A method for efficient training of road sign recognition models for resource-dependent ADAS systems

Maksym Hovorukha[1,*], Anatoliy Doroshenko[1,2]

[1] *National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Peremohy Ave 37, 03056 Kyiv, Ukraine*
[2] *Institute of Software Systems of the National Academy of Sciences of Ukraine, Akademika Glushkova Ave 40, 03187 Kyiv, Ukraine*

Abstract

The article is devoted to the development and training of a deep learning model for automatic road sign recognition based on computer vision technologies. It thoroughly examines the process of forming and preprocessing the training dataset, including scaling, normalization, and the use of data augmentation methods to improve model accuracy and generalization. Special attention is given to comparing different approaches to neural network design — including recurrent networks, transformers, and convolutional neural networks (CNNs) — in order to determine the most effective architecture for real-time image classification of traffic signs. As a result of the architectural analysis, the MobileNetV2 model was selected — a lightweight, fast, and accurate neural network specifically adapted for use on devices with limited computational resources. Within the scope of the study, the network was optimized through regularization techniques, the addition of dropout layers, quantization, and the use of data variation methods to enhance training quality. The model was implemented in Python using the TensorFlow and Keras libraries, which provide ease of development, scalability, and hardware acceleration support. Training was performed on the Kaggle platform with GPU usage, enabling high efficiency without compromising performance. The proposed approach lays the foundation for deploying efficient, low-cost, and accessible road sign recognition systems that can be integrated into driver assistance systems and mobile applications, contributing to improved road safety.

The article also discusses the results of experimental validation, where the model demonstrated impressive accuracy and robustness in recognizing road signs under various conditions. These results confirm that the proposed approach can be effectively deployed in real-world scenarios, further enhancing its potential for integration into driver assistance systems and mobile applications. This system has the potential to significantly improve road safety by providing drivers with real-time, accurate information about traffic signs, thereby reducing the risk of accidents and improving overall traffic flow.

Keywords

deep learning, road sign recognition, computer vision, training dataset, data augmentation, convolutional neural network (cnn), model optimization, real-time processing, mobile application, video-based detection, driver assistance, road safety.

1. Introduction

The increase in the number of vehicles on the road increases the risk of accidents, many of which are caused by human error, such as inattention, fatigue or misinterpretation of road signs. Therefore, the development of effective automatic sign recognition systems is an important area for improving road safety.

Despite the availability of modern ADAS solutions, their high cost and the need for additional equipment limit their widespread use. Low-cost alternatives often have inferior accuracy or significant delays in operation, making it impossible to use them effectively in real time. The

Workshop "Intelligent information technologies" UkrProg-IIT`2025 co-located with 15th International Scientific and Practical Programming Conference UkrPROG'2025, May 13-14, 2025, Kyiv, Ukraine

[*] Corresponding author.

maksgovruha@gmail.com (M. Hovorukha); a-y-doroshenko@ukr.net (A. Doroshenko)
: 0009-0009-5833-3039 (M. Hovorukha); 0000-0002-8435-1451 (A. Doroshenko)

© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-4049/paper18.pdf

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

variability of road signs, which can vary in size, lighting, viewing angles and weather conditions, creates additional complexity.

This article discusses an approach to developing an affordable and efficient model that can be implemented in road sign recognition systems that do not require specialised hardware, working on the basis of a video stream from a smartphone camera or dashcam. The main challenge is to ensure high accuracy and processing speed, which requires model optimisation and the use of data augmentation methods for training.

2. Choose the type of neural network for road sign recognition

Traffic sign recognition is a fundamental and challenging task in the field of computer vision, as it requires not only high classification accuracy but also real-time processing and computational efficiency—especially in the context of autonomous driving and advanced driver-assistance systems (ADAS). One of the most critical factors influencing the success of such systems is the choice of neural network architecture, as it directly affects both performance and resource consumption. Therefore, selecting the most appropriate model for this specific application is essential.

A variety of neural network architectures can be utilized for analyzing visual data, including recurrent neural networks (RNNs), Vision Transformers (ViTs), and convolutional neural networks (CNNs). Each of these paradigms has unique strengths and trade-offs. RNNs are well-suited for sequential data, ViTs have shown great potential in capturing long-range dependencies in images, and CNNs excel at extracting local spatial features through hierarchical layers. However, in the context of traffic sign recognition, convolutional neural networks remain the most effective and practical choice due to their balance of accuracy, speed, and relatively low computational demands [1].

2.1. Recurrent neural networks (RNNs) and their limitations

Recurrent neural networks (RNNs) are designed to work with sequential data. The ability to take into account the temporal context is their main characteristic. This is achieved through the use of hidden states that ‘remember’ the results of previous processing stages. The main idea of the model is described by the formula [2]:

$$\mathrm{h_t = f(W_x x_t + W_h h_{t-1} + b)}, \tag{1}$$

where h_t – is the hidden state at step t, which stores information about the current and previous elements of the sequence;

x_t – is the input vector at step t;

W_x, W_h – weight matrices responsible for the connections between the input and the state, as well as between the current and the previous states and b is the displacement vector;

f – is a nonlinear activation function (usually ReLU or tanh).

However, they should not be confused with regression models, which, on the contrary, are used to predict outcomes based on input variables, either continuous or categorical [3]. Although RNN networks are sometimes used for image sequence processing tasks (e.g. video stream), road sign recognition in a video stream is a task that requires local processing of each frame, the context between frames plays a minor role, as each sign is processed independently. The use of RNNs will not be appropriate for this task and will only complicate the model without adding any significant advantages. However, it should be noted that this type of neural network can also be actively used in ADAS, for example, to solve the problem of predicting the movement of potential obstacles.

2.2. Transformers and their disadvantages for character recognition

Transformers, in particular Vision Transformers (ViT), are modern constructs that have demonstrated high accuracy in many computer vision tasks. Their work is based on the self-attention mechanism, which allows the model to analyse global dependencies between parts of the data (in our case, an image).This mechanism is described by the formula [4]:

$$\mathrm{A}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2}$$

where *Q, K, V* – are matrices of queries, keys and values obtained through linear transformations from the input data;

d_k – the dimension of the key space;

QK^T – characterises the similarity between data elements;

softmax – is a function of normalising weights to make them probabilities.

To process images, transformers break them into small patches:

$$x_{\text{patch}} \in R^{(P\times P\times C)} \quad \rightarrow \quad x_{\text{patch}} \cdot W \in R^D, \tag{3}$$

where P×P – is the size of the patch, C is the number of channels (for images, this is usually RGB: C=3);

W – is a weighting magic that converts the patch into a vector of dimension D.

