Data Collection

```
import os
import cv2
import numpy as np
import pandas as pd
import zipfile
from tensorflow.keras.utils import to_categorical

#  Define the path for the zip file and extraction directory
zip_path = '/content/GTSRB_Final_Training_Images.zip'
extract_dir = '/content/GTSRB/Final_Training/Images'


# Extract the dataset if not already extracted
if not os.path.exists(extract_dir):
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(r"/content/")

#  Path to GTSRB training data
train_root = extract_dir

```

```
import os

# Check if extraction was successful
if os.path.exists(extract_dir):
    print(" Extraction successful. Checking dataset structure...")
    print("Classes found:", os.listdir(extract_dir))  # Should list class fc
else:
    print(" Extraction failed. Check the ZIP file.")
```

```
 Extraction successful. Checking dataset structure...
Classes found: ['00018', '00026', '00038', '00003', '00041', '00024', '0003
```

```
import os

dataset_path = "/content/GTSRB/Final_Training/Images"

# Verify folder contents
print("Checking dataset structure...")
for folder in os.listdir(dataset_path):
    folder_path = os.path.join(dataset_path, folder)
    print(f" {folder} - {'Folder' if os.path.isdir(folder_path) else 'File'}
```

```
# Pick a class and check images inside
sample_class = os.listdir(dataset_path)[0]  # First class folder
sample_class_path = os.path.join(dataset_path, sample_class)

print(f"\n Checking images inside {sample_class}...")
print(os.listdir(sample_class_path)[:5])  # Show first 5 files
```

```
Checking dataset structure...
 00018 - Folder
 00026 - Folder
 00038 - Folder
 00003 - Folder
 00041 - Folder
 00024 - Folder
 00030 - Folder
 00029 - Folder
 00004 - Folder
 00000 - Folder
 00016 - Folder
 00027 - Folder
 00017 - Folder
 00008 - Folder
 00032 - Folder
 00015 - Folder
 00011 - Folder
 00006 - Folder
 00033 - Folder
 00019 - Folder
 00023 - Folder
 00031 - Folder
 00012 - Folder
 00035 - Folder
 00021 - Folder
 00010 - Folder
 00034 - Folder
 00037 - Folder
 00042 - Folder
 00039 - Folder
 00028 - Folder
 00009 - Folder
 00022 - Folder
 00001 - Folder
 00007 - Folder
 00013 - Folder
 00002 - Folder
 00005 - Folder
 00025 - Folder
 00020 - Folder
 00040 - Folder
 00036 - Folder
 00014 - Folder

 Checking images inside 00018...
['00009_00016.ppm', '00015_00001.ppm', '00029_00009.ppm', '00008_00009.ppm'
```

```

#  Load and preprocess images
def load_gtsrb_dataset(root_dir, img_size=(32, 32)):
    images, labels = [], []  # FIXED
```

```
    for class_id in sorted(os.listdir(root_dir)):
        class_path = os.path.join(root_dir, class_id)
        if os.path.isdir(class_path):
            for img_file in os.listdir(class_path):
                if img_file.endswith(".ppm"):
                    img_path = os.path.join(class_path, img_file)
                    img = cv2.imread(img_path)
                    if img is not None:
                        img = cv2.resize(img, img_size)
                        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                        images.append(img)
                        labels.append(int(class_id))
    return np.array(images), np.array(labels)

# Load data
X_raw, y_raw = load_gtsrb_dataset(train_root)
print(f"Loaded {len(X_raw)} images, shape: {X_raw.shape}")

#  Display class-wise distribution
label_counts = np.bincount(y_raw)
label_df = pd.DataFrame({
    "ClassId": np.arange(len(label_counts)),
    "Image Count": label_counts
})
print("\nClass-wise Image Distribution:")
print(label_df.to_string(index=False))

```

```
Loaded 39209 images, shape: (39209, 32, 32, 3)

Class-wise Image Distribution:
 ClassId  Image Count
       0          210
       1         2220
       2         2250
       3         1410
       4         1980
       5         1860
       6          420
       7         1440
       8         1410
       9         1470
      10         2010
      11         1320
      12         2100
      13         2160
      14          780
      15          630
      16          420
      17         1110
      18         1200
      19          210
      20          360
      21          330
      22          390
      23          510
      24          270
      25         1500
      26          600
      27          240
      28          540
      29          270
      30          450
      31          780
      32          240
      33          689
      34          420
      35         1200
      36          390
      37          210
      38         2070
      39          300
      40          360
      41          240
      42          240
```

Data Cleaning

```
import glob

#  Clean using CSV ROI data
def load_and_crop_with_roi(root_dir, img_size=(32, 32)):
    images, labels = [], []

    # Get all class CSVs
    csv_files = glob.glob(os.path.join(root_dir, "*", "GT-*.csv"))

    for csv_path in csv_files:
        df = pd.read_csv(csv_path, sep=';')
        class_dir = os.path.dirname(csv_path)

        for _, row in df.iterrows():
            img_path = os.path.join(class_dir, row['Filename'])
            img = cv2.imread(img_path)

            if img is not None:
                # Crop using ROI
                x1, y1, x2, y2 = row['Roi.X1'], row['Roi.Y1'], row['Roi.X2'
                cropped = img[y1:y2, x1:x2]
                resized = cv2.resize(cropped, img_size)
                resized = cv2.cvtColor(resized, cv2.COLOR_BGR2RGB)
                images.append(resized)
                labels.append(row['ClassId'])

    return np.array(images)/255.0, np.array(labels)

#  Load cleaned dataset
X_cleaned, y_cleaned = load_and_crop_with_roi(train_root)
print(f" Cropped and resized {len(X_cleaned)} images, shape: {X_cleaned.sha
```

```
 Cropped and resized 39209 images, shape: (39209, 32, 32, 3)
```

