Leaf Classification for Local Area Network Deployment: Comparison and Pruning of Resnet18 and Attention Modules (SE/CBAM)

Yanze Guo [1], Mingyue Ma [2] and Zhijie Wang [3, *]

[1] School of Mathematics, Harbin Institute of Technology, Harbin, 150006, China

[2] Shanghai Jinqiu A-level Center, Shanghai, 200041, China

[3] Sino-German College, University of Shanghai for Science and Technology, Shanghai, 200093, China

* Corresponding Author Email: 2323030216@st.usst.edu.cn

Abstract. This paper focuses on resource-constrained edge deployment for the 15-class grayscale image classification task using the Swedish Leaf dataset. It systematically compares the standard ResNet18, the channel attention SE (Squeeze-and-Excitation) model, and the channel and spatial attention CBAM model. Structured pruning and distillation fine-tuning are then applied to the best-performing model. The data is divided into train, verification, and test sets, with class balance, each comprising 55, 10, and 10 images per class (totalling 825/150/150 images, respectively). Under a unified training strategy (frozen backbone, AdamW, label smoothing, no Mixup), SE achieved a test accuracy of 0.9133, outperforming the Baseline (0.8733) and CBAM (0.8667). Paper further performed progressive channel pruning on stages 3 and 4 of the SE models (6% per step, over three steps, resulting in total reduction parameters of approximately 23.3% and 19.1% FLOPs). This was followed by low-learning-rate, long-cycle fine-tuning. The final test accuracy was 0.8933, which remains higher than the Baseline despite significant model compression. Using Grad-CAM, the paper quantified interpretability through three metrics: leaf coverage, leaf IoU, and leaf margin IoU, with the following results: CBAM > SE > Baseline, indicating that attention modules focus more on the leaf regions. Experiments also showed that eight-view TTA significantly reduced accuracy in this setup, suggesting TTA can have side effects when mismatched with data augmentation/ normalisation. This paper presents comprehensive training/validation curves, confusion matrices, interpretability metrics, and the pruning-accuracy trade-off, providing a reproducible reference for the engineering implementation of lightweight plant phenotyping classification.

Keywords: Leaf Classification; ResNet18; SE; CBAM; Grad-CAM; Model Pruning; Edge Deployment.

1. Introduction

The automatic classification of plant leaves has significant application value in various fields, such as crop breeding, pest and disease monitoring, and ecological investigation. However, the application in reality often faces two significant challenges: the first is the accuracy of model needs to be high enough to handle the recognition uncertainty caused by changes in lighting and morphological differences in the field environment; another challenge is the limitation of edge device that resources are limited, and it must to control the number of model parameters and computational overhead to meet real-time and low-power requirements. If people only pursue accuracy, models often have complex structures, high computational complexity, and a high cost. If the model is compressed without careful consideration, it may lead to a decrease in accuracy and compromise the reliability of the application.

In recent years, in response to this problem, convolutional neural networks (CNNs) have made significant progress in deep vision applications. Structures represented by ResNet resolve the disappearance of the problem in deep training by introducing a constant short circuit, thereby improving the feature expression ability. Additionally, attention mechanisms adaptively calibrate features in the channel or spatial dimensions, thereby enhancing the model's ability to focus on key

information. Typical methods include channel attention SE (Squeeze and Excitation) and channel spatial attention CBAM (Convolutional Block Attention Module). These methods enhance accuracy while optimising the model structure.

In view of the technological evolution, plant leaf recognition initially relied mainly on artificial feature extraction and shallow classifiers, which were difficult to cope with morphological diversity. With the popularisation of deep learning, lightweight CNNs like ResNet have become mainstream, and attention mechanisms such as SE and CBAM have been gradually introduced to enhance feature expression. In recent years, research has focused on combining model compression, distillation, and interpretability analysis to achieve a balance between accuracy, efficiency, and interpretability.

Based on the above, the article selects ResNet18 as the lightweight baseline. The system compares three models — Baseline, SE, and CBAM — in terms of classification performance and interpretability on the Swedish leaf dataset. Additionally, progressive channel pruning and fine-tuning of distillation are implemented on the best model SE to maintain high accuracy while significantly reducing the number of parameters and computing power. Then, the interpretability of the model's "focus area" is evaluated by combining the Grad CAM heatmap with quantitative indicators such as blade coverage and IoU. This article aims to provide reproducible technical benchmarks and optimisation strategy references for plant leaf classification in resource-constrained environments.

2. Principles and related work

2.1. Residual Learning and ResNet18

ResNet addresses the vanishing gradient and degradation problems in deep networks through explicit identity shortcuts, enabling the training of deeper networks with stronger representational capabilities [1]. The ResNet18 architecture adopted in this work consists of four residual stages (with channel sizes of 64, 128, 256, and 512, respectively), making it suitable for deployment on small-to-medium-scale datasets and edge devices.

2.2. Channel and Spatial Attention

Squeeze-and-Excitation (SE) extracts channel descriptors through global average pooling, followed by two fully connected layers (squeeze and excitation) to learn channel weights, thereby recalibrating features channel-wise [2]. Conversely, CBAM first applies channel attention and then spatial attention, highlighting discriminative regions in a finer-grained manner through a two-stage gating mechanism [3].

