

Technical Report: *terichomonas gallinae* Image Analysis

Comparative Frameworks and ResNet-50 Pipeline

Contents

1 Designs of Previous Methods	2
1.1 VAE with Color Heuristic	2
1.1.1 Overview	2
1.1.2 Architecture	2
1.1.3 Color Heuristic	2
1.1.4 Training Details	2
1.1.5 Results	2
1.2 GAN-based Framework	3
1.2.1 Overview	3
1.2.2 Generator	3
1.2.3 Discriminator	3
1.2.4 Dataset and Training	3
1.2.5 Anomaly Detection	3
1.2.6 Results	4
2 Reconstruction Results of Initial Models	4
2.1 VAE Reconstructions	4
2.2 GAN Reconstructions	4
3 PyTorch ResNet-50 Pipeline with Focal Loss	4
3.1 Introduction	4
3.2 Focal Loss	5
3.2.1 Motivation	5
3.2.2 Definition	5
3.2.3 PyTorch Implementation	5
3.3 Data Preparation	6
3.3.1 Training Augmentations	6
3.3.2 Validation Transforms	6
3.3.3 Datasets and DataLoaders	6
3.4 Model Architecture	6
3.5 Training Loop	6
3.5.1 Hyperparameters	6
3.5.2 Loss Function	7
3.5.3 Optimization	7
3.5.4 Epoch Loop	7
3.6 Evaluation	7
3.6.1 F1-Score Computation	7
3.6.2 Threshold Optimization	7
3.7 Results	7
4 Conclusion	8

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 *terichomonas gallinae*. Since the dataset lacks negative samples, an *unusual color* heuristic is introduced to localize anomalous (*terichomonas gallinae*) 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 $3 \rightarrow 64 \rightarrow 128 \rightarrow 256 \rightarrow 512$, 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 $c \in \{R, G, B\}$:

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

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

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

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

1.1.4 Training Details

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

1.1.5 Results

- F1 Score (*terichomonas gallinae*-only): 0.87097.
- Chosen reconstruction-error threshold on validation set separated *terichomonas gallinae* vs. background regions.
- Qualitative: reconstructions preserved background well, but missed fine *terichomonas gallinae* 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 terichomonas gallinae images and detect anomalies.

1.2.2 Generator

```
1 class Generator(nn.Module):
2     def __init__(self, nz, ngf, nc):
3         super().__init__()
4         self.main = nn.Sequential(
5             # input: (nz) x1x1
6             nn.ConvTranspose2d(nz, ngf*8, 4, 1, 0),
7             nn.BatchNorm2d(ngf*8), nn.ReLU(True),
8             # upsample to 8x8
9             nn.ConvTranspose2d(ngf*8, ngf*4, 4, 2, 1),
10            nn.BatchNorm2d(ngf*4), nn.ReLU(True),
11            # upsample to 16x16, 32x32, 64x64...
12            nn.ConvTranspose2d(ngf, nc, 4, 2, 1), nn.Tanh()
13        )
14     def forward(self, z): return self.main(z)
```

1.2.3 Discriminator

```
1 class Discriminator(nn.Module):
2     def __init__(self, nc, ndf):
3         super().__init__()
4         self.main = nn.Sequential(
5             # input: ncx128x128
6             nn.Conv2d(nc, ndf, 4, 2, 1), nn.LeakyReLU(0.2, True),
7             # downsample to 64x64, 32x32, 16x16...
8             nn.Conv2d(ndf*8, 1, 4, 1, 0)
9         )
10        self.sigmoid = nn.Sigmoid()
11    def forward(self, x):
12        feat = self.main(x)
13        out = self.sigmoid(feat.view(-1, 1))
14        return out
```

1.2.4 Dataset and Training

- terichomonas gallinae-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 terichomonas gallinae regions.