Data Transformation

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
import requests
import numpy as np

#  One-hot encode labels
y_encoded = to_categorical(y_cleaned)
print(f" Labels one-hot encoded: shape = {y_encoded.shape}")

#  Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X_cleaned, y_encoded, test_size=0.2, stratify=y_cleaned, random_state=4
)
print(f" Split: {X_train.shape[0]} train, {X_test.shape[0]} test")

#  Real-time Weather API Integration
API_KEY = "4690348b38a4f5520d615776d7e9e01d"
CITY = "Texas"
API_URL = "https://api.openweathermap.org/data/2.5/weather"

def get_weather():
    try:
        params = {"q": CITY, "appid": API_KEY, "units": "metric"}
        response = requests.get(API_URL, params=params)
        data = response.json()
        if response.status_code == 200:
            temp = data["main"].get("temp", 20.0)
            humidity = data["main"].get("humidity", 50)
            print(f" Temp: {temp}°C,  Humidity: {humidity}%")
            return temp, humidity
    except Exception as e:
        print(" Weather API error:", e)
    return 20.0, 50

# Fetch real-time weather (assumed constant for training batch)
temperature, humidity = get_weather()
weather_train = np.array([[temperature, humidity]] * len(X_train))
weather_test = np.array([[temperature, humidity]] * len(X_test))

print(f" Weather metadata shape (train): {weather_train.shape}")

```

```
 Labels one-hot encoded: shape = (39209, 43)
 Split: 31367 train, 7842 test
 Temp: 29.97°C,  Humidity: 35%
 Weather metadata shape (train): (31367, 2)
```

```
# Augmentation
```

```
# After Step 3 is done — you already have these
# X_train, y_train, weather_train
# (All numpy arrays)

# INSERT AUGMENTATION HERE
import random

def add_fog(image):
    fog_layer = np.full_like(image, 200, dtype=np.uint8)
    return cv2.addWeighted(image, 0.7, fog_layer, 0.3, 0)

def add_rain(image, drop_count=100):
    rainy = image.copy()
    h, w, _ = image.shape
    for _ in range(drop_count):
        x1, y1 = random.randint(0, w), random.randint(0, h)
        x2, y2 = x1 + random.randint(-2, 2), y1 + random.randint(10, 20)
        cv2.line(rainy, (x1, y1), (x2, y2), (200, 200, 200), 1)
    return cv2.addWeighted(image, 0.8, rainy, 0.2, 0)

def add_glare(image):
    glare = image.copy()
    h, w = glare.shape[:2]
    cx = random.randint(w//3, 2*w//3)
    cy = random.randint(h//3, 2*h//3)
    r = random.randint(5, 15)
    overlay = glare.copy()
    cv2.circle(overlay, (cx, cy), r, (255, 255, 255), -1)
    return cv2.addWeighted(glare, 0.8, overlay, 0.2, 0)

# Augment 25% of training data
augment_fraction = 0.25
num_aug = int(len(X_train) * augment_fraction)
indices = random.sample(range(len(X_train)), num_aug)

X_aug, y_aug = [], []

for idx in indices:
    img = (X_train[idx] * 255).astype(np.uint8)
    label = y_train[idx]
    effect = random.choice(["fog", "rain", "glare"])

    if effect == "fog":
        aug_img = add_fog(img)
    elif effect == "rain":
```

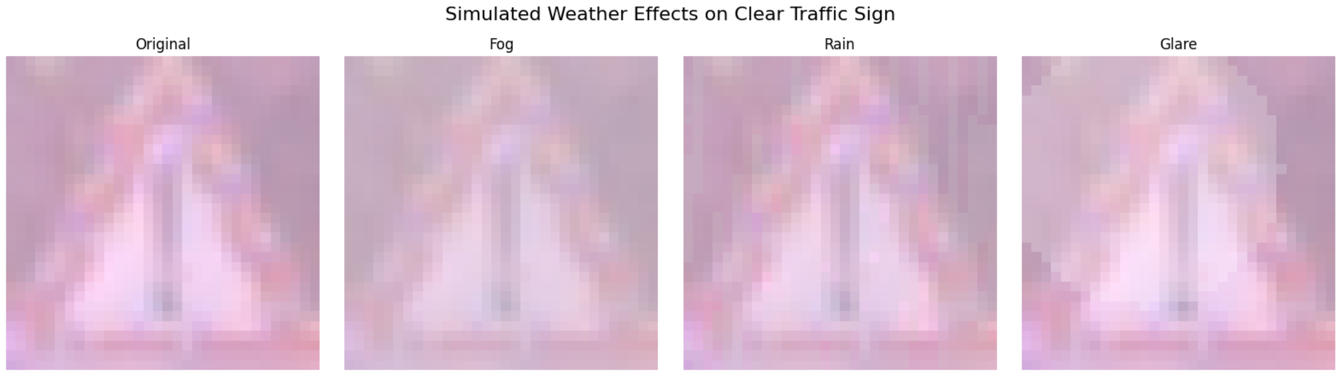
```
        aug_img = add_rain(img)
    else:
        aug_img = add_glare(img)

