Convolutional Neural Networks

Hybrid MobileNetV3: Impact of Activation Tuning and Feature Extractions

Hoang Tu Bui 24005665

CS6482 Deep Reinforcement Learning

J.J. Collins

Masters in Artificial Intelligence and Machine Learning - SEM 2

1. Introduction

In this project, we explore the architecture of MobileNetV3, delving into its internal mechanisms and leveraging its capabilities to classify satellite images from the EuroSAT-RGB dataset. Our objective is not only to understand the model's efficiency and lightweight design but also to investigate potential enhancements through modifications. Specifically, we incorporate traditional feature extraction techniques such as Sobel filtering, Histogram of Oriented Gradients (HOG) and different activation functions to examine their influence on performance. We integrate handcrafted features with deep learning-based representations. This aims to determine if they can complement modern convolutional networks and improve classification accuracy. Additionally, we experiment with activation functions like GELU, LeakyReLU, and ELU to evaluate their impact on feature expressiveness and overall model performance. Through these modifications and analyses, we seek to gain deeper insights into MobileNetV3's adaptability and potential improvements for remote sensing image classification.

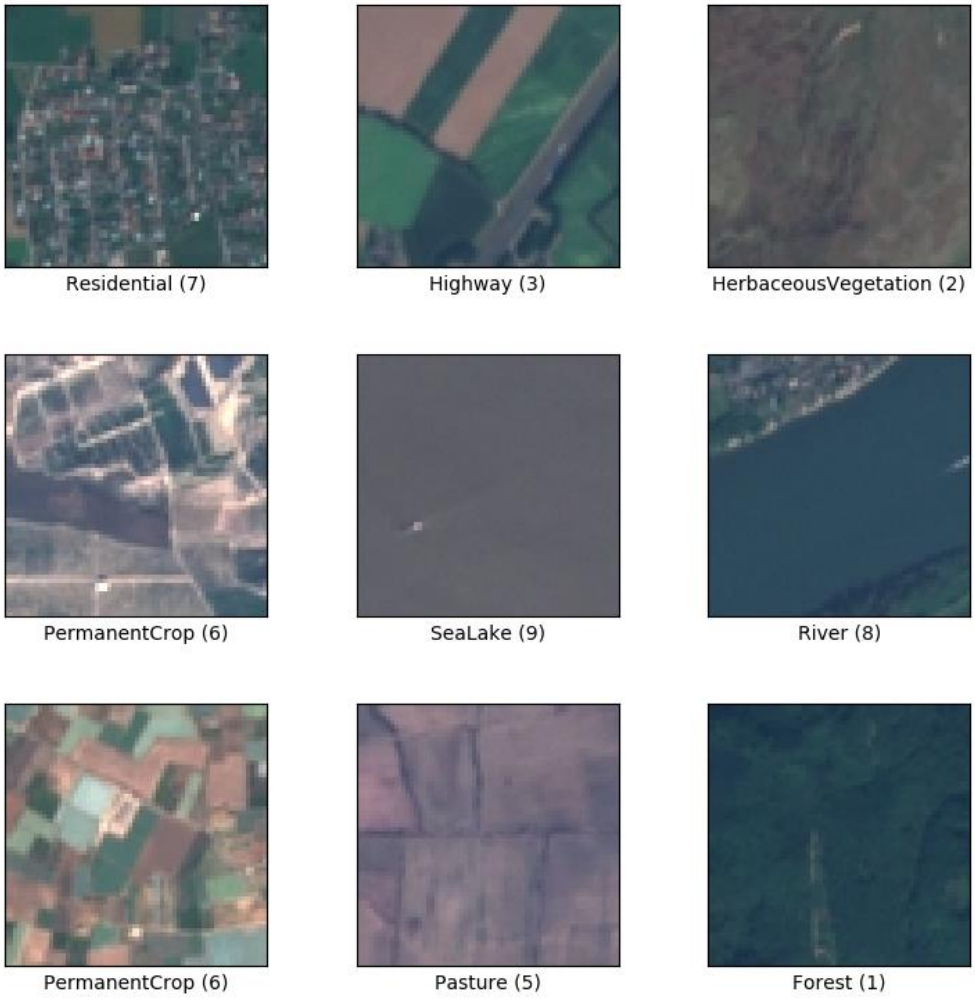
2. Dataset

For this project, we are using the EuroSAT dataset, which consists of satellite images captured by the Sentinel-2 mission of the European Space Agency (ESA).

The EuroSAT RGB dataset is a specialized subset of EuroSAT, focusing on the Red, Green, and Blue (RGB) spectral bands. It comprises 27,000 labeled and geo-referenced images, each with a resolution of 64×64 pixels. These images represent diverse land cover types across 34 European countries, making it a valuable resource for land classification tasks.

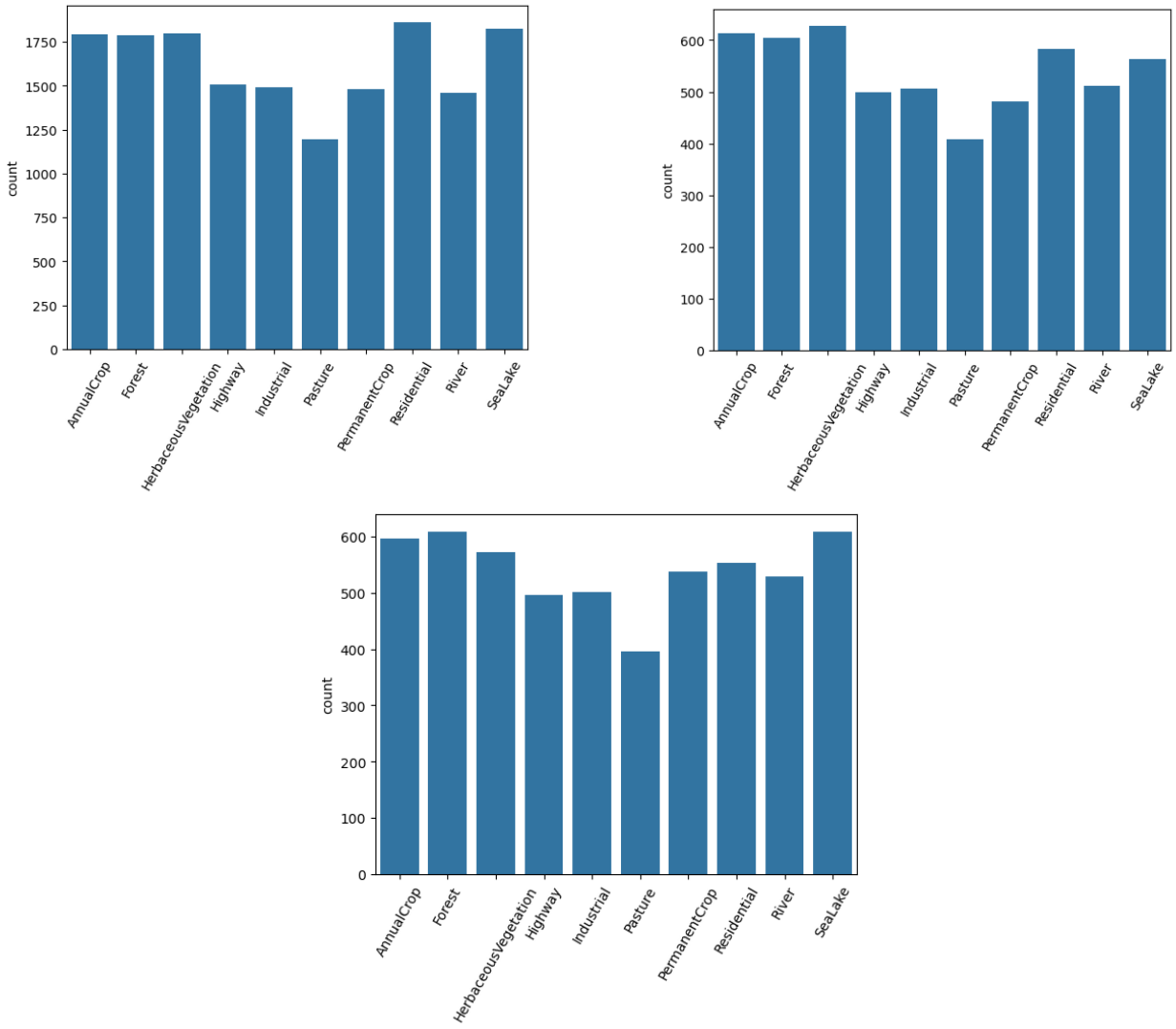
The dataset includes the following 10 land cover classes: Annual Crop, Forest, Herbaceous Vegetation, Highway, Industrial Buildings, Pasture, Permanent Crop, Residential Buildings, River, and Sea/Lake. Figure below shows some sample images with the associated labels.

With its high-quality imagery, the EuroSAT dataset is widely used for developing and benchmarking machine learning models in remote sensing, environmental monitoring, and land use analysis (Helber *et al.*, 2017).



2.1. Data Distribution

The dataset is split into a training set containing 16200 samples and a testing set with 5400 and validation also containing 5400, maintaining a proportional representation of all 10 classes in all subsets. This balanced distribution supports robust model generalization and fair performance assessment. The distribution of the trainset, the validation set, and the test set are shown respectively in Figure below.



2.2. Data Preprocessing

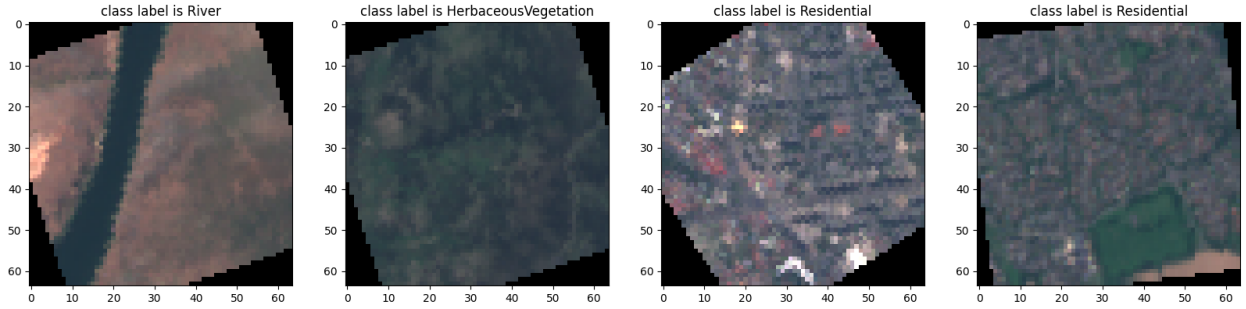
Preprocessing is a crucial step in machine learning tasks, it ensures data consistency and improves converts raw data to machine-friendly format. For images, it involves resizing, normalization, and augmentation techniques. In our case, the dataset is all in 64x64 and the color is RGB with value in range [0,1]. Applying normalization does not bring any significant improvement to the model's performance. Thus, we decided not to normalize the data and only resize all to 64x64 to avoid edge cases.

For data augmentation, we created a pipeline that includes Random Horizontal Flip, Random Vertical Flip, and Random Rotation. During each training epoch, the pipeline first randomly flips images horizontally and vertically with a probability of 50%. This means that from a single image, we can generate up to four variations: the original, a horizontally flipped version, a vertically flipped version, and a version flipped both horizontally and vertically.

Finally, the images are rotated uniformly by up to 45 degrees, introducing additional variability and enhancing the model's ability to generalize to different orientations. These transformations simulate real-world scenarios, reduce overfitting, and improve the robustness of the model by increasing the diversity of the training dataset.

It is interesting to note that these augmentations are only applied to the training dataset because their purpose is to artificially increase the diversity of the training data, helping the model generalize better to unseen data. Applying augmentations to the validation or test sets would distort the true representation of the data, making it difficult to evaluate the model's performance accurately. The validation and test datasets should reflect real-world, unaltered data to ensure fair and reliable assessment of the model's effectiveness.

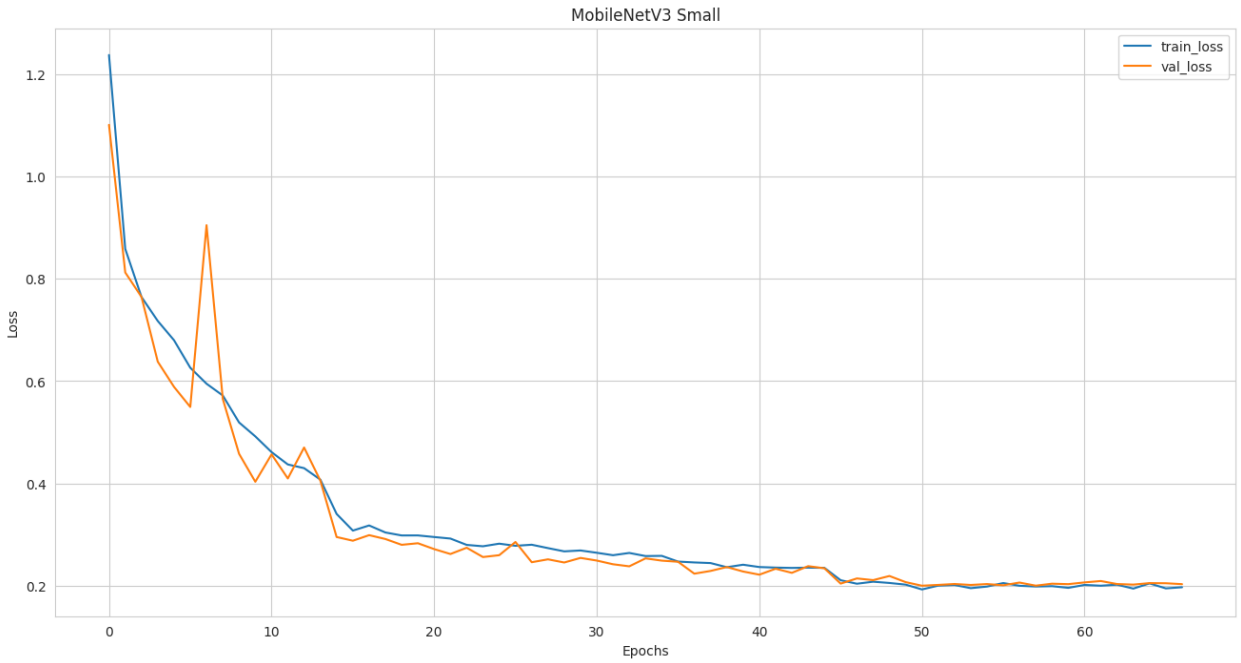
Below are some sample of data after augmenting:



3. Loss Function

The loss function observes the difference between the predictions and the actual values. For this task, which is classification, Cross-Entropy Loss is chosen as it is commonly used for multi-class classification tasks.

