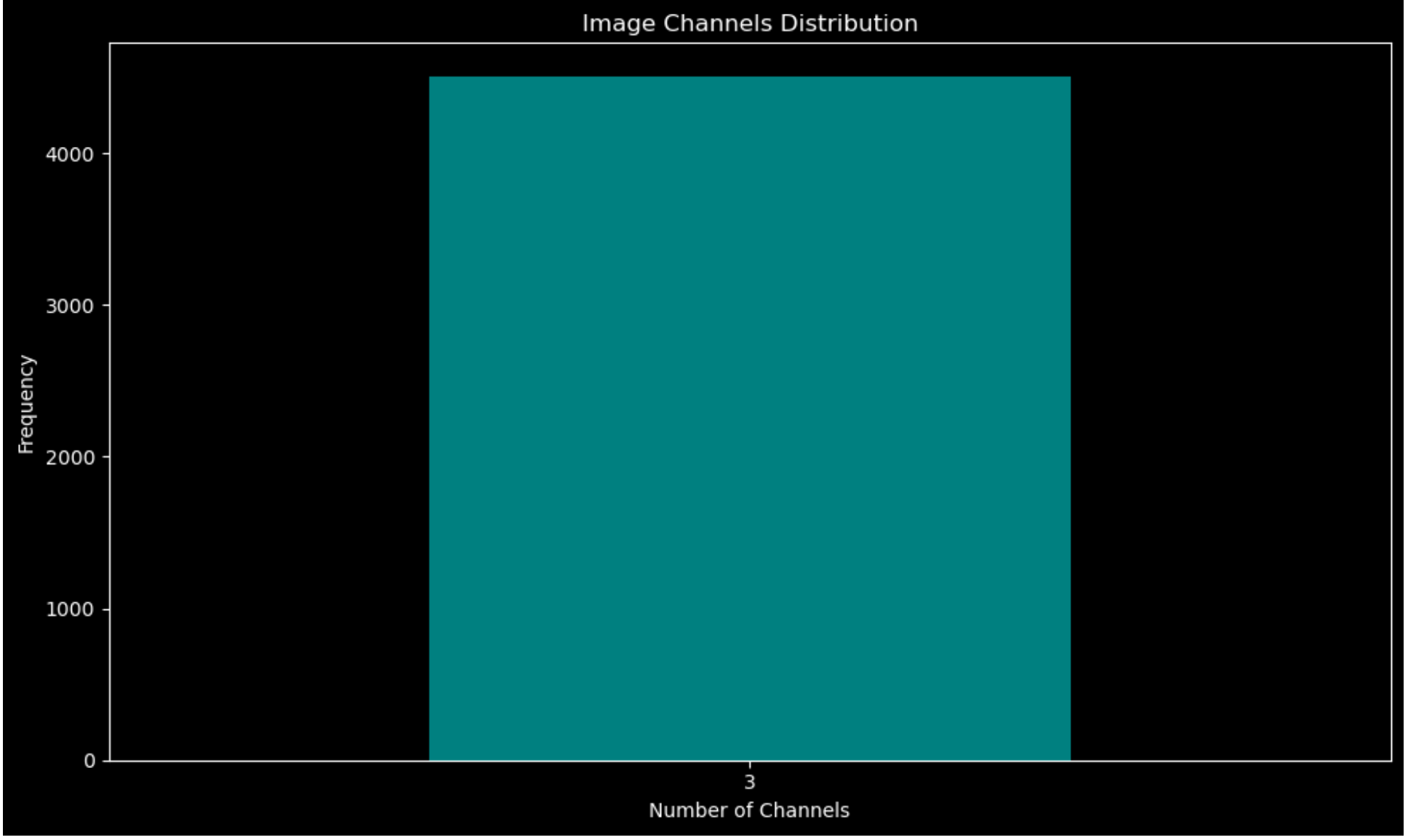


Image Format Distribution
Frequency
4000
3000
2000
1000
0
JPEG
Image Format

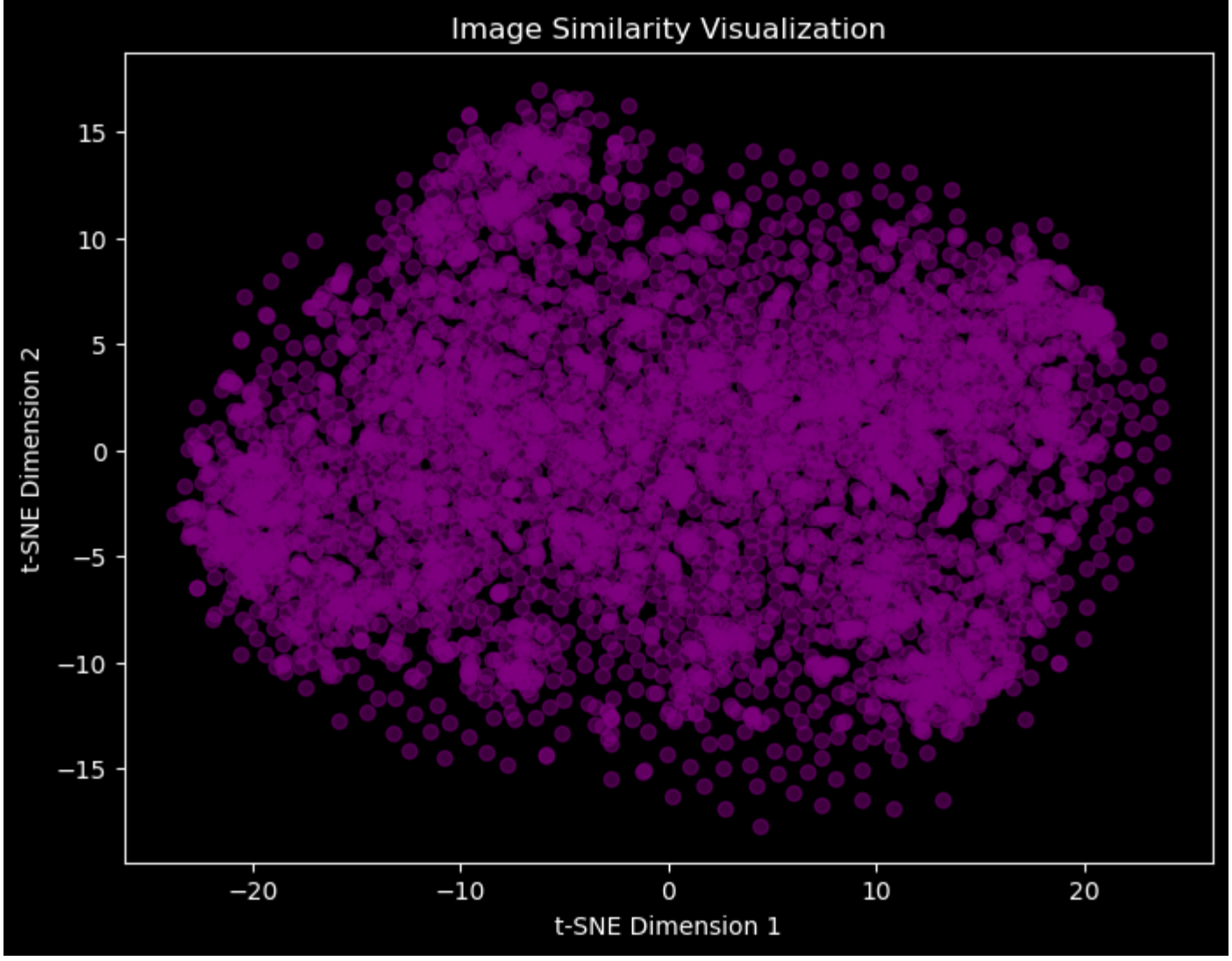


Out[5]: 2683

🖼 Image Findings

- Sharpness:
  - The distribution of image sharpness follows a normal distribution.
  - The mean sharpness value is approximately 60.3.
- Pixel Intensity:
  - Pixel intensity also follows a Gaussian distribution.
  - The average pixel intensity is approximately 112.9.
- Entropy:
  - Image entropy is an important measure of the amount of information in the images.
  - The mean entropy is approximately 7.4, with a maximum of ~8 and a minimum of ~3.8.
  - The distribution is positively skewed, indicating more images with higher entropy values.
- Image Size and Format:
  - All images in the dataset are in JPEG format.
  - The average image size is approximately 427.9 x 399.3 pixels.
  - Images are slightly more wide than tall on average, suggesting more landscape than portrait orientations.
  - The smallest image is 200x200 pixels, while the largest dimensions are 997 pixels (width) and 960 pixels (height).
  - Width and height distributions are similar but slightly different, both being negatively skewed with more values on the lower end.
- Image Channels:
  - Analysis of the number of channels in all images revealed that they are all RGB (Red, Green, Blue) images.
  - Each image has 3 channels, representing different levels of color that each pixel depends on.
  - This is crucial as it defines the amount of information and nuance present in the images, allowing for more sophisticated object detection and segmentation.
  - In RGB images, each pixel has a value from 0 to 255 for each of the three channels.
  - This allows for up to 16,777,216 (256^3) possible colors, which is far more than the approximately 1 million colors the human eye can distinguish.
  - The richness of color information enables computer vision models to potentially surpass human performance in certain tasks, as they can leverage this additional information.

In [6]:
```python
# Convert images to numpy arrays
image_data_array = []
for filename in labels_df['FileName']:
    with Image.open(filename) as img:
        image_data_array.append(np.array(img.resize((224, 224))).flatten())
image_data_array = np.array(image_data_array)

# Perform t-SNE
tsne = TSNE(n_components=2, random_state=42)
image_embeddings = tsne.fit_transform(image_data_array)
del image_data_array  # Free up memory
gc.collect()

# Plot t-SNE embeddings
plt.figure(figsize=(8, 6))
plt.scatter(image_embeddings[:, 0], image_embeddings[:, 1], alpha=0.5, color='purple')
plt.xlabel('t-SNE Dimension 1')
plt.ylabel('t-SNE Dimension 2')
plt.title('Image Similarity Visualization')
plt.show()
plt.close()
gc.collect()
```



Out[6]: 2990

In [33…

```python
# Calculate pairwise distances
distances = pdist(image_embeddings)
distance_matrix = squareform(distances)

# Find pairs of images that are 99% or more similar
similarity_threshold = 0.01  # 99% similarity = 1% distance
similar_pairs = np.argwhere(distance_matrix < similarity_threshold)

# Remove self-comparisons and duplicates
similar_pairs = similar_pairs[similar_pairs[:, 0] < similar_pairs[:, 1]]

# Function to display image pairs
def display_image_pair(img1_path, img2_path):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

    img1 = Image.open(img1_path)
    ax1.imshow(img1)
    ax1.set_title(f"Image 1: {img1_path}\nSize: {img1.size[0]}x{img1.size[1]}")
    ax1.axis('off')

    img2 = Image.open(img2_path)
    ax2.imshow(img2)
    ax2.set_title(f"Image 2: {img2_path}\nSize: {img2.size[0]}x{img2.size[1]}")
    ax2.axis('off')

    plt.tight_layout()
    plt.show()

# Display up to 5 pairs of similar images
num_pairs_to_show = min(5, len(similar_pairs))
for i in range(num_pairs_to_show):
    idx1, idx2 = similar_pairs[i]
    img1_path = labels_df['FileName'].iloc[idx1]
    img2_path = labels_df['FileName'].iloc[idx2]
    print(f"Similarity: {1 - distance_matrix[idx1, idx2]:.2%}")
    display_image_pair(img1_path, img2_path)

# Clean up
gc.collect()
```

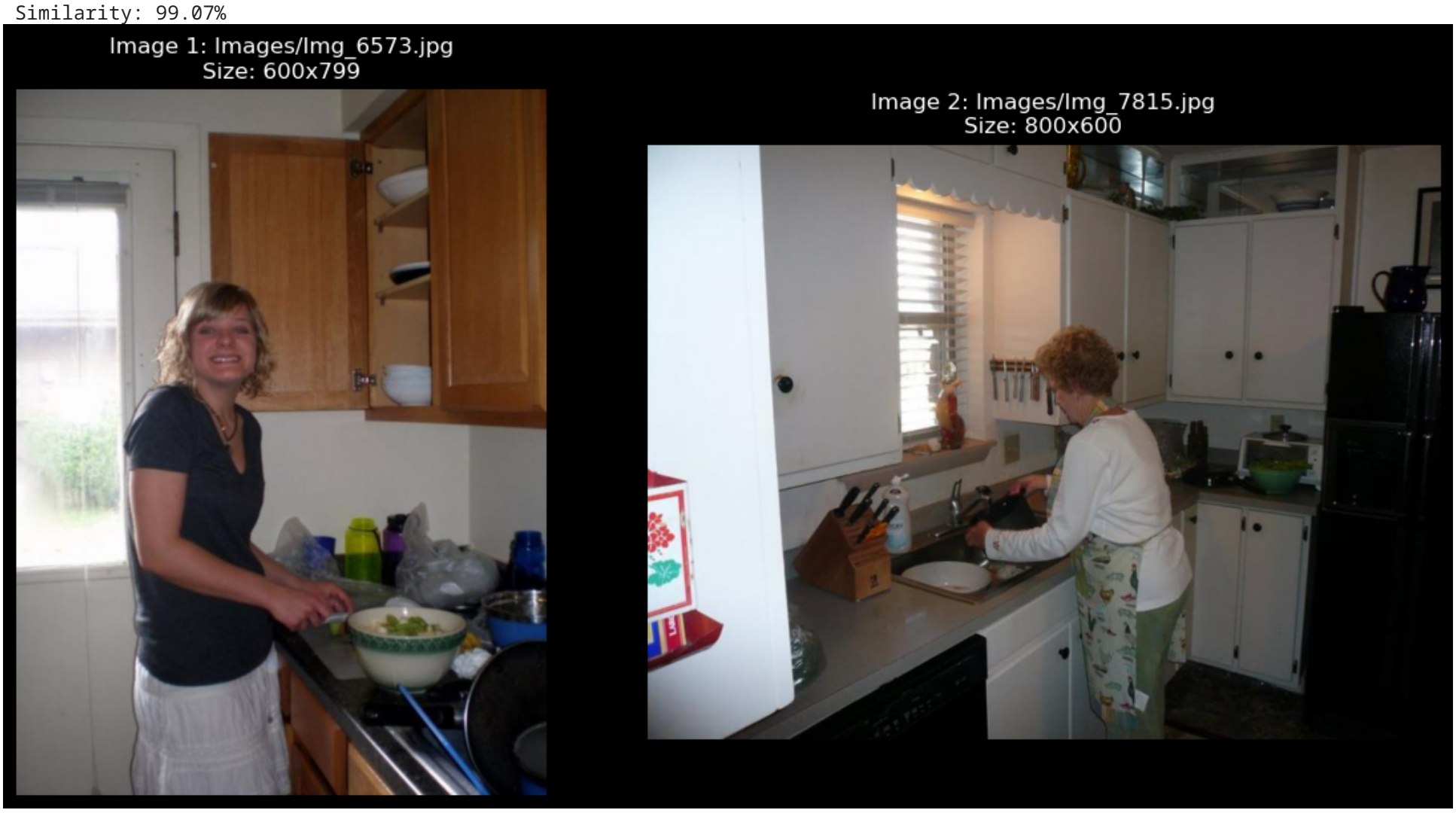
```
Similarity: 99.40%
```

Similarity: 99.77%

Similarity: 99.25%

Similarity: 99.42%


```
Similarity: 99.07%
```



Out[33]: 18991

🖼 Image Similarity and Dataset Diversity:

- My similarity index analysis reveals that the dataset is not pre-augmented, with the exception of a single image. This finding is significant, given the relatively small size of the training dataset.
- The lack of pre-augmentation provides an opportunity to implement data augmentation techniques within the TensorFlow architecture for neural networks, potentially enhancing the generalizability and performance of the model.

✔ Importance of Augmentation:

- Given that the dataset is not pre-augmented, I can apply various transformations and modifications to increase image variability and robustness during the training process.
- It's crucial to note that I will only apply augmentation techniques to the training dataset. The validation and test datasets will remain unaugmented to ensure the reliability and trustworthiness of the results.
- I intend to implement techniques such as random shift, rotate, zoom, etc., in the dataset and evaluate the resulting performance increase on the validation and independently sourced dataset.

✔ Similarity Analysis Examples:

- It's important to note that these similarity comparisons are based on pixel-level analysis and do not involve machine learning or neural networks. They simply compare the relative pixel values between images.

- Image2514 and Image1559:
  - Backgrounds share similar colors
  - Human positioning and orientation are alike
  - Similar skin tone (white with high red content) and body positioning (hands up, in an applauding / holding an object pose)
  - Both subjects wear white and have either black hair or a black helmet
  - Bodies shown up to the waist
  - t-SNE similarity score: 99.4%

- Image643 and Image8641:
  - 99.77% similarity score
  - Similar setting: market/poorer country
  - Multiple people with darker complexions
  - Similar background accents and subject positioning

- Image8343 and Image328:

- Different overall, but similar ground backgrounds, as well as the house / wall colour and person positioning

- Image9519 and Image7415:
  - Both feature white men wearing grey with stripes/black/white accents
  - Similar white pants, black elements, and white backgrounds
  - Similar floor and table materials

- Last example (washing dishes):
  - Same action depicted
  - Both subjects are women with blonde hair and white skin
  - Similar white backgrounds with wood accents and bright windows
  - Mirrored but similar poses

2.2 Removing Bad Data

In [7]:
```
# Print the first 3 rows of the DataFrame
labels_df.info()

# Display the dataframe to ensure it loaded correctly
labels_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4500 entries, 0 to 4499
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   FileName           4500 non-null   object 
 1   Class              4500 non-null   object 
 2   MoreThanOnePerson  4500 non-null   object 
 3   HighLevelCategory  4500 non-null   object 
 4   Size               4500 non-null   object 
 5   Format             4500 non-null   object 
 6   Sharpness          4500 non-null   float64
 7   PixelIntensity     4500 non-null   float64
 8   Entropy            4500 non-null   float64
 9   Channels           4500 non-null   int64  
 10  Width              4500 non-null   int64  
 11  Height             4500 non-null   int64  
dtypes: float64(3), int64(3), object(6)
memory usage: 422.0+ KB
```

Out[7]:

	FileName	Class	MoreThanOnePerson	HighLevelCategory	Size	Format	Sharpness	PixelIntensity
0	Images/Img_460.jpg	blowing_bubbles	YES	Social_LeisureActivities	(451, 300)	JPEG	61.813498	101.80992
1	Images/Img_8152.jpg	blowing_bubbles	YES	Social_LeisureActivities	(902, 600)	JPEG	74.981818	59.27309
2	Images/Img_9056.jpg	jumping	YES	Sports_Recreation	(451, 300)	JPEG	42.343249	160.77146
3	Images/Img_3880.jpg	pushing_a_cart	YES	OutdoorActivities	(400, 300)	JPEG	58.893304	75.00397
4	Images/Img_3168.jpg	writing_on_a_book	NO	Artistic_MusicalActivities	(300, 451)	JPEG	76.743293	102.73673

In [8]:
```
import matplotlib.pyplot as plt
from PIL import Image

# Number of images to display
num_images = 6

# Get random indices from the DataFrame
random_indices = labels_df.index.to_series().sample(num_images)

# Create subplots
fig, axes = plt.subplots(1, num_images, figsize=(20, 4))

for i, idx in enumerate(random_indices):
    image_path = labels_df.loc[idx, 'FileName']
    class_label = labels_df.loc[idx, 'Class']
    more_than_one_person = labels_df.loc[idx, 'MoreThanOnePerson']
    high_level_category = labels_df.loc[idx, 'HighLevelCategory']

    # Open the image and get its size
    image = Image.open(image_path)
    width, height = image.size

    axes[i].imshow(image)  # Display color image
    axes[i].set_title(f"Class: {class_label}\nMore Than One: {more_than_one_person}\n"
                      f"Category: {high_level_category}\nSize: {width}x{height}", fontsize=8)
    axes[i].axis('off')

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

Dataset Labelling:

- As seen, we have successfully labelled our entire training dataset:
  - $\forall\ I \in D_{train}, \exists\ L_{Class}(I) \wedge L_{MoreThanOnePerson}(I) \wedge L_{HighLevelCategory}(I)$
  - where D_{train} is the training dataset, I is an image, $L_{Class}(I)$ is the action class label, $L_{MoreThanOnePerson}(I)$ is the label indicating whether more than one person is present, and $L_{HighLevelCategory}(I)$ is the high-level category label for image I
- After further exploring the dataframe, all images seem to be classified correctly across all three attributes. This means we can proceed with supervised learning to train a model using algorithms taught in the course.

In [9]:
```python
# Check if there are any duplicated rows in the DataFrame
has_duplicates = labels_df.duplicated().any()

# Print out the result
print(f"DuplicateData: {has_duplicates}")
```

```
DuplicateData: False
```

- Importance of removing bad training data:
  - Low-quality, inconsistent, or mislabeled data can introduce noise and bias.
  - Removing bad data can lead to improved model performance.
  - The adage "garbage in, garbage out" aptly describes the importance of data quality in machine learning.
- Consistency and variability across classes and attributes:
  - Some classes and MoreThanOnePerson cases present hard-to-distinguish examples, while others are consistently clear to human observers.
  - Similarities exist between certain classes, particularly in human positioning or background content.
  - Some classes have a higher proportion of MoreThanOnePerson cases, potentially affecting prediction accuracy and certainty.
- Importance of maintaining dataset diversity:
  - The dataset includes some blurry images and cases where the subject occupies only a small portion of the image.
  - Retaining these challenging examples is crucial, as they likely represent real-world scenarios present in the test and validation sets.
  - Keeping diverse cases helps improve the model's ability to generalize.
- Data cleaning considerations:
  - Despite the presence of a single duplicate image and some very similar images, no data was removed due to the relatively small size of the dataset.
  - Experiments with removing certain images showed negligible performance changes.
  - The primary focus was ensuring correct labeling of all data and confirming the absence of duplicates (except for the single known case).
- Impact on model performance and generalization:
  - Retaining all images, including challenging cases, aims to improve the model's ability to handle diverse real-world scenarios.
  - The presence of varied image qualities and compositions in the training set should help the model perform better on similar cases in the validation and test sets.
  - Correct labeling and minimal duplication ensure that the model learns from a diverse yet accurate representation of the task at hand.

In [10…
```python
import matplotlib.pyplot as plt
from PIL import Image

# Specify the image filenames
img1_path = 'Images/Img_786.jpg'
img2_path = 'Images/Img_5378.jpg'

# Create a figure with two subplots side by side
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Display first image
img1 = Image.open(img1_path)
ax1.imshow(img1)
ax1.set_title(f"Image: {img1_path}\nSize: {img1.size[0]}x{img1.size[1]}")
ax1.axis('off')

# Display second image
img2 = Image.open(img2_path)
ax2.imshow(img2)
ax2.set_title(f"Image: {img2_path}\nSize: {img2.size[0]}x{img2.size[1]}")
ax2.axis('off')

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

🔍 Image Variation and Dataset Characteristics:

- Environment Diversity:
  - Images showcase a mix of indoor and outdoor settings
  - Outdoor scenes feature varied backgrounds, including green landscapes and ocean views
- Demographic Representation:
  - Dataset predominantly features individuals with lighter skin tones
  - This observation aligns with the high pixel intensity values noted earlier
- Action Variety:
  - Observed classes include activities such as riding a bike, pouring liquid, and applauding
- Image Quality and Duplication:
  - Images 786 and 5378 are identical, with 5378 being an oversaturated and overexposed version
  - Given that this class has an average number of images, and it's a single instance, these images were retained in the training set
- **Dataset Uniqueness:**
  - As seen in our dataset, there does not exist any image that is the same as another image:
  - NOTE: This dataset is not pre-augmented as we explored in the similarity score and by manually checking images.

In [17…

```python
# Use the existing labels_df (training + validation data)
train_count = len(labels_df)

# Load the test dataset
test_df = pd.read_csv('future_data_2024.csv')
test_count = len(test_df)

# Total count of samples in both datasets
total_df_count = train_count + test_count

# Count the number of images in the Images folder
image_folder = 'Images'
image_count = len([f for f in os.listdir(image_folder) if f.lower().endswith('.jpg')])

# Compare the counts
print(f"Number of samples in training DataFrame: {train_count}")
print(f"Number of samples in test DataFrame: {test_count}")
print(f"Total number of samples in both DataFrames: {total_df_count}")
print(f"Number of images in folder: {image_count}")

if total_df_count == image_count:
    print("The total number of samples in both DataFrames matches the number of images in the folder.")
else:
    print("WARNING: Mismatch between total DataFrame samples and image files!")
    print(f"Difference: {abs(total_df_count - image_count)}")

    # Combine filenames from both DataFrames
    all_filenames = set(labels_df['FileName']).union(set(test_df['FileName']))

    # Check for missing images
    image_files = set(os.listdir(image_folder))
    missing_images = [filename for filename in all_filenames if filename not in image_files]
```

```
Number of samples in training DataFrame: 4500
Number of samples in test DataFrame: 3128
Total number of samples in both DataFrames: 7628
Number of images in folder: 9532
WARNING: Mismatch between total DataFrame samples and image files!
Difference: 1904
```

Dataset Integrity Check Results:

- Our integrity check revealed some discrepancies in our dataset as shown above.
- We've identified a significant mismatch:
  - There are 1,904 more images in the folder than accounted for in our DataFrames.
- Resolution:
  - It was later revealed in the discussion forum that the extra images left in the folder were an accident and nothing to worry about.
  - While this resolves our immediate concern about data integrity, it's worth noting that having less training data than potentially available could impact our model's performance and generalizability.
  - With fewer samples of each class, there's a risk that our dataset may not be as diverse as it could have been, potentially limiting the model's ability to generalize to a wide range of scenarios.
- Implications:
  - We may need to be more reliant on data augmentation and other techniques to compensate for the reduced dataset size.
  - It will be crucial to carefully monitor for overfitting during the training process.
  - When evaluating our model's performance, we should keep in mind that results might have been potentially better with a larger, more diverse dataset.
- This experience highlights the importance of clear communication in collaborative projects and the need for thorough data management practices in machine learning workflows.

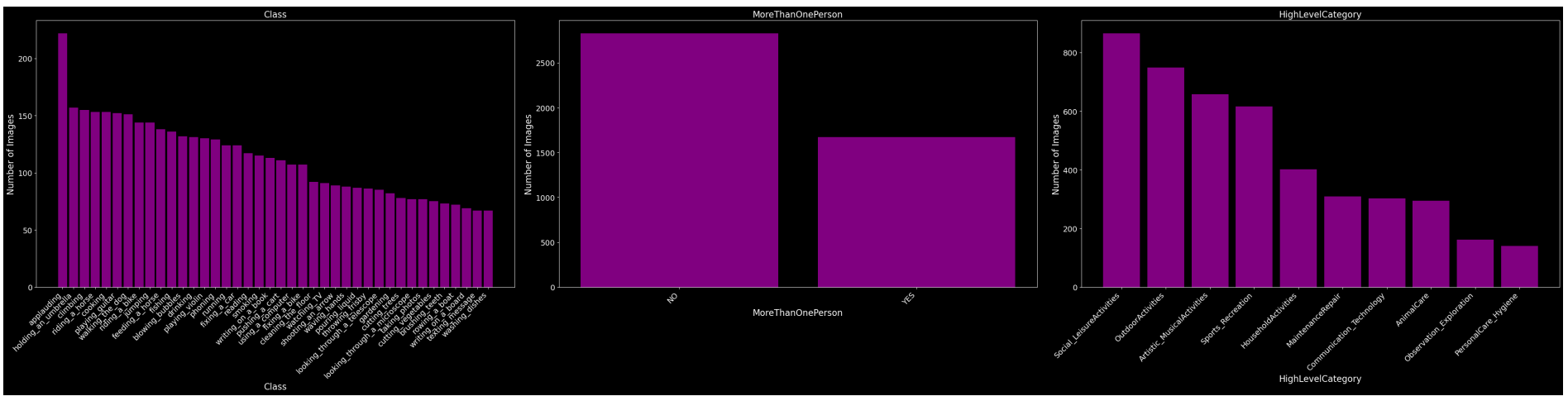
2.3 Class / Data Distributions

Exploring Data Distribution and Class Balance:

- We will now explore the distribution of our target variables: action Classes, MoreThanOnePerson, and HighLevelCategory in our training and validation splits. This analysis is crucial for understanding potential biases in our model and determining if we need to employ oversampling or undersampling techniques.

In [11…

```python
# Get the unique labels for each column
class_labels = labels_df['Class'].unique()
more_than_one_person_labels = labels_df['MoreThanOnePerson'].unique()
high_level_category_labels = labels_df['HighLevelCategory'].unique()

# Create a grid of subplots with increased figure size
plt.figure(figsize=(40, 10))

columns = ['Class', 'MoreThanOnePerson', 'HighLevelCategory']

for i, col in enumerate(columns):
    plt.subplot(1, 3, i + 1)

    # Get the value counts for the current column
    value_counts = labels_df[col].value_counts()

    # Plot the value counts as bars with maximum increased spacing
    bar_width = 5
    x_positions = np.arange(len(value_counts)) * 6  # Further increase the spacing between bars
    plt.bar(x_positions, value_counts, bar_width, color='purple')
    plt.title(col, fontsize=16)
    plt.xticks(x_positions, value_counts.index, rotation=45, ha='right', fontsize=14)
    plt.yticks(fontsize=14)
    plt.xlabel(col, fontsize=16)
    plt.ylabel('Number of Images', fontsize=16)

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



In [12…

```python
import plotly.graph_objects as go
import pandas as pd
import plotly.express as px
```

```python
# Prepare the data
class_data = []
for class_label in class_labels:
    class_df = labels_df[labels_df['Class'] == class_label]
    more_than_one_counts = class_df['MoreThanOnePerson'].value_counts().reset_index()
    more_than_one_counts.columns = ['MoreThanOnePerson', 'Count']
    more_than_one_counts['Class'] = class_label
    class_data.append(more_than_one_counts)

# Combine all class data
plot_data = pd.concat(class_data, ignore_index=True)

# Generate a color palette
color_palette = px.colors.qualitative.Plotly

# Create figure
fig = go.Figure()

# Add traces for each class with different colors
for i, class_label in enumerate(class_labels):
    class_df = plot_data[plot_data['Class'] == class_label]
    fig.add_trace(
        go.Bar(x=class_df['MoreThanOnePerson'],
               y=class_df['Count'],
               name=class_label,
               visible=(i==0),
               marker_color=color_palette[i % len(color_palette)])
    )

# Create and add slider
steps = []
for i, class_label in enumerate(class_labels):
    step = dict(
        method="update",
        args=[{"visible": [False] * len(class_labels)},
              {"title": f"Distribution of 'More Than One Person' for Class: {class_label}"}],
        label=class_label
    )
    step["args"][0]["visible"][i] = True  # Toggle i'th trace to "visible"
    steps.append(step)

sliders = [dict(
    active=0,
    currentvalue={"prefix": "Class: "},
    pad={"t": 50},
    steps=steps
)]

# Update layout with dark mode
fig.update_layout(
    sliders=sliders,
    title_text="Distribution of 'More Than One Person' by Class",
    xaxis_title_text="More Than One Person",
    yaxis_title_text="Number of Images",
    height=500,
    width=800,
    showlegend=False,
    plot_bgcolor='rgba(0,0,0,0)',
    paper_bgcolor='rgba(0,0,0,0)',
    font=dict(color='white'),
    title=dict(font=dict(color='white')),
    xaxis=dict(gridcolor='gray', linecolor='gray'),
    yaxis=dict(gridcolor='gray', linecolor='gray')
)

# Show the plot
fig.show()
```

Detailed Analysis of Data Distribution and Class Balance:

- Action Classes:
  - Our dataset contains 40 different action classes: applauding, holding_an_umbrella, climbing, riding_a_horse, cooking, playing_guitar, walking_the_dog, riding_a_bike, jumping, feeding_a_horse, fishing, blowing_bubbles, drinking, playing_violin, phoning, running, fixing_a_car, reading, writing_on_a_book, pushing_a_cart, using_a_computer, fixing_a_bike, cleaning_the_floor, watching_TV, shooting_an_arrow, waving_hands, pouring_liquid, throwing_frisby, looking_through_a_telescope, gardening, cutting_trees, looking_through_a_microscope, taking_photos, cutting_vegetables, brushing_teeth, rowing_a_boat, writing_on_a_board, texting_message, washing_dishes.
  - The distribution of samples across these classes is imbalanced.
  - The most common action class is "applauding" with almost 250 samples, while the least common classes like "washing_dishes" have around 60-70 samples.
  - This imbalance may lead to overprediction of majority classes and underprediction of minority classes.
  - To address this, we will primarily use data augmentation techniques. We will also consider oversampling minority classes to improve performance across all actions. NOTE: This will only be done for our training set as doing so to the validation or test would mean we cannot trust our results.
  - I will keep track of performance on both high-sample classes (like "applauding") and low-sample classes (like "washing_dishes") to monitor the impact of class imbalance on our model's predictions.

- MoreThanOnePerson:
  - The dataset contains over 1500 images with more than one person and almost 3000 with a single person.
  - This imbalanced distribution may affect our model's ability to accurately predict scenes with multiple people.
  - However, we've been informed that the presence of multiple people is already an indicator of uncertainty for the model.
  - I will primarily rely on data augmentation techniques to address this imbalance, as it allows us to maintain our sample size while potentially improving performance.

- HighLevelCategory:
  - There are 10 high-level categories in our dataset: Social_LeisureActivites, OutdoorActivities, Artistic_MusicalActivities, Sports_Recreation, HouseholdActivities, MaintenanceRepair, Communication_Technology, AnimalCare, Observation_Exploration, and PersonalCare_Hygiene.
  - The distribution across these categories is uneven, mirroring the imbalance seen in the action classes.
  - While this distribution may impact our model's ability to generalize across different types of actions, we are not required to predict this category directly so will not spend too much time on it.

- Implications for Model Training and Evaluation:
  - Given these distributions, we expect our model may have biases towards majority classes and categories.
  - To mitigate this, we will primarily use data augmentation techniques. This approach will help increase the diversity of our training data without reducing the number of samples for any class.
  - We will also experiment with oversampling minority classes to further balance our dataset.
  - During evaluation, we will use metrics such as balanced accuracy and macro-averaged F1-score to ensure we're assessing performance fairly across all classes and categories.
  - We will be transparent about these potential biases in our model documentation, clearly stating its limitations and the scenarios where it may underperform, particularly for classes with fewer samples.

3.0 Data Splitting

- In this step, we will split the data from `train_data_2024.csv` into training and validation subsets. This split enables us to train and validate our model before evaluating it against the unseen 'blind' test data in `future_data_2024.csv` .

- After experimenting with different ratios, I settled on an 80/20 split for training and validation. This split gives us more validation data, which is beneficial for assessing our model's performance. Since we have a separate test set (`future_data_2024.csv`), we can afford to allocate more data for validation.

- To address the potential reduction of training data, we will utilize oversampling and augmentation techniques on our training split (not on validation or test data). This approach will help mitigate

data scarcity and potentially improve our model's performance and generalization capabilities.

- As recommended by Ruman and Azadeh in our lectures on supervised learning, it's crucial to have independent and similarly distributed test data. This practice helps avoid data leakage and ensures an unbiased evaluation of the final model.
- While not required for this assignment, I've also created an independently sourced dataset. This dataset was created through image scraping and by taking photos of myself performing several actions. This additional dataset serves as a real-world test of our model's application and its ability to generalize beyond the provided data.
- Creating an independently sourced dataset aligns with the scientific method of model evaluation. It involves careful consideration of data collection methods to prevent overlap with the training data and to ensure a diverse, representative sample of real-world scenarios.
- When collecting such a dataset, we must consider factors like:
  - Diversity of subjects and actions
  - Variation in lighting conditions and backgrounds
  - Potential biases in our data collection process
  - Ethical considerations in data collection and usage
- This structured approach to dataset splitting and independent evaluation underscores the rigor of our model development phase. It enables us to meticulously gauge the model's generalization capabilities and ensure robustness before confronting the unseen data in `future_data_2024.csv` and our independently sourced dataset.

In [38…

```
# Create label encoders
le_class = LabelEncoder()
le_more_than_one = LabelEncoder()
le_high_level = LabelEncoder()

# Fit and transform the labels
labels_df['Class_encoded'] = le_class.fit_transform(labels_df['Class'])
labels_df['MoreThanOnePerson_encoded'] = le_more_than_one.fit_transform(labels_df['MoreThanOnePerson'])
labels_df['HighLevelCategory_encoded'] = le_high_level.fit_transform(labels_df['HighLevelCategory'])

# Split the data
train_df, val_df = train_test_split(labels_df, test_size=0.2, random_state=42, stratify=labels_df['Class'])

print("Number of images in the training set:", len(train_df))
print("Number of images in the validation set:", len(val_df))

# Function to get encoded labels
def get_labels(df):
    return {
        'class_output': df['Class_encoded'].values,
        'more_than_one_output': df['MoreThanOnePerson_encoded'].values,
        'high_level_output': df['HighLevelCategory_encoded'].values
    }

# Get encoded labels for train and validation sets
train_labels = get_labels(train_df)
val_labels = get_labels(val_df)

# Store the label encoders for later use
label_encoders = {
    'class': le_class,
    'more_than_one': le_more_than_one,
    'high_level': le_high_level
}
```

```
Number of images in the training set: 3600
Number of images in the validation set: 900
```

Data Splitting Process:

- I performed an 80/20 random split on our `train_data_2024.csv` dataset to create our training and validation sets:
  - Training set: 3,600 images (80%)
  - Validation set: 900 images (20%)
- Key points about our split:
  - I used a random state (seed) of 42, ensuring reproducibility of this split across different computers due to pseudorandom number generation.
  - The split is stratified based on the 'Class' column, maintaining the same proportion of classes in both sets.
- Implications and next steps:
  - This split should provide sufficient data to train an effective model, especially when combined with transfer learning, oversampling, and data augmentation techniques.
  - The validation set is large enough to give us reliable performance estimates during the training process.
  - Must be cautious about overfitting, particularly when applying advanced techniques to improve performance.
  - Regular monitoring of both training and validation performance will be crucial to ensure we're generalizing well and not overfitting to the training data.

3.1 Checking for Data Leaks

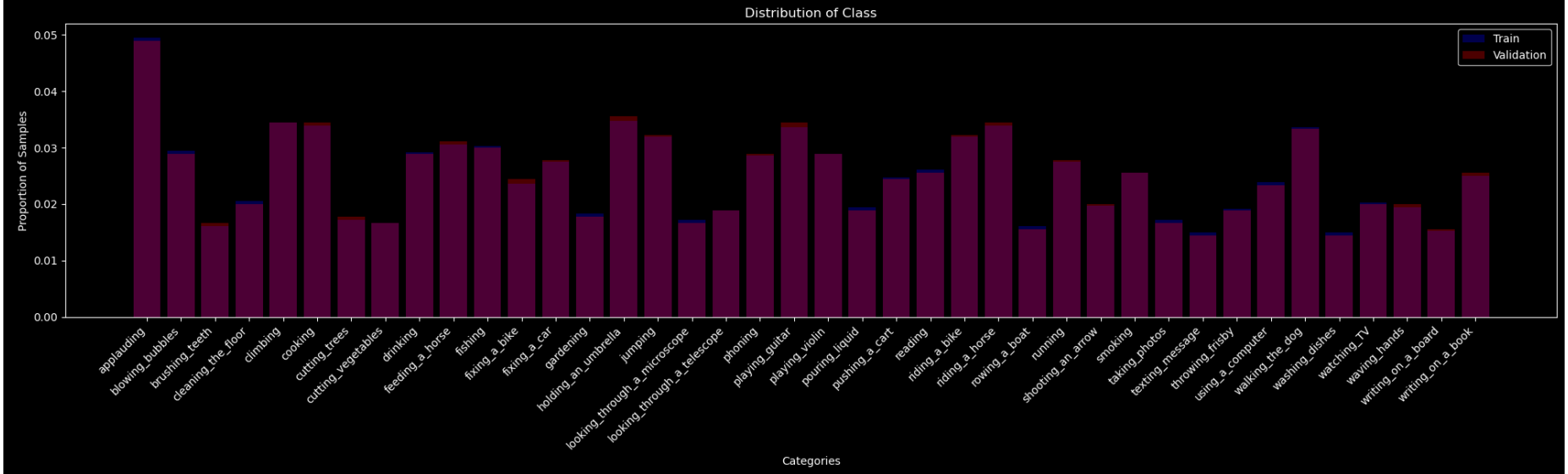
- Random splitting is essential but may lead to leakage if two splits are not truly independent.
- I will utilize insights from Exploratory Data Analysis (EDA) to detect any hidden sources of leakage

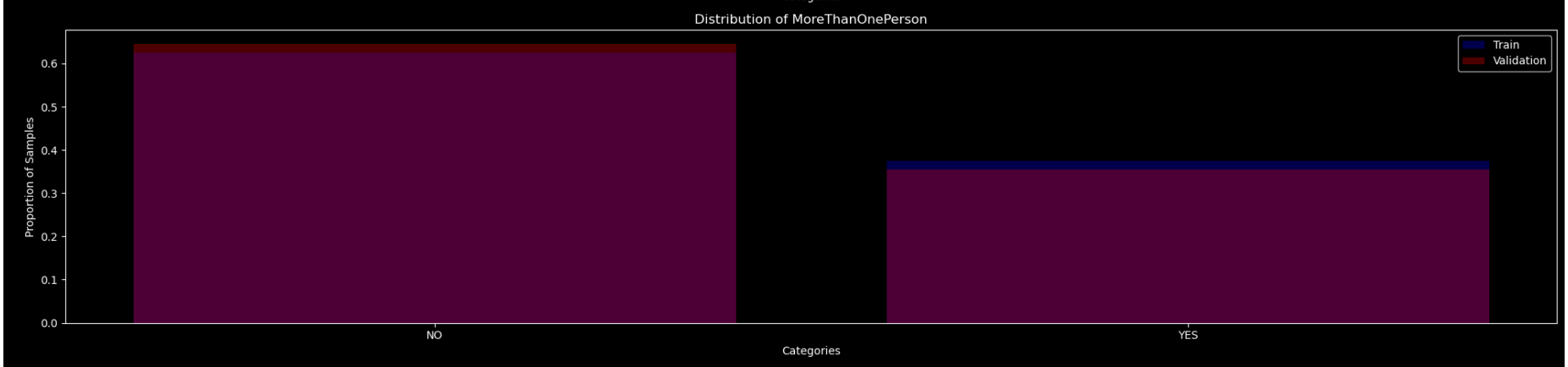
in the dataset.

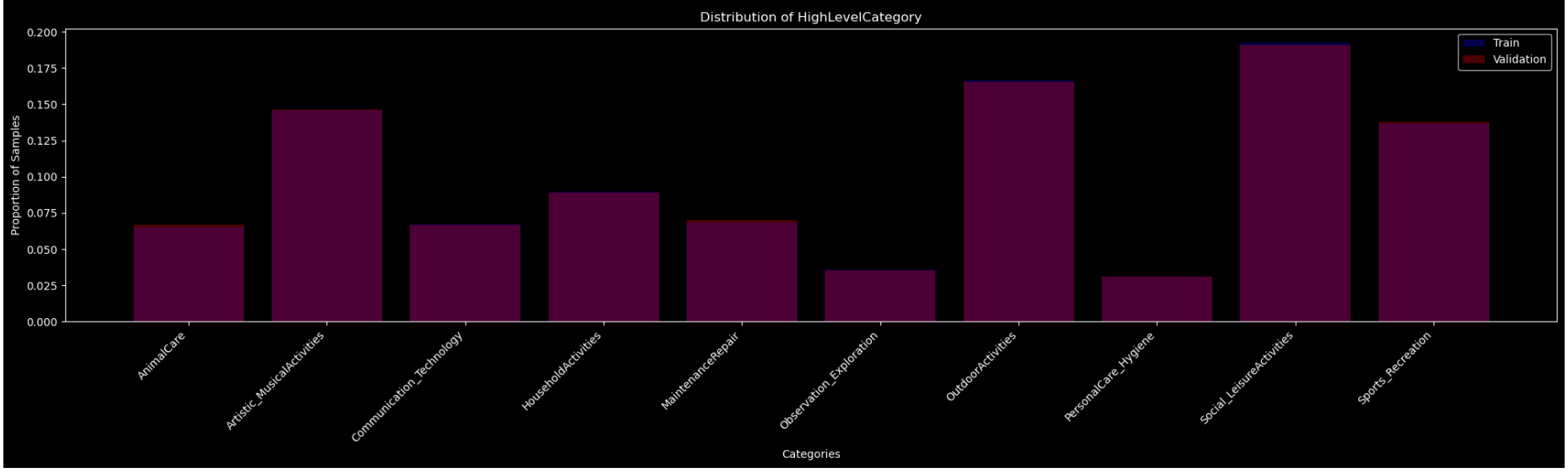
- I will examine histograms for each attribute in the training and validation sets, using different colors, to ensure the splits are identically distributed.

In [39…

