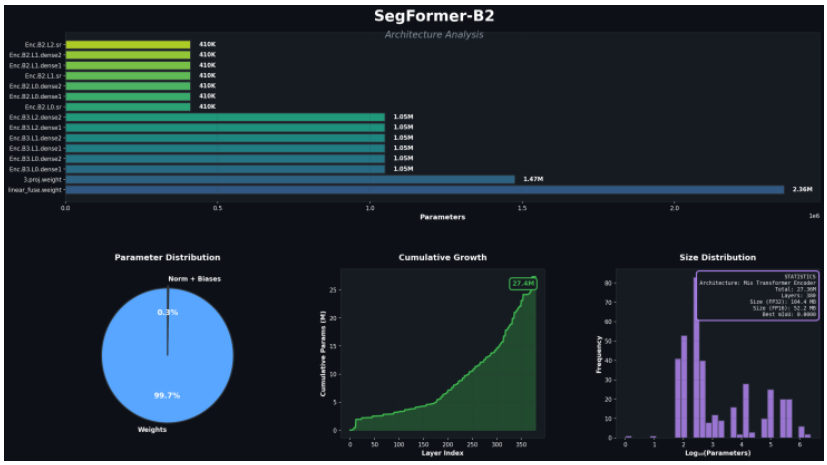
Krackheads: Offroad Semantic Scene Segmentation Challenge

Team Members: Riddham, Shikhar, Tushar, Vedansh

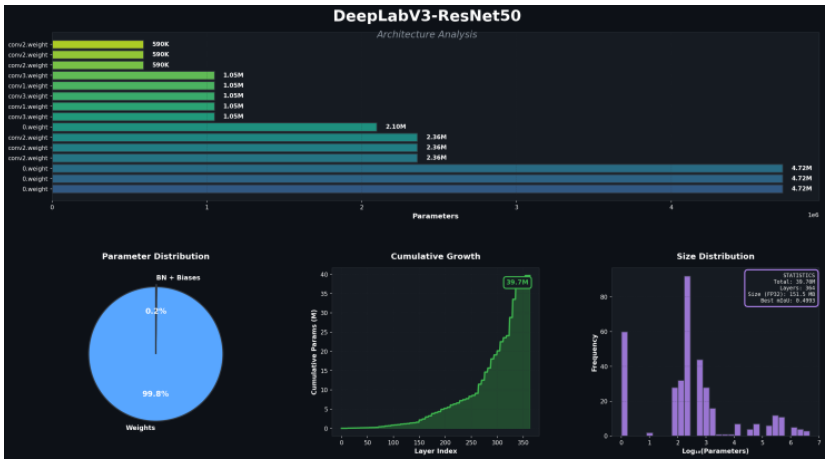
Tagline: Mapping the Desert – Pixel by Pixel

This report summarizes our participation in Duality AI's Offroad Autonomy Segmentation challenge. The goal is to train a robust semantic segmentation model on synthetic desert imagery and evaluate its generalization on novel offroad scenes. We leverage deep learning models to label each pixel across ten classes—trees, lush bushes, dry grass, dry bushes, ground clutter, flowers, logs, rocks, landscape, and sky—enabling unmanned vehicles to navigate complex terrains. This eight-page report follows the structure outlined by the organizers and provides a detailed account of our data pipeline, model architecture, training strategy, challenges, improvements, and final ensemble using knowledge distillation.



SegFormer-B2 uses a hierarchical Mix Transformer encoder with four stages and a lightweight MLP decoder for semantic segmentation. With ~25M parameters, it eliminates positional encodings through Mix-FFN layers, achieving resolution-robustness and efficient real-time performance.