Below is the learning curve which is plotted by the loss for each epoch:



4. Optimizer

Optimizers play a crucial role in the performance of neural networks. Various optimizers exist, each with its own strengths and limitations. In this case, the Adam optimizer (optim.Adam) is used for optimization. Adam is an adaptive learning rate algorithm that integrates the benefits of both AdaGrad and RMSProp. By adjusting the learning rate for each parameter individually, it enables faster convergence and improved generalization. The initial learning rate is set to 0.001 and will be reduced by 5 times when the loss stagnates during training.

5. Crossfold

Cross-validation, particularly k-fold cross-validation, is a widely used technique in machine learning to evaluate model performance, especially when dealing with limited datasets. It involves partitioning the dataset into k subsets, training the model on k-1 subsets, and

validating it on the remaining subset. This process is repeated k times, with each subset serving as the validation set once. The results are then averaged to provide a robust estimate of model performance. Cross-validation helps mitigate issues like overfitting and ensures that the model generalizes well to unseen data.

However, in scenarios where the dataset is large and well-balanced, as in the case of the EuroSAT-RGB dataset with 27,000 samples, cross-validation may not be necessary. With ample data, a single split into training, validation, and test sets is often sufficient to provide a reliable assessment of model performance. Cross-validation is computationally expensive, especially for large datasets and complex models like MobileNetV3, as it requires training the model multiple times. Given the size and diversity of the EuroSAT dataset, a standard train-validation-test split ensures efficient use of computational resources while still providing a robust evaluation of the model's generalization capabilities. Thus, in this project, we opted for a traditional split rather than cross-validation, leveraging the dataset's size and balance to achieve reliable results.

6. MobileNetV3

6.1. MobibleNetV1

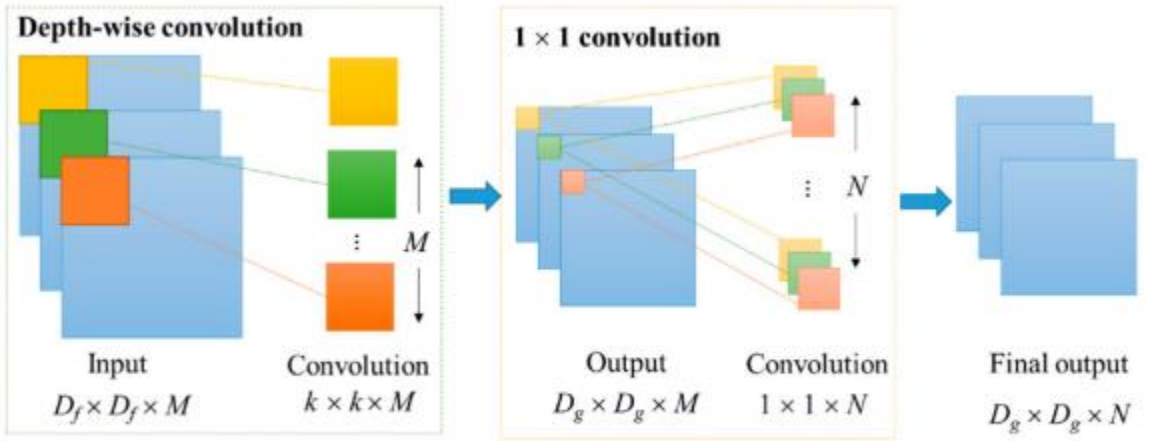
MobileNetV1, introduced by Google researchers in 2017, marked a significant breakthrough in deep learning, particularly for mobile and embedded vision applications. The primary innovation of MobileNetV1 lies in its use of depthwise separable convolutions, which drastically reduce computational cost and model size while maintaining competitive accuracy (Howard *et al.*, 2017).

Traditional convolutional layers perform both filtering and combining of input features in a single step, making them computationally expensive. MobileNetV1, in contrast, achieves state-of-the-art performance on vision tasks while being significantly smaller and faster than previous architectures like VGGNet and ResNet. For example, MobileNetV1 achieved comparable accuracy to these larger models on the ImageNet dataset with only 4.2 million parameters and 569 million multiply-add operations, making it ideal for real-time applications on resource-constrained devices such as smartphones, drones, and IoT systems.

Depth wise Separable Convolutions

One of the core contributions of Howard et al. (2017) was the introduction of depthwise separable convolutions, which significantly reduce computation and model complexity.

In traditional CNNs, filters operate across all input channels simultaneously. A convolutional filter of size K × K × C_{in} (kernel size × input channels) performs both feature extraction and combination in one step. In contrast, depthwise separable convolutions break this into two steps: Depthwise Convolution and Pointwise Convolution.



In Depthwise Convolution, a single filter of size K × K × 1 is applied to each input channel separately, reducing computational cost. While in the later, a 1 × 1 × C_{out} convolution is used to combine the outputs of the depthwise operation.

This approach leads to a dramatic reduction in computation, especially in deeper layers where the number of input channels is large. Mathematically, depthwise separable convolutions reduce computation by a factor of approximately:

$$\frac{1}{Cout} + \frac{1}{K^2}$$

For example, in a standard convolutional layer with a 3×3 kernel and 512 input channels, the reduction factor is around 8-9 times in term of number of multiplications.

Width and Resolution Multipliers

MobileNetV1 introduces two key hyperparameters that allow users to adjust model size and computational cost based on available hardware: Width Multiplier and Resolution Multiplier.

The first one controls the number of filters per layer while the later one Adjusts the input image resolution. These hyperparameters allow developers to tune the model based on their hardware constraints.

6.2. MobileNetV2

MobileNetV2 was designed to address some of the key limitations of MobileNetV1, specifically its ability to learn robust feature representations in low-dimensional spaces. While depthwise separable convolutions significantly reduced computational overhead, they sometimes led to weaker feature representations. To counter this, MobileNetV2 introduced two major architectural innovations: **Inverted Residual Blocks** and **Linear Bottlenecks** Sandler, M. *et al*. (2018).

These innovations made the model more effective for mobile and embedded vision applications while maintaining a low computational footprint. MobileNetV2 achieved an improved balance between efficiency and accuracy, enabling it to perform better on complex tasks while still being lightweight.

Inverted Residual

The inverted residual block is a key innovation introduced in MobileNetV2. Unlike traditional residual blocks in ResNet that follow a **wide → narrow → wide** pattern, MobileNetV2 follows an **inverted structure** of **narrow → wide → narrow**. This design significantly improves efficiency and feature representation while keeping the network compact.

Each inverted residual block consists of three primary layers:

1. **Expansion Layer (1x1 Convolution)**: This layer expands the number of channels in the input feature map, often by a factor of 6 (e.g., increasing 24 channels to 144). By operating in a higher-dimensional space, the model can capture richer feature representations.
2. **Depthwise Convolution (3x3)**: After expansion, a depthwise convolution is applied to each channel separately. This step enables spatial feature extraction while maintaining computational efficiency.
3. **Projection Layer (1x1 Convolution)**: The final step is a 1x1 convolution that reduces the number of channels back to a smaller number (e.g., from 144 back to 24). This step compresses the information, ensuring that only the most important features are retained while minimizing computational costs.

The reason behind this narrow → wide → narrow structure is to allow computations to happen in a high-dimensional space where features can be better separated and learned. Once the learning is complete, the information is projected back to a lower-dimensional space, where it can be stored efficiently with minimal loss. This approach improves both accuracy and efficiency, making MobileNetV2 highly suitable for mobile and embedded applications.

Bottleneck

The Linear Bottleneck is a crucial design improvement in MobileNetV2 that helps mitigate information loss during the final stage of an inverted residual block. Traditional deep networks apply non-linear activation functions (such as ReLU) at every stage of the network, including at the final feature projection. While ReLU is effective in filtering out unimportant activations, applying it to low-dimensional feature representations can cause excessive loss of information. This linear projection prevents the loss of crucial structural details that could be destroyed by a non-linearity like ReLU, especially in lower-dimensional spaces. By preserving feature integrity at the output of each residual block, the model improves feature expressiveness and classification performance, leading to a more accurate and efficient deep learning architecture.

6.3. MobileNetV3

MobileNetV3 is the latest iteration in the MobileNet family, designed to optimize both efficiency and accuracy for mobile and edge devices. Building upon the advancements of MobileNetV1 and V2, it integrates depthwise separable convolutions, inverted residuals, and linear bottlenecks while introducing Squeeze-and-Excitation (SE) modules and Hard-Swish activation to improve performance. Unlike its predecessors, MobileNetV3 was developed through Neural Architecture Search (NAS), allowing automated optimization of its structure. The model comes in two variants—MobileNetV3-Large for high-performance tasks and MobileNetV3-Small for ultra-lightweight applications. In the next section, we will walk through the key components of MobileNetV3 in detail in the next section.

Below is a detailed explanation of the custom MobileNetV3 implementation as described in the notebook. This section breaks down each component of the implementation, including the TRADITION feature extraction class, convolution block, squeeze and excite block, bottleneck block, and the main MobileNet class.

Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d, 3x3	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	✓	RE	2
$56^2 \times 16$	bneck, 3x3	72	24	-	RE	2
$28^2 \times 24$	bneck, 3x3	88	24	-	RE	1
$28^2 \times 24$	bneck, 5x5	96	40	✓	HS	2
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	120	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	144	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	288	96	✓	HS	2
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	conv2d, 1x1	-	576	✓	HS	1
$7^2 \times 576$	pool, 7x7	-	-	-	-	1
$1^2 \times 576$	conv2d 1x1, NBN	-	1024	-	HS	1
$1^2 \times 1024$	conv2d 1x1, NBN	-	k	-	-	1

Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

ConvBlock: Basic Convolutional Unit

The ConvBlock class is a foundational building block of MobileNetV3, encapsulating the core sequence of operations required for efficient feature extraction in lightweight neural networks. This block combines a convolutional layer, batch normalization, and an activation function into a single modular unit, streamlining the architecture while maintaining computational efficiency. Below, we break down its design, implementation, and significance in the MobileNetV3 framework.

```
    # Calculate padding to maintain spatial dimensions after
    padding = (kernel_size - 1) // 2

    # Define the convolutional layer
    self.conv = nn.Conv2d(
        in_chn,
        out_chn,
        kernel_size,
        stride=stride,
        padding=padding,
        bias=False,
        groups=group,
    )
    # Define the batch normalization layer
    self.bn = nn.BatchNorm2d(out_chn)

    # Define the activation function (if provided)
    self.activate = activation() if activation else None
```

The ConvBlock performs three critical operations in sequence:

1. Convolution (nn.Conv2d): Applies spatial filtering to extract features from the input.
2. Batch Normalization (nn.BatchNorm2d): Stabilizes training by normalizing layer outputs.
3. Activation Function: Introduces non-linearity to enable complex pattern learning.

This sequence ensures the network learns robust features while minimizing computational overhead, a key requirement for mobile-optimized architectures.

Squeeze and Excite: Channel Attention Mechanism

The Squeeze-and-Excitation (SE) block is a lightweight, adaptive mechanism designed to enhance the representational capacity of convolutional neural networks (CNNs) by explicitly modeling inter-channel dependencies. Integrated into MobileNetV3, this block dynamically recalibrates feature responses to prioritize informative channels while suppressing redundant ones, achieving a superior balance between accuracy and computational efficiency. Below, we detail its architecture, implementation, and role in MobileNetV3

```
class Squeeze_and_Excite(nn.Module):
    def __init__(self, in_chn, reduction=4):
        """
        Initializes a Squeeze-and-Excitation (SE) block.
        Args:
            in_chn (int): Number of input channels.
            reduction (int, optional): Reduction ratio for the intermediate channels. Default is 4.
        """
        super().__init__()

        # Calculate the number of channels after reduction
        reduction_chn = in_chn // reduction

        self.se = nn.Sequential(
            nn.AdaptiveAvgPool2d(
                1
            ),  # Input: (batch_size, in_chn, H, W) -> Output: (batch_size, in_chn, 1, 1)
            nn.Flatten(),  # Flatten the output to (batch_size, in_chn)
            nn.Linear(
                in_chn, reduction_chn, bias=False
            ),  ## (batch_size, in_chn) -> (batch_size, reduction_chn)
            nn.ReLU(inplace=True),  # Apply ReLU activation
            nn.Linear(
                reduction_chn, in_chn, bias=False
            ),  # # (batch_size, reduction_chn) -> (batch_size, in_chn)
            Hardsigmoid(),  # Apply Hard Sigmoid activation
            nn.Unflatten(
                1, (in_chn, 1, 1)
            ),  ## (batch_size, in_chn) -> (batch_size, in_chn, 1, 1)
        )
```

In conventional CNNs, convolutional filters apply fixed weights to all input channels, treating each channel uniformly regardless of its contextual relevance. The SE block addresses this limitation by introducing a self-attention mechanism that adaptively scales features channel-wise based on their global significance.

Squeeze-and-Excitation (SE) block works in three main steps:

1. Squeeze (Global Information Embedding)

Instead of treating each feature channel equally, the SE block compresses spatial information from the feature maps into a global descriptor using Global Average Pooling (GAP). This step effectively captures the most important global information across the entire feature map.

2. Excitation (Feature Recalibration)

After obtaining the squeezed feature representation, the network learns channel-wise dependencies using two fully connected (FC) layers with non-linear activations. This step enables the network to model the importance of each channel dynamically based on the input.

3. Reweighting (Feature Recalibration)

After learning the importance of each channel, the recalibrated weights are applied back to the feature maps through channel-wise multiplication. This enhances important features and suppresses less useful ones, improving the model's attention mechanism.

