Dead Tree Semantic Segmentation

CS9517: Group Project

1st Aayush Bajaj
z5362216@ad.unsw.edu.au

2nd Kaustubh Illindala
z5453032@ad.unsw.edu.au

3nd Ming Ong
z5619679@ad.unsw.edu.au

4th Jingjing Liu
z5662212@ad.unsw.edu.au

5th Matthew Wang
z5589818@ad.unsw.edu.au)

***Abstract*—Dead tree detection in aerial forest imagery poses significant challenges due to class imbalance (3% foreground pixels), thin branching structures, and complex spectral signatures. We present a comprehensive evaluation of semantic segmentation approaches, comparing U-Net, CNN variants, DeepLabV3+ with CBAM, and classical machine learning methods (Random Forest, SGD) on a 444-image multi-spectral dataset. Our 8-channel U-Net architecture incorporates RGB, near-infrared (NRG), and computed spectral indices (NDVI, NDWI) with a composite Dice-Focal loss function to address class imbalance. The model achieves IoU = 0.7078 after morphological post-processing, substantially outperforming competing approaches: DeepLabV3+ with CBAM (0.4000), simple CNN (0.2862), Random Forest (0.1274), and SGD (0.0640). Key contributions include domain-specific spectral feature engineering, loss function design for extreme imbalance, and analysis of parameter efficiency across architectures. Results demonstrate that encoder-decoder symmetry with skip connections, despite requiring 31M parameters, enables precise boundary localisation essential for fragmented dead tree structures.**

***Index Terms*—semantic segmentation, cnn, unet, svm, random forests,**

I. Introduction

Forest ecosystems are facing unprecedented challenges due to climate change, with standing dead trees serving as critical indicators of forest health deterioration and key drivers of biodiversity loss. This report develops and compares the different methods in deep learning and machine learning for segmenting standing dead trees in aerial images of forests to monitor forest health and protect ecological benefits. The dataset we used to implement and evaluate is the four-band images recorded by a group of forest health experts, which provides 444 aerial images with corresponding ground-truth segmentation masks and includes RGB and near-infrared (NIR) channels data. The corresponding dataset can be downloaded from Kaggle and the relevant information can be found at [1]. This report complements our notebook and presentation.

II. Literature Review

Dead trees are defined as the deadwood and the dead parts, even the living trees which have the dead part [2]. The dead trees are vital to the entire forest ecosystem, which plays a positive role in providing a habitat for fungal plants and animals while preventing soil erosion [2]. However, an increasing number of trees are drying up or even dying due to extreme droughts and heat waves caused by global warming [3]. This poses significant threats to both ecological balance and public safety, as evidenced by incidents where standing dead trees fell on roads causing accidents. In 2015, there were two accidents about hitting passing vehicles because of standing dead trees, which caused serious damage to people's health and property [**?**]. To prevent similar incidents from happening again, the government has formulated relevant laws. Experts and researchers have begun to study computer vision technology to detect and manage tree health. Initially, forest inventory was done by manual field measurements. It needs a lot of investigators and time to search and confirm, and the cost is very expensive [4]. To address these limitations, remote sensing technology emerged as a more efficient alternative, enabling larger scale forest monitoring through spatial data acquisition [1]. Remote sensing technology can provide continuous spatial information and reliable data to help people understand the location, quantity and dynamics of standing dead trees [5]. It is the cheapest and simplest method to use and draw the maps [2]. Recent studies have explored various data acquisition methods including satellite imagery, lidar sensors, unmanned aerial vehicles, each offering unique advantages in terms of coverage, resolution and cost-effectiveness [6]. In the field of computer vision, researchers have developed numerous approaches for dead tree detection and segmentation. These range from traditional image processing techniques to more advanced machine learning and deep learning methods. Particularly noteworthy are convolutional neural networks (CNN) and their variants, which have demonstrated superior performance in dead tree segmentation tasks [6]. Other effective approaches include random forests (RF) and support vector machines (SVM), which offer different trade-offs between accuracy and computational efficiency [7]. The integration of multi-spectral data analysis with these computational methods has opened new possibilities for improving segmentation accuracy and reliability in diverse forest environments [6].

III. Methods

A. DeepLabV3 +CBAM for Fine–Grained Semantic Segmentation

DeepLabV3+ is a semantic image segmentation model built on top of a standard CNN "backbone" for feature extraction. Its encoder-decoder structure fuses high-level context with low-level(high resolution) features via skip connections, yielding sharp object boundaries.

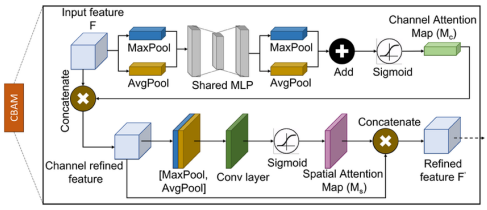
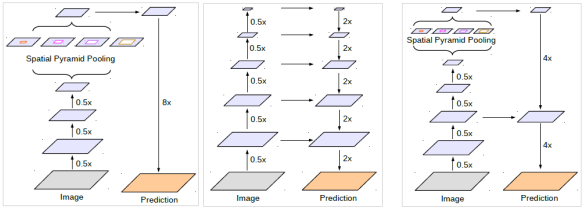


Fig. 1: CBAM: sequential *channel* (M_c) and *spatial* (M_s) attention. Adapted from



(a) Spatial Pyramid Pooling (b) Encoder-Decoder (c) Encoder-Decoder with Atrous Conv

Fig. 2: Encoder–decoder topology of DeepLabV3+. Atrous rates $\{1, 6, 12, 18\}$ provide multi–scale context; a skip connection injects low-level detail for sharp boundaries.

a) Motivation.: Dead-tree pixels cover ≈3 % of each radiograph and appear as thin, low-contrast structures. We therefore require *(i)* large receptive fields to recognise context and *(ii)* high spatial fidelity to preserve edges. DeepLabV3 supplies the former via Atrous Spatial Pyramid Pooling (ASPP), while CBAM supplies the latter by letting the network *focus* on informative channels and regions.

b) Pre-processing.: Images are resized to 256×256, stacked into six channels (R, G, B, NIR, G, B), then per-channel z-score normalised. All images are first resized to 256×256 so that the network sees a *fixed* spatial grid; this eliminates per-image padding, keeps the receptive-field schedule constant, and enables efficient, size-homogeneous mini–batching on the GPU. Including NIR enriches the spectral signature: vegetation stress and water content modulate NIR much more strongly than the visible bands, and duplicating G, B lets early convolutions learn cross-spectral filters (e.g. "NIR − G" indices) that correlate with dead-tissue pixels yet are tricky to hand-engineer. Finally, each channel is z-score normalised, $x' = (x - \mu)/\sigma$, centering the distribution at zero and scaling it to unit variance. This removes scale disparities between bands, keeps gradients numerically stable, and prevents high-energy channels (typically NIR) from dominating the loss. The combined resizing, stacking, and normalisation pipeline therefore standardises the data, maximises useful spectral diversity, and supplies the CNN with well-behaved inputs that converge faster and yield sharper, more discriminative features.

