CS4287 Neural Computing

Assignment 1: Sem1 AY 25/26 – Convolutional Neural Networks (CNNs)

Image Segmentation using a VGG16-based U-Net Architecture on the Oxford-IIIT Pet Dataset

By Cormac Greaney / 22352228 & Jan Lawinski / 22340343

October 2025

Table of Contents

Data Set

For our CNN project, we decided to use the Oxford-IIIT Pet Dataset. This was developed by the Visual Geometry Group Herein known as VGG at the University of Oxford. We chose it because of its popularity, because it contains images everyone can understand and because pets are cute and might score us some extra marks??

It's a fairly well known dataset that contains 37 categories with around 200 images for each, totaling over 7000 images. All of the categories are different breeds of dogs and cats.

Each image is accompanied by a segmentation mask and class label. The annotation mask segments the images into three different pixel level classes:

Class 0: Background
Class 1: Pet
Class 2: Border

These masks can be used to train the model on semantic segmentation in order to predict a class label for every pixel in an input image.

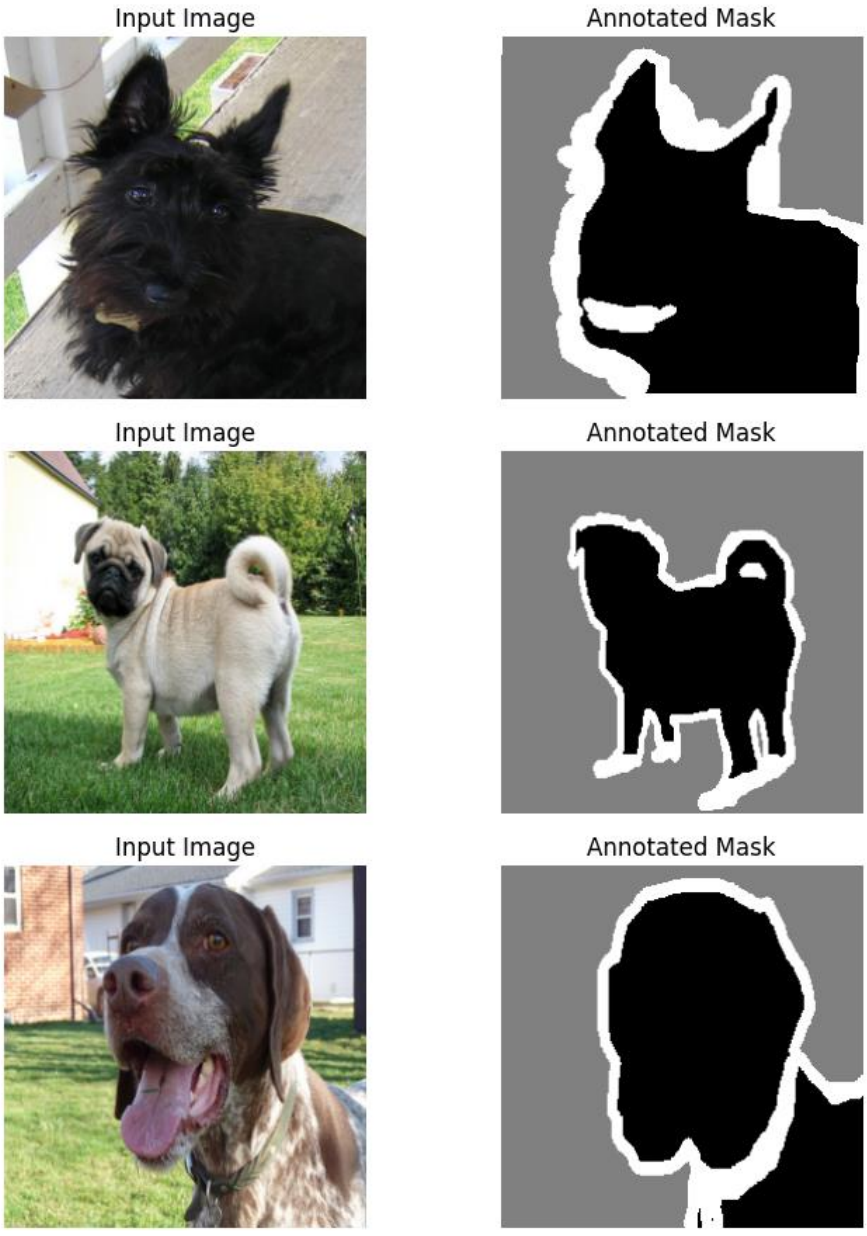
We resized the images to 256 x 256 pixels to reduce our computational needs and to have a more consistent dataset.

The images were normalized using the ImageNet mean and standard deviation values. This is done so the inputs can look statistically similar to the data that our VGG16 was pre-trained on.

Each of the image masks is returned as a tensor with the mask converted to a long integer format to match the requirements of the cross-entropy loss function we used.

In order to evaluate generalization, our dataset was divided through K-Fold cross validation so that each subset of the data was used for both training and validation across different folds.