    X_aug.append(aug_img / 255.0)
    y_aug.append(label)

# Final combined training data
X_train_aug = np.concatenate((X_train, np.array(X_aug)), axis=0)
y_train_aug = np.concatenate((y_train, np.array(y_aug)), axis=0)
weather_train_aug = np.concatenate((weather_train, weather_train[:len(X_aug

print(f" Augmented {len(X_aug)} images. Final training shape: {X_train_aug.

```

```
Augmented 7841 images. Final training shape: (39208, 32, 32, 3)
```

```
# Display Augmented Images
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random

# --- Weather effect functions ---
def add_fog(image):
    fog_layer = np.full_like(image, 200, dtype=np.uint8)
    return cv2.addWeighted(image, 0.7, fog_layer, 0.3, 0)

def add_rain(image, drop_count=100):
    rainy = image.copy()
    h, w, _ = image.shape
    for _ in range(drop_count):
        x1, y1 = random.randint(0, w), random.randint(0, h)
        x2, y2 = x1 + random.randint(-2, 2), y1 + random.randint(10, 20)
        cv2.line(rainy, (x1, y1), (x2, y2), (200, 200, 200), 1)
    return cv2.addWeighted(image, 0.8, rainy, 0.2, 0)

def add_glare(image):
    glare = image.copy()
    h, w = glare.shape[:2]
    cx = random.randint(w // 3, 2 * w // 3)
    cy = random.randint(h // 3, 2 * h // 3)
    radius = random.randint(5, 15)
    overlay = glare.copy()
    cv2.circle(overlay, (cx, cy), radius, (255, 255, 255), -1)
    return cv2.addWeighted(glare, 0.8, overlay, 0.2, 0)
```

```

# --- Sharp image filter ---
def is_clear_image(img, threshold=100.0):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    return cv2.Laplacian(gray, cv2.CV_64F).var() > threshold

def get_one_clear_image(X):
    for i in range(len(X)):
        img = (X[i] * 255).astype(np.uint8)
        if is_clear_image(img):
            return img
    return None

# --- Select and visualize one image with 4 versions ---
base_img = get_one_clear_image(X_cleaned)

if base_img is not None:
    fog = add_fog(base_img)
    rain = add_rain(base_img)
    glare = add_glare(base_img)

    images = [base_img, fog, rain, glare]
    titles = ["Original", "Fog", "Rain", "Glare"]

    plt.figure(figsize=(16, 4))
    for i in range(4):
        plt.subplot(1, 4, i+1)
        plt.imshow(images[i])
        plt.title(titles[i])
        plt.axis("off")

    plt.tight_layout()
    plt.suptitle("Simulated Weather Effects on Clear Traffic Sign", fontsiz
    plt.show()
else:
    print(" No clear image found in the dataset.")

```



Model Training (CNN + Weather Input)

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, De

def build_weather_aware_model(img_shape=(32, 32, 3), weather_shape=(2,), num
    #  CNN Branch for images
    cnn_input = Input(shape=img_shape)
    x = Conv2D(32, (3, 3), activation='relu')(cnn_input)
    x = MaxPooling2D((2, 2))(x)
    x = Conv2D(64, (3, 3), activation='relu')(x)
    x = MaxPooling2D((2, 2))(x)
    x = Flatten()(x)
    cnn_output = Dense(128, activation='relu')(x)
    cnn_output = Dropout(0.5)(cnn_output)

    #  DNN Branch for weather metadata
    weather_input = Input(shape=weather_shape)
    weather_dense = Dense(8, activation='relu')(weather_input)

    #  Fusion
```

```
    # Fusion
    merged = Concatenate()([cnn_output, weather_dense])
    final_output = Dense(num_classes, activation='softmax')(merged)

    model = Model(inputs=[cnn_input, weather_input], outputs=final_output)
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics
    return model

#  Build the model
model = build_weather_aware_model()
model.summary()

```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 32, 32, 3)	0	-
conv2d (Conv2D)	(None, 30, 30, 32)	896	input_layer[0][0]
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496	max_pooling2d[0]…
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0	conv2d_1[0][0]
flatten (Flatten)	(None, 2304)	0	max_pooling2d_1[…
dense (Dense)	(None, 128)	295,040	flatten[0][0]
input_layer_1 (InputLayer)	(None, 2)	0	-
dropout (Dropout)	(None, 128)	0	dense[0][0]
dense_1 (Dense)	(None, 8)	24	input_layer_1[0]…
concatenate (Concatenate)	(None, 136)	0	dropout[0][0], dense_1[0][0]
dense_2 (Dense)	(None, 43)	5,891	concatenate[0][0]