Transformers are able to process global context, which is important for classifying complex scenes or analysing interactions between objects. However, they have a quadratic computational complexity (O(n2)), even on medium-resolution images, which makes them difficult to use on mobile devices. Transformers are also overpowered for the task of road sign recognition. Road signs usually have distinct local features that can be efficiently extracted using simpler types of neural networks. However, transformer-based methods can be beneficial in tasks that require complex environment analysis and trajectory planning, such as maze navigation using coevolutionary algorithms like SAFE, where spatial awareness and exploratory behavior are prioritized [5].

2.3. Convolutional neural networks (CNN) - the optimal solution

Convolutional Neural Networks (CNNs) are the best for image processing and visual data analysis. Their architecture is based on the biological laws of the human visual system, especially on the mechanisms of local feature extraction in the brain. Classical CNNs consist of a combination of several types of layers in different architectural approaches, each of which performs recognition and selection of certain features. The convolutional layer is the main one, which uses a set of filters (called convolutional kernels) to scan the image and extract local features such as edges, corners, and textures. The formula for the convolution operation:

$$y(i,j) = \sum_{m=0}^{k-1}\sum_{n=0}^{k-1} x(i+m, j+n) \cdot w(m,n), \tag{4}$$

where *x(i,j)* – is the value of the input image pixel at coordinates *(i,j)*;

w(m,n) – convolution kernel at position *(m,n)*;

$k\times k$ – the size of the convolution kernel;

b – is the offset added for each output neuron;

y(i,j) – is the output value resulting from the convolution.

Convolutional kernels allow the model to automatically detect local features such as edges, corners, and textures, which is critical for road sign recognition. After convolutional layers, pooling is usually used to reduce the dimensionality of the feature map and increase the robustness to small shifts [6]:

$$y(i,j) = \max_{m,n}\{x(i+m, j+n)\}, \tag{5}$$

where m,n – is the size of the subsample;

$x(i,j)$, $y(i,j)$ – are the same as in formula (4).

Where max-pooling is used to select the maximum value in each region. The final stage is the fully connected layers that perform the classification:

$$y = Wx + b, \tag{6}$$

where x – is a feature vector obtained from the previous convolutional layers;

W – weight matrix;

b – offset;

y – is the output vector of classes (in our case, the probability for each sign).

Thus, in the initial convolutional layers, CNNs detect basic features such as edges or corners of each object. Then, as the depth increases, the network begins to identify more complex patterns, such as geometric shapes or even individual parts of road signs. Compared to fully connected networks, convolutional operations significantly reduce the number of parameters. For example, for a 64 × 64 × 3 image, if only a fully connected layer is used, 12.3 million parameters are required for a layer of 1000 neurons. Using the principle of local filters described above, this figure in CNN can be reduced to several thousand. For road sign recognition, local patterns such as sign shape, textures, or contrast are important, and CNNs automatically extract these patterns due to their architecture [7].

Thus, convolutional neural networks are the best choice for road sign recognition due to their ability to extract local features, robustness to biases, computational efficiency, and ability to work in real time.

3. Architecture selection

Convolutional neural networks (CNNs) are well-suited for traffic sign recognition due to their ability to extract local visual features like shape and texture, which are crucial under varying conditions. A wide range of CNN architectures exist, from simple models like LeNet-5 to more advanced ones such as VGG, Xception, ResNet, EfficientNet, and MobileNet.

For real-time or resource-constrained applications, lightweight models like MobileNet offer a good balance of speed and accuracy. More complex networks like ResNet-50 are better suited for high-performance environments. This section focuses on selecting an architecture that aligns with the system's computational constraints and performance requirements.

3.1. Xception architecture

The Xception architecture is an extension of the Inception model, built on the concept of depthwise separable convolutions. This technique splits standard convolutions into two steps: a depthwise convolution, which processes each input channel separately, and a pointwise convolution, which combines information across channels. This significantly reduces the number of parameters and computations. Xception takes input images of size 299x299x3 (RGB). It begins with a standard convolution layer, followed by a series of depthwise separable convolutions combined with ReLU activations and batch normalization. The core of the model consists of 36 convolutional layers organized into 14 modules with skip connections, similar to those in ResNet [8]. A schematic representation is shown in Figure 1.

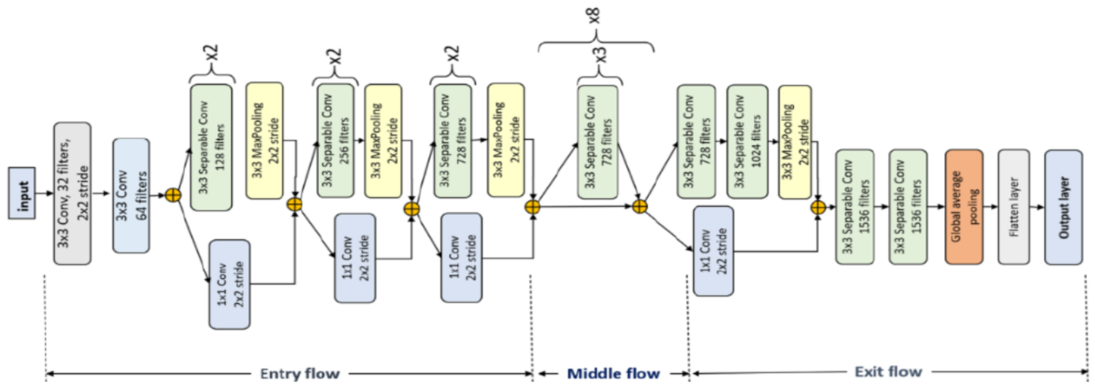

input
3x3 Conv, 32 filters, 2x2 stride
3x3 Conv 64 filters
x2
3x3 Separable Conv 128 filters
3x3 MaxPooling 2x2 stride
1x1 Conv 2x2 stride
x2
3x3 Separable Conv 256 filters
3x3 MaxPooling 2x2 stride
1x1 Conv 2x2 stride
x2
3x3 Separable Conv 728 filters
3x3 MaxPooling 2x2 stride
1x1 Conv 2x2 stride
x8
x3
3x3 Separable Conv 728 filters
3x3 Separable Conv 728 filters
3x3 Separable Conv 1024 filters
3x3 MaxPooling 2x2 stride
1x1 Conv 2x2 stride
3x3 Separable Conv 1536 filters
3x3 Separable Conv 1536 filters
Global average pooling
Flatten layer
Output layer
Entry flow
Middle flow
Exit flow

Figure 1: Schematic representation of the Xception architecture [9].