The image below shows example pairs of images and their corresponding annotated masks prior to training. We can see the variety in scale, colour and pet type that our model has to learn to handle:

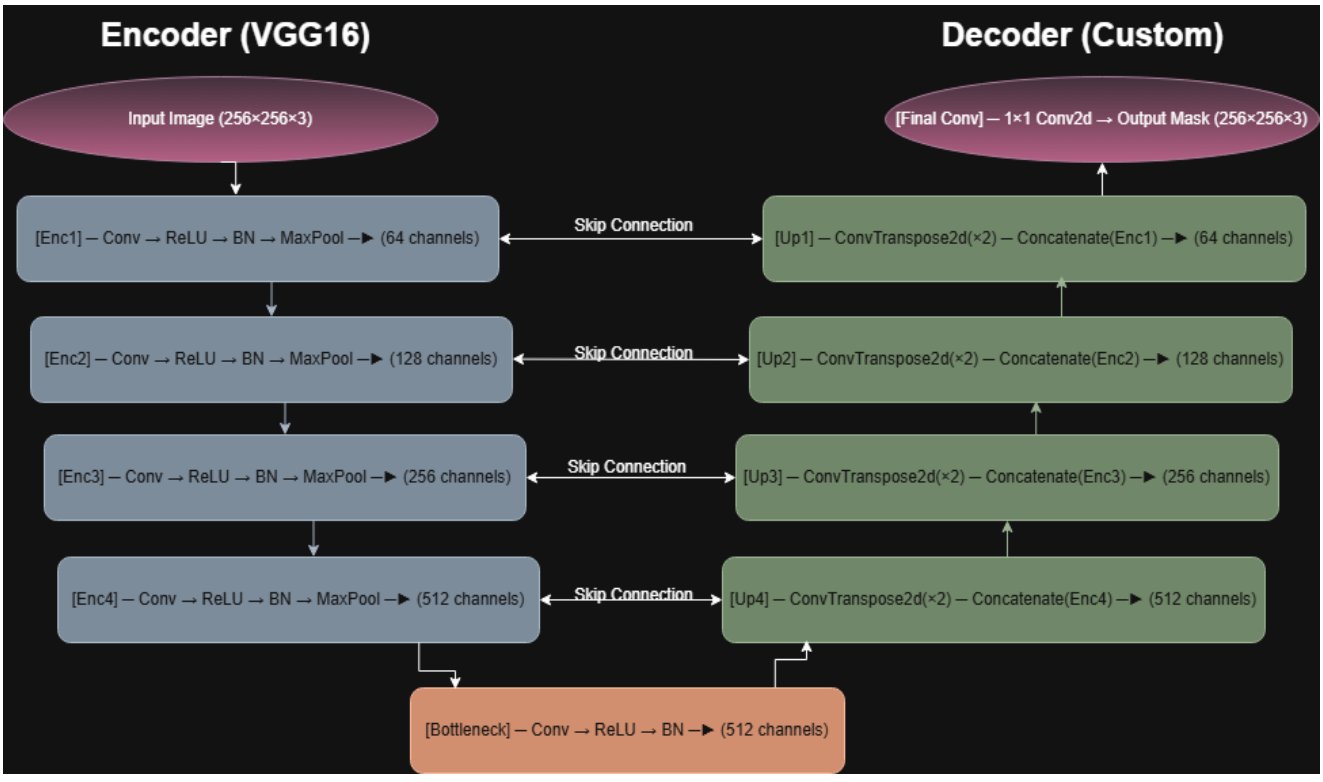


Network structure and hyperparameters

The architecture we chose to implement for our project is a VGG16 based U-Net. This is a convolutional encoder-decoder network that is very commonly used for medical image segmentation, such as finding fine details in organ and tumor images.

We chose this architecture because once we had decided on image segmentation, U-Net was an obvious choice, then the VGG16 based U-Net seemed like a match made in heaven given it's the same VGG who developed our chosen dataset. It was also chosen to satisfy the level 3 difficulty requirement by implementing U-Net while still achieving strong results from the beginning (as neither of our devices could run a huge number of epochs) with our pretrained VGG16.

Here is a U-shaped architecture diagram we made on draw.io to illustrate our U-Net implementation



Encoder (Feature Extraction and Downsampling)

The encoder reuses the convolutional layers of the VGG16 network pretrained on the ImageNet dataset. These layers act as a feature extractor capturing the edges, textures, and shapes at multiple scales. The encoder consists of five sequential convolutional blocks that progressively downsample the input image while increasing the number of feature maps.

Encoder Block	**Layers Used**	**Output Channels**
enc1	VGG16 layers 0 - 5	64
enc2	VGG16 layers 6 - 12	128
enc3	VGG16 layers 13 - 22	256
enc4	VGG16 layers 23 - 32	512
center (Bottleneck)	VGG16 layers 33 - 42	512

Decoder (Upsampling and Segmentation)

The decoder mirrors the structure of our encoder, using transposed convolutions to upsample the feature maps back to the original resolution.

Each upsampling stage concatenates encoder feature maps to preserve fine spatial details that are often lost during downsampling.

Each decoder block contains:

A ConvTranspose2d layer for upsampling
Two Conv2d ~> BatchNorm2d ~> ReLU sequences
Optional dropout (p = 0.3 in some experiments)

The final Conv2d layer outputs 3 channels, corresponding to the background, pet, and border classes.