Total params: 320,347 (1.22 MB)
Trainable params: 320,347 (1.22 MB)
Non-trainable params: 0 (0.00 B)

Training the Model

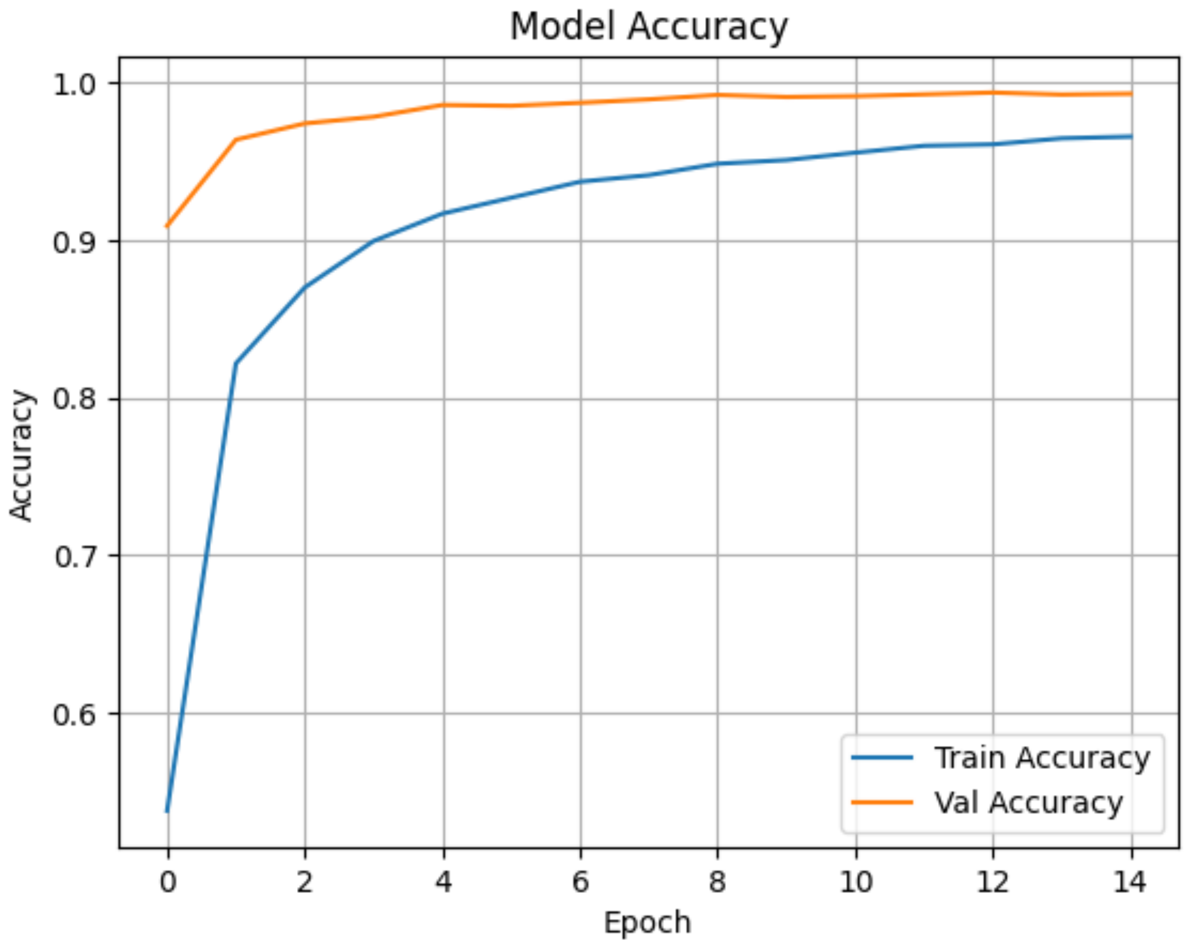
```
history = model.fit(
    [X_train_aug, weather_train_aug], y_train_aug,
    epochs=15,
    batch_size=64,
    validation_data=([X_test, weather_test], y_test)
)
```

