Computer Vision Mini-Project Report

B177280, B176913

Abstract

This report presents our investigation into semantic image segmentation using a variety of deep learning techniques, including UNet, autoencoder pretraining, CLIP-based feature extraction, and a prompt-based segmentation model. The Oxford-IIIT Pet Dataset is used to evaluate model performance based on Mean Intersection over Union (IoU), Dice coefficient, and Pixel accuracy. Our results demonstrate that prompt-based models leveraging CLIP features outperform the other approaches, achieving the highest segmentation quality. To further evaluate its reliability, we assess the robustness of this best-performing model under a range of input perturbations, simulating real-world conditions. Additionally, we developed an interactive application that allows users to experiment with the prompt-based model using either point- or text-based prompts.

1. Introduction

Image segmentation, also known as pixel-level classification, is the task of clustering parts of an image together that belong to the same object class. It is widely used in applications such as medical diagnosis for tumour detection or autonomous driving for obstacle detection. Segmentation can be semantic, where pixels are labelled by class, or instance-based, where individual object instances are distinguished. In this project, we focus on semantic segmentation, assigning every pixel to one of several classes.

The goal of this project is to investigate how different models and feature extraction strategies affect segmentation performance. Specifically, we evaluate the UNet architecture, three variants of autoencoder-based models, a CLIP-based segmentation model, and a prompt-based model. This comprehensive evaluation helps us understand the strengths and limitations of each approach and provides insights into designing more reliable and interactive segmentation systems.

2. Data Preprocessing and Augmentation

2.1. Dataset Overview

The dataset used for our image segmentation task is the Oxford-IIIT Pet Dataset, first introduced by Parkhi et al. (2012). It contains a total of 7,392 RGB images, each paired with a corresponding segmentation mask. The dataset includes 37 pet categories with approximately 200 images per class, featuring significant variations in scale, pose and lighting. As the original images vary in size, all inputs are resized for consistency during preprocessing. The dataset is divided into two predefined sets: *TrainVal* (3,680 images) and *Test* (3,710 images). From the *TrainVal* set, 20% is reserved for validation.

The ground-truth segmentation masks are also RGB images and follow a three-class colour labelling scheme. These colours are mapped to integer class indices as follows:

$$(0, 0, 0) \rightarrow \text{Background (Class 0)}$$
$$(0, 128, 0) \rightarrow \text{Dog (Class 1)}$$
$$(128, 0, 0) \rightarrow \text{Cat (Class 2)}$$

To handle this conversion, a function iterates over each pixel in the RGB mask and replaces it with its corresponding class label, creating a 2D class mask suitable for training.

2.2. Resizing and Normalisation

To ensure consistent input dimensions, all images and their corresponding masks are resized to a fixed size of 224x224 pixels using the `albumentations` library (Buslaev et al., 2020). After resizing, the input images are normalised using the standard ImageNet statistics [1] to standardise pixel intensity distribution. Lastly, the processed images are converted into PyTorch-compatible tensors for model training.

2.3. Augmentation Techniques

To enhance the generalisation and robustness of our models, we employ the Python library `albumentations` (Buslaev et al., 2020) to implement a variety of data augmentation techniques. These include:

- Horizontal flipping: to introduce left-right symmetry.
- Rotation: to simulate slight camera angle variations.
- Random resized cropping: to introduce spatial variation and scale diversity.
- Elastic transformations: to mimic non-rigid deformations.
- Grid distortion: to add further variation in local geometry.

[1] $\mu = [0.485, 0.456, 0.406]$, $\sigma = [0.229, 0.224, 0.225]$

Each augmentation is applied probabilistically, increasing the diversity of the training data and helping the models generalise more effectively, particularly given the dataset's variability in pose, lighting and background.

2.4. Dataset Class Implementation

Data preprocessing and augmentation are integrated directly into the training pipeline rather than performed offline, which avoids the need to store augmented images and improves storage efficiency. To manage this process dynamically and selectively, we implemented two custom Pytorch (Paszke et al., 2019) Dataset classes: `SegmentationDataset` and `ValSegmentationDataset`.

