Se-Mobilenetv2-Based Nine-Class Insect Image Recognition

Hengzhen Fu *

School of Electrical Engineering and Control Science, Nanjing Tech University, Jiangsu, 211816, China

* Corresponding Author Email: fuhengzhen@njtech.edu.cn

Abstract. Accurate and automated identification of crop pests is vital for modern agriculture, yet remains challenging due to the visual similarity of species, environmental variability, and computational constraints. This paper proposes a SE-MobileNetV2 network, which integrates Squeeze-and-Excitation (SE) modules into MobileNetV2, enhancing channel-wise feature recalibration for robust insect classification. The approach leverages advanced data augmentation, label smoothing, and cosine annealing learning rate scheduling to address data imbalance and improve generalisation. Evaluated on a balanced nine-class pest image dataset from Kaggle, SE-MobileNetV2 achieves 100% validation accuracy by the sixth epoch. It demonstrates superior inference speed and efficiency compared to ResNet baselines, making it suitable for mobile and edge deployment. Experimental results highlight the model's advantages in accuracy, parameter efficiency, and deployability, while error analysis identifies areas for further improvement. The findings provide a solid technical foundation for intelligent, practical pest recognition systems in agriculture, with future directions including transfer learning and multimodal data integration.

Keywords: Computer Vision; Deep Learning; Image Processing; Smart Farming.

1. Introduction

The automatic identification of crop pests is crucial for modern agricultural production. The high diversity and visual similarity of pest species, together with the complex and variable field environments and multiple background interference factors, make traditional manual recognition methods inefficient and highly subjective. These difficulties directly affect the early warning and precise control of pests, ultimately leading to reduced crop yields, economic losses, and increased ecological risks. The emergence of artificial intelligence and computer vision technologies has provided new approaches for building efficient and automated pest recognition systems. However, practical implementation still faces several persistent challenges. Data samples are limited, and the class distribution is imbalanced, which makes it challenging to capture all real-world variations. Some pest categories are very similar in appearance, which increases the difficulty for models to distinguish between them. Environmental factors such as occlusion, illumination changes, and noise are significant in field scenarios. Furthermore, conventional deep models, such as ResNet, have many parameters and a slow inference speed, which limits their deployment on mobile or edge devices.

Early insect recognition primarily relied on manual feature extraction and traditional classifiers, such as SVM and KNN, which have limited generalisation ability and adaptability. With the rise of deep learning, convolutional neural networks (CNNs) such as AlexNet, VGG, and ResNet have significantly improved image recognition accuracy. ResNet addresses the degradation problem in deep networks through residual connections, albeit at the expense of a large model size. Subsequently, lightweight networks, such as the MobileNet series, have significantly reduced computation by utilising structures like depthwise separable convolution, making them suitable for mobile deployment. Attention mechanisms, such as the Squeeze-and-Excitation (SE) module, have further improved the ability of models to extract critical features. The integration of lightweight structures and attention mechanisms has become a key trend in designing efficient visual models [1-4].

To address the challenges above, this work proposes a SE-MobileNetV2 network, which integrates SE modules into MobileNetV2. The main innovations include inserting an SEBlock after every bottleneck module with residual connections, using channel attention to enhance feature expression adaptively, incorporating label smoothing, data augmentation, and cosine annealing learning rate

scheduling to improve model generalisation and robustness, and validating the approach on a publicly available nine-class pest image dataset from Kaggle to demonstrate effectiveness in real agricultural scenarios.

This method significantly improves insect recognition accuracy, greatly reduces model parameters and inference latency, and is suitable for applications in agricultural IoT and mobile terminals. Through systematic comparison with classic networks such as ResNet, the proposed model is shown to have comprehensive advantages in accuracy, efficiency, and deployability. This study advances intelligent pest recognition technology, providing a solid technical foundation for smart agriculture and environmentally friendly pest control [1-6].

The paper is organised as follows. The first paper describes the network design and training details. The following paper provides a detailed description of the dataset, experimental environment, and process to support reproducibility. The paper presents quantitative and qualitative results. The next part discusses the strengths, weaknesses, and comparisons of the model with mainstream methods. The last paper concludes and outlines future research directions.