Bottleneck: Inverted Residual Block

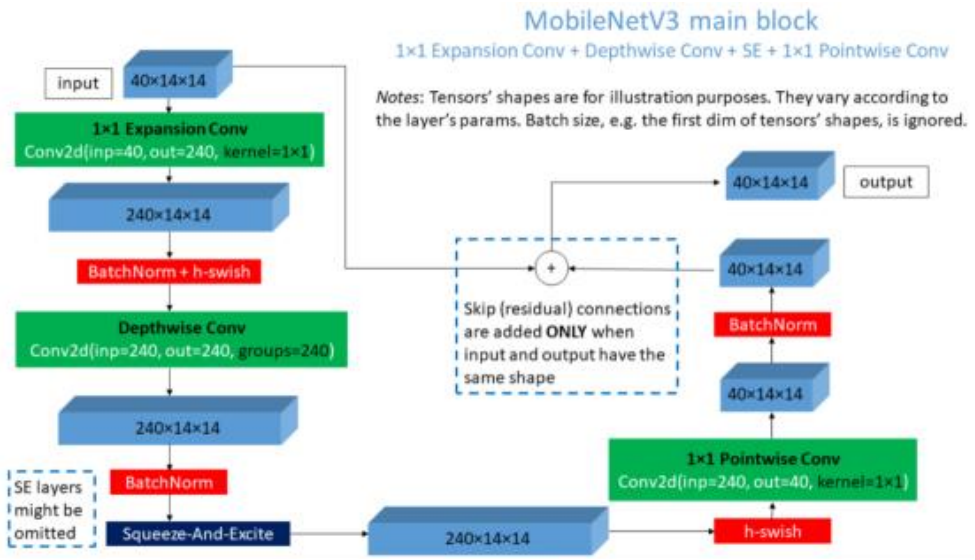
The Bottleneck class in MobileNetV3 is designed to enhance computational efficiency while maintaining expressive feature learning. It incorporates inverted residual connections, depthwise separable convolutions, and an optional Squeeze-and-Excitation (SE) module, all of which contribute to improved performance with minimal computational cost.

The block begins with an expansion layer, where a 1×1 pointwise convolution increases the number of channels, allowing for richer feature extraction. This is followed by a depthwise convolution, which applies spatial filtering independently to each channel, significantly reducing computational overhead. A subsequent 1×1 pointwise convolution compresses the output, ensuring efficient information flow while maintaining spatial integrity.

If enabled (se_flag=True), the SE module recalibrates feature maps by dynamically adjusting channel-wise importance. It first applies Global Average Pooling (GAP) to condense spatial information, followed by two fully connected layers that learn adaptive weights. The

recalibrated important scores are then multiplied elementwise with the original feature maps, emphasizing the most relevant channels.

The final stage is the projection layer, where another 1×1 convolution reduces the channels back to the desired output size. This adheres to the linear bottleneck principle, ensuring that no non-linearity distorts lower-dimensional representations, thus preserving essential feature information. A residual connection is included if the input and output channels are the same and stride is 1, allowing the input to be directly added back to the output. This facilitates stable gradient flow and improves training efficiency.



```
# Determine if a residual connection should be used
# Residual connection is used if stride == 1 and input channels == output channels
self.use_residual = (stride == 1) and (in_chn == out_chn)

# Define the layers of the residual block

self.residual_block_layers = [
    # Expansion convolution: 1x1 convolution to expand the number of channels
    ConvBlock(in_chn, expansion_channel, activation=activation, kernel_size=1),
    # Depthwise convolution: Grouped convolution with kernel_size x kernel_size
    ConvBlock(
        expansion_channel,
        expansion_channel,
        activation=activation,
        group=expansion_channel,
        stride=stride,
        kernel_size=kernel,
    ),  # (batch_size, expansion_channel, H, W) -> (batch_size, expansion_channel, H', W')
]

# Add Squeeze-and-Excitation (SE) block if se_flag is True
if se_flag:

    self.residual_block_layers.extend(
        [
            Squeeze_and_Excite(
                expansion_channel, reduction=se_reduction
            )  # Apply SE to recalibrate channel-wise features
        ]
    )

# Projection convolution: 1x1 convolution to reduce the number of channels back to out_chn
self.residual_block_layers.extend(
    [ConvBlock(expansion_channel, out_chn, activation=None, kernel_size=1)]
)
```

MobileNetV3 Class

The MobileNetV3 class serves as the core architecture, integrating lightweight convolutional blocks, bottleneck layers with inverted residuals, and an optional Squeeze-and-Excitation (SE) mechanism for improved feature recalibration. This implementation supports two configurations—MobileNetV3-Large and MobileNetV3-Small, each optimized for a different balance between computational efficiency and accuracy. Additionally, it introduces a hybrid mode that incorporates traditional feature extraction techniques, such as Sobel filtering and Histogram of Oriented Gradients (HOG), alongside deep learning-based representations.

A dedicated function determines which version of MobileNetV3 to use, defining the corresponding layer configuration for constructing the bottleneck residual blocks. These blocks are structured to first expand the input feature maps into a higher-dimensional space, apply depthwise convolution for spatial filtering, and then use pointwise convolution to compress the features while maintaining their expressiveness. When the SE module is enabled, it further refines the feature maps by reweighting channel importance based on global context.

One of the key highlights of this architecture is its support for hybrid feature learning. In hybrid mode, Sobel filtering captures edge-based structural information, with SobelX and SobelY added as additional input channels, which are then fed directly into MobileNetV3. Simultaneously, HOG features undergo a separate sequence of dense layers before being fused with the MobileNetV3 output in the final classification layer. This fusion enhances the model's ability to capture both handcrafted and deep hierarchical features, improving performance in specific domains where traditional and learned representations complement each other.

```
# Initialize feature extractors for hybrid mode
if self.hybrid:

    self.sobel = (
        SobelFeatureExtractor()
    )  # Outputs 5 channels (RGB+SobelX+SobelY)
    self.hog_layer = HOGLayer()  # HOG feature extractor (Dense layer)

# Architecture configuration
self.layers_config, last_channel = self._get_config(mode)

# Build core MobileNetV3 components
self.features = self._build_features(in_chn)
self.final = self._build_final_layers(last_channel, num_classes)

# Hybrid-specific components
if self.hybrid:
    self.hog_fc = nn.Linear(512, num_classes)  # 256 MobileNet + 256 HOG
```

```
"""Get layer configuration and final channel count"""
if mode == "large":
    return (
        [
            [3, 16, 16, False, ReLU, 1],
            [3, 64, 24, False, ReLU, 2],
            [3, 72, 24, False, ReLU, 1],
            [5, 72, 40, True, ReLU, 2],
            [5, 120, 40, True, ReLU, 1],
            [5, 120, 40, True, ReLU, 1],
            [3, 240, 80, False, self.deep_activation, 2],
            [3, 200, 80, False, self.deep_activation, 1],
            [3, 184, 80, False, self.deep_activation, 1],
            [3, 184, 80, False, self.deep_activation, 1],
            [3, 480, 112, True, self.deep_activation, 1],
            [3, 672, 112, True, self.deep_activation, 1],
            [5, 672, 160, True, self.deep_activation, 2],
            [5, 960, 160, True, self.deep_activation, 1],
            [5, 960, 160, True, self.deep_activation, 1],
        ],
        1280,
    )
else:
    return (
        [
            [3, 16, 16, True, ReLU, 2],
            [3, 72, 24, False, ReLU, 2],
            [3, 88, 24, False, ReLU, 1],
            [5, 96, 40, True, self.deep_activation, 2],
            [5, 240, 40, True, self.deep_activation, 1],
            [5, 240, 40, True, self.deep_activation, 1],
            [5, 120, 48, True, self.deep_activation, 1],
            [5, 144, 48, True, self.deep_activation, 1],
            [5, 288, 96, True, self.deep_activation, 2],
            [5, 576, 96, True, self.deep_activation, 1],
            [5, 576, 96, True, self.deep_activation, 1],
        ],
        1024,
    )  # return 2 thing config list and neuron in the last chanbel
```

```
f _build_features(self, in_chn):
  """Construct initial convolutional blocks"""
  features = [
      ConvBlock(in_chn, 16, activation=Hardswish, kernel_size=3, stride=2)
  ]
  input_channel = 16

  # create the bottleneck block based on the config choose
  for kernel, exp, out, se, act, stride in self.layers_config:
      features.append(
          Bottleneck(
              in_chn=input_channel,
              out_chn=out,
              expansion_channel=exp,
              activation=act,
              kernel=kernel,
              se_flag=se,
              stride=stride,
              se_reduction=self.se_reduction,
          ) # type: ignore
      )
      input_channel = out

  return nn.Sequential(*features)
```

7. Training

The training function implements a deep learning model training pipeline with features such as loss computation, accuracy tracking, validation evaluation, learning rate scheduling, and early stopping. The goal is to optimize model performance efficiently while preventing overfitting.

The function follows a structured approach to train the model over multiple epochs. Initially, the function accepts a PyTorch model, training and validation data loaders, a loss function (criterion), an optimizer, and additional training parameters. A learning rate scheduler (ReduceLROnPlateau) is used to adjust the learning rate based on validation loss. A dictionary (metrics) is initialized to store training and validation loss and accuracy over epochs.

During each epoch, the model is set to training mode, and the training loop iterates over batches of data, computing loss and accuracy for each batch. Gradients are computed and used to update model weights using backpropagation. Training loss and accuracy are averaged over the epoch to track the learning progress.

After completing the training phase of each epoch, the model is evaluated on the validation dataset. Validation loss and accuracy are computed to determine how well the model generalizes. Training and validation metrics are printed after each epoch for performance tracking.

To prevent unnecessary computations and overfitting, early stopping is implemented, halting the training process if validation loss does not improve for a specified number of epochs (earlyStopping). Additionally, the learning rate scheduler dynamically adjusts the learning rate when validation loss stagnates, enhancing training stability and efficiency.

The best model is restored after the training process ends.

8. Evaluations

To assess the performance of the trained model, we implemented a evaluation pipeline consisting of learning curves, confusion matrices, and detailed classification metrics. Below is a summary of the evaluation methods used:

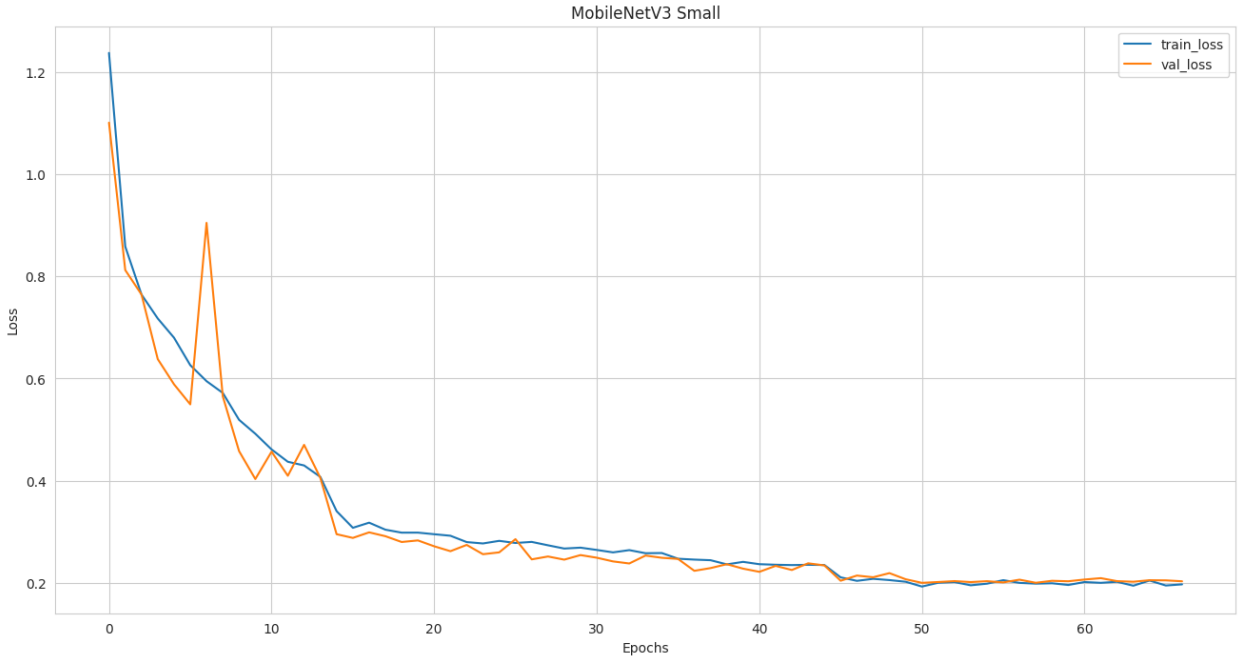
8.1. Learning Curve

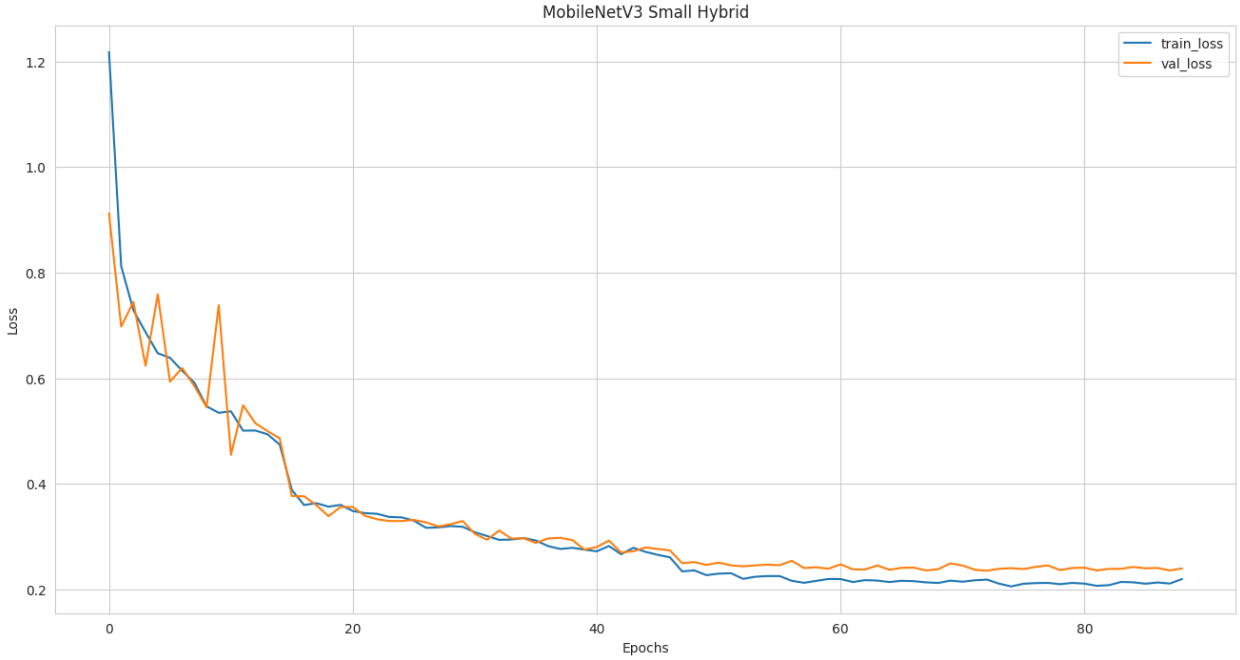
The objective of this evaluation is to determine whether the model is overfitting, underfitting, or learning effectively. A significant gap between training and validation loss could indicate overfitting, whereas a consistently high loss for both suggests underfitting. The plotted curve helps fine-tune the model by adjusting hyperparameters such as learning rate and batch size.