The `SegmentationDataset` is used during training and includes both data augmentation and preprocessing. In contrast, the `ValSegmentationDataset` is used for validation and testing. It does not perform any augmentation, only resizing and normalisation, ensuring that evaluation is done in unmodified data. Additionally, this class returns not only the transformed image and mask but also the original unaltered mask and the original image dimensions.

3. Metrics

To calculate the metrics, the predicted masks were converted to their original resolution and compared directly to the ground truth.

3.1. Mean Intersection over Union (IoU)

For each class c, IoU is computed as the ratio between the number of pixels where the predicted and ground truth masks agree, and the total number of pixels labelled as that class in either mask.

Instead of calculating IoU separately for each image, we accumulate the intersection and union counts for each class across the entire test set:

$$\text{IoU}_c = \frac{\text{total\_intersection}_c + \epsilon}{\text{total\_union}_c + \epsilon}$$

where ϵ is a small constant for numerical stability.

The final Mean IoU is obtained by averaging over all classes:

$$\text{Mean IoU} = \frac{1}{C} \sum_{c=1}^{C} \text{IoU}_c$$

3.2. Dice coefficient

Measures the similarity between the predicted and ground truth masks. It is calculated per class and averaged. For a given class c, the dice score is:

$$\text{Dice}_c = \frac{2 \cdot \text{TP}_c}{2 \cdot \text{TP}_c + \text{FP}_c + \text{FN}_c}$$

Or, equivalently:

$$\text{Dice}_c = \frac{2 \cdot |P_c \cap G_c|}{|P_c| + |G_c|}$$

Where, P_c are the predicted pixels for class c and G_c are the ground pixels for class c.

3.3. Pixel accuracy

It is defined as the ratio of correctly predicted pixels to the total number of pixels, regardless of class. The equation is:

$$\text{Pixel Accuracy} = \frac{\text{Number of Correctly Predicted Pixels}}{\text{Total Number of Pixels}}$$

4. Segmentation Models

4.1. UNet-Based Segmentation

The UNet architecture, introduced by Ronneberger et al. (2015), is a widely used model for image segmentation, composed of two main components: the encoder and the decoder, as seen in Figure 1.

Model Architecture:

The encoder, which forms the contracting path, captures the context and essential features of the input image. It typically consists of a series of convolutional blocks, each composed of two consecutive 3x3 convolutional layers, followed by batch normalisation and ReLU activation functions. Downsampling is performed using 2x2 max-pooling layers. As the network goes deeper, the spatial resolution of the feature maps decreases while the number of channels increases, allowing the model to learn increasingly abstract representations. The number of feature maps is doubled at each pooling layer.

Connecting the encoder and the decoder is a horizontal bottleneck, which consists of a 3x3 convolution followed by a 2x2 up-convolution. This stage marks the transition from compressing the input to progressively reconstructing it in the decoder.

The decoder, or expanding path, is in charge of reconstructing the spatial structure of the segmentation output. Each block consists of convolution operations and upsampling techniques, in our case bilinear interpolation, followed by 1x1 convolutions to reduce the number of feature channels. These upsampled features are concatenated with the corresponding encoder features via skip connections. The resulting tensor passes through a convolutional block composed

of two 3×3 convolutional layers, each followed by batch normalisation and a ReLU activation. After each upsampling step, the number of feature channels is halved. The final output layer is a 1x1 convolution that maps the features to the number of target classes.

We just mentioned a key feature of the UNet architecture: skip connections between corresponding layers in the encoder and decoder. These connections help combine high-level semantic information with low-level spatial detail, which significantly improves the precision of the segmentation, especially at object boundaries.