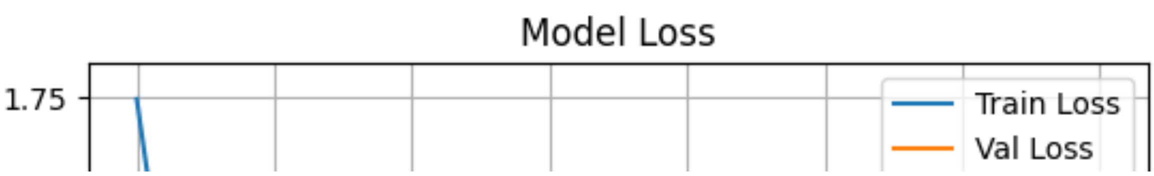
```
Epoch 1/15
613/613 ━━━━━━━━━━━━━━━━━━━━ 53s 82ms/step - accuracy: 0.3261 - loss: 2.766
Epoch 2/15
613/613 ━━━━━━━━━━━━━━━━━━━━ 83s 84ms/step - accuracy: 0.8020 - loss: 0.667
Epoch 3/15
613/613 ━━━━━━━━━━━━━━━━━━━━ 81s 82ms/step - accuracy: 0.8654 - loss: 0.439
Epoch 4/15
613/613 ━━━━━━━━━━━━━━━━━━━━ 84s 84ms/step - accuracy: 0.8983 - loss: 0.345
Epoch 5/15
613/613 ━━━━━━━━━━━━━━━━━━━━ 80s 81ms/step - accuracy: 0.9154 - loss: 0.283
Epoch 6/15
613/613 ━━━━━━━━━━━━━━━━━━━━ 84s 85ms/step - accuracy: 0.9254 - loss: 0.248
Epoch 7/15
613/613 ━━━━━━━━━━━━━━━━━━━━ 79s 80ms/step - accuracy: 0.9341 - loss: 0.213
Epoch 8/15
613/613 ━━━━━━━━━━━━━━━━━━━━ 83s 81ms/step - accuracy: 0.9435 - loss: 0.179
Epoch 9/15
613/613 ━━━━━━━━━━━━━━━━━━━━ 81s 79ms/step - accuracy: 0.9462 - loss: 0.172
Epoch 10/15
613/613 ━━━━━━━━━━━━━━━━━━━━ 50s 81ms/step - accuracy: 0.9527 - loss: 0.147
Epoch 11/15
613/613 ━━━━━━━━━━━━━━━━━━━━ 50s 81ms/step - accuracy: 0.9571 - loss: 0.138
Epoch 12/15
613/613 ━━━━━━━━━━━━━━━━━━━━ 80s 79ms/step - accuracy: 0.9581 - loss: 0.130
Epoch 13/15
613/613 ━━━━━━━━━━━━━━━━━━━━ 83s 80ms/step - accuracy: 0.9603 - loss: 0.123
Epoch 14/15
613/613 ━━━━━━━━━━━━━━━━━━━━ 82s 80ms/step - accuracy: 0.9646 - loss: 0.110
Epoch 15/15
613/613 ━━━━━━━━━━━━━━━━━━━━ 48s 79ms/step - accuracy: 0.9669 - loss: 0.102
```

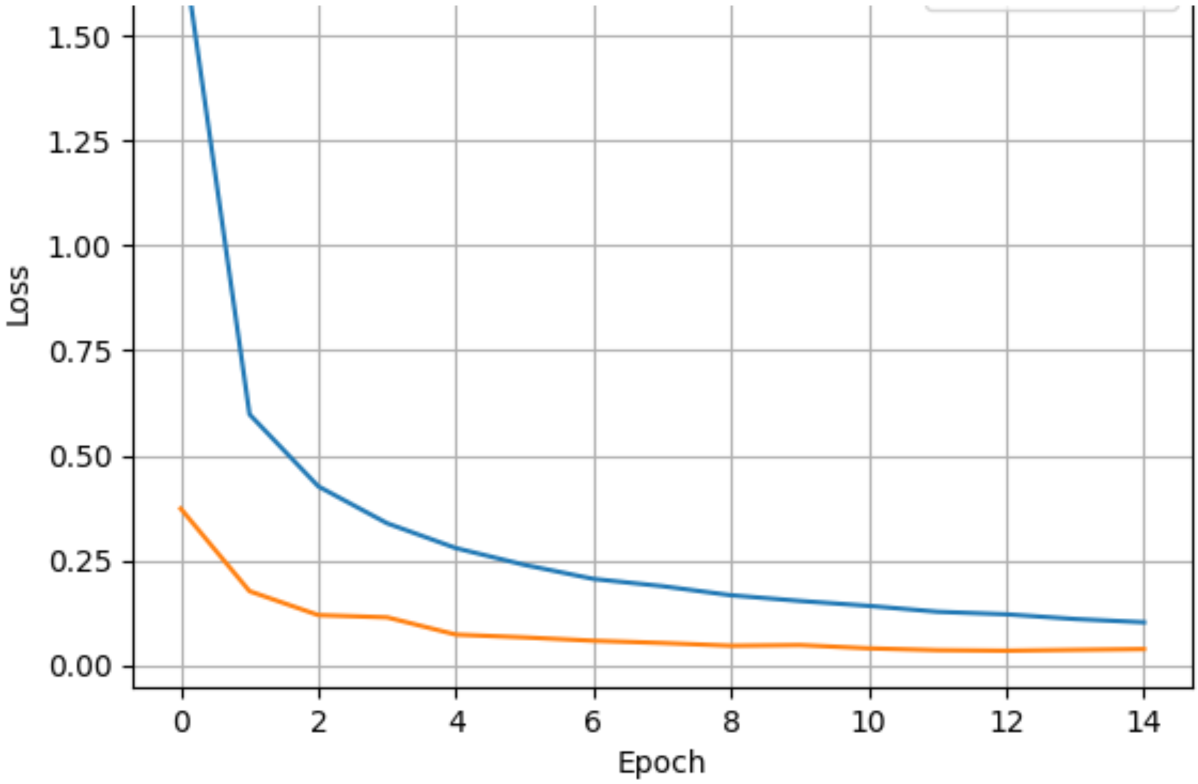
Plot Accuracy and Loss

```
import matplotlib.pyplot as plt

#  Accuracy
```

```
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()

#  Loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```







#