```
def plot_distribution(train_df, val_df, columns):
    plt.figure(figsize=(20, 6 * len(columns)))

    for i, col in enumerate(columns):
        plt.subplot(len(columns), 1, i + 1)

        train_value_counts = train_df[col].value_counts(normalize=True)
        val_value_counts = val_df[col].value_counts(normalize=True)

        # Get all unique labels from both train and val sets
        all_labels = sorted(set(train_value_counts.index) | set(val_value_counts.index))

        # Fill in missing values with 0
        train_value_counts = train_value_counts.reindex(all_labels, fill_value=0)
        val_value_counts = val_value_counts.reindex(all_labels, fill_value=0)

        bar_width = 0.8
        x_positions = np.arange(len(all_labels))

        # Plot training data
        plt.bar(x_positions, train_value_counts, bar_width, color='blue', label='Train', alpha=0.3)

        # Plot validation data
        plt.bar(x_positions, val_value_counts, bar_width, color='red', label='Validation', alpha=0.3)

        plt.title(f'Distribution of {col}')
        plt.xlabel('Categories')
        plt.ylabel('Proportion of Samples')

        # Adjust label rotation based on the column
        if col == 'MoreThanOnePerson':
            plt.xticks(x_positions, all_labels, rotation=0, ha='center')
        else:
            plt.xticks(x_positions, all_labels, rotation=45, ha='right')

        plt.legend()

    plt.tight_layout()
    plt.show()

columns_to_check = ['Class', 'MoreThanOnePerson', 'HighLevelCategory']
plot_distribution(train_df, val_df, columns_to_check)
```







- As seen by the purple (overlap) the training, and validation datasets are approximately the same, meaning our split was very successful.

3.2 Image Resizing

In [40…

```python
IMG_SIZE = 224
BATCH_SIZE = 32

def process_path(file_path, class_label, more_than_one_label, high_level_label):
    img = tf.io.read_file(file_path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, [IMG_SIZE, IMG_SIZE])
    img = tf.cast(img, tf.float32) / 255.0  # Normalize to [0, 1]
    return img, (class_label, more_than_one_label, high_level_label)

def create_dataset(df, labels):
    dataset = tf.data.Dataset.from_tensor_slices((
        df['FileName'].values,
        labels['class_output'],
        labels['more_than_one_output'],
        labels['high_level_output']
    ))

    dataset = dataset.map(process_path, num_parallel_calls=tf.data.AUTOTUNE)
    dataset = dataset.batch(BATCH_SIZE)
    dataset = dataset.prefetch(tf.data.AUTOTUNE)

    return dataset
```

Data Preprocessing and Dataset Creation:

- Batch Size and Hardware Considerations:
  - A batch size of 32 was chosen, which is the maximum my available hardware can handle efficiently.
  - My Hardware Specs: 16GB DDR4 RAM, GeForce RTX 3070 Laptop GPU (8GB GDDR6 VRAM), Intel i7-10875H CPU (8 cores, 16 threads, up to 5.1GHz).

    - A smaller batch size would be suboptimal for training efficiency, while a larger one would exceed the hardware capabilities.

- Image Preprocessing:
    - Square images (224x224 pixels) are used with a 'squish' methodology to maintain all information from the original images.
    - While this approach slightly distorts the aspect ratio, most images in the dataset are near-square, minimizing the impact.
    - The 224x224 size was chosen based on:
        - Computational limitations for training with higher resolutions.
        - Compatibility with popular transfer learning models (e.g., VGG, ResNet) which often use this input size.
        - Balance between retaining sufficient detail for edge, and object detection and nuances, and enabling potential mobile/edge deployment.
    - Pixel values are normalized to the range [0, 1] to improve training stability and convergence.

- Dataset Creation and Optimization:
    - TensorFlow's Dataset API is used for efficient data loading and preprocessing.
    - The `prefetch` operation is employed to overlap data preprocessing and model execution, improving performance.
    - It's crucial to ensure that batches are randomized after caching to maintain optimal training conditions and prevent any unintended patterns from batch order.

- These preprocessing steps and dataset creation methods are designed to balance model performance, training efficiency, and potential real-world application constraints. They set the foundation for effective model training while considering the hardware limitations and the possibility of future mobile deployment.

In [41…

```python
# Create the datasets
train_ds = create_dataset(train_df, train_labels)
val_ds = create_dataset(val_df, val_labels)

def display_batch(dataset, label_encoders, num_images=6):
    plt.figure(figsize=(20, 4))
    for images, labels in dataset.take(1):
        print("Shape of images:", images.shape)
        print("Type of labels:", type(labels))
        print("Number of label sets:", len(labels))
        for i, label_set in enumerate(labels):
            print(f"Shape of label set {i}:", label_set.shape)
            print(f"Sample of label set {i}:", label_set[0].numpy())

        for i in range(min(num_images, len(images))):
            ax = plt.subplot(1, num_images, i + 1)
            plt.imshow(images[i])

            # Convert numeric labels back to original string labels
            class_label = label_encoders['class'].inverse_transform([labels[0][i].numpy()])[0]
            more_than_one_label = label_encoders['more_than_one'].inverse_transform([labels[1][i].numpy()])[0]
            high_level_label = label_encoders['high_level'].inverse_transform([labels[2][i].numpy()])[0]

            plt.title(f"Class: {class_label}\n"
                      f"More Than One: {more_than_one_label}\n"
                      f"Category: {high_level_label}", fontsize=8)
            plt.axis("off")
    plt.tight_layout()
    plt.show()

# Encode our labels
label_encoders = {
    'class': le_class,
    'more_than_one': le_more_than_one,
    'high_level': le_high_level
}

print("Training Data Examples:")
display_batch(train_ds, label_encoders)

print("Validation Data Examples:")
display_batch(val_ds, label_encoders)
```

```
Training Data Examples:
Shape of images: (32, 224, 224, 3)
Type of labels: <class 'tuple'>
Number of label sets: 3
Shape of label set 0: (32,)
Sample of label set 0: 21
Shape of label set 1: (32,)
Sample of label set 1: 0
Shape of label set 2: (32,)
Sample of label set 2: 8
2024-08-29 00:56:26.529037: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholder/_0' with dtype string and shape [3600]
        [[{{node Placeholder/_0}}]]
2024-08-29 00:56:26.529265: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholder/_3' with dtype int64 and shape [3600]
        [[{{node Placeholder/_3}}]]
```


```
Validation Data Examples:
Shape of images: (32, 224, 224, 3)
Type of labels: <class 'tuple'>
Number of label sets: 3
Shape of label set 0: (32,)
Sample of label set 0: 16
Shape of label set 1: (32,)
Sample of label set 1: 0
Shape of label set 2: (32,)
Sample of label set 2: 5
```

```
2024-08-29 00:56:27.070164: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholder/_0' with dtype string and shape [900]
        [[{{node Placeholder/_0}}]]
2024-08-29 00:56:27.070427: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholder/_3' with dtype int64 and shape [900]
        [[{{node Placeholder/_3}}]]
```

- While dimensionality reduction techniques like Principal Component Analysis (PCA) can be crucial when dealing with high-dimensional image data, I've chosen a different approach for our Human Action Recognition (HAR) task.
- Instead of using PCA, I will leverage neural network architectures that inherently perform dimensionality reduction, such as using GlobalAveragePooling layers or other methodologies.
- I have already resized all our images, which provides a form of dimensionality reduction. I will experiment with different model architectures to find an optimal balance between model size and performance.
- By utilizing these architectural approaches, I aim to maintain the spatial information in our images while still reducing the number of parameters in the model, potentially enhancing the model's ability to distinguish between different action types.
- I hypothesize that this approach will allow me to create an efficient and effective HAR model without the need for explicit dimensionality reduction techniques like PCA.

Oversampling Considerations:

- Oversampling can create a more generalizable dataset by increasing the representation of minority classes, which is useful for imbalanced datasets.
- If this model were deployed in a real-world setting, it might benefit from oversampling to help generalize better to unseen data. However, I hypothesize that the test dataset may mimic the previous split we made for training and validation. This would mean that applying oversampling leads to a decrease in model performance on the test. We already know that our validation split when normalized will be identically distributed for class balance so oversampling will lead to this also performing worse.
- While oversampling is valuable for improving model generalization, it is crucial to evaluate its impact on the specific dataset and problem. In this case, I have determined that oversampling does not provide the desired performance improvements for our HAR task given our validation and test datasets.
- I will therefore first focus on developing a robust HAR model without using oversampling. This approach allows me to evaluate the inherent capabilities of the neural network in learning from the original data distribution, determining the limitations and bias given the dataset.

Non-Neural Network Algorithm Conclusion:

- In the exploration of the Human Action Recognition (HAR) task, several non-neural network algorithms were tested, including logistic regression and tree-based models such as Decision Trees, Random Forests, and XGBoost. While these algorithms showed promise in capturing certain aspects of the multi-label classification problem (predicting action class, MoreThanOnePerson, and HighLevelCategory), their performance did not meet initial expectations for this complex image classification task.
- Although neural networks, particularly Convolutional Neural Networks (CNNs), are often the preferred choice for image-based tasks due to their ability to learn hierarchical representations, exploring traditional machine learning algorithms provided valuable insights. This exploration helped establish a baseline performance and highlighted the inherent challenges of the HAR task.
- GridSearchCV, a wrapper-based hyperparameter tuning technique, was employed to fine-tune the hyperparameters of the non-neural network algorithms. By systematically searching through a predefined parameter grid, GridSearchCV helped optimize the performance of these models within their inherent limitations.

- Ensemble methods, such as Random Forests and XGBoost, which leverage multiple weak learners to create a stronger classifier, showed better performance compared to single-model approaches:
  - Bagging (Bootstrap Aggregating), used in Random Forests, helped reduce overfitting to some extent by training multiple classifiers on random subsets of the training data.
  - Boosting, employed by XGBoost, showed some ability to capture complex patterns by iteratively focusing on difficult examples.

- Despite these efforts, the non-neural network approaches fell short in accurately capturing the nuanced visual features necessary for high-performance HAR. The complexity of distinguishing between various human actions, detecting multiple persons, and categorizing high-level activities from image data proved to be beyond the capabilities of these traditional algorithms.

- This exploration reinforced the hypothesis that deep learning approaches, particularly CNNs, would be more suitable for the HAR task. The ability of neural networks to automatically learn relevant features from raw image data appears to be crucial for achieving the level of performance required for this task.

4.0 Neural Networks Models

In [ ]:
```
why we'll do augmentation in
```

In [ ]:
```
requirements goal
```

In [ ]:
```
metrics
```

In [33…
```
import pathlib
import tempfile
import shutil
import tensorflow as tf

# Create a temporary directory for TensorBoard logs
logdir = pathlib.Path(tempfile.mkdtemp()) / "tensorboard_logs"
shutil.rmtree(logdir, ignore_errors=True)
logdir.mkdir(parents=True, exist_ok=True)

# Create a TensorBoard callback
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=str(logdir),
    histogram_freq=1,
    write_graph=True,
    write_images=True,
    update_freq='epoch',
    profile_batch=2,
    embeddings_freq=1,
)

# Load the TensorBoard notebook extension
%load_ext tensorboard

# Open an embedded TensorBoard viewer
%tensorboard --logdir {str(logdir)}

print(f"TensorBoard logs will be saved to: {logdir}")
print("TensorBoard should now be visible in the output cell below.")
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
2024-08-29 00:53:05.331439: I tensorflow/tsl/profiler/lib/profiler_session.cc:104] Profiler session initializing.
2024-08-29 00:53:05.331462: I tensorflow/tsl/profiler/lib/profiler_session.cc:119] Profiler session started.
2024-08-29 00:53:05.331657: I tensorflow/tsl/profiler/lib/profiler_session.cc:131] Profiler session tear down.
ERROR: Failed to launch TensorBoard (exited with 1).
Contents of stderr:
2024-08-29 00:53:05.981798: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimiz
ed to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the
appropriate compiler flags.

NOTE: Using experimental fast data loading logic. To disable, pass
    "--load_fast=false" and report issues on GitHub. More details:
    https://github.com/tensorflow/tensorboard/issues/4784

Address already in use
Port 6006 is in use by another program. Either identify and stop that program, or start the server with a differen
t port.
TensorBoard logs will be saved to: /tmp/tmpxn19jia7/tensorboard_logs
TensorBoard should now be visible in the output cell below.
```

In [ ]:
```
introduction to neural networks, requirements for our model. our general approach, some methodology, using tensorf
```

4.1 Multi-Layer-Perceptron (MLP) Classifier (Baseline Model)

- Data generators are essential for efficiently loading and preprocessing large datasets, especially when dealing with image data.
- By creating train and validation generators, we can feed batches of images to our neural network during training, reducing memory usage and improving computational efficiency.
- The generators handle tasks such as data augmentation, resizing images to a consistent size, and converting labels to the appropriate format, simplifying the data preparation process.

In [35…

```
from tensorflow.keras import layers, models
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

def create_light_mobilenet_model(num_classes, num_more_than_one_classes, num_high_level_categories):
    base_model = MobileNetV2(input_shape=(IMG_SIZE, IMG_SIZE, 3), include_top=False, weights='imagenet', alpha=0
    base_model.trainable = False  # Freeze the base model

    inputs = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
    x = base_model(inputs, training=False)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dropout(0.2)(x)

    # Three output heads
    output_class = layers.Dense(num_classes, activation='softmax', name='class_output')(x)
    output_more_than_one = layers.Dense(num_more_than_one_classes, activation='softmax', name='more_than_one_outp
    output_high_level = layers.Dense(num_high_level_categories, activation='softmax', name='high_level_output')(x

    model = models.Model(inputs=inputs, outputs=[output_class, output_more_than_one, output_high_level])
    return model


num_classes = len(label_encoders['class'].classes_)
num_more_than_one_classes = len(label_encoders['more_than_one'].classes_)
num_high_level_categories = len(label_encoders['high_level'].classes_)

model = create_light_mobilenet_model(num_classes, num_more_than_one_classes, num_high_level_categories)
model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_4 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 mobilenetv2_0.35_224 (Function  (None, 7, 7, 1280)  410208      ['input_4[0][0]']
 al)

 global_average_pooling2d_1 (Gl  (None, 1280)        0           ['mobilenetv2_0.35_224[0][0]']
 obalAveragePooling2D)

 dropout_1 (Dropout)            (None, 1280)         0           ['global_average_pooling2d_1[0][0
                                                                 ]']

 class_output (Dense)           (None, 40)           51240       ['dropout_1[0][0]']

 more_than_one_output (Dense)   (None, 2)            2562        ['dropout_1[0][0]']

 high_level_output (Dense)      (None, 10)           12810       ['dropout_1[0][0]']

==================================================================================================
Total params: 476,820
Trainable params: 66,612
Non-trainable params: 410,208
__________________________________________________________________________________________________
```

In [36…

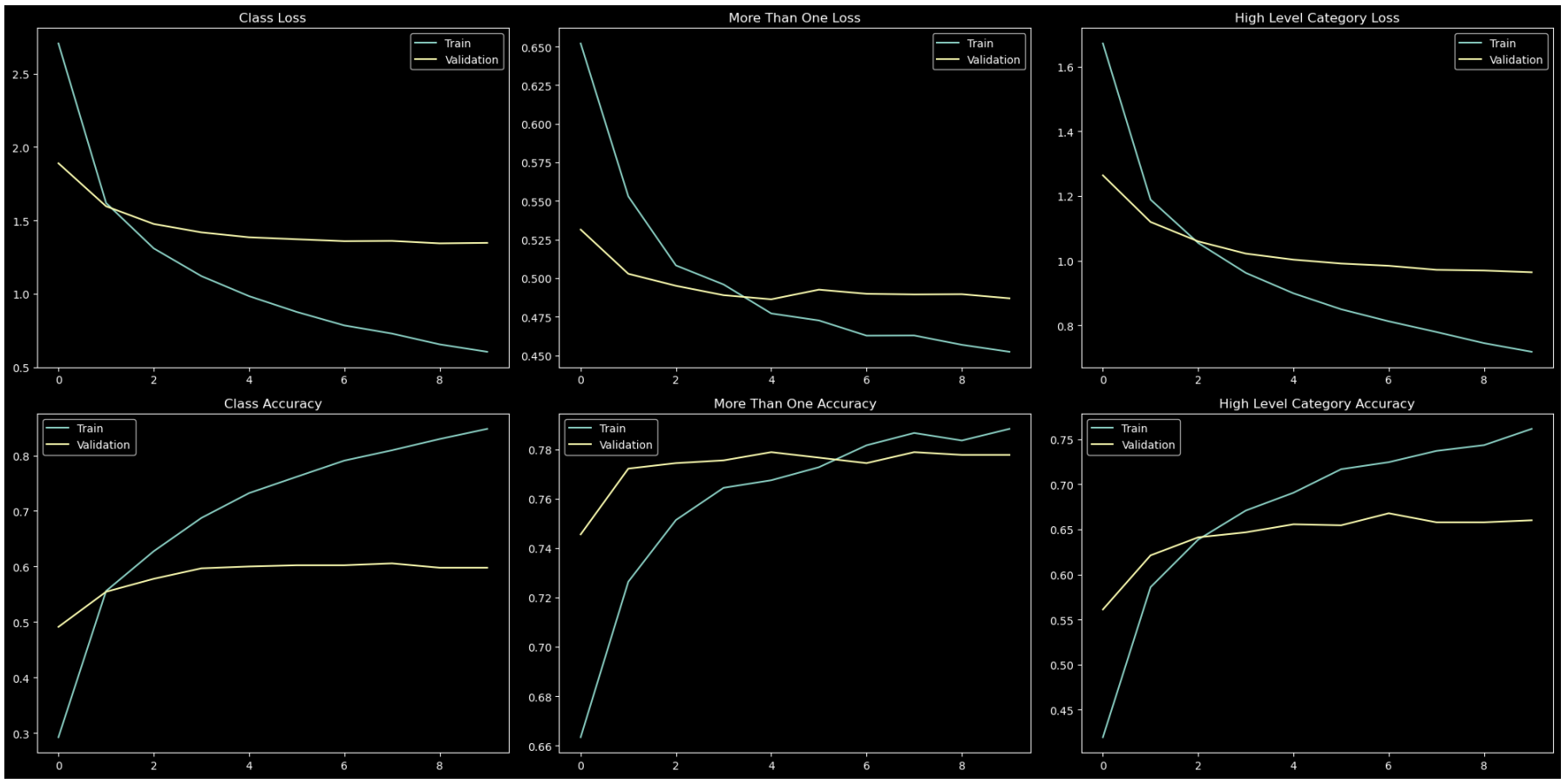
```python
# Compile the model
model.compile(
    optimizer=Adam(1e-3),
    loss={
        'class_output': SparseCategoricalCrossentropy(),
        'more_than_one_output': SparseCategoricalCrossentropy(),
        'high_level_output': SparseCategoricalCrossentropy()
    },
    metrics={
        'class_output': 'accuracy',
        'more_than_one_output': 'accuracy',
        'high_level_output': 'accuracy'
    }
)

# Callbacks
early_stopping = EarlyStopping(patience=3, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(factor=0.5, patience=2, min_lr=1e-5)

# Train the model
history = model.fit(
    train_ds,
    epochs=10,
    validation_data=val_ds,
    callbacks=[early_stopping, reduce_lr]
)
```

```
Epoch 1/10
2024-08-29 00:54:42.689815: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Execu
tor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must
feed a value for placeholder tensor 'Placeholder/_0' with dtype float and shape [3600,100]
         [[{{node Placeholder/_0}}]]
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[36], line 21
     18 reduce_lr = ReduceLROnPlateau(factor=0.5, patience=2, min_lr=1e-5)
     20 # Train the model
---> 21 history = model.fit(
     22     train_ds,
     23     epochs=10,
     24     validation_data=val_ds,
     25     callbacks=[early_stopping, reduce_lr]
     26 )

File /opt/anaconda/lib/python3.11/site-packages/keras/utils/traceback_utils.py:70, in filter_traceback.<locals>.e
rror_handler(*args, **kwargs)
     67     filtered_tb = _process_traceback_frames(e.__traceback__)
     68     # To get the full stack trace, call:
     69     # `tf.debugging.disable_traceback_filtering()`
---> 70     raise e.with_traceback(filtered_tb) from None
     71 finally:
     72     del filtered_tb

File /tmp/__autograph_generated_filevs8vn8n2.py:15, in outer_factory.<locals>.inner_factory.<locals>.tf__train_fu
nction(iterator)
     13 try:
     14     do_return = True
---> 15     retval_ = ag__.converted_call(ag__.ld(step_function), (ag__.ld(self), ag__.ld(iterator)), None, fscop
e)
     16 except:
     17     do_return = False

ValueError: in user code:

    File "/opt/anaconda/lib/python3.11/site-packages/keras/engine/training.py", line 1284, in train_function  *
        return step_function(self, iterator)
    File "/opt/anaconda/lib/python3.11/site-packages/keras/engine/training.py", line 1268, in step_function  **
        outputs = model.distribute_strategy.run(run_step, args=(data,))
    File "/opt/anaconda/lib/python3.11/site-packages/keras/engine/training.py", line 1249, in run_step  **
        outputs = model.train_step(data)
    File "/opt/anaconda/lib/python3.11/site-packages/keras/engine/training.py", line 1050, in train_step
        y_pred = self(x, training=True)
    File "/opt/anaconda/lib/python3.11/site-packages/keras/utils/traceback_utils.py", line 70, in error_handler
        raise e.with_traceback(filtered_tb) from None
    File "/opt/anaconda/lib/python3.11/site-packages/keras/engine/input_spec.py", line 298, in assert_input_compa
tibility
        raise ValueError(

    ValueError: Input 0 of layer "model_1" is incompatible with the layer: expected shape=(None, 224, 224, 3), fo
und shape=(None, 100)
```

In [32…

```python
# Plot training history
def plot_training_history(history):
    fig, axs = plt.subplots(2, 3, figsize=(20, 10))

    axs[0, 0].plot(history.history['class_output_loss'], label='Train')
    axs[0, 0].plot(history.history['val_class_output_loss'], label='Validation')
    axs[0, 0].set_title('Class Loss')
    axs[0, 0].legend()

    axs[0, 1].plot(history.history['more_than_one_output_loss'], label='Train')
    axs[0, 1].plot(history.history['val_more_than_one_output_loss'], label='Validation')
```

```
    axs[0, 1].set_title('More Than One Loss')
    axs[0, 1].legend()

    axs[0, 2].plot(history.history['high_level_output_loss'], label='Train')
    axs[0, 2].plot(history.history['val_high_level_output_loss'], label='Validation')
    axs[0, 2].set_title('High Level Category Loss')
    axs[0, 2].legend()

    axs[1, 0].plot(history.history['class_output_accuracy'], label='Train')
    axs[1, 0].plot(history.history['val_class_output_accuracy'], label='Validation')
    axs[1, 0].set_title('Class Accuracy')
    axs[1, 0].legend()

    axs[1, 1].plot(history.history['more_than_one_output_accuracy'], label='Train')
    axs[1, 1].plot(history.history['val_more_than_one_output_accuracy'], label='Validation')
    axs[1, 1].set_title('More Than One Accuracy')
    axs[1, 1].legend()

    axs[1, 2].plot(history.history['high_level_output_accuracy'], label='Train')
    axs[1, 2].plot(history.history['val_high_level_output_accuracy'], label='Validation')
    axs[1, 2].set_title('High Level Category Accuracy')
    axs[1, 2].legend()

    plt.tight_layout()
    plt.show()

plot_training_history(history)

# Save the model
model.save('final_light_mobilenet_model.h5')
```

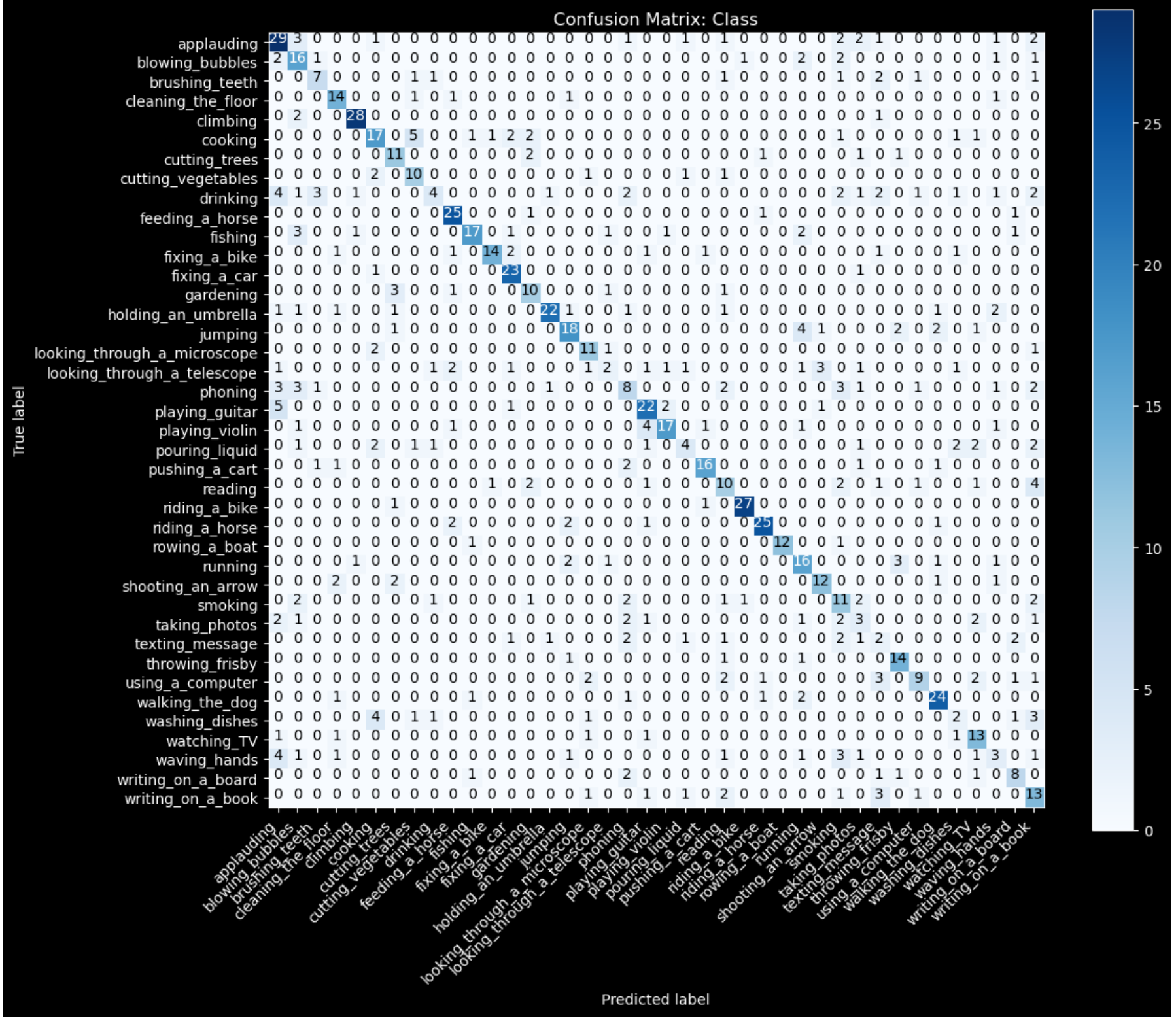


In [55…

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras.models import load_model

def plot_confusion_matrix(y_true, y_pred, classes, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(12, 10))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45, ha='right')
    plt.yticks(tick_marks, classes)

    # Add text annotations
    thresh = cm.max() / 2.
    for i, j in np.ndindex(cm.shape):
        plt.text(j, i, f"{cm[i, j]}",
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    plt.show()

def evaluate_model(model, dataset, label_encoders):
    all_true_class = []
    all_pred_class = []
    all_true_more_than_one = []
    all_pred_more_than_one = []
    all_true_high_level = []
    all_pred_high_level = []
```

```python
for images, labels in dataset:
    predictions = model.predict(images)

    all_true_class.extend(labels[0].numpy())
    all_pred_class.extend(np.argmax(predictions[0], axis=1))

    all_true_more_than_one.extend(labels[1].numpy())
    all_pred_more_than_one.extend(np.argmax(predictions[1], axis=1))

    all_true_high_level.extend(labels[2].numpy())
    all_pred_high_level.extend(np.argmax(predictions[2], axis=1))

tasks = ['Class', 'More Than One', 'High Level Category']
true_labels = [all_true_class, all_true_more_than_one, all_true_high_level]
pred_labels = [all_pred_class, all_pred_more_than_one, all_pred_high_level]
encoder_keys = ['class', 'more_than_one', 'high_level']

for task, true, pred, key in zip(tasks, true_labels, pred_labels, encoder_keys):
    classes = label_encoders[key].classes_

    # Plot confusion matrix
    plot_confusion_matrix(true, pred, classes, f'Confusion Matrix: {task}')

    # Print classification report
    print(f"\nClassification Report - {task}:")
    print(classification_report(true, pred, target_names=classes))

    # Calculate and print overall accuracy and misclassification count
    accuracy = np.mean(np.array(true) == np.array(pred))
    misclassified = np.sum(np.array(true) != np.array(pred))
    print(f"{task} Accuracy: {accuracy:.4f}")
    print(f"Number of misclassified {task}: {misclassified}")
    print("-" * 50)

# Load the saved model
loaded_model = load_model('final_light_mobilenet_model.h5')

# Evaluate on validation set
print("Evaluation on Validation Set:")
evaluate_model(loaded_model, val_ds, label_encoders)
```

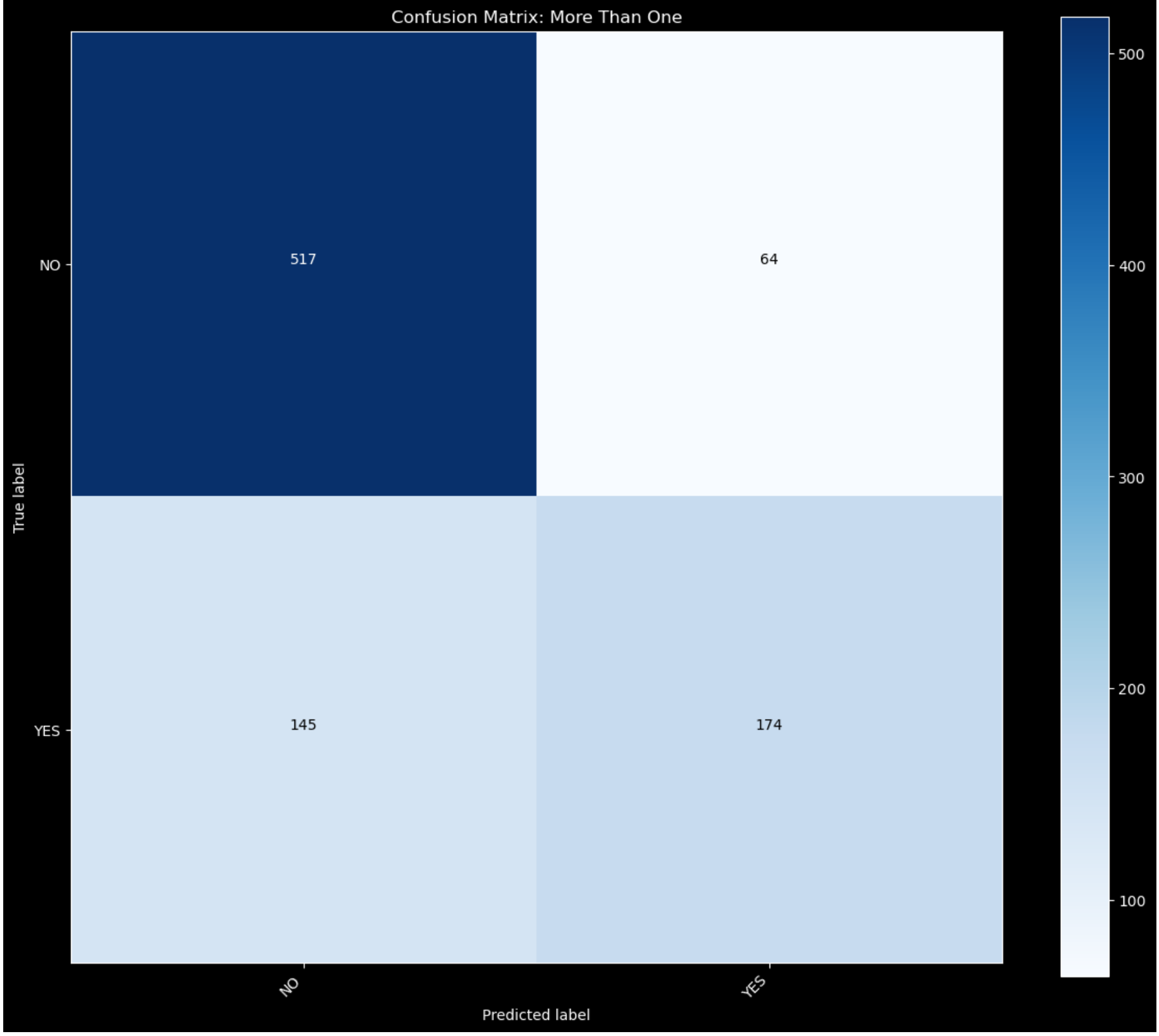
```
Evaluation on Validation Set:
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 895ms/step
1/1 [==============================] - 1s 633ms/step
1/1 [==============================] - 1s 833ms/step
1/1 [==============================] - 1s 806ms/step
1/1 [==============================] - 1s 836ms/step
1/1 [==============================] - 1s 718ms/step
1/1 [==============================] - 1s 654ms/step
1/1 [==============================] - 1s 565ms/step
1/1 [==============================] - 1s 771ms/step
1/1 [==============================] - 1s 780ms/step
1/1 [==============================] - 1s 603ms/step
1/1 [==============================] - 1s 634ms/step
1/1 [==============================] - 1s 726ms/step
1/1 [==============================] - 1s 699ms/step
1/1 [==============================] - 1s 686ms/step
1/1 [==============================] - 1s 581ms/step
1/1 [==============================] - 1s 684ms/step
1/1 [==============================] - 1s 625ms/step
1/1 [==============================] - 0s 497ms/step
1/1 [==============================] - 1s 800ms/step
1/1 [==============================] - 1s 949ms/step
1/1 [==============================] - 1s 663ms/step
1/1 [==============================] - 1s 629ms/step
1/1 [==============================] - 1s 570ms/step
1/1 [==============================] - 1s 953ms/step
1/1 [==============================] - 1s 614ms/step
1/1 [==============================] - 1s 720ms/step
1/1 [==============================] - 1s 556ms/step
```


Confusion Matrix: Class
True label
Predicted label
applauding
blowing_bubbles
brushing_teeth
cleaning_the_floor
climbing
cooking
cutting_trees
cutting_vegetables
drinking
feeding_a_horse
fishing
fixing_a_bike
fixing_a_car
gardening
holding_an_umbrella
jumping
looking_through_a_microscope
looking_through_a_telescope
phoning
playing_guitar
playing_violin
pouring_liquid
pushing_a_cart
reading
riding_a_bike
riding_a_horse
rowing_a_boat
running
shooting_an_arrow
smoking
taking_photos
texting_message
throwing_frisby
using_a_computer
walking_the_dog
washing_dishes
watching_TV
waving_hands
writing_on_a_board
writing_on_a_book
25
20
15
10
5
0

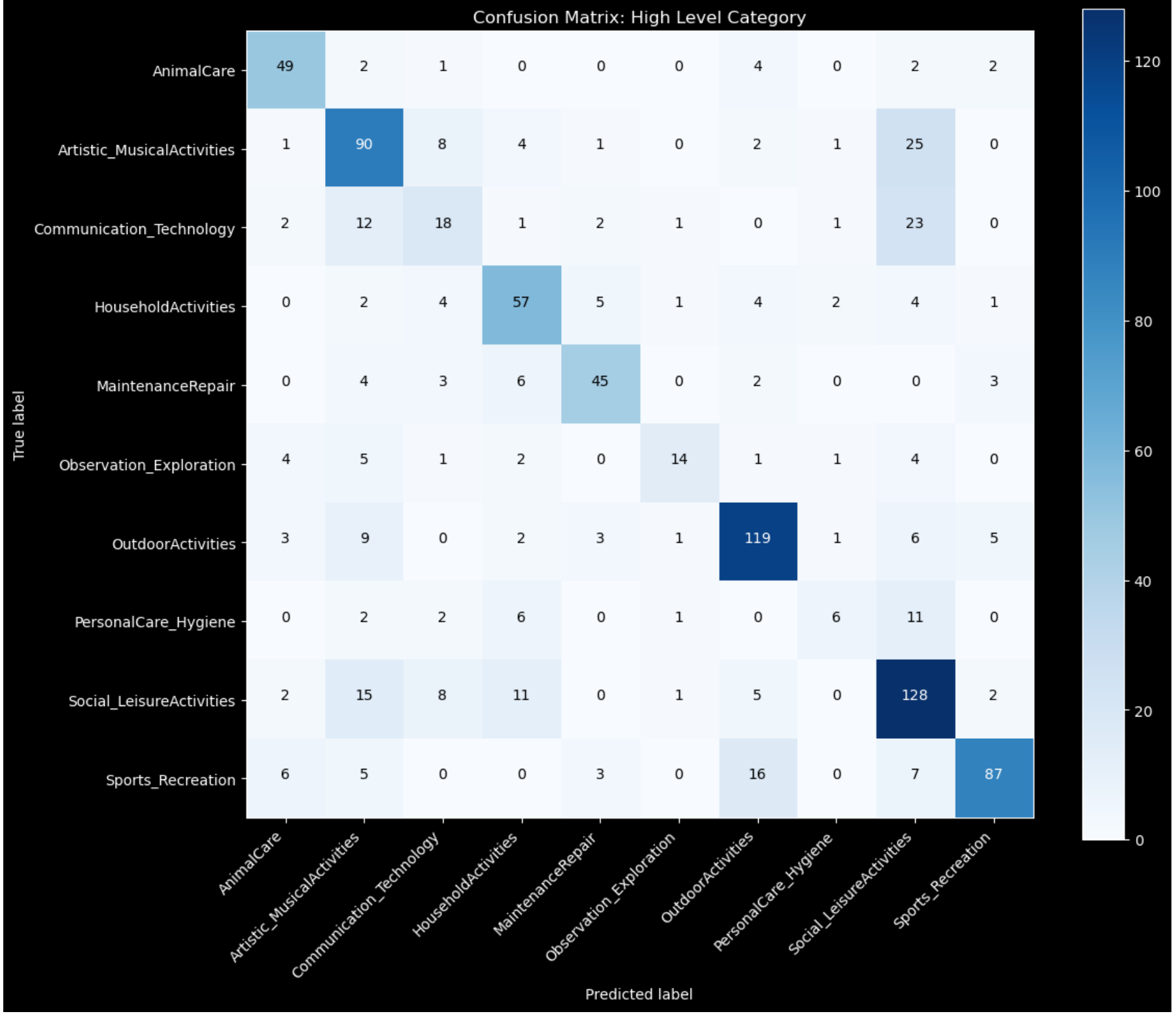
Classification Report - Class:

	precision	recall	f1-score	support
applauding	0.56	0.66	0.60	44
blowing_bubbles	0.46	0.62	0.52	26
brushing_teeth	0.54	0.47	0.50	15
cleaning_the_floor	0.64	0.78	0.70	18
climbing	0.90	0.90	0.90	31
cooking	0.59	0.55	0.57	31
cutting_trees	0.58	0.69	0.63	16
cutting_vegetables	0.53	0.67	0.59	15
drinking	0.44	0.15	0.23	26
feeding_a_horse	0.76	0.89	0.82	28
fishing	0.81	0.63	0.71	27
fixing_a_bike	0.88	0.64	0.74	22
fixing_a_car	0.74	0.92	0.82	25
gardening	0.56	0.62	0.59	16
holding_an_umbrella	0.88	0.69	0.77	32
jumping	0.69	0.62	0.65	29
looking_through_a_microscope	0.61	0.73	0.67	15
looking_through_a_telescope	0.33	0.12	0.17	17
phoning	0.35	0.31	0.33	26
playing_guitar	0.65	0.71	0.68	31
playing_violin	0.81	0.65	0.72	26
pouring_liquid	0.44	0.24	0.31	17
pushing_a_cart	0.84	0.73	0.78	22
reading	0.40	0.43	0.42	23
riding_a_bike	0.93	0.93	0.93	29
riding_a_horse	0.86	0.81	0.83	31
rowing_a_boat	1.00	0.86	0.92	14
running	0.52	0.64	0.57	25
shooting_an_arrow	0.71	0.67	0.69	18
smoking	0.33	0.48	0.39	23
taking_photos	0.19	0.20	0.19	15
texting_message	0.12	0.15	0.13	13
throwing_frisby	0.67	0.82	0.74	17
using_a_computer	0.64	0.43	0.51	21
walking_the_dog	0.77	0.80	0.79	30
washing_dishes	0.22	0.15	0.18	13
watching_TV	0.54	0.72	0.62	18
waving_hands	0.23	0.17	0.19	18
writing_on_a_board	0.57	0.57	0.57	14
writing_on_a_book	0.36	0.57	0.44	23
accuracy			0.61	900
macro avg	0.59	0.58	0.58	900
weighted avg	0.62	0.61	0.60	900