The output of the model is a vector derived from the Global Average Pooling layer with a size corresponding to the number of classes. As a result, Xception provides high accuracy of image classification on large datasets such as ImageNet. However, the high number of parameters (about 22 million) and significant computational requirements make it less suitable for use on resource-constrained devices such as mobile phones.

3.2. EfficientNet architecture

EfficientNet, introduced by Google Research in 2019, is an optimized architecture for image classification. Its core idea is Compound Scaling, a method that uniformly scales a model's depth, width, and input resolution to improve performance efficiently. The baseline model, EfficientNet-B0, is built upon MobileNetV2 and incorporates inverted residual blocks along with the Swish activation function. The architecture includes convolutional layers combined with batch normalization and activation, ending with global average pooling and a final fully connected layer [10, 11].

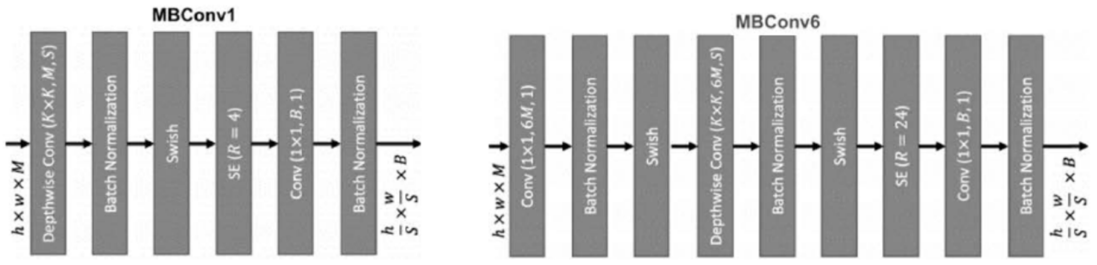

MBConv1
$h \times w \times M$
Depthwise Conv $(K \times K, M, S)$
Batch Normalization
Swish
SE $(R = 4)$
Conv $(1 \times 1, B, 1)$
Batch Normalization
$\frac{h}{S} \times \frac{w}{S} \times B$
MBConv6
$h \times w \times M$
Conv $(1 \times 1, 6M, 1)$
Batch Normalization
Swish
Depthwise Conv $(K \times K, 6M, S)$
Batch Normalization
Swish
SE $(R = 24)$
Conv $(1 \times 1, B, 1)$
Batch Normalization
$\frac{h}{S} \times \frac{w}{S} \times B$

Figure 2: Schematic representation of the EfficientNet architecture [11].

EfficientNet achieves a strong balance between accuracy and efficiency through compound scaling of depth, width, and resolution. As shown in Figure 2, it outperforms many traditional models, though even its compact versions are typically more resource-intensive than MobileNet, making the latter more suitable for real-time or resource-constrained applications.

3.3. MobileNet architecture

MobileNet is specifically designed for mobile devices, employing depthwise separable convolutions to minimize computational complexity. Unlike traditional convolutions, which process both spatial data (within an image) and channel dependencies (across different color channels) in a single operation, depthwise separable convolutions split this into two stages: the depthwise convolution operates on each channel individually, focusing only on spatial data, while the pointwise convolution uses 1x1 filters to merge information across channels. This method significantly reduces the number of computations required. While a conventional convolution has a computational complexity of $O(D_k^2 \times M \cdot N)$, where D_k is the size of the convolution kernel, M is the number of input channels, and N is the number of output channels, a deeply separated convolution has a complexity of $(D_k^2 \cdot M + M \cdot N)$. The key difference is also shown in Figure 3 [12].

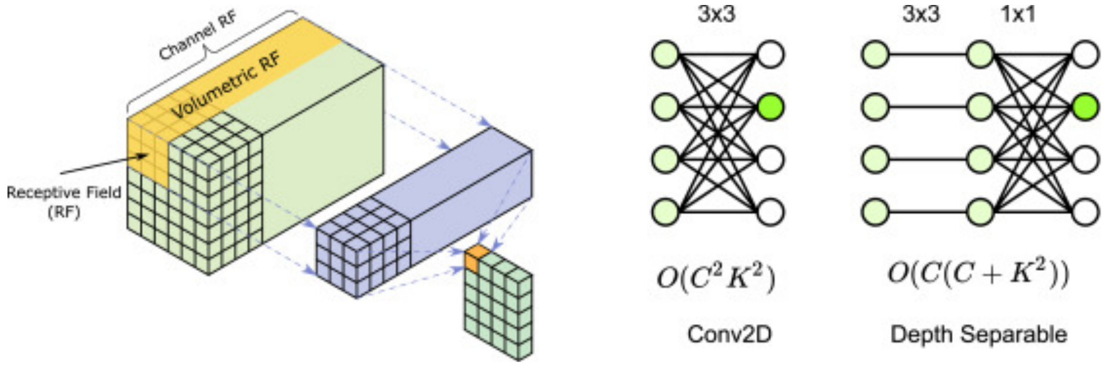


Figure 3: Comparison of conventional and distributed convolution [13].

MobileNet is optimized for speed and efficient use of limited memory resources, making it ideal for mobile and embedded devices. Its architecture is composed of a series of sequential blocks that include depthwise separable convolutions, Batch Normalization, and ReLU6 activations, which help prevent oversaturation of activations in resource-constrained environments. Pointwise (1x1) convolutions are employed to combine information across channels, while convolutional pooling is used in some configurations to reduce the spatial dimensions of data.

MobileNet's flexibility is further enhanced by key parameters: the Width Multiplier (α), which adjusts the number of channels in each layer (e.g., α=0.5 reduces the number of channels by half), and the Resolution Multiplier (ρ), which alters the resolution of input data to balance speed and performance. In its default configuration, MobileNet processes 224×224×3 images, generates a class probability vector, and uses Global Average Pooling to minimize the risk of overfitting, ensuring efficient learning in resource-limited settings.

Given the system's requirements, the MobileNetV2 architecture was selected, as it utilizes backpropagation to minimize parameters without compromising accuracy. While Xception and EfficientNet offer higher accuracy, their computational complexity exceeds the needs of a mobile application. MobileNetV2 guarantees performance and compatibility with most devices, which is a critical factor [14]. Additionally, accuracy is improved by augmenting the data during training, incorporating Dropout layers to mitigate overfitting, and quantizing the model to reduce its size and accelerate computation without a notable loss in performance.