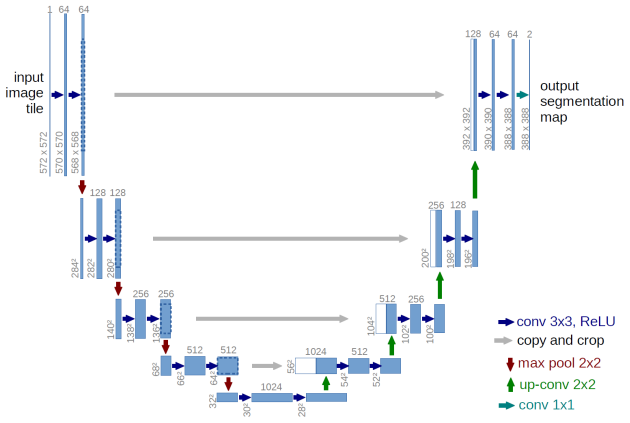

Figure 1. UNet architecture.

Training procedure:

This model was trained for 100 epochs using the Adam optimizer with an initial learning rate of $1e-4$ and L2 regularization with a weight decay of $1e-4$. To improve stability and convergence, we used a `ReduceLROnPlateau` learning rate scheduler to dynamically reduce the learning rate if the validation loss stopped improving. The scheduler was configured with a patience of 5 epochs and a decay factor of 0.5.

To mitigate class imbalance, we applied class weighting in the cross-entropy loss function: the background class was assigned a lower weight of 0.1, while both dog and cat classes were weighted equally at 1.0. This adjustment helped prevent the model from being biased toward predicting the background, which dominates the masks. The weighted cross-entropy loss used during training is given by:

$$\mathcal{L}_{\text{CE}} = -\sum_{c=1}^{C} w_c \cdot y_c \log(\hat{y}_c)$$

where w_c is the weight for class c, y_c is the ground-truth label, and $\hat{y}_c$ is the predicted probability for that class.

4.1.1 Results

From the results in Table 1, the model demonstrated robust performance. It obtained a high IoU, indicating a strong overlap between the predicted masks and ground truth, as well as a high pixel accuracy, labelling a large proportion of pixels.

Mean IoU	Dice Score	Pixel Accuracy
0.7278	0.7765	0.9088

Table 1. Performance of the UNet segmentation model on the test set.

The visualisations in Figure 2 indicate an accurate alignment with the actual object boundaries, meaning that the model is able to capture small details such as the curves along the fur. However, some visualisations show inconsistency in the class assignment. In Figure 3, each image is supposed to produce a segmentation mask of a single colour, but the model confuses class labels within the same object. This behaviour suggests that while the model is accurate when localising and delineating the object, it has issues with class discrimination.

Figure 2. Accurate classification on the test set using the UNet architecture.

As a result, the UNet architecture effectively captures the global structures and details of the objects, which can be attributed to the use of skip connections which preserve spatial information. Nonetheless, improvement is needed in order to ensure consistent class predictions.

Figure 3. Class misclassifications on the test set using the UNet architecture.

4.2. Autoencoder Pre-training for Segmentation

Autoencoders are unsupervised neural networks designed to learn compressed representations of input data by minimising reconstruction error. They consist of two main components: an encoder that reduces the spatial dimensionality of the input and learns feature representations, and a decoder that reconstructs the original input from the encoded features. In this project, the autoencoders perform unsupervised pre-training on raw images. The goal is to transfer the encoder's learned features to a downstream segmentation task by attaching a new decoder head for pixel-wise classification. We explored three variants of autoencoder architectures, which are detailed below.

For training the autoencoders, we exclusively use the unlabeled images, which are resized to 224x224 and undergo the augmentations explained above.

4.2.1 Method 1 — Simple Autoencoder

- **Autoencoder Model Architecture and Training:** The simple autoencoder employs a minimal convolutional encoder-decoder architecture. The encoder is made of two 3×3 convolutional layers with ReLU activations and max-pooling for spatial downsampling, while the decoder mirrors this structure using transposed convolutions to upsample the feature maps back to 224×224. A final convolutional layer with a sigmoid activation produces the reconstructed image with values in [0, 1]. The model is trained for 100 epochs using the Adam optimizer (lr = 0.001) with Mean Squared Error (MSE) loss to minimize reconstruction error.
- **Segmentation Model Architecture and Training:** After training, the encoder is extracted and frozen. A new segmentation decoder that consists of transposed convolution layers is appended to upsample the encoded features back to the input resolution. The final layer outputs a 3-channel segmentation mask with a softmax activation to yield class probabilities. This segmentation model is then fine-tuned on the labelled dataset using cross-entropy loss and the Adam optimizer (lr = 0.001) for 50 epochs, with performance monitored via metrics such as IoU, Dice score, and pixel accuracy.