2.3. Interpretability and Grad-CAM

Grad-CAM generates class-discriminative heatmaps on feature maps via gradient-weighted pooling, visualising the regions "seen" by the model [4]. For the plant leaf task, people use semantic segmentation masks as approximate ground truth. It defines three metrics: Leaf Coverage (the proportion of hotspots within the activation threshold τ that fall on leaf regions), Leaf IoU, and Leaf Margin IoU, to quantify whether the attention focuses on the main leaf body and its contours.

2.4. Model Compression

Structured channel pruning removes entire columns of convolutional kernels based on channel importance, directly contributing to inference acceleration [5-7]. To mitigate accuracy loss, people employ a gradual incremental pruning approach combined with fine-tuning at a low learning rate. For more stringent requirements, knowledge distilling and early stopping can be further incorporated [1,8].

3. Methods

3.1. Data and Partition

The Swedish Leaf dataset was used, comprising 15 classes with 75 grayscale images per class (resolution approximately 128 × 128), ensuring class balance. The dataset was split into training/validation/test sets at a ratio of 55:10:10 per class, resulting in 825/150/150 images, respectively [9].

3.2. Training Configuration

All models followed the same training protocol: the backbone was frozen (only the classification head and attention gates were fine-tuned), optimized with AdamW (initial learning rate 1e−4, weight decay 1e−2), cross-entropy loss with label smoothing (ε = 0.05); Mixup was not used; batch size = 64; training lasted for a maximum of 80 epochs with early stopping on the validation set. Implementations were based on PyTorch [2-4].

3.3. Models and Pruning

Compared approaches include: Baseline (ResNet18); SE: SE gates inserted into residual blocks of stages 2–4; CBAM: CBAM modules inserted at the exact locations; Pruned-SE: Structured channel pruning applied to stages 3/4 of the SE model using a progressive strategy (6% per step for three steps), with 12 epochs of fine-tuning between steps at a learning rate of 3e−5. The distillation setup (α = 0.70, T = 3.0) was experimentally found not to yield further improvement and was disabled in the final scheme [5-8,10-12].

3.4. Evaluation Metrics and Interpretability Measures

Classification performance was measured by Top-1 accuracy; class-wise behaviour was visualised via row-normalised confusion matrices. For interpretability, based on Grad-CAM heatmaps (threshold τ = 0.5), the mean and 95% confidence intervals of Leaf Coverage, Leaf IoU, and Leaf-Margin IoU were computed. For completeness:

$$\text{LeafCov} = \frac{|C \wedge L|}{|L|}, \text{LeafIoU} = \left| \frac{|\boldsymbol{C} \wedge \boldsymbol{L}|}{|\boldsymbol{C} \vee \boldsymbol{L}| + \boldsymbol{\varepsilon}}, \text{EdgeIoU} = \frac{|C \wedge E|,}{|C \vee E| + \varepsilon} \right. \tag{1}$$

Where CCC is the binarised CAM (τ = 0.5), LLL is the Otsu-segmented leaf mask, and EEE is a dilated edge mask (Sobel/Canny) [8,13,14].

3.5. Core Reproducibility Listings (Minimal Code)

```
class SELayer(nn.Module):
    def __init__(self, ch, r=16):
        super().__init__()
        self.avg = nn.AdaptiveAvgPool2d(1)
        self.fc  = nn.Sequential(
            nn.Linear(ch,    ch//r, bias=False), nn.ReLU(inplace=True),
            nn.Linear(ch//r, ch,    bias=False), nn.Sigmoid()
        )
    def forward(self, x):
        w = self.avg(x).flatten(1)
        w = self.fc(w).view(x.size(0), -1, 1, 1)
        return x * w
```

Fig. 1 SELayer and integration into ResNet-18 (layers 2–4) with a pretrained variant for layer four only.

```
def _insert_se(layer: nn.Sequential):
    for _, block in layer.named_children():
        if hasattr(block, "conv2"):
            block.add_module("se", SELayer(block.conv2.out_channels))

def resnet18_se(num_classes: int = 15) -> nn.Module:
    m = resnet18(weights=None)
    _insert_se(m.layer2); _insert_se(m.layer3); _insert_se(m.layer4)
    return _to_gray(m, num_classes)

# pretrained SE (layer4 only)
def resnet18_se_pre(num_classes: int = 15) -> nn.Module:
    m = resnet18(weights=ResNet18_Weights.IMAGENET1K_V1)
    # grayscale first conv, initialized by channel-averaged RGB weights
    w = m.conv1.weight.data
    m.conv1 = nn.Conv2d(1,64,7,2,3,bias=False)
    m.conv1.weight.data = w.mean(1,keepdim=True)
    _insert_se(m.layer4)
    return _to_gray(m, num_classes)
```