4. Technologies for implementing a neural network

The choice of programming language is essential for implementing a neural network for traffic sign classification. Python, C++, and C# are the main contenders, each with unique advantages and drawbacks. Python is widely used in machine learning due to its simple syntax and a vast ecosystem of libraries like TensorFlow, Keras, PyTorch, and NumPy, which support data processing, model training, and deployment. It also integrates well with cloud platforms like Google Colab and Kaggle, providing easy access to computational resources. However, Python's performance is lower than C++ for computationally intensive tasks.

C++ is known for its high performance and control over hardware, making it ideal for resource-demanding applications. It supports GPU computations via CUDA and libraries like TensorFlow and PyTorch, but its complexity and limited tools for neural network development make it less flexible than Python. C# is commonly used for Windows and mobile app development but has a less developed machine learning ecosystem, with tools like ML.NET not offering the same functionality as Python-based frameworks.

Given Python's advantages, it was chosen for the neural network development due to its flexibility, ease of use, and extensive library support, which facilitates rapid model prototyping and integration with cloud platforms. Python's ability to leverage GPUs and optimize models with TensorFlow Lite compensates for its performance limitations, ensuring real-time processing speed.

For the framework, TensorFlow with Keras was selected due to its support for distributed computing, easy model building, and deployment on various platforms. TensorFlow's extensive tools for data handling, optimization, and integration make it ideal for this project. Keras simplifies the process of creating, training, and evaluating models with its high-level API, while TensorFlow's advanced features like the TensorFlow Data API and TensorFlow Addons provide additional support for data augmentation and model customization [15].

For model training, the Kaggle platform was chosen due to its significant advantages over other free environments. Kaggle provides access to two NVIDIA Tesla T4 GPUs simultaneously, allowing efficient processing of large datasets and faster model training. The platform offers up to 30 hours per week of free GPU usage, with long sessions of up to 9 hours, enabling continuous experimentation and lengthy computations. In contrast, Google Colab provides free GPU access but limits continuous sessions to 4 hours and a total of 12 hours per day, with breaks between sessions. Kaggle also offers an easy way to upload custom datasets, which will be useful during model training.

For data preparation, Python is used due to its versatility and extensive library support. Libraries like NumPy are used for working with numerical arrays, pandas for handling tabular data, OpenCV for image preprocessing (resizing, normalization, augmentation), and Matplotlib for visualizing preparation stages.

5. Creating a dataset

The problem of creating an effective dataset for traffic sign recognition is crucial for quality model training. Most existing datasets, like GTSRB, use images with a size of 33×33 pixels. While this size is optimal for neural network training due to its compactness, it doesn't reflect real-world conditions. In practice, traffic signs are often captured in high-resolution video streams (e.g., 512×512 pixels or more), and resizing them to 33×33 pixels leads to a loss of important details. This is especially critical for real-time systems on client devices.

One possible solution is to "cut" the input image into smaller sections that match the model's input size. For instance, a large image can be split into 33×33 pixel fragments, and predictions are made for each fragment. However, this approach has several drawbacks: it significantly increases the number of predictions, affecting real-time processing speed, requires more memory and computational power, and complicates the client-side application architecture.

Another approach is to use deep learning methods like Super-Resolution to enhance image resolution before inputting them into the model. While this provides more details and potentially improves prediction accuracy, it has its own limitations: Super-Resolution increases processing time per frame, and artificially enhanced images may contain artifacts, which could reduce model accuracy.

Considering these limitations and popular solutions for adapting models to existing datasets, it was concluded that this approach would not significantly improve performance. Therefore, a decision was made to create a custom dataset tailored to the specific requirements of the task.

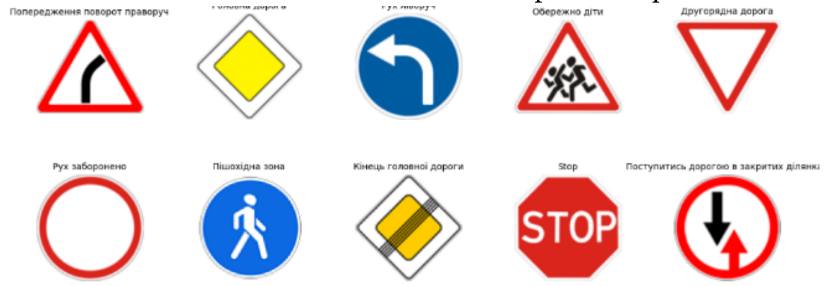


Figure 4: Example of a part of road sign classes for emulating a dataset

The main idea is to use road sign images with an initially higher resolution, which avoids the need to reduce the input layer of the model. However, collecting ready-made images of a higher resolution will be much more time-consuming, which does not fit into the timeframe of the system development. Therefore, it was decided to emulate the dataset through the following preparation stages:

- to create the background and prepare for the generation of compositions, about 2000 random images with natural environment, city roads, etc. were collected. These images will correspond to the type of data that the system will receive in real use
- due to the limited resources for training the model in this work, it was decided to use only 20 classes of road signs (Figure 4), which are the most common and important for traffic
- part of the sign images were taken from open datasets, such as Traffic Signs in Post-Soviet States, which contain real high resolution images of road signs. The rest of the images were collected manually.

After completing the preparatory stage, a Python script was developed that contains an algorithm for creating a dataset with the overlay of objects (road signs) on random images. A diagram of this algorithm is shown in Figure 5.