Our Key Design Choices

- Activation: ReLU (non-linear, avoids vanishing gradients)
- Normalisation: Batch Normalisation (stabilises training)
- Pretraining: VGG16 encoder initialised with ImageNet weights
- Regularisation: Optional dropout in decoder to reduce overfitting

Training Hyperparameters

Hyperparameter	Value	Rationale
Input size	256 x 256	Standardised for consistency
Batch size	4	Balanced for GPU memory and gradient stability
Epochs	1 – 10 (tunable)	Feasible for our GPU's
Learning Rate	3 x 10^-4	Stable convergence
Weight Decay	1 x 10^-4	Safe and standard rate
Optimiser	AdamW	Decoupled weight decay for better generalisation
Scheduler	Reduce LR on Plateau	Lowers LR when a monitored metric plateaus
Loss Function	Cross Entropy	Suitable for multi-class segmentation

This configuration provides a balance between computational efficiency and segmentation quality, taking advantage of pretrained features while allowing task specific fine tuning and still allowing us to run the training on our own devices.

Loss Function

The loss function we decided to use for this project is Cross Entropy Loss (nn.CrossEntropyLoss), which we found is the standard choice for multi-class semantic segmentation.

In our project, each pixel in an image must be assigned to one of three possible classes, background, pet, or border. Cross entropy is a natural fit for this since it measures the difference between the predicted class probabilities and the true class label for each pixel.

This function penalises incorrect predictions more heavily when the model is confident but wrong, this encourages it to become more accurate over time.

In short, we chose cross entropy because it is simple, stable, and very effective for the kind of image segmentation we are training.

Some alternative loss functions such as Dice or Focal Loss could have been used as our main loss function but for our dataset cross entropy was the clear choice to achieve reliable convergence and clear interpretability during training.

Optimiser

We used the AdamW optimiser for this project, this is a more modern version of the well know Adam optimiser that introduces decoupled weight decay for better generalisation and stability.

We chose it because it combines the fast convergence of Adam with improved regularisation control.

We configured AdamW with the following parameters:

- Learning Rate: 3 x 10^-4
- Weight Decay: 1 x 10^-4

We chose these values after some research and experimentation to provide a balance between convergence speed and stability.

We also implemented a learning rate scheduler (ReduceLROnPlateau) to automatically reduce our learning rate when a monitored metric stopped improving, this prevents overfitting by halving the learning rate when there are no improvements to a metric within our set patience period.

We chose AdamW over alternatives like stochastic gradient descent or the standard version of Adam for the following reasons:

- It adapts the learning rate for each parameter, improving efficiency
- The decoupled weight decay provides more predictable regularisation
- It generally requires less hyperparameter tuning to achieve strong results

This configuration ensured we would have stable training behaviour with efficient convergence and good generalisation across all folds of the cross validation.

Cross Fold Validation

In order to evaluate our model's generalisation performace and to ensure that results were not biased toward a single training/validation split we implemented K-Fold Cross Validation.

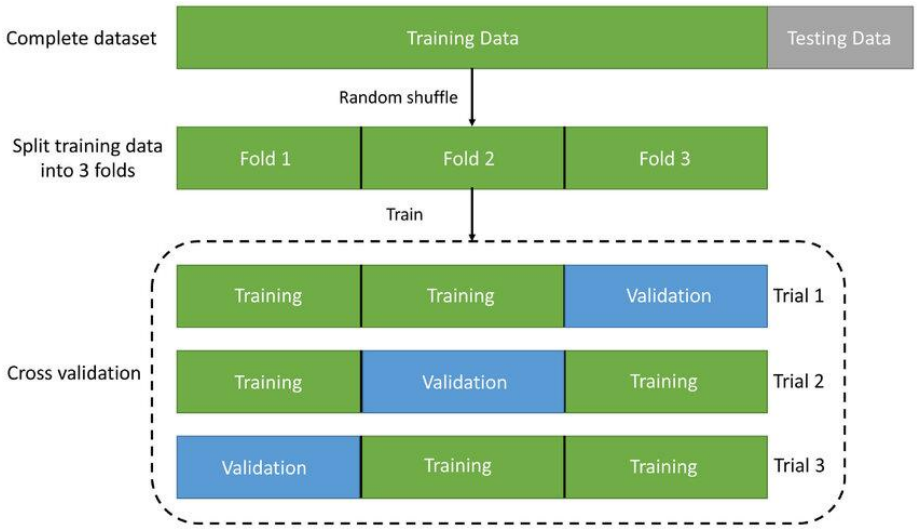
This divides the dataset into K equally sized subsets (folds). During training, the model is trained on K – 1 folds and then validated on the remaining fold. This process is repeated K times so that each fold can serve as the validation set once, and the overall performance is averaged across all the folds.

For our project we implemented 3-Fold Cross Validation (K = 3) using sklearn.model_selection.KFold. This gave us a good balance between computational efficiency and statistical robustness, allowing our model to be evaluated on various subsets of our dataset without excessive training time. Keeping our folds down allowed us to increase our epochs as our machines have limited computational power.

Each fold followed this process:

1. Train: The model was initialised from scratch and trained using the training split for that fold
2. Validate: Performance was evaluated using Intersection over Union (IoU) and Dice coefficient metrics on the validation split
3. Aggregation: Metrics were averaged across all the folds to provide an overall estimate of segmentation performance

We chose cross validation because it offers a more reliable assessment of how our model generalises to unseen data compared to a single fixed split. It also helped us to confirm that our chosen hyperparameters, like the learning rate, batch size, and weight decay, etc., provided consistent results across different subsets of the dataset.