Fig. 2 SELayer and integration into ResNet-18 (layers 2–4) with a pretrained variant for layer four only.

```
class ChannelGate(nn.Module):
    def __init__(self, ch, r=16):
        super().__init__()
        self.avg = nn.AdaptiveAvgPool2d(1)
        self.max = nn.AdaptiveMaxPool2d(1)
        self.fc  = nn.Sequential(
            nn.Conv2d(ch, ch//r,1,bias=False), nn.ReLU(inplace=True),
            nn.Conv2d(ch//r, ch,  1,bias=False)
        )
        self.sig = nn.Sigmoid()
    def forward(self, x):
        w = self.fc(self.avg(x) + self.max(x))
        return x * self.sig(w)

class SpatialGate(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Conv2d(2,1,7,padding=3,bias=False)
        self.sig  = nn.Sigmoid()
    def forward(self, x):
        avg = x.mean(1,keepdim=True)
        mx  = x.max(1,keepdim=True)[0]
        w   = self.conv(torch.cat([avg,mx],1))
        return x * self.sig(w)

def _insert_cbam(layer: nn.Sequential):
    for _, block in layer.named_children():
        if hasattr(block, "conv2"):
            block.add_module("cbam_c", ChannelGate(block.conv2.out_channels))
            block.add_module("cbam_s", SpatialGate())

CBAM_LAYERS = ["layer4"]

def resnet18_cbam(num_classes: int = 15) -> nn.Module:
    m = resnet18(weights=None)
    if "layer2" in CBAM_LAYERS: _insert_cbam(m.layer2)
    if "layer3" in CBAM_LAYERS: _insert_cbam(m.layer3)
    if "layer4" in CBAM_LAYERS: _insert_cbam(m.layer4)
    return _to_gray(m, num_classes)


def resnet18_cbam_pre(num_classes: int = 15) -> nn.Module:
    m = resnet18(weights=ResNet18_Weights.IMAGENET1K_V1)
    w = m.conv1.weight.data
    m.conv1 = nn.Conv2d(1,64,7,2,3,bias=False)
    m.conv1.weight.data = w.mean(1,keepdim=True)
    if "layer4" in CBAM_LAYERS: _insert_cbam(m.layer4)
    return _to_gray(m, num_classes)
```

Fig. 3 CBAM channel & spatial gates, insertion helper, and ResNet-18 wrappers (random/pretrained).

```python
# Build loaders and transforms (grayscale 128×128).
def _tfm_train():
    return transforms.Compose([
        transforms.RandomResizedCrop(128, scale=(0.8, 1.2)),
        transforms.RandomHorizontalFlip(),
        transforms.ColorJitter(0.2, 0.2, 0.2, 0.1),
        transforms.RandAugment(num_ops=2, magnitude=9),
        transforms.Grayscale(num_output_channels=1),
        transforms.ToTensor(),
        transforms.Normalize([0.5], [0.5]),
        transforms.RandomErasing(p=0.4, scale=(0.02, 0.2)),
    ])

def _tfm_eval():
    return transforms.Compose([
        transforms.Resize(128),
        transforms.CenterCrop(128),
        transforms.Grayscale(num_output_channels=1),
        transforms.ToTensor(),
        transforms.Normalize([0.5], [0.5]),
    ])

# DataLoader
def build_loaders(root, batch=64, workers=0):
    root = Path(root)
    if not (root / 'train').exists():
        raise FileNotFoundError(f'{root}/train not found — run quick_split.py first')

    tr = datasets.ImageFolder(root / 'train', _tfm_train())
    va = datasets.ImageFolder(root / 'val',   _tfm_eval())
    te = datasets.ImageFolder(root / 'test',  _tfm_eval())

    return {
        'train': DataLoader(tr, batch_size=batch, shuffle=True,  num_workers=workers, pin_memory=True),
        'val'  : DataLoader(va, batch_size=batch, shuffle=False, num_workers=workers, pin_memory=True),
        'test' : DataLoader(te, batch_size=batch, shuffle=False, num_workers=workers, pin_memory=True),
    }
```

Fig. 4 Data loaders and transforms (grayscale 128×128) for train/val/test.

```python
# Unified epoch routine with optional Mixup and freezing:
def run_one_epoch(net, loader, crit, opt=None, freeze=False, mixup_alpha=0.0):
    train = opt is not None
    net.train() if train else net.eval()
    if train and freeze:
        for n,p in net.named_parameters():
            p.requires_grad = n.startswith(("layer3","layer4","fc"))
    else:
        for p in net.parameters(): p.requires_grad=True
    meter=MetricMeter()
    for x,y in loader:
        x,y = x.cuda(), y.cuda()
        if train and mixup_alpha>0:
            xmix,ymix = mixup_data(x,y,mixup_alpha)
            logits = net(xmix)
            loss   = mixup_criterion(crit,logits,ymix)
        else:
            logits = net(x)
            loss   = crit(logits,y)
        if train:
            opt.zero_grad(); loss.backward(); opt.step()
        meter.add(logits,y,loss.item()*y.size(0))
    return meter.acc, meter.avg_loss
```