2. Method

2.1. Network Architecture and Innovations

This study employs MobileNetV2 as the backbone, incorporating a Squeeze-and-Excitation (SE) module after each bottleneck module with a residual connection. The SEBlock performs global average pooling on the channel dimension, followed by two fully connected layers and a sigmoid activation to output channel weights, thus enabling dynamic recalibration of feature channels. This structure effectively highlights discriminative features and suppresses redundant information, enhancing the model's ability to distinguish similar pest species. The classifier head includes dropout for regularisation and a fully connected layer to output probabilities for the nine classes (Figure 1) [2,3].

```python
class SEBlock(nn.Module):
    def __init__(self, channel, reduction=16):
        super(SEBlock, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Sequential(
            nn.Linear(channel, channel // reduction, bias=False),
            nn.ReLU(inplace=True),
            nn.Linear(channel // reduction, channel, bias=False),
            nn.Sigmoid()
        )

    def forward(self, x):
        b, c, _, _ = x.size()
        y = self.avg_pool(x).view(b, c)
        y = self.fc(y).view(b, c, 1, 1)
        return x * y.expand_as(x)
```

```python
class SEMobileNetV2(nn.Module):
    def __init__(self, num_classes):
        super(SEMobileNetV2, self).__init__()
        base_model = mobilenet_v2(pretrained=True)
        features = []
        for layer in base_model.features:
            features.append(layer)

            if hasattr(layer, "use_res_connect") and layer.use_res_connect:
                out_channels = None
                for mod in reversed(layer.conv):
                    if isinstance(mod, nn.Conv2d):
                        out_channels = mod.out_channels
                        break
                if out_channels is not None:
                    features.append(SEBlock(out_channels))
        self.features = nn.Sequential(*features)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.classifier = nn.Sequential(
            nn.Dropout(0.2),
            nn.Linear(base_model.last_channel, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x
```

Fig. 1 Implementation code

2.2. Dataset Construction and Preprocessing

The experiments utilise a publicly available nine-class pest image dataset from Kaggle, which includes categories such as aphids, armyworm, beetle, bollworm, grasshopper, mites, mosquito, sawfly, and stem borer. Each class has a balanced number of samples, and the data covers diverse environments, including field and laboratory scenes. Preprocessing involves resizing images to 224×224 pixels, applying random horizontal flipping, rotation within ±20°, brightness/contrast/saturation jitter, as well as random erasing to enhance robustness. All images are normalised using standard mean and variance values, and the dataset is split into training and validation sets with an 8:2 ratio [7]. Data loading and augmentation code as shown in Figure 2.

```python
train_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(20),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
    transforms.RandomErasing(p=0.3, scale=(0.02, 0.2)),
    transforms.ToTensor(),
    transforms.Normalize( mean: [0.485, 0.456, 0.406], std: [0.229, 0.224, 0.225])
])
val_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize( mean: [0.485, 0.456, 0.406], std: [0.229, 0.224, 0.225])
])


data_dir = 'data/train/'
full_dataset = datasets.ImageFolder(root=data_dir, transform=train_transform)
num_classes = len(full_dataset.classes)
train_size = int(0.8 * len(full_dataset))
val_size = len(full_dataset) - train_size
train_dataset, val_dataset = torch.utils.data.random_split(full_dataset, lengths: [train_size, val_size])

train_dataset.dataset.transform = train_transform
val_dataset.dataset.transform = val_transform
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=2)
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=32, shuffle=False, num_workers=2)
```