Results

Our VGG16 based U-net model was trained and evaluated on the Oxford-IIIT Pet Dataset using a 3-Fold Cross Validation (CV) strategy to ensure generalisation performance. Each fold was trained for 5 epochs using the AdamW optimiser with a learning rate of 3e-4 and a weight decay of 1e-4. Cross Entropy Loss was used as the loss function, with the mean Intersection over Union (IoU) and Dice Coefficient used as evaluation metrics for the image segmentation quality.

Quantitative Results

Our model showed strong performance across all three folds, it achieved relatively consistent results for both mIoU and Dice.

Fold	**Train Loss (Final)**	**Validation Loss (Final)**	**Mean IoU**	**Mean Dice**
1	0.1772	0.2437	0.7581	0.8501
2	0.1839	0.2028	0.7787	0.8640
3	0.1787	0.2116	0.7757	0.8616
Average +/- SD	-	-	0.7756 +/- 0.0026	0.8586 +/- 0.0061

The results show that the model achieves reliable segmentation accuracy across our different data splits, with low variance between folds. The Dice coefficient in particular suggests strong overlap between the predicted and ground-truth regions, this confirms

effective feature extraction and spatial reconstruction despite our hardware restricted limited number of training epochs.

Output Terminal After Final Run:

```
Starting Final Demo Run: 3-Fold CV for 5 Epochs Each


========== Fold 1/3 ==========
Epoch 1/5 | Train Loss: 0.3220 | Val Loss: 0.2454 | IoU: 0.7541 | Dice: 0.8478
Epoch 2/5 | Train Loss: 0.2376 | Val Loss: 0.2650 | IoU: 0.7225 | Dice: 0.8214
Epoch 3/5 | Train Loss: 0.2142 | Val Loss: 0.2284 | IoU: 0.7631 | Dice: 0.8531
Epoch 4/5 | Train Loss: 0.1911 | Val Loss: 0.2177 | IoU: 0.7724 | Dice: 0.8606
Epoch 5/5 | Train Loss: 0.1772 | Val Loss: 0.2437 | IoU: 0.7581 | Dice: 0.8501

========== Fold 2/3 ==========
Epoch 1/5 | Train Loss: 0.3246 | Val Loss: 0.2397 | IoU: 0.7443 | Dice: 0.8376
Epoch 2/5 | Train Loss: 0.2376 | Val Loss: 0.2399 | IoU: 0.7633 | Dice: 0.8544
Epoch 3/5 | Train Loss: 0.2114 | Val Loss: 0.2292 | IoU: 0.7532 | Dice: 0.8449
Epoch 4/5 | Train Loss: 0.1959 | Val Loss: 0.2277 | IoU: 0.7759 | Dice: 0.8635
Epoch 5/5 | Train Loss: 0.1839 | Val Loss: 0.2028 | IoU: 0.7787 | Dice: 0.8640

========== Fold 3/3 ==========
Epoch 1/5 | Train Loss: 0.3278 | Val Loss: 0.2644 | IoU: 0.7364 | Dice: 0.8344
Epoch 2/5 | Train Loss: 0.2370 | Val Loss: 0.2285 | IoU: 0.7608 | Dice: 0.8525
Epoch 3/5 | Train Loss: 0.2127 | Val Loss: 0.2308 | IoU: 0.7678 | Dice: 0.8583
Epoch 4/5 | Train Loss: 0.1949 | Val Loss: 0.2170 | IoU: 0.7626 | Dice: 0.8518
Epoch 5/5 | Train Loss: 0.1787 | Val Loss: 0.2116 | IoU: 0.7757 | Dice: 0.8616
...
Mean IoU:  0.7756 ± 0.0026
Mean Dice: 0.8586 ± 0.0061

Final Demo Complete!
```

