Advanced Data Mining project – German traffic sign recognition

Description of the solved problem

The dataset utilized in this study was sourced from Kaggle, specifically from the following link: https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign/data . It comprises over 39,000 images representing 43 different German traffic signs. The data has already been divided into training and testing sets.

The objective of this study was to tackle a multiclass image classification problem involving traffic signs. The primary aim was to develop a model—specifically, a neural network—that could identify traffic signs based on input images. To enhance the model's performance, initial steps involved data preprocessing and augmentation. Initially, the data was loaded from directories into memory and then divided into training, testing, and validation sets. Subsequently, resizing and scaling were applied, transforming the raw data into a TensorFlow Dataset.

The subsequent phase involved training the model. For this purpose, a Convolutional Neural Network (CNN) was employed. CNNs are specialized artificial neural networks tailored for grid-like data such as images and videos. They utilize convolutional layers to automatically learn spatial hierarchies of features from input data, making them highly effective in tasks like image recognition and computer vision. By employing convolutional filters, CNNs scan input data, capturing patterns and features at various spatial scales, enabling the recognition of intricate patterns and objects within images.

To address the problem, two models were trained: one was constructed from scratch with hyperparameter tuning, while the other utilized the pre-defined MobileNetV2 architecture. Both models yielded satisfactory results, but the custom-built model outperformed the other due to its size. The model achieved an accuracy of 0.989 on the test data.

Reference to the source of the data

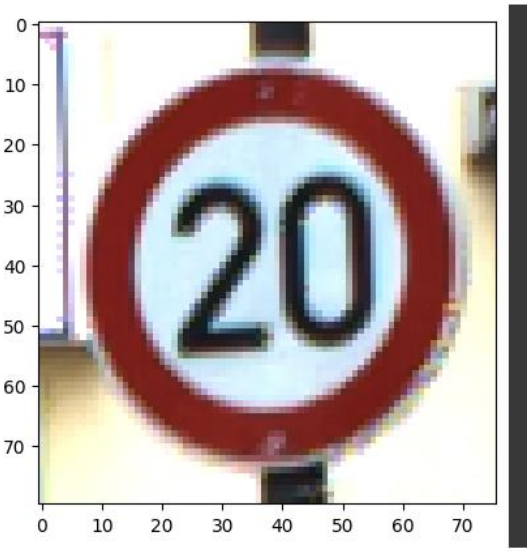
The GTSRB dataset contains images of traffic signs commonly encountered on the roads in Germany. These signs include speed limits, directional indicators, warnings, and other regulatory signs. The dataset is structured for training and testing machine learning algorithms, especially for image classification tasks. Each image in the dataset is associated with a specific class label, indicating the type of traffic sign it represents.

Dataset Contents:

Images: The dataset includes thousands of images of traffic signs. Each image is typically in color and varies in resolution.

Labels: Each image is labeled with a specific class corresponding to the type of traffic sign it depicts. These labels are essential for training machine learning models to recognize and classify the signs accurately.

The dataset poses various challenges, including different lighting conditions, weather effects, and partial occlusions, making it a valuable resource for testing the robustness of machine learning algorithms in real-world scenarios.



Description of the method used to solve the problem

Adam

It is an optimization algorithm used for training machine learning models, particularly deep learning neural networks. It combines the advantages of two other popular optimization techniques: RMSprop (Root Mean Square Propagation) and Momentum. Adam maintains two moving averages for each parameter: the first moment (mean) and the second moment (uncentered variance). These moving averages are used to adaptively adjust the learning rates of each parameter during training. Adam optimizes the learning process by adjusting the learning rates dynamically, allowing it to converge faster and often achieve better results compared to traditional gradient descent algorithms.

Sparse Categorical Crossentropy

It is a loss function commonly used in multiclass classification problems, where each input sample can belong to only one class. In the context of neural networks, especially deep learning models, this loss function measures the dissimilarity between the true class distribution and the predicted class probabilities.