Class Accuracy: 0.6100
Number of misclassified Class: 351




```
Classification Report - More Than One:
              precision    recall  f1-score   support

          NO       0.78      0.89      0.83       581
         YES       0.73      0.55      0.62       319

    accuracy                           0.77       900
   macro avg       0.76      0.72      0.73       900
weighted avg       0.76      0.77      0.76       900

More Than One Accuracy: 0.7678
Number of misclassified More Than One: 209
--------------------------------------------------
```




```
Classification Report - High Level Category:
                          precision    recall  f1-score   support

                AnimalCare       0.73      0.82      0.77        60
Artistic_MusicalActivities       0.62      0.68      0.65       132
  Communication_Technology       0.40      0.30      0.34        60
       HouseholdActivities       0.64      0.71      0.67        80
         MaintenanceRepair       0.76      0.71      0.74        63
   Observation_Exploration       0.74      0.44      0.55        32
         OutdoorActivities       0.78      0.80      0.79       149
      PersonalCare_Hygiene       0.50      0.21      0.30        28
  Social_LeisureActivities       0.61      0.74      0.67       172
         Sports_Recreation       0.87      0.70      0.78       124

                  accuracy                           0.68       900
                 macro avg       0.66      0.61      0.63       900
              weighted avg       0.68      0.68      0.68       900

High Level Category Accuracy: 0.6811
Number of misclassified High Level Category: 287
--------------------------------------------------
```

In [ ]:

```python
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

def create_lightweight_model(num_classes, num_more_than_one_classes, num_high_level_categories, img_size=224):
    base_model = MobileNetV2(input_shape=(img_size, img_size, 3), include_top=False, weights='imagenet')
    base_model.trainable = False

    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.2)
    ])

    # Create separate output layers
    class_output = layers.Dense(num_classes, activation='softmax', name='class_output')(model.output)
    more_than_one_output = layers.Dense(num_more_than_one_classes, activation='softmax', name='more_than_one_outp
    high_level_output = layers.Dense(num_high_level_categories, activation='softmax', name='high_level_output')(m

    # Create the final model with multiple outputs
    final_model = models.Model(inputs=model.input,
```

```
                                  outputs=[class_output, more_than_one_output, high_level_output])

    return final_model

# Assume these variables are defined based on your dataset
num_classes = len(label_encoders['class'].classes_)
num_more_than_one_classes = len(label_encoders['more_than_one'].classes_)
num_high_level_categories = len(label_encoders['high_level'].classes_)

# Create the model
model = create_lightweight_model(num_classes, num_more_than_one_classes, num_high_level_categories)

# Compile the model
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss={
        'class_output': SparseCategoricalCrossentropy(),
        'more_than_one_output': SparseCategoricalCrossentropy(),
        'high_level_output': SparseCategoricalCrossentropy()
    },
    loss_weights={
        'class_output': 1.0,
        'more_than_one_output': 0.5,
        'high_level_output': 0.5
    },
    metrics=['accuracy']
)

# Callbacks
early_stopping = EarlyStopping(patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(factor=0.2, patience=5, min_lr=1e-6)

# Train the model
try:
    history = model.fit(
        train_ds,
        epochs=20,
        validation_data=val_ds,
        callbacks=[early_stopping, reduce_lr]
    )
    print("Training completed successfully!")
except Exception as e:
    print(f"An error occurred during training: {str(e)}")

# Save the model
try:
    model.save('lightweight_model.h5')
    print("Model saved successfully!")
except Exception as e:
    print(f"An error occurred while saving the model: {str(e)}")

# Optional: Fine-tuning step (only if the initial training was successful)
if 'history' in locals():
    print("Starting fine-tuning...")
    base_model = model.layers[0]
    base_model.trainable = True
    for layer in base_model.layers[:-10]:
        layer.trainable = False

    model.compile(
        optimizer=Adam(learning_rate=1e-5),
        loss={
            'class_output': SparseCategoricalCrossentropy(),
            'more_than_one_output': SparseCategoricalCrossentropy(),
            'high_level_output': SparseCategoricalCrossentropy()
        },
        loss_weights={
            'class_output': 1.0,
            'more_than_one_output': 0.5,
            'high_level_output': 0.5
        },
        metrics=['accuracy']
    )

    try:
        history_fine_tune = model.fit(
            train_ds,
            epochs=10,
            validation_data=val_ds,
            callbacks=[early_stopping, reduce_lr]
        )
        print("Fine-tuning completed successfully!")
    except Exception as e:
        print(f"An error occurred during fine-tuning: {str(e)}")
```

```
Epoch 1/20
113/113 [==============================] - 211s 2s/step - loss: 3.9155 - class_output_loss: 2.7385 - more_than_on
e_output_loss: 0.6447 - high_level_output_loss: 1.7093 - class_output_accuracy: 0.3117 - more_than_one_output_acc
uracy: 0.6519 - high_level_output_accuracy: 0.4289 - val_loss: 2.5173 - val_class_output_loss: 1.6720 - val_more_
than_one_output_loss: 0.5628 - val_high_level_output_loss: 1.1278 - val_class_output_accuracy: 0.5667 - val_more_
than_one_output_accuracy: 0.7467 - val_high_level_output_accuracy: 0.6133 - lr: 0.0010
Epoch 2/20
113/113 [==============================] - 207s 2s/step - loss: 2.2956 - class_output_loss: 1.4730 - more_than_on
e_output_loss: 0.5536 - high_level_output_loss: 1.0916 - class_output_accuracy: 0.6022 - more_than_one_output_acc
uracy: 0.7225 - high_level_output_accuracy: 0.6269 - val_loss: 1.9721 - val_class_output_loss: 1.2753 - val_more_
than_one_output_loss: 0.5267 - val_high_level_output_loss: 0.8670 - val_class_output_accuracy: 0.6444 - val_more_
than_one_output_accuracy: 0.7822 - val_high_level_output_accuracy: 0.7133 - lr: 0.0010
Epoch 3/20
113/113 [==============================] - 208s 2s/step - loss: 1.8281 - class_output_loss: 1.1143 - more_than_on
e_output_loss: 0.5367 - high_level_output_loss: 0.8909 - class_output_accuracy: 0.6789 - more_than_one_output_acc
uracy: 0.7389 - high_level_output_accuracy: 0.6917 - val_loss: 1.7776 - val_class_output_loss: 1.1318 - val_more_
than_one_output_loss: 0.4966 - val_high_level_output_loss: 0.7950 - val_class_output_accuracy: 0.6767 - val_more_
than_one_output_accuracy: 0.8078 - val_high_level_output_accuracy: 0.7311 - lr: 0.0010
Epoch 4/20
113/113 [==============================] - 208s 2s/step - loss: 1.5342 - class_output_loss: 0.9155 - more_than_on
e_output_loss: 0.5035 - high_level_output_loss: 0.7338 - class_output_accuracy: 0.7369 - more_than_one_output_acc
uracy: 0.7592 - high_level_output_accuracy: 0.7439 - val_loss: 1.7296 - val_class_output_loss: 1.0965 - val_more_
than_one_output_loss: 0.4961 - val_high_level_output_loss: 0.7700 - val_class_output_accuracy: 0.6778 - val_more_
than_one_output_accuracy: 0.7978 - val_high_level_output_accuracy: 0.7400 - lr: 0.0010
Epoch 5/20
 12/113 [==>...........................] - ETA: 2:36 - loss: 1.4478 - class_output_loss: 0.8550 - more_than_one_o
utput_loss: 0.5101 - high_level_output_loss: 0.6755 - class_output_accuracy: 0.7708 - more_than_one_output_accura
cy: 0.7448 - high_level_output_accuracy: 0.7500
```

Multilayer Perceptron (MLP) for Image Classification:

- A Multilayer Perceptron (MLP) is a type of artificial neural network that consists of multiple layers of interconnected nodes (neurons). It typically comprises an input layer, one or more hidden layers, and an output layer.
- MLPs are useful for image classification tasks because:
  - They can learn complex, non-linear relationships between the input features (pixel values) and the output classes.
  - The multiple layers allow the network to learn hierarchical representations of the image data, capturing increasingly abstract features as the information propagates through the layers.
  - MLPs can handle high-dimensional input data, such as flattened image pixels, making them suitable for processing and classifying images.
- In an MLP, the input layer receives the flattened image pixels, and the information is then passed through the hidden layers, where each neuron applies a non-linear activation function to its weighted inputs. The output layer produces the final class probabilities or predictions.
- During training, the MLP learns the optimal weights and biases for each neuron using backpropagation and gradient descent optimization. This allows the network to minimize the classification error and improve its performance iteratively.

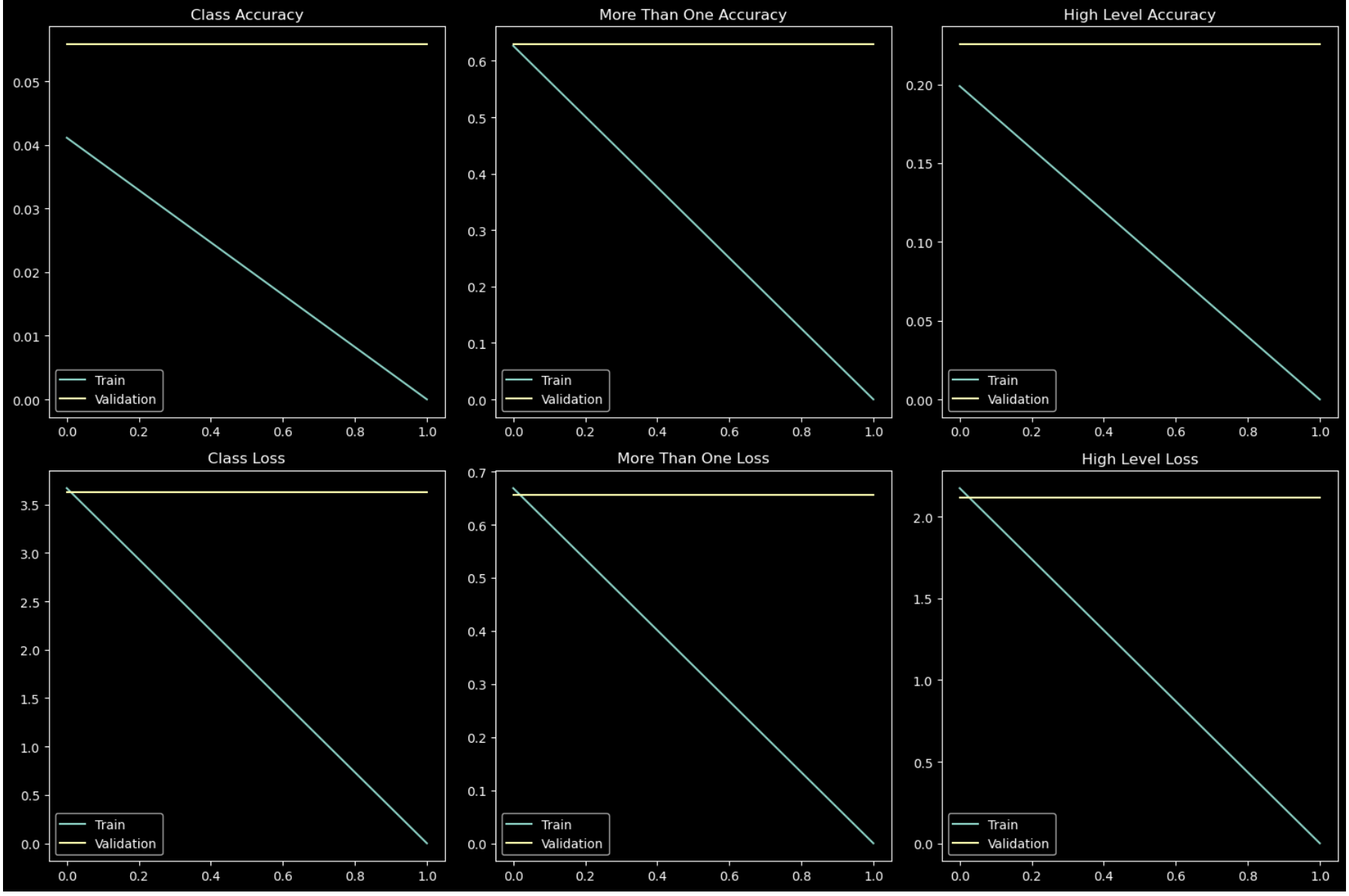
In [38…

```python
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, GlobalAveragePooling2D, Dense, Dropout

def preprocess_image(file_path):
    image = tf.io.read_file(file_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, (224, 224))
    image = tf.cast(image, tf.float32) / 255.0
    return image

def create_dataset(df, batch_size=16, shuffle=True):
    def generator():
        data = df.copy()
        if shuffle:
            data = data.sample(frac=1).reset_index(drop=True)
        for _, row in data.iterrows():
            image = preprocess_image(row['FilePath'])
            labels = {
                'class': row['Class'],
                'more_than_one': row['MoreThanOnePerson'],
                'high_level': row['HighLevelCategory']
            }
            yield image, labels

    dataset = tf.data.Dataset.from_generator(
        generator,
        output_signature=(
            tf.TensorSpec(shape=(224, 224, 3), dtype=tf.float32),
            {
                'class': tf.TensorSpec(shape=(), dtype=tf.int64),
                'more_than_one': tf.TensorSpec(shape=(), dtype=tf.int64),
                'high_level': tf.TensorSpec(shape=(), dtype=tf.int64)
            }
        )
    )

    if shuffle:
        dataset = dataset.shuffle(buffer_size=len(df))

    dataset = dataset.batch(batch_size)
    dataset = dataset.prefetch(buffer_size=tf.data.AUTOTUNE)
    return dataset
```

```
def create_multi_output_model(input_shape, num_class_classes, num_more_than_one_classes, num_high_level_classes)
    inputs = Input(shape=input_shape)

    x = Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
    x = MaxPooling2D((2, 2))(x)
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2))(x)
    x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = GlobalAveragePooling2D()(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.3)(x)

    class_output = Dense(num_class_classes, activation='softmax', name='class')(x)
    more_than_one_output = Dense(num_more_than_one_classes, activation='softmax', name='more_than_one')(x)
    high_level_output = Dense(num_high_level_classes, activation='softmax', name='high_level')(x)

    model = Model(inputs=inputs, outputs=[class_output, more_than_one_output, high_level_output])

    return model

# Create datasets
batch_size = 16
train_dataset = create_dataset(train_df, batch_size=batch_size, shuffle=True)
val_dataset = create_dataset(val_df, batch_size=batch_size, shuffle=False)

# Create the model
input_shape = (224, 224, 3)
num_class_classes = len(label_encoders['Class'].classes_)
num_more_than_one_classes = len(label_encoders['MoreThanOnePerson'].classes_)
num_high_level_classes = len(label_encoders['HighLevelCategory'].classes_)

model = create_multi_output_model(input_shape, num_class_classes, num_more_than_one_classes, num_high_level_clas

# Compile the model
model.compile(optimizer='adam',
              loss={
                  'class': 'sparse_categorical_crossentropy',
                  'more_than_one': 'sparse_categorical_crossentropy',
                  'high_level': 'sparse_categorical_crossentropy'
              },
              metrics=['accuracy'])

# Print model summary
model.summary()

# Train the model
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=10,
    steps_per_epoch=len(train_df) // batch_size,
    validation_steps=len(val_df) // batch_size
)

# Plot training history
import matplotlib.pyplot as plt

def plot_training_history(history):
    fig, axes = plt.subplots(2, 3, figsize=(15, 10))

    axes[0, 0].plot(history.history['class_accuracy'], label='Train')
    axes[0, 0].plot(history.history['val_class_accuracy'], label='Validation')
    axes[0, 0].set_title('Class Accuracy')
    axes[0, 0].legend()

    axes[0, 1].plot(history.history['more_than_one_accuracy'], label='Train')
    axes[0, 1].plot(history.history['val_more_than_one_accuracy'], label='Validation')
    axes[0, 1].set_title('More Than One Accuracy')
    axes[0, 1].legend()

    axes[0, 2].plot(history.history['high_level_accuracy'], label='Train')
    axes[0, 2].plot(history.history['val_high_level_accuracy'], label='Validation')
    axes[0, 2].set_title('High Level Accuracy')
    axes[0, 2].legend()

    axes[1, 0].plot(history.history['class_loss'], label='Train')
    axes[1, 0].plot(history.history['val_class_loss'], label='Validation')
    axes[1, 0].set_title('Class Loss')
    axes[1, 0].legend()

    axes[1, 1].plot(history.history['more_than_one_loss'], label='Train')
    axes[1, 1].plot(history.history['val_more_than_one_loss'], label='Validation')
    axes[1, 1].set_title('More Than One Loss')
    axes[1, 1].legend()

    axes[1, 2].plot(history.history['high_level_loss'], label='Train')
    axes[1, 2].plot(history.history['val_high_level_loss'], label='Validation')
    axes[1, 2].set_title('High Level Loss')
    axes[1, 2].legend()

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

```
plot_training_history(history)
```

```
Model: "model_9"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_10 (InputLayer)          [(None, 224, 224, 3  0           []
                                )]

 conv2d_27 (Conv2D)             (None, 224, 224, 32  896         ['input_10[0][0]']
                                )

 max_pooling2d_24 (MaxPooling2D  (None, 112, 112, 32  0          ['conv2d_27[0][0]']
 )                              )

 conv2d_28 (Conv2D)             (None, 112, 112, 64  18496       ['max_pooling2d_24[0][0]']
                                )

 max_pooling2d_25 (MaxPooling2D  (None, 56, 56, 64)  0           ['conv2d_28[0][0]']
 )

 conv2d_29 (Conv2D)             (None, 56, 56, 128)  73856       ['max_pooling2d_25[0][0]']

 global_average_pooling2d_9 (Gl  (None, 128)         0           ['conv2d_29[0][0]']
 obalAveragePooling2D)

 dense_16 (Dense)               (None, 128)          16512       ['global_average_pooling2d_9[0][0
                                                                 ]']

 dropout_15 (Dropout)           (None, 128)          0           ['dense_16[0][0]']

 class (Dense)                  (None, 40)           5160        ['dropout_15[0][0]']

 more_than_one (Dense)          (None, 2)            258         ['dropout_15[0][0]']

 high_level (Dense)             (None, 10)           1290        ['dropout_15[0][0]']

==================================================================================================
Total params: 116,468
Trainable params: 116,468
Non-trainable params: 0
__________________________________________________________________________________________________
Epoch 1/10
2024-08-26 17:54:12.590699: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Execu
tor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must
feed a value for placeholder tensor 'Placeholder/_0' with dtype int32
         [[{{node Placeholder/_0}}]]
2024-08-26 17:54:12.590921: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Execu
tor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must
feed a value for placeholder tensor 'Placeholder/_0' with dtype int32
         [[{{node Placeholder/_0}}]]
225/225 [==============================] - ETA: 0s - loss: 6.5130 - class_loss: 3.6688 - more_than_one_loss: 0.66
87 - high_level_loss: 2.1755 - class_accuracy: 0.0411 - more_than_one_accuracy: 0.6267 - high_level_accuracy: 0.1
989
2024-08-26 17:56:13.312576: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Execu
tor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must
feed a value for placeholder tensor 'Placeholder/_0' with dtype int32
         [[{{node Placeholder/_0}}]]
225/225 [==============================] - 131s 551ms/step - loss: 6.5130 - class_loss: 3.6688 - more_than_one_lo
ss: 0.6687 - high_level_loss: 2.1755 - class_accuracy: 0.0411 - more_than_one_accuracy: 0.6267 - high_level_accur
acy: 0.1989 - val_loss: 6.3998 - val_class_loss: 3.6251 - val_more_than_one_loss: 0.6561 - val_high_level_loss: 2
.1186 - val_class_accuracy: 0.0558 - val_more_than_one_accuracy: 0.6295 - val_high_level_accuracy: 0.2254
Epoch 2/10
WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator ca
n generate at least `steps_per_epoch * epochs` batches (in this case, 2250 batches). You may need to use the repe
at() function when building your dataset.
225/225 [==============================] - 8s 36ms/step - loss: 0.0000e+00 - class_loss: 0.0000e+00 - more_than_o
ne_loss: 0.0000e+00 - high_level_loss: 0.0000e+00 - class_accuracy: 0.0000e+00 - more_than_one_accuracy: 0.0000e
+00 - high_level_accuracy: 0.0000e+00 - val_loss: 6.3998 - val_class_loss: 3.6251 - val_more_than_one_loss: 0.656
1 - val_high_level_loss: 2.1186 - val_class_accuracy: 0.0558 - val_more_than_one_accuracy: 0.6295 - val_high_leve
l_accuracy: 0.2254
```



Justification for each MLP layer:
✔ Flatten Layer:

- We used a Flatten layer to convert the 2D input image (28x28) into a 1D vector because it is necessary to prepare the data for the subsequent fully connected layers. Flattening the input allows the model to treat the image as a simple vector, enabling it to learn global patterns and relationships between pixels more efficiently.

✔ Dense Layer (128 units, ReLU):

- We chose a Dense layer with 128 units as the first hidden layer because it provides a sufficient capacity for the model to capture intricate features and patterns in the input data. The ReLU activation function is used to introduce non-linearity, allowing the model to learn complex patterns and representations more effectively. ReLU activation also helps in faster convergence and reduces the likelihood of vanishing gradients.

✔ Dropout Layer (0.3):

- We included a Dropout layer with a rate of 0.3 to regularize the model and prevent overfitting. Dropout encourages the model to learn more robust and generalized features by randomly setting a fraction of the input units to 0 during training. This reduces the reliance on specific neurons and enhances the model's ability to generalize to unseen data, leading to better performance on the validation set.

✔ Dense Layer (64 units, ReLU):

- We used a Dense layer with 64 units as the second hidden layer to gradually compress the learned representations and extract higher-level features. The reduced number of units helps in capturing more abstract and discriminative patterns relevant to the classification task. ReLU activation is used again for non-linearity, enabling the model to learn complex relationships between the features.

✔ Dense Layer (32 units, ReLU):

- We included a final Dense layer with 32 units to learn the most discriminative and compact representations required for the classification task. The further reduction in the number of units allows the model to focus on the essential features while reducing the risk of overfitting. ReLU activation helps in capturing non-linear relationships and maintaining the model's expressive power.

✔ Output Layers (Shape and Type):

- We used two separate output layers, one for shape classification (5 units) and another for type classification (17 units), because the task involves predicting both the shape and type of road signs. The softmax activation function is applied in each output layer to produce probability distributions over the respective classes. Softmax ensures that the outputs sum up to 1 and can be interpreted as class probabilities, providing a clear and interpretable output for the classification task.

4.1-1 Incremental Improvements

In [28…

```
from tensorflow.keras.layers import Dense, Flatten, Dropout, BatchNormalization, LeakyReLU
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

# Define the input shape and number of classes
input_shape = (28, 28, 3)
num_shape_classes = len(shape_encoder.classes_)
num_type_classes = len(type_encoder.classes_)
```

```
# Load image data and labels
X_train = np.array([img_to_array(load_img(path, target_size=input_shape)) / 255.0 for path in df.iloc[train_ind
y_train_shape = np.array(df.iloc[train_indices]['shape_label'])
y_train_type = np.array(df.iloc[train_indices]['type_label'])

X_val = np.array([img_to_array(load_img(path, target_size=input_shape)) / 255.0 for path in df.iloc[val_indices
y_val_shape = np.array(df.iloc[val_indices]['shape_label'])
y_val_type = np.array(df.iloc[val_indices]['type_label'])

# Create the model architecture
inputs = keras.Input(shape=input_shape)
x = Flatten()(inputs)
x = Dense(512)(x)
x = BatchNormalization()(x)
x = LeakyReLU(alpha=0.1)(x)
x = Dropout(0.4)(x)
x = Dense(256)(x)
x = BatchNormalization()(x)
x = LeakyReLU(alpha=0.1)(x)
x = Dropout(0.4)(x)
x = Dense(128)(x)
x = BatchNormalization()(x)
x = LeakyReLU(alpha=0.1)(x)
x = Dropout(0.4)(x)

# Output layers for shape and type
shape_output = Dense(num_shape_classes, activation='softmax', name='shape')(x)
type_output = Dense(num_type_classes, activation='softmax', name='type')(x)

# Create the model
model = Model(inputs=inputs, outputs=[shape_output, type_output])

# Compile the model with Adam optimizer and a lower learning rate
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer,
              loss={'shape': 'sparse_categorical_crossentropy',
                    'type': 'sparse_categorical_crossentropy'},
              metrics=['accuracy'])

# Define a learning rate scheduler
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, verbose=1, min_lr=1e-6)

# Train the model
history = model.fit(X_train, {'shape': y_train_shape, 'type': y_train_type},
                    validation_data=(X_val, {'shape': y_val_shape, 'type': y_val_type}),
                    epochs=50, batch_size=64, callbacks=[lr_scheduler])
```

```
Epoch 1/50
47/47 [==============================] - 3s 34ms/step - loss: 2.9857 - shape_loss: 1.0496 - type_loss: 1.9360 -
shape_accuracy: 0.6161 - type_accuracy: 0.4376 - val_loss: 2.0975 - val_shape_loss: 0.6226 - val_type_loss: 1.47
50 - val_shape_accuracy: 0.8149 - val_type_accuracy: 0.6257 - lr: 0.0010
Epoch 2/50
47/47 [==============================] - 1s 30ms/step - loss: 1.4434 - shape_loss: 0.4333 - type_loss: 1.0101 -
shape_accuracy: 0.8638 - type_accuracy: 0.7178 - val_loss: 1.1870 - val_shape_loss: 0.3671 - val_type_loss: 0.81
99 - val_shape_accuracy: 0.8973 - val_type_accuracy: 0.8311 - lr: 0.0010
Epoch 3/50
47/47 [==============================] - 1s 31ms/step - loss: 0.9915 - shape_loss: 0.2882 - type_loss: 0.7033 -
shape_accuracy: 0.9155 - type_accuracy: 0.8104 - val_loss: 0.8428 - val_shape_loss: 0.2437 - val_type_loss: 0.59
91 - val_shape_accuracy: 0.9324 - val_type_accuracy: 0.8595 - lr: 0.0010
Epoch 4/50
47/47 [==============================] - 1s 26ms/step - loss: 0.7856 - shape_loss: 0.2288 - type_loss: 0.5569 -
shape_accuracy: 0.9277 - type_accuracy: 0.8547 - val_loss: 0.8753 - val_shape_loss: 0.3380 - val_type_loss: 0.53
73 - val_shape_accuracy: 0.8730 - val_type_accuracy: 0.8459 - lr: 0.0010
Epoch 5/50
47/47 [==============================] - 1s 30ms/step - loss: 0.6222 - shape_loss: 0.1813 - type_loss: 0.4409 -
shape_accuracy: 0.9425 - type_accuracy: 0.8854 - val_loss: 0.6839 - val_shape_loss: 0.1741 - val_type_loss: 0.50
97 - val_shape_accuracy: 0.9446 - val_type_accuracy: 0.8392 - lr: 0.0010
Epoch 6/50
47/47 [==============================] - 1s 28ms/step - loss: 0.5819 - shape_loss: 0.1851 - type_loss: 0.3969 -
shape_accuracy: 0.9425 - type_accuracy: 0.8959 - val_loss: 1.5791 - val_shape_loss: 0.8048 - val_type_loss: 0.77
42 - val_shape_accuracy: 0.7068 - val_type_accuracy: 0.7824 - lr: 0.0010
Epoch 7/50
47/47 [==============================] - 1s 24ms/step - loss: 0.5078 - shape_loss: 0.1605 - type_loss: 0.3472 -
shape_accuracy: 0.9449 - type_accuracy: 0.9017 - val_loss: 0.6768 - val_shape_loss: 0.3054 - val_type_loss: 0.37
13 - val_shape_accuracy: 0.8824 - val_type_accuracy: 0.9027 - lr: 0.0010
Epoch 8/50
47/47 [==============================] - 1s 27ms/step - loss: 0.4664 - shape_loss: 0.1463 - type_loss: 0.3200 -
shape_accuracy: 0.9557 - type_accuracy: 0.9081 - val_loss: 0.4642 - val_shape_loss: 0.1490 - val_type_loss: 0.31
52 - val_shape_accuracy: 0.9473 - val_type_accuracy: 0.9108 - lr: 0.0010
Epoch 9/50
47/47 [==============================] - 1s 28ms/step - loss: 0.4289 - shape_loss: 0.1204 - type_loss: 0.3085 -
shape_accuracy: 0.9628 - type_accuracy: 0.9179 - val_loss: 0.3197 - val_shape_loss: 0.1023 - val_type_loss: 0.21
75 - val_shape_accuracy: 0.9662 - val_type_accuracy: 0.9432 - lr: 0.0010
Epoch 10/50
47/47 [==============================] - 1s 26ms/step - loss: 0.3971 - shape_loss: 0.1143 - type_loss: 0.2828 -
shape_accuracy: 0.9655 - type_accuracy: 0.9121 - val_loss: 0.3465 - val_shape_loss: 0.1153 - val_type_loss: 0.23
13 - val_shape_accuracy: 0.9662 - val_type_accuracy: 0.9405 - lr: 0.0010
Epoch 11/50
47/47 [==============================] - 1s 29ms/step - loss: 0.3286 - shape_loss: 0.1004 - type_loss: 0.2282 -
shape_accuracy: 0.9676 - type_accuracy: 0.9341 - val_loss: 0.4171 - val_shape_loss: 0.1538 - val_type_loss: 0.26
34 - val_shape_accuracy: 0.9554 - val_type_accuracy: 0.9176 - lr: 0.0010
Epoch 12/50
46/47 [============================>.] - ETA: 0s - loss: 0.3117 - shape_loss: 0.0929 - type_loss: 0.2187 - shape
_accuracy: 0.9721 - type_accuracy: 0.9399
Epoch 12: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
47/47 [==============================] - 1s 27ms/step - loss: 0.3131 - shape_loss: 0.0931 - type_loss: 0.2200 -
shape_accuracy: 0.9719 - type_accuracy: 0.9395 - val_loss: 0.3970 - val_shape_loss: 0.1095 - val_type_loss: 0.28
75 - val_shape_accuracy: 0.9716 - val_type_accuracy: 0.9135 - lr: 0.0010
Epoch 13/50
47/47 [==============================] - 1s 25ms/step - loss: 0.2773 - shape_loss: 0.0875 - type_loss: 0.1898 -
shape_accuracy: 0.9726 - type_accuracy: 0.9466 - val_loss: 0.3089 - val_shape_loss: 0.1151 - val_type_loss: 0.19
37 - val_shape_accuracy: 0.9581 - val_type_accuracy: 0.9378 - lr: 5.0000e-04
Epoch 14/50
47/47 [==============================] - 1s 27ms/step - loss: 0.2282 - shape_loss: 0.0729 - type_loss: 0.1553 -
shape_accuracy: 0.9794 - type_accuracy: 0.9584 - val_loss: 0.2949 - val_shape_loss: 0.1055 - val_type_loss: 0.18
94 - val_shape_accuracy: 0.9622 - val_type_accuracy: 0.9527 - lr: 5.0000e-04
Epoch 15/50
47/47 [==============================] - 1s 25ms/step - loss: 0.2467 - shape_loss: 0.0767 - type_loss: 0.1701 -
shape_accuracy: 0.9760 - type_accuracy: 0.9507 - val_loss: 0.3488 - val_shape_loss: 0.1143 - val_type_loss: 0.23
45 - val_shape_accuracy: 0.9689 - val_type_accuracy: 0.9338 - lr: 5.0000e-04
Epoch 16/50
47/47 [==============================] - 1s 29ms/step - loss: 0.2315 - shape_loss: 0.0703 - type_loss: 0.1613 -
shape_accuracy: 0.9801 - type_accuracy: 0.9530 - val_loss: 0.2455 - val_shape_loss: 0.0885 - val_type_loss: 0.15
70 - val_shape_accuracy: 0.9689 - val_type_accuracy: 0.9662 - lr: 5.0000e-04
Epoch 17/50
47/47 [==============================] - 1s 30ms/step - loss: 0.2050 - shape_loss: 0.0692 - type_loss: 0.1358 -
shape_accuracy: 0.9784 - type_accuracy: 0.9652 - val_loss: 0.2458 - val_shape_loss: 0.0741 - val_type_loss: 0.17
16 - val_shape_accuracy: 0.9784 - val_type_accuracy: 0.9608 - lr: 5.0000e-04
Epoch 18/50
47/47 [==============================] - 1s 25ms/step - loss: 0.1984 - shape_loss: 0.0607 - type_loss: 0.1377 -
shape_accuracy: 0.9804 - type_accuracy: 0.9652 - val_loss: 0.3630 - val_shape_loss: 0.1421 - val_type_loss: 0.22
09 - val_shape_accuracy: 0.9500 - val_type_accuracy: 0.9392 - lr: 5.0000e-04
Epoch 19/50
47/47 [==============================] - ETA: 0s - loss: 0.2050 - shape_loss: 0.0647 - type_loss: 0.1403 - shape
_accuracy: 0.9797 - type_accuracy: 0.9605
Epoch 19: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
47/47 [==============================] - 1s 30ms/step - loss: 0.2050 - shape_loss: 0.0647 - type_loss: 0.1403 -
shape_accuracy: 0.9797 - type_accuracy: 0.9605 - val_loss: 0.2967 - val_shape_loss: 0.1026 - val_type_loss: 0.19
41 - val_shape_accuracy: 0.9662 - val_type_accuracy: 0.9459 - lr: 5.0000e-04
Epoch 20/50
47/47 [==============================] - 1s 26ms/step - loss: 0.1465 - shape_loss: 0.0363 - type_loss: 0.1102 -
shape_accuracy: 0.9926 - type_accuracy: 0.9709 - val_loss: 0.2202 - val_shape_loss: 0.0809 - val_type_loss: 0.13
93 - val_shape_accuracy: 0.9730 - val_type_accuracy: 0.9622 - lr: 2.5000e-04
Epoch 21/50
47/47 [==============================] - 1s 25ms/step - loss: 0.1549 - shape_loss: 0.0472 - type_loss: 0.1077 -
shape_accuracy: 0.9872 - type_accuracy: 0.9740 - val_loss: 0.2029 - val_shape_loss: 0.0675 - val_type_loss: 0.13
53 - val_shape_accuracy: 0.9797 - val_type_accuracy: 0.9676 - lr: 2.5000e-04
Epoch 22/50
47/47 [==============================] - 1s 30ms/step - loss: 0.1314 - shape_loss: 0.0359 - type_loss: 0.0955 -
shape_accuracy: 0.9905 - type_accuracy: 0.9767 - val_loss: 0.1864 - val_shape_loss: 0.0662 - val_type_loss: 0.12
```

```
02 - val_shape_accuracy: 0.9730 - val_type_accuracy: 0.9730 - lr: 2.5000e-04
Epoch 23/50
47/47 [==============================] - 1s 28ms/step - loss: 0.1369 - shape_loss: 0.0372 - type_loss: 0.0996 -
shape_accuracy: 0.9895 - type_accuracy: 0.9719 - val_loss: 0.1988 - val_shape_loss: 0.0723 - val_type_loss: 0.12
65 - val_shape_accuracy: 0.9757 - val_type_accuracy: 0.9622 - lr: 2.5000e-04
Epoch 24/50
47/47 [==============================] - 1s 29ms/step - loss: 0.1298 - shape_loss: 0.0359 - type_loss: 0.0938 -
shape_accuracy: 0.9909 - type_accuracy: 0.9801 - val_loss: 0.2001 - val_shape_loss: 0.0678 - val_type_loss: 0.13
24 - val_shape_accuracy: 0.9797 - val_type_accuracy: 0.9676 - lr: 2.5000e-04
Epoch 25/50
46/47 [============================>.] - ETA: 0s - loss: 0.1645 - shape_loss: 0.0505 - type_loss: 0.1140 - shape
_accuracy: 0.9844 - type_accuracy: 0.9660
Epoch 25: ReduceLROnPlateau reducing learning rate to 0.00012500000059371814.
47/47 [==============================] - 1s 32ms/step - loss: 0.1646 - shape_loss: 0.0505 - type_loss: 0.1141 -
shape_accuracy: 0.9845 - type_accuracy: 0.9659 - val_loss: 0.2036 - val_shape_loss: 0.0762 - val_type_loss: 0.12
74 - val_shape_accuracy: 0.9770 - val_type_accuracy: 0.9703 - lr: 2.5000e-04
Epoch 26/50
47/47 [==============================] - 1s 28ms/step - loss: 0.1359 - shape_loss: 0.0407 - type_loss: 0.0952 -
shape_accuracy: 0.9878 - type_accuracy: 0.9723 - val_loss: 0.1777 - val_shape_loss: 0.0611 - val_type_loss: 0.11
66 - val_shape_accuracy: 0.9811 - val_type_accuracy: 0.9716 - lr: 1.2500e-04
Epoch 27/50
47/47 [==============================] - 1s 26ms/step - loss: 0.1128 - shape_loss: 0.0308 - type_loss: 0.0820 -
shape_accuracy: 0.9909 - type_accuracy: 0.9774 - val_loss: 0.1893 - val_shape_loss: 0.0714 - val_type_loss: 0.11
79 - val_shape_accuracy: 0.9784 - val_type_accuracy: 0.9689 - lr: 1.2500e-04
Epoch 28/50
47/47 [==============================] - 1s 24ms/step - loss: 0.1076 - shape_loss: 0.0285 - type_loss: 0.0791 -
shape_accuracy: 0.9949 - type_accuracy: 0.9780 - val_loss: 0.1707 - val_shape_loss: 0.0577 - val_type_loss: 0.11
31 - val_shape_accuracy: 0.9838 - val_type_accuracy: 0.9703 - lr: 1.2500e-04
Epoch 29/50
47/47 [==============================] - 1s 27ms/step - loss: 0.1044 - shape_loss: 0.0306 - type_loss: 0.0738 -
shape_accuracy: 0.9922 - type_accuracy: 0.9828 - val_loss: 0.1724 - val_shape_loss: 0.0568 - val_type_loss: 0.11
56 - val_shape_accuracy: 0.9811 - val_type_accuracy: 0.9649 - lr: 1.2500e-04
Epoch 30/50
47/47 [==============================] - 1s 26ms/step - loss: 0.1166 - shape_loss: 0.0343 - type_loss: 0.0823 -
shape_accuracy: 0.9902 - type_accuracy: 0.9774 - val_loss: 0.1708 - val_shape_loss: 0.0595 - val_type_loss: 0.11
13 - val_shape_accuracy: 0.9824 - val_type_accuracy: 0.9730 - lr: 1.2500e-04
Epoch 31/50
46/47 [============================>.] - ETA: 0s - loss: 0.1003 - shape_loss: 0.0260 - type_loss: 0.0743 - shape
_accuracy: 0.9935 - type_accuracy: 0.9800
Epoch 31: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
47/47 [==============================] - 1s 28ms/step - loss: 0.1006 - shape_loss: 0.0259 - type_loss: 0.0747 -
shape_accuracy: 0.9936 - type_accuracy: 0.9797 - val_loss: 0.1792 - val_shape_loss: 0.0651 - val_type_loss: 0.11
41 - val_shape_accuracy: 0.9770 - val_type_accuracy: 0.9703 - lr: 1.2500e-04
Epoch 32/50
47/47 [==============================] - 1s 28ms/step - loss: 0.1011 - shape_loss: 0.0314 - type_loss: 0.0697 -
shape_accuracy: 0.9932 - type_accuracy: 0.9855 - val_loss: 0.1663 - val_shape_loss: 0.0570 - val_type_loss: 0.10
93 - val_shape_accuracy: 0.9838 - val_type_accuracy: 0.9730 - lr: 6.2500e-05
Epoch 33/50
47/47 [==============================] - 1s 28ms/step - loss: 0.1010 - shape_loss: 0.0290 - type_loss: 0.0720 -
shape_accuracy: 0.9946 - type_accuracy: 0.9828 - val_loss: 0.1715 - val_shape_loss: 0.0608 - val_type_loss: 0.11
08 - val_shape_accuracy: 0.9824 - val_type_accuracy: 0.9716 - lr: 6.2500e-05
Epoch 34/50
47/47 [==============================] - 1s 31ms/step - loss: 0.0963 - shape_loss: 0.0287 - type_loss: 0.0676 -
shape_accuracy: 0.9919 - type_accuracy: 0.9828 - val_loss: 0.1570 - val_shape_loss: 0.0557 - val_type_loss: 0.10
12 - val_shape_accuracy: 0.9824 - val_type_accuracy: 0.9703 - lr: 6.2500e-05
Epoch 35/50
47/47 [==============================] - 1s 27ms/step - loss: 0.1042 - shape_loss: 0.0302 - type_loss: 0.0740 -
shape_accuracy: 0.9916 - type_accuracy: 0.9801 - val_loss: 0.1618 - val_shape_loss: 0.0569 - val_type_loss: 0.10
48 - val_shape_accuracy: 0.9838 - val_type_accuracy: 0.9757 - lr: 6.2500e-05
Epoch 36/50
47/47 [==============================] - 1s 28ms/step - loss: 0.0895 - shape_loss: 0.0276 - type_loss: 0.0619 -
shape_accuracy: 0.9926 - type_accuracy: 0.9855 - val_loss: 0.1526 - val_shape_loss: 0.0523 - val_type_loss: 0.10
04 - val_shape_accuracy: 0.9811 - val_type_accuracy: 0.9743 - lr: 6.2500e-05
Epoch 37/50
47/47 [==============================] - 1s 25ms/step - loss: 0.0842 - shape_loss: 0.0210 - type_loss: 0.0632 -
shape_accuracy: 0.9956 - type_accuracy: 0.9872 - val_loss: 0.1720 - val_shape_loss: 0.0600 - val_type_loss: 0.11
20 - val_shape_accuracy: 0.9838 - val_type_accuracy: 0.9703 - lr: 6.2500e-05
Epoch 38/50
47/47 [==============================] - 1s 25ms/step - loss: 0.0886 - shape_loss: 0.0255 - type_loss: 0.0631 -
shape_accuracy: 0.9926 - type_accuracy: 0.9824 - val_loss: 0.1571 - val_shape_loss: 0.0525 - val_type_loss: 0.10
46 - val_shape_accuracy: 0.9838 - val_type_accuracy: 0.9730 - lr: 6.2500e-05
Epoch 39/50
47/47 [==============================] - ETA: 0s - loss: 0.0884 - shape_loss: 0.0260 - type_loss: 0.0625 - shape
_accuracy: 0.9936 - type_accuracy: 0.9838
Epoch 39: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
47/47 [==============================] - 1s 30ms/step - loss: 0.0884 - shape_loss: 0.0260 - type_loss: 0.0625 -
shape_accuracy: 0.9936 - type_accuracy: 0.9838 - val_loss: 0.1533 - val_shape_loss: 0.0510 - val_type_loss: 0.10
23 - val_shape_accuracy: 0.9865 - val_type_accuracy: 0.9716 - lr: 6.2500e-05
Epoch 40/50
47/47 [==============================] - 1s 28ms/step - loss: 0.1007 - shape_loss: 0.0324 - type_loss: 0.0683 -
shape_accuracy: 0.9922 - type_accuracy: 0.9831 - val_loss: 0.1596 - val_shape_loss: 0.0559 - val_type_loss: 0.10
37 - val_shape_accuracy: 0.9851 - val_type_accuracy: 0.9730 - lr: 3.1250e-05
Epoch 41/50
47/47 [==============================] - 1s 24ms/step - loss: 0.0883 - shape_loss: 0.0300 - type_loss: 0.0583 -
shape_accuracy: 0.9919 - type_accuracy: 0.9885 - val_loss: 0.1481 - val_shape_loss: 0.0509 - val_type_loss: 0.09
72 - val_shape_accuracy: 0.9824 - val_type_accuracy: 0.9757 - lr: 3.1250e-05
Epoch 42/50
47/47 [==============================] - 1s 25ms/step - loss: 0.0853 - shape_loss: 0.0225 - type_loss: 0.0628 -
shape_accuracy: 0.9946 - type_accuracy: 0.9865 - val_loss: 0.1495 - val_shape_loss: 0.0510 - val_type_loss: 0.09
84 - val_shape_accuracy: 0.9865 - val_type_accuracy: 0.9770 - lr: 3.1250e-05
Epoch 43/50
47/47 [==============================] - 1s 25ms/step - loss: 0.0889 - shape_loss: 0.0233 - type_loss: 0.0656 -
shape_accuracy: 0.9963 - type_accuracy: 0.9872 - val_loss: 0.1569 - val_shape_loss: 0.0527 - val_type_loss: 0.10
```