Fig. 5 Unified training epoch with optional Mixup and early-layer freezing.

```python
class ResNet18Flex(nn.Module):
    def __init__(self, cfg: Dict, num_classes: int, in_ch: int = 1):
        super().__init__()
        self.conv1 = nn.Conv2d(in_ch, cfg['stem_out'], 7, stride=2, padding=3, bias=False
        self.bn1   = nn.BatchNorm2d(cfg['stem_out'])
        self.relu  = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(3, stride=2, padding=1)
        self.layer1 = self._make_layer(cfg['layer1'])
        self.layer2 = self._make_layer(cfg['layer2'])
        self.layer3 = self._make_layer(cfg['layer3'])
        self.layer4 = self._make_layer(cfg['layer4'])
        self.avgpool = nn.AdaptiveAvgPool2d((1,1))
        self.fc = nn.Linear(cfg['fc_in'], num_classes)

    def _make_layer(self, blocks_cfg: List[Dict]) -> nn.Sequential:
        mods = []
        for b in blocks_cfg:
            mods.append(BasicBlockFlex(
                in_c=b['in_c'], mid_c=b['mid_c'], out_c=b['out_c'], stride=b['stride'],
                se_pos=b['se_pos'], se_shapes=b['se_shapes']
            ))
        return nn.Sequential(*mods)

    def forward(self, x):
        x = self.relu(self.bn1(self.conv1(x)))
        x = self.maxpool(x)
        x = self.layer1(x); x = self.layer2(x); x = self.layer3(x); x = self.layer4(x)
        x = self.avgpool(x); x = torch.flatten(x,1)
        return self.fc(x)
```

```python
def infer_cfg_from_sd(sd: Dict[str, torch.Tensor]) -> Dict:
    stem_out = sd['conv1.weight'].shape[0]
    layers = {}
    for lname, default_stride in [('layer1',1), ('layer2',2), ('layer3',2), ('layer4',2)]:
        blocks = []
        for b in [0,1]:
            kk = _block_keys(lname, b)
            in_c  = sd[kk['conv1']].shape[1]
            mid_c = sd[kk['conv1']].shape[0]
            out_c = sd[kk['conv2']].shape[0]
            stride = default_stride if b == 0 else 1


            se_pos  = 'none'
            se_shape_tuple = None
            has_se = (kk['se_w0'] in sd) and (kk['se_w1'] in sd)
            if has_se:
                w0 = sd[kk['se_w0']]  # [r_in, C]
                w1 = sd[kk['se_w1']]  # [C, r_in]
                C  = w0.shape[1]
                r_in  = w0.shape[0]
                r_out = w1.shape[0]

                if C == mid_c:
                    se_pos = 'conv1'
                    se_shape_tuple = (C, r_in, r_out)
                elif C == out_c:
                    se_pos = 'conv2'
                    se_shape_tuple = (C, r_in, r_out)
                else:

                    se_pos = 'none'
                    se_shape_tuple = None

            blocks.append(dict(
                in_c=in_c, mid_c=mid_c, out_c=out_c, stride=stride,
                se_pos=se_pos, se_shapes=se_shape_tuple
            ))
        layers[lname] = blocks

    cfg = dict(
        stem_out=stem_out,
        layer1=layers['layer1'],
        layer2=layers['layer2'],
        layer3=layers['layer3'],
        layer4=layers['layer4'],
        fc_in=sd['fc.weight'].shape[1]
    )
    return cfg
```

Fig. 6 Pruning-aware reconstruction from a pruned state_dict.

```python
def build_model_from_sd(sd: Dict[str, torch.Tensor], num_classes: int) -> nn.Module:
    cfg = infer_cfg_from_sd(sd)
    return ResNet18Flex(cfg, num_classes, in_ch=1)

@torch.no_grad()
```

Fig. 7 Pruning-aware reconstruction from a pruned state_dict.

```python
def build_views(x, tta):
    if tta == 'none':
        return [x]
    if tta == 'hflip':
        return [x, torch.flip(x, dims=[3])]
    if tta == 'hvflip':
        return [x, torch.flip(x, dims=[3]), torch.flip(x, dims=[2])]
    if tta == 'rot4':
        return [x,
                torch.rot90(x, 1, [2,3]),
                torch.rot90(x, 2, [2,3]),
                torch.rot90(x, 3, [2,3])]
    raise ValueError(f'Unknown TTA mode: {tta}')

@torch.no_grad()
def predict_tta(net, x, tta):
    views = build_views(x, tta)
    zs = [net(v) for v in views]
    return torch.stack(zs, dim=0).mean(0)
```

Fig. 8 Test-time augmentation (TTA): view builder and logits averaging.

These self-contained excerpts document the exact logic used in our pipeline and can be dropped into a minimal reproduction. Helper symbols, such as _to_gray, _tfm_train/_tfm_eval, infer_cfg_from_sd, and ResNet18Flex, are standard utilities defined in the project (Figs. 1-8).