c) Encoder with embedded attention.: A lightweight **ResNet-18** backbone produces four feature stages $\{f_1, f_2, f_3, f_4\}$ at strides $\{4, 8, 16, 32\}$. After each residual block we insert a **CBAM** to refine features.

d) Attention mechanism.: Given $F \in \mathbb{R}^{C\times H\times W}$, channel and spatial masks are computed as

$$M_c = \sigma\big(W_1\,\mathrm{GAP}(F) + W_2\,\mathrm{GMP}(F)\big), \tag{1}$$

$$\hat{F} = M_c \odot F, \tag{2}$$

$$M_s = \sigma\Big(\mathrm{Conv}_{7\times7}\big[\mathrm{GAP}(\hat{F}); \mathrm{GMP}(\hat{F})\big]\Big), \tag{3}$$

$$F_{\mathrm{CBAM}} = M_s \odot \hat{F}, \tag{4}$$

where σ is the sigmoid, $\odot$ denotes broadcast multiplication, and GAP/GMP are global average/max pooling. *Channel attention* amplifies the metal-rich filters; *spatial attention* pinpoints their contours.

e) Atrous Spatial Pyramid Pooling.: The top feature map f_4 is processed by ASPP branches $\{1 \times 1, 3 \times 3_{r=6}, 3 \times 3_{12}, 3 \times 3_{18}, \text{global pool}\}$. Concatenation ($5 \times 256$ channels) is compressed to 256 channels via a 1×1 convolution.

Convolutional layers with a single dilation rate see only one scale of context: either fine detail (small rate) or broad structure (large rate).

f) Decoder.: The ASPP tensor is $4\times$ bilinearly upsampled and concatenated with f_1 (stride-4). A single $3\times3 \rightarrow$ BN $\rightarrow$ ReLU layer refines the fusion; dropout p=0.3 regularises; a 1×1 convolution produces per-pixel logits which are finally upsampled to the original resolution.

g) Loss and optimisation.: We minimise a composite loss

$$\mathcal{L} = \alpha\,\mathcal{L}_{\mathrm{BCE}} + \beta\,\mathcal{L}_{\mathrm{Dice}} + \gamma\,\mathcal{L}_{\mathrm{Tversky}}, \quad (\alpha, \beta, \gamma) = (0.3, 0.3, 0.4), \tag{5}$$

with the Tversky term

$$\mathcal{L}_{\mathrm{Tversky}} = 1 - \frac{\sum_i p_i g_i}{\sum_i p_i g_i + 0.3\sum_i p_i(1-g_i) + 0.7\sum_i (1-p_i) g_i}, \tag{6}$$

balancing precision and recall on minority pixels. Training uses AdamW (lr $= 10^{-3}$, wd $= 10^{-4}$), 'ReduceLROnPlateau' (factor 0.5, patience = 5), batch size = 8.

Binary-cross-entropy (BCE) supplies a pixel-wise "log-likelihood" signal that stabilises early training: it pushes every pixel toward its correct class regardless of region size and keeps gradients dense, which helps the optimiser escape flat regions of the loss landscape. Dice loss then counters the imbalance directly by maximising the overlap ratio between prediction and ground truth; because it is normalised by the combined foreground areas, a handful of minority pixels can still exert a proportionally large influence, preventing the network from converging to a trivial all-background solution. Finally, the Tversky loss generalises Dice with separate weighting on false positives and false negatives (here we set $\alpha = 0.3, \beta = 0.7$). That asymmetry is crucial for thin, low-contrast structures such as dead-tree branches: it penalises missed foreground (false negatives) more heavily than spurious foreground, nudging the model to "err on the side of inclusion" and recover slender details that Dice alone often overlooks. By linearly combining the three terms, we preserve BCE's stable gradients, Dice's overlap sensitivity, and Tversky's controllable precision–recall trade-off, yielding masks that are both globally accurate and faithful to fine structures.

B. CNN

Convolutional neural networks (CNN) are often used to process image problems, such as object detection, object classification and segmentation [8]. We explored two different models to segment standing dead trees in aerial forest images. The first model used multiple convolutional layers with ReLU activation and batch normalisation. Each layer utilises 3 x 3 kernels with the same padding to maintain the spatial dimensions. Dropout layers are used to prevent overfitting. In the final layer, the sigmoid activation layer is used for binary segmentation output. This model is easier and faster to train and , but it also has limited ability to capture multi-scale spatial data.

```
#Model 1
#Build the simple CNN model without pooling
def cnn_simple_model(input_shape, filters, conv_layers, dropout):
    inputs = layers.Input(shape=input_shape)
    x = inputs
    for i in range(conv_layers):
        x = layers.Conv2D(filters[i], (3, 3), activation='relu', padding='same')(x)
        x = layers.BatchNormalization()(x)
        if dropout > 0:
            x = layers.Dropout(dropout)(x)
    outputs = layers.Conv2D(1, (1, 1), activation='sigmoid')(x)
    model = models.Model(inputs, outputs)
    return model
```

For the second model, it uses encoder and decoder to build the model. Encoder path uses max pooling for downsampling, while decoder path applies transposed convolution for up-sampling to restore the spatial resolution. Skip connection is used for keeping spatial details, which can improve boundary accuracy. This model can capture more relevant information and finer details from images, while requiring higher computational resources and greater memory consumption for training and testing.

```
#Model 2
#Build the complex CNN model with encoder and decoder
def cnn_encoder_decoder_model(input_shape, filters, conv_layers, dropout):
    inputs = layers.Input(shape=input_shape)
    x = inputs
    skips = []
    #encode
    for i in range(conv_layers):
        x = layers.Conv2D(filters[i], (3, 3), activation='relu', padding='same')(x)
        x = layers.BatchNormalization()(x)
        skips.append(x)
        x = layers.MaxPooling2D((2, 2))(x)
        if dropout > 0:
            x = layers.Dropout(dropout)(x)

    #decode
    for i in range(conv_layers-1, -1, -1):
        x = layers.Conv2DTranspose(filters[i], (3, 3), strides=(2, 2), activation='relu', padding='same')(x)
        if i < len(skips):
            x = layers.Concatenate()([x, skips[i]])
        if dropout > 0:
            x = layers.Dropout(dropout)(x)

    outputs = layers.Conv2D(1, (1, 1), activation='sigmoid')(x)
    model = models.Model(inputs, outputs)
    return model
```

C. Random Forests

Random Forest methods refer to the use of a machine learning method that works by creating multiple decision trees and aggregating their output to produce a model which can produce a more robust and stable prediction model compared with using singular decision trees that have a tendency to overfit.