1.2.6 Results

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

2 Reconstruction Results of Initial Models

2.1 VAE Reconstructions

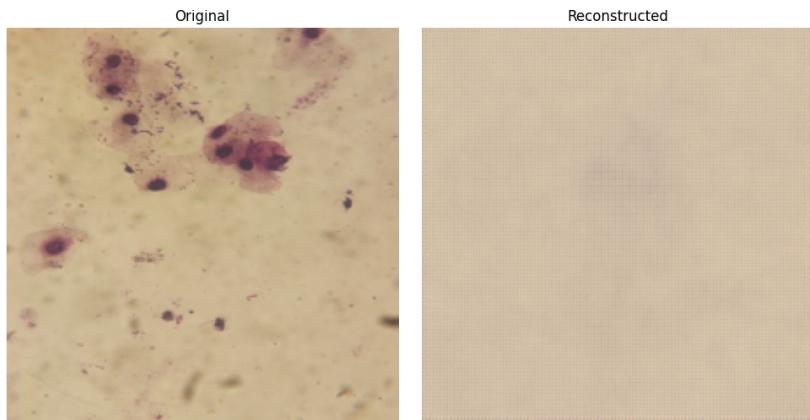


Figure 1: Example VAE reconstruction: original (left) vs. reconstructed (right). The VAE captures global texture but loses fine terichomonas gallinae details.

2.2 GAN Reconstructions

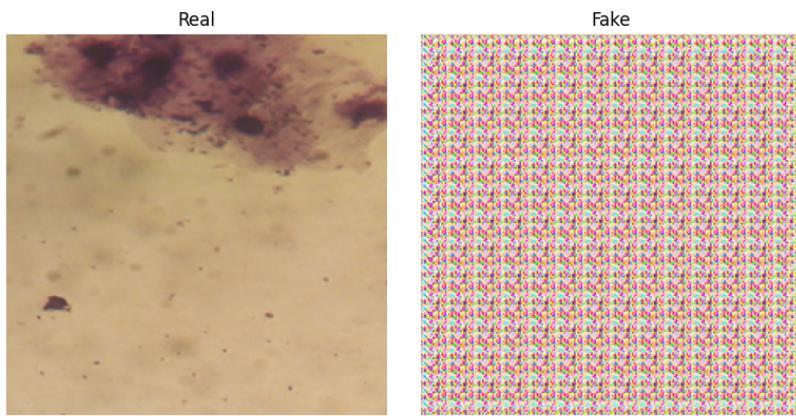


Figure 2: Example GAN reconstruction: generated image (right) compared to real sample (left). Background is realistic, terichomonas gallinae morphology is inconsistent.

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 $y \in \{0, 1\}$:

$$\text{BCE}(x, y) = -[y \log \sigma(x) + (1 - y) \log(1 - \sigma(x))], \quad (1)$$

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

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

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

```

1  class FocalLoss(torch.nn.Module):
2      def __init__(self, alpha=0.25, gamma=2.0, reduction='mean'):
3          super(FocalLoss, self).__init__()
4          self.alpha    = alpha
5          self.gamma   = gamma
6          self.reduction = reduction
7
8      def forward(self, inputs, targets):
9          # Flatten predictions and targets
10         inputs   = inputs.view(-1)
11         targets  = targets.view(-1)
12
13         # Compute per-example BCE loss (logits version)
14         BCE_loss = F.binary_cross_entropy_with_logits(
15             inputs, targets, reduction='none',
16             )
17         # Convert to probability scale
18         pt = torch.exp(-BCE_loss)
19         # Compute focal loss
20         F_loss = self.alpha * (1 - pt) ** self.gamma * BCE_loss
21
22         if self.reduction == 'mean':
23             return F_loss.mean()
24         else:
25             return F_loss.sum()
```

3.3 Data Preparation

3.3.1 Training Augmentations

```
1 train_transforms = transforms.Compose([
2     RandomRotation(degrees=30),
3     RandomHorizontalFlip(p=0.5),
4     RandomVerticalFlip(p=0.5),
5     RandomResizedCrop(size=(224,224), scale=(0.8,1.0)),
6     transforms.ToTensor(),
7     transforms.Normalize([0.5],[0.5])
8 ])
```

- `RandomRotation`: rotate image up to $\pm 30^\circ$.
- `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

```
1 val_transforms = transforms.Compose([
2     transforms.Resize((224,224)),
3     transforms.ToTensor(),
4     transforms.Normalize([0.5],[0.5])
5 ])
```

3.3.3 Datasets and DataLoaders

```
1 train_dataset = ImageFolder("./new_data/train", transform=train_transforms)
2 val_dataset   = ImageFolder("./new_data/dev",    transform=val_transforms)
3
4 train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
5 val_loader   = DataLoader(val_dataset,   batch_size=8, shuffle=False)
```

Classes are indexed as `{'no_terichomonas gallinaeed':0, 'terichomonas gallinaeed':1}`.

