Traffic Light detection Using Deep Learning

A CNN-Based Approach for Varying Lighting Conditions

Mashael aljuhani
Ebtsam Asiri
Nouf abdullah

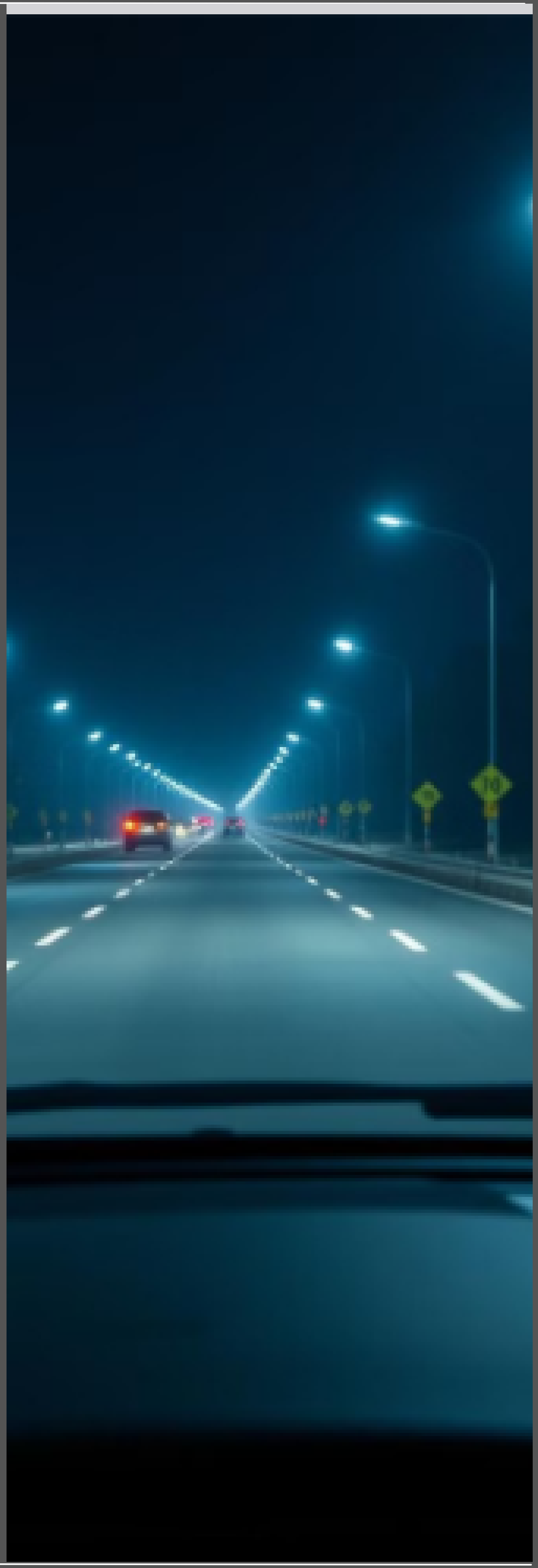
Introduction:

Problem:

Develop a deep learning model capable of detection traffic .lights under different lighting conditions (day and night)

Objective:

Build a model that accurately classifies traffic lights from images captured under varying conditions, with a focus on .practical application in autonomous driving systems



Data Collection

Source

-Traffic light images with metadata
by kaggle
The database is collected in San Diego, California, USA.

Exploration

-The dataset contained one primary class, "go," with images taken in different lighting conditions. Basic .cleaning and path adjustments were necessary

Data Preparation:

Process: Data paths were updated, and images were augmente using
ImageDataGenerator (rotation, shift, zoom, flip) .
.The data was split into training and validation sets

Model Building

Model Choice:
A Convolutional Neural Network (CNN)
was selected for
its efficiency in image classification tasks

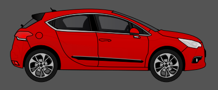
Architecture

The model includes multiple Conv2D, MaxPooling, BatchNormalization, and Dropout layers, with a final .dense layer using sigmoid for binary classification

Model Training

Training Setup: The model was trained over 30 epochs with a batch size of 32.
.EarlyStopping was implemented to prevent overfitting

Challenges



Limited Time for Data Collection: We faced a challenge with the time constraints, which limited our ability to collect and curate a more diverse and comprehensive dataset.
This constraint impacted the model's performance,
as a richer dataset would likely have led to better results.

improving Accuracy: One of the main challenges we faced was improving the model's accuracy. Despite efforts in data augmentation and model tuning, achieving higher accuracy proved difficult due to the limited diversity in the dataset and the presence of only one primary class. Further improvements could be made by gathering more diverse data and experimenting with more complex models or hyperparameters.

Model Evaluation:

The model is functional but could benefit from additional data diversity and further tuning.