Bayesian Optimization

It is a probabilistic model-based optimization technique used for hyperparameter tuning. The Bayesian Optimization Tuner uses probabilistic models to predict which hyperparameter configurations are likely to result in improved model performance. It builds a surrogate probabilistic model of the objective function (in this case, the model's accuracy or loss) and uses this model to decide which hyperparameters to explore next.

The tuner evaluates different hyperparameter configurations, updating its probabilistic model as it goes along to guide the search toward the most promising regions of the hyperparameter space. By iteratively selecting and evaluating hyperparameters based on the surrogate model's predictions, Bayesian optimization can efficiently find near-optimal configurations in a relatively small number of iterations, making it a powerful tool for hyperparameter tuning, especially when the objective function (such as accuracy) is expensive to evaluate.

Details on how the method was used to solve the problem

Firstly, the dataset was read into memory and split into three sets: train , test and validation and then resizing to median height and width was performed to improve results.

```
] data = []
  labels = []

  directory = './train'

  for i in range(43):
      img_path = os.path.join(directory, str(i))
      for img in os.listdir(img_path):
          im = Image.open(directory + '/' + str(i) + '/' + img)
          im = np.array(im)
          data.append(im)
          labels.append(i)
  data = np.array(data)
  labels = np.array(labels)
  len(data)
  len(labels)
```

```
[ ] X_data, X_test, y_data, y_test = train_test_split(data, labels, test_size=0.1, random_state=7)
    X_train, X_val, y_train, y_val = train_test_split(X_data, y_data, test_size=0.2, random_state=7)
    len(X_train), len(y_train), len(X_val), len(y_val), len(X_test), len(y_test)

    (28230, 28230, 7058, 7058, 3921, 3921)
```

```
[ ] def prepare_dataset(X, y, batch_size=32, resize=None, test=None):
        if resize is not None:
            X = [ resize(img).numpy() for img in X ]

        ds = tf.data.Dataset.from_tensor_slices((tf.constant(X), tf.constant(y)))
        if test is None:
            ds = ds.shuffle(len(X))

        ds = ds.batch(batch_size)

        return ds
```

```
] ds_train = prepare_dataset(X_train, y_train, resize=resizing)
  ds_train.element_spec

  (TensorSpec(shape=(None, 32, 32, 3), dtype=tf.float32, name=None),
   TensorSpec(shape=(None,), dtype=tf.int64, name=None))

] ds_test = prepare_dataset(X_test, y_test, resize=resizing, test=True)
  ds_test.element_spec

  (TensorSpec(shape=(None, 32, 32, 3), dtype=tf.float32, name=None),
   TensorSpec(shape=(None,), dtype=tf.int64, name=None))

] ds_val = prepare_dataset(X_val, y_val, resize=resizing)
  ds_val.element_spec
```

Model 1

```
[ ] def build_model(hp):
        cnn_model = models.Sequential([
            layers.Conv2D(16, (3, 3), activation='relu', input_shape=(median_height, median_width, 3)),
            layers.MaxPooling2D(2, 2),
            layers.Dropout(0.1),

            layers.Conv2D(32, (3, 3), activation='relu', input_shape=(median_height, median_width, 3)),
            layers.MaxPooling2D(2, 2),
            layers.Dropout(0.1),


            layers.Flatten(),

            layers.Dense(512, activation='relu'),
            layers.Dense(43, activation='softmax')
        ])

        cnn_model.compile(
            loss='sparse_categorical_crossentropy',
            optimizer='adam',
            metrics=['accuracy']
        )

        return cnn_model
```

```
[ ] from keras_tuner.tuners import BayesianOptimization

    tuner = BayesianOptimization(
        build_model,
        objective='val_accuracy',
        max_trials=10,
        num_initial_points=5,
        directory='bayesian_optimization',
        project_name='traffic_sign_classification'
    )
```

The first model was defined from scratch. Model contains multiple layers:

- Convolutional Layers: Two convolutional layers with 16 and 32 filters, respectively, using a kernel size of (3, 3) and relu activation. These layers learn spatial features from the input images.
- MaxPooling Layers: After each convolutional layer, max-pooling layers with a pool size of (2, 2) are applied to reduce spatial dimensions, retaining essential information.