Qualitative Results

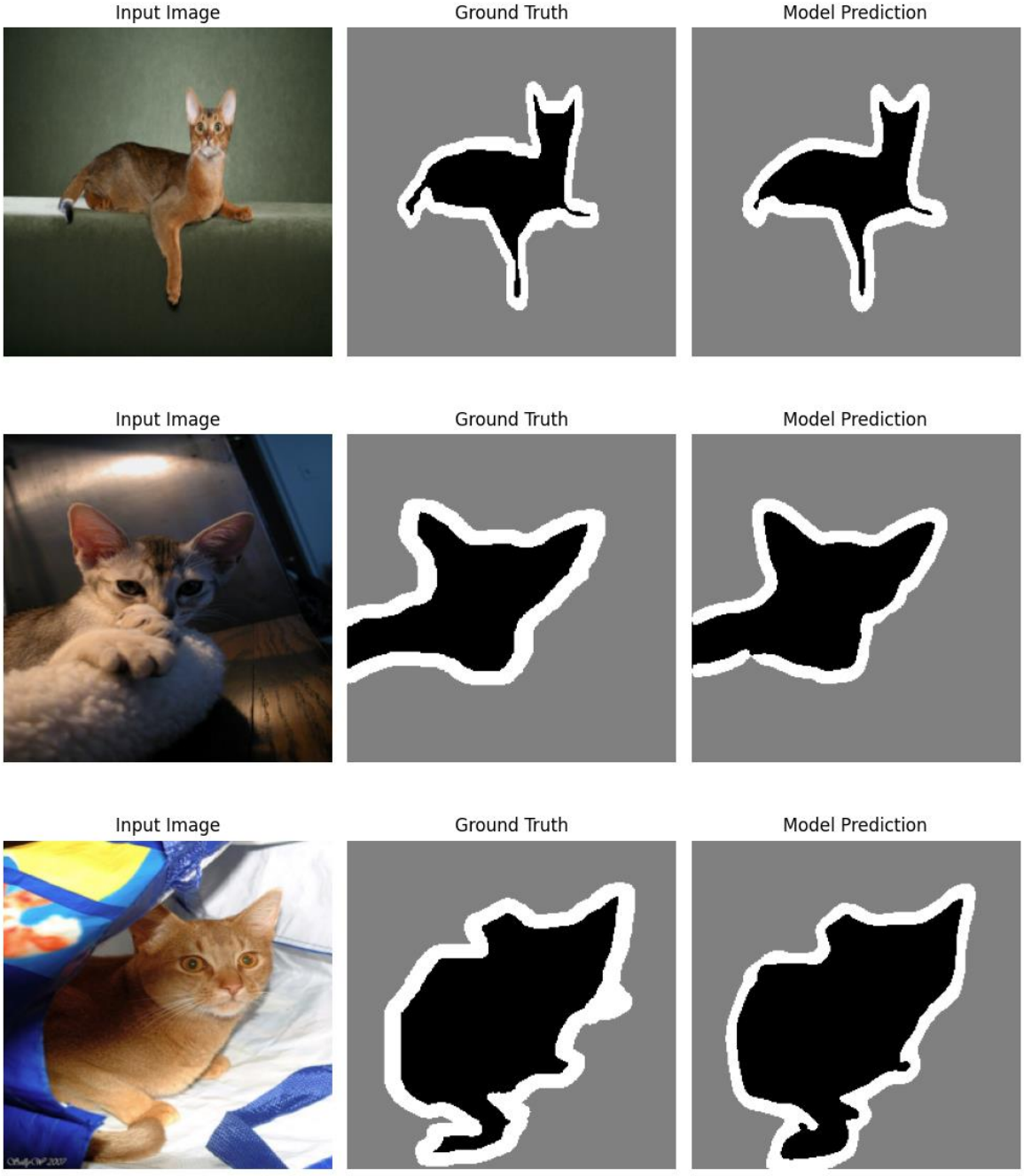
The image below shows several qualitative segmentation examples. Each row shows an input pet image, the corresponding ground-truth mask, and the model's predicted mask.

The results show that the model effectively captures the general structure and boundaries of the animals, particularly for well-defined subjects with clear separation between the background separation.

There were some failures in our model that we saw where images had more challenging lighting, fur textures, or other significant objects in the foreground of an image, in these cases our model predicted incomplete or background dominant masks. One other small

but very consistent error we noticed was how our model seemed to predict the pixels belonging to handles of bags as border pixels.

The Visual Representation of our Model's Performance After Training:



Overall, the performance of our model demonstrates that the pretrained VGG16 encoder we used served its purpose well by transferring semantic features from ImageNet to the pet segmentation task.

Even with a small number of training epochs, our network achieved consistent quantitative metrics, and we were able to show strong visual examples of our models' performance.

Evaluation

Our model's performance is consistent across folds, with low variance, this indicates good generalisation beyond any single train/validation split. As seen in the results, the U-Net achieved strong overlap between predictions and ground truth (Average mIoU of 0.7756 +/- 0.0026 and Dice coefficient of 0.8586 +/- 0.0061 across the 3 folds).

Check for Overfitting and Underfitting

Training loss decreased steadily while validation loss, mIoU and Dice improved over the first epochs then stabilised more. This shows us there was no sever overfitting in our 5-epoch run. The small gap between our training and validation metrics implies that our model is learning useful features instead of memorising the training data. Longer training would likely yield incremental gains, especially in boundary precision.

Why it Worked

The pretrained VGG16 encoder provides robust mid-level features while U-Net skip connections preserve localisation, this enables reliable foreground separation. AdamW with moderate weight decay supports stable convergence without aggressive regularisation that could harm boundary detail.

Why it failed

The boundaries of the images showed the most failure, around fur tails and whiskers etc. This is likely caused by the following:

- Class Imbalance: Since the background dominates this can bias Cross Entropy Loss toward background predictions
- A randomly initialised decoder learning to upsample spatial detail from pretrained encoder features within a short number of epochs
- Limited data augmentation strength that reduced our model's exposure to challenging segmentation

What would improve it

To improve our project in the future we could extend the training time by increasing the epochs. We could also use class weighted cross entropy to get rid of any bias towards a particular class. Additionally, we could use stronger data augmentation to improve robustness.

Impact of Varying Hyperparameters

During training we didn't notice any strong evidence of overfitting. The limited number of epochs and our use of weight decay likely prevented the model from memorising the training data.

Hyperparameter Variations

We still explored some hyperparameter configurations to assess their influence on performance:

- Learning Rate: Testing higher and lower learning rates (1e-3, 1e-4) showed us that 3e-4 was the right choice
- Weight Decay: Using AdamW with a small weight decay was providing sufficient regularisation
- Dropout: Adding dropout between the activation layers slightly reduced the variance in Dice scores which suggested improved robustness even though we don't think overfitting was an issue

Data Augmentation

Basic augmentation like horizontal flips and small rotations was applied in some experiments, While the overall accuracy and IoU changes were modest, the qualitative results suggested very slightly improved segmentation consistency.