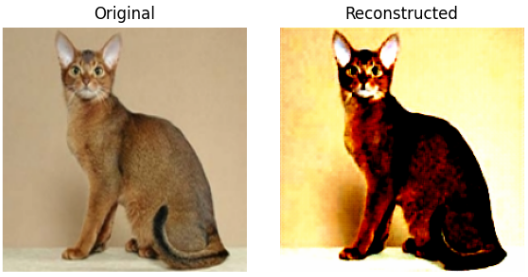

Figure 4. Comparison between the predicted mask and ground truth of a cat, using the simple autoencoder model.

Figure 5. Comparison between the predicted mask and ground truth of a cat, using the simple autoencoder.

4.2.2 Method 2 — Improved Autoencoder

- **Autoencoder Model Architecture and Training:** The improved autoencoder features a deeper encoder with three convolutional blocks that progressively increase the channel depth (32→64→128) using strided convolutions for downsampling. Each block is enhanced with batch normalisation and LeakyReLU activations for better training stability. The decoder employs bilinear upsampling followed by convolution, batch normalisation, and LeakyReLU, with a final convolution and sigmoid activation to produce the output image. This autoencoder is trained for 100 epochs with the Adam optimiser (lr = 0.001) using MSE loss.
- **Segmentation Model Architecture and Training:**

The segmentation model leverages the pre-trained encoder from the improved autoencoder. A new decoder is designed using bilinear upsampling blocks and, in some cases, skip connections (by incorporating early encoder features) to better recover spatial details. The final layer outputs semantic masks using a LogSoftmax activation, which is suited for multi-class segmentation. Fine-tuning is performed on a labelled dataset with cross-entropy loss and the Adam optimiser (lr = 0.0001) over 50 epochs, with validation metrics (Dice, IoU, pixel accuracy) guiding the training progress.

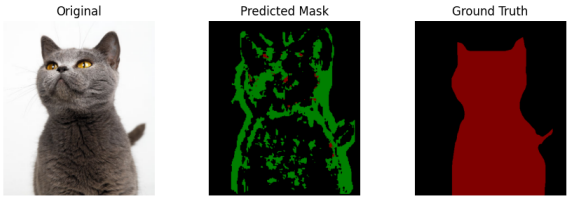

Figure 6. Comparison between the predicted mask and ground truth of a cat, using the improved autoencoder.

4.2.3 Method 3 — UNet-based Autoencoder

- **Autoencoder Model Architecture and Training:** This approach is based on a UNet architecture. The encoder consists of five convolutional blocks—each featuring two convolutional layers with ReLU activations followed by max-pooling—that downsample the input progressively. The decoder mirrors the encoder with five upsampling blocks employing transposed convolutions and additional convolutional layers to recover spatial details, culminating in a final 1×1 convolution layer. The UNet autoencoder is trained for 50 epochs using the Adam optimizer (lr = 0.001) and MSE loss for reconstruction.
- **Segmentation Model Architecture and Training:** For segmentation, the pre-trained UNet encoder is extracted and frozen. A new segmentation decoder is constructed using five upconv blocks that employ transposed convolutions with ReLU activations, followed by a final convolution layer that produces predictions for each semantic class (3 classes in total). This segmentation model is fine-tuned on the labelled dataset using cross-entropy loss and the Adam optimiser (lr = 0.001) for 50 epochs, with performance assessed through IoU, Dice score, and pixel accuracy.