- Dropout Layers: Dropout layers with a dropout rate of 0.1 are included to prevent overfitting by randomly deactivating a fraction of neurons during training.
- Flatten Layer: Flattens the 2D output to 1D before passing it to the dense layers.
- Dense Layers: Two dense layers with 512 units and relu activation, followed by an output layer with 43 units and softmax activation to match the number of classes.

The sparse_categorical_crossentrophy was used as a loss function. In this model also use hyperparameter tunning with Bayesian Omptimization Tuner was perfomed to find the best configuration.

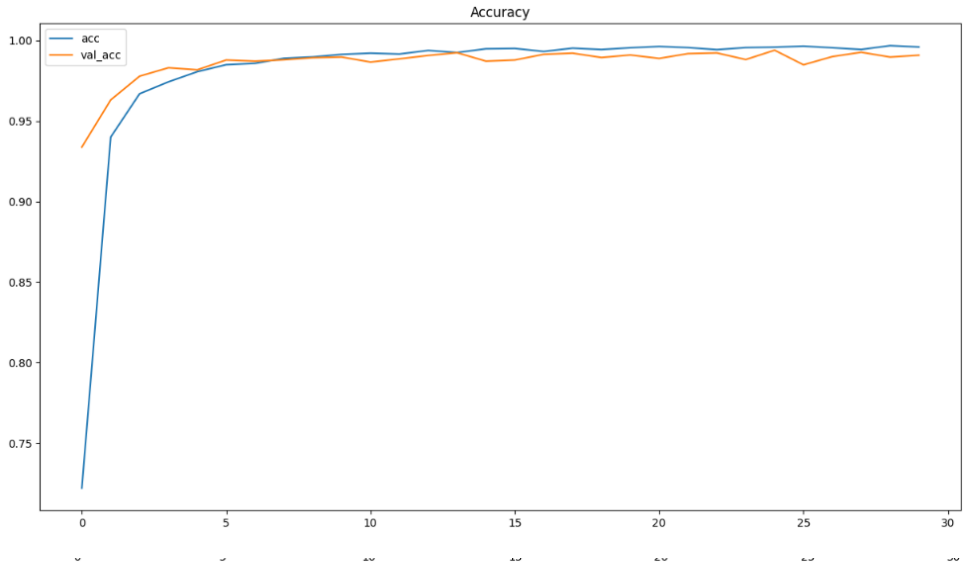
Model 2

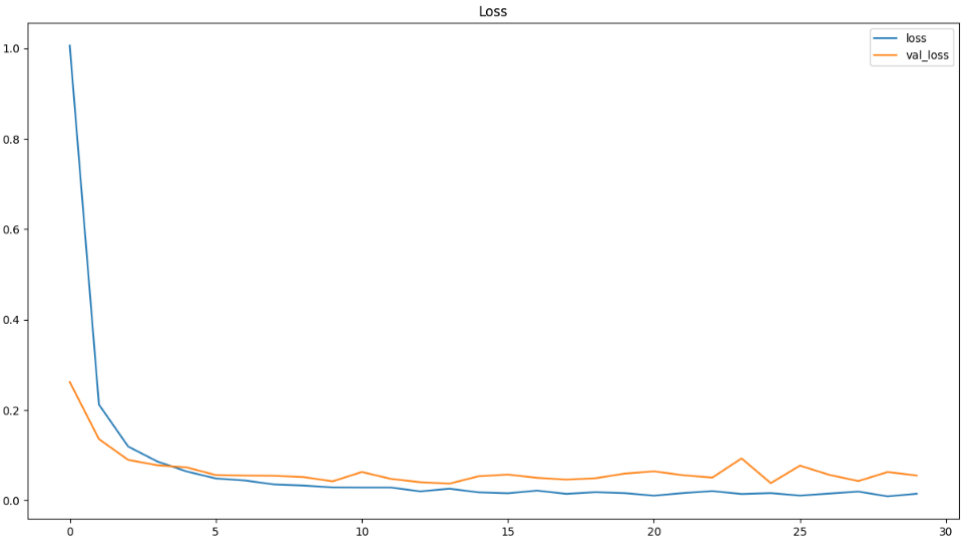
```
[ ]
    def define_mobilenetv2_model():
        # Load MobileNetV2 model
        base_model = MobileNetV2(include_top=False, input_shape=(median_height, median_width, 3))

        # Add custom classifier layers
        x = base_model.output
        x = layers.GlobalAveragePooling2D()(x)
        x = layers.Dense(128, activation='relu')(x)
        predictions = layers.Dense(43, activation='softmax')(x)

        # Create the full model
        full_model = Model(inputs=base_model.input, outputs=predictions)

        # Freeze the base model layers
        # Compile the model

        full_model.summary()
        full_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

        return full_model

    mobilenet_model = define_mobilenetv2_model()
```