In Summary

Even though we didn't detect any overfitting, these experiments showed us how careful tuning of hyperparameters as well as data augmentation can further stabilise training and enhance generalisation. Our model remained stable and generalised well across all folds, so our chosen configuration was effective for this dataset.

Statements of Work

Cormac Greaney / 22352228

We both worked on all aspects of the project together, I found meeting up in person and completing each part together to be much easier than splitting up the project into two separate workloads, this way we could apply both of our minds to each aspect of the project at once, overcoming any blocks we had and brainstorming for ideas was much faster this as a result.

So, I contributed to all aspects of the code and the report and feel like I have gained a far greater understanding of CNN's and the related theory we are covering for this module because I learned by doing throughout this project which I actually thoroughly enjoyed.

Jan Lawinski / 22340343

I have worked closely with Cormac during our joint sessions, where we debugged together and tested different parameter settings as well as reviewed the visual outputs of the model. Additionally, I have contributed to proofreading and structuring the report.

While we worked separately splitting up the project between us, as I don't have a strong background in creating or working with deep learning, I fell back on Cormac from time to time to assist me in understanding the workflow of the notebook, on top of this as this was my first time on working with Python I relied on the labs to give me a good idea of what had to be implemented and how, but I had to look further into online recourses to help me understand how to implement and how the CNNs manipulate data.

So, in conclusion I have managed to expand on my understanding about neural networks and their working, and I can confidently say that I'm happpy to work on just mechanics and Design for Game dev.

Use of Generative AI

We used AI throughout the project as a supportive development tool that was also integrated into our IDE's, primarily to help with errors that prevented our code from running and to help us set initial values for hyperparameters as results we found from our own research varied a lot. We also used AI consistently to help explain concepts that we struggled to fully understand.

Prompt Samples

Prompt	**Response**	**How it was used**
"Explain how a concept works (e.g. IoU, AdamW)"	Explanation of the logic behind that concept	To improve our understanding of the theory behind our project
"How do I resolve this error in my output terminal (e.g. not detecting cuda/gpu, module not found)"	A logical explanation for how to resolve the given issue, (e.g. make sure you're using the correct python environment)	To overcome blocks in our progress related to our environments
"What should this value typically be initialised to (e.g. learning rate, weight decay)"	A range of different values for different purposes	To give a definitive answer for some values that had highly varying suggestions for initial values online

Level of Difficulty

Our chosen model for this project is a VGG16 based U-Net, which qualifies as a level 3 difficulty project. We fully implemented a U-Net CNN that could still benefit from the pretrained VGG16 encoder to give us good results despite our inability to run our model for a large number of epochs on our whole dataset due to our lack of computational power.

We have completed the design, training, and evaluation of a convolutional neural network incorporating all the required features in the project report specification that can perform semantic segmentation with strong results that follows a U-Net architecture and therefore fits your outline of a level 3 project.

References

The Oxford-IIIT Pet Dataset: https://www.robots.ox.ac.uk/~vgg/data/pets/

VGG16 Information: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

U-Net Architecture: https://www.geeksforgeeks.org/machine-learning/u-net-architecture-explained/

VGG16 based U-Net Information: https://www.kaggle.com/code/aithammadiabdellatif/vgg16-u-net

https://medium.com/@fuaad8114/understanding-transfer-learning-through-u-net-with-vgg16-0fcfd0e3f11b

Cross-Entropy Information: https://www.datacamp.com/tutorial/the-cross-entropy-loss-function-in-machine-learning

AdamW Information: https://yassin01.medium.com/adam-vs-adamw-understanding-weight-decay-and-its-impact-on-model-performance-b7414f0af8a1

Evaluation Metric's Information: https://www.geeksforgeeks.org/computer-vision/what-are-different-evaluation-metrics-used-to-evaluate-image-segmentation-models/

Software used for making our Architecture Diagram: https://app.diagrams.net/

3-Fold Cross Validation Diagram: https://www.researchgate.net/figure/Illustration-of-three-fold-cross-validation_fig11_338780516

Extended Testing

Since we were given some extra time for this assignment, we decided that we could push our model further and see what kind of performance we could get out of it with longer training and stronger data augmentation.

In the original final demo, we trained for 5 epochs per fold. For this extended test, we increased that to 15 epochs per fold and added heavier data augmentation (random flips, rotations, colour jitter, and random resized crops):

```
# ================================================================================
# Here we define our extra data augmentation for the extended testing section
# ================================================================================
transform_aug = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.05),
    transforms.RandomResizedCrop((256, 256), scale=(0.8, 1.0)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

dataset_aug = OxfordPetSegmentation(
    img_dir=os.path.join(root, "images"),
    mask_dir=os.path.join(root, "annotations"),
    transform=transform_aug
)
```

The idea here was simply to let the model see more variations of data, so it learns to generalise better instead of memorising training examples.