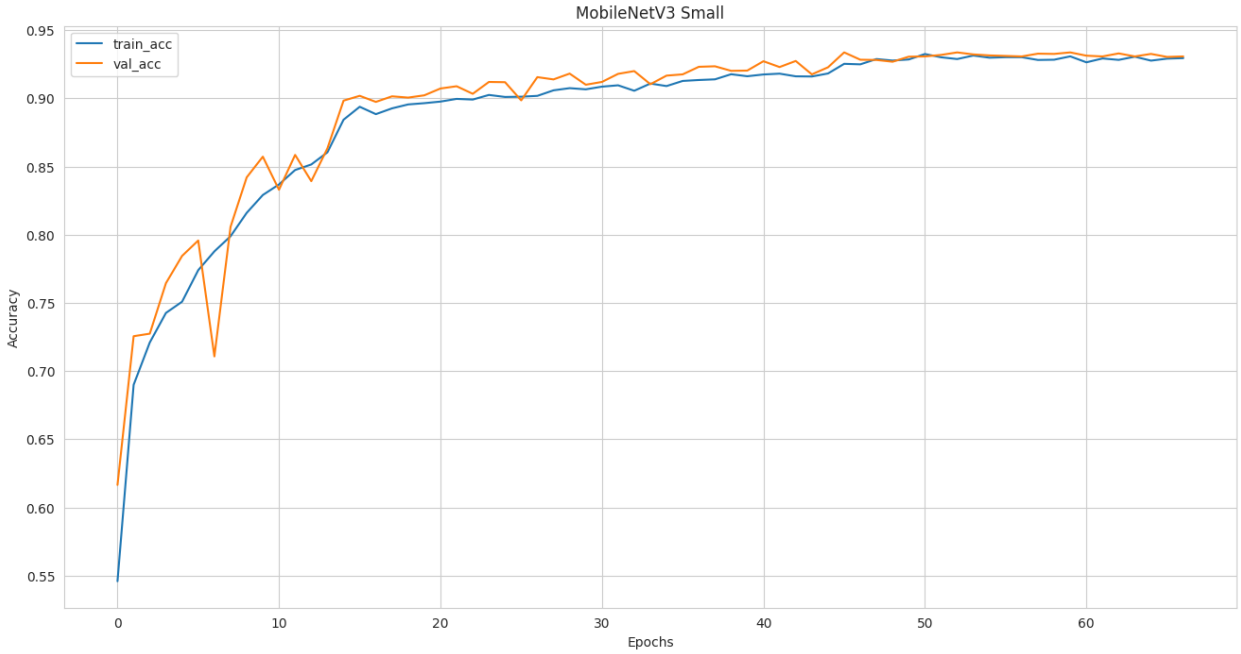
In addition, an accuracy curve is also provided to track model's performance over epochs. Similar to the loss curve, this metric assesses the model's learning efficiency. A steady increase in both training and validation accuracy suggests effective learning, whereas divergence between these curves indicates possible overfitting. One advantage of accuracy curve is that its values are more intuitive than the loss value.

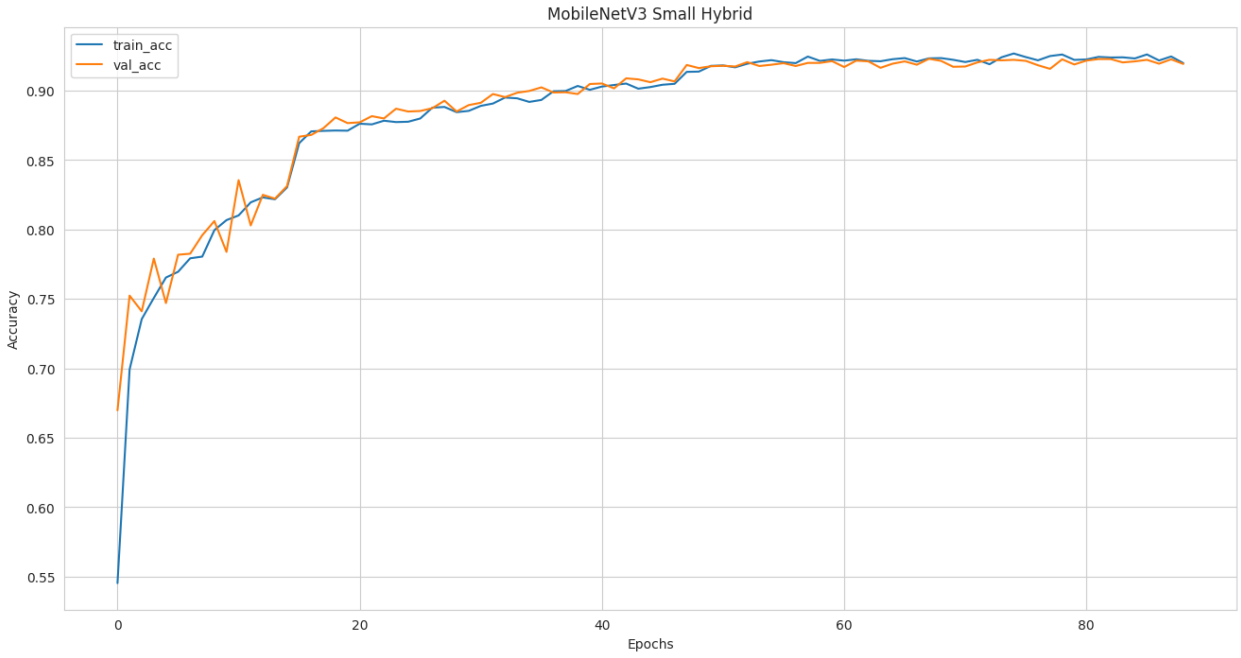
From Figure below, it is clear that the hybrid model converges faster than the classical MobileNetV3. For instance, at epoch 20^{th}, the loss of them is around 0.35 and 0.45 respectively. But the traditional one converges progressively at the later epochs.

For the accuracy curve, we can see there is not much difference in terms of performance. This indicates that the hybrid model has prospective in scenarios where there are enormous amounts of data, and the training time is limited.


MobileNetV3 Small
train_loss
val_loss
Loss
Epochs
1.2
1.0
0.8
0.6
0.4
0.2
0
10
20
30
40
50
60

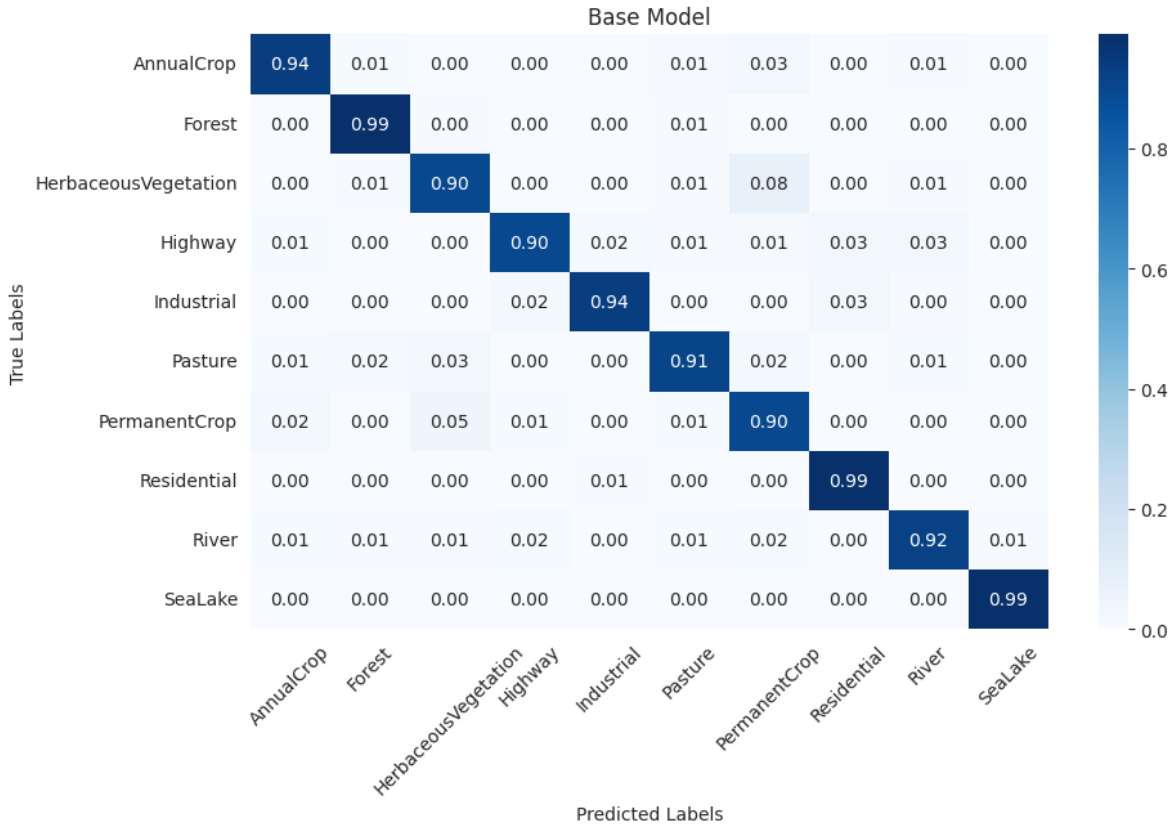

MobileNetV3 Small Hybrid
train_loss
val_loss
Loss
Epochs
1.2
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80

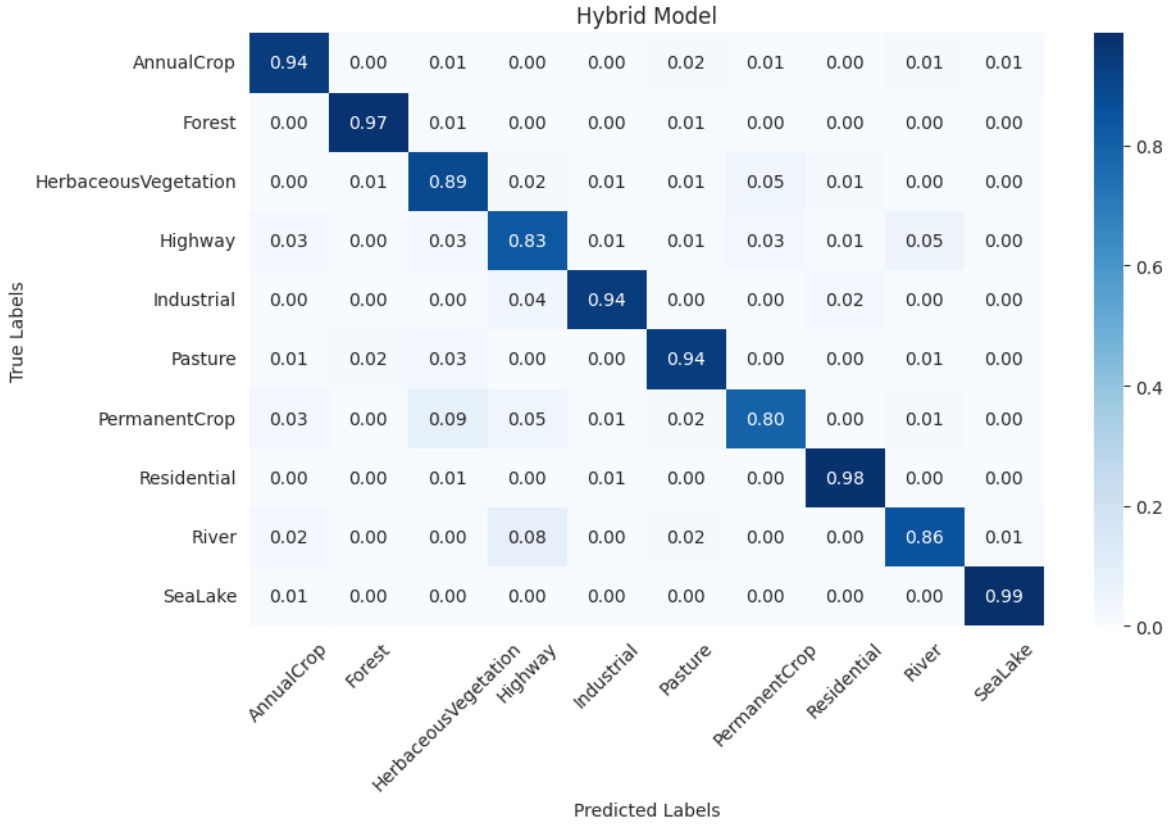




8.2. Confusion Matrix

The confusion matrix highlights correct and incorrect predictions for each class, normalized to aid interpretability. This visualization is particularly useful for understanding class-wise performance, detecting biases, and identifying misclassification trends. This function also relies on class mappings for accurate label interpretation.