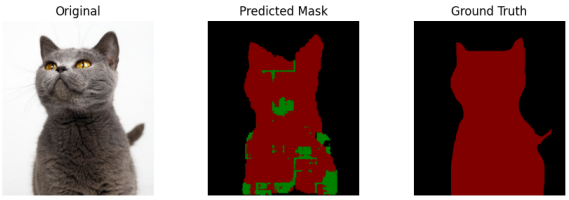

Figure 8. Comparison between the predicted mask and ground truth of a cat, using the UNet-based autoencoder model.

4.2.4 Results

The results in Table 2 show that the simplest autoencoder segmentation model achieves the best mean IoU among all. However, upon visualisation, it shows that all the predicted masks consist of the background, as shown in Figure 5. This shows that the model is too simple, failing to learn meaningful object features, and since the dataset has a large background region relative to the objects, always predicting the background will achieve an inflated IoU score.

The deeper architecture captures more spatial features than the simple autoencoder segmentation model and at least outlines the shape of the cat, as seen in Figure 6. However, it is classified as a dog, meaning the model does not fully capture the differences between the animals. It is the UNet-based model that achieves the best results, as seen in Figure 8. The predicted mask closely follows the cat's shape and is mostly correct, as the majority of it is composed of the cat class. This shows that the skip connections and a deeper architecture result in better segmentation performance, with

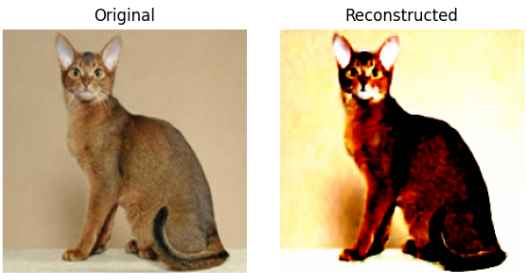

Figure 7. Comparison between the predicted mask and ground truth of a cat, using the improved autoencoder model.

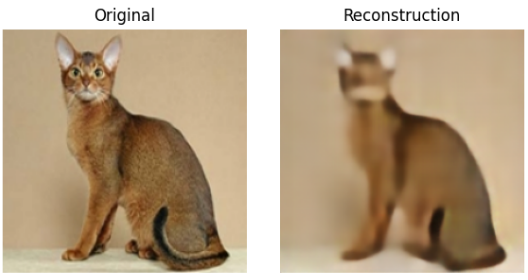

Figure 9. Comparison between the original and the reconstructed image using the UNet-based autoencoder model.

Method	Mean IoU	Dice Score	Pixel Accuracy
Simple Autoencoder	0.5682	0.6050	0.7002
Improved Autoencoder	0.3663	0.4350	0.7186
UNet-based Autoencoder	0.4399	0.5113	0.8028

Table 2. Performance comparison of the autoencoder methods on the test set.

accurate shapes and class identification.

When looking at the autoencoder reconstruction results, the simple and the improved models achieve similar results, as seen in Figures 4 and 7. The overall shape and silhouette of the animal are captured, but there is a noticeable struggle and colour details, resulting in over- and underexposure in certain regions. Compared to these, the UNet-based autoencoder, which uses skip connections, achieves better spatial coherence and more accurate colour consistency, as seen in Figure 9. Even though it produces a blurred-out with less pronounced details, it provides a more balanced and natural reconstruction in terms of structure and colours.

4.3. CLIP Features

Contrastive Language-Image Pre-training (CLIP), proposed by Radford et al. (2021), is a multimodal learning architecture designed to learn from both visual and textual data. It is trained on a large-scale dataset consisting of images paired with their corresponding natural language descriptions, enabling it to align images and text in a shared embedding space as shown in Figure 10. While CLIP is primarily used for tasks such as image classification and zero-shot recognition, its pre-trained image encoder can also serve as a powerful feature extractor for other vision tasks.