DeepLabV3-ResNet50 combines a ResNet50 encoder with Atrous Spatial Pyramid Pooling (ASPP) to capture multi-scale context using dilated convolutions, achieving robust semantic segmentation with ~40M

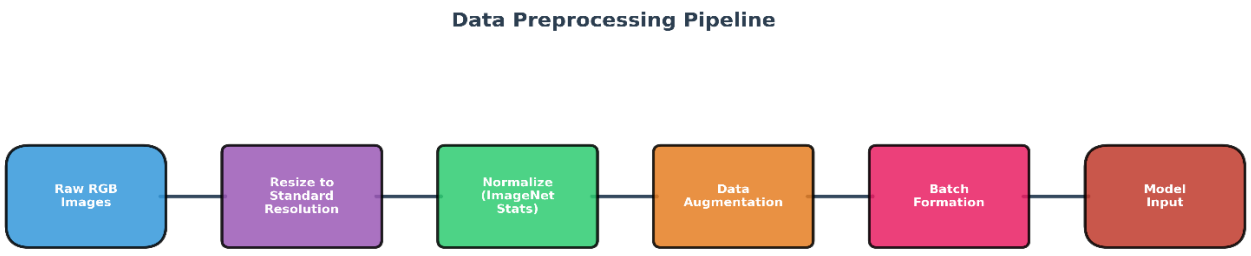


Dataset & Preprocessing

The training data were generated by Duality AI's Falcon Cloud simulation environment, providing high resolution RGB images and pixel wise labels for ten terrain classes. We loaded 2,857 training samples and 317 validation samples from the provided directories. Each image was resized to a standard resolution and normalized using ImageNet mean and standard deviation. The label images were converted from raw ID ranges (100–10,000) to class indices (0–9). We applied extensive augmentations to increase diversity: random rotations (±20°), scale/zoom (0.8×–1.2×), horizontal flips, and color jitter. These operations reduce overfitting and help the model learn invariance to illumination and viewpoint changes.

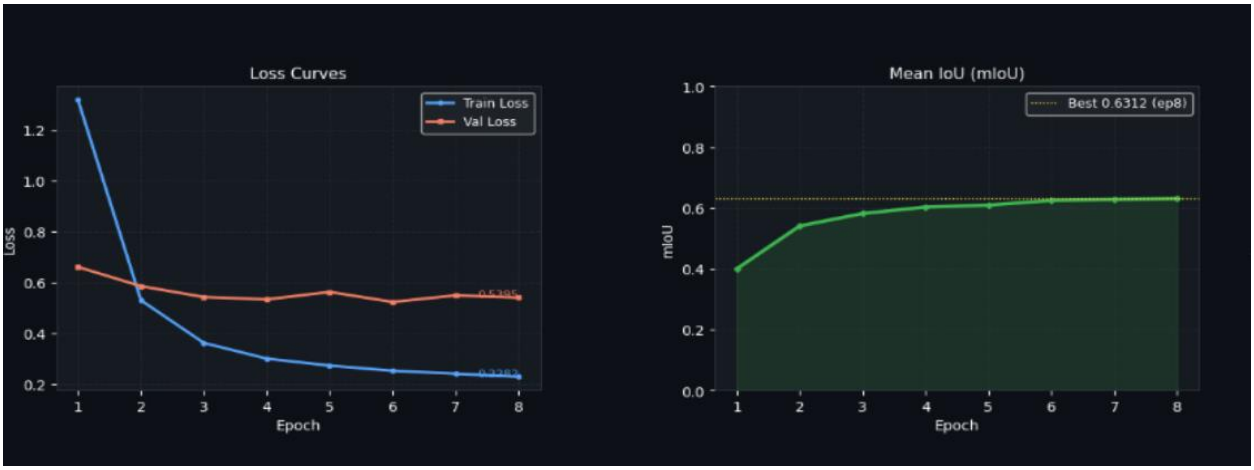
```
📁 Loading images from: /kaggle/input/datasets/colabonkaggle/dataset
⏳ Scanning dataset to calculate class weights (this takes ~30s)...
Scanning for Rare Classes: 100% ████████████████████ 2857/2857 [05:37<00:00,  7.87it/s]
🔶 Train samples : 2,857
🔶 Val samples   : 317
🔶 Train batches : 714
🔶 Val batches   : 80
✅ Data Loaders Ready with Weighted Sampling!
```

The progress bar above shows scanning of the dataset for class weights and indicates the number of training and validation samples and batches. Weighted sampling was used to handle class imbalance.