Model Deployment:

Usage: The trained model can be used to predict traffic light classes in new images, supporting potential applications in driver assistance systems

Data Splitting and Data Generators:

```
# Create a training data generator
train_generator = datagen.flow_from_dataframe(
    dataframe=df_bulb,          # Use the DataFrame containing image paths and labels
    directory=image_folder_path,  # The directory where the images are stored
    x_col='Filename',            # Column in the DataFrame that contains the filenames
    y_col='Annotation tag',      # Column in the DataFrame that contains the labels
    target_size=(64, 64),        # Resize all images to 64x64 pixels
    batch_size=32,               # Number of images to process in each batch
    class_mode='binary',         # Perform binary classification ('go' or 'not go')
    subset='training'            # Use this subset of data for training
)

# Create a validation data generator
validation_generator = datagen.flow_from_dataframe(
    dataframe=df_bulb,          # Use the same DataFrame for validation
    directory=image_folder_path,  # The directory where the images are stored
    x_col='Filename',            # Column in the DataFrame that contains the filenames
    y_col='Annotation tag',      # Column in the DataFrame that contains the labels
    target_size=(64, 64),        # Resize all images to 64x64 pixels
    batch_size=32,               # Number of images to process in each batch
    class_mode='binary',         # Perform binary classification ('go' or 'not go')
    subset='validation'          # Use this subset of data for validation
)
```

Model Building

```python
# Define the CNN model
model = models.Sequential([

    # First convolutional layer with 64 filters, 3x3 kernel size, and ReLU activation
    layers.Conv2D(64, (3, 3), activation='relu', input_shape=(64, 64, 3)),
       # Apply Batch Normalization to stabilize and accelerate training
    layers.BatchNormalization(),
       # MaxPooling layer to reduce the spatial dimensions (2x2 pool size)
    layers.MaxPooling2D((2, 2)),
   # Second convolutional layer with 128 filters, 3x3 kernel size, and ReLU activation
    layers.Conv2D(128, (3, 3), activation='relu'),
      # Apply Batch Normalization to stabilize and accelerate training
    layers.BatchNormalization(),
    # MaxPooling layer to reduce the spatial dimensions (2x2 pool size)
    layers.MaxPooling2D((2, 2)),
  # Third convolutional layer with 128 filters, 3x3 kernel size, and ReLU activation
    layers.Conv2D(128, (3, 3), activation='relu'),
      # Apply Batch Normalization to stabilize and accelerate training
    layers.BatchNormalization(),
      # MaxPooling layer to reduce the spatial dimensions (2x2 pool size)
    layers.MaxPooling2D((2, 2)),
   # Flatten the output from the convolutional layers to feed into the fully connected layers
    layers.Flatten(),
      # Fully connected layer with 512 units and ReLU activation
    layers.Dense(512, activation='relu'),
     # Dropout layer to prevent overfitting by randomly dropping 50% of the units
    layers.Dropout(0.5),
     # Output layer with 1 unit and sigmoid activation for binary classification
    layers.Dense(1, activation='sigmoid')
])

# Compile the model with Adam optimizer, binary crossentropy loss, and accuracy metric
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.00001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```python
# Display the model's architecture
model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 62, 62, 64)	1,792
batch_normalization_9 (BatchNormalization)	(None, 62, 62, 64)	256
max_pooling2d_9 (MaxPooling2D)	(None, 31, 31, 64)	0
conv2d_10 (Conv2D)	(None, 29, 29, 128)	73,856
batch_normalization_10 (BatchNormalization)	(None, 29, 29, 128)	512
max_pooling2d_10 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_11 (Conv2D)	(None, 12, 12, 128)	147,584
batch_normalization_11 (BatchNormalization)	(None, 12, 12, 128)	512
max_pooling2d_11 (MaxPooling2D)	(None, 6, 6, 128)	0
flatten_3 (Flatten)	(None, 4608)	0
dense_6 (Dense)	(None, 512)	2,359,808
dropout_3 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 1)	513

Total params: 2,584,833 (9.86 MB)
Trainable params: 2,584,193 (9.86 MB)
Non-trainable params: 640 (2.50 KB)

Model Training

```
# Train the model code
history = model.fit
# Pass the training data generator to the fit function
train_generator,
# Define the number of steps per epoch, based on the size of the training data
steps_per_epoch=train_generator.samples // 32,
# Pass the validation data generator to the fit function for model evaluation
validation_data=validation_generator,
# Define the number of validation steps, based on the size of the validation data
validation_steps=validation_generator.samples // 32,
# Set the number of epochs (iterations over the entire dataset)
epochs=30,
# Include callbacks, such as early stopping, to prevent overfitting
callbacks=[early_stopping]
```

```
Epoch 1/30
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDa
  self._warn_if_super_not_called()
47/47 ━━━━━━━━━━━━━━━━━━━━ 135s 2s/step - accuracy: 0.5275 - loss: 1.1683 - val_accuracy: 0.4773 - val_loss: 0.6972
Epoch 2/30
 1/47 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.4062 - loss: 1.2147/usr/lib/python3.10/contextlib.py:153: UserWarning:
```

Model prediction

```python
    # Create an empty image (black image) with the size 64x64 pixels and 3 color channels (RGB)
img_array = np.zeros((64, 64, 3))

# Expand the dimensions of the image array to add a batch dimension (required by the model)
img_array = np.expand_dims(img_array, axis=0)

# Use the trained model to predict the probability that the image is "go"
prediction = model.predict(img_array)

# Print the predicted probability for the "go" class
print(f'Probability of "go": {prediction[0][0]}')

# Define the threshold for classification
threshold = 0.5

# Compare the predicted probability with the threshold to classify the image
if prediction[0][0] > threshold:
    print("The image is classified as 'go'")  # If the probability is greater than 0.5, classify as "go"
else:
    print("The image is classified as 'not go'")  # If the probability is 0.5 or less, classify as "not go"
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step
Probability of "go": 0.4742961823940277
The image is classified as 'not go'
```

Conclusions:

Successfully built and trained a CNN for traffic Light detection, with key learnings on handling imbalanced data and enhancing model stabilit

Future Improvements:

Gather more diverse data, experiment with more complex models, and refine augmentation techniques

Thanks