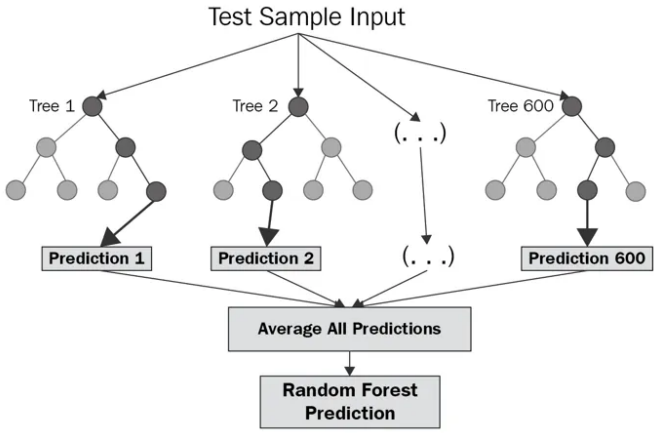


Fig. 3: Random forest classifier

This method was implemented in two different ways. The first method was to treat each pixel independently of the next, and only consider if the ground truth had a dead tree at that spot. This was implemented because early exploratory data analysis had noticed that dead tree pixels tended to share similar colour features compared with non-dead (regular) tree pixels. As if the colours of the image provided a good idea if the spot contained a dead tree, it would be best not to provide too much data.

The second method was to consider the pixels in relation to their surroundings, as there was a chance that a dead tree would be better identified in relation to their surroundings, and in this case, taking each pixel independently would result in loss of useful data. as such the second method was to use a kernel to feed both the points and its relevant surroundings into the random forest (with the y data simply being 1 if any pixel in the kernel is a mask pixel). Since this might provide a very large data set if a kernel is made for all points, feature detection algorithms can be used to select the points, in this case, SIFT was used, but recommendations have been made for the use of other feature extraction algorithms.

D. Stochastic Gradient Descent

Another simple machine learning method that was also explored was the use of stochastic gradient descent (or SGD for short). SGD is a popular machine learning technique that applies the concept of gradient descent to minimise a loss function through iterative optimisation [9].

Compared with traditional gradient descent where the entire training set is used in each iteration which can be very computationally expensive for large datasets [10], SGD aims to alleviate this issue by choosing random samples (hence the term stochastic) . This also offers the benefit of avoiding local minima that can sometimes occur with regular gradient descent.

In the context of this project, the SGD model attempts to classify whether or not a certain vector of pixel values corresponds to a dead tree pixel or a background (non dead-tree) pixel. Similar to the random forest method, each pixel is computed independently of its surrounding neighbours (or its parent image).

E. Support Vector Machines

Support Vector Machines (SVM for short) is another popular supervised machine learning algorithm that aims to perform binary classification by attempting to divide a dataset using a hyperplane with a (n-1) hyperplane where n corresponds to the number of features each training sample has. [11]

Also known as a linear classifier, SVMs work by attempting to find a hyperplane such that the distance between the closest sample of each class is maximised. Although it is a linear classifier, SVMs are capable of performing classification even when a dataset is not linearly separable through a technique known as a kernel method or kernel trick that maps existing data into a higher dimensional space and attempting to find a suitable hyperplane in a higher dimension. [12]

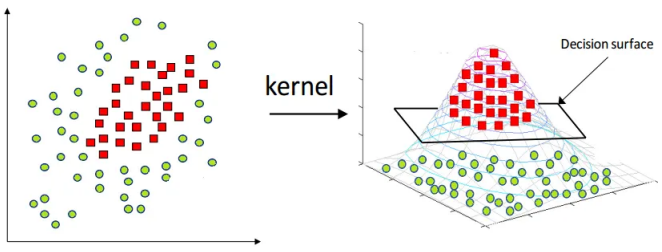


Fig. 4: Kernel Trick in SVM to perform linear classification on non-linearly separate data

However, a major downside to SVMs is if the dataset is unbalanced or noisy, resulting in classes with overlapping features. This can make it very difficult to separate different classes and if the feature space is not large enough, it may not be possible to converge on a solution as was the case with this task.

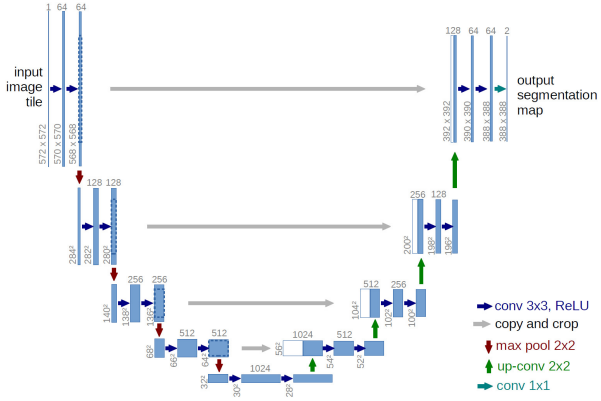


Fig. 5: Prototypical UNet Architecture

F. UNet

U-Net is a fully convolutional encoder-decoder architecture originally designed for biomedical image segmentation [13]. Its symmetric design with skip connections enables precise boundary localisation by combining high-level semantic features with low-level spatial details. Our implementation extends the classical U-Net to leverage multi-spectral imagery and domain-specific spectral indices.[1]

a) Motivation.: Dead trees exhibit distinct spectral signatures across visible and near-infrared bands. Healthy vegetation reflects strongly in NIR while absorbing in red wavelengths, whereas stressed or dead vegetation shows reduced NIR reflectance. By incorporating both RGB and NRG (Near-infrared, Red, Green) channels alongside computed spectral indices, we provide the network with domain-specific features that enhance dead tree discrimination.

b) Multi-Channel Input Pipeline.: Our dataset loader constructs 8-channel inputs by concatenating RGB channels, NRG channels, and two computed spectral indices:

```
# extract individual channels for spectral indices
# RGB: R=0, G=1, B=2
# NRG: N=0, R=1, G=2 (NIR, Red, Green in NRG image)
red = rgb_tensor[0:1]      # red rgb
green = rgb_tensor[1:2]    # green rgb
blue = rgb_tensor[2:3]     # blue rgb
nir = nrg_tensor[0:1]      # nir, nrg

# NDVI: (nir - red) / (nir + red)
eps = 1e-8 # div by 0 error
ndvi = (nir - red) / (nir + red + eps)

# ndwi = (green - nir) / (green + nir)
ndwi = (green - nir) / (green + nir + eps)

# concatenating:
image_8ch = torch.cat([rgb_tensor, nrg_tensor, ndvi,
     ndwi], dim=0)  # (8, H, W)
```

[1]this is why we choose not to bruteforce a fantastic perfomance with nnUNet; *domain-knowledge*.