Base Model
AnnualCrop 0.94 0.01 0.00 0.00 0.00 0.01 0.03 0.00 0.01 0.00
Forest 0.00 0.99 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.00
HerbaceousVegetation 0.00 0.01 0.90 0.00 0.00 0.01 0.08 0.00 0.01 0.00
Highway 0.01 0.00 0.00 0.90 0.02 0.01 0.01 0.03 0.03 0.00
Industrial 0.00 0.00 0.00 0.02 0.94 0.00 0.00 0.03 0.00 0.00
Pasture 0.01 0.02 0.03 0.00 0.00 0.91 0.02 0.00 0.01 0.00
PermanentCrop 0.02 0.00 0.05 0.01 0.00 0.01 0.90 0.00 0.00 0.00
Residential 0.00 0.00 0.00 0.00 0.01 0.00 0.00 0.99 0.00 0.00
River 0.01 0.01 0.01 0.02 0.00 0.01 0.02 0.00 0.92 0.01
SeaLake 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.99
True Labels
AnnualCrop Forest HerbaceousVegetation Highway Industrial Pasture PermanentCrop Residential River SeaLake
Predicted Labels
0.0 0.2 0.4 0.6 0.8


Hybrid Model
AnnualCrop 0.94 0.00 0.01 0.00 0.00 0.02 0.01 0.00 0.01 0.01
Forest 0.00 0.97 0.01 0.00 0.00 0.01 0.00 0.00 0.00 0.00
HerbaceousVegetation 0.00 0.01 0.89 0.02 0.01 0.01 0.05 0.01 0.00 0.00
Highway 0.03 0.00 0.03 0.83 0.01 0.01 0.03 0.01 0.05 0.00
Industrial 0.00 0.00 0.00 0.04 0.94 0.00 0.00 0.02 0.00 0.00
Pasture 0.01 0.02 0.03 0.00 0.00 0.94 0.00 0.00 0.01 0.00
PermanentCrop 0.03 0.00 0.09 0.05 0.01 0.02 0.80 0.00 0.01 0.00
Residential 0.00 0.00 0.01 0.00 0.01 0.00 0.00 0.98 0.00 0.00
River 0.02 0.00 0.00 0.08 0.00 0.02 0.00 0.00 0.86 0.01
SeaLake 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.99
True Labels
AnnualCrop Forest HerbaceousVegetation Highway Industrial Pasture PermanentCrop Residential River SeaLake
Predicted Labels
0.0 0.2 0.4 0.6 0.8

From the Figure above, we can see that both models do not show any bias toward specific labels, indicating the model is robust and balanced in their predictions. Additionally, this balanced performance highlights the effectiveness of the training process and the potential suitability of these models for real-world deployment.

For comparison, the only defect in hybrid model compared to the traditional one is in label “PermanentCrop”, where the model is twice likely to misclassify to “HerbaceousVegetation” or “Highway”.

8.3. Accuracy, Classification Report, ROC AUC

In this subsection, we are diving into numerically evaluating the model’s performance. The metrics we used are Accuracy, F1-score, precision, recall and ROC-AUC.

Accuracy measures the proportion of correctly predicted instances (both true positives and true negatives) out of the total number of instances. It is a straightforward metric that provides an overall sense of how well the model is performing. However, accuracy can be misleading in imbalanced datasets, where one class significantly outweighs the other, as it may not reflect the model's performance on the minority class.

Precision calculates the proportion of true positive predictions out of all positive predictions made by the model. It is particularly important in scenarios where false positives are costly, such as in spam detection or medical diagnosis. A high precision indicates that the model is reliable when it predicts the positive class.

Recall (or sensitivity) measures the proportion of true positives out of all actual positives in the dataset. It is crucial in situations where a missing positive instance (false negative) has serious consequences, such as in disease screening or fraud detection. A high recall ensures that the model captures most of the relevant instances.

The F1-score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance. It is especially useful when there is an uneven class distribution, as it takes both false positives and false negatives into account. The F1-score is ideal for evaluating models in scenarios where both precision and recall are important, such as in information retrieval or classification tasks with imbalanced data.

The ROC-AUC (Receiver Operating Characteristic - Area Under the Curve) metric evaluates the model's ability to distinguish between classes across different classification thresholds. It plots the true positive rate (recall) against the false positive rate and measures the area under this curve. A high ROC-AUC score indicates that the model has a strong ability to differentiate between classes, making it a robust metric for evaluating performance, especially in binary classification tasks.

Class	Precision	Recall	F1-Score	Support
AnnualCrop	0.95	0.94	0.94	596
Forest	0.97	0.99	0.98	608
HerbaceousVegetation	0.91	0.90	0.91	573
Highway	0.94	0.90	0.92	496
Industrial	0.97	0.94	0.95	501
Pasture	0.92	0.91	0.92	396
PermanentCrop	0.85	0.90	0.87	538
Residential	0.94	0.99	0.97	554
River	0.94	0.92	0.93	529
SeaLake	0.99	0.99	0.99	609
Accuracy			**0.94**	**5400**
Macro Avg	0.94	0.94	0.94	5400
Weighted Avg	0.94	0.94	0.94	5400

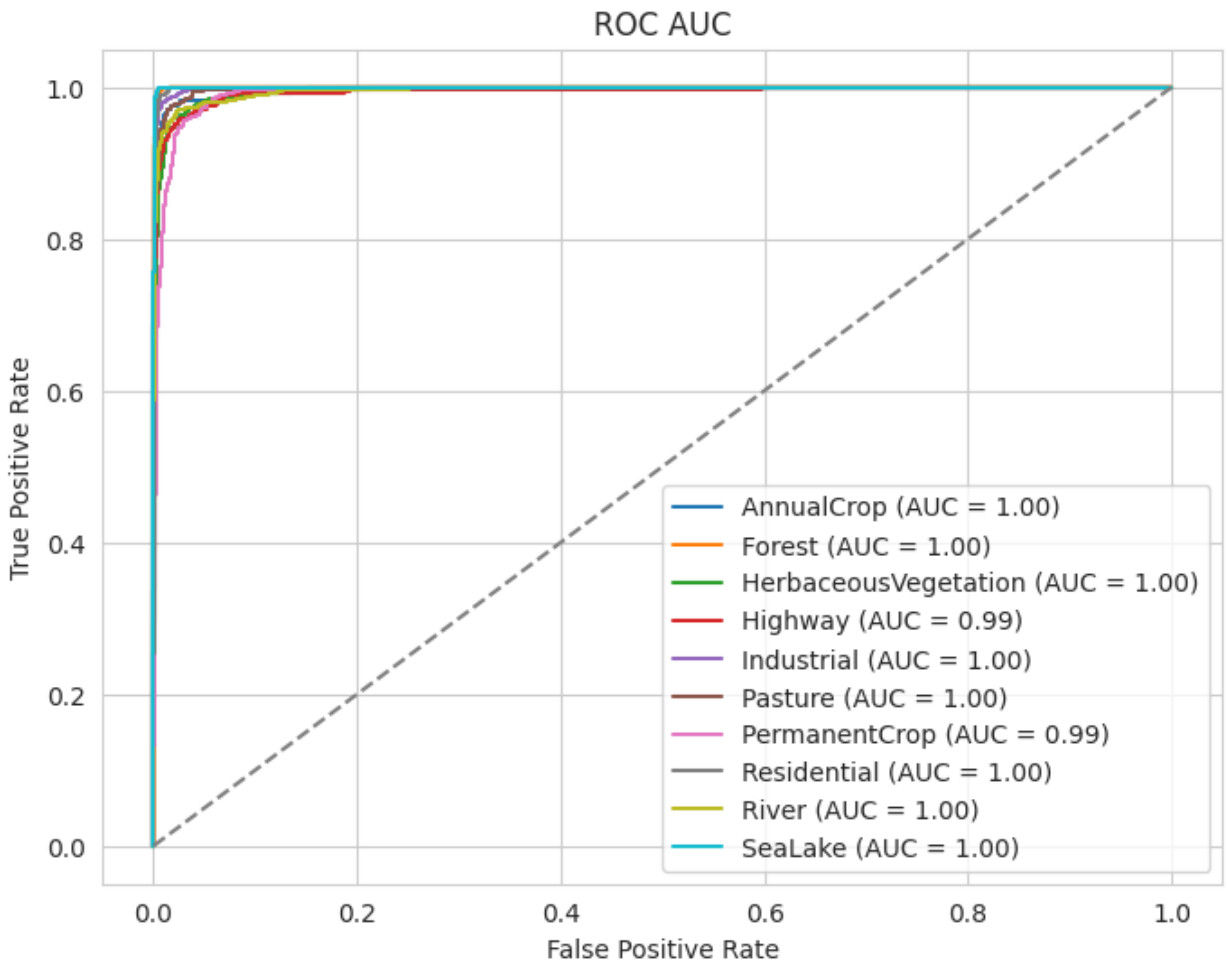
Class	Precision	Recall	F1-Score	Support
AnnualCrop	0.91	0.94	0.93	596
Forest	0.98	0.97	0.97	608
HerbaceousVegetation	0.85	0.89	0.87	573
Highway	0.80	0.83	0.82	496
Industrial	0.97	0.94	0.95	501
Pasture	0.89	0.94	0.91	396
PermanentCrop	0.89	0.80	0.84	538
Residential	0.96	0.98	0.97	554
River	0.91	0.86	0.88	529
SeaLake	0.98	0.99	0.98	609
Accuracy			**0.92**	**5400**
Macro Avg	0.91	0.91	0.91	5400
Weighted Avg	0.92	0.92	0.91	5400

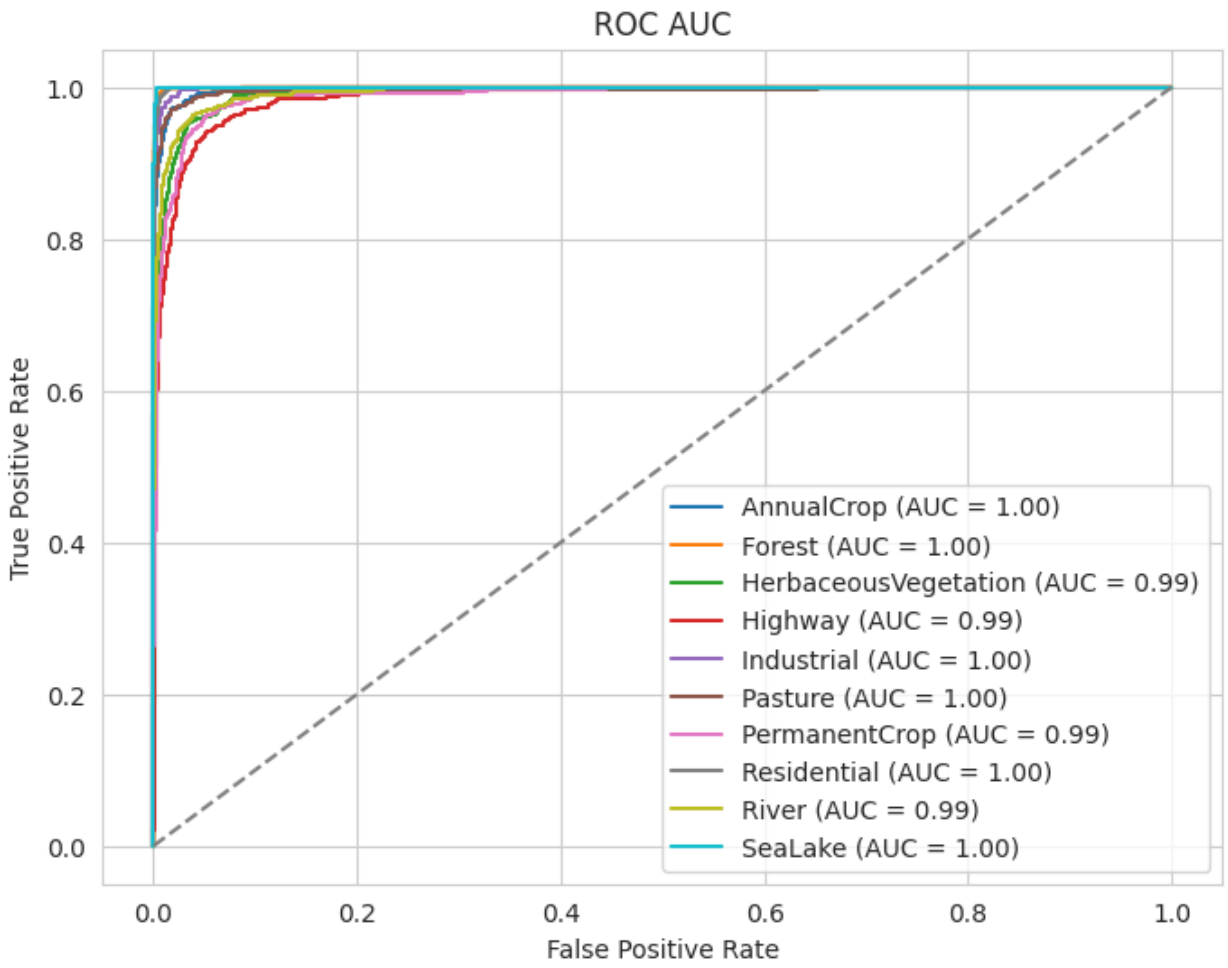
From Table 1, the original model achieves an accuracy of 0.94, with macro avg and weighted avg scores for precision, recall, and F1-score all at 0.94. This indicates strong and balanced performance across all classes. Most classes, such as SeaLake, Forest, and Residential, show high precision and recall (above 0.90), with SeaLake standing out with near-perfect metrics (precision: 0.99, recall: 0.99, F1-score: 0.99). While some classes like PermanentCrop and HerbaceousVegetation have slightly lower performance, they still maintain strong F1-scores (0.87 and 0.91, respectively). The Highway class, though slightly weaker (precision: 0.94, recall: 0.90, F1-score: 0.92), performs reasonably well. Overall, the first table demonstrates that the model generalizes effectively across all classes without significant bias.