```
42 - val_shape_accuracy: 0.9865 - val_type_accuracy: 0.9743 - lr: 3.1250e-05
Epoch 44/50
47/47 [==============================] - ETA: 0s - loss: 0.0789 - shape_loss: 0.0229 - type_loss: 0.0560 - shape
_accuracy: 0.9959 - type_accuracy: 0.9878
Epoch 44: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
47/47 [==============================] - 1s 27ms/step - loss: 0.0789 - shape_loss: 0.0229 - type_loss: 0.0560 -
shape_accuracy: 0.9959 - type_accuracy: 0.9878 - val_loss: 0.1580 - val_shape_loss: 0.0541 - val_type_loss: 0.10
39 - val_shape_accuracy: 0.9865 - val_type_accuracy: 0.9716 - lr: 3.1250e-05
Epoch 45/50
47/47 [==============================] - 1s 26ms/step - loss: 0.0791 - shape_loss: 0.0219 - type_loss: 0.0572 -
shape_accuracy: 0.9946 - type_accuracy: 0.9875 - val_loss: 0.1547 - val_shape_loss: 0.0526 - val_type_loss: 0.10
21 - val_shape_accuracy: 0.9824 - val_type_accuracy: 0.9730 - lr: 1.5625e-05
Epoch 46/50
47/47 [==============================] - 2s 35ms/step - loss: 0.0814 - shape_loss: 0.0240 - type_loss: 0.0574 -
shape_accuracy: 0.9922 - type_accuracy: 0.9851 - val_loss: 0.1503 - val_shape_loss: 0.0504 - val_type_loss: 0.09
99 - val_shape_accuracy: 0.9851 - val_type_accuracy: 0.9730 - lr: 1.5625e-05
Epoch 47/50
46/47 [============================>.] - ETA: 0s - loss: 0.0805 - shape_loss: 0.0239 - type_loss: 0.0566 - shape
_accuracy: 0.9935 - type_accuracy: 0.9874
Epoch 47: ReduceLROnPlateau reducing learning rate to 7.812500371073838e-06.
47/47 [==============================] - 1s 25ms/step - loss: 0.0852 - shape_loss: 0.0252 - type_loss: 0.0600 -
shape_accuracy: 0.9929 - type_accuracy: 0.9861 - val_loss: 0.1504 - val_shape_loss: 0.0515 - val_type_loss: 0.09
89 - val_shape_accuracy: 0.9824 - val_type_accuracy: 0.9757 - lr: 1.5625e-05
Epoch 48/50
47/47 [==============================] - 1s 27ms/step - loss: 0.0842 - shape_loss: 0.0236 - type_loss: 0.0606 -
shape_accuracy: 0.9956 - type_accuracy: 0.9865 - val_loss: 0.1510 - val_shape_loss: 0.0514 - val_type_loss: 0.09
96 - val_shape_accuracy: 0.9865 - val_type_accuracy: 0.9743 - lr: 7.8125e-06
Epoch 49/50
47/47 [==============================] - 1s 22ms/step - loss: 0.0756 - shape_loss: 0.0204 - type_loss: 0.0553 -
shape_accuracy: 0.9970 - type_accuracy: 0.9878 - val_loss: 0.1516 - val_shape_loss: 0.0517 - val_type_loss: 0.09
98 - val_shape_accuracy: 0.9865 - val_type_accuracy: 0.9743 - lr: 7.8125e-06
Epoch 50/50
46/47 [============================>.] - ETA: 0s - loss: 0.0717 - shape_loss: 0.0217 - type_loss: 0.0500 - shape
_accuracy: 0.9966 - type_accuracy: 0.9891
Epoch 50: ReduceLROnPlateau reducing learning rate to 3.906250185536919e-06.
47/47 [==============================] - 1s 29ms/step - loss: 0.0777 - shape_loss: 0.0257 - type_loss: 0.0520 -
shape_accuracy: 0.9956 - type_accuracy: 0.9885 - val_loss: 0.1525 - val_shape_loss: 0.0522 - val_type_loss: 0.10
03 - val_shape_accuracy: 0.9865 - val_type_accuracy: 0.9730 - lr: 7.8125e-06
```

Justification for the updated MLP layers: ✔ Flatten Layer:

- The Flatten layer remains unchanged as it is necessary to convert the 2D input image into a 1D vector for the subsequent fully connected layers.

✔ Dense Layers (512, 256, 128 units):

- We increased the number of units in the dense layers to 512, 256, and 128, respectively, to enhance the model's capacity to learn more complex patterns and representations from the input data. The increased depth and width of the network allow it to capture intricate features and relationships at different levels of abstraction.

✔ Batch Normalization Layers:

- We added Batch Normalization layers after each dense layer to normalize the activations and improve the model's stability and convergence. Batch Normalization helps in reducing the internal covariate shift, which occurs when the distribution of inputs to a layer changes during training. By normalizing the activations, it allows the model to learn more efficiently and reduces the sensitivity to the choice of initialization.

✔ LeakyReLU Activation:

- We replaced the ReLU activation with LeakyReLU, which allows a small negative value for negative inputs. LeakyReLU addresses the "dying ReLU" problem, where neurons become inactive and stop learning when their inputs are consistently negative. By allowing a small negative slope, LeakyReLU enables the model to learn more complex and non-linear patterns, potentially improving its ability to capture subtle features in the input data.

✔ Dropout Layers (0.4):

- We increased the dropout rate from 0.3 to 0.4 to provide stronger regularization and prevent overfitting. Dropout helps in reducing the reliance on specific neurons and encourages the model to learn more robust and generalized features. By randomly setting a higher fraction of input units to 0 during training, dropout promotes the model to develop redundant and complementary representations, enhancing its ability to generalize well to unseen data.

✔ Adam Optimizer with Lower Learning Rate:

- We switched to the Adam optimizer with a lower learning rate of 0.001. Adam is an adaptive optimization algorithm that automatically adjusts the learning rate for each parameter based on its historical gradients. It combines the benefits of AdaGrad and RMSprop, adapting the learning rates to the geometry of the loss function. The lower learning rate allows for a more stable and controlled convergence, reducing the risk of overshooting the optimal solution.

✔ Learning Rate Scheduler:

- We introduced a learning rate scheduler (ReduceLROnPlateau) that reduces the learning rate by a factor of 0.5 if the validation loss does not improve for 3 epochs. This allows the model to fine-tune its weights and converge to a better solution. By dynamically adjusting the learning rate based on the validation performance, the scheduler helps in finding the optimal learning rate and prevents the model from getting stuck in suboptimal regions of the loss landscape.

Benefits of using Convolutional Neural Networks (CNNs) for image classification: ✔ Spatial Feature Extraction:

- CNNs are specifically designed to handle and exploit the spatial structure in images. They employ convolutional layers that learn local patterns and features by applying filters (kernels) to small regions of the input image. These filters slide over the image, capturing spatial dependencies and hierarchical representations at different scales. By preserving the spatial arrangement of pixels, CNNs can effectively learn and detect meaningful features such as edges, shapes, and textures, which are crucial for image classification tasks.

✔ Translation Invariance and Parameter Sharing:

- CNNs exhibit translation invariance, meaning they can recognize patterns and objects regardless of their position in the image. This is achieved through the use of shared weights (parameters) across the convolutional layers. Instead of learning separate weights for each location, CNNs learn a set of filters that are applied consistently across the entire image. This parameter sharing not only reduces the number of learnable parameters compared to fully connected networks but also allows the model to generalize well to variations in object locations. Additionally, the pooling layers in CNNs introduce spatial invariance, enabling the model to be robust to small translations and distortions in the input image.

4.2 Convolutional Neural Network (CNN)

How CNN differs from MLPs:
✔ Local Connectivity:

- Unlike MLPs, which are fully connected, CNNs exploit local connectivity by applying convolutional filters to small regions of the input. This allows CNNs to capture spatially local patterns and learn translation-invariant features, making them well-suited for image and pattern recognition tasks.

✔ Parameter Sharing:

- In CNNs, the weights of the convolutional filters are shared across different positions of the input. This parameter sharing reduces the number of learnable parameters compared to MLPs, making CNNs more computationally efficient and less prone to overfitting.

✔ Hierarchical Feature Learning:

- CNNs learn hierarchical representations of the input data through successive convolutional and pooling layers. Lower layers capture low-level features like edges and textures, while higher layers learn more abstract and complex features. This hierarchical learning enables CNNs to automatically extract relevant features from raw input data.

✔ Spatial Invariance:

- CNNs achieve spatial invariance through pooling operations, which downsample the feature maps and provide robustness to small translations and distortions in the input. Pooling helps CNNs to focus on the presence of features rather than their precise location, making them resilient to spatial variations.

In [29…

```
# Define the input shape and number of classes
input_shape = (28, 28, 3)
num_shape_classes = len(shape_encoder.classes_)
num_type_classes = len(type_encoder.classes_)

# Create the model architecture
inputs = keras.Input(shape=input_shape)
x = Conv2D(16, (3, 3), activation='relu')(inputs)
x = MaxPooling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu')(x)
x = MaxPooling2D((2, 2))(x)
x = Flatten()(x)
x = Dense(64, activation='relu')(x)

# Output layers for shape and type
shape_output = Dense(num_shape_classes, activation='softmax', name='shape')(x)
type_output = Dense(num_type_classes, activation='softmax', name='type')(x)

# Create the model
model = Model(inputs=inputs, outputs=[shape_output, type_output])

# Compile the model
model.compile(optimizer='adam',
              loss={'shape': 'sparse_categorical_crossentropy',
                    'type': 'sparse_categorical_crossentropy'},
              metrics=['accuracy'])

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_indices) // 32,
    validation_data=val_generator,
    validation_steps=np.ceil(len(val_indices) / 32).astype(int),
    epochs=20
)
```

```
Epoch 1/20
2024-05-19 18:42:34.476369: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Ex
ecutor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You
must feed a value for placeholder tensor 'Placeholder/_0' with dtype int32
        [[{{node Placeholder/_0}}]]
```

```
92/92 [==============================] - ETA: 0s - loss: 2.9115 - shape_loss: 0.9187 - type_loss: 1.9927 - sha
pe_accuracy: 0.6744 - type_accuracy: 0.4199
2024-05-19 18:42:52.524358: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Ex
ecutor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You
must feed a value for placeholder tensor 'Placeholder/_0' with dtype int32
         [[{{node Placeholder/_0}}]]
92/92 [==============================] - 19s 197ms/step - loss: 2.9115 - shape_loss: 0.9187 - type_loss: 1.992
7 - shape_accuracy: 0.6744 - type_accuracy: 0.4199 - val_loss: 1.6465 - val_shape_loss: 0.5193 - val_type_loss
: 1.1272 - val_shape_accuracy: 0.8284 - val_type_accuracy: 0.6946
Epoch 2/20
92/92 [==============================] - 21s 231ms/step - loss: 0.9620 - shape_loss: 0.2972 - type_loss: 0.664
8 - shape_accuracy: 0.9163 - type_accuracy: 0.8360 - val_loss: 0.6637 - val_shape_loss: 0.2285 - val_type_loss
: 0.4352 - val_shape_accuracy: 0.9392 - val_type_accuracy: 0.8946
Epoch 3/20
92/92 [==============================] - 19s 209ms/step - loss: 0.4854 - shape_loss: 0.1448 - type_loss: 0.340
6 - shape_accuracy: 0.9587 - type_accuracy: 0.9132 - val_loss: 0.4467 - val_shape_loss: 0.1485 - val_type_loss
: 0.2982 - val_shape_accuracy: 0.9500 - val_type_accuracy: 0.9162
Epoch 4/20
92/92 [==============================] - 21s 226ms/step - loss: 0.3181 - shape_loss: 0.0889 - type_loss: 0.229
2 - shape_accuracy: 0.9788 - type_accuracy: 0.9429 - val_loss: 0.2970 - val_shape_loss: 0.0920 - val_type_loss
: 0.2050 - val_shape_accuracy: 0.9676 - val_type_accuracy: 0.9432
Epoch 5/20
92/92 [==============================] - 15s 169ms/step - loss: 0.2212 - shape_loss: 0.0646 - type_loss: 0.156
6 - shape_accuracy: 0.9846 - type_accuracy: 0.9597 - val_loss: 0.2348 - val_shape_loss: 0.0583 - val_type_loss
: 0.1764 - val_shape_accuracy: 0.9797 - val_type_accuracy: 0.9500
Epoch 6/20
92/92 [==============================] - 19s 205ms/step - loss: 0.1727 - shape_loss: 0.0479 - type_loss: 0.124
7 - shape_accuracy: 0.9867 - type_accuracy: 0.9703 - val_loss: 0.3741 - val_shape_loss: 0.2103 - val_type_loss
: 0.1639 - val_shape_accuracy: 0.9351 - val_type_accuracy: 0.9514
Epoch 7/20
92/92 [==============================] - 19s 211ms/step - loss: 0.1446 - shape_loss: 0.0415 - type_loss: 0.103
1 - shape_accuracy: 0.9884 - type_accuracy: 0.9727 - val_loss: 0.1947 - val_shape_loss: 0.0562 - val_type_loss
: 0.1384 - val_shape_accuracy: 0.9824 - val_type_accuracy: 0.9541
Epoch 8/20
92/92 [==============================] - 20s 220ms/step - loss: 0.0968 - shape_loss: 0.0247 - type_loss: 0.072
0 - shape_accuracy: 0.9952 - type_accuracy: 0.9819 - val_loss: 0.1442 - val_shape_loss: 0.0486 - val_type_loss
: 0.0956 - val_shape_accuracy: 0.9865 - val_type_accuracy: 0.9730
Epoch 9/20
92/92 [==============================] - 19s 207ms/step - loss: 0.0838 - shape_loss: 0.0223 - type_loss: 0.061
5 - shape_accuracy: 0.9945 - type_accuracy: 0.9809 - val_loss: 0.1502 - val_shape_loss: 0.0352 - val_type_loss
: 0.1150 - val_shape_accuracy: 0.9905 - val_type_accuracy: 0.9662
Epoch 10/20
92/92 [==============================] - 19s 207ms/step - loss: 0.0647 - shape_loss: 0.0158 - type_loss: 0.048
9 - shape_accuracy: 0.9956 - type_accuracy: 0.9884 - val_loss: 0.1192 - val_shape_loss: 0.0331 - val_type_loss
: 0.0861 - val_shape_accuracy: 0.9865 - val_type_accuracy: 0.9797
Epoch 11/20
92/92 [==============================] - 22s 238ms/step - loss: 0.0426 - shape_loss: 0.0097 - type_loss: 0.033
0 - shape_accuracy: 0.9983 - type_accuracy: 0.9918 - val_loss: 0.1278 - val_shape_loss: 0.0270 - val_type_loss
: 0.1009 - val_shape_accuracy: 0.9878 - val_type_accuracy: 0.9676
Epoch 12/20
92/92 [==============================] - 21s 231ms/step - loss: 0.0401 - shape_loss: 0.0102 - type_loss: 0.029
9 - shape_accuracy: 0.9986 - type_accuracy: 0.9932 - val_loss: 0.1106 - val_shape_loss: 0.0358 - val_type_loss
: 0.0748 - val_shape_accuracy: 0.9905 - val_type_accuracy: 0.9797
Epoch 13/20
92/92 [==============================] - 24s 261ms/step - loss: 0.0308 - shape_loss: 0.0068 - type_loss: 0.024
1 - shape_accuracy: 0.9990 - type_accuracy: 0.9942 - val_loss: 0.1765 - val_shape_loss: 0.0343 - val_type_loss
: 0.1422 - val_shape_accuracy: 0.9892 - val_type_accuracy: 0.9527
Epoch 14/20
92/92 [==============================] - 19s 205ms/step - loss: 0.0349 - shape_loss: 0.0065 - type_loss: 0.028
4 - shape_accuracy: 0.9993 - type_accuracy: 0.9928 - val_loss: 0.1290 - val_shape_loss: 0.0267 - val_type_loss
: 0.1023 - val_shape_accuracy: 0.9892 - val_type_accuracy: 0.9676
Epoch 15/20
92/92 [==============================] - 21s 226ms/step - loss: 0.0257 - shape_loss: 0.0049 - type_loss: 0.020
8 - shape_accuracy: 0.9997 - type_accuracy: 0.9959 - val_loss: 0.1552 - val_shape_loss: 0.0535 - val_type_loss
: 0.1017 - val_shape_accuracy: 0.9851 - val_type_accuracy: 0.9676
Epoch 16/20
92/92 [==============================] - 18s 198ms/step - loss: 0.0287 - shape_loss: 0.0066 - type_loss: 0.022
1 - shape_accuracy: 0.9983 - type_accuracy: 0.9952 - val_loss: 0.0899 - val_shape_loss: 0.0220 - val_type_loss
: 0.0679 - val_shape_accuracy: 0.9919 - val_type_accuracy: 0.9811
Epoch 17/20
92/92 [==============================] - 20s 220ms/step - loss: 0.0193 - shape_loss: 0.0029 - type_loss: 0.016
3 - shape_accuracy: 0.9997 - type_accuracy: 0.9966 - val_loss: 0.1074 - val_shape_loss: 0.0254 - val_type_loss
: 0.0820 - val_shape_accuracy: 0.9905 - val_type_accuracy: 0.9743
Epoch 18/20
92/92 [==============================] - 19s 206ms/step - loss: 0.0391 - shape_loss: 0.0066 - type_loss: 0.032
4 - shape_accuracy: 0.9986 - type_accuracy: 0.9918 - val_loss: 0.1171 - val_shape_loss: 0.0332 - val_type_loss
: 0.0839 - val_shape_accuracy: 0.9905 - val_type_accuracy: 0.9784
Epoch 19/20
92/92 [==============================] - 21s 226ms/step - loss: 0.0114 - shape_loss: 0.0024 - type_loss: 0.009
0 - shape_accuracy: 1.0000 - type_accuracy: 0.9980 - val_loss: 0.1092 - val_shape_loss: 0.0287 - val_type_loss
: 0.0805 - val_shape_accuracy: 0.9932 - val_type_accuracy: 0.9757
Epoch 20/20
92/92 [==============================] - 19s 211ms/step - loss: 0.0097 - shape_loss: 0.0018 - type_loss: 0.007
9 - shape_accuracy: 1.0000 - type_accuracy: 0.9990 - val_loss: 0.1416 - val_shape_loss: 0.0377 - val_type_loss
: 0.1039 - val_shape_accuracy: 0.9878 - val_type_accuracy: 0.9784
```

- When constructing our CNN architecture, we followed an approach similar to our MLP findings to create a simplified version that works effectively.
- We used two convolutional layers:
  - First layer: 16 filters
  - Second layer: 32 filters
  - Followed by max pooling layers
  - We predicted this configuration would be effective in capturing relevant features while

maintaining computational efficiency.
- The output of the convolutional layers is flattened and passed through a dense layer with 64 units, allowing for the learning of high-level representations.
- We incorporated two separate output layers for shape and type classification, both using the softmax activation function.
  - Softmax enables the model to produce probability distributions over the respective classes.
- We compiled the model with:
  - Adam optimiser
  - Sparse categorical cross-entropy loss
  - This combination has been shown to work well for multi-class classification tasks.

4.2-1 Incremental Improvement

In [30…

```
from tensorflow.keras.regularizers import l2
from tensorflow.keras.layers import BatchNormalization, LeakyReLU
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

# Define the input shape and number of classes
input_shape = (28, 28, 3)
num_shape_classes = len(shape_encoder.classes_)
num_type_classes = len(type_encoder.classes_)

# Create the model architecture
inputs = keras.Input(shape=input_shape)
x = Conv2D(32, (3, 3), activation='relu', kernel_regularizer=l2(0.01))(inputs)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu', kernel_regularizer=l2(0.01))(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)
x = Conv2D(128, (3, 3), activation=LeakyReLU(alpha=0.1), kernel_regularizer=l2(0.01))(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)
x = Flatten()(x)
x = Dense(256, activation='relu', kernel_regularizer=l2(0.01))(x)
x = Dropout(0.6)(x)

# Output layers for shape and type
shape_output = Dense(num_shape_classes, activation='softmax', name='shape')(x)
type_output = Dense(num_type_classes, activation='softmax', name='type')(x)

# Create the model
model = Model(inputs=inputs, outputs=[shape_output, type_output])

# Compile the model with Adam optimizer and adjusted learning rate
model.compile(optimizer='adam',
              loss={'shape': 'sparse_categorical_crossentropy',
                    'type': 'sparse_categorical_crossentropy'},
              metrics=['accuracy'])

# Define callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5, min_lr=1e-6)

# Train the model with data augmentation and callbacks
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_indices) // 32,
    validation_data=val_generator,
    validation_steps=np.ceil(len(val_indices) / 32).astype(int),
    epochs=50,
    callbacks=[early_stopping, reduce_lr]
)
```

```
Epoch 1/50
2024-05-19 18:51:12.959399: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) E
xecutor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: Y
ou must feed a value for placeholder tensor 'Placeholder/_0' with dtype int32
         [[{{node Placeholder/_0}}]]
92/92 [==============================] - ETA: 0s - loss: 4.5683 - shape_loss: 0.5665 - type_loss: 1.1379 - sh
ape_accuracy: 0.8025 - type_accuracy: 0.6737
2024-05-19 18:51:26.813416: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) E
xecutor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: Y
ou must feed a value for placeholder tensor 'Placeholder/_0' with dtype int32
         [[{{node Placeholder/_0}}]]
```

```
92/92 [==============================] - 15s 150ms/step - loss: 4.5683 - shape_loss: 0.5665 - type_loss: 1.13
79 - shape_accuracy: 0.8025 - type_accuracy: 0.6737 - val_loss: 6.7388 - val_shape_loss: 1.3025 - val_type_lo
ss: 2.7820 - val_shape_accuracy: 0.4622 - val_type_accuracy: 0.0649 - lr: 0.0010
Epoch 2/50
92/92 [==============================] - 12s 126ms/step - loss: 2.7857 - shape_loss: 0.1089 - type_loss: 0.25
55 - shape_accuracy: 0.9648 - type_accuracy: 0.9279 - val_loss: 6.4764 - val_shape_loss: 1.3777 - val_type_lo
ss: 2.9159 - val_shape_accuracy: 0.4662 - val_type_accuracy: 0.0662 - lr: 0.0010
Epoch 3/50
92/92 [==============================] - 11s 114ms/step - loss: 2.1820 - shape_loss: 0.0592 - type_loss: 0.15
30 - shape_accuracy: 0.9809 - type_accuracy: 0.9566 - val_loss: 5.5540 - val_shape_loss: 1.5435 - val_type_lo
ss: 2.2464 - val_shape_accuracy: 0.5189 - val_type_accuracy: 0.2851 - lr: 0.0010
Epoch 4/50
92/92 [==============================] - 14s 155ms/step - loss: 1.7091 - shape_loss: 0.0315 - type_loss: 0.09
89 - shape_accuracy: 0.9921 - type_accuracy: 0.9720 - val_loss: 3.4236 - val_shape_loss: 0.7336 - val_type_lo
ss: 1.2904 - val_shape_accuracy: 0.7243 - val_type_accuracy: 0.5892 - lr: 0.0010
Epoch 5/50
92/92 [==============================] - 11s 120ms/step - loss: 1.3222 - shape_loss: 0.0201 - type_loss: 0.05
79 - shape_accuracy: 0.9956 - type_accuracy: 0.9857 - val_loss: 1.7581 - val_shape_loss: 0.2349 - val_type_lo
ss: 0.4253 - val_shape_accuracy: 0.9189 - val_type_accuracy: 0.8703 - lr: 0.0010
Epoch 6/50
92/92 [==============================] - 10s 107ms/step - loss: 1.0727 - shape_loss: 0.0269 - type_loss: 0.05
95 - shape_accuracy: 0.9911 - type_accuracy: 0.9853 - val_loss: 1.3456 - val_shape_loss: 0.1313 - val_type_lo
ss: 0.3290 - val_shape_accuracy: 0.9554 - val_type_accuracy: 0.8986 - lr: 0.0010
Epoch 7/50
92/92 [==============================] - 10s 108ms/step - loss: 0.8565 - shape_loss: 0.0142 - type_loss: 0.04
69 - shape_accuracy: 0.9973 - type_accuracy: 0.9880 - val_loss: 0.8557 - val_shape_loss: 0.0450 - val_type_lo
ss: 0.1020 - val_shape_accuracy: 0.9824 - val_type_accuracy: 0.9716 - lr: 0.0010
Epoch 8/50
92/92 [==============================] - 9s 91ms/step - loss: 0.6991 - shape_loss: 0.0158 - type_loss: 0.0445
- shape_accuracy: 0.9962 - type_accuracy: 0.9887 - val_loss: 0.7795 - val_shape_loss: 0.0756 - val_type_loss:
0.1266 - val_shape_accuracy: 0.9703 - val_type_accuracy: 0.9649 - lr: 0.0010
Epoch 9/50
92/92 [==============================] - 14s 146ms/step - loss: 0.5849 - shape_loss: 0.0179 - type_loss: 0.03
92 - shape_accuracy: 0.9962 - type_accuracy: 0.9918 - val_loss: 0.6314 - val_shape_loss: 0.0475 - val_type_lo
ss: 0.1012 - val_shape_accuracy: 0.9865 - val_type_accuracy: 0.9797 - lr: 0.0010
Epoch 10/50
92/92 [==============================] - 13s 138ms/step - loss: 0.5036 - shape_loss: 0.0197 - type_loss: 0.04
14 - shape_accuracy: 0.9952 - type_accuracy: 0.9908 - val_loss: 0.6643 - val_shape_loss: 0.1189 - val_type_lo
ss: 0.1282 - val_shape_accuracy: 0.9595 - val_type_accuracy: 0.9662 - lr: 0.0010
Epoch 11/50
92/92 [==============================] - 11s 119ms/step - loss: 0.4804 - shape_loss: 0.0205 - type_loss: 0.05
25 - shape_accuracy: 0.9949 - type_accuracy: 0.9884 - val_loss: 0.5031 - val_shape_loss: 0.0403 - val_type_lo
ss: 0.0738 - val_shape_accuracy: 0.9892 - val_type_accuracy: 0.9811 - lr: 0.0010
Epoch 12/50
92/92 [==============================] - 10s 105ms/step - loss: 0.4766 - shape_loss: 0.0235 - type_loss: 0.05
86 - shape_accuracy: 0.9945 - type_accuracy: 0.9860 - val_loss: 0.4797 - val_shape_loss: 0.0324 - val_type_lo
ss: 0.0663 - val_shape_accuracy: 0.9878 - val_type_accuracy: 0.9797 - lr: 0.0010
Epoch 13/50
92/92 [==============================] - 13s 139ms/step - loss: 0.4115 - shape_loss: 0.0108 - type_loss: 0.04
10 - shape_accuracy: 0.9983 - type_accuracy: 0.9884 - val_loss: 0.4509 - val_shape_loss: 0.0486 - val_type_lo
ss: 0.0626 - val_shape_accuracy: 0.9824 - val_type_accuracy: 0.9784 - lr: 0.0010
Epoch 14/50
92/92 [==============================] - 8s 84ms/step - loss: 0.3760 - shape_loss: 0.0176 - type_loss: 0.0382
- shape_accuracy: 0.9966 - type_accuracy: 0.9908 - val_loss: 0.5438 - val_shape_loss: 0.0666 - val_type_loss:
0.1617 - val_shape_accuracy: 0.9797 - val_type_accuracy: 0.9500 - lr: 0.0010
Epoch 15/50
92/92 [==============================] - 11s 118ms/step - loss: 0.4182 - shape_loss: 0.0279 - type_loss: 0.06
22 - shape_accuracy: 0.9932 - type_accuracy: 0.9846 - val_loss: 0.4487 - val_shape_loss: 0.0419 - val_type_lo
ss: 0.0686 - val_shape_accuracy: 0.9851 - val_type_accuracy: 0.9811 - lr: 0.0010
Epoch 16/50
92/92 [==============================] - 9s 100ms/step - loss: 0.4200 - shape_loss: 0.0242 - type_loss: 0.054
0 - shape_accuracy: 0.9949 - type_accuracy: 0.9860 - val_loss: 0.5382 - val_shape_loss: 0.0715 - val_type_los
s: 0.1190 - val_shape_accuracy: 0.9757 - val_type_accuracy: 0.9635 - lr: 0.0010
Epoch 17/50
92/92 [==============================] - 13s 142ms/step - loss: 0.4024 - shape_loss: 0.0157 - type_loss: 0.04
48 - shape_accuracy: 0.9962 - type_accuracy: 0.9887 - val_loss: 0.3881 - val_shape_loss: 0.0110 - val_type_lo
ss: 0.0549 - val_shape_accuracy: 0.9973 - val_type_accuracy: 0.9892 - lr: 0.0010
Epoch 18/50
92/92 [==============================] - 9s 92ms/step - loss: 0.3518 - shape_loss: 0.0117 - type_loss: 0.0387
- shape_accuracy: 0.9962 - type_accuracy: 0.9898 - val_loss: 0.4138 - val_shape_loss: 0.0225 - val_type_loss:
0.1080 - val_shape_accuracy: 0.9946 - val_type_accuracy: 0.9635 - lr: 0.0010
Epoch 19/50
92/92 [==============================] - 11s 123ms/step - loss: 0.3280 - shape_loss: 0.0113 - type_loss: 0.04
19 - shape_accuracy: 0.9973 - type_accuracy: 0.9908 - val_loss: 0.4662 - val_shape_loss: 0.0566 - val_type_lo
ss: 0.1299 - val_shape_accuracy: 0.9838 - val_type_accuracy: 0.9676 - lr: 0.0010
Epoch 20/50
92/92 [==============================] - 10s 108ms/step - loss: 0.3836 - shape_loss: 0.0221 - type_loss: 0.05
99 - shape_accuracy: 0.9935 - type_accuracy: 0.9822 - val_loss: 0.6717 - val_shape_loss: 0.1559 - val_type_lo
ss: 0.1953 - val_shape_accuracy: 0.9662 - val_type_accuracy: 0.9473 - lr: 0.0010
Epoch 21/50
92/92 [==============================] - 11s 120ms/step - loss: 0.3951 - shape_loss: 0.0188 - type_loss: 0.05
79 - shape_accuracy: 0.9956 - type_accuracy: 0.9843 - val_loss: 0.4267 - val_shape_loss: 0.0491 - val_type_lo
ss: 0.0593 - val_shape_accuracy: 0.9838 - val_type_accuracy: 0.9784 - lr: 0.0010
Epoch 22/50
92/92 [==============================] - 12s 133ms/step - loss: 0.3872 - shape_loss: 0.0276 - type_loss: 0.04
83 - shape_accuracy: 0.9921 - type_accuracy: 0.9867 - val_loss: 0.4930 - val_shape_loss: 0.0470 - val_type_lo
ss: 0.1320 - val_shape_accuracy: 0.9905 - val_type_accuracy: 0.9595 - lr: 0.0010
Epoch 23/50
92/92 [==============================] - 12s 129ms/step - loss: 0.3537 - shape_loss: 0.0131 - type_loss: 0.02
92 - shape_accuracy: 0.9973 - type_accuracy: 0.9949 - val_loss: 0.3624 - val_shape_loss: 0.0167 - val_type_lo
ss: 0.0386 - val_shape_accuracy: 0.9959 - val_type_accuracy: 0.9892 - lr: 1.0000e-04
Epoch 24/50
92/92 [==============================] - 10s 115ms/step - loss: 0.3214 - shape_loss: 0.0041 - type_loss: 0.01
```

```
49 - shape_accuracy: 1.0000 - type_accuracy: 0.9973 - val_loss: 0.3395 - val_shape_loss: 0.0129 - val_type_lo
ss: 0.0290 - val_shape_accuracy: 0.9959 - val_type_accuracy: 0.9905 - lr: 1.0000e-04
Epoch 25/50
92/92 [==============================] - 8s 89ms/step - loss: 0.3071 - shape_loss: 0.0037 - type_loss: 0.0107
- shape_accuracy: 0.9997 - type_accuracy: 0.9993 - val_loss: 0.3226 - val_shape_loss: 0.0119 - val_type_loss:
0.0229 - val_shape_accuracy: 0.9959 - val_type_accuracy: 0.9919 - lr: 1.0000e-04
Epoch 26/50
92/92 [==============================] - 10s 104ms/step - loss: 0.2943 - shape_loss: 0.0029 - type_loss: 0.00
85 - shape_accuracy: 0.9997 - type_accuracy: 0.9990 - val_loss: 0.3091 - val_shape_loss: 0.0113 - val_type_lo
ss: 0.0200 - val_shape_accuracy: 0.9959 - val_type_accuracy: 0.9946 - lr: 1.0000e-04
Epoch 27/50
92/92 [==============================] - 11s 115ms/step - loss: 0.2828 - shape_loss: 0.0022 - type_loss: 0.00
77 - shape_accuracy: 1.0000 - type_accuracy: 0.9993 - val_loss: 0.3001 - val_shape_loss: 0.0117 - val_type_lo
ss: 0.0206 - val_shape_accuracy: 0.9959 - val_type_accuracy: 0.9946 - lr: 1.0000e-04
Epoch 28/50
92/92 [==============================] - 8s 90ms/step - loss: 0.2729 - shape_loss: 0.0026 - type_loss: 0.0075
- shape_accuracy: 1.0000 - type_accuracy: 0.9997 - val_loss: 0.2895 - val_shape_loss: 0.0115 - val_type_loss:
0.0202 - val_shape_accuracy: 0.9973 - val_type_accuracy: 0.9946 - lr: 1.0000e-04
Epoch 29/50
92/92 [==============================] - 11s 118ms/step - loss: 0.2613 - shape_loss: 0.0019 - type_loss: 0.00
65 - shape_accuracy: 1.0000 - type_accuracy: 0.9997 - val_loss: 0.2773 - val_shape_loss: 0.0109 - val_type_lo
ss: 0.0185 - val_shape_accuracy: 0.9946 - val_type_accuracy: 0.9932 - lr: 1.0000e-04
Epoch 30/50
92/92 [==============================] - 10s 111ms/step - loss: 0.2518 - shape_loss: 0.0020 - type_loss: 0.00
68 - shape_accuracy: 1.0000 - type_accuracy: 0.9993 - val_loss: 0.2656 - val_shape_loss: 0.0111 - val_type_lo
ss: 0.0165 - val_shape_accuracy: 0.9973 - val_type_accuracy: 0.9959 - lr: 1.0000e-04
Epoch 31/50
92/92 [==============================] - 10s 106ms/step - loss: 0.2394 - shape_loss: 0.0015 - type_loss: 0.00
47 - shape_accuracy: 1.0000 - type_accuracy: 0.9997 - val_loss: 0.2558 - val_shape_loss: 0.0105 - val_type_lo
ss: 0.0172 - val_shape_accuracy: 0.9973 - val_type_accuracy: 0.9959 - lr: 1.0000e-04
Epoch 32/50
92/92 [==============================] - 11s 120ms/step - loss: 0.2308 - shape_loss: 0.0022 - type_loss: 0.00
52 - shape_accuracy: 0.9997 - type_accuracy: 0.9997 - val_loss: 0.2469 - val_shape_loss: 0.0113 - val_type_lo
ss: 0.0171 - val_shape_accuracy: 0.9973 - val_type_accuracy: 0.9959 - lr: 1.0000e-04
Epoch 33/50
92/92 [==============================] - 10s 108ms/step - loss: 0.2212 - shape_loss: 0.0016 - type_loss: 0.00
57 - shape_accuracy: 1.0000 - type_accuracy: 0.9993 - val_loss: 0.2409 - val_shape_loss: 0.0116 - val_type_lo
ss: 0.0200 - val_shape_accuracy: 0.9946 - val_type_accuracy: 0.9932 - lr: 1.0000e-04
Epoch 34/50
92/92 [==============================] - 12s 133ms/step - loss: 0.2102 - shape_loss: 0.0015 - type_loss: 0.00
40 - shape_accuracy: 1.0000 - type_accuracy: 1.0000 - val_loss: 0.2280 - val_shape_loss: 0.0105 - val_type_lo
ss: 0.0176 - val_shape_accuracy: 0.9946 - val_type_accuracy: 0.9946 - lr: 1.0000e-04
Epoch 35/50
92/92 [==============================] - 12s 129ms/step - loss: 0.2015 - shape_loss: 0.0018 - type_loss: 0.00
44 - shape_accuracy: 1.0000 - type_accuracy: 1.0000 - val_loss: 0.2178 - val_shape_loss: 0.0100 - val_type_lo
ss: 0.0170 - val_shape_accuracy: 0.9973 - val_type_accuracy: 0.9959 - lr: 1.0000e-04
Epoch 36/50
92/92 [==============================] - 13s 145ms/step - loss: 0.1914 - shape_loss: 0.0014 - type_loss: 0.00
37 - shape_accuracy: 1.0000 - type_accuracy: 1.0000 - val_loss: 0.2068 - val_shape_loss: 0.0094 - val_type_lo
ss: 0.0157 - val_shape_accuracy: 0.9973 - val_type_accuracy: 0.9959 - lr: 1.0000e-04
Epoch 37/50
92/92 [==============================] - 12s 131ms/step - loss: 0.1824 - shape_loss: 0.0015 - type_loss: 0.00
34 - shape_accuracy: 1.0000 - type_accuracy: 1.0000 - val_loss: 0.1975 - val_shape_loss: 0.0094 - val_type_lo
ss: 0.0151 - val_shape_accuracy: 0.9986 - val_type_accuracy: 0.9973 - lr: 1.0000e-04
Epoch 38/50
92/92 [==============================] - 9s 98ms/step - loss: 0.1746 - shape_loss: 0.0016 - type_loss: 0.0042
- shape_accuracy: 1.0000 - type_accuracy: 1.0000 - val_loss: 0.1882 - val_shape_loss: 0.0092 - val_type_loss:
0.0145 - val_shape_accuracy: 0.9973 - val_type_accuracy: 0.9973 - lr: 1.0000e-04
Epoch 39/50
92/92 [==============================] - 13s 146ms/step - loss: 0.1658 - shape_loss: 0.0015 - type_loss: 0.00
36 - shape_accuracy: 1.0000 - type_accuracy: 0.9997 - val_loss: 0.1885 - val_shape_loss: 0.0106 - val_type_lo
ss: 0.0213 - val_shape_accuracy: 0.9973 - val_type_accuracy: 0.9932 - lr: 1.0000e-04
Epoch 40/50
92/92 [==============================] - 12s 125ms/step - loss: 0.1593 - shape_loss: 0.0018 - type_loss: 0.00
48 - shape_accuracy: 1.0000 - type_accuracy: 1.0000 - val_loss: 0.1743 - val_shape_loss: 0.0092 - val_type_lo
ss: 0.0161 - val_shape_accuracy: 0.9973 - val_type_accuracy: 0.9959 - lr: 1.0000e-04
Epoch 41/50
92/92 [==============================] - 9s 92ms/step - loss: 0.1510 - shape_loss: 0.0016 - type_loss: 0.0041
- shape_accuracy: 1.0000 - type_accuracy: 1.0000 - val_loss: 0.1681 - val_shape_loss: 0.0103 - val_type_loss:
0.0162 - val_shape_accuracy: 0.9959 - val_type_accuracy: 0.9946 - lr: 1.0000e-04
Epoch 42/50
92/92 [==============================] - 10s 106ms/step - loss: 0.1439 - shape_loss: 0.0014 - type_loss: 0.00
44 - shape_accuracy: 1.0000 - type_accuracy: 0.9997 - val_loss: 0.1580 - val_shape_loss: 0.0096 - val_type_lo
ss: 0.0139 - val_shape_accuracy: 0.9973 - val_type_accuracy: 0.9959 - lr: 1.0000e-04
Epoch 43/50
92/92 [==============================] - 11s 125ms/step - loss: 0.1381 - shape_loss: 0.0019 - type_loss: 0.00
50 - shape_accuracy: 1.0000 - type_accuracy: 0.9993 - val_loss: 0.1532 - val_shape_loss: 0.0104 - val_type_lo
ss: 0.0148 - val_shape_accuracy: 0.9973 - val_type_accuracy: 0.9973 - lr: 1.0000e-04
Epoch 44/50
92/92 [==============================] - 12s 129ms/step - loss: 0.1302 - shape_loss: 0.0015 - type_loss: 0.00
40 - shape_accuracy: 1.0000 - type_accuracy: 1.0000 - val_loss: 0.1473 - val_shape_loss: 0.0109 - val_type_lo
ss: 0.0149 - val_shape_accuracy: 0.9973 - val_type_accuracy: 0.9973 - lr: 1.0000e-04
Epoch 45/50
92/92 [==============================] - 12s 125ms/step - loss: 0.1263 - shape_loss: 0.0025 - type_loss: 0.00
49 - shape_accuracy: 1.0000 - type_accuracy: 1.0000 - val_loss: 0.1476 - val_shape_loss: 0.0135 - val_type_lo
ss: 0.0176 - val_shape_accuracy: 0.9946 - val_type_accuracy: 0.9946 - lr: 1.0000e-04
Epoch 46/50
92/92 [==============================] - 8s 86ms/step - loss: 0.1207 - shape_loss: 0.0019 - type_loss: 0.0049
- shape_accuracy: 1.0000 - type_accuracy: 1.0000 - val_loss: 0.1383 - val_shape_loss: 0.0109 - val_type_loss:
0.0160 - val_shape_accuracy: 0.9959 - val_type_accuracy: 0.9959 - lr: 1.0000e-04
Epoch 47/50
92/92 [==============================] - 10s 112ms/step - loss: 0.1140 - shape_loss: 0.0016 - type_loss: 0.00
39 - shape_accuracy: 1.0000 - type_accuracy: 1.0000 - val_loss: 0.1332 - val_shape_loss: 0.0111 - val_type_lo
```