4. Experimental results

4.1. Training and Validation Curves

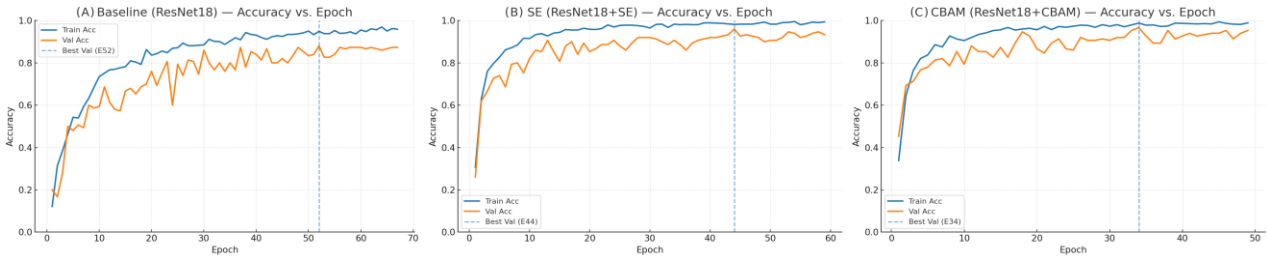

Fig. 9 Training and validation accuracy curves of Baseline (ResNet18), SE, and CBAM.

Figure 9 presents the training and validation accuracy curves of the Baseline model (ResNet18) and the models enhanced with SE and CBAM modules. Compared with the Baseline, both attention-based models converge faster and achieve higher peak performance. Specifically, the Baseline reaches its best validation accuracy of approximately 0.88 at epoch 52, with a test accuracy of 0.8733. The SE model achieves its highest validation accuracy of about 0.96 at epoch 44 and a test accuracy of 0.9133. The CBAM model achieves a similar peak validation accuracy of 0.96 at epoch 34; however, its test accuracy drops to 0.8667. Overall, the SE model improves test accuracy by around 4.0 percentage points compared to the Baseline. In contrast, the CBAM model exhibits a noticeable gap between validation and test performance, indicating weaker generalisation in this setting.

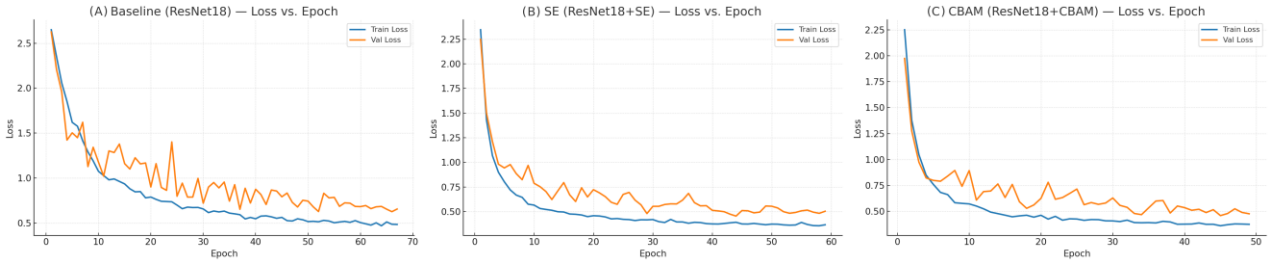

Fig. 10 Training and validation loss curves of the three models.

Figure 10 illustrates that the training and validation losses of all three models decrease steadily with iterations. The attention-based models (SE and CBAM) achieve lower validation losses in the

mid-to-late stages, which, combined with the higher validation accuracies shown in Figure 1, indicate that the attention mechanisms extract more discriminative features. Meanwhile, the difference between training and validation curves remains within a reasonable range, and no significant signs of overfitting are observed, consistent with the validation trends in Figure 9.

4.2. Overall Model Performance and Confusion Matrices

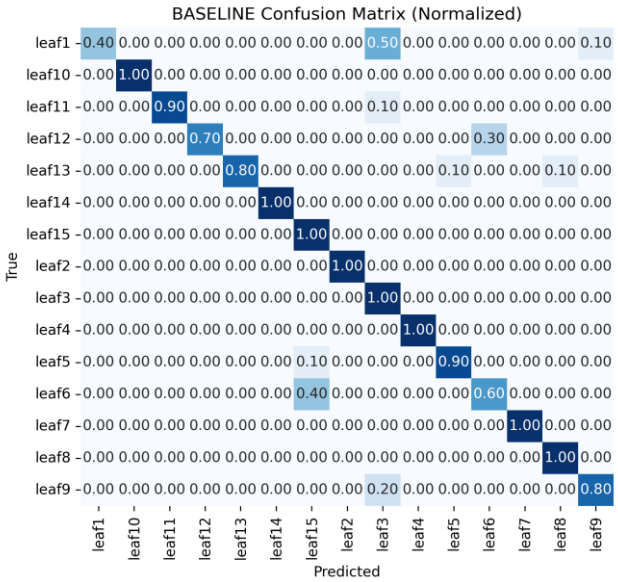

Fig. 11 Normalised confusion matrix of the Baseline model (test set, N = 150)[8][11]

As shown in Figure 11, the Baseline (ResNet-18) already yields a strong main diagonal, indicating stable recognition for most species. However, several localised confusions remain—for example, leaf1 → leaf2 (e.g., a noticeable off-diagonal mass around 0.50), leaf6 → leaf2 (≈0.40), and leaf12 → leaf14 (≈0.30). These errors are consistent with classes that share similar silhouettes or venation patterns.