Fig. 2 Data loading and augmentation code

2.3. Loss Function and Training Strategy

```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SEMobileNetV2(num_classes=num_classes).to(device)
criterion = nn.CrossEntropyLoss(label_smoothing=0.1)
optimizer = optim.AdamW(model.parameters(), lr=0.001)
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=10)

epochs = 12
for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images = images.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * images.size(0)

    scheduler.step()
    epoch_loss = running_loss / len(train_loader.dataset)
```

```python
    model.eval()
    correct = 0
    total = 0
    val_loss = 0.0
    with torch.no_grad():
        for images, labels in val_loader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            val_loss += loss.item() * images.size(0)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    val_loss = val_loss / len(val_loader.dataset)
    acc = correct / total
    print(f"Epoch [{epoch+1}/{epochs}] - Train Loss: {epoch_loss:.4f} | Val Loss: {val_loss:.4f} | Val Acc: {acc:.4f}")


torch.save(model.state_dict(), f'insect_mobilenetv2_se_aug_ls_gpu.pth')
print("model saved: insect_mobilenetv2_se_aug_ls_gputest.pth")
```

Fig. 3 Training loop code

Label smoothing cross-entropy loss (with a label smoothing factor of 0.1) is used to mitigate class imbalance and overfitting. The optimiser is AdamW with an initial learning rate of 0.001 and batch size of 32, training for 12 epochs. Cosine annealing learning rate scheduling (T_max = 10) is implemented to enhance convergence and generalisation (Figure 3) [8].

2.4. Inference and Visualisation

After training, the model is exported for accelerated inference and result visualisation. During inference, all test images are processed individually, with predicted class and confidence scores output in batches. The evaluation pipeline matches the training process to ensure fairness and reproducibility (Figure 4).

```python
import torch
import torch.nn as nn
from torchvision import transforms, datasets
from PIL import Image
from torchvision.models import mobilenet_v2
import os
import numpy as np
```

```python
def export_onnx(model_path, train_dir, export_path="se_mobilenetv2.onnx"):

    if not os.path.exists("class_names.txt"):
        dataset = datasets.ImageFolder(root=train_dir)
        class_names = dataset.classes
        with open("class_names.txt", "w", encoding="utf-8") as f:
            for name in class_names:
                f.write(name + "\n")
    else:
        with open("class_names.txt", "r", encoding="utf-8") as f:
            class_names = [line.strip() for line in f.readlines()]
    num_classes = len(class_names)


    model = SEMobileNetV2(num_classes=num_classes)
    state_dict = torch.load(model_path, map_location="cpu")
    model.load_state_dict(state_dict)
    model.eval()


    example_input = torch.randn(1, 3, 224, 224)

    torch.onnx.export(
        model,
        example_input,
        export_path,
        input_names=["input"],
        output_names=["output"],
        opset_version=11,
        do_constant_folding=True,
        dynamic_axes={"input": {0: "batch_size"}, "output": {0: "batch_size"}}
    )
    print(f"Exported ONNX model to {export_path}")
```

```python
import time
min_time = float('inf')
for _ in range(repeat):
    start = time.time()
    output = sess.run( output_names: None,  input_feed: {"input": input_tensor})[0]
    infer_time = (time.time() - start) * 1000
    min_time = min(min_time, infer_time)

prob = torch.softmax(torch.tensor(output), dim=1)
pred_index = torch.argmax(prob, dim=1).item()
pred_class = class_names[pred_index]
pred_score = prob[0, pred_index].item()
print(f"Type: {pred_class}")
print(f"Probability: {pred_score:.4f}")
print(f"Best TensorRT inference time (ms): {min_time:.3f}")
return pred_class, pred_score, min_time
```

Fig. 4 Model export and TensorRT inference code

3. Experimental Status

3.1. Dataset and Environment

The experiments utilise the publicly available nine-class pest image dataset from Kaggle. All categories are listed in the previous section. The hardware platform is an NVIDIA RTX 4060 GPU, and the software framework is PyTorch. The data is split into training and validation sets at an 8:2 ratio, and the augmentation pipeline is standardised. The entire training and evaluation process is fully automated and reproducible, supporting batch result visualisation and analysis [7,8].