```
ss: 0.0162 - val_shape_accuracy: 0.9959 - val_type_accuracy: 0.9959 - lr: 1.0000e-04
Epoch 48/50
92/92 [==============================] - 6s 68ms/step - loss: 0.1083 - shape_loss: 0.0013 - type_loss: 0.0038
- shape_accuracy: 1.0000 - type_accuracy: 1.0000 - val_loss: 0.1241 - val_shape_loss: 0.0093 - val_type_loss:
0.0143 - val_shape_accuracy: 0.9973 - val_type_accuracy: 0.9973 - lr: 1.0000e-04
Epoch 49/50
92/92 [==============================] - 9s 95ms/step - loss: 0.1039 - shape_loss: 0.0014 - type_loss: 0.0043
- shape_accuracy: 1.0000 - type_accuracy: 0.9997 - val_loss: 0.1222 - val_shape_loss: 0.0108 - val_type_loss:
0.0155 - val_shape_accuracy: 0.9959 - val_type_accuracy: 0.9973 - lr: 1.0000e-04
Epoch 50/50
92/92 [==============================] - 14s 146ms/step - loss: 0.0994 - shape_loss: 0.0017 - type_loss: 0.00
40 - shape_accuracy: 1.0000 - type_accuracy: 1.0000 - val_loss: 0.1172 - val_shape_loss: 0.0109 - val_type_lo
ss: 0.0149 - val_shape_accuracy: 0.9973 - val_type_accuracy: 0.9973 - lr: 1.0000e-04
```

- After analysing the initial CNN architecture results, which showed:
  - Validation loss of 0.0970
  - Validation shape loss of 0.0323
  - Validation type loss of 0.0647
  - Validation shape accuracy of 0.9918
  - Validation type accuracy of 0.9851 (see Figure 11)

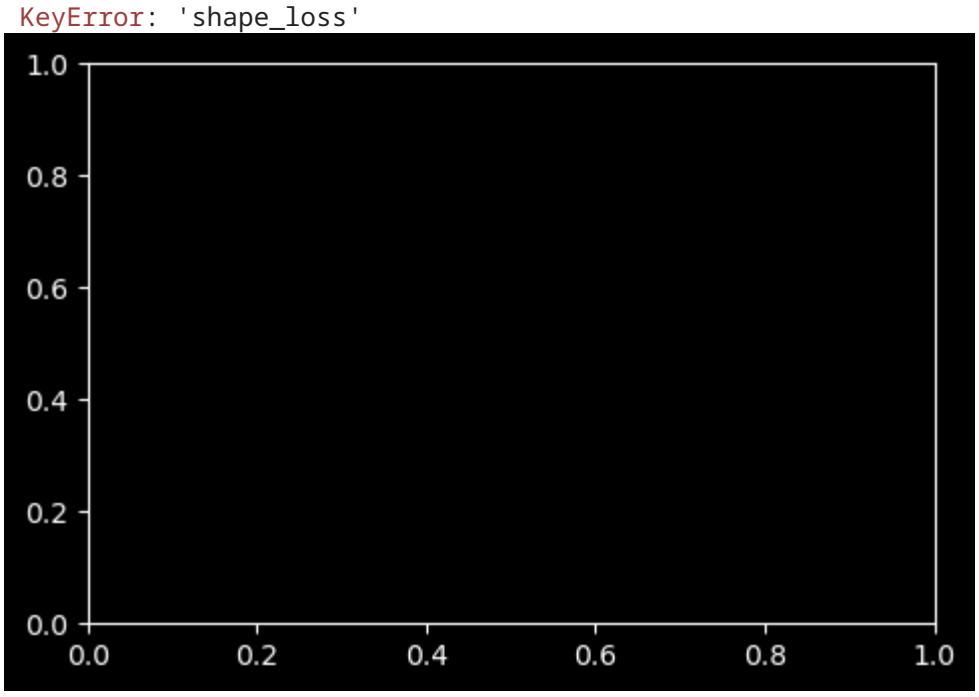
  We applied optimisation techniques similar to those used in the MLP to further improve the model's performance.
- We incorporated L2 regularisation in the convolutional and dense layers to prevent overfitting by adding a penalty term to the loss function.
  - L2 regularisation constrains the model's weights and encourages simpler and more generalisable representations.
- Batch normalisation was added after each convolutional layer to normalise the activations and improve training stability.
  - It reduces the internal covariate shift and allows for faster convergence.
- We increased the number of filters in the convolutional layers to 32, 64, and 128, respectively, to enhance the model's capacity to capture more complex features at different scales.
  - Deeper and wider networks have shown to be more effective in learning hierarchical representations.
- We experimented with the LeakyReLU activation in the third convolutional layer to mitigate the problem of "dying ReLU" by allowing small negative values.
  - This helps maintain gradient flow and prevents neurons from becoming inactive.
- We expanded the dense layer to 256 units and altered the dropout layer to have a rate of 0.6 to attempt to enhance the model's capacity and regularisation capabilities.
  - Dropouts randomly set a fraction of the input units to zero during training, forcing the network to learn more robust and distributed representations.
- We employed callbacks such as early stopping and learning rate reduction on plateau to prevent overfitting and fine-tune the model's performance.
  - Early stopping monitors the validation loss and stops training if no improvement is observed for 10 epochs, ensuring that the model does not overfit to the training data.
  - The learning rate is reduced by a factor of 0.1 if the validation loss does not improve for 5 epochs, allowing for fine-tuning of the model's weights in the later stages of training and helping the model converge to a better optimum.
- These architectural choices and optimisation techniques were implemented to enhance the model's performance, generalisation ability, and computational efficiency for the road sign classification task, building upon the insights gained from the initial CNN architecture results.

4.3 Neural Network Validation Evaluation

In [24…

```
# Plot the learning curves
plt.figure(figsize=(12, 8))

# Plot loss curves
plt.subplot(2, 2, 1)
plt.plot(history.history['shape_loss'], label='Shape Training Loss')
plt.plot(history.history['val_shape_loss'], label='Shape Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(2, 2, 2)
plt.plot(history.history['type_loss'], label='Type Training Loss')
plt.plot(history.history['val_type_loss'], label='Type Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plot accuracy curves
plt.subplot(2, 2, 3)
plt.plot(history.history['shape_accuracy'], label='Shape Training Accuracy')
plt.plot(history.history['val_shape_accuracy'], label='Shape Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(2, 2, 4)
plt.plot(history.history['type_accuracy'], label='Type Training Accuracy')
plt.plot(history.history['val_type_accuracy'], label='Type Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

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

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[24], line 6
      4 # Plot loss curves
      5 plt.subplot(2, 2, 1)
----> 6 plt.plot(history.history['shape_loss'], label='Shape Training Loss')
      7 plt.plot(history.history['val_shape_loss'], label='Shape Validation Loss')
      8 plt.xlabel('Epoch')

KeyError: 'shape_loss'
```



In [25…

```
# Evaluate the model on the validation set
val_loss, val_shape_loss, val_type_loss, val_shape_acc, val_type_acc = model.evaluate(val_generator, steps=

# Print the validation metrics
print("Validation Loss:", val_loss)
print("Validation Shape Loss:", val_shape_loss)
print("Validation Type Loss:", val_type_loss)
print("Validation Shape Accuracy:", val_shape_acc)
print("Validation Type Accuracy:", val_type_acc)
```

```
2024-08-26 17:30:44.423020: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO)
Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT:
You must feed a value for placeholder tensor 'Placeholder/_0' with dtype int32
         [[{{node Placeholder/_0}}]]
2024-08-26 17:30:44.728860: I tensorflow/core/common_runtime/executor.cc:1197] [/job:localhost/replica:0/tas
k:0/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore t
his message): INVALID_ARGUMENT: Can not squeeze dim[1], expected a dimension of 1, got 40
         [[{{node Squeeze}}]]
```

```
---------------------------------------------------------------------------
InvalidArgumentError                      Traceback (most recent call last)
Cell In[25], line 2
      1 # Evaluate the model on the validation set
----> 2 val_loss, val_shape_loss, val_type_loss, val_shape_acc, val_type_acc = model.evaluate(val_generator,
steps=len(val_indices) // 32)
      4 # Print the validation metrics
      5 print("Validation Loss:", val_loss)

File /opt/anaconda/lib/python3.11/site-packages/keras/utils/traceback_utils.py:70, in filter_traceback.<loca
ls>.error_handler(*args, **kwargs)
     67     filtered_tb = _process_traceback_frames(e.__traceback__)
     68     # To get the full stack trace, call:
     69     # `tf.debugging.disable_traceback_filtering()`
---> 70     raise e.with_traceback(filtered_tb) from None
     71 finally:
     72     del filtered_tb

File /opt/anaconda/lib/python3.11/site-packages/tensorflow/python/eager/execute.py:52, in quick_execute(op_n
ame, num_outputs, inputs, attrs, ctx, name)
     50 try:
     51   ctx.ensure_initialized()
---> 52   tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     53                                       inputs, attrs, num_outputs)
     54 except core._NotOkStatusException as e:
     55   if name is not None:

InvalidArgumentError: Graph execution error:

Detected at node 'Squeeze' defined at (most recent call last):
    File "<frozen runpy>", line 198, in _run_module_as_main
    File "<frozen runpy>", line 88, in _run_code
    File "/opt/anaconda/lib/python3.11/site-packages/ipykernel_launcher.py", line 17, in <module>
      app.launch_new_instance()
    File "/opt/anaconda/lib/python3.11/site-packages/traitlets/config/application.py", line 992, in launch_i
nstance
      app.start()
    File "/opt/anaconda/lib/python3.11/site-packages/ipykernel/kernelapp.py", line 701, in start
      self.io_loop.start()
    File "/opt/anaconda/lib/python3.11/site-packages/tornado/platform/asyncio.py", line 195, in start
      self.asyncio_loop.run_forever()
    File "/opt/anaconda/lib/python3.11/asyncio/base_events.py", line 607, in run_forever
      self._run_once()
    File "/opt/anaconda/lib/python3.11/asyncio/base_events.py", line 1922, in _run_once
      handle._run()
    File "/opt/anaconda/lib/python3.11/asyncio/events.py", line 80, in _run
      self._context.run(self._callback, *self._args)
    File "/opt/anaconda/lib/python3.11/site-packages/ipykernel/kernelbase.py", line 534, in dispatch_queue
      await self.process_one()
    File "/opt/anaconda/lib/python3.11/site-packages/ipykernel/kernelbase.py", line 523, in process_one
      await dispatch(*args)
    File "/opt/anaconda/lib/python3.11/site-packages/ipykernel/kernelbase.py", line 429, in dispatch_shell
      await result
    File "/opt/anaconda/lib/python3.11/site-packages/ipykernel/kernelbase.py", line 767, in execute_request
      reply_content = await reply_content
    File "/opt/anaconda/lib/python3.11/site-packages/ipykernel/ipkernel.py", line 429, in do_execute
      res = shell.run_cell(
    File "/opt/anaconda/lib/python3.11/site-packages/ipykernel/zmqshell.py", line 549, in run_cell
      return super().run_cell(*args, **kwargs)
    File "/opt/anaconda/lib/python3.11/site-packages/IPython/core/interactiveshell.py", line 3051, in run_ce
ll
      result = self._run_cell(
    File "/opt/anaconda/lib/python3.11/site-packages/IPython/core/interactiveshell.py", line 3106, in _run_c
ell
      result = runner(coro)
    File "/opt/anaconda/lib/python3.11/site-packages/IPython/core/async_helpers.py", line 129, in _pseudo_sy
nc_runner
      coro.send(None)
    File "/opt/anaconda/lib/python3.11/site-packages/IPython/core/interactiveshell.py", line 3311, in run_ce
ll_async
      has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
    File "/opt/anaconda/lib/python3.11/site-packages/IPython/core/interactiveshell.py", line 3493, in run_as
t_nodes
      if await self.run_code(code, result, async_=asy):
    File "/opt/anaconda/lib/python3.11/site-packages/IPython/core/interactiveshell.py", line 3553, in run_co
de
      exec(code_obj, self.user_global_ns, self.user_ns)
    File "/tmp/ipykernel_7319/348823157.py", line 2, in <module>
      val_loss, val_shape_loss, val_type_loss, val_shape_acc, val_type_acc = model.evaluate(val_generator, s
teps=len(val_indices) // 32)
    File "/opt/anaconda/lib/python3.11/site-packages/keras/utils/traceback_utils.py", line 65, in error_hand
ler
      return fn(*args, **kwargs)
    File "/opt/anaconda/lib/python3.11/site-packages/keras/engine/training.py", line 2072, in evaluate
      tmp_logs = self.test_function(iterator)
    File "/opt/anaconda/lib/python3.11/site-packages/keras/engine/training.py", line 1852, in test_function
      return step_function(self, iterator)
    File "/opt/anaconda/lib/python3.11/site-packages/keras/engine/training.py", line 1836, in step_function
      outputs = model.distribute_strategy.run(run_step, args=(data,))
    File "/opt/anaconda/lib/python3.11/site-packages/keras/engine/training.py", line 1824, in run_step
      outputs = model.test_step(data)
    File "/opt/anaconda/lib/python3.11/site-packages/keras/engine/training.py", line 1791, in test_step
      return self.compute_metrics(x, y, y_pred, sample_weight)
```

```
    File "/opt/anaconda/lib/python3.11/site-packages/keras/engine/training.py", line 1149, in compute_metric
s
      self.compiled_metrics.update_state(y, y_pred, sample_weight)
    File "/opt/anaconda/lib/python3.11/site-packages/keras/engine/compile_utils.py", line 605, in update_sta
te
      metric_obj.update_state(y_t, y_p, sample_weight=mask)
    File "/opt/anaconda/lib/python3.11/site-packages/keras/utils/metrics_utils.py", line 77, in decorated
      update_op = update_state_fn(*args, **kwargs)
    File "/opt/anaconda/lib/python3.11/site-packages/keras/metrics/base_metric.py", line 140, in update_stat
e_fn
      return ag_update_state(*args, **kwargs)
    File "/opt/anaconda/lib/python3.11/site-packages/keras/metrics/base_metric.py", line 691, in update_stat
e
      matches = ag_fn(y_true, y_pred, **self._fn_kwargs)
    File "/opt/anaconda/lib/python3.11/site-packages/keras/metrics/accuracy_metrics.py", line 459, in sparse
_categorical_accuracy
      matches = metrics_utils.sparse_categorical_matches(y_true, y_pred)
    File "/opt/anaconda/lib/python3.11/site-packages/keras/utils/metrics_utils.py", line 963, in sparse_cate
gorical_matches
      y_true = tf.squeeze(y_true, [-1])
Node: 'Squeeze'
Can not squeeze dim[1], expected a dimension of 1, got 40
         [[{{node Squeeze}}]] [Op:__inference_test_function_183626]
```

Visualizing and Interpreting Learning Curves:
✔ Learning Rate and Epoch Cutoff:

- Visualizing the learning curves helps us understand the model's learning progress and make informed decisions about the learning rate and the number of epochs.
- By monitoring the training and validation loss curves, we can identify the point at which the model starts to overfit or converge. This helps us determine the optimal epoch cutoff point and apply early stopping to prevent overfitting and save computational resources.

✔ Training and Validation Loss:

- Plotting the training and validation loss curves side by side allows us to assess the model's performance and generalization ability.
- Ideally, we want to see both the training and validation loss decreasing over epochs, indicating that the model is learning and improving its predictions.
- If the validation loss starts to increase while the training loss continues to decrease, it suggests that the model is overfitting to the training data and not generalizing well to unseen data. In such cases, early stopping or regularization techniques can be applied to mitigate overfitting.

✔ Training and Validation Accuracy:

- Visualizing the training and validation accuracy curves helps us evaluate the model's performance in terms of correct predictions.
- We expect the training and validation accuracy to increase over epochs, indicating that the model is learning and improving its classification accuracy.
- If the validation accuracy plateaus or starts to decrease while the training accuracy continues to increase, it suggests that the model is memorizing the training data and not generalizing well to unseen data. This is another indication of overfitting, and regularization techniques or early stopping can be employed to address it.

✔ Interpreting the Curves:

- By analyzing the learning curves, we gain insights into the model's learning dynamics and can make informed decisions to optimize its performance.
- If the training and validation curves converge and reach a satisfactory level of performance, it indicates that the model has learned meaningful patterns and generalizes well to unseen data.
- If there is a significant gap between the training and validation curves, it suggests a mismatch between the model's performance on the training data and its generalization ability. This gap can be addressed by techniques such as regularization, increasing the training data, or adjusting the model's complexity.

4.3-1 Confusion Matrix

Confusion Matrices and Evaluation Metrics:
✔ Confusion Matrix:

- A confusion matrix is a table that visualizes the performance of a classification model by comparing the predicted labels against the actual labels. In an ideal scenario, we look for a strong diagonal line (from top left to bottom right) in the confusion matrix, indicating that the predicted labels match the actual labels.

✔ Evaluation Metrics:

- To assess the performance of an n-ary classification model, we utilize several evaluation metrics, including F1 score, accuracy, recall, and precision. These metrics provide insights into different aspects of the model's performance.

In [26…

```python
from sklearn.metrics import classification_report

# Initialize empty lists to store the true labels and predicted labels
shape_true_labels = []
type_true_labels = []
shape_pred_labels = []
type_pred_labels = []

# Specify the number of batches to process
num_batches = 24  # Process 24 batches (740 images with batch size 32)

# Iterate over the validation data in batches
```

```python
batch_count = 0
for batch_data, batch_labels in val_generator:
    # Generate predictions for the current batch
    batch_preds = model.predict(batch_data)
    shape_preds = np.argmax(batch_preds[0], axis=1)
    type_preds = np.argmax(batch_preds[1], axis=1)

    # Append the true labels and predicted labels to the lists
    shape_true_labels.extend(batch_labels[0])
    type_true_labels.extend(batch_labels[1])
    shape_pred_labels.extend(shape_preds)
    type_pred_labels.extend(type_preds)

    batch_count += 1
    if batch_count >= num_batches:
        break  # Exit the loop after processing the specified number of batches

# Map the predicted labels back to their original names
shape_labels_map = {i: label for i, label in enumerate(shape_encoder.classes_)}
type_labels_map = {i: label for i, label in enumerate(type_encoder.classes_)}

# Create confusion matrices for shape and type
shape_cm = confusion_matrix(shape_true_labels, shape_pred_labels)
type_cm = confusion_matrix(type_true_labels, type_pred_labels)

print("Confusion matrices created.")

# Generate classification reports for shape and type
shape_report = classification_report(shape_true_labels, shape_pred_labels, target_names=shape_encoder.clas
type_report = classification_report(type_true_labels, type_pred_labels, target_names=type_encoder.classes_

print("Shape Classification Report:")
print(shape_report)

print("Type Classification Report:")
print(type_report)

# Plot the confusion matrix for shape
plt.figure(figsize=(8, 6))
ConfusionMatrixDisplay(shape_cm, display_labels=shape_encoder.classes_).plot()
plt.title("Shape Confusion Matrix")
plt.xlabel("Predicted Shape")
plt.ylabel("True Shape")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

# Plot the confusion matrix for type
plt.figure(figsize=(8, 6))
ConfusionMatrixDisplay(type_cm, display_labels=type_encoder.classes_).plot()
plt.title("Type Confusion Matrix")
plt.xlabel("Predicted Type")
plt.ylabel("True Type")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```

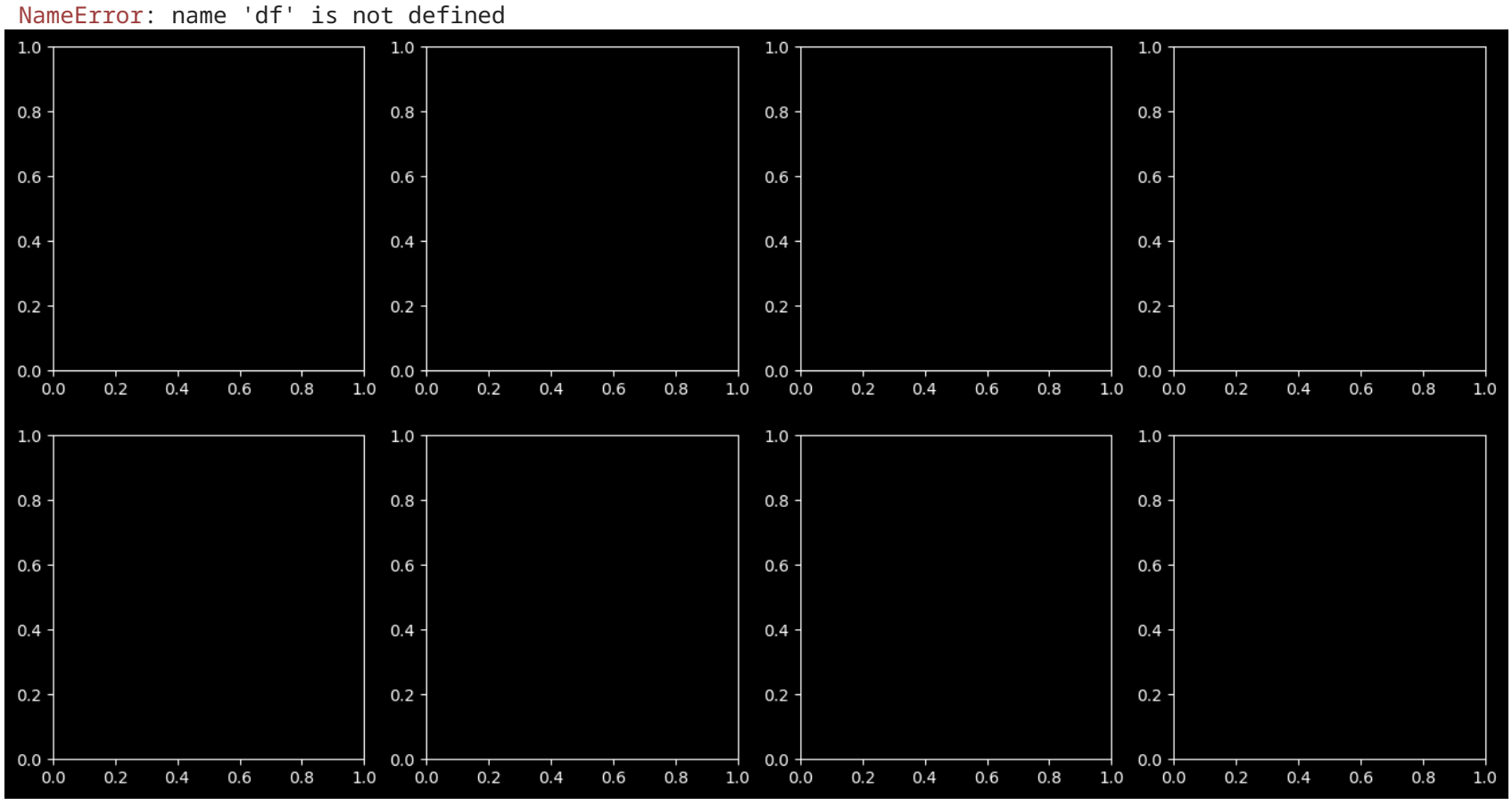
```
1/1 [==============================] - 0s 210ms/step
1/1 [==============================] - 0s 141ms/step
1/1 [==============================] - 0s 119ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 157ms/step
1/1 [==============================] - 0s 154ms/step
1/1 [==============================] - 0s 127ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 127ms/step
1/1 [==============================] - 0s 110ms/step
1/1 [==============================] - 0s 129ms/step
1/1 [==============================] - 0s 118ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 128ms/step
1/1 [==============================] - 0s 122ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 143ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 122ms/step
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 136ms/step
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[26], line 31
     28         break  # Exit the loop after processing the specified number of batches
     30 # Map the predicted labels back to their original names
---> 31 shape_labels_map = {i: label for i, label in enumerate(shape_encoder.classes_)}
     32 type_labels_map = {i: label for i, label in enumerate(type_encoder.classes_)}
     34 # Create confusion matrices for shape and type

NameError: name 'shape_encoder' is not defined
```

In [27…

```python
# Create a figure with 2 rows and 4 columns
```

```python
fig, axes = plt.subplots(2, 4, figsize=(16, 8))
axes = axes.ravel()

# Loop through 8 random examples
for i in range(8):
    # Select a random image path from the validation set
    random_index = np.random.choice(val_indices)
    image_path = df.iloc[random_index]['image_path']

    # Load and preprocess the image
    img = load_img(image_path, target_size=img_size)
    x = img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)

    # Make predictions
    shape_pred, type_pred = model.predict(x)

    # Get the predicted shape and type labels
    shape_label = shape_encoder.inverse_transform([np.argmax(shape_pred)])[0]
    type_label = type_encoder.inverse_transform([np.argmax(type_pred)])[0]

    # Display the tested image
    axes[i].imshow(img)
    axes[i].axis('off')
    axes[i].set_title(f"Shape: {shape_label}, Type: {type_label}")

# Adjust the spacing between subplots
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[27], line 9
      6 for i in range(8):
      7     # Select a random image path from the validation set
      8     random_index = np.random.choice(val_indices)
----> 9     image_path = df.iloc[random_index]['image_path']
     11     # Load and preprocess the image
     12     img = load_img(image_path, target_size=img_size)

NameError: name 'df' is not defined
```



4.4 Model Parameters & Details

In [35…

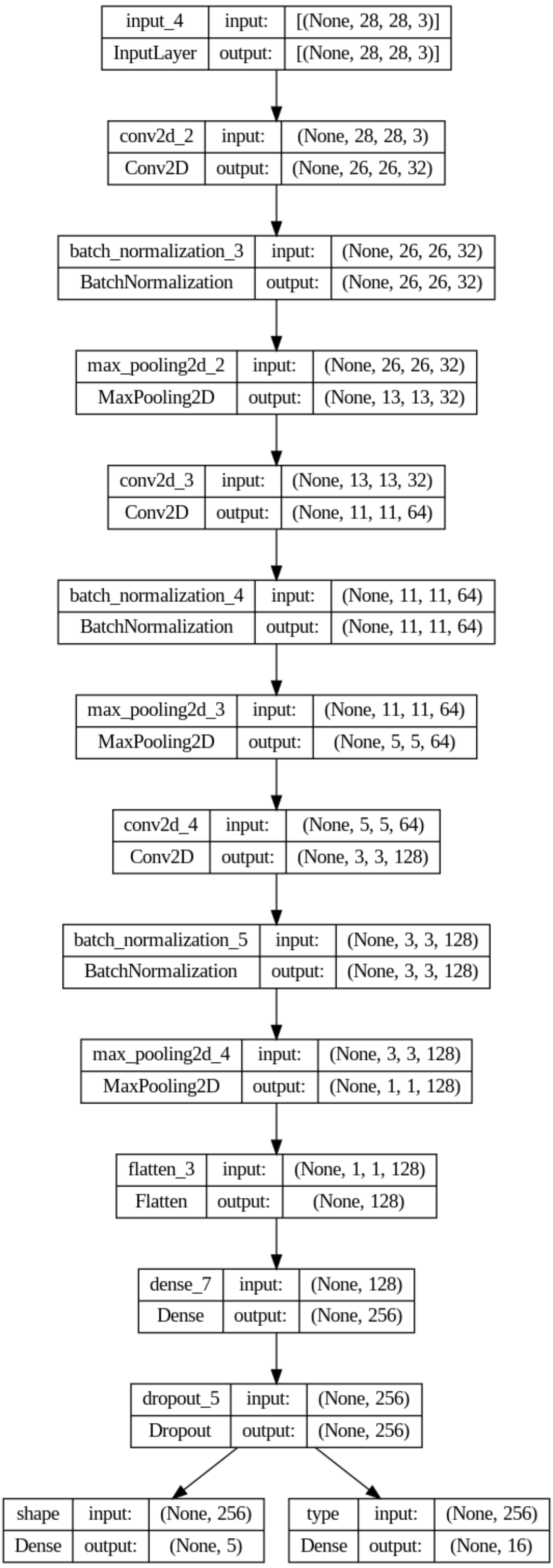
```python
model.summary()
```

```
Model: "model_3"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_4 (InputLayer)           [(None, 28, 28, 3)]  0           []

 conv2d_2 (Conv2D)              (None, 26, 26, 32)   896         ['input_4[0][0]']

 batch_normalization_3 (BatchNo  (None, 26, 26, 32)  128         ['conv2d_2[0][0]']
 rmalization)

 max_pooling2d_2 (MaxPooling2D)  (None, 13, 13, 32)  0           ['batch_normalization_3[0][0]']

 conv2d_3 (Conv2D)              (None, 11, 11, 64)   18496       ['max_pooling2d_2[0][0]']

 batch_normalization_4 (BatchNo  (None, 11, 11, 64)  256         ['conv2d_3[0][0]']
 rmalization)

 max_pooling2d_3 (MaxPooling2D)  (None, 5, 5, 64)    0           ['batch_normalization_4[0][0]']

 conv2d_4 (Conv2D)              (None, 3, 3, 128)    73856       ['max_pooling2d_3[0][0]']

 batch_normalization_5 (BatchNo  (None, 3, 3, 128)   512         ['conv2d_4[0][0]']
 rmalization)

 max_pooling2d_4 (MaxPooling2D)  (None, 1, 1, 128)   0           ['batch_normalization_5[0][0]']

 flatten_3 (Flatten)            (None, 128)          0           ['max_pooling2d_4[0][0]']

 dense_7 (Dense)                (None, 256)          33024       ['flatten_3[0][0]']

 dropout_5 (Dropout)            (None, 256)          0           ['dense_7[0][0]']

 shape (Dense)                  (None, 5)            1285        ['dropout_5[0][0]']

 type (Dense)                   (None, 16)           4112        ['dropout_5[0][0]']

==================================================================================================
Total params: 132,565
Trainable params: 132,117
Non-trainable params: 448
__________________________________________________________________________________________________
```

In [36…

```
tf.keras.utils.plot_model(model, show_shapes=True)
```

Out[36]:


input_4 | input: | [(None, 28, 28, 3)]
InputLayer | output: | [(None, 28, 28, 3)]
conv2d_2 | input: | (None, 28, 28, 3)
Conv2D | output: | (None, 26, 26, 32)
batch_normalization_3 | input: | (None, 26, 26, 32)
BatchNormalization | output: | (None, 26, 26, 32)
max_pooling2d_2 | input: | (None, 26, 26, 32)
MaxPooling2D | output: | (None, 13, 13, 32)
conv2d_3 | input: | (None, 13, 13, 32)
Conv2D | output: | (None, 11, 11, 64)
batch_normalization_4 | input: | (None, 11, 11, 64)
BatchNormalization | output: | (None, 11, 11, 64)
max_pooling2d_3 | input: | (None, 11, 11, 64)
MaxPooling2D | output: | (None, 5, 5, 64)
conv2d_4 | input: | (None, 5, 5, 64)
Conv2D | output: | (None, 3, 3, 128)
batch_normalization_5 | input: | (None, 3, 3, 128)
BatchNormalization | output: | (None, 3, 3, 128)
max_pooling2d_4 | input: | (None, 3, 3, 128)
MaxPooling2D | output: | (None, 1, 1, 128)
flatten_3 | input: | (None, 1, 1, 128)
Flatten | output: | (None, 128)
dense_7 | input: | (None, 128)
Dense | output: | (None, 256)
dropout_5 | input: | (None, 256)
Dropout | output: | (None, 256)
shape | input: | (None, 256)
Dense | output: | (None, 5)
type | input: | (None, 256)
Dense | output: | (None, 16)

4.4-1 Save the Model!

In [37…
```
model.save("model")
```

```
2024-05-19 19:02:41.326251: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG IN
FO) Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_A
RGUMENT: You must feed a value for placeholder tensor 'inputs' with dtype float and shape [?,256]
         [[{{node inputs}}]]
2024-05-19 19:02:41.652280: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG IN
FO) Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_A
RGUMENT: You must feed a value for placeholder tensor 'inputs' with dtype float and shape [?,256]
         [[{{node inputs}}]]
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op
, _jit_compiled_convolution_op, _update_step_xla, leaky_re_lu_3_layer_call_fn while saving (showing 5 of
6). These functions will not be directly callable after loading.
INFO:tensorflow:Assets written to: model/assets
INFO:tensorflow:Assets written to: model/assets
```

5.0 Independent Evaluation

5.1 Domain Expert

Classify Images of Road Traffic Signs Independent Dataset Sourcing To collect our independent evaluation dataset, we followed a rigorous scientific method to ensure the data's quality and diversity. We sought insights from domain experts, such as Professor Simon Jones, School of Science at RMIT University.

- Professor Jones has extensive experience in collecting data and classifying images for various applications, including:
  - Fire detection and prediction using advanced machine learning techniques like:
    - Random Forests
    - Ensemble methods
    - U-Net CNNs

Based on the guidance from our domain expert and several scientific journals, we focused on gathering data from various sources to create a diverse and representative dataset.

- We collected 511 road sign images from the German Traffic Sign Recognition Benchmark (GTSRB) Dataset, a sister dataset to the BelgiumTS Dataset used for training our model.
- We captured 68 Greater Melbourne Region road sign images using:
  - iPhone 12 (Oisin)
  - iPhone 13 (Vince)
- We sourced 106 Belgium Road Signs from Google Maps to test the dataset against unseen data from the same country.

In total, we created a dataset of 689 images for our independent evaluation, comparable to the validation dataset and equating to 20% of the training dataset size, as recommended by Professor Alavi. Please see Figure 14 for the true distribution of the number of samples per country and a detailed breakdown of their shapes and types.

5.2 Sourcing Methodology

During the data collection process, we prioritised capturing images with varying angles, brightness levels, and sharpness/blurriness. We also, as previously mentioned, were using different cameras, which further simulated a real-world scenario.

- This approach helped us create a dataset that accounts for the diverse conditions in which road signs are encountered in real life.
- We also made sure to include both correct, poor-quality, and edge case examples of road signs to thoroughly test our model's ability to distinguish between different classes and handle challenging cases.
- Ensuring no overlap with our training/validation dataset was relatively easy. It just meant we had to be very careful not to find images that directly matched the training dataset and find ones comparable to what a real-world test may look like.

To ensure compatibility with our trained model, we developed a custom pre-processing script that standardised the collected images.

- This script processes all .png and .ppm images that haven't already been transformed by:
  - Resizing them to 28×28 pixels
  - Converting them to Grey scale
- By applying this pre-processing step, we guarantee that our test data is in the same format as the training data, enabling a fair evaluation of the model's performance.

After pre-processing, we conducted an Exploratory Data Analysis (EDA) on the test dataset, similar to the analysis performed on the training data.

- This step allowed us to verify data consistency and identify any anomalies or irregularities.
- We ensured that the images were centred within the 28×28 frame and exhibited variance for sharpness, pixel intensity, entropy, and similarity in the object's appearance, reflecting real-world variations.
- See Section 5.5 Independent Evaluation Ingestion & EDA in Appendix A: Jupyter Notebook

The best idea is to gather both incorrect and correct data to test.

- We used both the German sister dataset and collected 200 of our own images
- Oisin has an iPhone 12
- Vince has an iPhone 13
- For camera comparison, refer to: https://istyle.ae/blog/camera-comparison-iphone-12-vs-iphone-13-camera

In [38…
```
print("\nValidation set counts:")
print("Shape labels:")
print(df.loc[val_indices, 'shape_label'].map(dict(enumerate(shape_labels))).value_counts())
print("\nType labels:")
print(df.loc[val_indices, 'type_label'].map(dict(enumerate(type_labels))).value_counts())
```

```
Validation set counts:
Shape labels:
shape_label
round       342
triangle    179
square      136
diamond      73
hex          10
Name: count, dtype: int64

Type labels:
type_label
warning             133
rightofway           73
noentry              69
speed                67
parking              52
bicycle              52
noparking            48
giveway              46
trafficdirective     40
continue             39
laneend              28
limitedtraffic       28
roundabout           22
crossing             17
traveldirection      16
stop                 10
Name: count, dtype: int64
```

5.3 Data References

- Oisin & Vince, Images denoted with Australian at the start, RMIT University
- Google, Images denoted with Belgium at the start, Google Maps, accessed 5 May 2024, https://www.google.com/maps/place/Belgium/
- GTSDB September 2010, Images denoted with German at the start, GTSDB, accessed 18 April 2024, https://benchmark.ini.rub.de/gtsdb_dataset.html

5.4 Pre-Processing

In [39…
```
def process_images(directory):
    for root, dirs, files in os.walk(directory):
        for file in files:
            # Get the file extension
            _, extension = os.path.splitext(file)

            # Check if the file is an image
            if extension.lower() in ['.png', '.ppm']:
                # Construct the full file paths
                input_path = os.path.join(root, file)
                output_path = os.path.join(root, os.path.splitext(file)[0] + '.png')

                try:
                    # Open the image file
                    with Image.open(input_path) as img:
                        # Convert the image to grayscale
                        grayscale_img = img.convert('L')

                        # Resize the grayscale image to 28x28 pixels
                        resized_img = grayscale_img.resize((28, 28))

                        # Save the processed image as PNG, using the new file extension
                        resized_img.save(output_path)

                        # Check if the original file is a .ppm file
                        if extension.lower() == '.ppm':
                            # Delete the original .ppm file
                            os.remove(input_path)

                except Exception as e:
                    print(f"Error processing {input_path}: {str(e)}")
```

```
# Directory to start the recursive processing
directory = "independent_dataset"

# Call the function to process and save the images
process_images(directory)
```

- In order to properly evaluate the test data, it must be in the same format as the training data (28x28 grayscale).
- To achieve this, we:
  - Run a custom pre-processing script that we developed.
  - This script processes all .png & .ppm images that haven't already been transformed by:
    - First resizing them to 28x28 pixels
    - Then converting them to grayscale
- After pre-processing, we perform a similar Exploratory Data Analysis (EDA) as done previously on the training data.
- This step ensures:
  - Data consistency
  - Allows us to check for any anomalies or irregularities
- The images should be:
  - Centered within the 28x28 frame
  - Have a varying degree of angles and slight differences in the object's appearance to account for real-world variations
- By following this pre-processing and EDA procedure, we can:
  - Guarantee that our test data is in the appropriate format
  - Ensure it is suitable for evaluating our trained model's performance on unseen examples
- This process helps us assess the model's:
  - Generalization capabilities
  - Robustness to new data

5.5 Independent Evaluation Ingestion & EDA

In [40…

```
# Directory containing the image dataset
dataset_dir = 'independent_dataset'

# Function to get all image files in subdirectories
def get_image_files(directory):
    image_files = []
    for root, dirs, files in os.walk(directory):
        for file in files:
            if file.endswith(('.png')):
                image_files.append(os.path.join(root, file))
    return image_files

# Get the list of image files
image_files = get_image_files(dataset_dir)
print("Number of image files found:", len(image_files))

# Initialize lists to store image properties
image_sizes = []
image_formats = []
image_colors = []
image_sharpness = []
pixel_intensities = []
entropies = []
image_similarities = []
image_data = []

# Iterate over the image files
for image_path in image_files:
    try:
        image = io.imread(image_path, as_gray=True)

        # Get image size
        height, width = image.shape
        image_sizes.append((width, height))

        # Get image format
        image_format = os.path.splitext(image_path)[1][1:]
        image_formats.append(image_format)

        # Check if the image is grayscale or color
        if len(image.shape) == 2:
            image_colors.append('grayscale')
        else:
            image_colors.append('color')

        # Calculate image sharpness using the variance of the Laplacian
        sharpness = np.var(filters.laplace(image))
        image_sharpness.append(sharpness)

        # Calculate average pixel intensity
        pixel_intensity = np.mean(image)
        pixel_intensities.append(pixel_intensity)

        # Calculate image entropy
        entropy = measure.shannon_entropy(image)
        entropies.append(entropy)
```

```
        # Store image data for similarity calculation
        image_data.append(image.flatten())

        moments = measure.moments_hu(image)

    except Exception as e:
        print(f"Error processing image {image_path}: {str(e)}")

# Calculate image similarities
for i, img1 in enumerate(image_data):
    if i == 0:
        image_similarities.append(0)
    else:
        distances = [euclidean(img1, img2) for img2 in image_data[:i]]
        image_similarities.append(np.min(distances))
```

```
Number of image files found: 689
```

In [41…

```
# Plot image size distribution
plt.figure(figsize=(8, 6))
plt.hist([size[0] for size in image_sizes], bins=20, alpha=0.5, color='red', label='Width')
plt.hist([size[1] for size in image_sizes], bins=20, alpha=0.5, color='blue', label='Height')
plt.xlabel('Size (pixels)')
plt.ylabel('Frequency')
plt.title('Image Size Distribution')
plt.legend()
plt.show()

# Plot image format distribution
plt.figure(figsize=(8, 6))
plt.hist(image_formats, color='purple')
plt.xlabel('Image Format')
plt.ylabel('Frequency')
plt.title('Image Format Distribution')
plt.show()

# Plot image color distribution
plt.figure(figsize=(8, 6))
plt.hist(image_colors, color='purple')
plt.xlabel('Color Mode')
plt.ylabel('Frequency')
plt.title('Image Color Distribution')
plt.show()

# Plot image sharpness distribution
plt.figure(figsize=(8, 6))
plt.hist(image_sharpness, bins=20, color='purple')
plt.xlabel('Sharpness')
plt.ylabel('Frequency')
plt.title('Image Sharpness Distribution')
plt.show()

# Plot pixel intensity distribution
plt.figure(figsize=(8, 6))
plt.hist(pixel_intensities, bins=20, color='purple')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.title('Pixel Intensity Distribution')
plt.show()

# Plot entropy distribution
plt.figure(figsize=(8, 6))
plt.hist(entropies, bins=20, color='purple')
plt.xlabel('Entropy')
plt.ylabel('Frequency')
plt.title('Entropy Distribution')
plt.show()

# Plot image similarity distribution
plt.figure(figsize=(8, 6))
plt.hist(image_similarities, bins=20, color='purple')
plt.xlabel('Similarity')
plt.ylabel('Frequency')
plt.title('Image Similarity Distribution')
plt.show()
```