This model was built with predefined CNN network MobileNetV2 which provides a highly efficient and lightweight neural network, utilizing depth wise separable convolutions and linear bottlenecks to achieve state-of-the-art accuracy in various computer vision tasks while minimizing computational resources, making it ideal for real-time applications on devices with limited processing power.

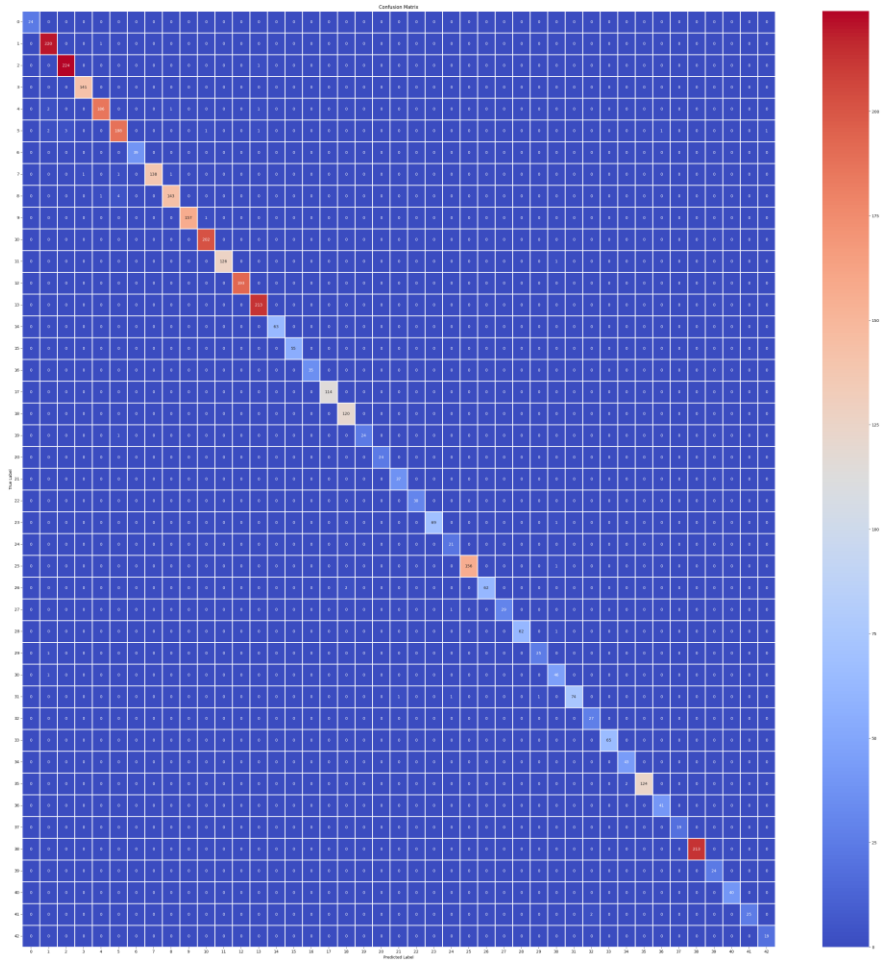
Results

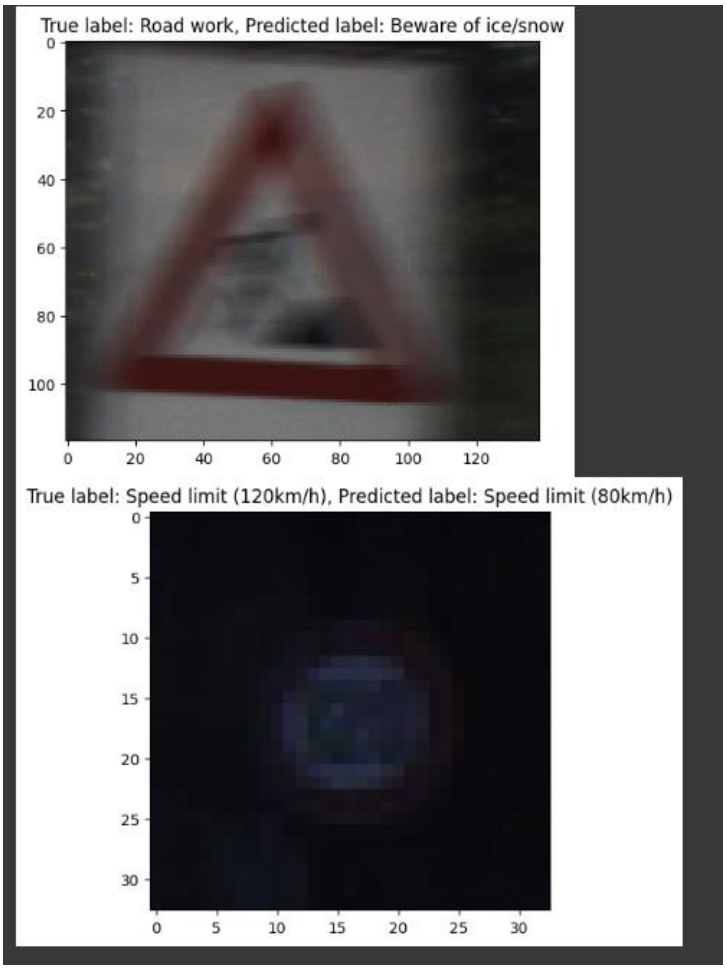
Model 1



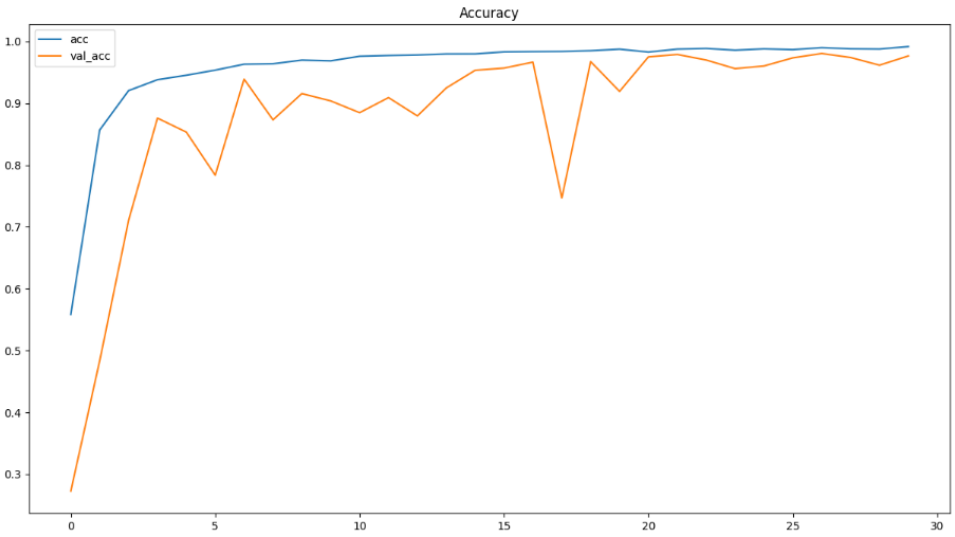