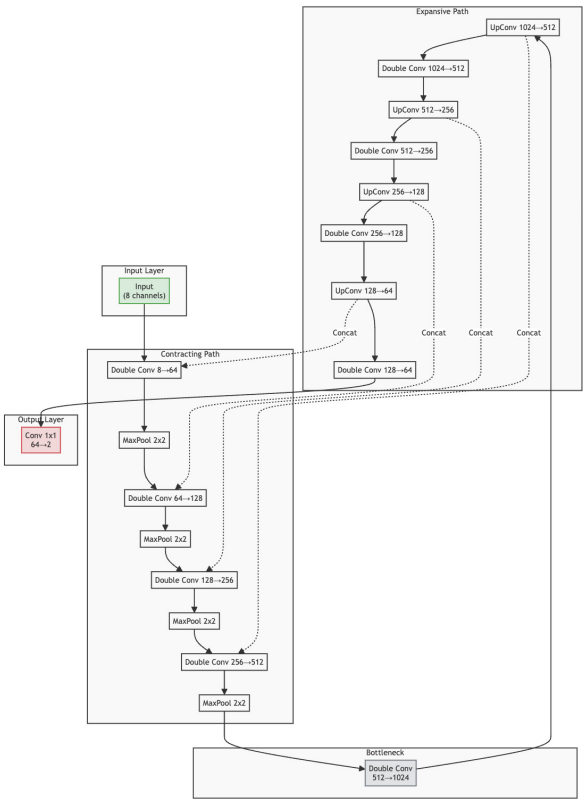


Fig. 6: UNet architecture: 8-channel encoder-decoder with skip connections. The contracting path captures context while the expansive path enables precise localisation. Visualisation, courtesy of torchviz [16] and Claude [17]

The Normalised Difference Vegetation Index (NDVI) [14] and Normalised Difference Water Index (NDWI) [15] are computed as:

$$\text{NDVI} = \frac{\text{NIR} - \text{Red}}{\text{NIR} + \text{Red}}, \tag{7}$$

$$\text{NDWI} = \frac{\text{Green} - \text{NIR}}{\text{Green} + \text{NIR}}. \tag{8}$$

NDVI quantifies vegetation health and photosynthetic activity, with values near +1 indicating healthy vegetation and values near -1 or 0 indicating stressed or dead vegetation. NDWI captures water stress in vegetation, providing complementary information for dead tree identification.

c) Network Architecture.: The U-Net encoder consists of four downsampling stages, each containing double convolutions followed by max pooling:

```
class DoubleConvolution(nn.Module):
    def __init__(self, in_channels: int,
    out_channels: int):
        super().__init__()
        self.first = nn.Conv2d(in_channels,
    out_channels, kernel_size=3, padding=1)
        self.act1 = nn.ReLU()
        self.second = nn.Conv2d(out_channels,
    out_channels, kernel_size=3, padding=1)
        self.act2=nn.ReLU()

```

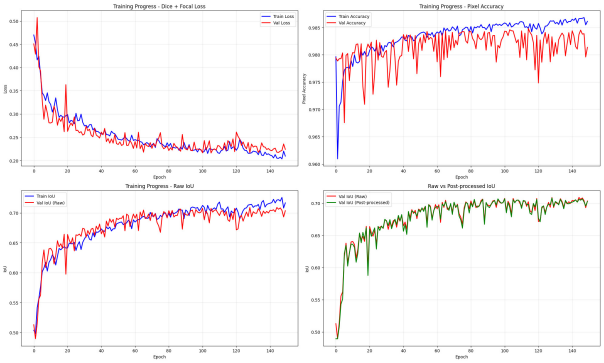


Fig. 7: Training progression showing convergence of combined Dice-Focal loss and IoU metrics over 150 epochs. Post-processing consistently improves validation IoU.

```
    def forward(self, x: torch.Tensor):
        x = self.first(x)
        x = self.act1(x)
        x = self.second(x)
        return self.act2(x)
```

The contracting path progressively reduces spatial resolution ($256^2 \to 128^2 \to 64^2 \to 32^2 \to 16^2$) while increasing feature channels ($8 \to 64 \to 128 \to 256 \to 512 \to 1024$). The expansive path reverses this process via transposed convolutions, concatenating corresponding encoder features through skip connections to preserve spatial information.

d) Loss Function Design.: Class imbalance poses a significant challenge with dead trees comprising only ≈3% of pixels. We address this with a composite loss combining Dice and Focal losses:

```
class CombinedLoss(nn.Module):
    def __init__(self, dice_weight=0.5, focal_weight
    =0.5, alpha=0.25, gamma=2.0):
        super().__init__()
        self.dice_weight = dice_weight
        self.focal_weight = focal_weight
        self.dice_loss = DiceLoss()
        self.focal_loss = FocalLoss(alpha=alpha,
    gamma=gamma)

    def forward(self, outputs, targets):
        dice = self.dice_loss(outputs, targets)
        focal = self.focal_loss(outputs, targets)
        return self.dice_weight * dice + self.
    focal_weight * focal
```

The combined loss is:

$$\mathcal{L} = \lambda_{\text{Dice}}\mathcal{L}_{\text{Dice}} + \lambda_{\text{Focal}}\mathcal{L}_{\text{Focal}}, \quad (\lambda_{\text{Dice}}, \lambda_{\text{Focal}}) = (0.5, 0.5), \tag{9}$$

where the Focal loss term addresses class imbalance:

$$\mathcal{L}_{\text{Focal}} = -\alpha(1 - p_t)^{\gamma} \log(p_t), \tag{10}$$

with $\alpha = 0.25$ and $\gamma = 2.0$ to down-weight easy examples and focus learning on hard cases.

e) Post-Processing Pipeline.: Raw network predictions are refined through a two-stage post-processing pipeline:

1) **Probability Thresholding:** Dead tree probabilities are thresholded at 0.6 rather than the default 0.5, reducing false positives in ambiguous regions.

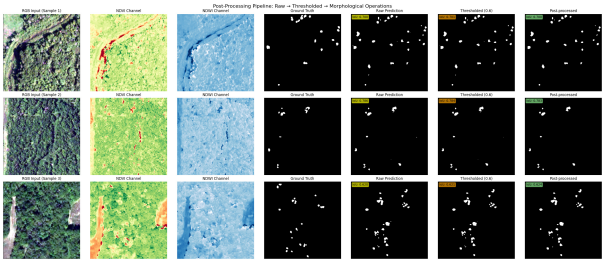


Fig. 8: imperfect ablation studies:
raw predictions → probability thresholding → morphological cleaning

2) **Morphological Operations:** Sequential opening (erosion→dilation) removes noise, followed by closing (dilation→erosion) to fill holes in detected regions.