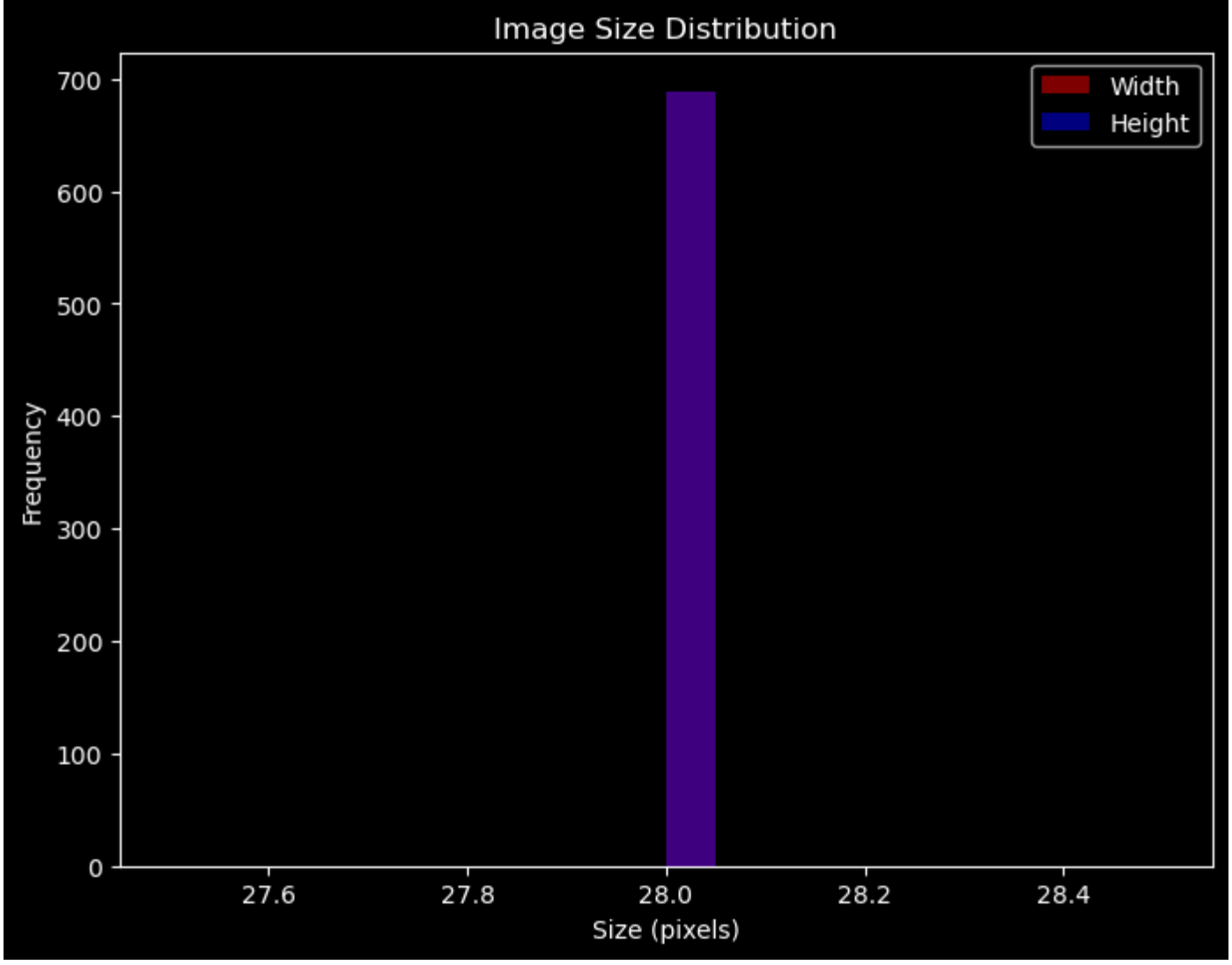


Image Size Distribution
Width
Height
Frequency
700
600
500
400
300
200
100
0
27.6
27.8
28.0
28.2
28.4
Size (pixels)

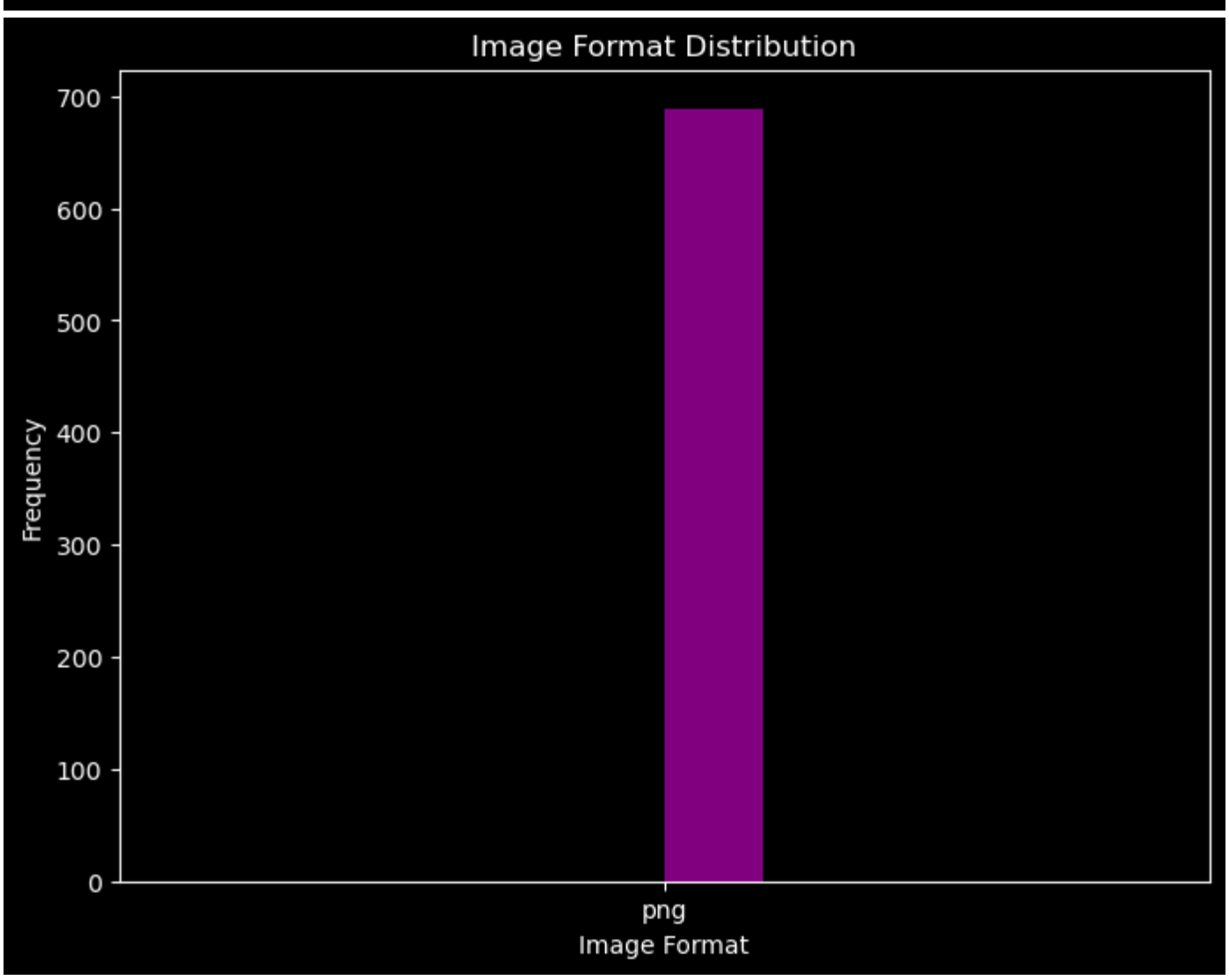


Image Format Distribution
Frequency
700
600
500
400
300
200
100
0
png
Image Format

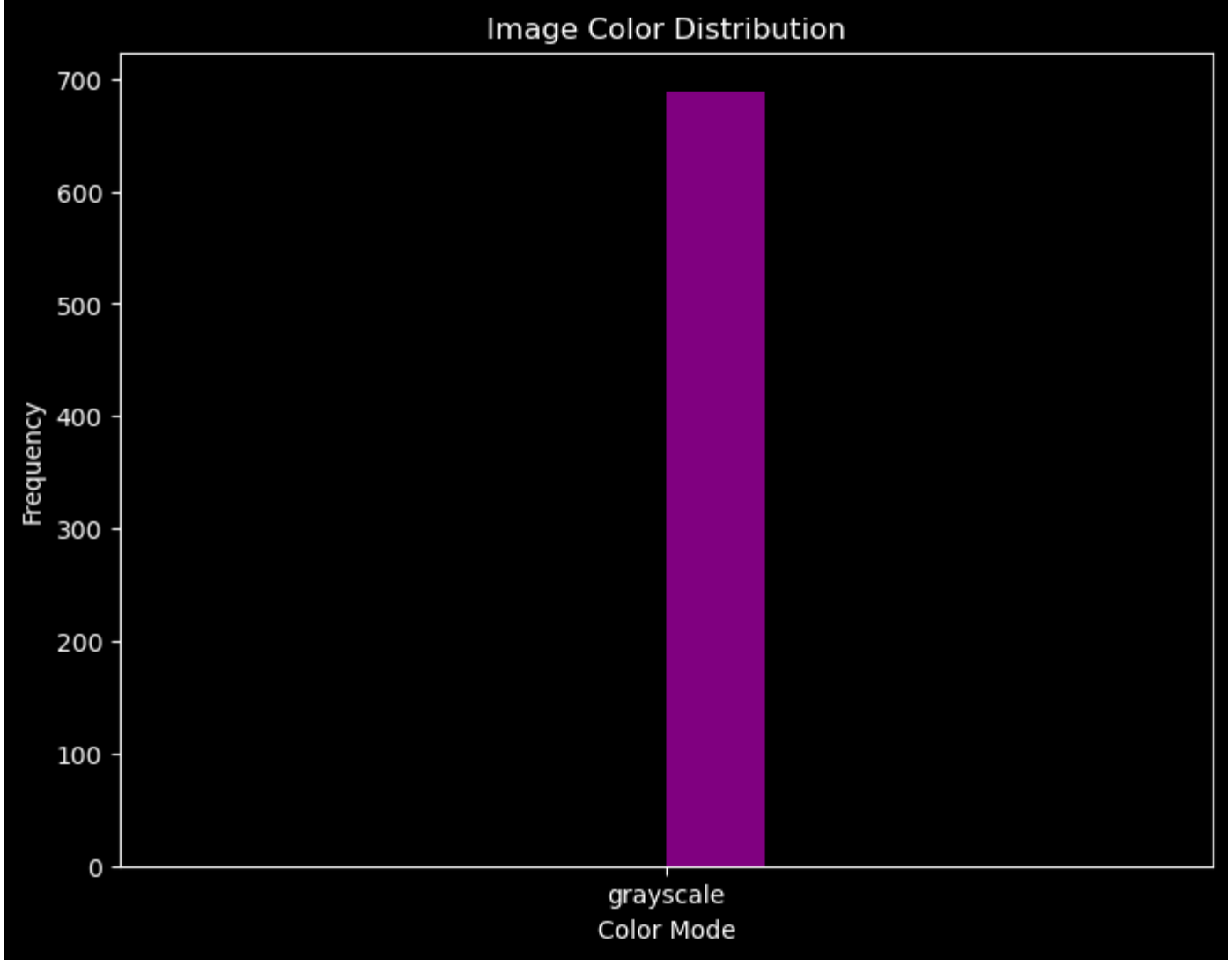


Image Color Distribution
Frequency
700
600
500
400
300
200
100
0
grayscale
Color Mode

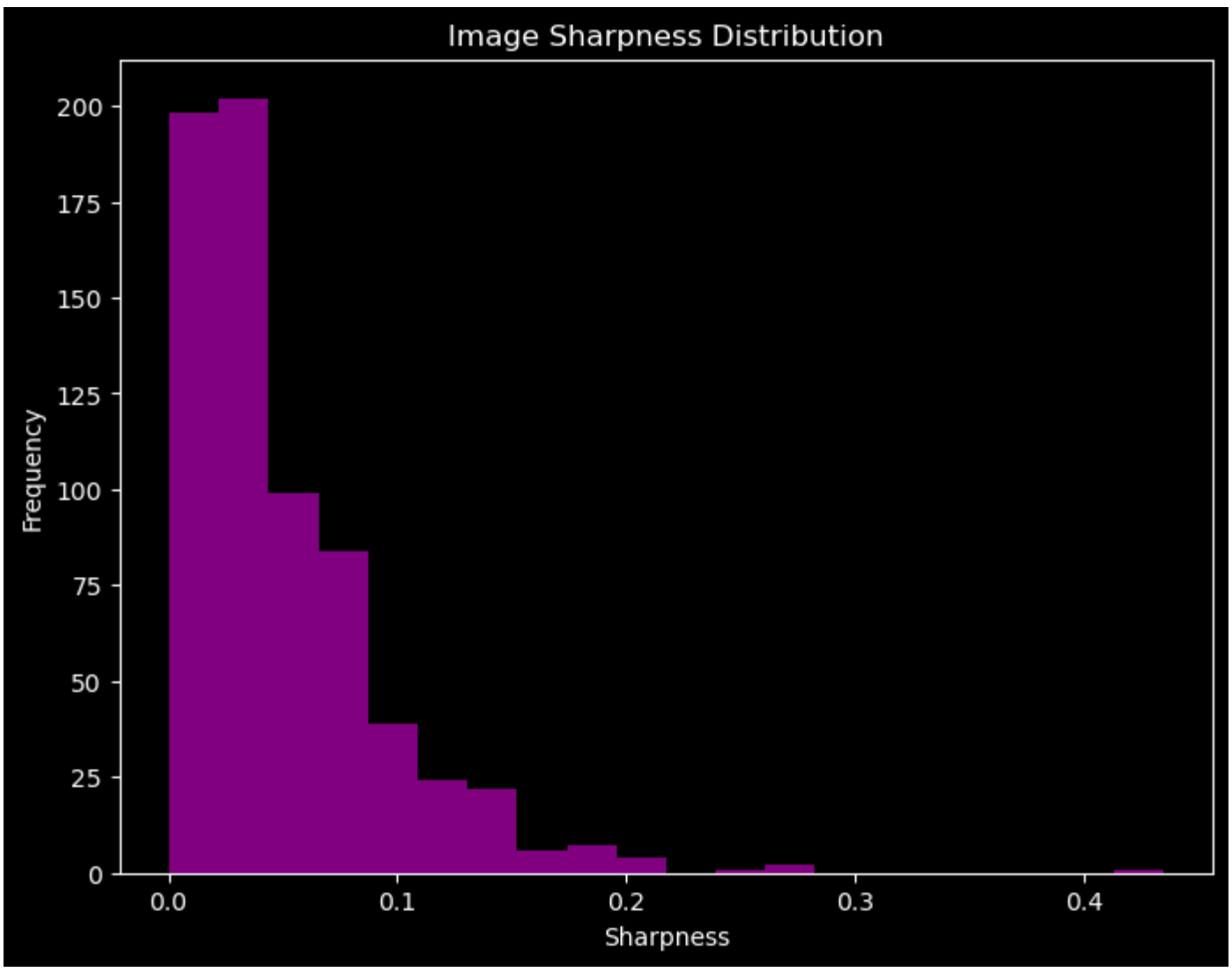
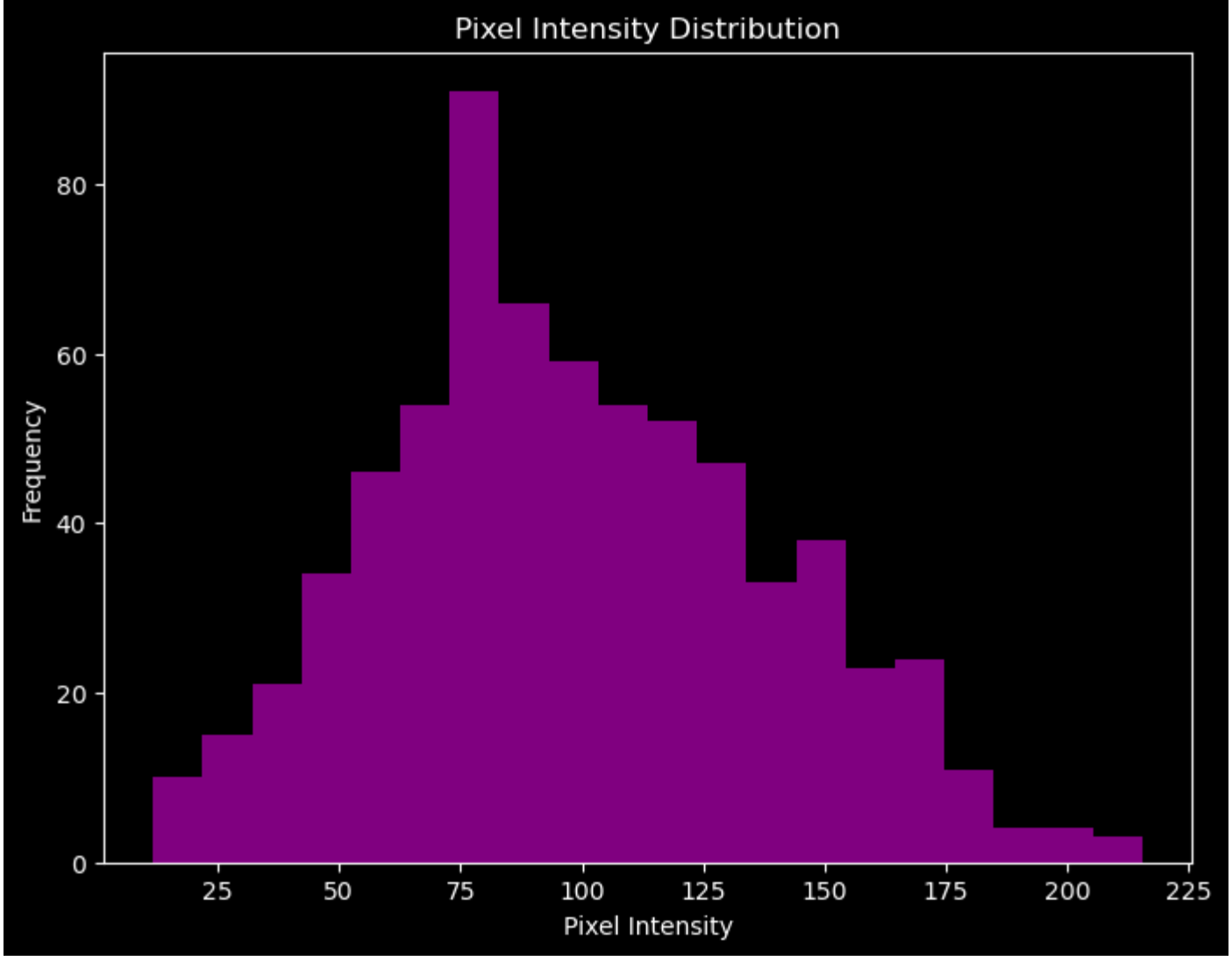
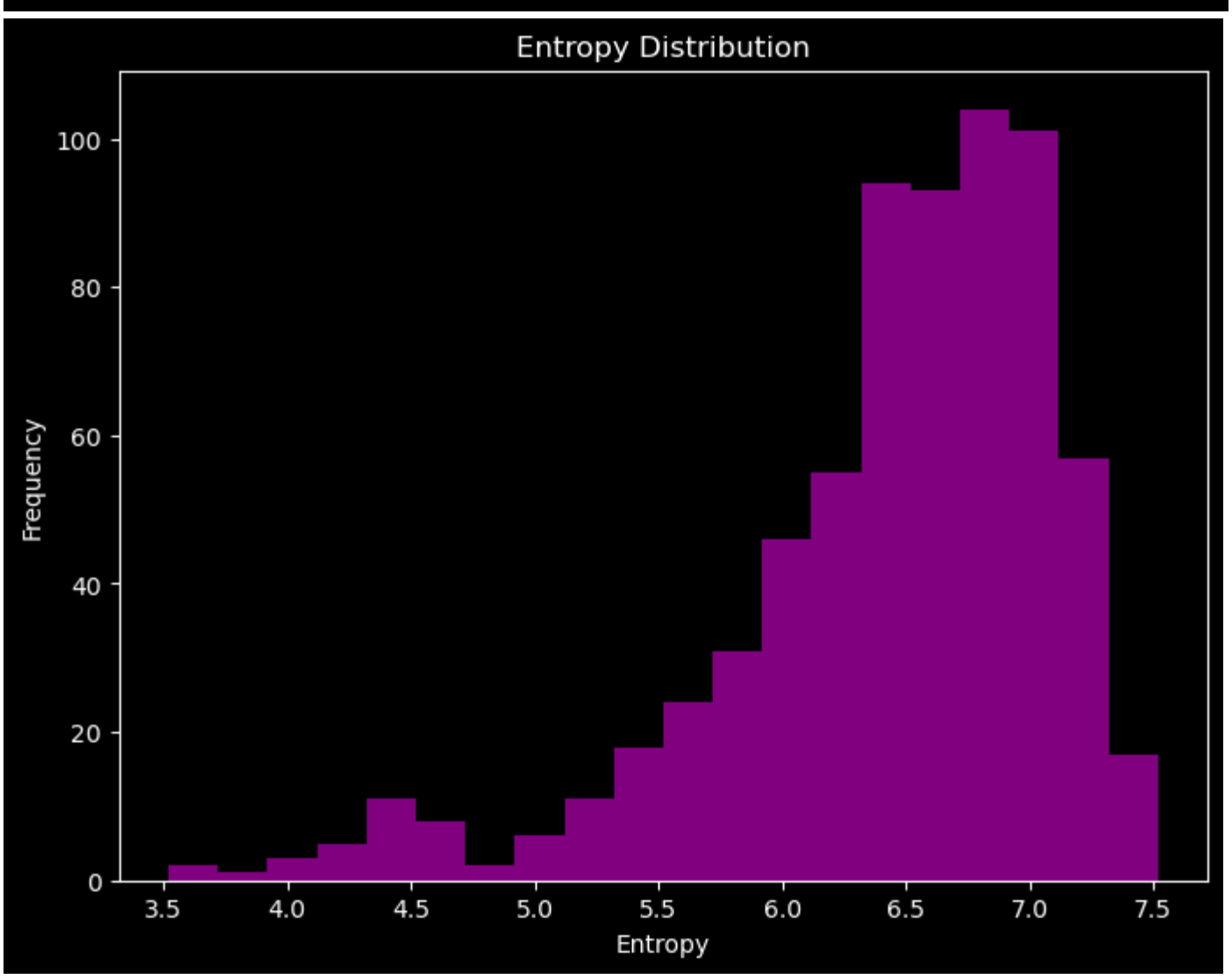
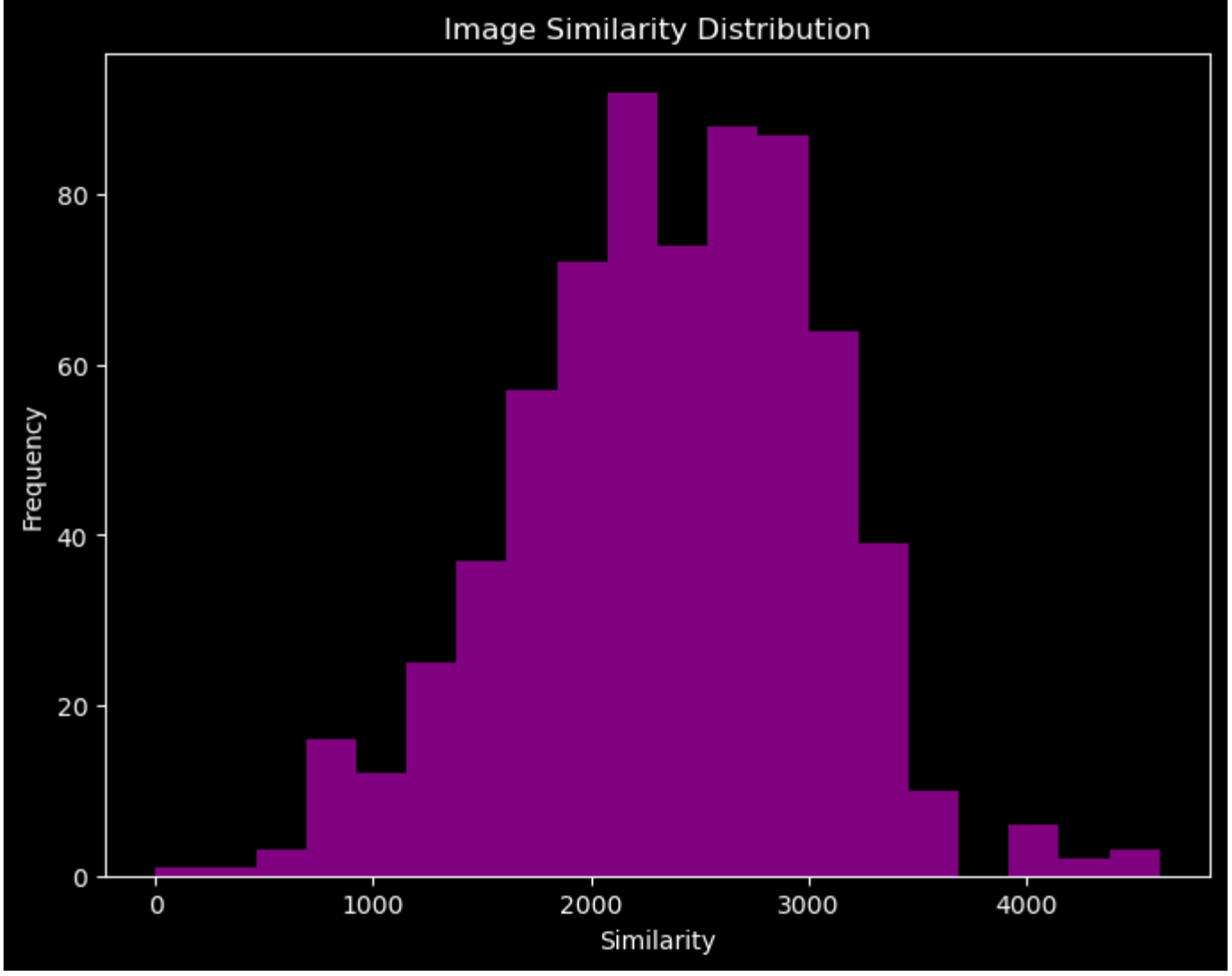


Image Sharpness Distribution
Frequency
200
175
150
125
100
75
50
25
0
0.0
0.1
0.2
0.3
0.4
Sharpness


Pixel Intensity Distribution
Frequency
0
20
40
60
80
25
50
75
100
125
150
175
200
225
Pixel Intensity


Entropy Distribution
Frequency
0
20
40
60
80
100
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
Entropy



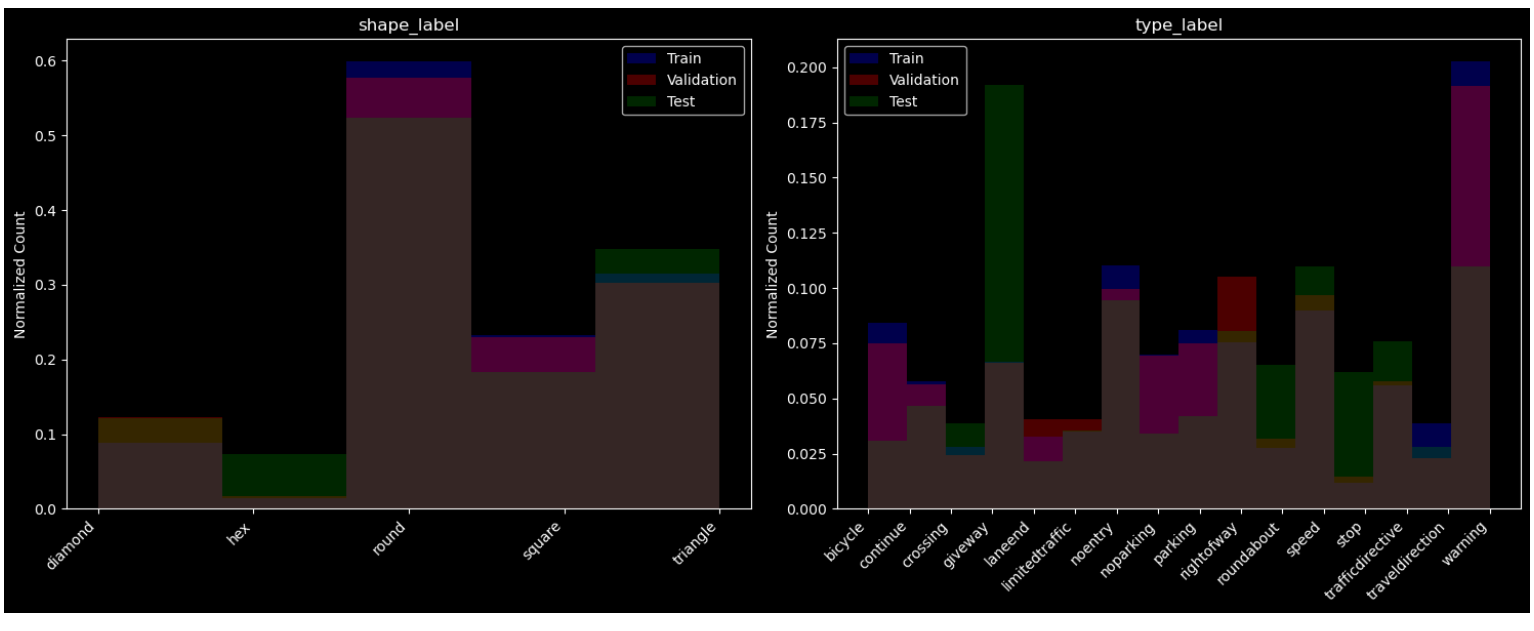
In [43…

```
test_dataset_path = 'independent_dataset'
img_size = (28, 28)
# img_size = (32, 32) # for transfer learning
```

NOTE: To run this cell you must run the previous data leakage cell.

In [73…

```
plt.figure(figsize=(15, 6))

for i, col in enumerate(['shape_label', 'type_label']):
    plt.subplot(1, 2, i+1)
    if col == 'shape_label':
        unique_values = shape_labels
    else:
        unique_values = type_labels

    plt.hist(df.loc[train_indices, col], alpha=0.3, color='b', density=True,
             bins=len(unique_values), label='Train')
    plt.hist(df.loc[val_indices, col], alpha=0.3, color='r', density=True,
             bins=len(unique_values), label='Validation')
    plt.hist(test_df[col], alpha=0.3, color='g', density=True,
             bins=len(unique_values), label='Test')

    plt.title(col)
    plt.xticks(range(len(unique_values)), unique_values, rotation=45, ha='right')
    plt.ylabel('Normalized Count')  # Add this line to set the y-label
    plt.legend()

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



- Illustrates the normalised distributions of our training, validation, and test datasets, which are reasonably close, with extra test data for underrepresented classes.
  - Blue represents training data
  - Red indicates validation data
  - Green corresponds to test data
  - Other colours signify dataset overlaps:
    - Brown for all three datasets
    - Yellow for validation and test only
- Due to time constraints, we focused on easily accessible data, resulting in an abundance of certain classes like:

    - Hexagon stop signs
    - Triangle giveway signs

  Compared to less common signs like:

    - Bicycle signs
    - No Parking signs

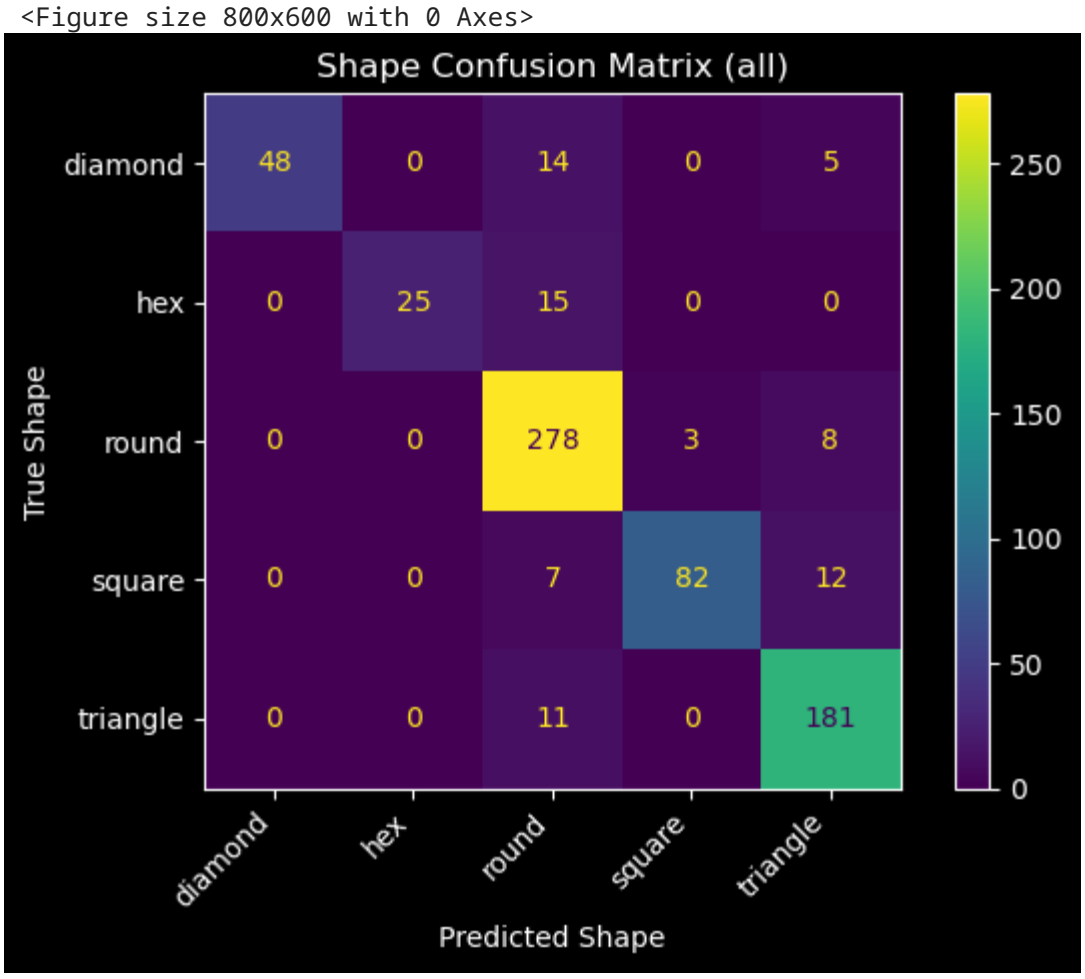
- By following this meticulous data collection process, leveraging insights from domain experts, and applying appropriate pre-processing and EDA techniques, we created a high-quality and diverse independent evaluation dataset.
- This dataset will enable us to assess our trained model's:
    - Generalisation capabilities
    - Robustness when exposed to unseen examples

  Providing a reliable measure of its performance in real-world scenarios.

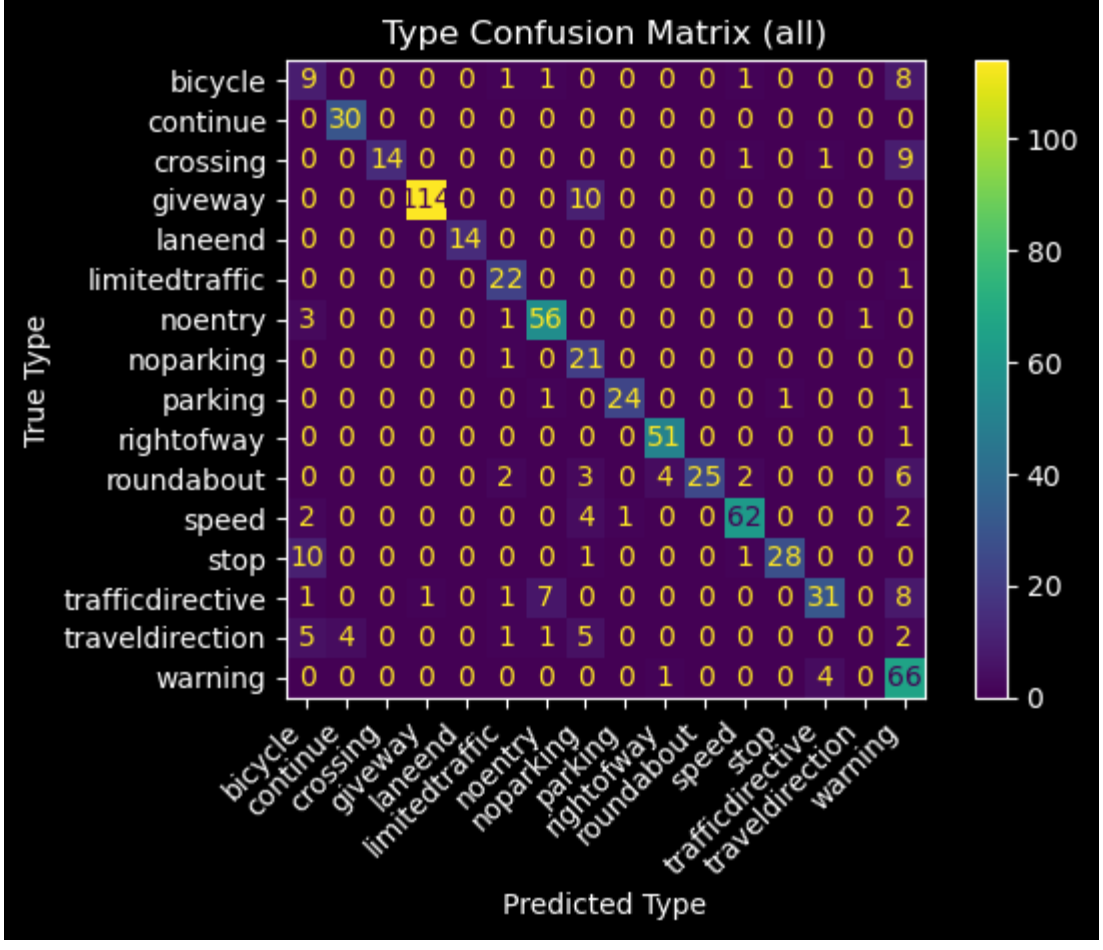
In [45…

```
def generate_confusion_matrices(subcategory):
    # Load the test image data based on the subcategory
    if subcategory == 'all':
        X_test = np.array([img_to_array(load_img(path, target_size=img_size)) / 255.0 for path i
        shape_true_labels = test_df['shape_label'].values
        type_true_labels = test_df['type_label'].values
    elif subcategory == 'australian':
        X_test = np.array([img_to_array(load_img(path, target_size=img_size)) / 255.0 for path i
        shape_true_labels = test_df[test_df['image_path'].str.contains('australian')]['shape_lab
        type_true_labels = test_df[test_df['image_path'].str.contains('australian')]['type_label
    elif subcategory == 'belgium':
        X_test = np.array([img_to_array(load_img(path, target_size=img_size)) / 255.0 for path i
        shape_true_labels = test_df[test_df['image_path'].str.contains('belgium')]['shape_label'
        type_true_labels = test_df[test_df['image_path'].str.contains('belgium')]['type_label'].
    elif subcategory == 'german':
        X_test = np.array([img_to_array(load_img(path, target_size=img_size)) / 255.0 for path i
        shape_true_labels = test_df[test_df['image_path'].str.contains('german')]['shape_label']
        type_true_labels = test_df[test_df['image_path'].str.contains('german')]['type_label'].v
    else:
        raise ValueError(f"Invalid subcategory: {subcategory}")

    # Generate predictions for the test data
    test_preds = model.predict(X_test)
    shape_preds = np.argmax(test_preds[0], axis=1)
    type_preds = np.argmax(test_preds[1], axis=1)

    # Get the unique shape and type labels present in the subset of data
    shape_labels = np.unique(shape_true_labels)
    type_labels = np.unique(type_true_labels)

    # Map the predicted labels to their original names based on the subset of data
    shape_labels_map = {label: shape_encoder.inverse_transform([label])[0] for label in shape_la
    type_labels_map = {label: type_encoder.inverse_transform([label])[0] for label in type_label

    # Create confusion matrices for shape and type
    shape_cm = confusion_matrix(shape_true_labels, shape_preds, labels=shape_labels)
    type_cm = confusion_matrix(type_true_labels, type_preds, labels=type_labels)

    print(f"Confusion matrices created for subcategory: {subcategory}")

    # Generate classification report for shape
    print(f"Classification Report - Shape ({subcategory}):")
    print(classification_report(shape_true_labels, shape_preds, labels=shape_labels, target_name

    # Generate classification report for type
    print(f"Classification Report - Type ({subcategory}):")
    print(classification_report(type_true_labels, type_preds, labels=type_labels, target_names=[

    # Plot the confusion matrix for shape
    plt.figure(figsize=(8, 6))
    ConfusionMatrixDisplay(shape_cm, display_labels=[shape_labels_map[label] for label in shape_
    plt.title(f"Shape Confusion Matrix ({subcategory})")
    plt.xlabel("Predicted Shape")
    plt.ylabel("True Shape")
    plt.xticks(rotation=45, ha="right")
    plt.tight_layout()
    plt.show()

    # Plot the confusion matrix for type
    plt.figure(figsize=(8, 6))
    ConfusionMatrixDisplay(type_cm, display_labels=[type_labels_map[label] for label in type_lab
    plt.title(f"Type Confusion Matrix ({subcategory})")
    plt.xlabel("Predicted Type")
    plt.ylabel("True Type")
    plt.xticks(rotation=45, ha="right")
    plt.tight_layout()
    plt.show()