In contrast, the hybrid model in Table 2 shows a slight drop in overall performance, with an accuracy of 0.92 and macro avg and weighted avg scores of 0.91 and 0.92, respectively. While many classes, such as SeaLake, Forest, and Residential, continue to perform well, there are noticeable declines in performance for specific classes. For example, the Highway class shows a significant drop in precision (0.80), recall (0.83), and F1-score (0.82), making it the weakest-performing class in this table. Similarly, the PermanentCrop class has lower recall (0.80) compared to the first table (0.90), resulting in a reduced F1-score (0.84 vs. 0.87). The HerbaceousVegetation class also experiences a slight decline in precision (0.85 vs. 0.91) and F1-score (0.87 vs. 0.91). These drops suggest that the model in the second table may struggle with more challenging or ambiguous instances in these categories.

The comparison between the two tables highlights both the strengths and weaknesses of the model. While the first table demonstrates robust generalization across all classes, the second table reveals specific areas for improvement, particularly for the Highway and PermanentCrop classes. The consistent high performance of classes like SeaLake, Forest, and Residential across both tables indicates that the model handles these categories well regardless of the dataset or conditions. However, the degradation in performance for certain classes in the second table suggests that the model may benefit from targeted improvements, such as data augmentation, class rebalancing, or additional training on challenging instances. Overall, the analysis provides valuable insights for refining the model and addressing its weaknesses to achieve more balanced and reliable performance across all classes.

In terms of testing accuracy, it is obvious now that the classical model performs better. It achieves 93.94% compared to its counterpart with just 91.52%. For visualize the performance. We employ ROC-AUC as below:


ROC AUC
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AnnualCrop (AUC = 1.00)
Forest (AUC = 1.00)
HerbaceousVegetation (AUC = 1.00)
Highway (AUC = 0.99)
Industrial (AUC = 1.00)
Pasture (AUC = 1.00)
PermanentCrop (AUC = 0.99)
Residential (AUC = 1.00)
River (AUC = 1.00)
SeaLake (AUC = 1.00)


ROC AUC
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AnnualCrop (AUC = 1.00)
Forest (AUC = 1.00)
HerbaceousVegetation (AUC = 0.99)
Highway (AUC = 0.99)
Industrial (AUC = 1.00)
Pasture (AUC = 1.00)
PermanentCrop (AUC = 0.99)
Residential (AUC = 1.00)
River (AUC = 0.99)
SeaLake (AUC = 1.00)

9. Varying parameters

A. Howard et al. (2021) claimed that the Swish activation improves MobileNet accuracy and reduces training time compared to ReLU activation then they further reduce the inference time by introducing Hard Swish – an approximation which require less computation and successfully reduce 3ms. Since then, a lot of activations have been created aiming to resolve ReLU's problems. In this Section, I will replace Hard Swish with GELU – which has the same value graph as Swish, LeakyReLU – a new activation replacing ReLU, ELU and then Hard ELU – my approximation of ELU, inspired by Hard Swish (Howard *et al.*, 2019).

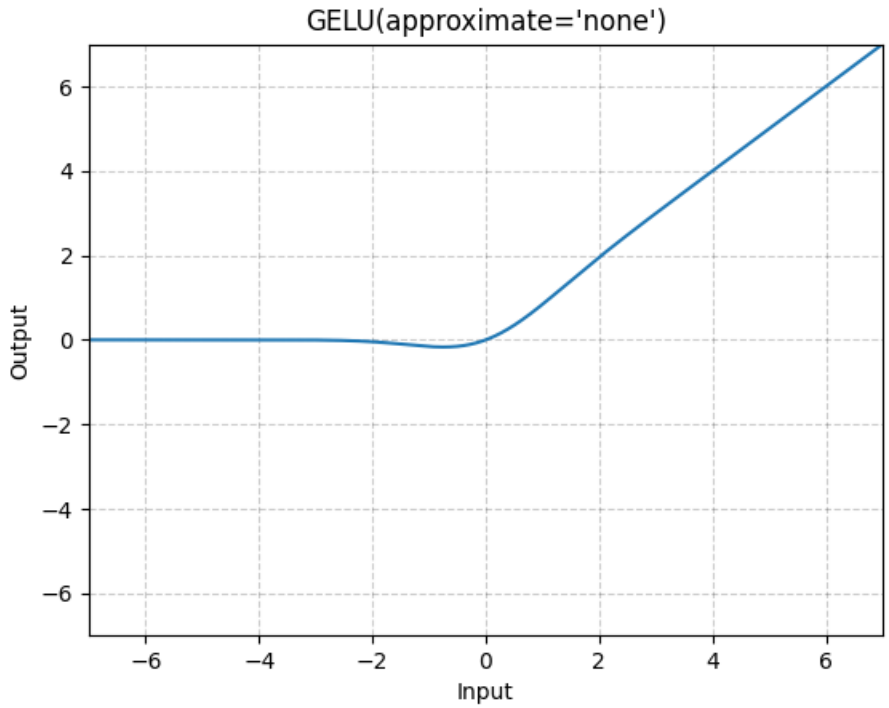
9.1. GELU

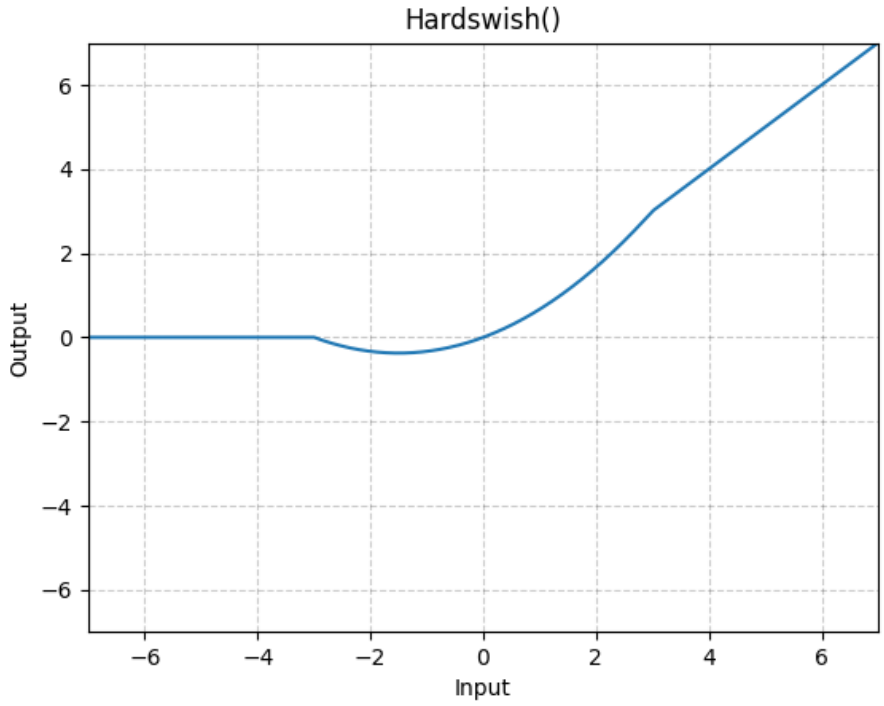
The Gaussian Error Linear Unit (GELU) is an activation function that has gained significant attention in modern deep learning architectures, particularly in transformer-based models like BERT and GPT. Unlike traditional activation functions such as ReLU (Rectified Linear Unit) or sigmoid, GELU incorporates a probabilistic approach by weighting inputs based on their magnitude relative to a Gaussian distribution. This allows GELU to smoothly approximate the behavior of ReLU while introducing non-linearity in a more nuanced way. Mathematically, GELU can be expressed as $x\Phi(x)$, where $\Phi(x)$ is the cumulative distribution function of the standard Gaussian distribution. This formulation enables GELU to combine the benefits of stochastic regularization with deterministic transformations, making it particularly effective in capturing complex patterns in data.

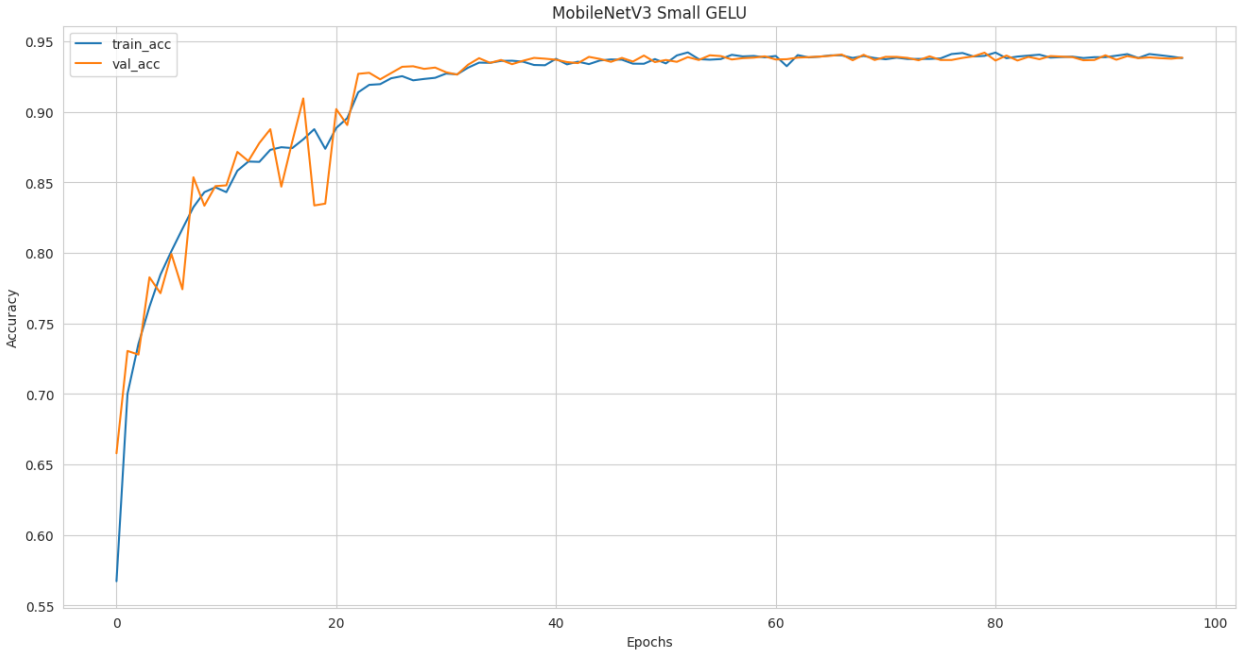
From the value graph in Figure below, we can see the resemblance between 2 activations, indicating that GELU will give similar results.

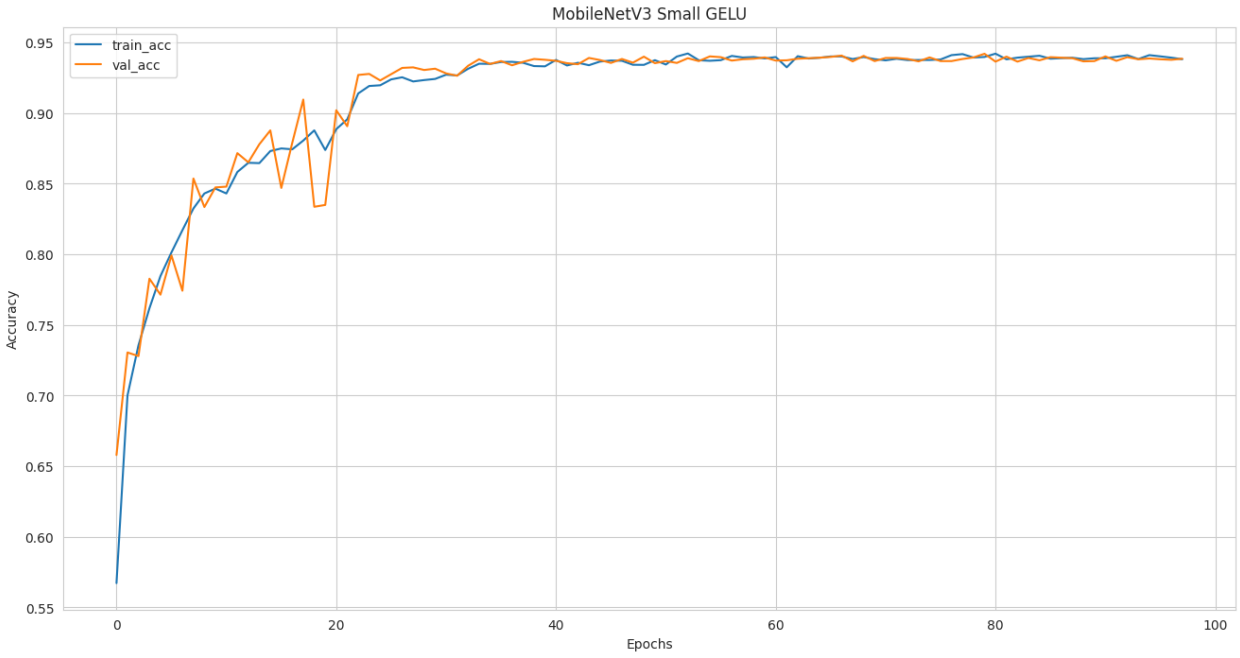
Using this activation, the training time takes a total of 98 epochs, 12 minutes and 43 seconds (7.8 s/epoch) with test accuracy of 95.00%. In contrast, the classical model takes 67 epochs to train, 8min 36s (7.7 s/epoch) with accuracy of 93.94%. As mentioned in the Section Training, we use early stopping, meaning GELU allows models to better fit the data.

Below are the evaluations of GELU based model. They are overall improved compared to Hard Swish.