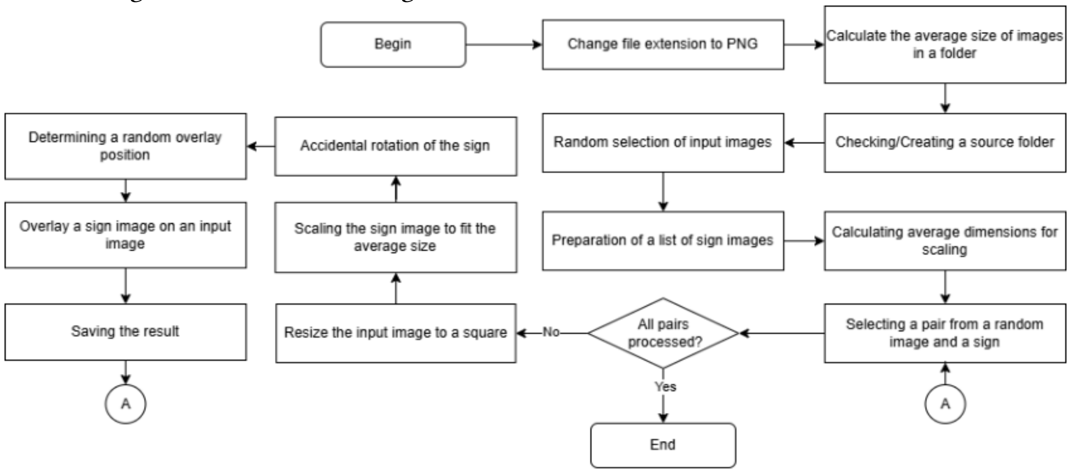
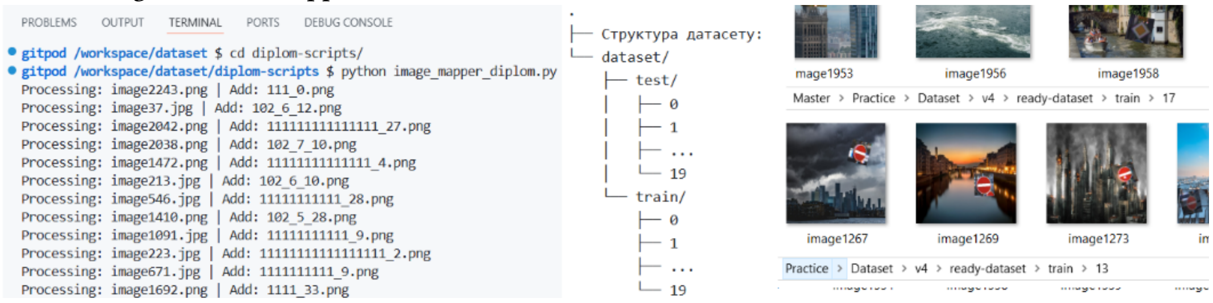

Begin
Change file extension to PNG
Calculate the average size of images in a folder
Checking/Creating a source folder
Random selection of input images
Preparation of a list of sign images
Calculating average dimensions for scaling
Selecting a pair from a random image and a sign
All pairs processed?
No
Yes
End
Resize the input image to a square
Scaling the sign image to fit the average size
Accidental rotation of the sign
Determining a random overlay position
Overlay a sign image on an input image
Saving the result
A
A

Figure 5: Data synthesis algorithm for the dataset

At the initial stage, the necessary directories for image processing are created. The input directory contains base images that will serve as backgrounds for overlaying. The second directory holds the set of objects (traffic signs) to be overlaid on the base images. An output directory is also created to store the results. At this stage, it is checked whether all images are in compatible formats for processing and whether there are enough base images and objects to ensure the required data volume. To ensure proper scaling of objects before overlaying, the average size of the base images is calculated. The algorithm computes the average width and height of all images in the input directory, and these values are used as a reference to resize the objects to the correct proportions, maintaining the natural appearance of the overlaid elements.



```
PROBLEMS   OUTPUT   TERMINAL   PORTS   DEBUG CONSOLE

● gitpod /workspace/dataset $ cd diplom-scripts/
● gitpod /workspace/dataset/diplom-scripts $ python image_mapper_diplom.py
  Processing: image2243.png | Add: 111_0.png
  Processing: image37.jpg | Add: 102_6_12.png
  Processing: image2042.png | Add: 111111111111111_27.png
  Processing: image2038.png | Add: 102_7_10.png
  Processing: image1472.png | Add: 11111111111111_4.png
  Processing: image213.jpg | Add: 102_6_10.png
  Processing: image546.jpg | Add: 11111111111_28.png
  Processing: image1410.png | Add: 102_5_28.png
  Processing: image1091.jpg | Add: 11111111111_9.png
  Processing: image223.jpg | Add: 1111111111111111_2.png
  Processing: image671.jpg | Add: 1111111111_9.png
  Processing: image1692.png | Add: 1111_33.png
```

```
.
├── Структура датасету:
└── dataset/
    ├── test/
    │   ├── 0
    │   ├── 1
    │   ├── ...
    │   └── 19
    └── train/
        ├── 0
        ├── 1
        ├── ...
        └── 19
```

mage1953
image1956
image1958
Master › Practice › Dataset › v4 › ready-dataset › train › 17
image1267
image1269
image1273
Practice › Dataset › v4 › ready-dataset › train › 13

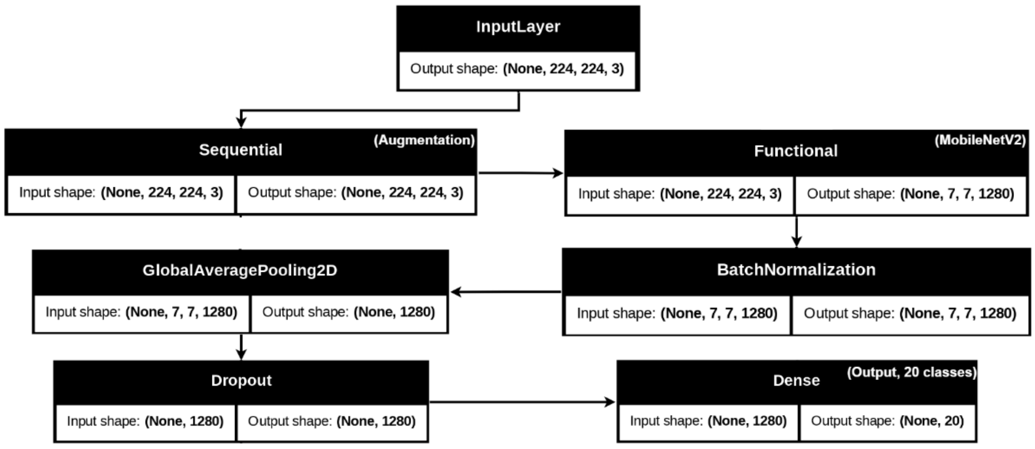
Figure 6: How the algorithm works and examples of images in the dataset

Next, the base images are scaled or made square, if necessary, to ensure consistency with other images in the dataset. The objects to be overlaid are loaded and resized according to the average size of the base images. During processing, random rotation is applied within a specified range (-20° to +20°), adding variation to the appearance of the overlaid elements. For each base image, a random position is selected for the object overlay. The algorithm ensures that the object fully fits within the bounds of the base image and does not extend beyond its edges. The overlay is applied considering transparency, if present in the object images. After processing, the image is saved in the output directory, typically in PNG format. If the dataset includes multiple object classes, the algorithm repeats the steps for each class separately.

The finished images are divided into classes and split into training (80%) and test (20%) samples. An example of the generated data is shown in Figure 6. This approach ensures the variability of the dataset and its adaptation to the real conditions of the model.

6. Creating and training a model

The architecture of our modified MobileNetV2 is designed to efficiently extract object features while minimizing computational costs. It consists of several groups of layers, each of which processes certain aspects of the input data and gradually builds the feature space (Figure 7). The model's input data is 224×224×3, and it starts with an augmentation block that applies random scaling and rotation. Reflections are not used because they could, for example, turn a “Turn Left” sign into a “Turn Right” sign while maintaining the original class, which would confuse the model [16].