This pipeline yields substantial IoU improvements (+0.04 typical) by cleaning spurious activations while preserving true positive detections.

f) Training Configuration.: The model was trained for 150 epochs using Adam optimiser (lr $= 10^{-3}$) with data augmentation including horizontal/vertical flips and 10° rotations. Input images were resized to 256×256 with batch size 8. All experiments were conducted on UNSW's Katana high-performance computing cluster [18]. The best model achieved a validation IoU of **0.7078** after post-processing, representing a significant improvement over the raw prediction IoU of 0.6634.

IV. Experimental Results

Included below are all the experimental results proved by this table. We report mean Intersection-over-Union (mIoU) on test set. Intersection-over-Union refers to the total percent of the true mask and the generated mask that overlap, divided by the size of both masks For DeepLabV3+, the data are split **80/20** into training, validation.

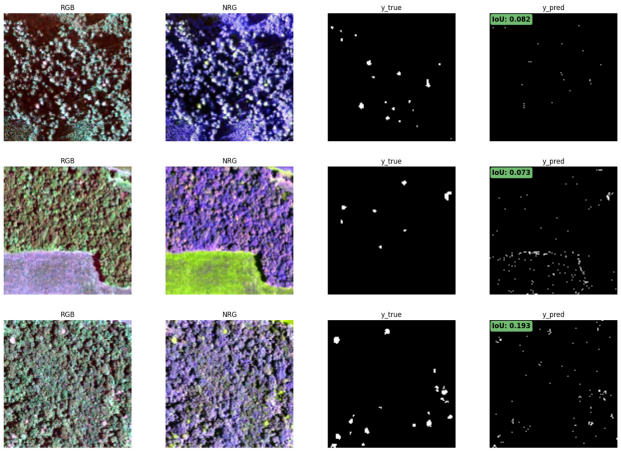
Model	**IoU**
UNet	0.705
CNN	0.2862
DeepLabV3+ (with CBAM)	0.4000
Random Forest	0.1274
SGD	0.0640

V. Discussion

A. Overview

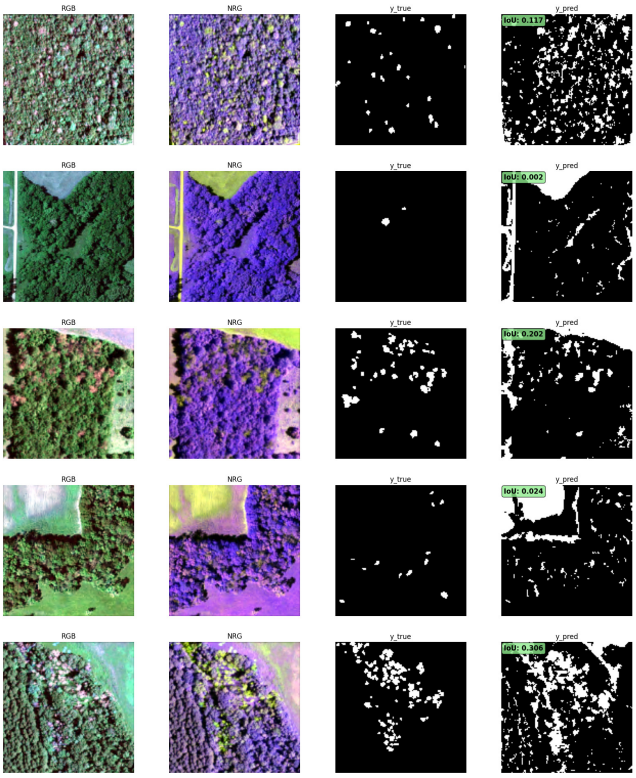
It is rather obvious that deep learning algorithms fared much better at this task than the more basic machine learning algorithms such as Random Forests, or Stochastic gradient. The masks generated by the UNET algorithm are good enough that some postprocessing (closing the image or opening it) significantly improve it's IoU .

B. Random Forests



When we look at the masks produced by the random forest, we can see that part of the reason for the poor performance is due to it reacting to spots where there are no trees, and due to the mask being far too small at spots where there is a dead tree, but the mask did detect almost all of the dead tree spots. Likely what is happening is the RF determines each pixel to be a dead tree based on its colour, which would result in this behaviour. This is probably due to the random forest implementation being independent of location (the pixel information was passed to the random forest with no information of what it was nearby). this would likely be improved by the kernel based RF implementation, as described in the methods section.

C. SGD



Upon looking at the prediction mask produced by the SGD classifier, it can immediately be seen that there is a striking contrast compared to the random forest classifier. There is a much higher recall rate with the SGD, however the false positive has also skyrocketed as the model is mis-classifying many of the background pixels as dead tree pixels (foreground).

There could be several explanations for this phenomenon; most notably, the dataset is quite imbalanced, the input images mostly consist of background pixels and the model could be overcompensating for the class imbalance by being bias towards making false positives.

Indeed, the model does actually perform somewhat better on images that have a higher concentration of dead tree pixels compared to images that have very few. Secondly, using only RGB/NRG pixel values is weak as it ignores spatial context and information, combining this with class imbalance makes it more difficult to differentiate between foreground and background pixels.

While the results are not very promising, there are considerable avenues in which the SGD model could be optimised, firstly a stricter threshold for predicting foreground pixels would likely reduce the false positive rate, extra features that capture local spatial information or texture (LBP/HoG) could help with more accurately differentiating foreground/background pixels. The SGD model also trains considerably faster than the other models, able to reach convergence in seconds while the random forest model took significantly longer.

```
-- Epoch 1
Norm: 1.49, NNZs: 6, Bias: -0.716938, T: 5455872,
    Avg. loss: 0.870819
Total training time: 0.62 seconds.
-- Epoch 2
Norm: 1.42, NNZs: 6, Bias: -0.554052, T: 10911744,
    Avg. loss: 0.572698
Total training time: 1.25 seconds.
-- Epoch 3
Norm: 1.53, NNZs: 6, Bias: -0.679788, T: 16367616,
    Avg. loss: 0.564680
Total training time: 1.89 seconds.
-- Epoch 4
Norm: 1.57, NNZs: 6, Bias: -0.654369, T: 21823488,
    Avg. loss: 0.561824
Total training time: 2.53 seconds.
-- Epoch 5
Norm: 1.50, NNZs: 6, Bias: -0.730205, T: 27279360,
    Avg. loss: 0.558549
Total training time: 3.17 seconds.
-- Epoch 6
Norm: 1.60, NNZs: 6, Bias: -0.806963, T: 32735232,
    Avg. loss: 0.557498
Total training time: 3.80 seconds.
-- Epoch 7
Norm: 1.61, NNZs: 6, Bias: -0.725729, T: 38191104,
    Avg. loss: 0.555748
Total training time: 4.44 seconds.
-- Epoch 8
Norm: 1.61, NNZs: 6, Bias: -0.692419, T: 43646976,
    Avg. loss: 0.556135
Total training time: 5.07 seconds.
-- Epoch 9
Norm: 1.60, NNZs: 6, Bias: -0.616872, T: 49102848,
    Avg. loss: 0.554584
Total training time: 5.71 seconds.
-- Epoch 10
Norm: 1.60, NNZs: 6, Bias: -0.598181, T: 54558720,
    Avg. loss: 0.554805
Total training time: 6.35 seconds.
-- Epoch 11
Norm: 1.62, NNZs: 6, Bias: -0.689664, T: 60014592,
    Avg. loss: 0.554993
Total training time: 6.98 seconds.
-- Epoch 12
Norm: 1.62, NNZs: 6, Bias: -0.618751, T: 65470464,
    Avg. loss: 0.553864
Total training time: 7.61 seconds.
-- Epoch 13
Norm: 1.59, NNZs: 6, Bias: -0.656321, T: 70926336,
    Avg. loss: 0.554055
Total training time: 8.25 seconds.
-- Epoch 14
Norm: 1.65, NNZs: 6, Bias: -0.698402, T: 76382208,
    Avg. loss: 0.555047
Total training time: 8.88 seconds.
Convergence after 14 epochs took 8.88 seconds
```