```
# 6.1. Evaluate on Test Set
```

```
# Final evaluation
from sklearn.metrics import accuracy_score, classification_report, confusic
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Predict on entire test set
y_pred_probs = model.predict([X_test, weather_test])
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = np.argmax(y_test, axis=1)

# Accuracy
acc = accuracy_score(y_true, y_pred)
print(f"\n🎯 Final Test Accuracy: {acc * 100:.2f}%")

# 📄 Classification Report
print("\n Classification Report:")
print(classification_report(y_true, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_true, y_pred)
```

```python

plt.figure(figsize=(12, 10))
sns.heatmap(cm, annot=False, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.tight_layout()
plt.show()
```

```
246/246 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step

🎯 Final Test Accuracy: 99.30%

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        42
           1       0.99      0.99      0.99       444
           2       0.99      0.99      0.99       450
           3       0.98      0.99      0.98       282
           4       1.00      0.99      1.00       396
           5       0.99      0.98      0.99       372
           6       1.00      1.00      1.00        84
           7       0.98      1.00      0.99       288
           8       0.99      0.99      0.99       282
           9       0.99      1.00      0.99       294
          10       0.99      1.00      0.99       402
          11       1.00      0.98      0.99       264
          12       1.00      1.00      1.00       420
          13       1.00      1.00      1.00       432
          14       0.99      1.00      1.00       156
          15       0.98      1.00      0.99       126
          16       1.00      1.00      1.00        84
          17       0.99      1.00      0.99       222
          18       0.99      1.00      1.00       240
          19       0.98      1.00      0.99        42
          20       1.00      0.97      0.99        72
          21       1.00      0.98      0.99        66
          22       0.99      1.00      0.99        78
          23       1.00      1.00      1.00       102
          24       1.00      1.00      1.00        54
          25       0.99      0.99      0.99       300
          26       0.98      0.98      0.98       120
          27       1.00      1.00      1.00        48
          28       1.00      0.98      0.99       108
          29       1.00      0.98      0.99        54
          30       0.98      0.99      0.98        90
          31       0.99      0.99      0.99       156
          32       1.00      1.00      1.00        48
          33       1.00      1.00      1.00       138
          34       1.00      1.00      1.00        84
          35       1.00      1.00      1.00       240
          36       1.00      1.00      1.00        78
```

```
          37       1.00      0.95      0.98        42
          38       1.00      1.00      1.00       414
          39       0.98      0.98      0.98        60
          40       0.96      1.00      0.98        72
          41       1.00      1.00      1.00        48
          42       1.00      1.00      1.00        48

    accuracy                           0.99      7842
   macro avg       0.99      0.99      0.99      7842
weighted avg       0.99      0.99      0.99      7842
```


```
# 6.2. Save Model
```

```
model.save("weather_augmented_traffic_sign_model.keras")
print(" Final model saved as weather_augmented_traffic_sign_model.keras")
```

```
 Final model saved as weather_augmented_traffic_sign_model.keras
```

```
# 6.3.  Make a Few Predictions
```

```
# Predict on first 10 test samples
y_pred_probs = model.predict([X_test[:10], weather_test[:10]])
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = np.argmax(y_test[:10], axis=1)

# Compare predictions vs true
for i in range(10):
    print(f"Image {i+1}: Predicted = {y_pred[i]}, Actual = {y_true[i]}")

```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
Image 1: Predicted = 28, Actual = 28
Image 2: Predicted = 2, Actual = 2
Image 3: Predicted = 1, Actual = 1
Image 4: Predicted = 25, Actual = 25
Image 5: Predicted = 38, Actual = 38
Image 6: Predicted = 9, Actual = 9
Image 7: Predicted = 31, Actual = 31
Image 8: Predicted = 12, Actual = 12
Image 9: Predicted = 1, Actual = 1
Image 10: Predicted = 4, Actual = 4
```

```
# 6.4.  Visual Display
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 4))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X_test[i])
    plt.title(f"Pred: {y_pred[i]}, True: {y_true[i]}")
    plt.axis("off")
plt.tight_layout()
plt.show()