InputLayer
Output shape: (None, 224, 224, 3)
Sequential
(Augmentation)
Input shape: (None, 224, 224, 3)
Output shape: (None, 224, 224, 3)
Functional
(MobileNetV2)
Input shape: (None, 224, 224, 3)
Output shape: (None, 7, 7, 1280)
BatchNormalization
Input shape: (None, 7, 7, 1280)
Output shape: (None, 7, 7, 1280)
GlobalAveragePooling2D
Input shape: (None, 7, 7, 1280)
Output shape: (None, 1280)
Dropout
Input shape: (None, 1280)
Output shape: (None, 1280)
Dense
(Output, 20 classes)
Input shape: (None, 1280)
Output shape: (None, 20)
Figure 7: Simplified view of the model architecture

The model begins with a standard 3×3 convolution layer with stride 2 and ReLU6 activation, reducing spatial dimensions and detecting basic features. The core of the architecture is a series of inverted residual blocks, each following a specific pattern:

1. First, an expansion phase increases the number of channels in the input tensor by a certain factor, allowing the model to shift toward higher-level features that better capture complex structures like sign contours and textures.
2. Then, depthwise convolutions process each channel individually to extract localized spatial features, such as circular shapes or distinctive angles common in traffic signs.
3. Next, a compression phase reduces the number of channels back to the original size, improving computational efficiency by retaining only the most relevant features.
4. Finally, a skip connection links the input and output of the block, preserving low-level features like colors or contrast, which are crucial for recognizing traffic signs.

After passing through the bottleneck layers, the model performs GlobalAveragePooling, which compresses each feature map into a single value. This operation creates a compact, high-level feature vector that summarizes the entire input image and significantly reduces the number of parameters. This vector is then passed to a fully connected (dense) layer, where each output corresponds to a specific class. The softmax activation function ensures the outputs represent class probabilities that sum to one. To improve training stability and generalization, the model uses BatchNormalization to normalize intermediate layer outputs and Dropout (rate 0.5) to randomly deactivate neurons during training. These techniques help prevent overfitting, especially when training data is limited or imbalanced.

Initially, the base MobileNetV2 model is loaded with pretrained weights (e.g., from ImageNet) and frozen, meaning its parameters remain unchanged. Only the new top layers are trained to adapt to the specific traffic sign recognition task, allowing fast and stable convergence. After the top layers are trained, the model enters a fine-tuning phase. Some base layers are unfrozen, and training continues with a lower learning rate. This gradual adaptation refines deeper features to better match the new dataset while preserving the benefits of pretraining, often resulting in significantly improved accuracy.

Now that the model is ready for training, we proceed to load the dataset. For deep learning tasks with large image collections, loading all images into memory at once is inefficient and often leads to memory overflow. In our case, with 16,000 images sized 224×224 pixels, doing so could crash the runtime environment. To handle this, we use TensorFlow's tf.data.Dataset API, which allows for streaming and preprocessing data on-the-fly in small batches, greatly reducing memory usage.

The dataset is organized into train and test folders, each containing subfolders for every class. File paths and labels are generated automatically based on these subfolder names. Each image is read from disk using tf.io.read_file, decoded into RGB format, resized to 224×224 pixels with tf.image.resize, and normalized to values between 0 and 1. Using tf.data.Dataset.from_tensor_slices, the file paths and labels are combined into a dataset object. The pipeline then applies:

- shuffle to randomize data order
- batch(32) to process in small chunks
- prefetch to load future batches in the background for better performance

Training proceeds in two stages. In the first stage, the base MobileNetV2 layers remain frozen to retain knowledge from pretraining. Only the added classification layers are trained, allowing the model to adapt to the new dataset. The model is compiled with the Adam optimizer, which balances gradient direction and variance, and sparse categorical crossentropy is used as the loss function, suitable for multiclass classification with integer labels. The process is controlled by several callbacks:

- ModelCheckpoint – saving the best version of the model
- EarlyStopping – stopping when there is no improvement for 15 epochs
- ReduceLROnPlateau – reduce the learning rate in case of stagnation

At this stage, the model is trained for a limited number of epochs (15 in our case), primarily to quickly adapt the newly added classification layers to the new dataset. The goal is not full convergence, but rather initial tuning of the top layers.

Once this initial training is complete, we unfreeze a portion of the base model—typically the upper layers, which capture more task-specific features. This allows the model to refine not only the new top layers but also adjust deeper feature representations for better accuracy.

In our case, we unfreeze all layers after index 100, keeping the earlier ones frozen to maintain stability. For this fine-tuning phase, we use a lower learning rate (1e-5) to avoid drastic weight updates that could disrupt the pretrained knowledge. This phase runs longer (around 50 epochs) to allow the model to gradually refine its internal representations. Training curves (Figure 8) show how the model's performance improves over time. Initially, validation accuracy may remain low as the model focuses on learning basic patterns. However, with continued training and fine-tuning, both accuracy and loss improve significantly, indicating successful adaptation.