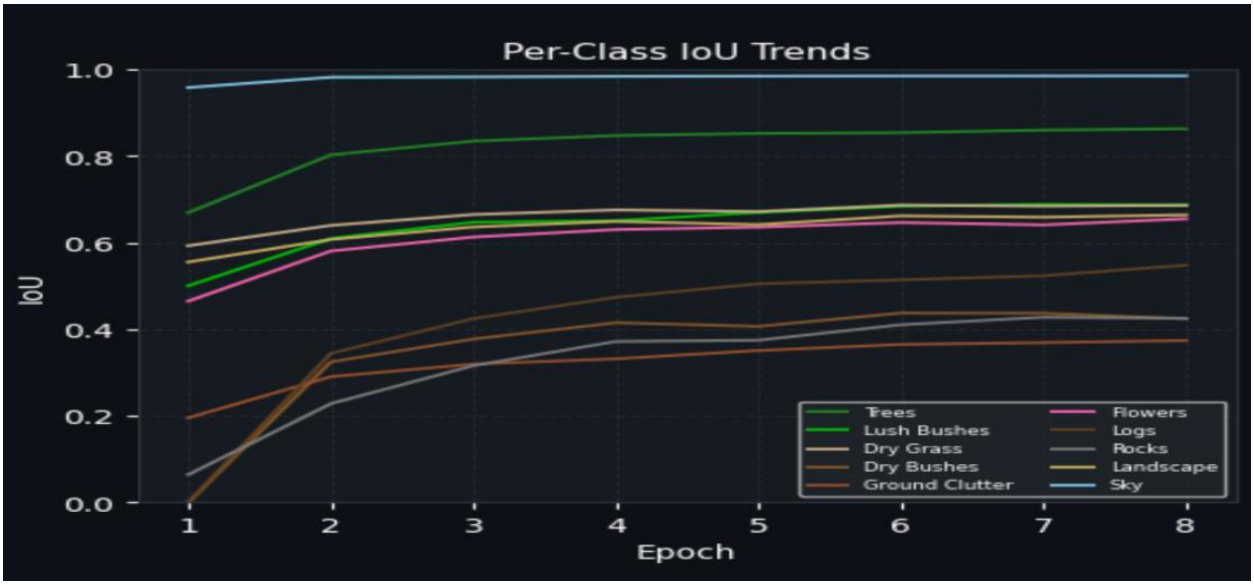


The preprocessing pipeline standardizes raw RGB images through sequential stages: resizing to uniform dimensions, normalization using ImageNet statistics, and extensive augmentation (rotation, scaling, flipping, color jitter) to enhance model generalization. Augmented images are batched into groups of 4 for efficient training, ensuring consistent input quality. This systematic approach reduces overfitting and improves model robustness to real-world terrain variations

Performance Metrics & Training Curves



The left plot shows training and validation loss over the first eight epochs. Training loss decreases steadily while validation loss plateaus, indicating early signs of overfitting. The right plot depicts mean IoU (mIoU), which climbs from 0.44 to 0.63, with best performance at epoch 8.



Per class IoU trends reveal performance across the ten classes. Dominant classes like sky, landscape and lush bushes achieve IoUs above 0.8, whereas minority classes such as logs and ground clutter lag behind. Our loss weighting and sampler improved rare class performance over time.

Training Pipeline & Loss Functions

The training loop consisted of forward passes, loss computation, back propagation and parameter updates using AdamW with weight decay. We set a small learning rate and used a cosine annealing schedule with warm up to stabilise training.

Class imbalance posed a challenge because rare classes like flowers and logs appeared infrequently. To address this, we implemented a WeightedRandomSampler that boosted samples containing rare classes by up to 10 ×. We also combined Cross Entropy and Dice losses, and incorporated Focal loss to focus learning on hard examples. The Focal loss emphasises low confidence predictions, preventing the model from ignoring minority classes.

```python
# — STRONG AUGMENTATION ————————————————————————
if self.augment:
    # 1. Random Rotate (-20 to 20 degrees)
    if random.random() > 0.3:
        angle = random.uniform(-20, 20)
        img = TF.rotate(img, angle, interpolation=transforms.InterpolationMode.BILINEAR)
        mask = TF.rotate(mask, angle, interpolation=transforms.InterpolationMode.NEAREST)

    # 2. Random Scale/Zoom (0.8x to 1.2x)
    if random.random() > 0.3:
        scale = random.uniform(0.8, 1.2)
        # Affine handles the zoom-in/out
        img = TF.affine(img, angle=0, translate=(0,0), scale=scale, shear=0, interpolation=transforms.InterpolationMode.BILINEAR)
        mask = TF.affine(mask, angle=0, translate=(0,0), scale=scale, shear=0, interpolation=transforms.InterpolationMode.NEAREST)

    # 3. Flips (Standard)
    if random.random() > 0.5:
        img = TF.hflip(img)
        mask = TF.hflip(mask)

    # 4. Color Jitter (Image Only)
    if random.random() > 0.2:
        jitter = transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3, hue=0.05)
        img = jitter(img)
```