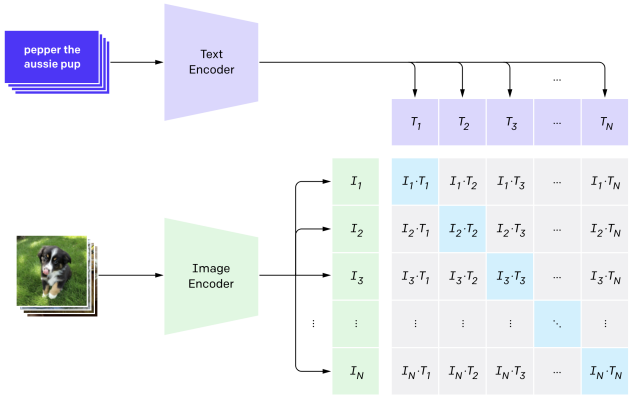

Figure 10. CLIP model uses contrastive learning to bring image and text pairs closer in the embedding space.

In our case, we use CLIP's pre-trained features as fixed inputs to the segmentation network. These features are rich in high-level semantic information, thanks to the contrastive learning objective used during pretraining. As a result, they tend to generalise well across different domains and datasets, even those not seen during training. This makes CLIP particularly suitable for image segmentation, where capturing semantic context and generalising to diverse visual patterns are crucial.

Model architecture:

We used the pre-trained CLIP visual encoder, `ViT-B/32` to extract high-level semantic features from the input image. A dedicated low-level encoder—consisting of two convolutional layers with 64 filters and ReLU activations—captures fine spatial details. Then, these features are passed through a decoder composed of four stages:

1. A transposed convolution upsamples the 768-channel CLIP features to 256 channels. Low-level features are pooled to a 14×14 resolution and fused (via concatenation and a 3×3 convolution) with these upsampled features.
2. Another transposed convolution upsamples the fused 256-channel features to 128 channels. Additional low-level features pooled to 28×28 are concatenated and refined through another convolution block.
3. A transposed convolution upsamples the 128-channel features to 64 channels, preparing the feature map for the final mask prediction.
4. A final transposed convolution upsamples the 64-channel features to the number of segmentation classes (3 channels in this implementation). Then, a bilinear upsampling operation scales the output to the original 224×224 resolution.

Training Procedure:

The model was trained for 100 epochs using the Adam optimizer, with an initial learning rate of $1e-4$ and L2 regularization with a weight decay also set to $1e-4$. As in previous models, we used a `ReduceLROnPlateau` to improve convergence.

The cross-entropy loss function was used with class weights: 1.0 for both dog and cat classes, and 0.1 for the background. This weighting discourages the model from over-predicting the background class and, therefore, obtains a more balanced segmentation performance across all categories.

4.3.1 Results

From the results in Table 3, we can see that it is the best-performing model out of all, achieving a Mean IoU of 0.8323. The UNet model effectively captured fine details but sometimes produced inconsistent class predictions. This limitation is overcome by the model using CLIP features, as seen in Figure 11. This illustrates that the animals are properly classified, even correctly classifying animals that are not shown in the ground truth.

Figure 11. Comparison between the predicted mask and ground truth, using the CLIP segmentation model, showing correct classifications in comparison to Figure 3.

Mean IoU	Dice Score	Pixel Accuracy
0.8323	0.9078	0.9258

Table 3. Performance of the CLIP segmentation model on the test set.

This shows that pre-training can improve segmentation results by providing a strong initialisation for the network. With pre-training, the model has already learnt useful features from a large amount of data, which can help the network capture low-level details. In this case, CLIP features help the segmentation model better distinguish between classes and reduce misclassifications, as seen in the visual results.

4.4. Prompt-Based Segmentation

To enable interactive segmentation, we implemented a prompt-based model that leverages CLIP's pre-trained visual encoder as a frozen feature extractor and integrates spatial prompts in the form of Gaussian heatmaps. The overall goal is to allow the segmentation of a region specified by a single annotated point in the input image.

Dataset Generation:
We modified the training data to include point-based prompts. For each object class present in a training mask, we randomly selected a pixel location belonging to that class. A 2D Gaussian heatmap centred at this point was generated and concatenated to the original image as a fourth input channel. This approach results in training samples of the form: (image, prompt heatmap, ground-truth mask).