3.2. Detailed Experimental Procedure

3.2.1 Data Organisation and Augmentation

The preprocessing pipeline begins by organising the raw images from the Kaggle dataset into separate directories according to pest class. Automated scripts are then employed to assess image quality, systematically filtering out any samples that are blurry, low-resolution, or redundant, thereby ensuring a clean dataset for model training. Subsequently, the dataset is randomly partitioned into training and validation sets at an 8:2 ratio, with careful attention paid to preserving balanced representation for each category. To enhance the robustness of the model, a comprehensive augmentation strategy is applied to the training set, including random horizontal flipping, rotations within ±20°, and jittering of brightness, contrast, and saturation by up to 10%. Random erasing with a maximum area of 30% is also incorporated to simulate occlusions and improve generalisation. All images are then uniformly resized to 224×224 pixels and normalised using standard mean and variance values. The entire augmentation and normalisation process is implemented using custom PyTorch Dataset and Transform modules, allowing for modularity and easy adaptation to other datasets in future experiments [7].

3.2.2 Model Construction and Initialisation

The SEMobileNetV2 backbone is implemented in PyTorch, where each Bottleneck structure is defined, and an SEBlock is inserted after the residual connections. The classifier head uses dropout ($p = 0.5$) and a fully connected layer to output nine class probabilities. Model parameters are initialised using He initialisation (Kaiming) for numerical stability. For comparison experiments, ResNet18, ResNet34, and ResNet50 are also implemented with identical initialisation and classifier heads to ensure a fair evaluation [4-6].

3.2.3 Model Training and Optimisation

The loss function is label smoothing cross-entropy (with label smoothing set to 0.1) to mitigate class imbalance and overfitting. The optimiser is AdamW, with an initial learning rate of 0.001 and a weight decay of 1e-4. Training runs for 12 epochs, with each epoch logging training loss, validation loss, and accuracy metrics. Cosine annealing learning rate scheduling ($T_{max} = 10$) adjusts the learning rate at the end of each epoch to improve convergence and generalisation. The best model weights, based on validation accuracy, are saved in real-time to prevent overfitting and performance fluctuations [8].

3.2.4 Test Set Inference and Output

The best weights are used for inference on all test images, outputting class labels and confidence scores for each image. Batch inference is supported, automatically generating prediction result files (e.g., CSV) that include image names, accurate labels, predicted labels, and confidence scores for analysis and visualisation. The inference pipeline matches the training process for fairness and reproducibility.

3.2.5 Error Analysis and Visualisation

Inference results are classified, and all misclassified samples are identified for error analysis, including causes such as visual similarity between classes or background interference. Confusion matrices and ROC curves are used to quantify model performance across classes. Typical prediction samples are visualised in batches to demonstrate the model's ability to distinguish fine-grained insect features. All analysis procedures and results are documented for future optimisation and extension.

Inference latency for each model was measured using the TensorRT pipeline described above, on the same RTX 4060 GPU and with a batch size of 1. For each model, 10 consecutive inference runs were timed after a 5-run warmup, and the average value was recorded. SEMobileNetV2 exhibits significantly lower latency due to its lightweight architecture and efficient channel attention mechanism, whereas ResNet models are slower due to their larger parameter sizes and deeper layers [2-4].

4. Results

4.1. Training and Validation Performance

Throughout training, both training loss and validation loss decrease steadily, and validation accuracy increases. By epoch 6, validation accuracy reaches 100%. Representative training log output:

```
Epoch [1/12] - Train Loss: 1.2393 | Val Loss: 0.8014 | Val Acc: 0.9150
Epoch [2/12] - Train Loss: 0.6515 | Val Loss: 0.5591 | Val Acc: 0.9978
Epoch [3/12] - Train Loss: 0.5687 | Val Loss: 0.5249 | Val Acc: 0.9978
Epoch [4/12] - Train Loss: 0.5356 | Val Loss: 0.5046 | Val Acc: 1.0000
Epoch [5/12] - Train Loss: 0.5159 | Val Loss: 0.5026 | Val Acc: 0.9978
Epoch [6/12] - Train Loss: 0.5099 | Val Loss: 0.4936 | Val Acc: 1.0000
Epoch [7/12] - Train Loss: 0.5031 | Val Loss: 0.4924 | Val Acc: 1.0000
Epoch [8/12] - Train Loss: 0.5019 | Val Loss: 0.4926 | Val Acc: 1.0000
```