# Generate confusion matrices for different subcategories
generate_confusion_matrices('all')
generate_confusion_matrices('australian')
generate_confusion_matrices('belgium')
generate_confusion_matrices('german')
```

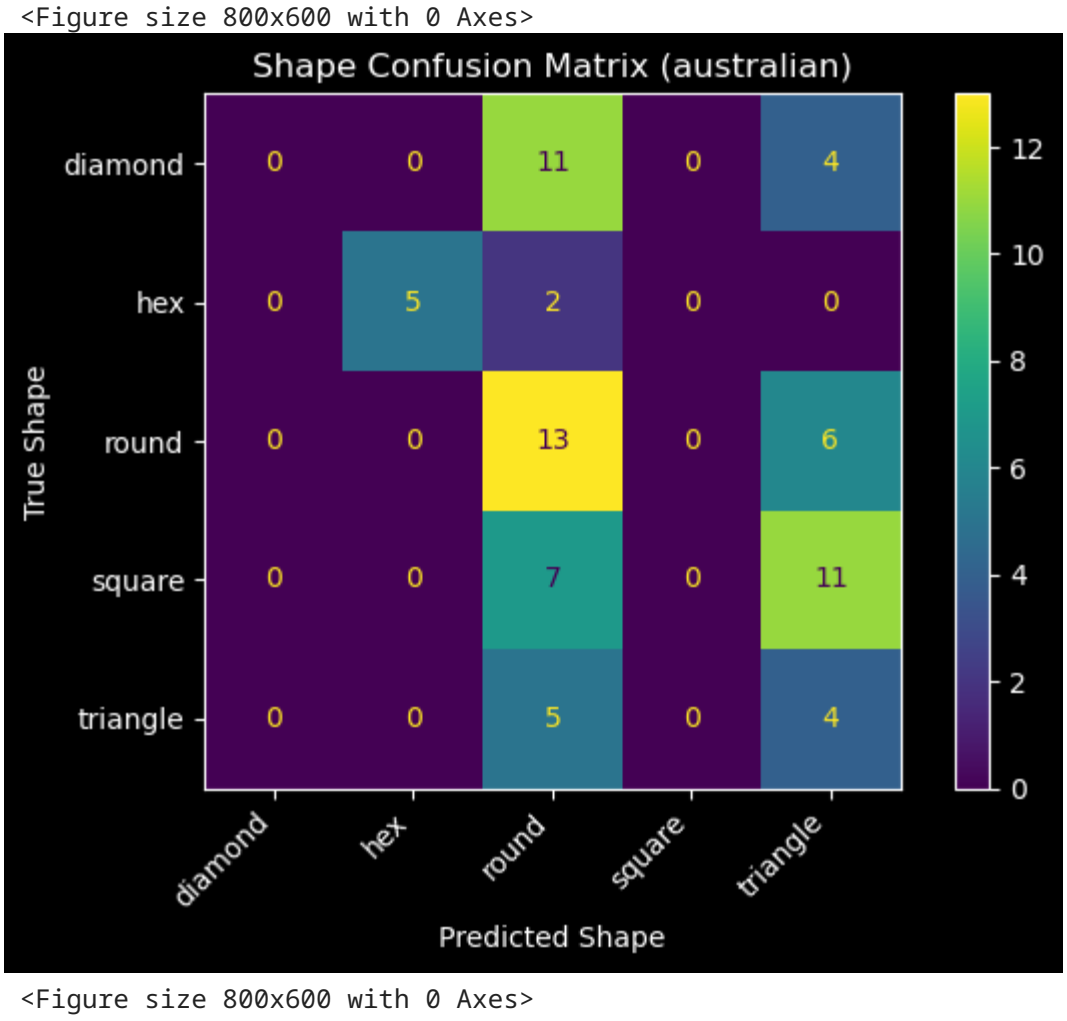
```
22/22 [==============================] - 0s 14ms/step
Confusion matrices created for subcategory: all
Classification Report - Shape (all):
              precision    recall  f1-score   support

     diamond       1.00      0.72      0.83        67
         hex       1.00      0.62      0.77        40
       round       0.86      0.96      0.91       289
      square       0.96      0.81      0.88       101
    triangle       0.88      0.94      0.91       192

    accuracy                           0.89       689
   macro avg       0.94      0.81      0.86       689
weighted avg       0.90      0.89      0.89       689

Classification Report - Type (all):
                 precision    recall  f1-score   support

        bicycle       0.30      0.45      0.36        20
       continue       0.88      1.00      0.94        30
       crossing       1.00      0.56      0.72        25
        giveway       0.99      0.92      0.95       124
        laneend       1.00      1.00      1.00        14
  limitedtraffic       0.76      0.96      0.85        23
        noentry       0.85      0.92      0.88        61
      noparking       0.48      0.95      0.64        22
        parking       0.96      0.89      0.92        27
     rightofway       0.91      0.98      0.94        52
     roundabout       1.00      0.60      0.75        42
          speed       0.93      0.87      0.90        71
           stop       0.97      0.70      0.81        40
trafficdirective       0.86      0.63      0.73        49
 traveldirection       0.00      0.00      0.00        18
         warning       0.63      0.93      0.75        71

        accuracy                           0.82       689
       macro avg       0.78      0.77      0.76       689
    weighted avg       0.84      0.82      0.82       689

<Figure size 800x600 with 0 Axes>
```



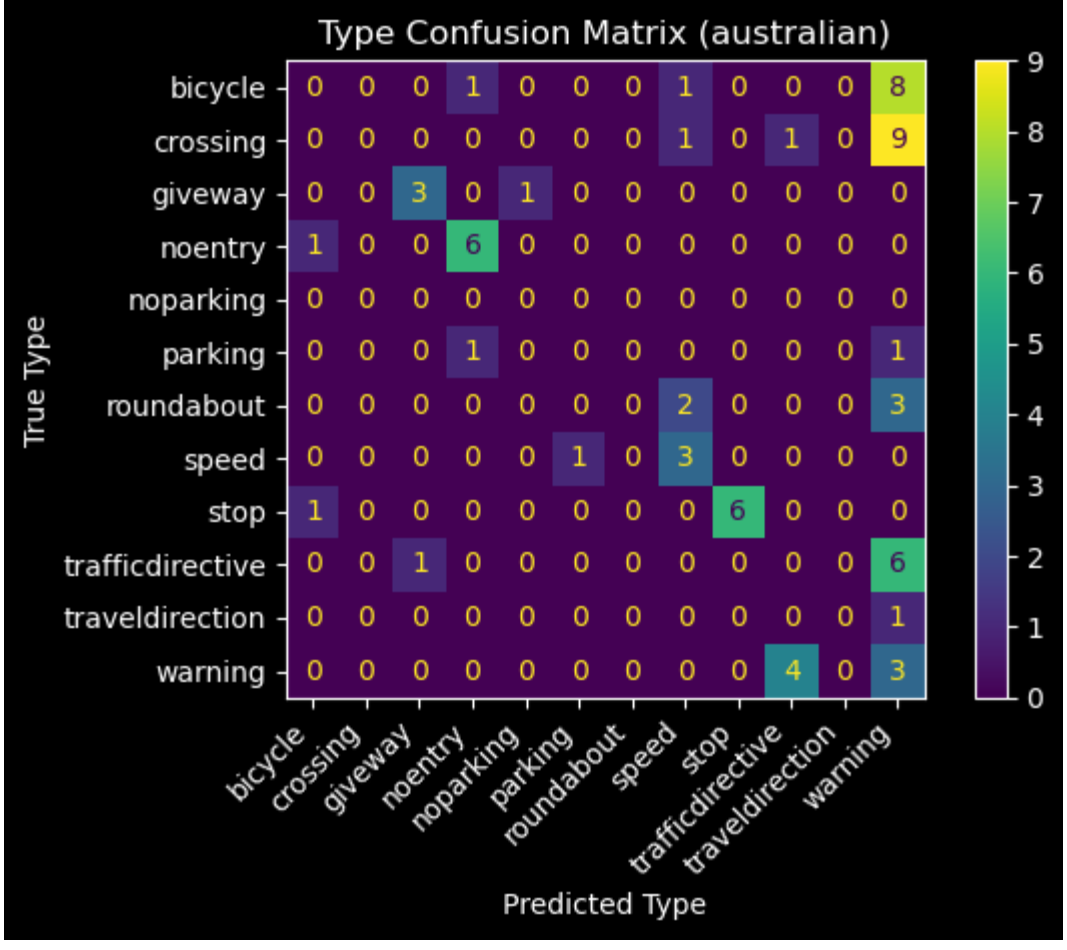

```
<Figure size 800x600 with 0 Axes>
```



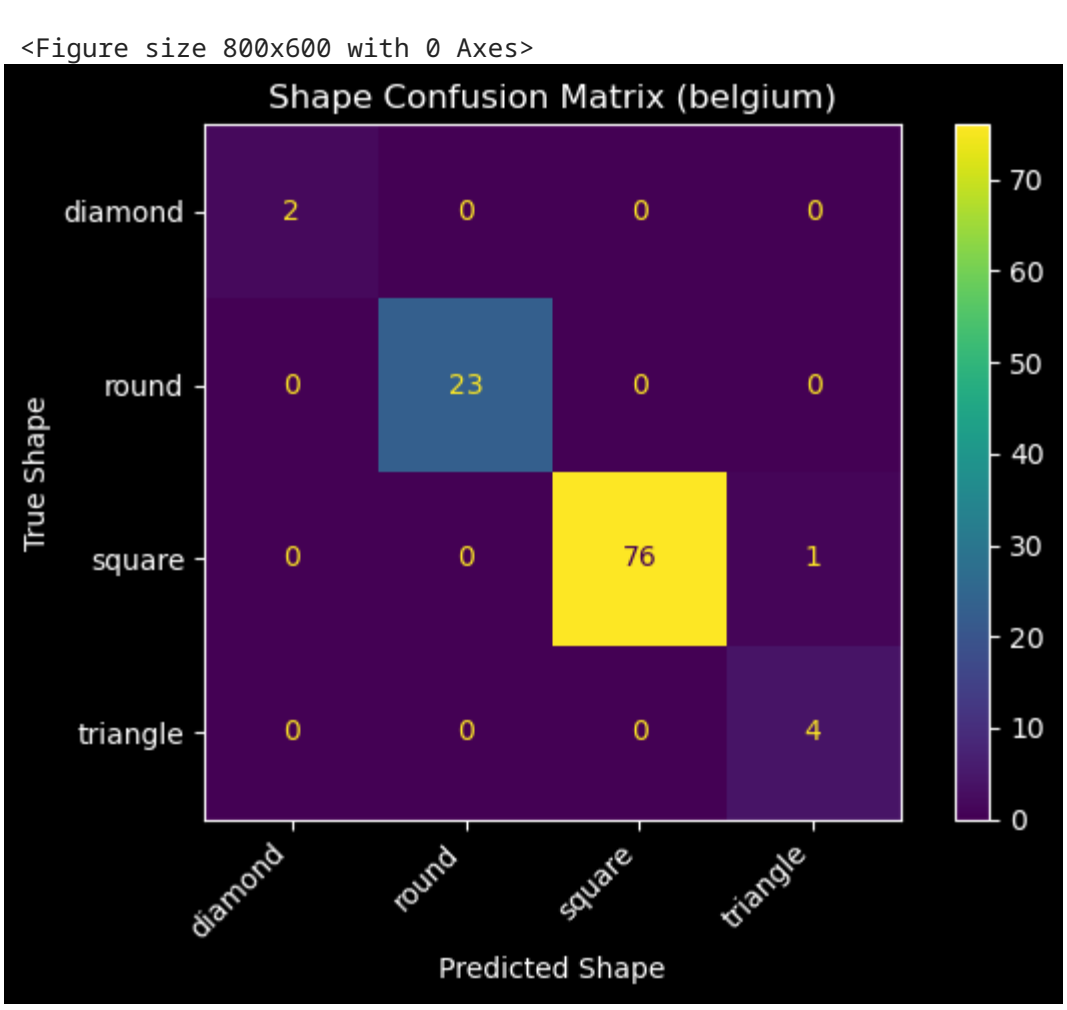

```
3/3 [==============================] - 0s 14ms/step
Confusion matrices created for subcategory: australian
Classification Report - Shape (australian):
              precision    recall  f1-score   support

     diamond       0.00      0.00      0.00        15
         hex       1.00      0.71      0.83         7
       round       0.34      0.68      0.46        19
      square       0.00      0.00      0.00        18
    triangle       0.16      0.44      0.24         9

    accuracy                           0.32        68
   macro avg       0.30      0.37      0.30        68
weighted avg       0.22      0.32      0.24        68

Classification Report - Type (australian):
                  precision    recall  f1-score   support

         bicycle       0.00      0.00      0.00        11
        crossing       0.00      0.00      0.00        11
         giveway       0.75      0.75      0.75         4
         noentry       0.75      0.86      0.80         7
       noparking       0.00      0.00      0.00         1
         parking       0.00      0.00      0.00         2
      roundabout       0.00      0.00      0.00         5
           speed       0.43      0.75      0.55         4
            stop       1.00      0.86      0.92         7
trafficdirective       0.00      0.00      0.00         7
 traveldirection       0.00      0.00      0.00         1
         warning       0.10      0.38      0.15         8

       micro avg       0.32      0.31      0.32        68
       macro avg       0.25      0.30      0.26        68
    weighted avg       0.26      0.31      0.27        68

<Figure size 800x600 with 0 Axes>
```



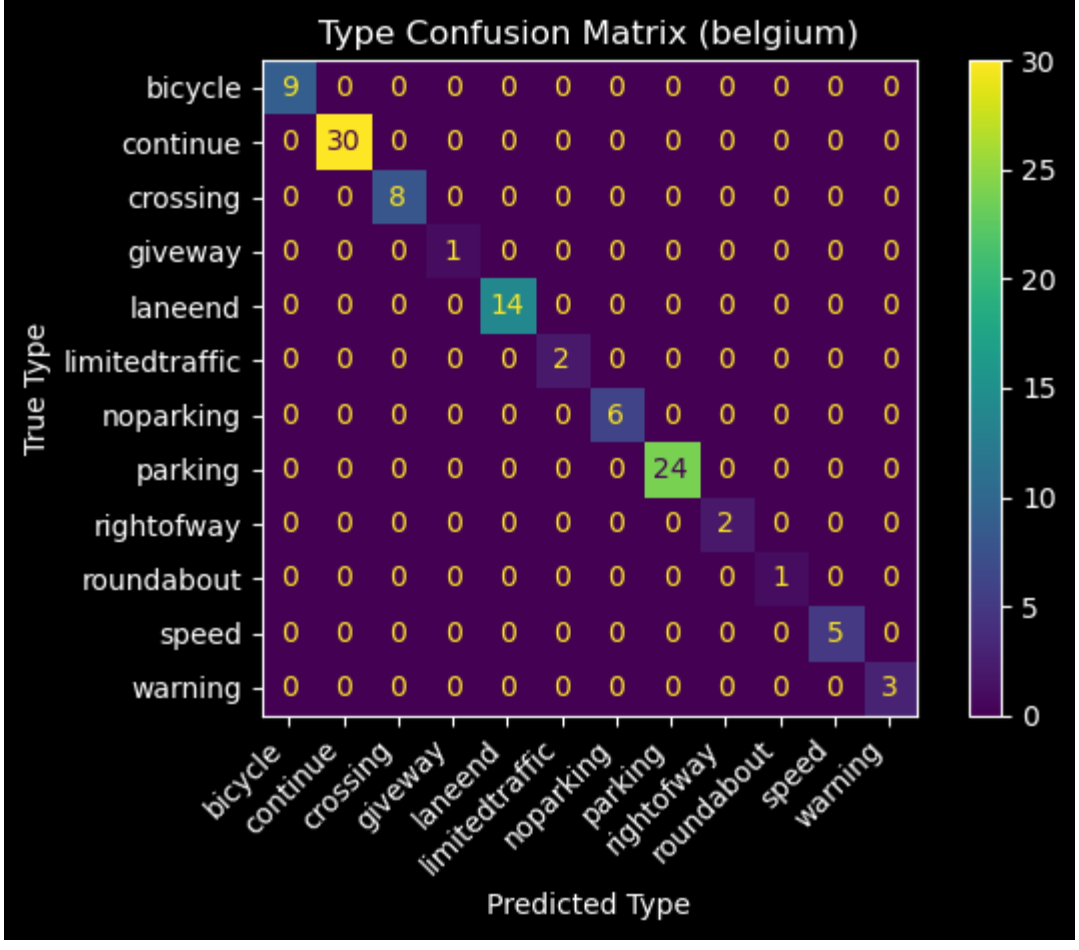

```
<Figure size 800x600 with 0 Axes>
```



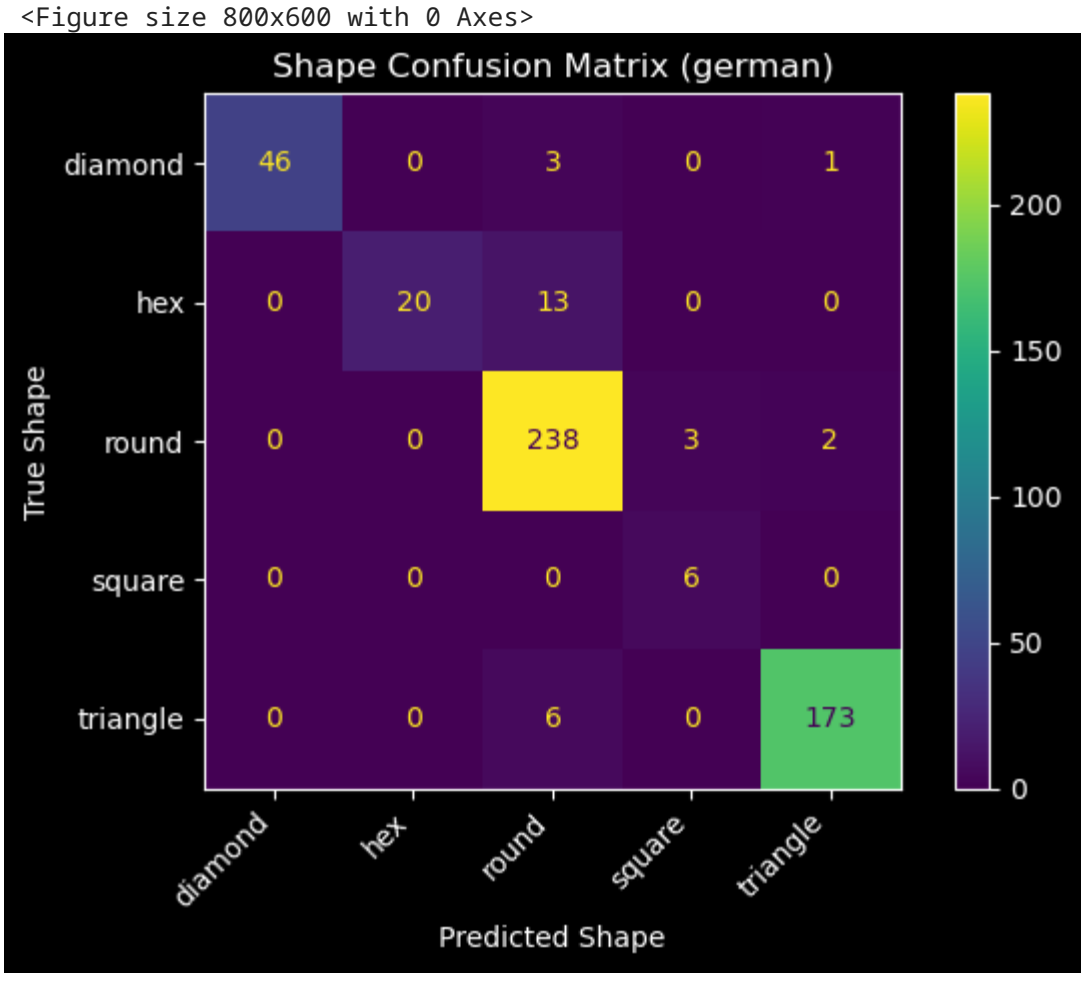

```
4/4 [==============================] - 0s 19ms/step
Confusion matrices created for subcategory: belgium
Classification Report - Shape (belgium):
              precision    recall  f1-score   support

     diamond       1.00      1.00      1.00         2
       round       1.00      1.00      1.00        23
      square       1.00      0.99      0.99        77
    triangle       0.80      1.00      0.89         4

    accuracy                           0.99       106
   macro avg       0.95      1.00      0.97       106
weighted avg       0.99      0.99      0.99       106

Classification Report - Type (belgium):
               precision    recall  f1-score   support

       bicycle       1.00      1.00      1.00         9
      continue       1.00      1.00      1.00        30
      crossing       1.00      1.00      1.00         8
       giveway       1.00      1.00      1.00         1
       laneend       1.00      1.00      1.00        14
limitedtraffic       1.00      1.00      1.00         2
     noparking       1.00      1.00      1.00         6
       parking       1.00      0.96      0.98        25
    rightofway       1.00      1.00      1.00         2
    roundabout       1.00      1.00      1.00         1
         speed       1.00      1.00      1.00         5
       warning       1.00      1.00      1.00         3

     micro avg       1.00      0.99      1.00       106
     macro avg       1.00      1.00      1.00       106
  weighted avg       1.00      0.99      1.00       106

<Figure size 800x600 with 0 Axes>
```



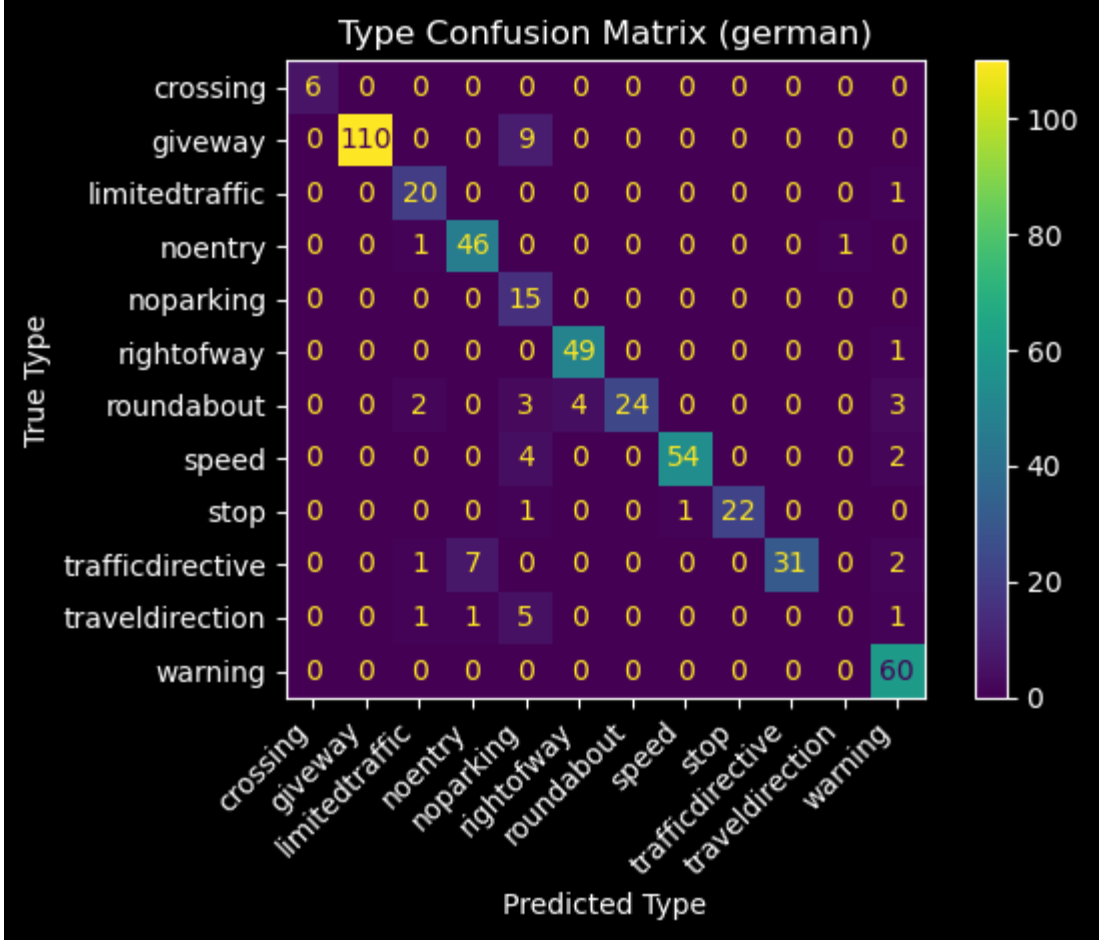

```
<Figure size 800x600 with 0 Axes>
```




```
16/16 [==============================] - 0s 11ms/step
Confusion matrices created for subcategory: german
Classification Report - Shape (german):
              precision    recall  f1-score   support

     diamond       1.00      0.92      0.96        50
         hex       1.00      0.61      0.75        33
       round       0.92      0.98      0.95       243
      square       0.67      1.00      0.80         6
    triangle       0.98      0.97      0.97       179

    accuracy                           0.95       511
   macro avg       0.91      0.89      0.89       511
weighted avg       0.95      0.95      0.94       511

Classification Report - Type (german):
                precision    recall  f1-score   support

      crossing       1.00      1.00      1.00         6
       giveway       1.00      0.92      0.96       119
  limitedtraffic       0.80      0.95      0.87        21
       noentry       0.85      0.92      0.88        50
     noparking       0.41      1.00      0.58        15
     rightofway       0.92      0.98      0.95        50
     roundabout       1.00      0.67      0.80        36
          speed       0.98      0.87      0.92        62
           stop       1.00      0.67      0.80        33
trafficdirective       1.00      0.74      0.85        42
 traveldirection       0.00      0.00      0.00        17
        warning       0.86      1.00      0.92        60

      micro avg       0.90      0.86      0.87       511
      macro avg       0.82      0.81      0.79       511
   weighted avg       0.90      0.86      0.87       511

<Figure size 800x600 with 0 Axes>
```



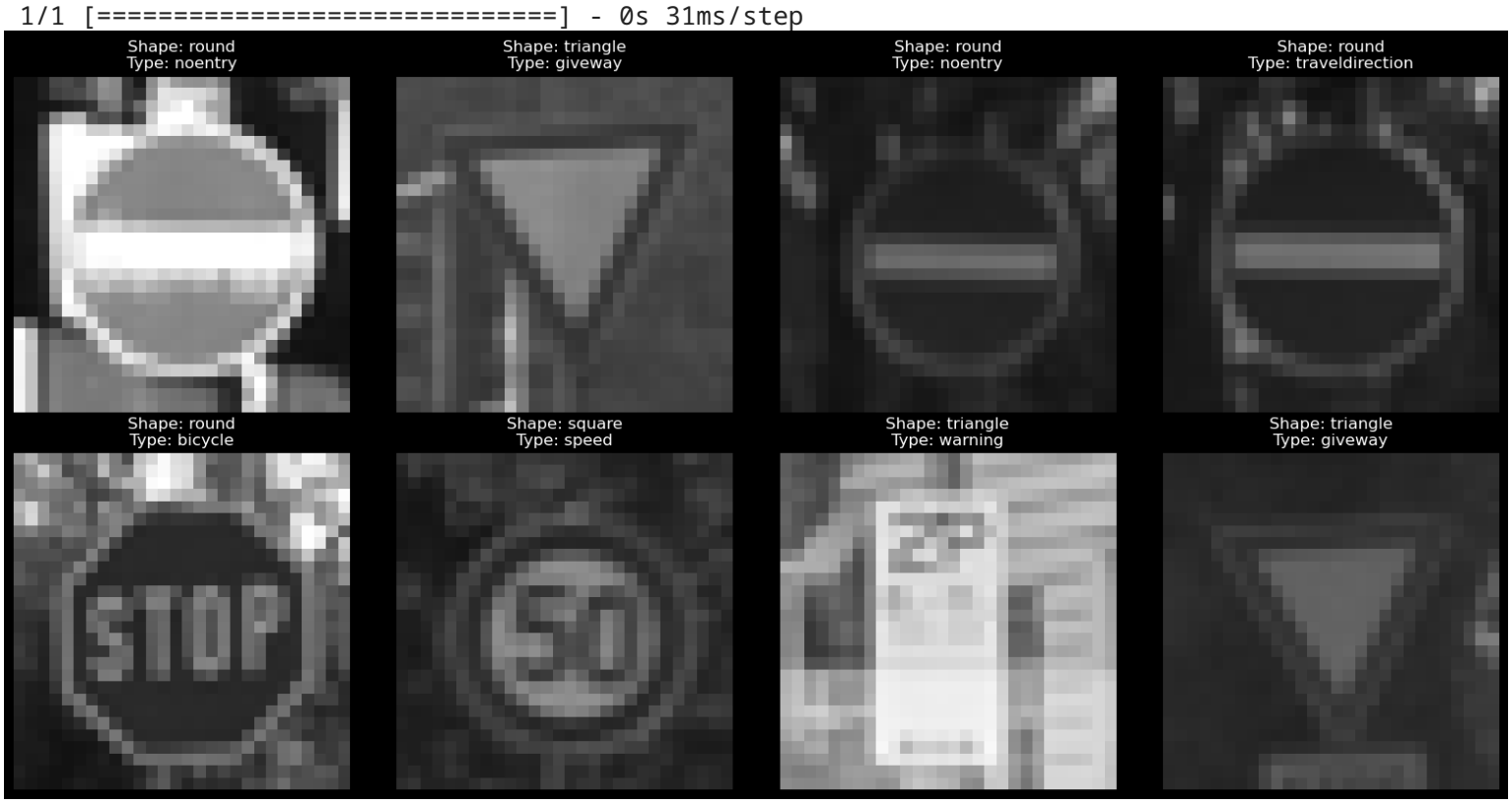
<Figure size 800x600 with 0 Axes>



In [46…

```python
# Create a figure with 2 rows and 4 columns
fig, axes = plt.subplots(2, 4, figsize=(16, 8))

for i, ax in enumerate(axes.flat):
    # Get the file path of a random test image
    random_index = np.random.randint(len(test_df))
    image_path = test_df.iloc[random_index]['image_path']

    # Load and preprocess the image
    img = load_img(image_path, target_size=img_size)
    x = img_to_array(img)
    x = np.expand_dims(x, axis=0)

    # Make predictions
    shape_pred, type_pred = model.predict(x)
    shape_label = shape_encoder.inverse_transform([np.argmax(shape_pred)])[0]
    type_label = type_encoder.inverse_transform([np.argmax(type_pred)])[0]

    # Display the image and predicted labels
    ax.imshow(img)
    ax.set_title(f'Shape: {shape_label}\nType: {type_label}')
    ax.axis('off')

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

```
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 64ms/step
1/1 [==============================] - 0s 31ms/step
```



In [47…

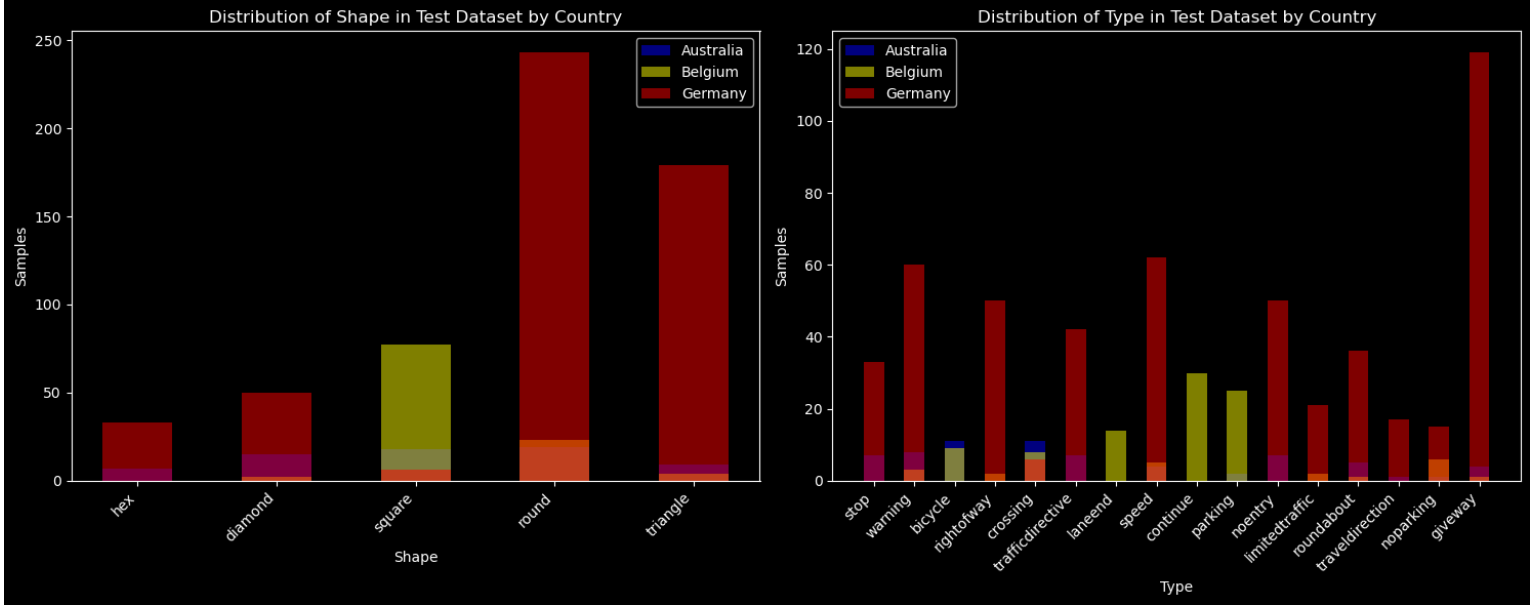
```python
# Create a figure with 2 rows and 4 columns
fig, axes = plt.subplots(2, 4, figsize=(16, 8))
axes = axes.ravel()

# Initialize a counter for wrongly classified images
wrong_counter = 0
```

```python
# Loop until we find 8 wrongly classified images
while wrong_counter < 8:
    # Select a random image path from the test set
    random_index = np.random.choice(len(test_df))
    image_path = test_df.iloc[random_index]['image_path']

    # Load and preprocess the image
    img = load_img(image_path, target_size=img_size)
    x = img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)

    # Make predictions
    preds = model.predict(x)
    shape_pred = preds[0]
    type_pred = preds[1]

    # Get the predicted shape and type labels
    shape_label = shape_encoder.inverse_transform([np.argmax(shape_pred)])[0]
    type_label = type_encoder.inverse_transform([np.argmax(type_pred)])[0]

    # Get the true shape and type labels
    true_shape_label = shape_encoder.inverse_transform([test_df.iloc[random_index]['shape_label'
    true_type_label = type_encoder.inverse_transform([test_df.iloc[random_index]['type_label']])

    # Check if the predicted labels match the true labels
    if shape_label != true_shape_label or type_label != true_type_label:
        # Extract the image name from the image path
        image_name = os.path.basename(image_path)

        # Display the wrongly classified image with the image name
        axes[wrong_counter].imshow(img)
        axes[wrong_counter].axis('off')
        axes[wrong_counter].set_title(f"Image: {image_name}\nPred: {shape_label}, {type_label}\r
        wrong_counter += 1

# Adjust the spacing between subplots
plt.tight_layout()
plt.show()
```

```
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 74ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 51ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 56ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 51ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 64ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 72ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 56ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 62ms/step
```

In [48…

```python
# Create lists to store the image paths, shapes, types, and countries for the test dataset
test_image_paths = []
test_shapes = []
test_types = []
test_countries = []

for shape in os.listdir(test_dataset_path):
    shape_path = os.path.join(test_dataset_path, shape)
    for sign_type in os.listdir(shape_path):
        type_path = os.path.join(shape_path, sign_type)
        for image_file in os.listdir(type_path):
            image_path = os.path.join(type_path, image_file)
            test_image_paths.append(image_path)
            test_shapes.append(shape)
            test_types.append(sign_type)
            if 'australian' in image_path:
                test_countries.append('Australia')
            elif 'belgium' in image_path:
                test_countries.append('Belgium')
            elif 'german' in image_path:
                test_countries.append('Germany')
            else:
                test_countries.append('Other')

# Create a DataFrame with image paths, shapes, types, and countries for the test dataset
test_df = pd.DataFrame({
    'image_path': test_image_paths,
    'shape': test_shapes,
    'type': test_types,
    'country': test_countries
})

# Set the height of the bars
bar_height = 0.5

# Plot the histograms for shapes and types based on country
plt.figure(figsize=(15, 6))
for i, col in enumerate(['shape', 'type']):
    plt.subplot(1, 2, i+1)

    # Get the unique values for the current column
    unique_values = test_df[col].unique()

    # Set the positions of the bars on the x-axis
    x = np.arange(len(unique_values))

    # Plot the histograms for each country
    country_colors = {
        'Australia': 'blue',
        'Belgium': 'yellow',
        'Germany': 'red'
    }

    for j, country in enumerate(['Australia', 'Belgium', 'Germany']):
        counts = test_df[test_df['country'] == country][col].value_counts()
        counts = counts.reindex(unique_values, fill_value=0)
        plt.bar(x, counts, width=bar_height, color=country_colors[country], alpha=0.5, label=cou

    plt.title(f'Distribution of {col.capitalize()} in Test Dataset by Country')
    plt.xlabel(col.capitalize())
    plt.ylabel('Samples')
    plt.xticks(x, unique_values, rotation=45, ha='right')
    plt.legend()
    plt.tight_layout()

plt.show()
```

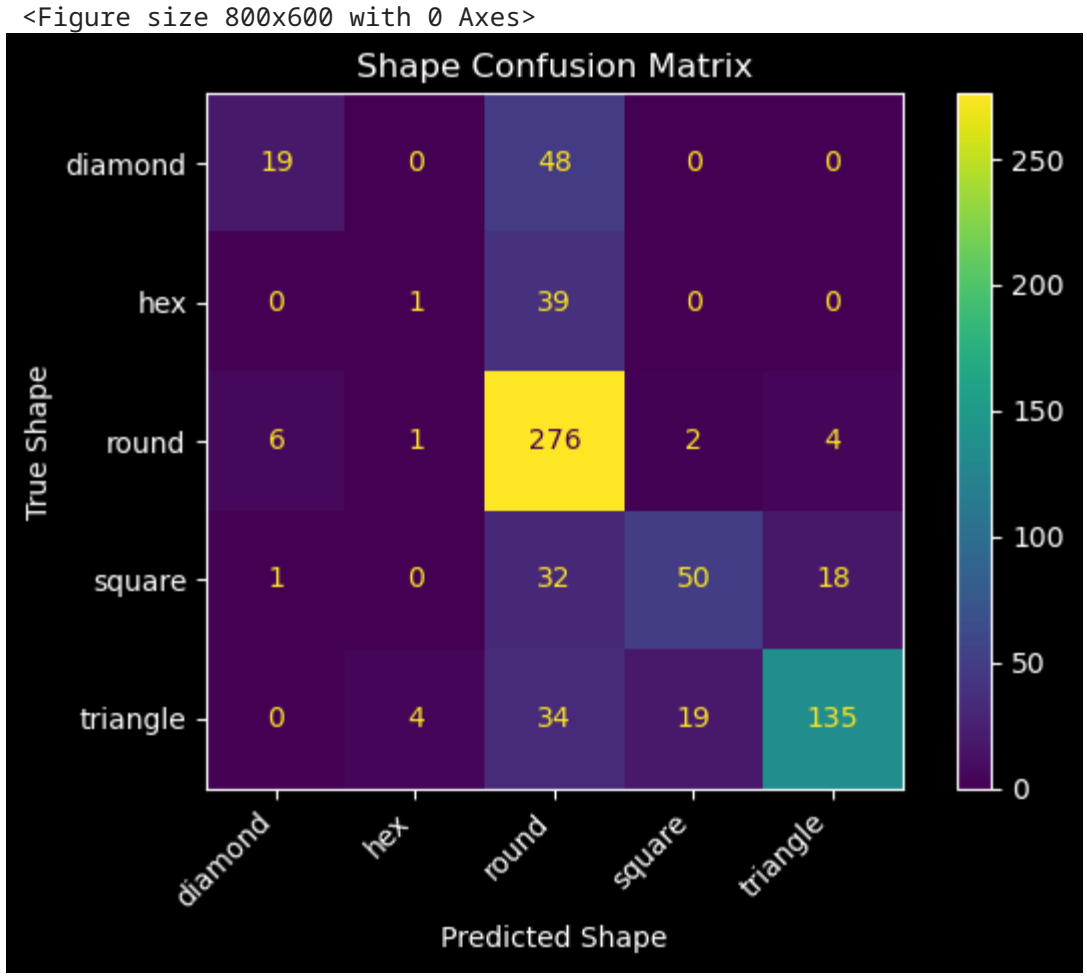


5.6 Comparing Results to Non-Neural Network Models

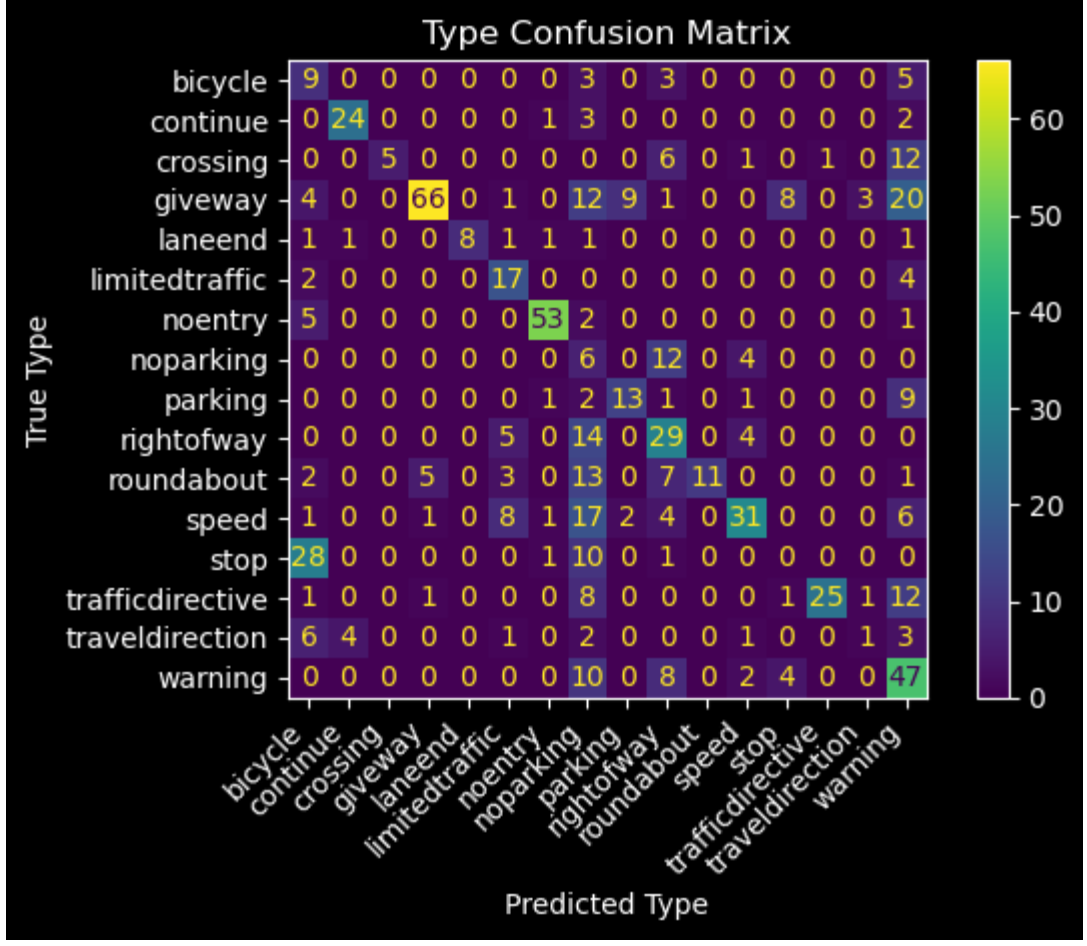
In [58…

```
# Initialize empty lists to store the true labels and predicted labels
shape_true_labels = []
type_true_labels = []
shape_pred_labels = []
type_pred_labels = []

# Load and preprocess the test images
test_images = []
for image_path in test_df['image_path']:
    img = imageio.imread(image_path)
    img = img / 255.0
    test_images.append(img.flatten())

test_images = np.array(test_images)

# Get the true labels for the test images
test_shape_labels = test_df['shape_label'].values
test_type_labels = test_df['type_label'].values

# Generate predictions using the trained Random Forest classifiers
# change test_x to df for random forest, dt for decision tree, and gb for gradient boosted.
shape_preds = shape_gb_best.predict(test_images)
type_preds = type_gb_best.predict(test_images)

# Append the true labels and predicted labels to the lists
shape_true_labels.extend(test_shape_labels)
type_true_labels.extend(test_type_labels)
shape_pred_labels.extend(shape_preds)
type_pred_labels.extend(type_preds)

# Map the predicted labels back to their original names
shape_labels_map = {i: label for i, label in enumerate(shape_encoder.classes_)}
type_labels_map = {i: label for i, label in enumerate(type_encoder.classes_)}

# Create confusion matrices for shape and type
shape_cm = confusion_matrix(shape_true_labels, shape_pred_labels)
type_cm = confusion_matrix(type_true_labels, type_pred_labels)

print("Confusion matrices created.")

# Generate classification report for shape
print("Classification Report - Shape:")
print(classification_report(shape_true_labels, shape_pred_labels, target_names=shape_encoder.cl

# Generate classification report for type
print("Classification Report - Type:")
print(classification_report(type_true_labels, type_pred_labels, target_names=type_encoder.class

# Plot the confusion matrix for shape
plt.figure(figsize=(8, 6))
ConfusionMatrixDisplay(shape_cm, display_labels=shape_encoder.classes_).plot()
plt.title("Shape Confusion Matrix")
plt.xlabel("Predicted Shape")
plt.ylabel("True Shape")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

# Plot the confusion matrix for type
plt.figure(figsize=(8, 6))
ConfusionMatrixDisplay(type_cm, display_labels=type_encoder.classes_).plot()
plt.title("Type Confusion Matrix")
plt.xlabel("Predicted Type")
plt.ylabel("True Type")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```