```

```
import tensorflow as tf
import time
import numpy as np

batch_size = 1
X_batch = X_test[:batch_size]
weather_batch = weather_test[:batch_size]

# Define the prediction function with tf.function
@tf.function
def predict_fn(inputs):
    return model(inputs, training=False)

# Warm-up to avoid initialization overhead
for _ in range(5):
    _ = predict_fn([X_batch, weather_batch])

# Measure over multiple iterations for accuracy
iterations = 100
start_time = time.perf_counter()  # Higher precision timing
for _ in range(iterations):
    _ = predict_fn([X_batch, weather_batch])
total_time = time.perf_counter() - start_time

# Calculate FPS
if total_time > 0:  # Avoid division by zero
    fps = (batch_size * iterations) / total_time
    print(f"Optimized FPS for {batch_size} frames (over {iterations} iterat
else:
    print("Execution too fast to measure accurately with current setup.")
    fps_per_run = float('inf')  # Indicates extremely high FPS
```

```
Optimized FPS for 1 frames (over 100 iterations): 630.92
```

```
import tensorflow as tf
import time
import numpy as np

batch_size = 4
X_batch = X_test[:batch_size]
weather_batch = weather_test[:batch_size]

# Define the prediction function with tf.function
@tf.function
def predict_fn(inputs):
    return model(inputs, training=False)

# Warm-up to avoid initialization overhead
for _ in range(5):
    _ = predict_fn([X_batch, weather_batch])

# Measure over multiple iterations for accuracy
iterations = 100
start_time = time.perf_counter()  # Higher precision timing
for _ in range(iterations):
    _ = predict_fn([X_batch, weather_batch])
total_time = time.perf_counter() - start_time

# Calculate FPS
if total_time > 0:  # Avoid division by zero
    fps = (batch_size * iterations) / total_time
    print(f"Optimized FPS for {batch_size} frames (over {iterations} iterat
else:
    print("Execution too fast to measure accurately with current setup.")
    fps_per_run = float('inf')  # Indicates extremely high FPS
```

```
Optimized FPS for 4 frames (over 100 iterations): 1560.52
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import requests
from tensorflow.keras.models import load_model
from IPython.display import display
import zipfile
import os
import tempfile

# Load the trained model
try:
    model = load_model("weather_augmented_traffic_sign_model.keras")
```

```
    print("Model loaded successfully.")
except Exception as e:
    print(f"Error loading model: {e}")
    raise

# Real-time Weather API Integration
API_KEY = "4690348b38a4f5520d615776d7e9e01d"
CITY = "Texas"
API_URL = "https://api.openweathermap.org/data/2.5/weather"

def get_weather():
    try:
        params = {"q": CITY, "appid": API_KEY, "units": "metric"}
        response = requests.get(API_URL, params=params)
        response.raise_for_status()
        data = response.json()
        temp = data["main"].get("temp", 20.0)
        humidity = data["main"].get("humidity", 50)
        print(f"Weather in {CITY}: Temp: {temp}°C, Humidity: {humidity}%")
        return temp, humidity
    except Exception as e:
        print(f"Weather API error: {e}")
        return 20.0, 50

# Preprocess an image
def preprocess_image(img_path, img_size=(32, 32)):
    img = cv2.imread(img_path)
    if img is None:
        raise ValueError(f"Failed to load image: {img_path}")
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, img_size)
    img = img / 255.0
    return img

# Predict traffic sign class
def predict_traffic_sign(img_path):
    try:
        img = preprocess_image(img_path)
        img = np.expand_dims(img, axis=0)
        temperature, humidity = get_weather()
        weather_data = np.array([[temperature, humidity]])
        prediction = model.predict([img, weather_data], verbose=0)
        predicted_class = np.argmax(prediction, axis=1)[0]
        confidence = np.max(prediction) * 100
        return predicted_class, confidence, img[0]
    except Exception as e:
        raise ValueError(f"Prediction error for {img_path}: {e}")

```

```
# Visualize the result
def display_result(img, predicted_class, confidence, img_name):
    plt.figure(figsize=(6, 6))
    plt.imshow(img)
    plt.title(f"Image: {img_name}\nPredicted Class: {predicted_class}\nCor
    plt.axis("off")
    plt.show()

# Process images from zip file
def process_zip_file(zip_path):
    if not os.path.exists(zip_path):
        print(f"Zip file not found at: {zip_path}")
        return
    with tempfile.TemporaryDirectory() as temp_dir:
        try:
            with zipfile.ZipFile(zip_path, 'r') as zip_ref:
                zip_ref.testzip()  # Check for corruption
                zip_ref.extractall(temp_dir)
            print(f"Extracted zip file to temporary directory: {temp_dir}'

            supported_extensions = ('.png', '.jpg', '.jpeg', '.ppm')
            image_found = False
            for root, _, files in os.walk(temp_dir):
                for file in files:
                    if file.lower().endswith(supported_extensions):
                        image_found = True
                        img_path = os.path.join(root, file)
                        try:
                            print(f"\nProcessing image: {file}")
                            predicted_class, confidence, processed_img = p
                            print(f"Prediction: Class {predicted_class}, C
                            display_result(processed_img, predicted_class,
                        except Exception as e:
                            print(f"Error processing {file}: {e}")
            if not image_found:
                print("No supported images (.png, .jpg, .jpeg, .ppm) founc
        except zipfile.BadZipFile:
            print(f"Error: The file {zip_path} is not a valid zip file or
        except Exception as e:
            print(f"Error extracting or processing zip file: {e}")

# Main function to handle input and prediction
def main():
    print("Please enter the name of the zip file in the notebook's directc
    zip_name = input().strip()
    if not zip_name:
        print("No file name provided. Exiting.")
        return
```

```
    zip_path = os.path.join(os.getcwd(), zip_name)
    process_zip_file(zip_path)

# Run the prediction
main()
```