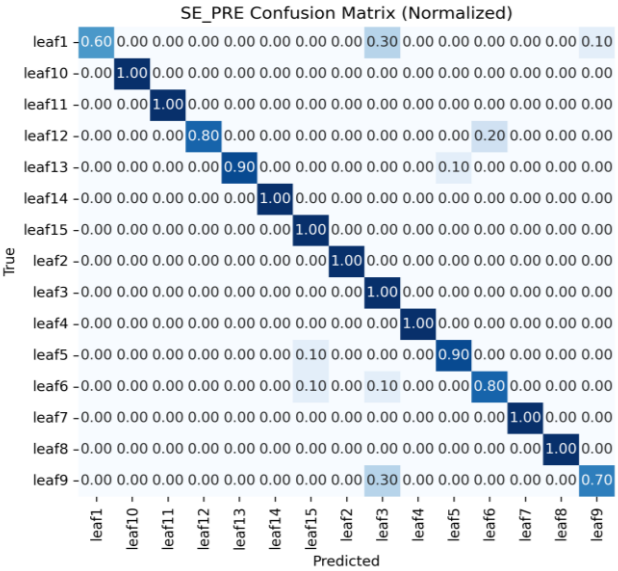

Fig. 12 Normalised confusion matrix of the SE model (test set, N = 150)

As shown in Figure 12, the SE variant further concentrates probability mass on the diagonal. The previously noted pairs (e.g., leaf1↔leaf2, leaf6↔leaf2, leaf12↔leaf14) show reduced off-diagonal activations, and many classes reach near-perfect (≈1.00) per-class accuracy on the test set. Residual weaknesses are mainly found in a few challenging courses, such as leaf1 and leaf9, where small off-diagonal entries remain.

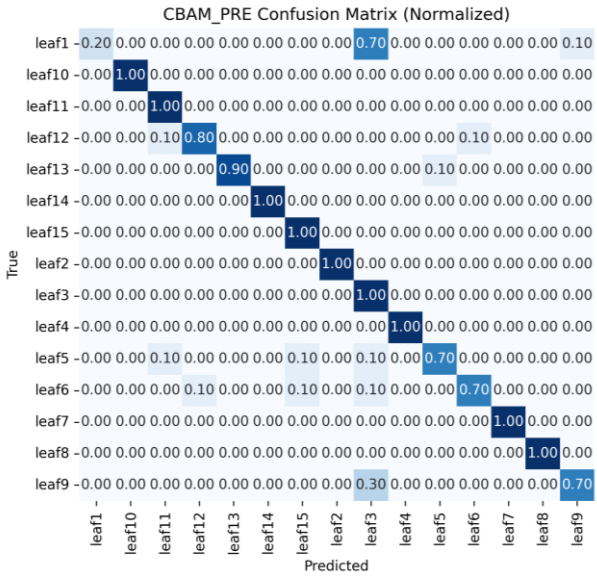

Fig. 13 Normalised confusion matrix of the CBAM model (test set, N = 150)

As shown in Figure 13, CBAM also produces compact diagonal and overall good discrimination, but slightly larger spillovers reappear on a few challenging classes (notably leaf1 and leaf9) compared with SE. This aligns with our aggregate metrics: CBAM has competitive validation curves and stronger explainability scores, yet its test accuracy lags SE, suggesting reduced generalisation on this split.

Takeaway. Attention mechanisms improve inter-class discrimination over the Baseline (tighter diagonals, fewer error blocks). Among them, SE provides the most consistent test-set gains across classes, which is why people adopt SE as the backbone for compression and deployment in subsequent experiments.

Table 1. Summary of validation and test accuracies of the three models and the pruned SE model

Model	Validation Accuracy	Test Accuracy	Notes
Baseline(ResNet18)	0.8800(best)	0.8733	Frozen backbone, E ≤ 80, early stopping
SE	0.9600	0.9133	Channel attention
CBAM	0.9667	0.8667	Channel + spatial attention
Pruned-SE	0.9533	0.8933	Stage 3/4 progressive pruning 6% × 3 + fine-tuning

As shown in Table 1, under the unified training strategy, the SE model achieves validation and test accuracies of 0.9600 and 0.9133, representing the best test performance among the three models. The Baseline ranks second (0.8800/0.8733), while CBAM attains a higher validation accuracy (0.9667) but a lower test accuracy of 0.8667. On the test set, SE improves by +0.0400 compared with the Baseline, whereas CBAM slightly decreases by −0.0066. This observation is consistent with Table 1, suggesting that attention does not necessarily guarantee better generalisation, and that channel recalibration in SE proves to be more robust for this task.