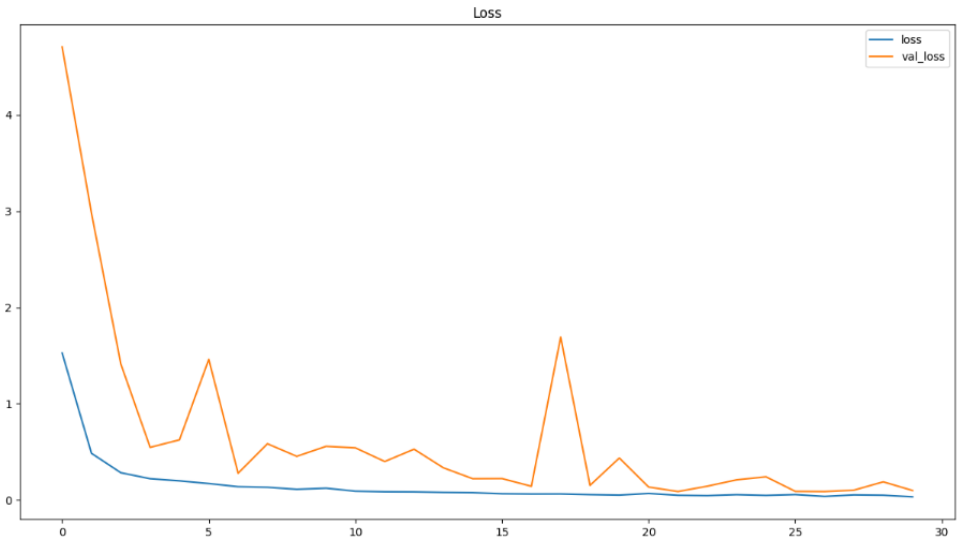
Acc: 0.9895434975624084


Confusion Matrix
True Label
Predicted Label

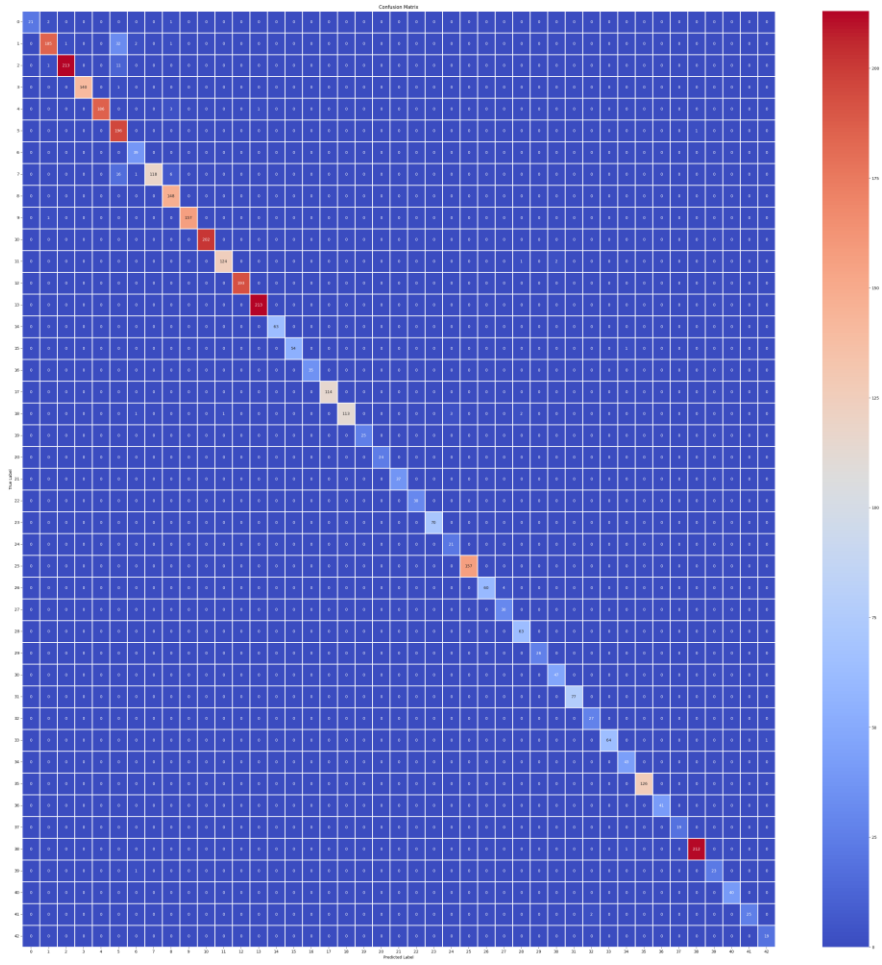

True label: Road work, Predicted label: Beware of ice/snow
True label: Speed limit (120km/h), Predicted label: Speed limit (80km/h)

