Technical Report: Worm Image Analysis

Comparative Frameworks and ResNet-50 Pipeline

Contents

1 Designs of Previous Methods

1.1 VAE with Color Heuristic

1.1.1 Overview

The first approach leverages a Variational Autoencoder (VAE) trained exclusively on images containing worms. Since the dataset lacks negative samples, an *unusual color* heuristic is introduced to localize anomalous (worm) regions via pixel-wise color deviations.

1.1.2 Architecture

- **Encoder:**
  - Input: $128 \times 128 \times 3$ RGB image.
  - Conv layers: Four blocks of `Conv2d` (kernel 4, stride 2, padding 1) with channel progression $3 \to 64 \to 128 \to 256 \to 512$, each followed by BatchNorm and LeakyReLU(0.2).
  - Flatten to vector, then two parallel linear layers outputting $\mu,\ \log \sigma^2 \in \mathbb{R}^z$.
- **Decoder:**
  - Sample latent $z \sim \mathcal{N}(\mu, \sigma^2)$.
  - Linear layer to reshape into $512 \times 8 \times 8$.
  - Four `ConvTranspose2d` blocks mirroring encoder (kernel 4, stride 2, padding 1), with ReLU activations and BatchNorm, culminating in a 3-channel output via `Tanh`.

1.1.3 Color Heuristic

Define per-channel mean μ_c over the image. For each pixel (x, y) and channel $c \in \{R, G, B\}$:

$$d_c(x, y) = \left|I_c(x, y) - \mu_c\right|, \quad D(x, y) = \sum_c d_c(x, y).$$

Let $(x^*, y^*) = \arg\max D(x, y)$ and $U_c = I_c(x^*, y^*)$. Construct a dynamic mask:

$$\mathrm{LB}_c = U_c - \alpha_c U_c, \quad \mathrm{UB}_c = U_c + \beta_c U_c,$$

and compute the fraction of pixels within $[\mathrm{LB}_c, \mathrm{UB}_c]$ as a *color ratio* feature.

1.1.4 Training Details

- Loss: $\mathcal{L}_{\mathrm{VAE}} = \mathrm{BCE}(x_{\mathrm{recon}}, x_{\mathrm{orig}}) + \mathrm{KL}(\mathcal{N}(\mu, \sigma^2) \| \mathcal{N}(0, 1))$.
- Optimizer: Adam, $\mathrm{lr} = 2 \times 10^{-4}$.
- Epochs: 100, batch size 64.
- Data: 122 worm train images, 16 eval images and 35 test images

1.1.5 Results

- F1 Score (worm-only): 0.87097.
- Chosen reconstruction-error threshold on validation set separated worm vs. background regions.
- Qualitative: reconstructions preserved background well, but missed fine worm structures.

1.2 GAN-based Framework

1.2.1 Overview

The second design integrates a Generative Adversarial Network without the color heuristic to both generate realistic worm images and detect anomalies.

1.2.2 Generator

```
class Generator(nn.Module):
    def __init__(self, nz, ngf, nc):
        super().__init__()
        self.main = nn.Sequential(
            # input: (nz) x1x1
            nn.ConvTranspose2d(nz, ngf*8, 4,1,0),
            nn.BatchNorm2d(ngf*8), nn.ReLU(True),
            # upsample to 8x8
            nn.ConvTranspose2d(ngf*8, ngf*4, 4,2,1),
            nn.BatchNorm2d(ngf*4), nn.ReLU(True),
            # upsample to 16x16, 32x32, 64x64...
            nn.ConvTranspose2d(ngf, nc, 4,2,1), nn.Tanh()
        )
    def forward(self, z): return self.main(z)
```

1.2.3 Discriminator

```
class Discriminator(nn.Module):
    def __init__(self, nc, ndf):
        super().__init__()
        self.main = nn.Sequential(
            # input: ncx128x128
            nn.Conv2d(nc, ndf, 4,2,1), nn.LeakyReLU(0.2, True),
            # downsample to 64x64, 32x32, 16x16...
            nn.Conv2d(ndf*8, 1, 4,1,0)
        )
        self.sigmoid = nn.Sigmoid()
    def forward(self, x):
        feat = self.main(x)
        out  = self.sigmoid(feat.view(-1,1))
        return out
```

1.2.4 Dataset and Training

- Worm-only images, same samples.
- Batch size 64, 150 epochs.
- Losses: BCE for $\mathcal{L}_D$ and $\mathcal{L}_G$.
- Learning rates: 2×10^{-4} for both networks.