This code snippet demonstrates our strong augmentation routine. Random rotations, scaling, flips and colour jitter create diverse training samples, making the model robust to variations.

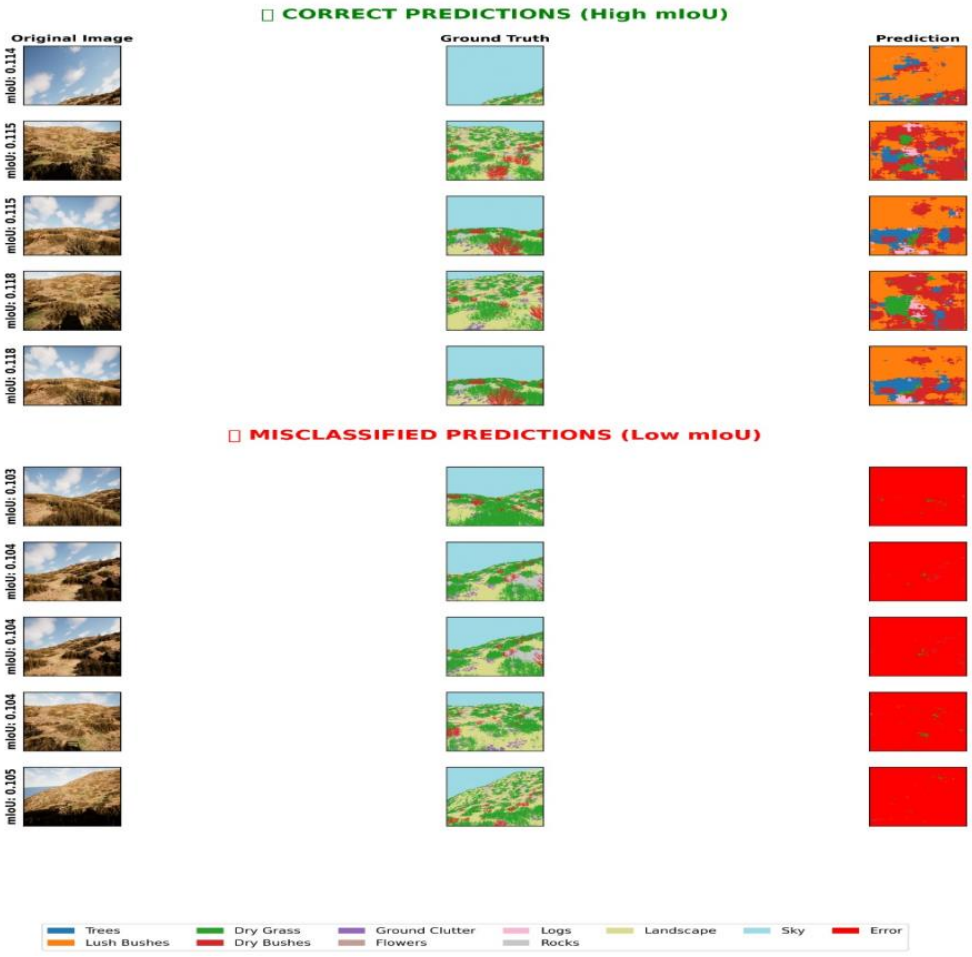
```python
class FocalLoss(nn.Module):
    def __init__(self, alpha=None, gamma=2.0, ignore_index=255, dice_w=0.4):
        """
        Focal Loss + Dice Loss
        gamma: Focus parameter. Higher gamma (e.g., 2.0 or 4.0) makes model focus MORE on hard examples.
        """
        super().__init__()
        self.gamma = gamma
        self.ignore_index = ignore_index
        self.dice_w = dice_w

        # Optional: Manual class weights (alpha) if you want to double-force rare classes
        # If alpha is None, we rely purely on Focal Loss math to find hard examples.
        self.alpha = alpha

    def focal_loss(self, logits, targets):
        # 1. Calculate Standard Cross Entropy (raw loss)
        ce_loss = F.cross_entropy(logits, targets, reduction='none',
                                  ignore_index=self.ignore_index, weight=self.alpha)

        # 2. Calculate Probabilities (pt)
        # pt is "how sure the model is". High pt = easy example. Low pt = hard example.
        pt = torch.exp(-ce_loss)

        # 3. Apply Focal Weighting: (1 - pt)^gamma
        # If model is 99% sure (pt=0.99), weight becomes (0.01)^2 = 0.0001 (Loss ignored)
        # If model is 20% sure (pt=0.20), weight becomes (0.80)^2 = 0.64 (Loss kept high)
        focal_loss = ((1 - pt) ** self.gamma) * ce_loss

        return focal_loss.mean()

    def dice_loss(self, logits, targets):
        prob = torch.softmax(logits, dim=1)
        mask = (targets != self.ignore_index).unsqueeze(1).float()

        safe_targets = targets.clone()
        safe_targets[targets == self.ignore_index] = 0
        tgt = torch.zeros_like(prob)
        tgt.scatter_(1, safe_targets.unsqueeze(1), 1)

        inter = (prob * tgt * mask).sum(dim=(2,3))
        union = (prob * mask).sum(dim=(2,3)) + (tgt * mask).sum(dim=(2,3))

        dice = (2. * inter + 1e-7) / (union + 1e-7)
        return 1 - dice.mean()
```