GELU(approximate='none')
Output
Input
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

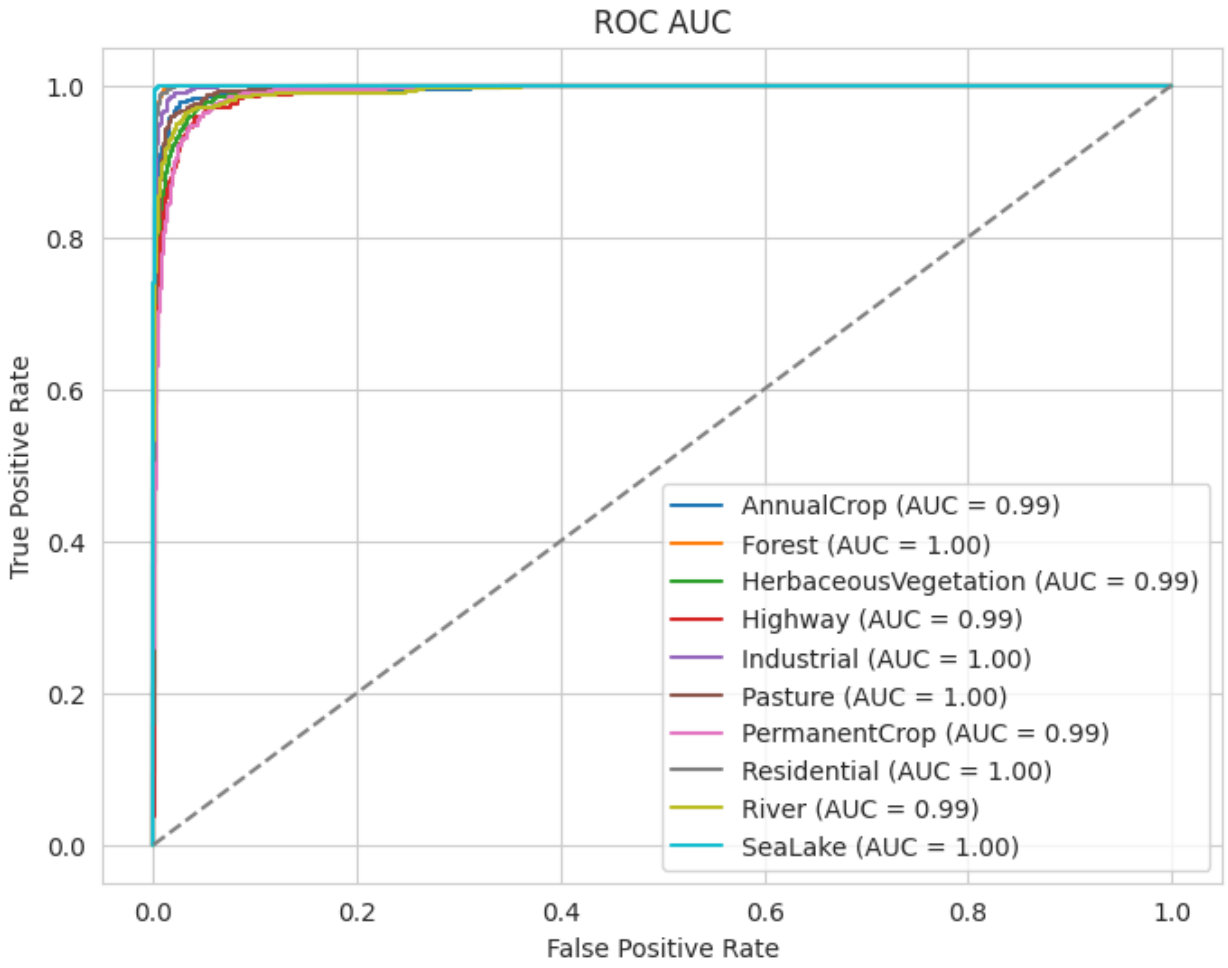

Hardswish()
Output
Input
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6





Class	Precision	Recall	F1-Score	Support
AnnualCrop	0.90	0.93	0.92	596

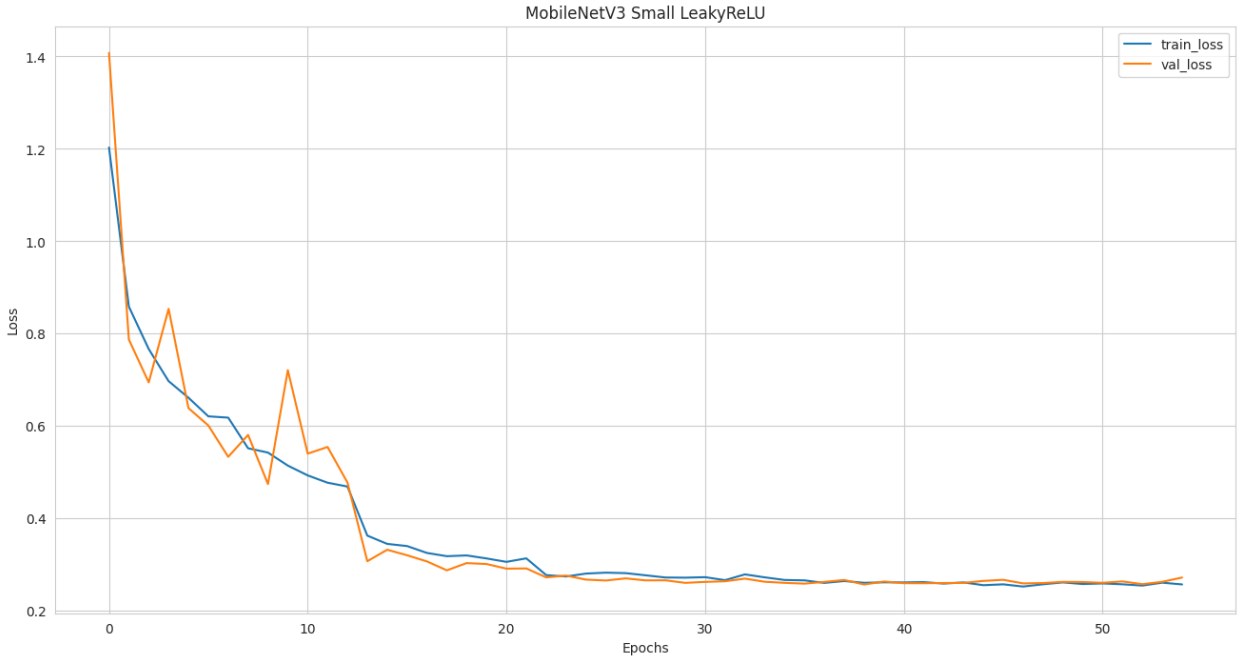
Forest	0.97	0.97	0.97	608
HerbaceousVegetation	0.87	0.90	0.89	573
Highway	0.88	0.84	0.86	496
Industrial	0.96	0.93	0.94	501
Pasture	0.89	0.90	0.90	396
PermanentCrop	0.85	0.86	0.85	538
Residential	0.94	0.98	0.96	554
River	0.93	0.86	0.90	529
SeaLake	0.98	1.00	0.99	609
Accuracy			**0.92**	**5400**
Macro Avg	0.92	0.92	0.92	5400
Weighted Avg	0.92	0.92	0.92	5400