1.2.5 Anomaly Detection

During inference, input images are passed through the Generator (for reconstruction) and Discriminator (for realism score), combined with the color-ratio feature to yield an anomaly score. A threshold on this score distinguishes worm regions.

1.2.6 Results

- F1 Score (worm-only): 0.88.
- F1 Score (mixed worm vs. novel): 0.66 (indicating overfitting).
- Visual inspection showed good background modeling but inconsistent worm shape generation.

2 Reconstruction Results of Initial Models

2.1 VAE Reconstructions

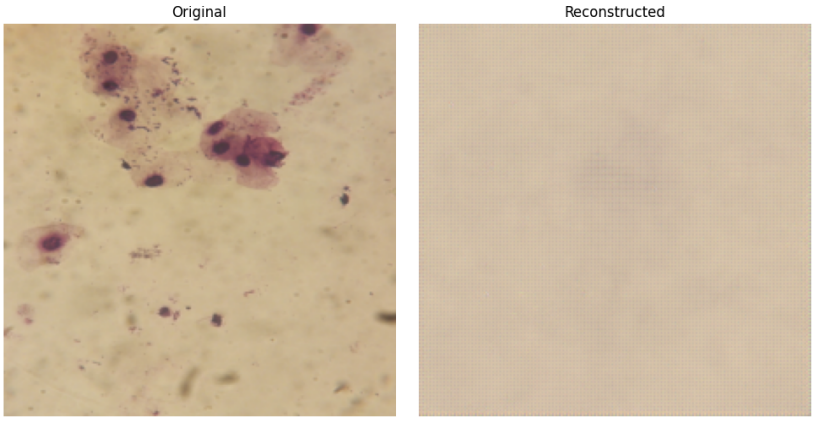


Figure 1: Example VAE reconstruction: original (left) vs. reconstructed (right). The VAE captures global texture but loses fine worm details.

2.2 GAN Reconstructions

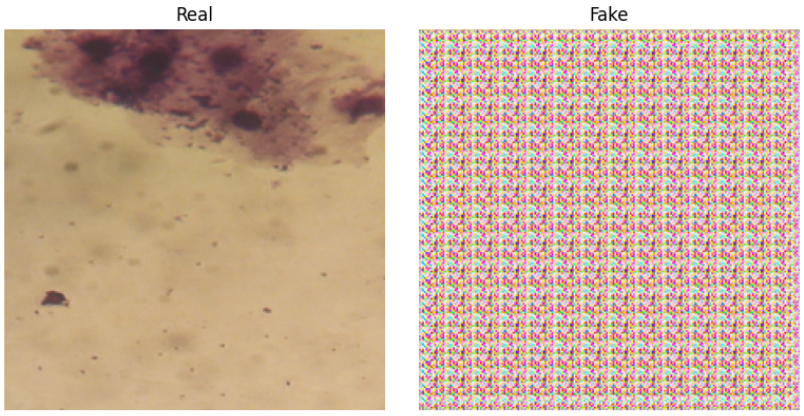


Figure 2: Example GAN reconstruction: generated image (right) compared to real sample (left). Background is realistic, worm morphology is inconsistent.

3 PyTorch ResNet-50 Pipeline with Focal Loss

3.1 Introduction

Now we provide a technical overview of a PyTorch-based image classification pipeline. It covers:

- Custom Focal Loss implementation
- Data augmentation and normalization
- Dataset and DataLoader setup
- Model modification (ResNet-50)
- Training loop with early stopping
- Evaluation using F1-score and threshold optimization

3.2 Focal Loss

3.2.1 Motivation

Focal Loss [1] addresses class imbalance by down-weighting easy examples and focusing training on hard negatives.

3.2.2 Definition

Given binary classification logits $x \in \mathbb{R}$ and targets $y \in \{0, 1\}$:

$$\mathrm{BCE}(x, y) = -\big[y \log \sigma(x) + (1 - y) \log(1 - \sigma(x))\big], \tag{1}$$

$$p_t = \exp\big(-\mathrm{BCE}(x, y)\big) = \begin{cases} \sigma(x), & y = 1, \\ 1 - \sigma(x), & y = 0, \end{cases} \tag{2}$$

$$\mathcal{L}_{\mathrm{FL}}(x, y) = \alpha \, (1 - p_t)^{\gamma} \, \mathrm{BCE}(x, y), \tag{3}$$

where $\sigma(x) = 1/(1 + e^{-x})$, $\alpha \in [0, 1]$ balances class weights, and $\gamma \geq 0$ focuses on difficult examples.