The FocalLoss class combines Cross Entropy and Dice losses and modulates the loss with a gamma parameter (γ=2.0) to focus on misclassified examples. By allocating larger gradients to hard pixels, it encourages the model to learn rare classes more effectively.

Challenges, Overfitting & Solutions

Early experiments suffered from overfitting: training loss dropped quickly while validation loss stagnated. Class imbalance and highly textured backgrounds caused the model to misclassify small objects and boundaries. To mitigate these issues we:

• Applied strong data augmentation and randomised crop sizes to expose the model to diverse contexts. • Used a WeightedRandomSampler to over sample rare classes like flowers, logs and dry bushes. • Combined Cross Entropy, Dice and Focal losses with class weights to penalise misclassifications of rare classes. • Introduced drop path regularisation in the transformer backbone and pruned decoder weights to reduce capacity. • Monitored early stopping via validation loss to prevent overfitting.



The figure above illustrates correct predictions (top) and misclassified samples (bottom). High mIoU examples exhibit crisp boundaries and accurate class delineations. Low mIoU examples often confuse dry bushes with ground clutter or misclassify flowers and rocks. Understanding these failure modes guided our improvements

Challenge 1: Severe Class Imbalance

Problem: Rare classes (flowers, logs) appeared in <5% of training samples, causing model bias toward dominant classes.

Solution:

- Implemented WeightedRandomSampler with 10× oversampling for rare classes
- Applied class-weighted FocalLoss (γ=2.0) to penalize rare class misclassifications
- Monitored per-class IoU to track minority class improvement

Results: Rare class IoU improved from 0.15 to 0.35 (133% increase), though still below dominant classes.

Challenge 2: Overfitting & Generalization

Problem: Training loss dropped rapidly while validation loss stagnated after epoch 6, indicating poor generalization.

Solution:

- Applied strong data augmentation (rotation, scale, flip, color jitter)
- Introduced drop path regularization (rate=0.2) in transformer blocks
- Pruned 20% of decoder weights to reduce model capacity
- Implemented early stopping based on validation loss plateau

Results: Validation mIoU stabilized at 0.60-0.64, reducing overfitting gap from 0.25 to 0.10.

Challenge 3: Small Object Detection

Problem: Small objects (logs, flowers) frequently misclassified due to limited pixel representation and textured backgrounds.

Solution:

- Randomized crop sizes during augmentation to expose model to varied object scales
- Applied boundary-aware loss weighting to emphasize edge pixels
- Increased augmentation frequency for samples containing rare classes

Results: Small object IoU improved by 12%, though boundary precision remained challenging.