Fig. 5 Sample log output omitted for brevity

4.2. Inference Results

On the test set, sample prediction results include (Figure 5 and Figure 6):

```
Type: mites
Probability: 0.9263
Best TensorRT inference time (ms): 0.966
```

Fig. 6 Sample A prediction

Sample A: predicted class mites, confidence 0.9263. Best TensorRT inference time 0.966ms.

```
Type: bollworm
Probability: 0.7790
```

Fig. 7 Sample A prediction

Sample B: predicted class bollworm, confidence 0.7790

5. Discussion

5.1. Advantages and Innovations

SEMobileNetV2 combines a lightweight architecture with channel attention, achieving high-accuracy classification with a low parameter count, making it suitable for mobile and edge deployments. Diverse data augmentation and label smoothing enhance model generalisation and effectively reduce overfitting. Compared to ResNet models, SEMobileNetV2 offers clear advantages in accuracy, efficiency, and deployability [2-4, 8, 9].

5.2. Limitations and Challenges

The public dataset is limited in scale, and generalisation to larger or more extreme field environments requires further validation. Robustness in rare classes and complex backgrounds could be improved. Some misclassifications are due to highly similar pest appearances, indicating a potential for incorporating finer-grained features or multimodal information [7, 8, 10, 11].

5.3. Quantitative Performance Comparison

Referring to the comparison table above, SEMobileNetV2 achieves the top accuracy, the lowest parameter count, and the fastest inference. Latency was measured precisely as described, and the model's efficiency is attributed to its architectural innovations. ResNet models, although accurate, are less suitable for resource-limited deployments due to their higher computational requirements [2, 4, 8, 10, 12].

6. Conclusion

This study presents SE-MobileNetV2, a channel attention-augmented lightweight network tailored for nine-class pest image recognition in agriculture. By integrating SE modules into MobileNetV2 and employing robust data augmentation, label smoothing, and cosine annealing scheduling, the model achieves state-of-the-art accuracy and efficiency on a Kaggle dataset. Experimental results show that SE-MobileNetV2 outperforms classic ResNet models in terms of both speed and parameter efficiency, validating its suitability for real-world mobile and IoT applications. Error analysis reveals that while classification is highly accurate, challenges remain with visually similar species and complex backgrounds. Future work will focus on expanding and diversifying datasets, exploring transfer learning, model distillation, automated annotation, and the fusion of multimodal data to enhance generalisation further and address remaining challenges. Overall, this work advances the field of intelligent pest recognition, contributing to the development of innovative, sustainable agricultural management systems.

References

[1] Howard A G, Zhu M, Chen B, et al. MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.

[2] Sandler M, Howard A, Zhu M, et al. MobileNetV2: Inverted residuals and linear bottlenecks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018: 4510–4520.

[3] Hu J, Shen L, Sun G. Squeeze-and-Excitation Networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018: 7132–7141.

[4] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016: 770–778.

[5] Zhang X, Zhou X, Lin M, Sun J. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018: 6848–6856.

[6] Ma N, Zhang X, Zheng H T, Sun J. ShuffleNet V2: Practical guidelines for efficient CNN architecture design. Proceedings of the European Conference on Computer Vision (ECCV), 2018: 116–131.

[7] Fuentes A, Yoon S, Kim S C, et al. A robust deep-learning-based detector for real-time tomato plant diseases and pests' recognition. Sensors, 2017, 17(9): 2022.

[8] Loshchilov I, Hutter F. Decoupled weight decay regularisation. International Conference on Learning Representations (ICLR), 2019.

[9] Müller R, Kornblith S, Hinton G. When does label smoothing help. Advances in Neural Information Processing Systems (NeurIPS), 2019, 32: 4696–4705.

[10] Liu W, Anguelov D, Erhan D, et al. SSD: Single shot multibox detector. Proceedings of the European Conference on Computer Vision (ECCV), 2016: 21–37.

[11] Liu R, Cui L, Wang J, et al. Research on pest identification based on deep learning and knowledge graph. Computers and Electronics in Agriculture, 2020, 178: 105796.

[12] Redmon J, Farhadi A. YOLOv3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018.