3.2.3 PyTorch Implementation

```python
class FocalLoss(torch.nn.Module):
    def __init__(self, alpha=0.25, gamma=2.0, reduction='mean'):
        super(FocalLoss, self).__init__()
        self.alpha   = alpha
        self.gamma   = gamma
        self.reduction = reduction

    def forward(self, inputs, targets):
        # Flatten predictions and targets
        inputs  = inputs.view(-1)
        targets = targets.view(-1)

        # Compute per-example BCE loss (logits version)
        BCE_loss = F.binary_cross_entropy_with_logits(
                        inputs, targets, reduction='none'
                   )
        # Convert to probability scale
        pt = torch.exp(-BCE_loss)
        # Compute focal loss
        F_loss = self.alpha * (1 - pt) ** self.gamma * BCE_loss

        if self.reduction == 'mean':
            return F_loss.mean()
        else:
            return F_loss.sum()
```

3.3 Data Preparation

3.3.1 Training Augmentations

```
train_transforms = transforms.Compose([
    RandomRotation(degrees=30),
    RandomHorizontalFlip(p=0.5),
    RandomVerticalFlip(p=0.5),
    RandomResizedCrop(size=(224,224), scale=(0.8,1.0)),
    transforms.ToTensor(),
    transforms.Normalize([0.5],[0.5])
])
```

- `RandomRotation`: rotate image up to $\pm 30°$.
- `RandomHorizontalFlip`/`VerticalFlip`: flip with 50% chance.
- `RandomResizedCrop`: crop and resize to 224×224.
- $\mu = 0.5, \sigma = 0.5$ normalization.

3.3.2 Validation Transforms

```
val_transforms = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5],[0.5])
])
```

3.3.3 Datasets and DataLoaders

```
train_dataset = ImageFolder("./new_data/train", transform=train_transforms)
val_dataset   = ImageFolder("./new_data/dev",   transform=val_transforms)

train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
val_loader   = DataLoader(val_dataset,   batch_size=8, shuffle=False)
```

Classes are indexed as {`'no_wormed':0, 'wormed':1`}.

3.4 Model Architecture

- Load pretrained ResNet-50.
- Replace final fully-connected layer with one output unit:

$$\texttt{model.fc} = \text{Sequential}\big(\text{Linear}(2048, 1)\big).$$

- Move model to designated device (GPU/CPU).

3.5 Training Loop

3.5.1 Hyperparameters

$$\text{learning rate} = 3 \times 10^{-6}, \quad \text{epochs} = 15, \quad \text{early stop patience} = 5.$$

3.5.2 Loss Function

Two options:

- **Focal Loss**: `FocalLoss(alpha=0.25, gamma=2)`.
- **Weighted BCE**: `BCEWithLogitsLoss(pos_weight=129/121)`.

3.5.3 Optimization

```
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```

3.5.4 Epoch Loop

1. **Training phase**: set `model.train()`, iterate over `train_loader`, compute loss, back-propagate, update weights.
2. **Validation phase**: set `model.eval()`, disable gradients, compute loss on `val_loader`.
3. **Logging**: record loss with TensorBoard `writer.add_scalar()`.
4. **Early Stopping**: if validation loss improves, reset counter and save model; else increment counter and stop if patience exceeded.

3.6 Evaluation

3.6.1 F1-Score Computation