4.3. Quantitative Explainability Analysis (τ = 0.5)

Table 2. Grad-CAM quantitative results (mean ± 95% CI, test set N = 150)

Model	Leaf Coverage	Leaf IoU	Leaf-Edge IoU
Baseline	0.3093±0.0330	0.2842±0.0320	0.1387±0.0102
SE	0.3816±0.0252	0.3212±0.0238	0.1682±0.0143
CBAM	0.4262±0.0265	0.3482±0.0242	0.1726±0.0113

As shown in Table 2, the three explainability metrics follow a consistent ranking: CBAM > SE > Baseline. Specifically, for leaf coverage, the Baseline, SE, and CBAM models achieve 0.3093 ±

0.0330, 0.3816 ± 0.0252, and 0.4262 ± 0.0265, respectively. For leaf IoU, the results are 0.2842 ± 0.0320, 0.3212 ± 0.0238, and 0.3482 ± 0.0242. For leaf-edge IoU, the corresponding values are 0.1387 ± 0.0102, 0.1682 ± 0.0143, and 0.1726 ± 0.0113. These results indicate that attention modules significantly enhance the model's focus on both the leaf body and leaf edges. However, although CBAM outperforms SE and the Baseline in explainability metrics, its test accuracy does not surpass SE (see Table 1). This suggests that "better localised attention" does not necessarily translate into "higher generalisation accuracy" in this task.

4.4. Pruning–Accuracy Trade-off

As shown in Table 3, by applying progressive channel pruning (6% × 3 steps) to stages 3 and 4 of the SE models, the parameter count is reduced from 11,243,471 to 8,554,311 (↓a decrease of 23.3%), while FLOPs decrease from 569,146,895 to 460,703,399 (↓a reduction of 19.1%). Meanwhile, the test accuracy remains stable at 0.8933, only slightly lower than the unpruned SE model (0.9133) and still higher than the Baseline without attention (0.8733). These results demonstrate that, under lightweight design objectives, redundancy in SE can be effectively exploited through mild progressive pruning, resulting in notable efficiency gains with only marginal accuracy degradation. This offers a better trade-off compared to using the Baseline model directly.

Table 3. Parameter and FLOPs reduction versus test accuracy for SE and pruned SE

Models	Parameters	FLOPs/ops	Change	Test Accuracy
SE (unpruned)	11,243,471	569,146,895	—	0.9133
Pruned-SE	8,554,311	460,703,399	Parameters ↓23.3%, FLOPs ↓19.1%	0.8933

4.5. Ablation Study: Test-Time Augmentation (TTA)

Table 4. Test accuracy under different TTA settings using the final pruned-SE model

TTA Setting	Test Accuracy	Notes
none	0.8933	Recommended, stable
hflip	0.8933	Comparable to none
8-view	0.5333	Mismatched with task-specific normalisation/augmentation, significant degradation

As shown in Table 4, experiment compare different TTA strategies on the final pruned-SE model, including no TTA, horizontal flipping (hflip), and 8-view augmentation. As shown in Table 4, both the "none" and "hflip" settings achieve identical test accuracy (0.8933), whereas the 8-view TTA results in a significant drop to 0.5333. This suggests that excessive or mismatched TTA, under the current normalisation and augmentation scheme, can introduce distributional shifts and compromise generalisation. Therefore, for deployment, it is recommended to disable complex TTA and retain only essential input normalisation.

Under the unified training configuration, the SE model achieves a test accuracy of 0.9133, representing a +0.0400 improvement over the Baseline (0.8733). Although CBAM attains the highest validation accuracy (0.9667), its test accuracy drops to 0.8667, revealing a notable generalisation gap ("better validation, weaker test").

In terms of explainability, Grad-CAM quantitative metrics (leaf coverage, leaf IoU, and leaf-edge IoU) consistently follow the order CBAM > SE > Baseline, indicating that attention mechanisms enable better focus on the leaf body and edges. However, such "better localised attention" does not necessarily translate into higher test accuracy in this task.

When applying progressive pruning (6% × 3 steps on stages 3/4) to the SE model, the number of parameters is reduced from 11.24M to 8.55M (−23.3%), and FLOPs decrease from 5.69×10^8 to 4.61×10^8 (−19.1%), while test accuracy only slightly drops from 0.9133 to 0.8933—still higher

than the Baseline. This demonstrates a favourable accuracy–efficiency trade-off, making pruned-SE a better choice for lightweight deployment.

Regarding inference-time augmentation, simple horizontal flipping (hflip) provides no noticeable gain (0.8933), while aggressive 8-view TTA severely degrades performance (0.5333). This highlights the importance of keeping the inference-time preprocessing aligned with the training distribution.

In summary, on the given dataset and task, SE serves as a more robust backbone. For practical deployment, the recommended configuration is: SE + mild progressive pruning + no or simplified TTA.

5. Discussion

5.1. SE comparing with CBAM Trade-off

According to the test set results, SE achieved a significantly higher accuracy of 0.9133 compared to the Baseline (0.8733) and outperformed CBAM (0.8667). In terms of interpretability metrics (Coverage/IoU/Leaf Margin IoU), CBAM showed a slight advantage. For edge deployment, more robust test performance and lower complexity are more critical; hence, SE was selected as the backbone for subsequent compression and deployment. This suggests that in small-sample or fine-grained inter-class scenarios, channel recalibration (SE) offers better generalisation stability than "channel + spatial" attention (CBAM). If stronger interpretability is required, introducing lightweight spatial attention on top of SE could be considered a compromise.