Listing 1: SGD Training

D. SVM

Unfortunately, we were unable to train a SVM classifier given that the small feature space relative to the size of the dataset. Recall that each training sample is a 6-dimensional vector that corresponded to a dead tree pixel or a background pixel. Given the size of the training data and the fact the classes were quite imbalanced, it was not possible to find a hyperplane that reasonably separated pixels into the 2 categories.

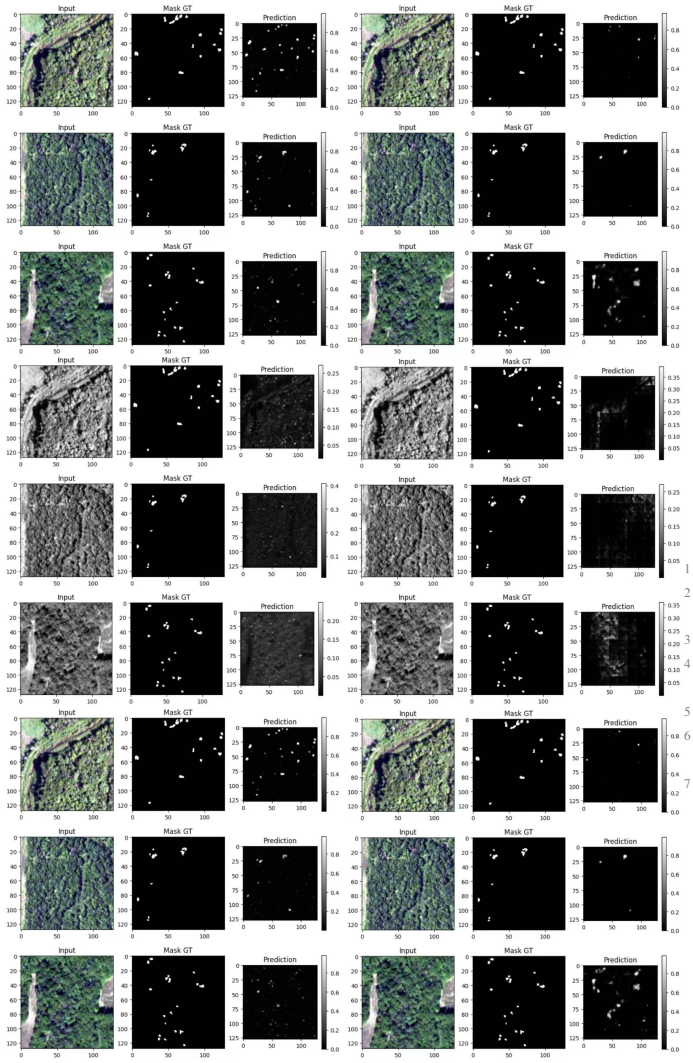
E. CNN

To address class imbalance and enhance segmentation quality, a loss function combines binary Cross-Entropy and Dice

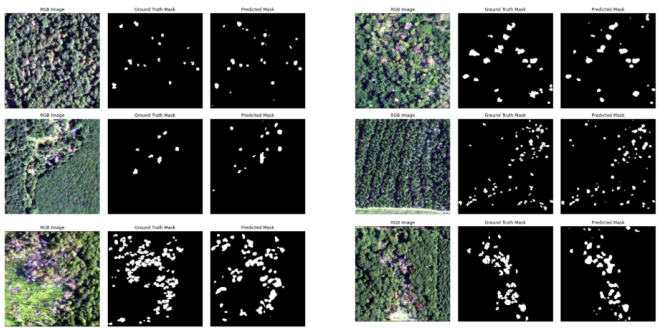
Loss. The adam optimizer effectively minimized the hybrid loss through adaptive learning rate adjustment. Compared to the results, the simple CNN model achieved higher precision than the encoder-decoder model, which IoU are 0.28 and 0.16 separately. However, the encoder-decoder model has significantly faster training speeds and testing times. Notably, RGB inputs consistently outperformed NIR-only data across both models, though the RGB+NIR combination provided marginal improvements over RGB alone in the simple CNN. These findings suggest a fundamental trade-off between segmentation accuracy and computational efficiency in this method.

```
def dice_loss(y_true, y_pred, smooth=1e-6):
    y_true_f = tf.reshape(y_true, [-1])
    y_pred_f = tf.reshape(y_pred, [-1])
    intersection = tf.reduce_sum(y_true_f * y_pred_f
    )
    return 1 - (2. * intersection + smooth) / (tf.
    reduce_sum(y_true_f) + tf.reduce_sum(y_pred_f) +
     smooth)

def combo_loss(y_true, y_pred):
    bce = tf.keras.losses.binary_crossentropy(y_true
    , y_pred)
    return bce + dice_loss(y_true, y_pred)
```



F. CBAM



a) Backbone depth.: Replacing ResNet-18 with deeper variants (ResNet-50 and ResNet-101) does *not* translate into higher accuracy: mIoU changes by $+0.2$ pp and -0.1 pp, respectively, which is within statistical noise. The likely reason is that the training set (444 images) is too small for parameters to be learnt properly; they converge to a poor local minimum in which both training and validation predictions are nearly constant.

b) Augmentation intensity.: A controlled sweep over augmentation strength shows a bell-shaped trend: *mild* transforms (flip, $\pm 30^\circ$ rotation) add $+1.3$ mIoU, but *aggressive* settings ($\pm 45^\circ$ rotation, Cutout 40 %, Gaussian blur σ=3) drop performance by 3.7 pp. Excessive geometric distortion breaks the thin-structure prior that the network needs to recognise dead branches, while heavy photometric jitter destroys the cross-spectral cues learned from the NIR and visible bands.