```
def evaluate_f1(model, test_loader, device):
    model.eval()
    all_preds, all_labels = [], []
    with torch.no_grad():
        for imgs, lbls in test_loader:
            outputs = model(imgs.to(device)).squeeze(1)
            probs = torch.sigmoid(outputs).cpu().numpy()
            preds = (probs > 0.55).astype(int)
            all_preds.extend(preds)
            all_labels.extend(lbls.numpy())
    f1 = f1_score(all_labels, all_preds, average="binary")
    return f1, all_labels, all_preds
```

Achieved F1 = 0.9032; confusion matrix:

$$\begin{bmatrix} 26 & 6 \\ 0 & 28 \end{bmatrix}$$

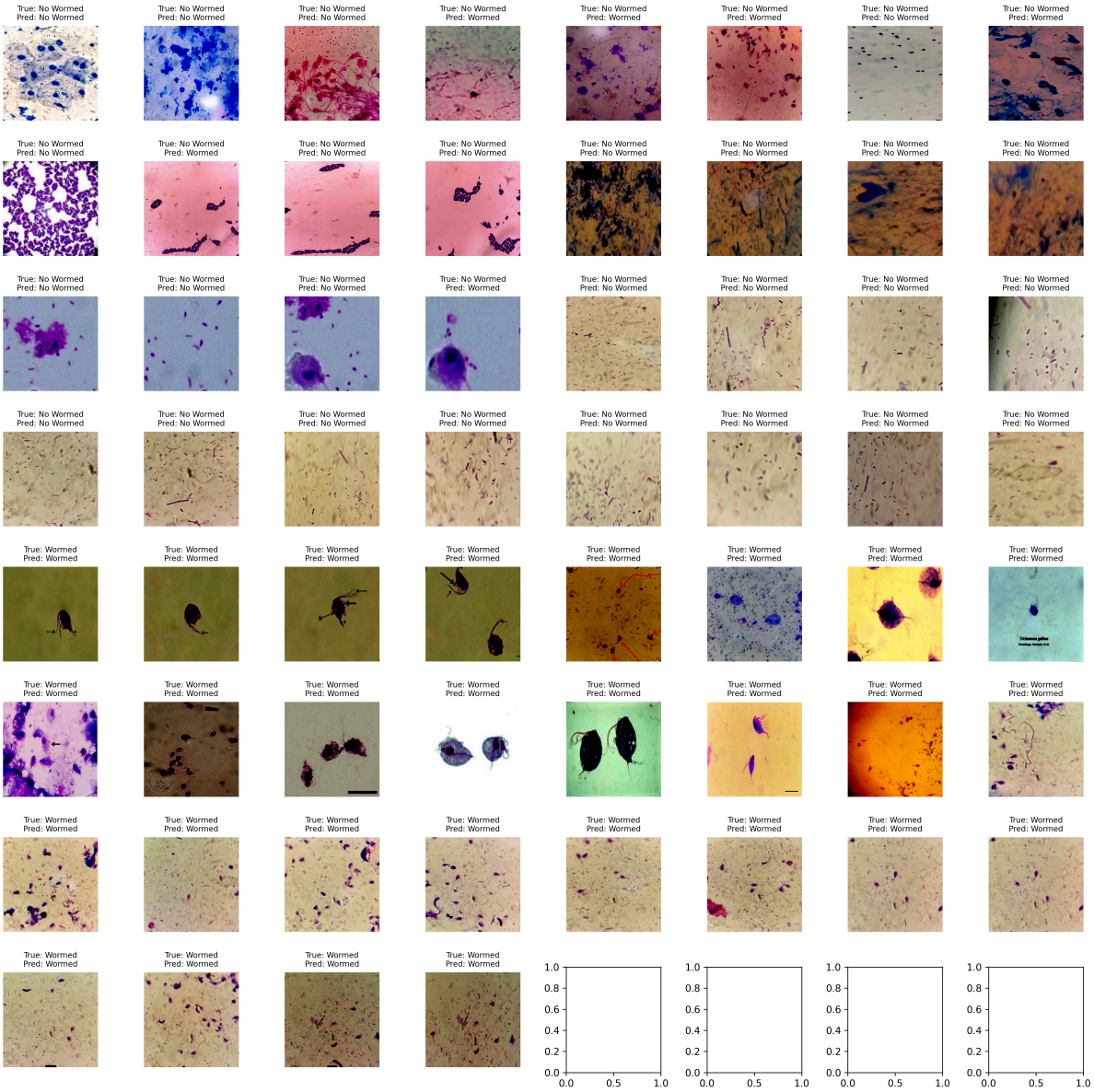
3.6.2 Threshold Optimization

Search threshold $\tau \in [0.1, 0.9]$ to maximize F1:

```
thresholds = np.linspace(0.1,0.9,100)
# ...
best_threshold = 0.55, best_f1 = 0.9032
```

3.7 Results

The presented pipeline leverages a strong backbone (ResNet-50), focal loss for class imbalance, extensive data augmentation, and rigorous evaluation through F1-score and threshold tuning. Early stopping prevents overfitting, and results indicate robust classification performance on the worm detection task. This is the results on test images .



4 Conclusion

Comparing three methods:

- VAE + Color Heuristic: F1=0.8709, limited structure learning.
- GAN + Color Heuristic: F1=0.88 on worm-only, poor generalization.
- ResNet-50 + Focal Loss: F1=0.9032, strong binary classification.

The ResNet-50 pipeline with focal loss demonstrates the best performance and generalization for worm detection.

References

[1] T.-Y. Lin et al., *Focal Loss for Dense Object Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence (2018).