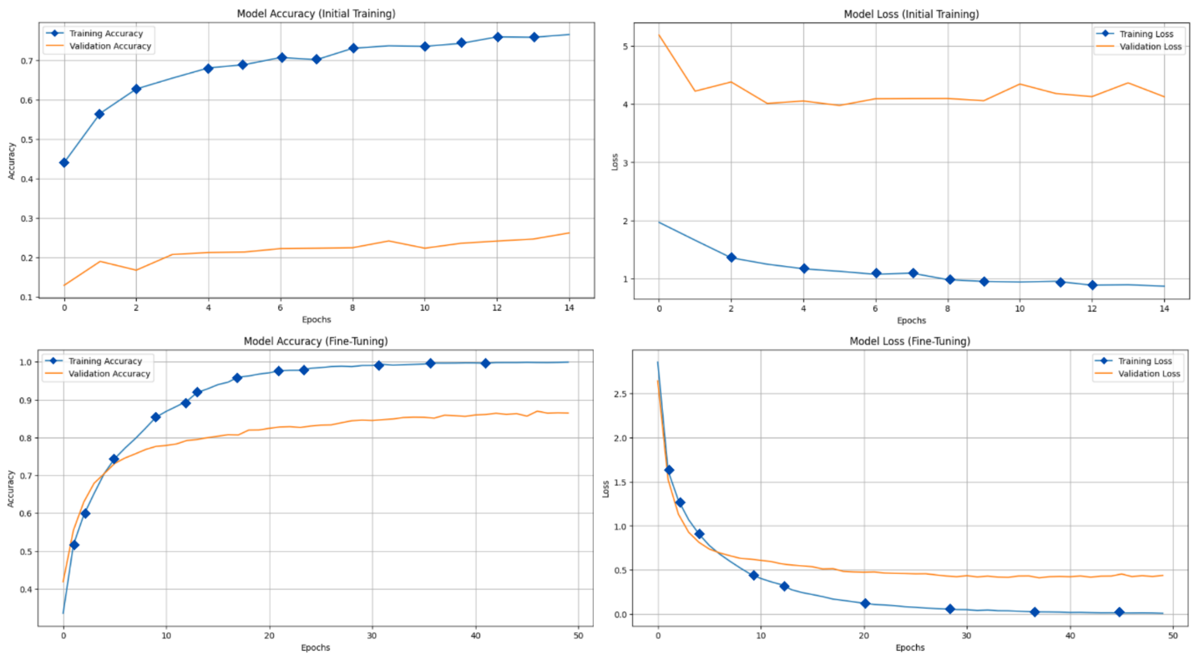
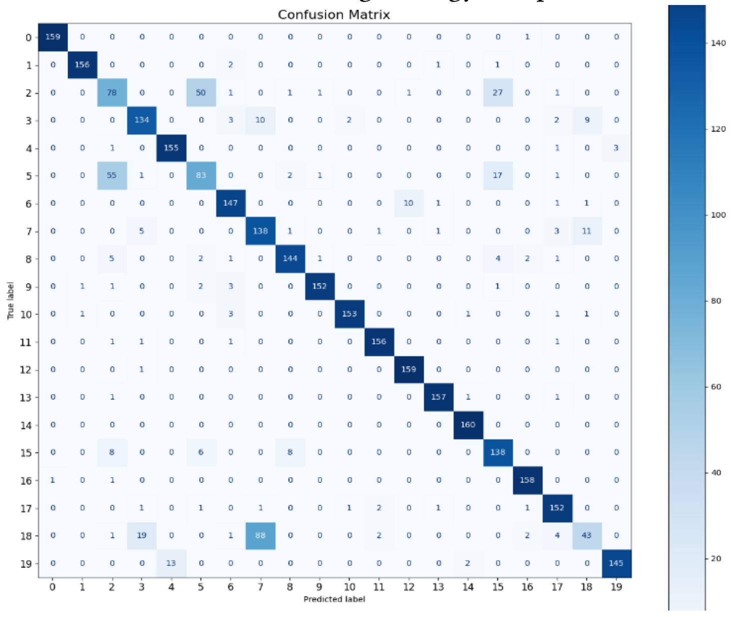
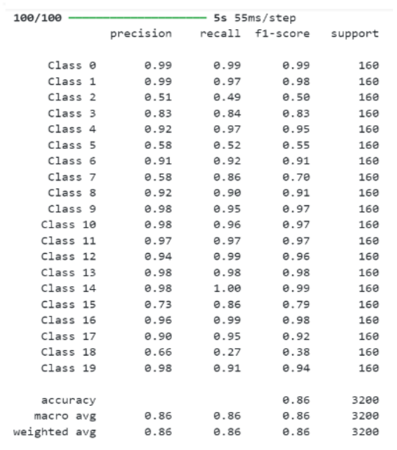


Figure 8: Accuracy and loss plots for model training

Throughout both training stages, the model was trained within optimal limits to avoid underfitting or overfitting. Early in fine-tuning, training accuracy reached 75% while validation accuracy was 68%, indicating initial adaptation. As training progressed, these improved to 85% and 76%, respectively. By the final epochs, the model achieved 99% training accuracy and 87% validation accuracy, demonstrating strong generalization to unseen data. These results confirm the effectiveness of the chosen training strategy and parameter settings.





```
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 55ms/step
              precision    recall  f1-score   support

     Class 0       0.99      0.99      0.99       160
     Class 1       0.99      0.97      0.98       160
     Class 2       0.51      0.49      0.50       160
     Class 3       0.83      0.84      0.83       160
     Class 4       0.92      0.97      0.95       160
     Class 5       0.58      0.52      0.55       160
     Class 6       0.91      0.92      0.91       160
     Class 7       0.58      0.86      0.70       160
     Class 8       0.92      0.90      0.91       160
     Class 9       0.98      0.95      0.97       160
    Class 10       0.98      0.96      0.97       160
    Class 11       0.97      0.97      0.97       160
    Class 12       0.94      0.99      0.96       160
    Class 13       0.98      0.98      0.98       160
    Class 14       0.98      1.00      0.99       160
    Class 15       0.73      0.86      0.79       160
    Class 16       0.96      0.99      0.98       160
    Class 17       0.90      0.95      0.92       160
    Class 18       0.66      0.27      0.38       160
    Class 19       0.98      0.91      0.94       160

    accuracy                           0.86      3200
   macro avg       0.86      0.86      0.86      3200
weighted avg       0.86      0.86      0.86      3200
```

Figure 9: Confusion matrix and classification report

The confusion matrix analysis (Figure 9) confirms strong model performance, with most predictions correctly aligned along the diagonal, indicating high classification accuracy across categories. Some misclassifications are observed, primarily between visually similar signs—such as left and right turn warnings—which is expected and acceptable within the task's scope. The classification report shows an overall accuracy and average metrics (macro and weighted) of 86%, which is a solid result for multi-class image recognition. Classes like 0, 1, 4, 9–14, and 16 achieved excellent precision, recall, and F1-scores (0.91–1.00), while classes such as 2, 5, 7, 15, and 18 showed lower scores due to visual similarity. Notably, class 7 had high recall but low precision, indicating overprediction. Still, balanced macro and weighted averages confirm that the model performs reliably across all classes.

An additional manual test was conducted using a separate set of 30 randomly selected images not involved in training or validation. The model correctly classified 28 out of 30, achieving an accuracy of 93.3%, with only 2 misclassifications (Figure 10). Overall, the results confirm that the model is both effective and reliable for traffic sign classification. Despite initial fluctuations in accuracy, the model consistently improved and maintained strong performance across training, validation, and independent testing stages.