3.4 Model Architecture

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

```
model.fc = Sequential(Linear(2048,1)).
```

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

3.5 Training Loop

3.5.1 Hyperparameters

learning rate = 3×10^{-6} , epochs = 15, 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

```
1 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

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

Achieved F1 = 0.9032; confusion matrix:

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

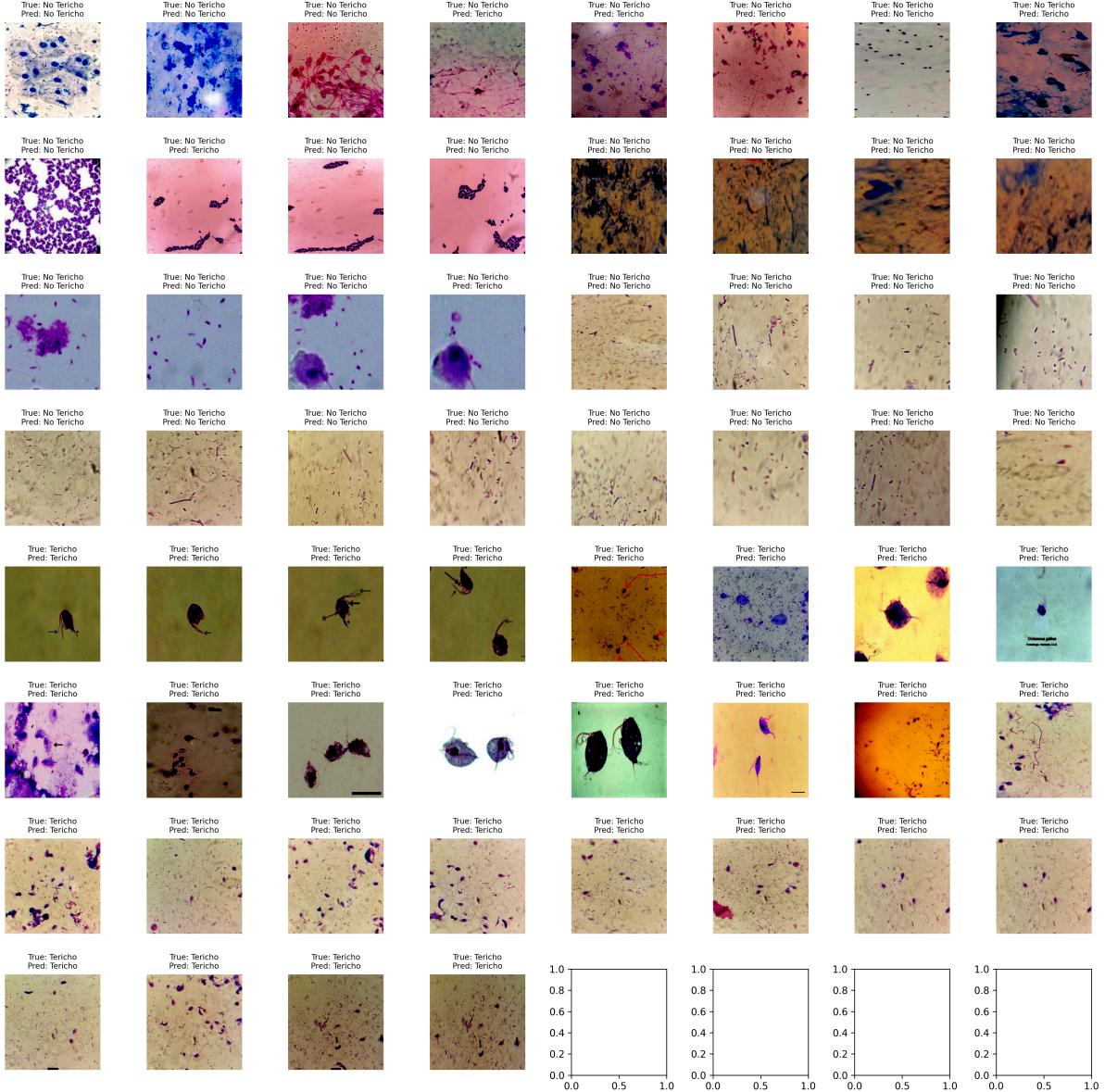
3.6.2 Threshold Optimization

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

```
1 thresholds = np.linspace(0.1, 0.9, 100)  
2 # ...  
3 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 terichomonas gallinae 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 terichomonas gallinae-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 terichomonas gallinae detection.

References

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