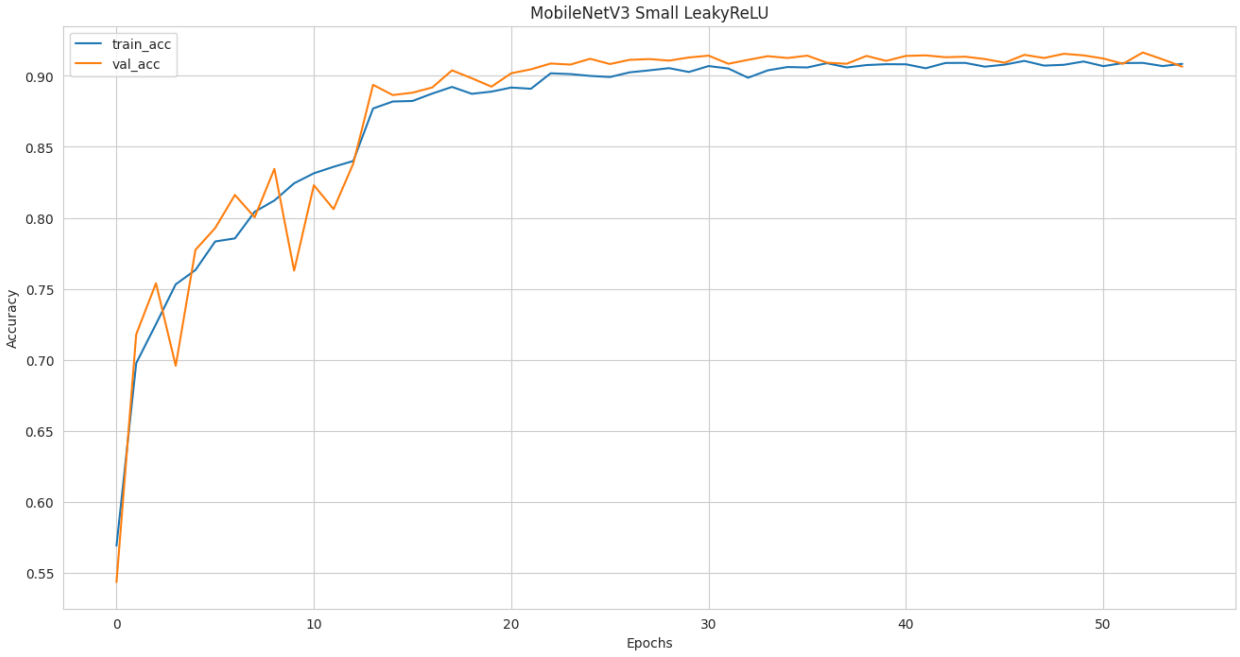


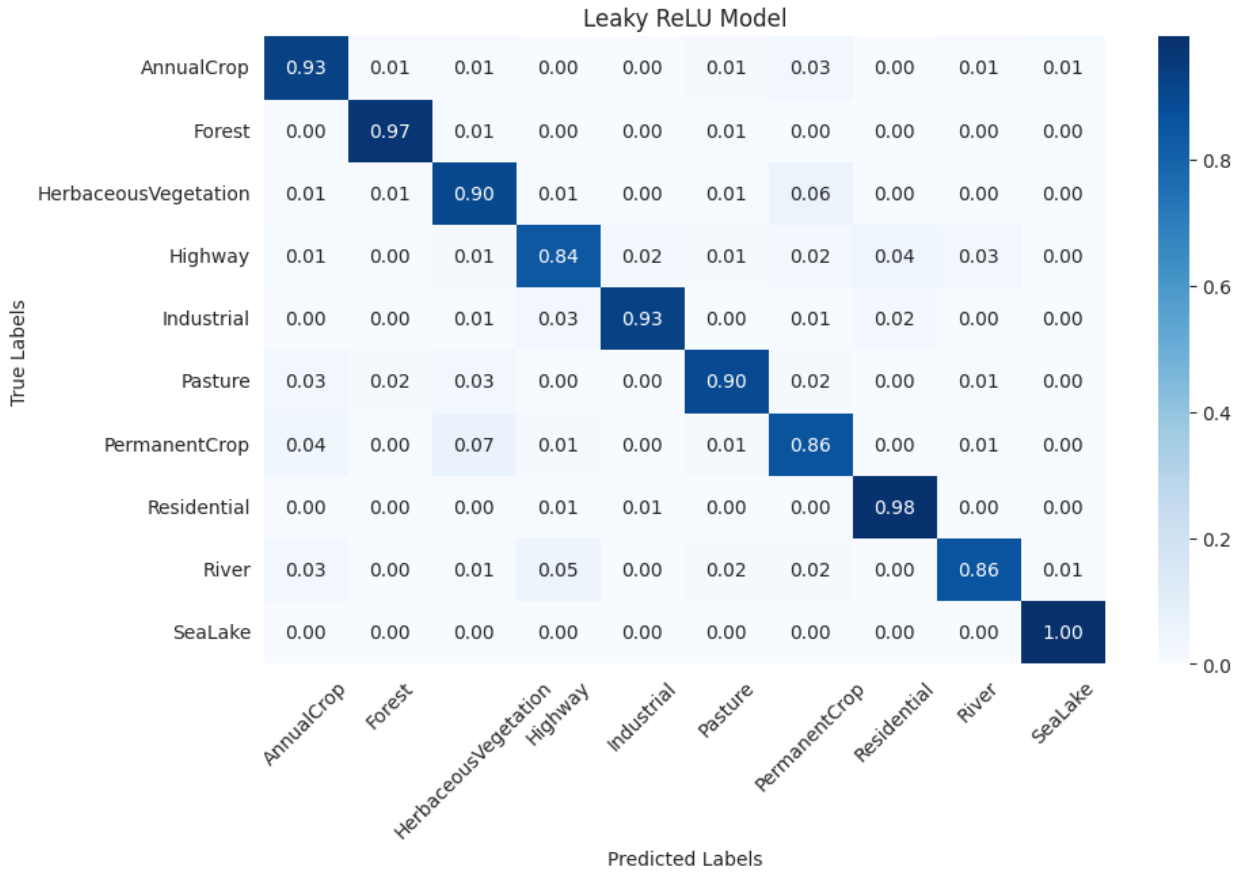
9.2. LeakyReLU

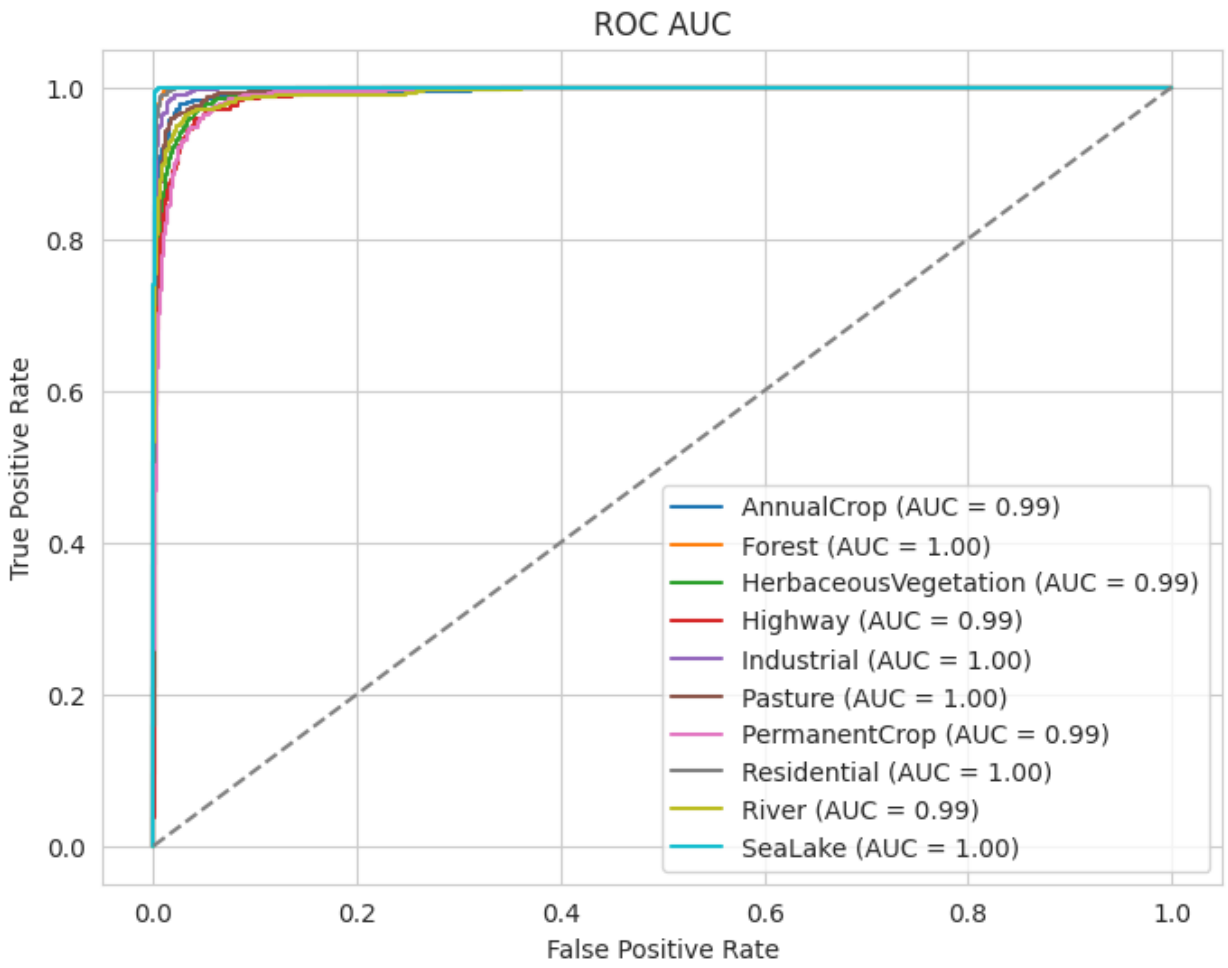
The Leaky Rectified Linear Unit (LeakyReLU) is a variant of the popular ReLU (Rectified Linear Unit) activation function, designed to address one of ReLU's key limitations: the "dying ReLU" problem. Unlike ReLU, which outputs zero for all negative inputs, LeakyReLU allows a small, non-zero gradient for negative values. Mathematically, LeakyReLU is defined as $\max(\alpha x, x)$ where α is a small constant. This ensures that even when the input is negative, the neuron remains active, preventing the network from becoming stuck during training. By introducing this small slope for negative inputs, LeakyReLU helps maintain gradient flow, improving the stability and performance of deep neural networks. It is particularly useful in scenarios where models suffer from sparse gradients or slow convergence, making it a popular choice in architectures like convolutional neural networks (CNNs) and generative adversarial networks (GANs).

Using this activation, the training time takes a total of 55 epochs, 6min 51s (7.5 s/epoch) with test accuracy of 91.96%. Overall, this activation reduces the training time of the model in exchange for accuracy. Interestingly, in Figure below, the model converges much sooner than the classical MobileNet, indicating a new direction for quicker training.


MobileNetV3 Small LeakyReLU
train_loss
val_loss
Loss
Epochs
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
10
20
30
40
50


MobileNetV3 Small LeakyReLU
train_acc
val_acc
Accuracy
Epochs
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
10
20
30
40
50

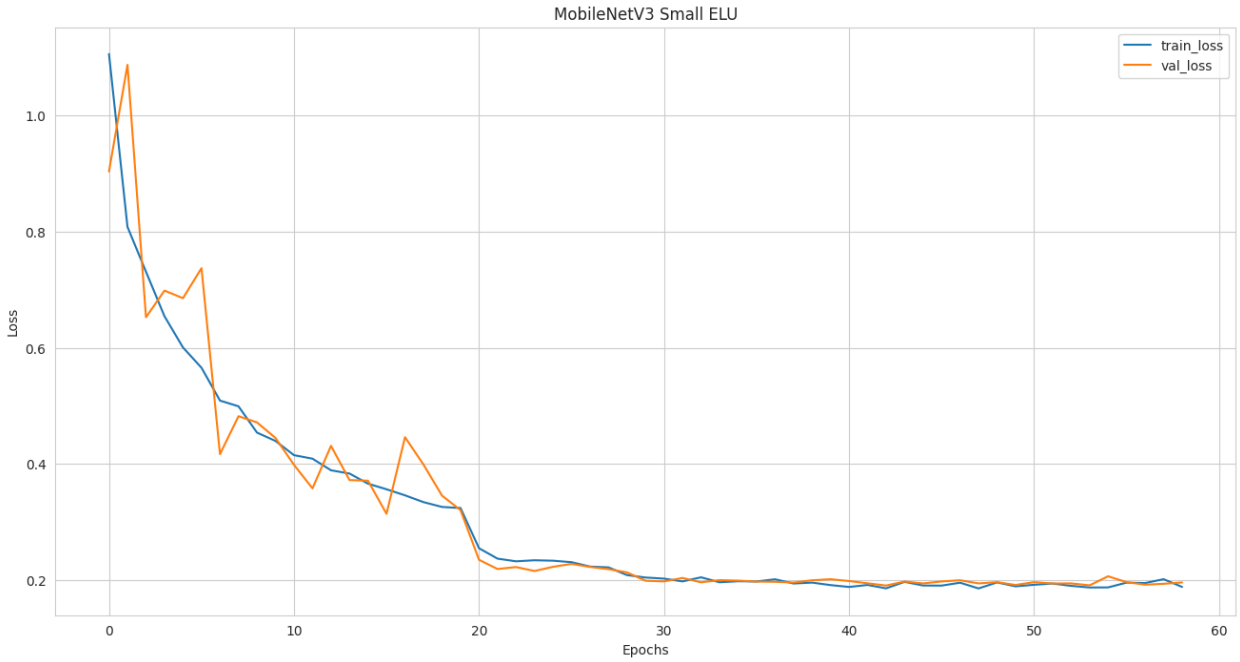

Leaky ReLU Model
AnnualCrop 0.93 0.01 0.01 0.00 0.00 0.01 0.03 0.00 0.01 0.01
Forest 0.00 0.97 0.01 0.00 0.00 0.01 0.00 0.00 0.00 0.00
HerbaceousVegetation 0.01 0.01 0.90 0.01 0.00 0.01 0.06 0.00 0.00 0.00
Highway 0.01 0.00 0.01 0.84 0.02 0.01 0.02 0.04 0.03 0.00
Industrial 0.00 0.00 0.01 0.03 0.93 0.00 0.01 0.02 0.00 0.00
Pasture 0.03 0.02 0.03 0.00 0.00 0.90 0.02 0.00 0.01 0.00
PermanentCrop 0.04 0.00 0.07 0.01 0.00 0.01 0.86 0.00 0.01 0.00
Residential 0.00 0.00 0.00 0.01 0.01 0.00 0.00 0.98 0.00 0.00
River 0.03 0.00 0.01 0.05 0.00 0.02 0.02 0.00 0.86 0.01
SeaLake 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
True Labels
AnnualCrop
Forest
HerbaceousVegetation
Highway
Industrial
Pasture
PermanentCrop
Residential
River
SeaLake
Predicted Labels
0.8
0.6
0.4
0.2
0.0



9.3. ELU

The Exponential Linear Unit (ELU) is an advanced activation function designed to address some of the limitations of traditional activation functions like ReLU (Rectified Linear Unit) and LeakyReLU. ELU introduces exponential behavior for negative inputs, which helps to push the mean activations closer to zero, thereby improving the normalization of the learning process. Mathematically, ELU is defined as: x if $x > 0$ else $\alpha(e^x - 1)$. Where α is a hyperparameter typically set to 1.0. For positive inputs, ELU behaves like ReLU, providing the benefits of linearity and sparsity. However, for negative inputs, ELU outputs a smooth, non-linear curve that helps reduce the vanishing gradient problem and improves learning dynamics. This smooth transition for negative values allows ELU to achieve faster convergence and better generalization in many deep learning tasks. Additionally, ELU's ability to produce negative outputs helps center the data, which can lead to more stable and efficient training. ELU is particularly effective in deep neural networks, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), where it has been shown to outperform ReLU and its variants in certain scenarios.

Using this activation, the training time takes a total of 59 epochs, 7min 29s (7.6 s/epoch) with test accuracy of 93.69%. This is both slightly worse and slower than the classical MobileNet.

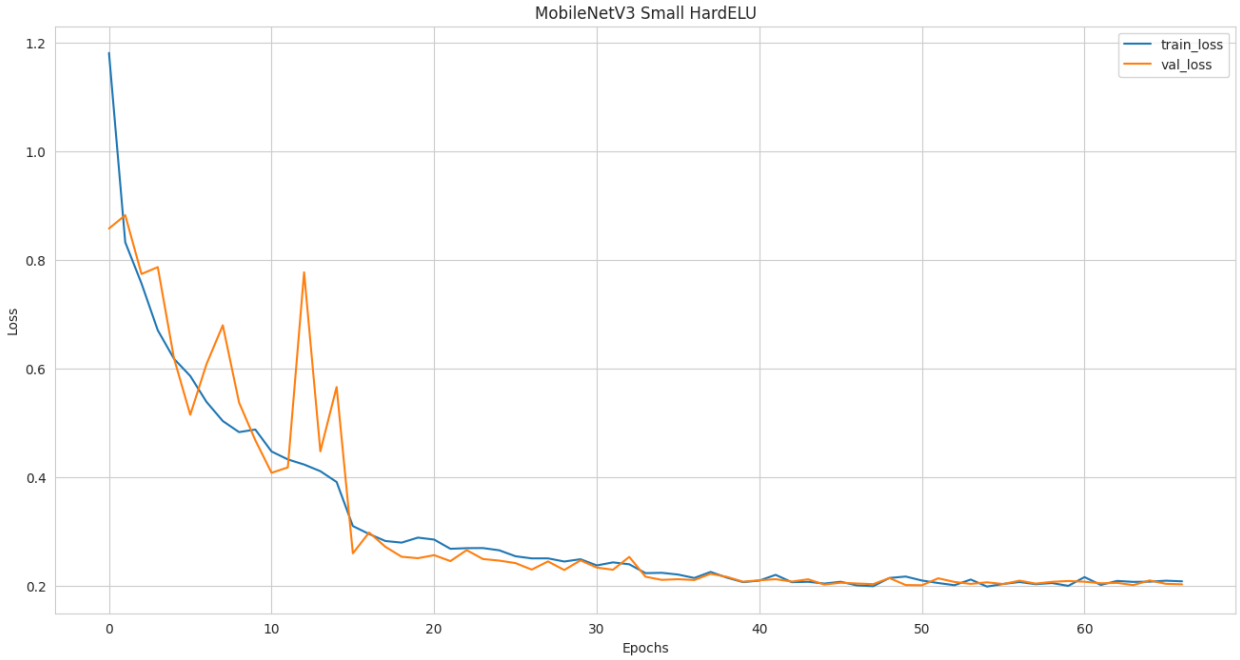


But, from the learning curve above, it is clear that the model is the fastest converging, reaching its peak at only 30 epochs.

9.4. Hard ELU

Inspired by Hard Swish, I approximate the exponential part of ELU. Castro (2015) stated a good approximation for general usage: $e^x \sim \frac{1}{1-x}$. Using this approximation, when x < 0, HardElu = $\alpha \cdot \frac{x}{1-x}$. With accuracy of 93.15%, and 67 epochs to converge, the activation does not show a clear future improvement.

The figure below shows a similar convergent rate compared to ELU.


MobileNetV3 Small HardELU
train_loss
val_loss
Loss
Epochs
1.2
1.0
0.8
0.6
0.4
0.2
0
10
20
30
40
50
60

10. Conclusion

In this project, we explored the architecture of MobileNetV3 and its application to the EuroSAT-RGB dataset for land cover classification. By integrating traditional feature extraction techniques like Sobel filtering and HOG, we demonstrated that hybrid models can achieve competitive performance while reducing training time. Our experiments with activation functions such as GELU, LeakyReLU, and ELU revealed that GELU offers the best balance between accuracy and convergence, outperforming the default Hard Swish activation. However, the hybrid model showed slight degradation in performance for certain classes, such as Highway and PermanentCrop, indicating areas for future improvement.

Future work could focus on optimizing the fusion of traditional and deep learning-based features, exploring additional activation functions, and applying the model to larger and more diverse datasets. Overall, this project highlights the adaptability of MobileNetV3 and its potential for real-world applications in remote sensing and environmental monitoring.

11. References

Castro, C. (2015) “Approximation of Exp(x) Deduced from the Implicit Euler Numerical Solution of First Order Linear Differential Equations,” *viXra* [Preprint]. Available at: https://www.semanticscholar.org/paper/Approximation-of-Exp%28x%29-Deduced-from-the-Implicit-Castro/a8f379a98998501e3b32ea94e5fb85a7a9c1e6b1.

Helber, P. *et al*. (2017) “EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification.”

Howard, A. *et al*. (2019) “Searching for MobileNetV3.”

Howard, A.G. *et al*. (2017) “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.”

Sandler, M. *et al*. (2018) “MobileNetV2: Inverted Residuals and Linear Bottlenecks.”