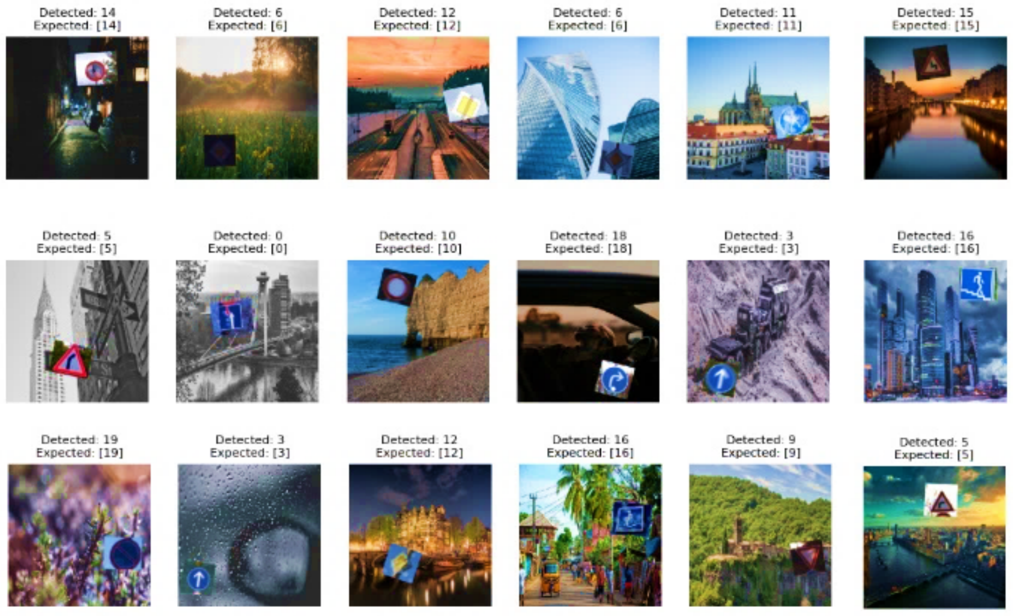


Figure 10: Model performance on the test set

7. Conclusions

The study presented an efficient method for developing a road sign recognition model suitable for mobile devices with limited computational resources. A key accomplishment was the enhancement of the training sample generation process, where data augmentation and synthetic data generation techniques increased the model's robustness to variations in lighting, perspective, and noise.

The selection of the MobileNetV2 architecture proved to be a practical choice for image classification in resource-constrained environments. By employing stepwise training—freezing the initial layers to preserve pre-trained features and adapting the model to a new class set—we achieved high classification accuracy. Additionally, model optimisation through quantisation significantly reduced its size while maintaining the necessary predictive accuracy.

Experimental results validated the effectiveness of the proposed approach: the model achieved high accuracy in road sign recognition under real-world conditions, demonstrating its potential for practical deployment. These methods can be applied to further enhance computer vision models designed for environments with limited computational capacity.

Declaration on Generative AI

During the preparation of this article, the authors used Gemini 2.5 Flash artificial intelligence tools to assist with grammar and spelling correction, as well as to check the translation of some syntactic structures. The final content has been carefully reviewed and edited by the authors, who are solely responsible for the accuracy and integrity of the publication.

References

[1] Graupe, D. (2016). Deep Learning Neural Networks: design and case studies. World Scientific Publishing Company.

[2] Cardot, H. (2011). Recurrent neural networks for temporal data processing. BoD – Books on Demand.

[3] Anatoliy Doroshenko, Dmitry Zhora, Olena Savchuk, and Olena Yatsenko. Application of Machine Learning Techniques for Forecasting Electricity Generation and Consumption in Ukraine. Information Technology and Implementation (IT&I-2023), November 20-21, 2023, Kyiv, Ukraine. CEUR Workshop Proceedings (CEUR-WS.org), vol-3624, pp. 136-146. https://ceur-ws.org/Vol-3624/Paper_12.pdf.

[4] Doshi, K. (2021) Transformers explained visually (part 3): Multi-head attention, Deep Dive – TowardsDataScience. URL: https://towardsdatascience.com/ transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853.

[5] Omelianenko I., Doroshenko A., Rodin Y. Autonomous navigation through the maze using coevolution strategy. In: I. Sinitsyn, Ph. Andon (Eds.) Proceedings of the 14th International Scientific and Practical Programming Conference (UkrPROG 2024). Kyiv, Ukraine, May 14–15, 2024. P. 301–311. https://ceur-ws.org/Vol-3806/S_29_Omelianenko_Doroshenko_Rodin.pdf.

[6] Ozturk, S. (2022). Convolutional neural networks for medical image processing applications. CRC.

[7] Liang, Q., Wang, W., Liu, X., Na, Z., Jia, M., & Zhang, B. (2020). Communications, signal processing, and systems: Proceedings of the 8th International Conference on Communications, Signal Processing, and Systems. Springer Nature.

[8] Hussein, A. A. (2024). Renewable energy: generation and application: ICREGA'24. Materials Research Forum LLC.

[9] M. M. Shibly, T. A. Tisha, T. A. Tani, S. Ripon, "Convolutional neural network-based ensemble methods to recognize Bangla handwritten character," PeerJ Comput. Sci., vol. 7, 2021.

[10] Khang, A., Abdullayev, V., Jadhav, B., Gupta, S., & Morris, G. (2023). AI-Centric Modeling and Analytics: Concepts, Technologies, and Applications. CRC Press.

[11] Truong,T.T. (2021) Recognition framework using transfer learning – IEEEAccess URL: https://www.researchgate.net/publication/357852732_A_Dish_Reco gnition_Framework_Using_Transfer_Learning.

[12] Oliver, N., Pérez-Cruz, F., Kramer, S., Read, J., & Lozano, J. A. (2021). Machine learning and knowledge discovery in databases. Research track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part I. Springer Nature.

[13] T. Wei. (2022) Optimized separable convolution: Yet another efficient convolution operator – AI Open. URL: https://www.sciencedirect.com/science/ article/pii/S2666651022000158.

[14] K_04 understanding of mobilenet – Wikidocs. URL: https://wikidocs.net/165429.

[15] Introduction to TensorFlow and Keras – Deep learning with TensorFlow. URL: https://developmentseed.org/tensorflow-eo-training/docs/Lesson1b_Intro_Tensor Flow_Keras.html.

[16] Anatoliy Doroshenko, Dmytro Zhora, Vladyslav Haidukevych, Yaroslav Haidukevych, and Olena Yatsenko. Predicting 24-Hour Nationwide Electrical Energy Consumption Based on Regression Techniques. CEUR-WS, 2024, vol. 3806, 17 p. https://ceur-ws.org/Vol-3806/S_4_Doroshenko_Zhora_Haidukevych_Yatsenko.pdf.

A. Online Resources

The model training notebook is available at https://gitlab.com/MaksGovor/road-assistant/-/blob/main/road-assistant-model/rsv2.ipynb.