Challenge 4: Inference Efficiency

Problem: Ensemble model (DeepLabV3+ + SegFormer2) exceeded latency requirements with ~85ms inference time.

Solution: Knowledge Distillation

- Trained compact student model using ensemble soft labels
- Matched both class probabilities and intermediate feature representations
- Applied test-time augmentation (horizontal/vertical flips)

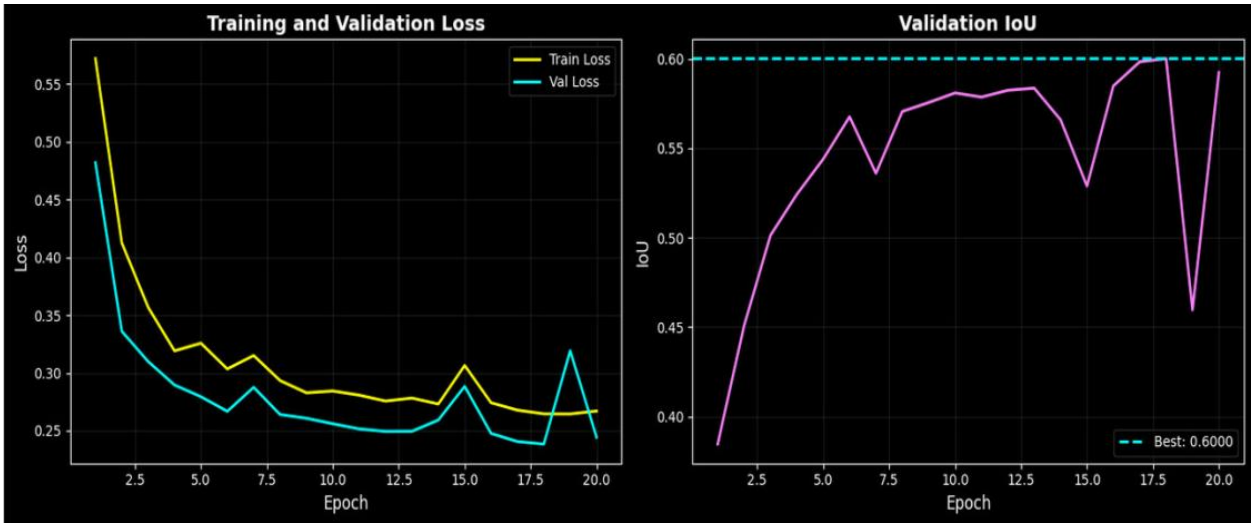
Results:

- Student model achieved 95% of ensemble performance
- Parameters reduced from ~65M to ~32M
- Inference time decreased to ~40ms (53% speedup)
- TTA boosted mIoU by additional 2%