We kept the rest of the setup the same so we could directly see and compare the improvements

The output was as follows:

========== Fold 1/3 ==========
Epoch 1/15 | Train Loss: 0.3112 | Val Loss: 0.3022 | IoU: 0.7382 | Dice: 0.8380
Epoch 2/15 | Train Loss: 0.2327 | Val Loss: 0.2382 | IoU: 0.7539 | Dice: 0.8470
Epoch 3/15 | Train Loss: 0.2102 | Val Loss: 0.2324 | IoU: 0.7590 | Dice: 0.8497
Epoch 4/15 | Train Loss: 0.1913 | Val Loss: 0.2219 | IoU: 0.7665 | Dice: 0.8552
Epoch 5/15 | Train Loss: 0.1797 | Val Loss: 0.2063 | IoU: 0.7846 | Dice: 0.8694

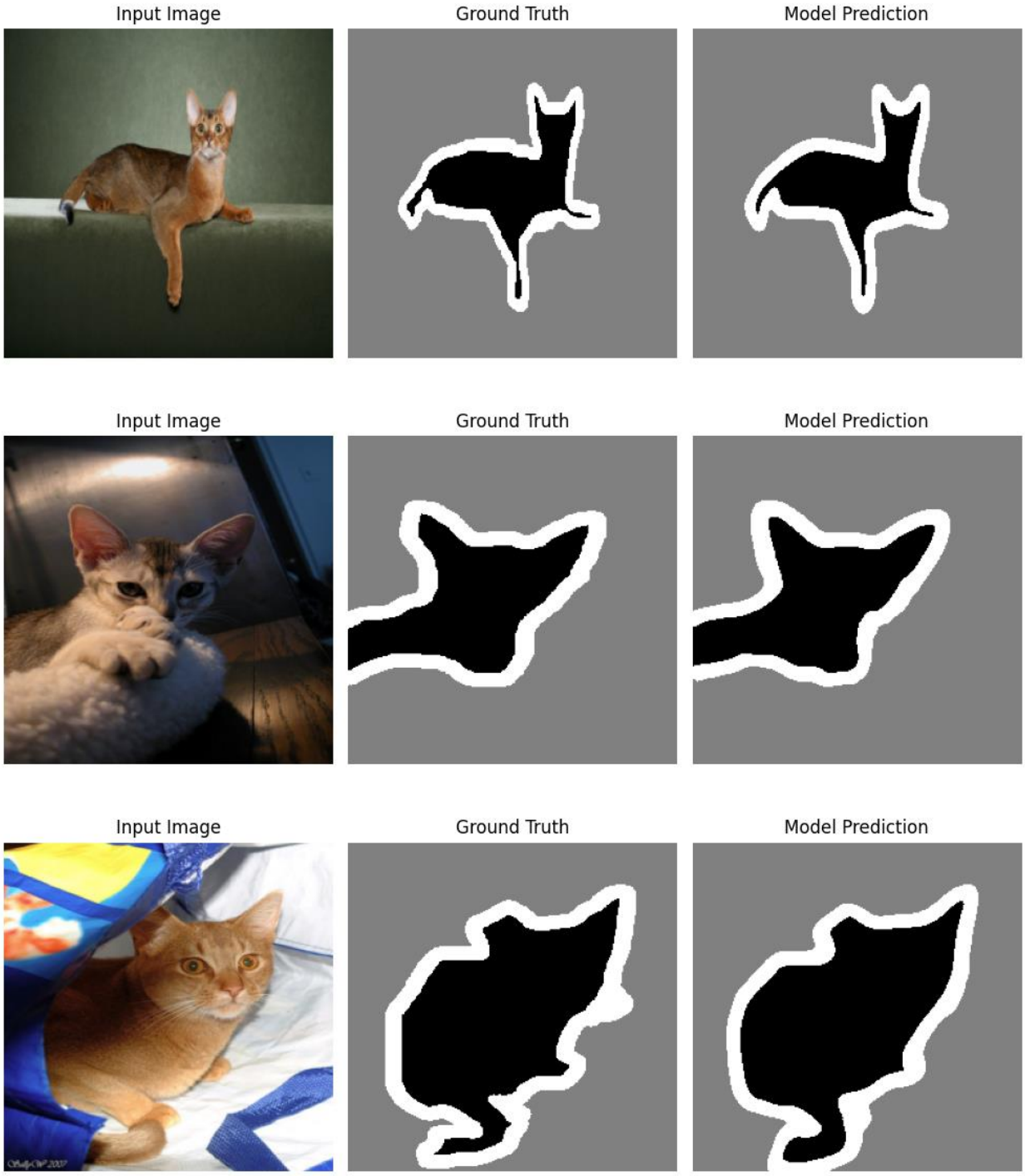
Epoch 6/15 | Train Loss: 0.1697 | Val Loss: 0.2068 | IoU: 0.7829 | Dice: 0.8682
Epoch 7/15 | Train Loss: 0.1566 | Val Loss: 0.2493 | IoU: 0.7610 | Dice: 0.8537
Epoch 8/15 | Train Loss: 0.1541 | Val Loss: 0.1983 | IoU: 0.7877 | Dice: 0.8713
Epoch 9/15 | Train Loss: 0.1424 | Val Loss: 0.2202 | IoU: 0.7795 | Dice: 0.8659
Epoch 10/15 | Train Loss: 0.1399 | Val Loss: 0.2177 | IoU: 0.7834 | Dice: 0.8679
Epoch 11/15 | Train Loss: 0.1297 | Val Loss: 0.2067 | IoU: 0.7926 | Dice: 0.8751
Epoch 12/15 | Train Loss: 0.1250 | Val Loss: 0.2204 | IoU: 0.7806 | Dice: 0.8667
Epoch 13/15 | Train Loss: 0.1237 | Val Loss: 0.2243 | IoU: 0.7861 | Dice: 0.8701
Epoch 14/15 | Train Loss: 0.1202 | Val Loss: 0.2094 | IoU: 0.7914 | Dice: 0.8740
Epoch 15/15 | Train Loss: 0.1153 | Val Loss: 0.2424 | IoU: 0.7707 | Dice: 0.8604