5.2. Pruning Strategy and Layer-wise Sensitivity

A one-time global pruning of 40% resulted in significant degradation. In contrast, progressive pruning applied only to the last two stages (∼6% per step for three steps) reduced parameters from 11.24M to 8.55M (−23.3%) and FLOPs from 5.69×10^8 to 4.61×10^8 (−19.1%), while test accuracy only slightly decreased from 0.9133 to 0.8933, still outperforming the Baseline. This aligns with the empirical observation that front layers retain low-level texture/edge representations, while rear layers exhibit higher redundancy. In practice, stage-wise, incremental, and rear-layer-first pruning is a more robust compression path. Further refinement could involve sensitivity-aware or structured channel importance scoring for quota allocation.

5.3. Relationship Between Interpretability and Generalisation

Attention modules significantly improved the alignment between Grad-CAM and leaf/leaf margin regions (CBAM > SE > Baseline), indicating that the models focus more on discriminative regions. However, "better attention ≠, higher test accuracy": CBAM had the best interpretability metrics but did not translate to higher test accuracy. Potential reasons include overfitting of spatial attention to background textures and slight shifts in the train/validation distribution. Therefore, for deployment, stable test performance should be the primary goal, with interpretability serving as supplementary evidence. To optimise both, boundary-aware regularisation or distillation could be explored to improve "Leaf Margin IoU" without sacrificing generalisation.

5.4. Inference Strategy and Distribution Consistency

Eight-view TTA (Test-Time Augmentation) caused a significant drop from 0.8933 to 0.5333, indicating that input distribution misalignment with training/validation amplifies deviation. Horizontal flip TTA performed similarly to no TTA (both 0.8933). The conclusion is that under small-sample or strong regularisation settings, complex TTA should be avoided, and lightweight preprocessing/scaling consistent with the training phase is more reliable. If robustness needs improvement, prioritising slight averaging via model ensembles over aggressive TTA is recommended.

6. Conclusion

This paper systematically compared ResNet18, SE, and CBAM on the Swedish Leaf dataset. Progressive channel pruning and fine-tuning were implemented on SE, resulting in a test accuracy (0.8933) that remains higher than the Baseline (0.8733) despite significant reductions in parameters and computational load. Experiments also showed that attention modules enhance the focus on the leaf's main body, and TTA is not always effective. Future work will consider combining Neural Architecture Search (NAS) with joint pruning optimisation. Introducing more robust discriminative distillation and feature alignment strategies to further recover the accuracy loss from pruning. Providing end-to-end ONNX/NCNN/TensorRT deployment scripts and actual latency reports. Validating generalisation ability on larger-scale, multi-species leaf datasets and exploring the promotion of interpretability through multi-task (classification and segmentation) joint training.

Authors Contribution

All the authors contributed equally, and their names were listed in alphabetical order.

References

[1] Hu Jie, Shen Li, Sun Gang. Squeeze-and-Excitation Networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018: 7132–7141.

[2] Paszke Adam, Gross Sam, Massa Francisco, et al. PyTorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems (NeurIPS), 2019, 32: 8024–8035.

[3] Loshchilov Ilya, Hutter Frank. Decoupled weight decay regularization. International Conference on Learning Representations (ICLR), 2019.

[4] Zhang Hongyi, Cisse Moustapha, Dauphin Yann N., Lopez-Paz David. mixup: Beyond empirical risk minimization. International Conference on Learning Representations (ICLR), 2018.

[5] He Yihui, Zhang Xiangyu, Sun Jian. Channel pruning for accelerating very deep neural networks. Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017: 1389–1397.

[6] Molchanov Pavlo, Tyree Stephan, Karras Tero, Aila Timo, Kautz Jan. Pruning CNNs for resource efficient inference. International Conference on Learning Representations (ICLR), 2017.

[7] Hinton Geoffrey, Vinyals Oriol, Dean Jeff. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015.

[8] Chattopadhyay Aditya, Sarkar Anirban, Howlader Prantik, Balasubramanian Vineeth N. Grad-CAM++: Improved visual explanations for deep convolutional networks. IEEE Winter Conference on Applications of Computer Vision (WACV), 2018: 839–847.

[9] Söderkvist Olof J. O. Computer vision classification of leaves from Swedish trees. Master's thesis, Linköping University, 2001.

[10] He Kaiming, Zhang Xiangyu, Ren Shaoqing, Sun Jian. Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016: 770–778.

[11] Woo Sanghyun, Park Jongchan, Lee Joon-Young, Kweon in So. CBAM: Convolutional block attention module. Proceedings of the European Conference on Computer Vision (ECCV), 2018: 3–19.

[12] Li Hao, Kadav Asim, Durdanovic Igor, Samet Hanan, Graf Hans Peter. Pruning filters for efficient ConvNets. International Conference on Learning Representations (ICLR), 2017.

[13] Selvaraju Ramprasaath R., Cogswell Michael, Das Abhishek, et al. Grad-CAM: Visual explanations from deep networks via gradient-based localization. Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017: 618–626.

[14] Girshick Ross B. Fast R-CNN. Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2015: 1440–1448.