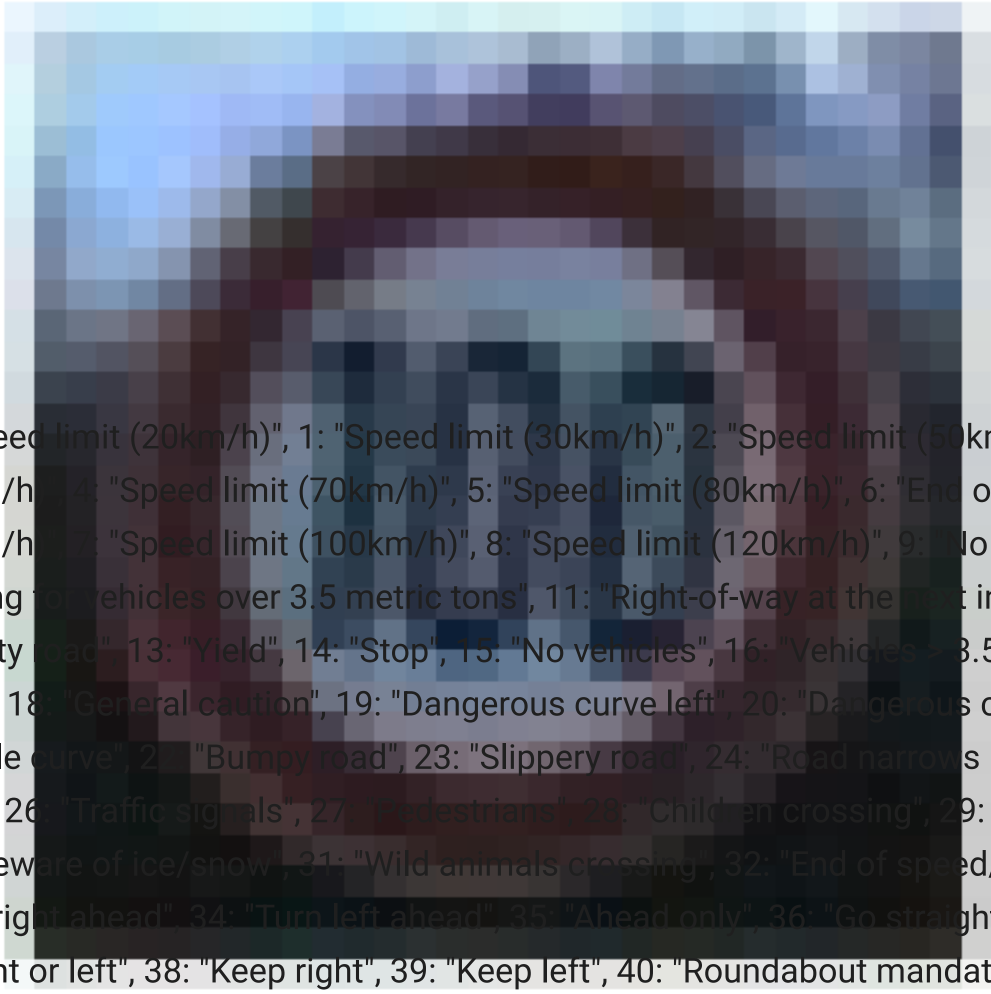
```
Model loaded successfully.
Please enter the name of the zip file in the notebook's directory (e.g., tr
traffic images.zip
Extracted zip file to temporary directory: /tmp/tmp7v31q07s

Processing image: IMG-20250405-WA0048.png
Weather in Texas: Temp: 30.97°C, Humidity: 35%
Prediction: Class 30, Confidence: 95.86%
```

Image: IMG-20250405-WA0048.png
Predicted Class: 30
Confidence: 95.86%

```
Processing image: IMG-20250405-WA0047.png
Weather in Texas: Temp: 30.97°C, Humidity: 35%
Prediction: Class 7, Confidence: 100.00%
```

Image: IMG-20250405-WA0047.png
Predicted Class: 7

Confidence: 100.00%



0: "Speed limit (20km/h)", 1: "Speed limit (30km/h)", 2: "Speed limit (50km/h)", 3: "Speed limit (60km/h)", 4: "Speed limit (70km/h)", 5: "Speed limit (80km/h)", 6: "End of speed limit (80km/h)", 7: "Speed limit (100km/h)", 8: "Speed limit (120km/h)", 9: "No passing", 10: "No passing for vehicles over 3.5 metric tons", 11: "Right-of-way at the next intersection", 12: "Priority road", 13: "Yield", 14: "Stop", 15: "No vehicles", 16: "Vehicles > 3.5t prohibited", 17: "No entry", 18: "General caution", 19: "Dangerous curve left", 20: "Dangerous curve right", 21: "Double curve", 22: "Bumpy road", 23: "Slippery road", 24: "Road narrows right", 25: "Road work", 26: "Traffic signals", 27: "Pedestrians", 28: "Children crossing", 29: "Bicycles crossing", 30: "Beware of ice/snow", 31: "Wild animals crossing", 32: "End of speed/pass limits", 33: "Turn right ahead", 34: "Turn left ahead", 35: "Ahead only", 36: "Go straight or right", 37: "Go straight or left", 38: "Keep right", 39: "Keep left", 40: "Roundabout mandatory", 41: "End of no passing", 42: "End of no passing by > 3.5t"

```
Processing image: IMG-20250405-WA0058.png
Weather in Texas: Temp: 30.97°C, Humidity: 35%
Prediction: Class 21, Confidence: 98.90%
```

Image: IMG-20250405-WA0058.png
Predicted Class: 21
Confidence: 98.90%