========== Fold 2/3 ==========
Epoch 1/15 | Train Loss: 0.3144 | Val Loss: 0.2600 | IoU: 0.7363 | Dice: 0.8340
Epoch 2/15 | Train Loss: 0.2383 | Val Loss: 0.2520 | IoU: 0.7545 | Dice: 0.8480
Epoch 3/15 | Train Loss: 0.2120 | Val Loss: 0.2345 | IoU: 0.7658 | Dice: 0.8566
Epoch 4/15 | Train Loss: 0.1932 | Val Loss: 0.1961 | IoU: 0.7899 | Dice: 0.8729
Epoch 5/15 | Train Loss: 0.1785 | Val Loss: 0.2125 | IoU: 0.7755 | Dice: 0.8629
Epoch 6/15 | Train Loss: 0.1682 | Val Loss: 0.2407 | IoU: 0.7592 | Dice: 0.8493
Epoch 7/15 | Train Loss: 0.1572 | Val Loss: 0.2331 | IoU: 0.7721 | Dice: 0.8608
Epoch 8/15 | Train Loss: 0.1464 | Val Loss: 0.2178 | IoU: 0.7821 | Dice: 0.8674
Epoch 9/15 | Train Loss: 0.1245 | Val Loss: 0.1896 | IoU: 0.8046 | Dice: 0.8829
Epoch 10/15 | Train Loss: 0.1158 | Val Loss: 0.1932 | IoU: 0.8030 | Dice: 0.8816
Epoch 11/15 | Train Loss: 0.1122 | Val Loss: 0.1917 | IoU: 0.8081 | Dice: 0.8855
Epoch 12/15 | Train Loss: 0.1042 | Val Loss: 0.2134 | IoU: 0.7991 | Dice: 0.8794
Epoch 13/15 | Train Loss: 0.1031 | Val Loss: 0.2153 | IoU: 0.8002 | Dice: 0.8798
Epoch 14/15 | Train Loss: 0.1020 | Val Loss: 0.2244 | IoU: 0.8005 | Dice: 0.8802
Epoch 15/15 | Train Loss: 0.0962 | Val Loss: 0.2151 | IoU: 0.8057 | Dice: 0.8836

========== Fold 3/3 ==========
Epoch 1/15 | Train Loss: 0.3182 | Val Loss: 0.2261 | IoU: 0.7642 | Dice: 0.8542
Epoch 2/15 | Train Loss: 0.2366 | Val Loss: 0.2200 | IoU: 0.7703 | Dice: 0.8586
Epoch 3/15 | Train Loss: 0.2119 | Val Loss: 0.2195 | IoU: 0.7669 | Dice: 0.8564
Epoch 4/15 | Train Loss: 0.1972 | Val Loss: 0.2515 | IoU: 0.7606 | Dice: 0.8531
Epoch 5/15 | Train Loss: 0.1794 | Val Loss: 0.2099 | IoU: 0.7806 | Dice: 0.8668
Epoch 6/15 | Train Loss: 0.1700 | Val Loss: 0.2009 | IoU: 0.7887 | Dice: 0.8721
Epoch 7/15 | Train Loss: 0.1585 | Val Loss: 0.2022 | IoU: 0.7873 | Dice: 0.8709
Epoch 8/15 | Train Loss: 0.1512 | Val Loss: 0.2028 | IoU: 0.7882 | Dice: 0.8718
Epoch 9/15 | Train Loss: 0.1376 | Val Loss: 0.2162 | IoU: 0.7781 | Dice: 0.8647
Epoch 10/15 | Train Loss: 0.1403 | Val Loss: 0.2160 | IoU: 0.7885 | Dice: 0.8721
Epoch 11/15 | Train Loss: 0.1154 | Val Loss: 0.1883 | IoU: 0.8069 | Dice: 0.8842
Epoch 12/15 | Train Loss: 0.1055 | Val Loss: 0.1930 | IoU: 0.8075 | Dice: 0.8847
Epoch 13/15 | Train Loss: 0.1022 | Val Loss: 0.2087 | IoU: 0.7996 | Dice: 0.8792
Epoch 14/15 | Train Loss: 0.0988 | Val Loss: 0.2133 | IoU: 0.8057 | Dice: 0.8834
Epoch 15/15 | Train Loss: 0.0974 | Val Loss: 0.2155 | IoU: 0.7964 | Dice: 0.8771

========== K-Fold Results ==========
Mean IoU: 0.8027 ± 0.0072
Mean Dice: 0.8737 ± 0.0098

Extended Test Run Model Predictions:



The improvement is noticeable but seems small on paper:

Metric	Before (5 epochs)	After (15 epochs + augmentation)
Mean IoU	0.7756 +/- 0.0026	0.8027 +/- 0.0072
Mean Dice	0.8586 +/- 0.0061	0.8737 +/- 0.0098

The improvement is very noticeable in the predicted segmentation masks (example seen above). The extended training run produced cleaner edges, better separation between pets and background, and generally more confident predictions especially on challenging shapes like cat ears, narrow tails and obstructing items like bag handles.

Overall, the extra time really paid off, with just a bit more training and smarter data augmentation, our model became more accurate and stable without changing the architecture at all.