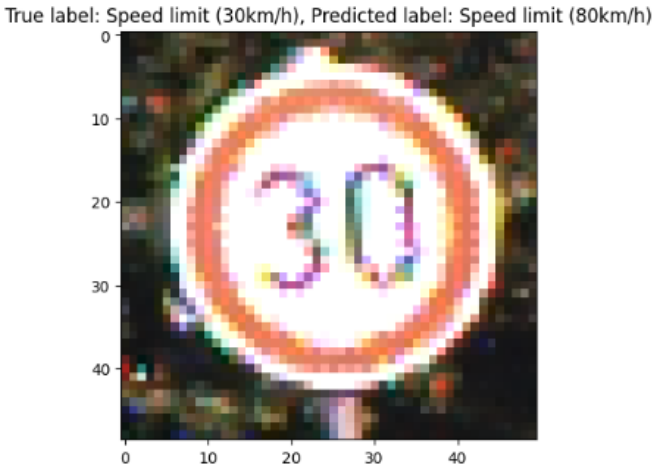
Model 2

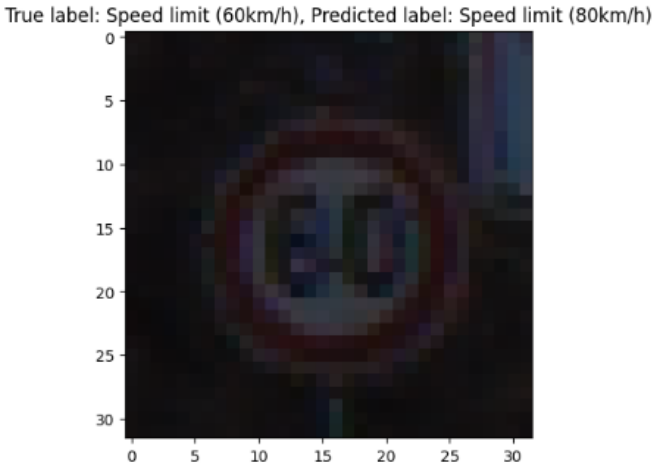

Accuracy
acc
val_acc



Accuracy: 0.9742412567138672


Confusion Matrix
True Label
Predicted Label





Discussion

In this project, the primary challenge was the multiclass image classification of German traffic signs. To tackle this, two distinct models were employed: one crafted from the ground up with careful hyperparameter tuning and the other utilizing the pre-existing architecture of MobileNetV2.

Both models exhibited satisfactory performance, but the custom-built model stood out as the superior choice. It achieved an impressive accuracy level of 0.989, surpassing the results obtained from MobileNetV2. Additionally, the custom model demonstrated a notable advantage in terms of its lightweight structure, making it a practical choice for real-world applications where computational resources are limited.

The choice of the sparse categorical crossentropy loss function proved to be appropriate for this multiclass classification task. This loss function is widely utilized in similar scenarios, contributing to the models' accuracy and effectiveness.

To comprehensively assess the models' performance, various evaluation techniques were employed. Learning curves were visualized to understand the models' training progress, providing insights into their convergence and potential areas for improvement. The confusion matrix provided a detailed breakdown of the models' predictions, highlighting specific areas where the models excelled or struggled. Additionally, the analysis of misclassified objects shed light on the challenges faced by the models, allowing for targeted improvements in future iterations.

In conclusion, this project successfully addressed the complex task of German traffic sign classification through meticulous model development, rigorous evaluation, and insightful analysis. The superior performance of the custom-built model underscores the importance of tailored solutions in specific machine learning tasks, emphasizing the significance of hyperparameter tuning and model customization. The most challenging part was crafting a neural network architecture from scratch and tuning its hyperparameters to achieve optimal performance was a complex task.