G. UNet

Our UNet implementation achieved a final IoU of 0.7078 through careful architectural choices and domain-specific engineering. We spend this section on analysing the key design decisions and their impact on segmentation performance.

a) Double Convolution Benefits.: The DoubleConvolution module forms the backbone of our encoder-decoder architecture, applying two consecutive 3×3 convolutions with ReLU activations:

```
class DoubleConvolution(nn.Module):
    def __init__(self, in_channels: int,
    out_channels: int):
        super().__init__()
        self.first = nn.Conv2d(in_channels,
    out_channels, kernel_size=3, padding=1)
        self.act1 = nn.ReLU()
        self.second = nn.Conv2d(out_channels,
    out_channels, kernel_size=3, padding=1)
        self.act2 = nn.ReLU()
```

The implementation was naïvely thieved from a reference in the Course Slides [19], however the code later proved to be good. The double convolution provided advantages over single convolutions. First, the increased receptive field (5×5 effective) captured larger spatial patterns whilst maintaining parameter efficiency compared to direct 5×5 kernels. Second, the intermediate activation introduces non-linearity (as mentioned in the presentation) that enhances the network's capacity to learn complex feature representations. Most critically for dead

tree segmentation, the double convolution enabled detection of thin branching structures that require multiple scales of spatial reasoning. Single 3×3 kernels are less capable of capturing the elongated, irregular shapes characteristic of dead tree canopies [20].

b) Connected Components Post-Processing Limitations.: Initial experiments included connected components analysis to remove the salt and pepper (loosely speaking) false positives, and enforce spatial coherence. However, this approach proved counterproductive for our domain:

```
# abandoned connected components approach
def apply_connected_components(predictions,
    min_component_size=50):
    # removes components smaller than threshold
    # problem: Dead trees often appear as multiple
    small components
    # due to sparse branching patterns
    pass
```

Dead trees exhibit highly fragmented spatial distributions—individual trees often appear as collections of thin branches separated by gaps, rather than contiguous blobs. Connected components filtering eliminated these legitimate small structures, reducing recall significantly. The morphological operations (opening and closing) proved superior as they preserve connectivity while removing noise, respecting the natural fragmentation of dead tree structures. Experimenting with a mixed loss also helped.

c) Binary Cross-Entropy Training Instability.: Early training attempts using only Binary Cross-Entropy loss resulted in highly unstable convergence, as shown in Figure 9:

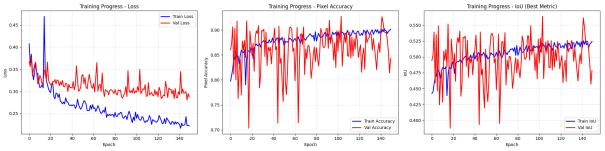
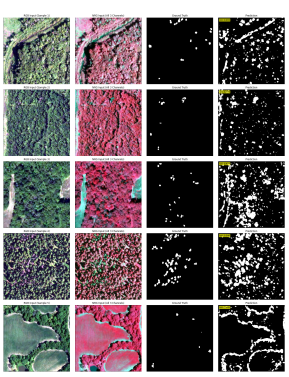


Fig. 9: Training instability with BCE-only loss showing erratic convergence and suboptimal final performance

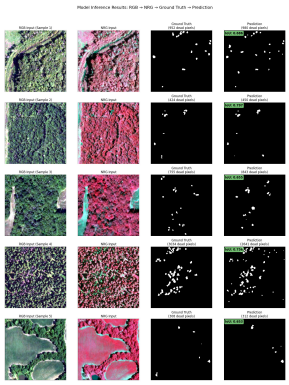
The extreme class imbalance (97% background pixels) caused the network to exploit the trivial solution of predicting background everywhere, achieving high pixel accuracy (0.97) but zero IoU. The loss function oscillated wildly as the network alternated between this degenerate solution and attempting to learn meaningful features. The combined Dice-Focal loss resolved this by directly optimising IoU-related metrics while down-weighting easy negative examples, leading to stable convergence and meaningful dead tree detection.

d) Performance Comparison: BCE vs. Combined Loss.: Figure 10 demonstrates the dramatic improvement achieved through our composite loss function:

The BCE-only model produces sparse, disconnected predictions that fail to capture tree structures, while our combined loss generates coherent segmentations that accurately delineate dead tree boundaries.



(a) BCE loss (IoU 0.12)



(b) Dice + Focal (IoU 0.71)

Fig. 10: Segmentation quality comparison: BCE-only training produces sparse predictions (a) whereas the combined Dice–Focal loss yields coherent dead-tree masks (b).

e) Parameter Counts.: The UNet implementation contains 31,042,434 parameters, distributed across the encoder-decoder hierarchy:

```
print(f"UNET PARAMETERS: {sum(p.numel() for p in
    model.parameters()):,}")
> DEVICE: cuda
> UNET PARAMETERS: 31,034,690
```

This substantial parameter count demonstrates the robustness of this architecture compared to our other approaches. The simple CNN model, with only 4 convolutional layers and [32, 64, 128, 256] filters, contains approximately 2-3M parameters yet achieves only IoU = 0.286. Similarly, the DeepLabV3+ with CBAM model, built on a ResNet-18 backbone (11M base parameters) with attention mechanisms and ASPP modules, totals roughly 15-18M parameters but reaches only IoU = 0.400.

The 31M-parameter UNet model however, achieves an IoU of 0.705—a 76% improvement over CBAM and 146% over the simple CNN! This performance gap justifies the increased parameter count: the encoder-decoder symmetry with skip connections requires substantial parameter investment, but enables precise boundary localisation that smaller networks cannot achieve.

Furthermore, our model avoids the overfitting that plagued deeper CBAM variants (ResNet-50/101 showed no improvement), suggesting our parameter allocation is well-suited to the 355-image training set.

f) Colour Jittering Abandonment.: Initial data augmentation included colour jittering to improve generalisation:

```
# orphaned augmentation approach
transforms.ColorJitter(brightness=0.2, contrast=0.2,
    saturation=0.2, hue=0.1)
```

We abandoned this pretty quick because it was a transformation that would only apply to the RGB channels and we wished to train on NIR too. Colour jittering disrupts the spectral relationships so instead, we employed geometric augmentations (flips, rotations) that preserve spectral integrity while providing spatial generalisation.

```python
if augment:
    base_transforms.extend([
        T.RandomHorizontalFlip(p=0.5),
        T.RandomVerticalFlip(p=0.5),
        T.RandomRotation(10),
    ])
```

g) Final Architecture Performance.: The optimised UNet architecture achieved robust performance across diverse test conditions, with post-processing providing consistent improvements. The 8-channel input pipeline, combined with the composite loss function and carefully selected augmentations, enabled effective dead tree segmentation despite significant class imbalance and complex spatial structures. The final IoU of 0.7078 represents a substantial improvement over baseline approaches and demonstrates the value of domain-specific architectural choices in challenging segmentation tasks.