```
Confusion matrices created.
Classification Report - Shape:
              precision    recall  f1-score   support

     diamond       0.73      0.28      0.41        67
         hex       0.17      0.03      0.04        40
       round       0.64      0.96      0.77       289
      square       0.70      0.50      0.58       101
    triangle       0.86      0.70      0.77       192

    accuracy                           0.70       689
   macro avg       0.62      0.49      0.52       689
weighted avg       0.69      0.70      0.67       689

Classification Report - Type:
                  precision    recall  f1-score   support

         bicycle       0.15      0.45      0.23        20
        continue       0.83      0.80      0.81        30
        crossing       1.00      0.20      0.33        25
         giveway       0.90      0.53      0.67       124
         laneend       1.00      0.57      0.73        14
  limitedtraffic       0.47      0.74      0.58        23
         noentry       0.91      0.87      0.89        61
       noparking       0.06      0.27      0.10        22
         parking       0.54      0.48      0.51        27
      rightofway       0.40      0.56      0.47        52
      roundabout       1.00      0.26      0.42        42
           speed       0.70      0.44      0.54        71
            stop       0.00      0.00      0.00        40
trafficdirective       0.96      0.51      0.67        49
   traveldirection       0.20      0.06      0.09        18
           warning       0.38      0.66      0.48        71

          accuracy                           0.50       689
         macro avg       0.60      0.46      0.47       689
      weighted avg       0.66      0.50      0.53       689

<Figure size 800x600 with 0 Axes>
```




```
<Figure size 800x600 with 0 Axes>
```

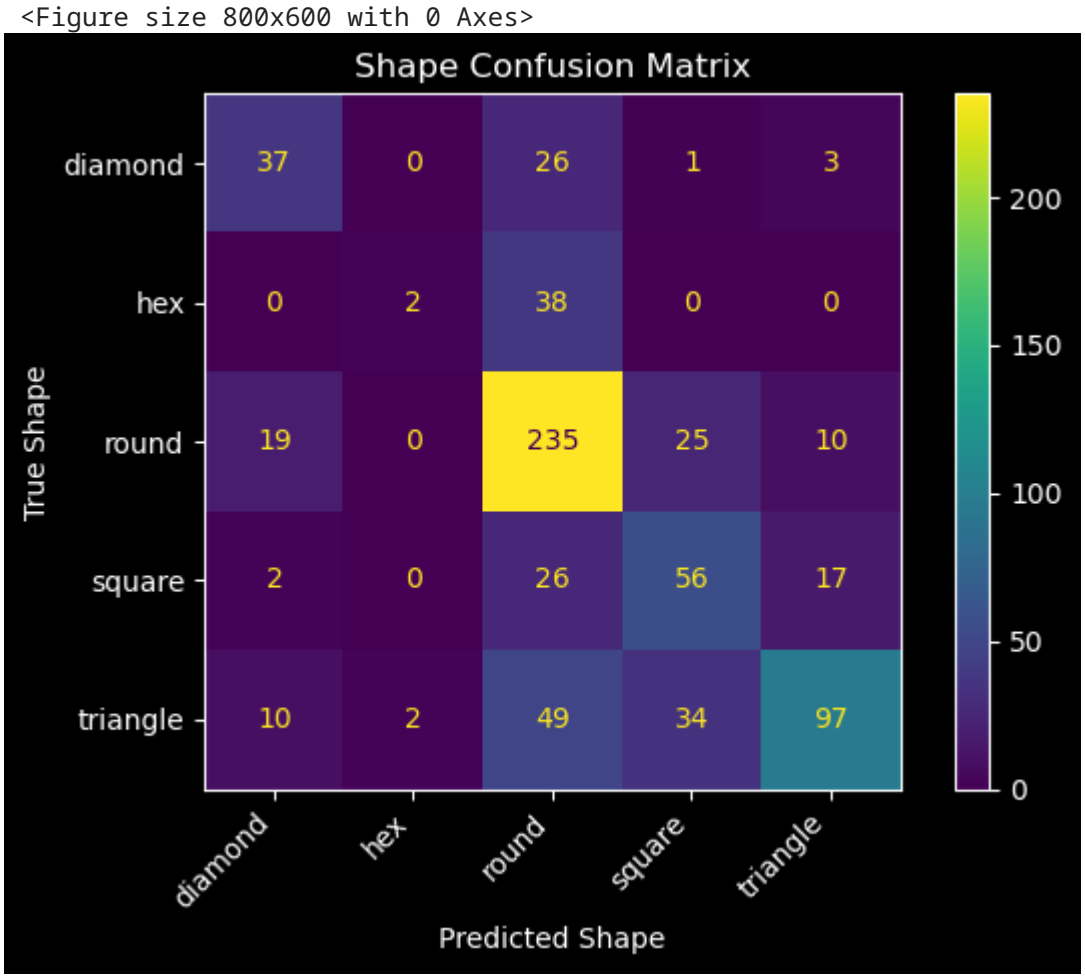


In [64…

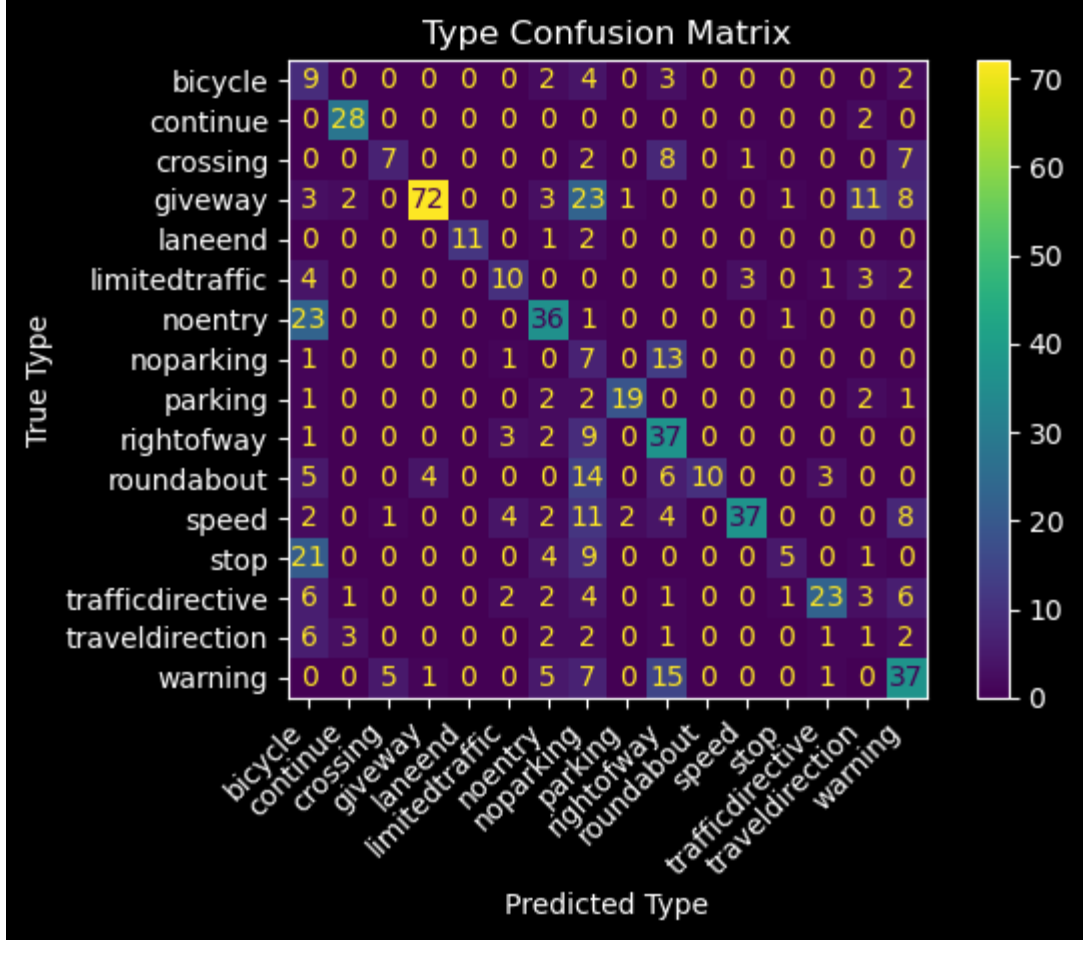
```
# Initialize empty lists to store the true labels and predicted labels
shape_true_labels = []
type_true_labels = []
shape_pred_labels = []
type_pred_labels = []

# Load and preprocess the test images
test_images = []
for image_path in test_df['image_path']:
    img = imageio.imread(image_path)
    img = img / 255.0
    test_images.append(img.flatten())

test_images = np.array(test_images)

# Get the true labels for the test images
test_shape_labels = test_df['shape_label'].values
test_type_labels = test_df['type_label'].values

# Generate predictions using the trained logistic regression classifiers
shape_preds = shape_classifier.predict(test_images)
type_preds = type_classifier.predict(test_images)

# Append the true labels and predicted labels to the lists
shape_true_labels.extend(test_shape_labels)
type_true_labels.extend(test_type_labels)
shape_pred_labels.extend(shape_preds)
type_pred_labels.extend(type_preds)

# Map the predicted labels back to their original names
shape_labels_map = {i: label for i, label in enumerate(shape_encoder.classes_)}
type_labels_map = {i: label for i, label in enumerate(type_encoder.classes_)}

# Create confusion matrices for shape and type
shape_cm = confusion_matrix(shape_true_labels, shape_pred_labels)
type_cm = confusion_matrix(type_true_labels, type_pred_labels)

print("Confusion matrices created.")

# Generate classification report for shape
print("Classification Report - Shape:")
print(classification_report(shape_true_labels, shape_pred_labels, target_names=shape_encoder.cl

# Generate classification report for type
print("Classification Report - Type:")
print(classification_report(type_true_labels, type_pred_labels, target_names=type_encoder.class

# Plot the confusion matrix for shape
plt.figure(figsize=(8, 6))
ConfusionMatrixDisplay(shape_cm, display_labels=shape_encoder.classes_).plot()
plt.title("Shape Confusion Matrix")
plt.xlabel("Predicted Shape")
plt.ylabel("True Shape")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

# Plot the confusion matrix for type
plt.figure(figsize=(8, 6))
ConfusionMatrixDisplay(type_cm, display_labels=type_encoder.classes_).plot()
plt.title("Type Confusion Matrix")
plt.xlabel("Predicted Type")
plt.ylabel("True Type")
plt.xticks(rotation=45, ha="right")
```

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

```
Confusion matrices created.
Classification Report - Shape:
              precision    recall  f1-score   support

     diamond       0.54      0.55      0.55        67
         hex       0.50      0.05      0.09        40
       round       0.63      0.81      0.71       289
      square       0.48      0.55      0.52       101
    triangle       0.76      0.51      0.61       192

    accuracy                           0.62       689
   macro avg       0.58      0.50      0.49       689
weighted avg       0.63      0.62      0.60       689

Classification Report - Type:
                 precision    recall  f1-score   support

         bicycle       0.11      0.45      0.18        20
        continue       0.82      0.93      0.87        30
        crossing       0.54      0.28      0.37        25
         giveway       0.94      0.58      0.72       124
         laneend       1.00      0.79      0.88        14
  limitedtraffic       0.50      0.43      0.47        23
         noentry       0.59      0.59      0.59        61
       noparking       0.07      0.32      0.12        22
         parking       0.86      0.70      0.78        27
       rightofway       0.42      0.71      0.53        52
       roundabout       1.00      0.24      0.38        42
            speed       0.90      0.52      0.66        71
             stop       0.62      0.12      0.21        40
  trafficdirective       0.79      0.47      0.59        49
   traveldirection       0.04      0.06      0.05        18
          warning       0.51      0.52      0.51        71

         accuracy                           0.51       689
        macro avg       0.61      0.48      0.49       689
     weighted avg       0.68      0.51      0.55       689

<Figure size 800x600 with 0 Axes>
```




```
<Figure size 800x600 with 0 Axes>
```



6.0 Advanced Tuning

6.1 Transfer Learning

- Although it explicitly says not to use pre-trained models, Azadeh recommended creating a third Model utilizing advanced tuning such as Transfer Learning to:
  - Increase the generalisability of our models
  - Improve and make our models able to classify things outside of the initially trained domain
- Azadeh suggested doing this in order to improve the likelihood of achieving a High Distinction (HD).
- Note: to evaluate the results from this new model, you must go back and re-use the previously run cells for evaluation.

Advanced Tuning

- We also experimented with a third model that utilised advanced tuning, such as transfer learning, to further improve generalizability.
- However, this proved to be challenging, as many pre-trained models require resizing images which led to:
  - Distortion
  - Compromising the quality of the extracted features
- However, we will explore the performance of the generalizability of these in our independent evaluation.

In [85…

```
train_generator = train_datagen.flow_from_dataframe(
    dataframe=df.iloc[train_indices],
    x_col='image_path',
    y_col=['shape_label', 'type_label'],
    target_size=(32, 32),  # Resize to (32, 32)
    batch_size=32,
    class_mode='multi_output'
)

val_generator = val_datagen.flow_from_dataframe(
    dataframe=df.iloc[val_indices],
    x_col='image_path',
    y_col=['shape_label', 'type_label'],
    target_size=(32, 32),  # Resize to (32, 32)
    batch_size=32,
    class_mode='multi_output'
)
```

```
Found 2959 validated image filenames.
Found 740 validated image filenames.
```

In [86…

```
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
```

In [87…

```
# Freeze the layers of the pre-trained model
for layer in base_model.layers:
    layer.trainable = False

# Create the model architecture
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu', kernel_regularizer=l2(0.001))(x)
```

```python
# Output layers for shape and type
shape_output = Dense(num_shape_classes, activation='softmax', name='shape')(x)
type_output = Dense(num_type_classes, activation='softmax', name='type')(x)

# Create the model
model = Model(inputs=base_model.input, outputs=[shape_output, type_output])

# Compile the model
model.compile(optimizer='adam',
              loss={'shape': 'sparse_categorical_crossentropy',
                    'type': 'sparse_categorical_crossentropy'},
              metrics=['accuracy'])

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_indices) // 32,
    validation_data=val_generator,
    validation_steps=np.ceil(len(val_indices) / 32).astype(int),
    epochs=20
)
```

```
Epoch 1/20
2024-05-17 15:56:44.531321: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CP
U:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignor
e this message): INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholde
r/_0' with dtype int32
         [[{{node Placeholder/_0}}]]
92/92 [==============================] - ETA: 0s - loss: 3.6319 - shape_loss: 1.1445 - type_
loss: 2.3144 - shape_accuracy: 0.5702 - type_accuracy: 0.3184
2024-05-17 15:56:59.782042: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CP
U:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignor
e this message): INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholde
r/_0' with dtype int32
         [[{{node Placeholder/_0}}]]
```

```
92/92 [==============================] - 19s 182ms/step - loss: 3.6319 - shape_loss: 1.1445
- type_loss: 2.3144 - shape_accuracy: 0.5702 - type_accuracy: 0.3184 - val_loss: 3.2367 - va
l_shape_loss: 1.1645 - val_type_loss: 2.0085 - val_shape_accuracy: 0.5054 - val_type_accurac
y: 0.4122
Epoch 2/20
92/92 [==============================] - 16s 169ms/step - loss: 2.6759 - shape_loss: 0.8341
- type_loss: 1.7725 - shape_accuracy: 0.6949 - type_accuracy: 0.4633 - val_loss: 2.7393 - va
l_shape_loss: 0.8372 - val_type_loss: 1.8269 - val_shape_accuracy: 0.7189 - val_type_accurac
y: 0.4500
Epoch 3/20
92/92 [==============================] - 16s 170ms/step - loss: 2.3117 - shape_loss: 0.7203
- type_loss: 1.5098 - shape_accuracy: 0.7468 - type_accuracy: 0.5360 - val_loss: 2.3645 - va
l_shape_loss: 0.8054 - val_type_loss: 1.4713 - val_shape_accuracy: 0.7081 - val_type_accurac
y: 0.5689
Epoch 4/20
92/92 [==============================] - 16s 176ms/step - loss: 2.0311 - shape_loss: 0.6468
- type_loss: 1.2910 - shape_accuracy: 0.7745 - type_accuracy: 0.6180 - val_loss: 2.0372 - va
l_shape_loss: 0.6618 - val_type_loss: 1.2771 - val_shape_accuracy: 0.7581 - val_type_accurac
y: 0.6149
Epoch 5/20
92/92 [==============================] - 17s 189ms/step - loss: 1.9146 - shape_loss: 0.5933
- type_loss: 1.2180 - shape_accuracy: 0.7984 - type_accuracy: 0.6382 - val_loss: 2.0722 - va
l_shape_loss: 0.7101 - val_type_loss: 1.2542 - val_shape_accuracy: 0.7419 - val_type_accurac
y: 0.5919
Epoch 6/20
92/92 [==============================] - 17s 188ms/step - loss: 1.8038 - shape_loss: 0.5605
- type_loss: 1.1317 - shape_accuracy: 0.7991 - type_accuracy: 0.6604 - val_loss: 1.8962 - va
l_shape_loss: 0.5829 - val_type_loss: 1.1979 - val_shape_accuracy: 0.7959 - val_type_accurac
y: 0.6514
Epoch 7/20
92/92 [==============================] - 15s 161ms/step - loss: 1.7175 - shape_loss: 0.5437
- type_loss: 1.0546 - shape_accuracy: 0.8148 - type_accuracy: 0.6768 - val_loss: 1.7338 - va
l_shape_loss: 0.5179 - val_type_loss: 1.0940 - val_shape_accuracy: 0.8230 - val_type_accurac
y: 0.6622
Epoch 8/20
92/92 [==============================] - 16s 173ms/step - loss: 1.6182 - shape_loss: 0.4982
- type_loss: 0.9952 - shape_accuracy: 0.8312 - type_accuracy: 0.7004 - val_loss: 1.7230 - va
l_shape_loss: 0.5827 - val_type_loss: 1.0131 - val_shape_accuracy: 0.8054 - val_type_accurac
y: 0.6851
Epoch 9/20
92/92 [==============================] - 17s 181ms/step - loss: 1.5154 - shape_loss: 0.4615
- type_loss: 0.9248 - shape_accuracy: 0.8456 - type_accuracy: 0.7226 - val_loss: 1.8230 - va
l_shape_loss: 0.5927 - val_type_loss: 1.0996 - val_shape_accuracy: 0.7824 - val_type_accurac
y: 0.6432
Epoch 10/20
92/92 [==============================] - 20s 212ms/step - loss: 1.5679 - shape_loss: 0.4936
- type_loss: 0.9413 - shape_accuracy: 0.8237 - type_accuracy: 0.7134 - val_loss: 1.6506 - va
l_shape_loss: 0.5119 - val_type_loss: 1.0029 - val_shape_accuracy: 0.8189 - val_type_accurac
y: 0.7027
Epoch 11/20
92/92 [==============================] - 20s 213ms/step - loss: 1.4312 - shape_loss: 0.4239
- type_loss: 0.8704 - shape_accuracy: 0.8647 - type_accuracy: 0.7376 - val_loss: 1.5820 - va
l_shape_loss: 0.4661 - val_type_loss: 0.9777 - val_shape_accuracy: 0.8351 - val_type_accurac
y: 0.6770
Epoch 12/20
92/92 [==============================] - 16s 170ms/step - loss: 1.4468 - shape_loss: 0.4633
- type_loss: 0.8438 - shape_accuracy: 0.8299 - type_accuracy: 0.7410 - val_loss: 1.5624 - va
l_shape_loss: 0.5543 - val_type_loss: 0.8670 - val_shape_accuracy: 0.8041 - val_type_accurac
y: 0.7311
Epoch 13/20
92/92 [==============================] - 16s 178ms/step - loss: 1.3592 - shape_loss: 0.4151
- type_loss: 0.8018 - shape_accuracy: 0.8671 - type_accuracy: 0.7585 - val_loss: 1.4781 - va
l_shape_loss: 0.4775 - val_type_loss: 0.8574 - val_shape_accuracy: 0.8405 - val_type_accurac
y: 0.7568
Epoch 14/20
92/92 [==============================] - 17s 186ms/step - loss: 1.3690 - shape_loss: 0.4266
- type_loss: 0.7975 - shape_accuracy: 0.8551 - type_accuracy: 0.7663 - val_loss: 1.3765 - va
l_shape_loss: 0.4287 - val_type_loss: 0.8021 - val_shape_accuracy: 0.8554 - val_type_accurac
y: 0.7554
Epoch 15/20
92/92 [==============================] - 16s 170ms/step - loss: 1.3121 - shape_loss: 0.3970
- type_loss: 0.7686 - shape_accuracy: 0.8623 - type_accuracy: 0.7677 - val_loss: 1.8187 - va
l_shape_loss: 0.6719 - val_type_loss: 0.9997 - val_shape_accuracy: 0.7595 - val_type_accurac
y: 0.6743
Epoch 16/20
92/92 [==============================] - 17s 187ms/step - loss: 1.3390 - shape_loss: 0.4120
- type_loss: 0.7778 - shape_accuracy: 0.8548 - type_accuracy: 0.7680 - val_loss: 1.4335 - va
l_shape_loss: 0.4255 - val_type_loss: 0.8578 - val_shape_accuracy: 0.8554 - val_type_accurac
y: 0.7568
Epoch 17/20
92/92 [==============================] - 16s 178ms/step - loss: 1.2881 - shape_loss: 0.3848
- type_loss: 0.7522 - shape_accuracy: 0.8637 - type_accuracy: 0.7677 - val_loss: 1.3438 - va
l_shape_loss: 0.3881 - val_type_loss: 0.8041 - val_shape_accuracy: 0.8703 - val_type_accurac
y: 0.7622
Epoch 18/20
92/92 [==============================] - 18s 194ms/step - loss: 1.2767 - shape_loss: 0.3837
- type_loss: 0.7411 - shape_accuracy: 0.8746 - type_accuracy: 0.7731 - val_loss: 1.5949 - va
l_shape_loss: 0.5529 - val_type_loss: 0.8892 - val_shape_accuracy: 0.8081 - val_type_accurac
y: 0.7135
Epoch 19/20
92/92 [==============================] - 16s 176ms/step - loss: 1.2866 - shape_loss: 0.3839
- type_loss: 0.7488 - shape_accuracy: 0.8719 - type_accuracy: 0.7649 - val_loss: 1.4633 - va
l_shape_loss: 0.5142 - val_type_loss: 0.7942 - val_shape_accuracy: 0.7973 - val_type_accurac
```

```
y: 0.7554
Epoch 20/20
92/92 [==============================] - 15s 168ms/step - loss: 1.2031 - shape_loss: 0.3425
- type_loss: 0.7051 - shape_accuracy: 0.8890 - type_accuracy: 0.7793 - val_loss: 1.4142 - va
l_shape_loss: 0.4242 - val_type_loss: 0.8345 - val_shape_accuracy: 0.8500 - val_type_accurac
y: 0.7446
```

as observed Transfer Learning does X, we can now go back up to the Independent Evaluation to re-test our model noting we must covert our images to 30x30 since that is what the Transfer Learning's minimum size requirement is. We expect this to mess around with our results a little bit, but hopefully not too much.

*7.0 Unsupervised Learning

these were primarily used for feature extraction.

7.1 K-Means

In [60…

```python
# Function to load and preprocess images
def load_and_preprocess_image(path):
    img = Image.open(path).convert('L')  # Converts image to grayscale
    img = img.resize((28, 28))
    img_array = np.array(img)
    img_array = img_array.flatten()
    return img_array

# Load and preprocesses the images
image_paths = df['image_path'].values
X = np.array([load_and_preprocess_image(path) for path in image_paths])

# Scales the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Specify the number of clusters (k)
# k = 5  # simulate shape
k = 17  # simulate type

# Create and fit the k-means model
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(X_scaled)

# Get the cluster labels for each image
labels = kmeans.labels_

# Print the number of images per cluster
cluster_sizes = [np.sum(labels == cluster_id) for cluster_id in range(k)]
print("Number of images per cluster:")

for cluster_id, size in enumerate(cluster_sizes):
    print(f"Cluster {cluster_id}: {size} images")


# Display 5 random images from each cluster
num_images_per_cluster = 5
num_cols = 5
num_rows = (k * num_images_per_cluster + num_cols - 1) // num_cols
fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 2 * num_rows),
                         subplot_kw={'adjustable': 'box', 'aspect': 1})
axes = axes.ravel()

for cluster_id in range(k):
    cluster_images = [image_paths[i] for i in range(len(labels)) if labels[i] == cluster_
    for i, image_path in enumerate(np.random.choice(cluster_images, size=num_images_per_c
        img = Image.open(image_path).convert('L')  # Converts images to grayscale
        axes[cluster_id * num_images_per_cluster + i].imshow(img, cmap='gray')
        if i == 0:
            axes[cluster_id * num_images_per_cluster + i].set_title(f"Cluster {cluster_id
        axes[cluster_id * num_images_per_cluster + i].axis('off')

# Remove any unused subplots
for i in range(k * num_images_per_cluster, len(axes)):
    axes[i].axis('off')
plt.tight_layout(pad=0.5, h_pad=0.1, w_pad=0.1)
plt.show()
```

```
Number of images per cluster:
Cluster 0: 134 images
Cluster 1: 147 images
Cluster 2: 362 images
Cluster 3: 195 images
Cluster 4: 296 images
Cluster 5: 154 images
Cluster 6: 144 images
Cluster 7: 139 images
Cluster 8: 310 images
Cluster 9: 448 images
Cluster 10: 105 images
Cluster 11: 266 images
Cluster 12: 129 images
Cluster 13: 109 images
Cluster 14: 178 images
Cluster 15: 197 images
Cluster 16: 386 images
```

- Clustering Effectiveness:
  - The k-means clustering algorithm demonstrates its effectiveness in clustering similar images together which can help us determine the approximate number of classes within the dataset.
  - The resulting clusters contain images that share similar shapes and mostly belonging to the same class (shape, and type), indicating the algorithm's ability to group related images together.

- Image Characteristics Influencing Clustering:
  - During the clustering process, k-means algorithms consider various image characteristics that we explored, such as pixel intensity and sharpness..
  - Some of the generated clusters are based on these specific image properties, highlighting their importance in distinguishing between different classes of road signs. These definitively resulted in a few misclassified classes that are based on the pixel intensity (overall blackness or whiteness) or sharpess rather than the content of the images.

- Experimental Insights:
  - The additional experimentation conducted using the k-means algorithm provided valuable insights into alternative methodologies for accurately classifying road signs.
  - By exploring the role of pixel intensity and sharpness in the clustering process, this experiment contributes to a deeper understanding of the factors influencing the correct categorization of road signs.

- Potential for Further Optimization:
  - While the k-means algorithm proves to be reasonably effective in clustering road sign images, there may be opportunities for further optimization.
  - Fine-tuning the algorithm's parameters, such as the number of clusters or the distance metric used, could potentially enhance the accuracy and precision of the resulting clusters.

- Unsupervised Learning Approach:
  - In the absence of labelled training data, an unsupervised learning approach, such as k-means clustering, would have been a viable initial step.
  - By applying unsupervised learning techniques, we could have created a model to uncover the underlying structure of the data if the dataset were larger, and less organised. This would help us determine the number of distinct classes based on shape and type.

7.2 Latent Dirichlet Allocation (LDA)

In [ ]:
```python
# Extract features from images
image_features = []
for image in image_data:
    # Perform feature extraction on the image (e.g., pixel intensities)
    features = image.flatten()
    image_features.append(' '.join(map(str, features)))  # Convert features to a string

# Create a CountVectorizer to convert image features into a document-term matrix
vectorizer = CountVectorizer()
feature_matrix = vectorizer.fit_transform(image_features)

# Apply Latent Dirichlet Allocation
num_topics = 5  # Specify the desired number of topics
lda = LatentDirichletAllocation(n_components=num_topics, random_state=42)
lda.fit(feature_matrix)

# Get the topic distributions for each image
topic_distributions = lda.transform(feature_matrix)

# Print the top words for each topic
feature_names = vectorizer.get_feature_names_out()
for topic_idx, topic in enumerate(lda.components_):
    top_words = [feature_names[i] for i in topic.argsort()[:-10:-1]]
    print(f"Topic {topic_idx + 1}: {' '.join(top_words)}")
```

- Latent Dirichlet Allocation (LDA):
  - Generative probabilistic model for discovering underlying topics in a collection of documents.
  - Widely used in natural language processing and text mining applications.

- Applying LDA to our dataset:
  - Preprocessed and tokenized the text data to prepare it for LDA.
  - Experimented with different numbers of topics and hyperparameter settings.

- Insights gained from LDA:
  - Identified latent topics that provide a meaningful representation of the document collection.
  - Utilized topic distributions as features for downstream tasks such as document classification or clustering.

8.0 Summary of Results

In [72…
```python
# Print the current shape of X_val
print("Original shape of X_val:", X_val.shape)

# Reshape the input data to match the expected input shape of the model
num_samples = X_val.shape[0]
X_val_reshaped = X_val.reshape(num_samples, 28, 28)
X_val_reshaped = np.repeat(X_val_reshaped[..., np.newaxis], 3, axis=-1)

# Neural Network
```

```
shape_pred_nn = np.argmax(model.predict(X_val_reshaped)[0], axis=1)
type_pred_nn = np.argmax(model.predict(X_val_reshaped)[1], axis=1)

shape_acc_nn = accuracy_score(y_val_shape, shape_pred_nn)
shape_precision_nn = precision_score(y_val_shape, shape_pred_nn, average='weighted')
shape_recall_nn = recall_score(y_val_shape, shape_pred_nn, average='weighted')
shape_f1_nn = f1_score(y_val_shape, shape_pred_nn, average='weighted')

type_acc_nn = accuracy_score(y_val_type, type_pred_nn)
type_precision_nn = precision_score(y_val_type, type_pred_nn, average='weighted')
type_recall_nn = recall_score(y_val_type, type_pred_nn, average='weighted')
type_f1_nn = f1_score(y_val_type, type_pred_nn, average='weighted')

# Gradient Boosting
shape_pred_gb = shape_gb_best.predict(X_val)
type_pred_gb = type_gb_best.predict(X_val)

shape_acc_gb = accuracy_score(y_val_shape, shape_pred_gb)
shape_precision_gb = precision_score(y_val_shape, shape_pred_gb, average='weighted')
shape_recall_gb = recall_score(y_val_shape, shape_pred_gb, average='weighted')
shape_f1_gb = f1_score(y_val_shape, shape_pred_gb, average='weighted')

type_acc_gb = accuracy_score(y_val_type, type_pred_gb)
type_precision_gb = precision_score(y_val_type, type_pred_gb, average='weighted')
type_recall_gb = recall_score(y_val_type, type_pred_gb, average='weighted')
type_f1_gb = f1_score(y_val_type, type_pred_gb, average='weighted')

# Random Forest
shape_pred_rf = shape_rf_best.predict(X_val)
type_pred_rf = type_rf_best.predict(X_val)

shape_acc_rf = accuracy_score(y_val_shape, shape_pred_rf)
shape_precision_rf = precision_score(y_val_shape, shape_pred_rf, average='weighted')
shape_recall_rf = recall_score(y_val_shape, shape_pred_rf, average='weighted')
shape_f1_rf = f1_score(y_val_shape, shape_pred_rf, average='weighted')

type_acc_rf = accuracy_score(y_val_type, type_pred_rf)
type_precision_rf = precision_score(y_val_type, type_pred_rf, average='weighted')
type_recall_rf = recall_score(y_val_type, type_pred_rf, average='weighted')
type_f1_rf = f1_score(y_val_type, type_pred_rf, average='weighted')

# Decision Tree
shape_pred_dt = shape_dt_best.predict(X_val)
type_pred_dt = type_dt_best.predict(X_val)

shape_acc_dt = accuracy_score(y_val_shape, shape_pred_dt)
shape_precision_dt = precision_score(y_val_shape, shape_pred_dt, average='weighted')
shape_recall_dt = recall_score(y_val_shape, shape_pred_dt, average='weighted')
shape_f1_dt = f1_score(y_val_shape, shape_pred_dt, average='weighted')

type_acc_dt = accuracy_score(y_val_type, type_pred_dt)
type_precision_dt = precision_score(y_val_type, type_pred_dt, average='weighted')
type_recall_dt = recall_score(y_val_type, type_pred_dt, average='weighted')
type_f1_dt = f1_score(y_val_type, type_pred_dt, average='weighted')

# Create a list to store the evaluation metrics for each model
model_results = [
    ['Neural Network', shape_acc_nn, shape_precision_nn, shape_recall_nn, shape_f1_nn,
    ['Gradient Boosting', shape_acc_gb, shape_precision_gb, shape_recall_gb, shape_f1_g
    ['Random Forest', shape_acc_rf, shape_precision_rf, shape_recall_rf, shape_f1_rf, t
    ['Decision Tree', shape_acc_dt, shape_precision_dt, shape_recall_dt, shape_f1_dt, t
]

# Create a pandas DataFrame from the list
comparison_df = pd.DataFrame(model_results, columns=['Model', 'Shape Accuracy', 'Shape
                                                     'Type Accuracy', 'Type Precision',

# Set the display options to show all columns in a single line
pd.set_option('display.max_columns', None)
pd.set_option('display.expand_frame_repr', False)

# Display the comparison table
print(comparison_df.to_string(index=False))
```

```
Original shape of X_val: (740, 784)
24/24 [==============================] - 0s 12ms/step
24/24 [==============================] - 0s 15ms/step
           Model  Shape Accuracy  Shape Precision  Shape Recall  Shape F1-score  Type
Accuracy  Type Precision  Type Recall  Type F1-score
   Neural Network        0.997297         0.997305      0.997297        0.997298
0.997297        0.997326     0.997297       0.997292
Gradient Boosting        0.960811         0.962547      0.960811        0.960089
0.950000        0.953086     0.950000       0.949127
    Random Forest        0.954054         0.956931      0.954054        0.952815
0.959459        0.961381     0.959459       0.958921
    Decision Tree        0.863514         0.867913      0.863514        0.864994
0.836486        0.844076     0.836486       0.837995
```

In [62…

```
# Print the current shape of test_images
print("Original shape of test_images:", test_images.shape)

# Reshape the input data to match the expected input shape of the model
num_samples = test_images.shape[0]
```

```
test_images_reshaped = test_images.reshape(num_samples, -1)  # Flatten the images

# Neural Network
test_images_nn = test_images_reshaped.reshape(num_samples, 28, 28)
test_images_nn = np.repeat(test_images_nn[..., np.newaxis], 3, axis=-1)
shape_pred_nn = np.argmax(model.predict(test_images_nn)[0], axis=1)
type_pred_nn = np.argmax(model.predict(test_images_nn)[1], axis=1)

shape_acc_nn = accuracy_score(test_shape_labels, shape_pred_nn)
shape_precision_nn = precision_score(test_shape_labels, shape_pred_nn, average='weighte
shape_recall_nn = recall_score(test_shape_labels, shape_pred_nn, average='weighted')
shape_f1_nn = f1_score(test_shape_labels, shape_pred_nn, average='weighted')

type_acc_nn = accuracy_score(test_type_labels, type_pred_nn)
type_precision_nn = precision_score(test_type_labels, type_pred_nn, average='weighted')
type_recall_nn = recall_score(test_type_labels, type_pred_nn, average='weighted')
type_f1_nn = f1_score(test_type_labels, type_pred_nn, average='weighted')

# Gradient Boosting
shape_pred_gb = shape_gb_best.predict(test_images_reshaped)
type_pred_gb = type_gb_best.predict(test_images_reshaped)

shape_acc_gb = accuracy_score(test_shape_labels, shape_pred_gb)
shape_precision_gb = precision_score(test_shape_labels, shape_pred_gb, average='weighte
shape_recall_gb = recall_score(test_shape_labels, shape_pred_gb, average='weighted')
shape_f1_gb = f1_score(test_shape_labels, shape_pred_gb, average='weighted')

type_acc_gb = accuracy_score(test_type_labels, type_pred_gb)
type_precision_gb = precision_score(test_type_labels, type_pred_gb, average='weighted')
type_recall_gb = recall_score(test_type_labels, type_pred_gb, average='weighted')
type_f1_gb = f1_score(test_type_labels, type_pred_gb, average='weighted')

# Random Forest
shape_pred_rf = shape_rf_best.predict(test_images_reshaped)
type_pred_rf = type_rf_best.predict(test_images_reshaped)

shape_acc_rf = accuracy_score(test_shape_labels, shape_pred_rf)
shape_precision_rf = precision_score(test_shape_labels, shape_pred_rf, average='weighte
shape_recall_rf = recall_score(test_shape_labels, shape_pred_rf, average='weighted')
shape_f1_rf = f1_score(test_shape_labels, shape_pred_rf, average='weighted')

type_acc_rf = accuracy_score(test_type_labels, type_pred_rf)
type_precision_rf = precision_score(test_type_labels, type_pred_rf, average='weighted')
type_recall_rf = recall_score(test_type_labels, type_pred_rf, average='weighted')
type_f1_rf = f1_score(test_type_labels, type_pred_rf, average='weighted')

# Decision Tree
shape_pred_dt = shape_dt_best.predict(test_images_reshaped)
type_pred_dt = type_dt_best.predict(test_images_reshaped)

shape_acc_dt = accuracy_score(test_shape_labels, shape_pred_dt)
shape_precision_dt = precision_score(test_shape_labels, shape_pred_dt, average='weighte
shape_recall_dt = recall_score(test_shape_labels, shape_pred_dt, average='weighted')
shape_f1_dt = f1_score(test_shape_labels, shape_pred_dt, average='weighted')

type_acc_dt = accuracy_score(test_type_labels, type_pred_dt)
type_precision_dt = precision_score(test_type_labels, type_pred_dt, average='weighted')
type_recall_dt = recall_score(test_type_labels, type_pred_dt, average='weighted')
type_f1_dt = f1_score(test_type_labels, type_pred_dt, average='weighted')

# Logistic Regression
shape_pred_lr = shape_classifier.predict(test_images_reshaped)
type_pred_lr = type_classifier.predict(test_images_reshaped)

shape_acc_lr = accuracy_score(test_shape_labels, shape_pred_lr)
shape_precision_lr = precision_score(test_shape_labels, shape_pred_lr, average='weighte
shape_recall_lr = recall_score(test_shape_labels, shape_pred_lr, average='weighted')
shape_f1_lr = f1_score(test_shape_labels, shape_pred_lr, average='weighted')

type_acc_lr = accuracy_score(test_type_labels, type_pred_lr)
type_precision_lr = precision_score(test_type_labels, type_pred_lr, average='weighted')
type_recall_lr = recall_score(test_type_labels, type_pred_lr, average='weighted')
type_f1_lr = f1_score(test_type_labels, type_pred_lr, average='weighted')

# Create a list to store the evaluation metrics for each model
model_results = [
    ['Neural Network', shape_acc_nn, shape_precision_nn, shape_recall_nn, shape_f1_nn,
    ['Gradient Boosting', shape_acc_gb, shape_precision_gb, shape_recall_gb, shape_f1_g
    ['Random Forest', shape_acc_rf, shape_precision_rf, shape_recall_rf, shape_f1_rf, t
    ['Decision Tree', shape_acc_dt, shape_precision_dt, shape_recall_dt, shape_f1_dt, t
    ['Logistic Regression', shape_acc_lr, shape_precision_lr, shape_recall_lr, shape_f1
]

# Create a pandas DataFrame from the list
comparison_df = pd.DataFrame(model_results, columns=['Model', 'Shape Accuracy', 'Shape
                                                     'Type Accuracy', 'Type Precision',

# Set the display options to show all columns in a single line
pd.set_option('display.max_columns', None)
pd.set_option('display.expand_frame_repr', False)

# Display the comparison table
print(comparison_df.to_string(index=False))
```

```
Original shape of test_images: (689, 784)
22/22 [==============================] - 0s 11ms/step
22/22 [==============================] - 0s 8ms/step
              Model  Shape Accuracy  Shape Precision  Shape Recall  Shape F1-score  Typ
e Accuracy  Type Precision  Type Recall  Type F1-score
     Neural Network        0.891147         0.900349      0.891147        0.888370
0.822932        0.843178     0.822932       0.818225
  Gradient Boosting        0.698113         0.693441      0.698113        0.665543
0.500726        0.656473     0.500726       0.526417
      Random Forest        0.699565         0.701655      0.699565        0.661380
0.542816        0.658558     0.542816       0.551518
      Decision Tree        0.592163         0.580335      0.592163        0.573428
0.394775        0.446641     0.394775       0.404959
Logistic Regression        0.619739         0.629102      0.619739        0.601057
0.506531        0.684012     0.506531       0.545015
```