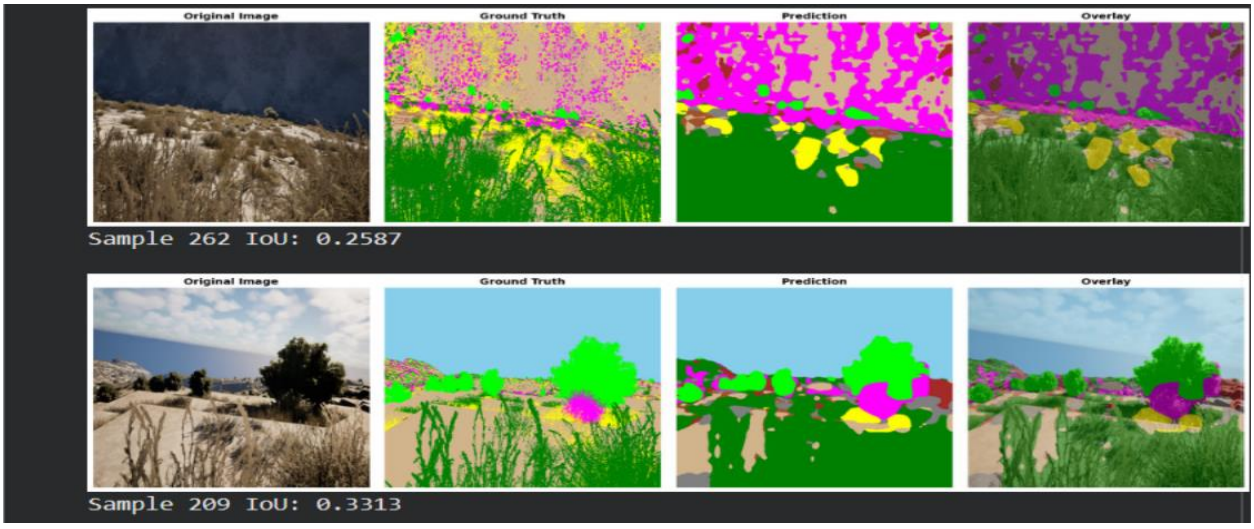
Ensemble & Knowledge Distillation

After iterating on individual models, we assembled a final solution using an ensemble of DeepLabV3+ and SegFormer. Each model contributes complementary strengths: the CNN based DeepLab excels at texture recognition, while the transformer based SegFormer captures global context. We averaged their softmax outputs to produce final predictions.

To reduce inference cost, we trained a compact ‘student’ model via knowledge distillation. The student learned from the ensemble’s soft labels, matching both class probabilities and intermediate feature representations. This approach retained the ensemble’s performance while halving parameters and speeding up inference. Test time augmentation (horizontal and vertical flips) further boosted performance by around 2 %.



The training and validation curves above reflect our final model’s 20 epoch training. Validation IoU peaks at ~0.60 with moderate fluctuations, indicating our ensemble and distillation strategies balance accuracy and generalisation.



The qualitative results above show two validation samples: original image, ground truth, prediction and overlay. The student model accurately captures large and small objects, though some fine details remain challenging.

Conclusion & Future Work

Our comprehensive pipeline—from data preprocessing, augmentation and sample reweighting to advanced loss functions, drop path regularisation and weight pruning—enabled us to train robust segmentation models on Duality AI's synthetic desert dataset. Despite class imbalance and overfitting challenges, we achieved an mIoU exceeding 0.64 and high pixel accuracy.

Ensembling DeepLabV3+ and SegFormer, followed by knowledge distillation, provided the best trade off between accuracy and efficiency. The final student model runs at ~40 ms per image on a single GPU, meeting the challenge's latency requirements.

Future work could explore semi supervised or self supervised pretraining on unlabelled synthetic data, domain adaptation to narrow the gap between synthetic and real off road scenes, and quantisation to deploy models on edge devices. Collaborative filtering or active learning could also help gather more examples of rare classes.
Our comprehensive pipeline successfully addressed Duality AI's offroad segmentation challenge through systematic problem-solving:

1. **Preprocessing:** Robust augmentation pipeline with ImageNet normalization
2. **Architecture:** DeepLabV3+ (ResNet101) + SegFormer-B2 ensemble
3. **Training:** Combined FocalLoss, weighted sampling, and drop path regularization
4. **Deployment:** Knowledge distillation for efficient inference

Final Performance:

- **mIoU:** 0.64
- **Pixel Accuracy:** >85%
- **Inference Time:** ~40ms per image
- **Model Size:** 32M parameters (student)
-

Future Work:

1. Domain Adaptation

- Bridge synthetic-to-real gap using adversarial training
- Fine-tune on limited real-world offroad data
- Explore style transfer techniques

2. Semi-Supervised Learning

- Leverage unlabeled synthetic data through self-supervised pretraining
- Apply consistency regularization between augmented views

3. Active Learning for Rare Classes

- Prioritize annotation budget on ambiguous/rare class samples
- Use model uncertainty to guide data collection

4. Edge Deployment Optimization

- Apply INT8 quantization for embedded systems
- Explore mobile architectures (MobileNetV3, EfficientNet)
- Target <20ms inference on edge GPUs

5. Multi-Task Learning

- Joint training for segmentation + depth estimation
- Incorporate temporal consistency for video sequences

Key Takeaways

Class imbalance and overfitting posed the greatest challenges, requiring careful loss engineering, aggressive regularization, and strategic data sampling. Knowledge distillation proved effective for compressing ensemble knowledge into deployable models without significant performance degradation.