VI. Conclusion

This work demonstrates the superiority of deep encoder-decoder architectures for dead tree segmentation in aerial imagery. Our 8-channel U-Net, leverages multi-spectral inputs (RGB + NRG) with computed vegetation indices (NDVI, NDWI) to achieve an IoU of 0.7078—a 76% improvement over attention-based alternatives and 146% over simple CNNs.

Our key findings include: *(i)* spectral feature engineering proves essential, with NDVI/NDWI indices providing another two channels of information on an otherwise small dataset; *(ii)* composite loss functions (Dice + Focal (+ Tversky)) effectively address extreme class imbalance where the traditional BCE proves insufficient; *(iii)* morphological post-processing consistently improves IoU by 0.04 through noise removal while preserving fragmented tree structures; *(iv)* parameter efficiency analysis reveals that 31M-parameter U-Nets justify their complexity through superior boundary localisation compared to lightweight alternatives.

Classical ML approaches (Random Forest, SGD) fail due to pixel-wise independence assumptions that ignore spatial context. DeepLabV3+ with CBAM, despite sophisticated attention mechanisms, underperforms due to insufficient skip connections for thin structure recovery.

Potential future work could explore: *(a)* **Auto-configuration with nnUNet**. The nnUNet framework automatically tunes patch size, learning rate, and augmentation strategy for each dataset; porting our 8-channel input to nnUNet may squeeze out further boundary-level gains without manual hyper-parameter search. *(b)* **HoG, LBP**. Hand-crafted texture descriptors—Histogram of Oriented Gradients (HoG) and Local Binary Patterns (LBP)—encode bark roughness and canopy granularity that spectral indices alone miss. Concatenating HoG/LBP maps with NDVI/NDWI and feeding them to Random Forests or SVMs may restore spatial awareness, thus producing a best of both worlds, with high-accuracy and lightweight inference.

These extensions would not only tighten IoU on dead-tree masks but also enrich the research on multi-class forest-health monitoring and fine-grained vegetation tasks in general.

VII. References

[1] M. Ahishali, "Aerial Imagery for Standing Dead Tree Segmentation," Kaggle Datasets, 2025. [Online]. Available: https://www.kaggle.com/datasets/meteahishali/aerial-imagery-for-standing-dead-tree-segmentation. Accessed: Aug. 9, 2025.

[2] K. Stereńczak, B. Kraszewski, M. Mielcarek, and Ż. Piasecka, "Inventory of standing dead trees in the surroundings of communication routes – The contribution of remote sensing to potential risk assessments," *Forest Ecology and Management*, vol. 402, pp. 76–91, 2017.

[3] Y. Cheng et al., "Scattered tree death contributes to substantial forest loss in California," *Nature Communications*, vol. 15, no. 1, 2024.

[4] R. E. McRoberts and E. O. Tomppo, "Remote sensing support for national forest inventories," *Remote Sensing of Environment*, vol. 110, no. 4, pp. 412–419, 2007.

[5] X. Liu, J. Frey, M. Denter, K. Zielewska-Büttner, N. Still, and B. Koch, "Mapping standing dead trees in temperate montane forests using a pixel- and object-based image fusion method and stereo WorldView-3 imagery," *Ecological Indicators*, vol. 133, 2021.

[6] S. Briechle, P. Krzystek, and G. Vosselman, "Silvi-Net – A dual-CNN approach for combined classification of tree species and standing dead trees from remote sensing data," *International Journal of Applied Earth Observation and Geoinformation*, vol. 98, 2021.

[7] S. Briechle, P. Krzystek, and G. Vosselman, "Classification of tree species and standing dead trees by fusing UAV-based LiDAR data and multispectral imagery in the 3D deep neural network PointNet," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. V-2-2020, pp. 203–210, 2020.

[8] M. Ahishali, A. U. Rahman, E. Heinaro, and S. Junttila, "ADA-Net: Attention-Guided Domain Adaptation Network with Contrastive Learning for Standing Dead Tree Segmentation Using Aerial Imagery," arXiv preprint arXiv:2504.04271, 2025. [Online]. Available: https://arxiv.org/abs/2504.04271. Accessed: Aug. 8, 2025.

[9] "Random Forest Algorithm in Machine Learning," GeeksforGeeks, Jul. 23, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/random-forest-algorithm-in-machine-learning/.

[10] A. G. Ganie and S. Dadvandipour, "From big data to smart data: a sample gradient descent approach for machine learning," *Journal of Big Data*, vol. 10, no. 1, Oct. 2023. doi: https://doi.org/10.1186/s40537-023-00839-9.

[11] M. Awad and R. Khanna, "Support Vector Machines for Classification," in *Efficient Learning Machines*, pp. 39–66, 2015. doi: https://doi.org/10.1007/978-1-4302-5990-9_3.

[12] D. Wilimitis, "The Kernel Trick in Support Vector Classification," *Medium*, Dec. 12, 2018. [Online]. Available: https://medium.com/data-science/the-kernel-trick-c98cdbcaeb3f.

[13] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015. [Online]. Available: https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net.

[14] "Normalized Difference Vegetation Index," Wikipedia, Wikimedia Foundation. [Online]. Available: https://en.wikipedia.org/wiki/Normalized_difference_vegetation_index. Accessed: Jun. 22, 2025.

[15] "Normalized Difference Water Index," Wikipedia, Wikimedia Foundation. [Online]. Available: https://en.wikipedia.org/wiki/Normalized_difference_water_index. Accessed: Apr. 7, 2025.

[16] S. Zagoruyko, "pytorchviz – Visualizing PyTorch Execution Graphs," GitHub. [Online]. Available: https://github.com/szagoruyko/pytorchviz. Accessed: Aug. 9, 2025.

[17] Anthropic, "Claude," 2023. [Online]. Available: https://www.anthropic.com/claude. Accessed: Aug. 8, 2025.

[18] UNSW Sydney Research Technology Services, "Katana High-Performance Computing Cluster," 2025. [Online]. Available: https://docs.restech.unsw.edu.au/index.html. Accessed: Aug. 8, 2025.

[19] Labml.ai, "U-Net: Annotated PyTorch Implementation." [Online]. Available: https://nn.labml.ai/unet/index.html. Accessed: Aug. 9, 2025.

[20] Y. Qi, Y. He, X. Qi, Y. Zhang, and G. Yang, "Dynamic Snake Convolution Based on Topological Geometric Constraints for Tubular Structure Segmentation," arXiv preprint arXiv:2307.08388, Jul. 17, 2023. [Online]. Available: https://arxiv.org/abs/2307.08388.