Model Architecture:
The backbone of our model is the frozen CLIP visual transformer (`ViT-B/32`), as it was the best-performing model we obtained. The input image (3 channels) is passed through CLIP to produce a 768-dimensional feature map. In parallel, the prompt heatmap is downsampled via adaptive average pooling to match the feature map's spatial resolution and then concatenated, resulting in a 769-channel tensor.

A shallow low-level encoder (2 convolutional layers with ReLU activations) extracts fine spatial details from the original image. The concatenated features are then progressively upsampled by a decoder consisting of 4 transposed convolutional layers interleaved with two fusion blocks. The fusion blocks integrate low-level features that have been pooled to 14×14 and 28×28 resolutions, thus refining the segmentation boundaries. A final bilinear upsampling operation ensures the output mask matches the original 224×224 resolution.

Training Procedure:
We trained the network for 100 epochs using the Adam optimiser with weight decay and a learning rate scheduler that reduces the rate on plateaus. Cross-entropy loss with class weights was used to handle class imbalance, assigning a lower weight to the background (0.1) relative to object classes (1.0). Then, during validation, dummy zero heatmaps were supplied to preserve consistent input dimensionality.

4.4.1 Results

To evaluate model performance accurately, each input image from the validation or test set was first resized to the model's expected input resolution of 224×224 using an `albumentations` (Buslaev et al., 2020) transformation pipeline. The model then produced a predicted segmentation mask at this resolution, with class probabilities across the three categories (background, dog, and cat). To compare the prediction with the original ground-truth annotation, the predicted mask was resized back to the original image dimensions using nearest-neighbour interpolation, preserving

discrete label values. We then computed the per-class Dice coefficient, IoU, and pixel-wise accuracy using the resized predictions, with the results illustrated in Table 4.

Mean IoU	Dice Score	Pixel Accuracy
0.8915	0.9352	0.9354

Table 4. Performance comparison of the prompt-based segmentation model on the test set.

When visualising the results from the test set, we observe that the predicted masks closely match the ground truth, as seen in Figures 12 and 13. The boundaries around the cat and dog are generally well-defined, with minor discrepancies around the cat's fur edges or the dog's ears, which could be due to colour similarity to the background. This shows that the model can accurately respond to the single-point prompts, capturing the main shape of the target class.

Figure 12. Persian cat mask prediction and ground truth using a point-prompt.

Figure 13. English Cocker Spaniel dog mask prediction and ground truth using a point-prompt.

4.4.2 Prompt-Based Segmentation – User Interface

We developed an interactive segmentation app that showcases our prompt-based segmentation model, available at the following link[2]. Users can upload an image of their choice, and the app generates both a segmentation mask and an overlay of the original image, highlighting the detected object. To enhance interpretability, the interface includes a legend that maps each class (background, dog, cat) to a specific colour (blue, green, red), as well as a label indicating the detected class for the selected prompt.

[2] `https://huggingface.co/spaces/B176913/prompt-segmentation`

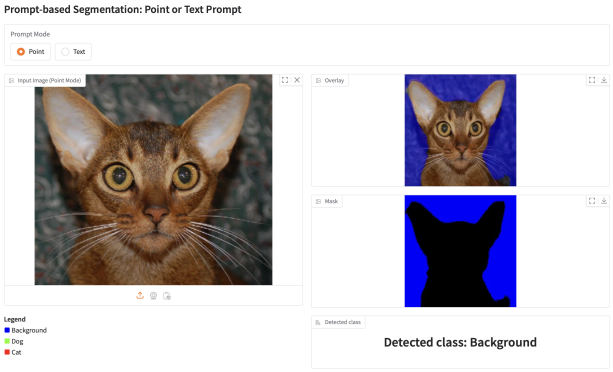

Figure 14. Gradio app interface in point-prompt mode. The user selects a location on the image, prompting the model to segment the corresponding object. In this case, the background is selected.

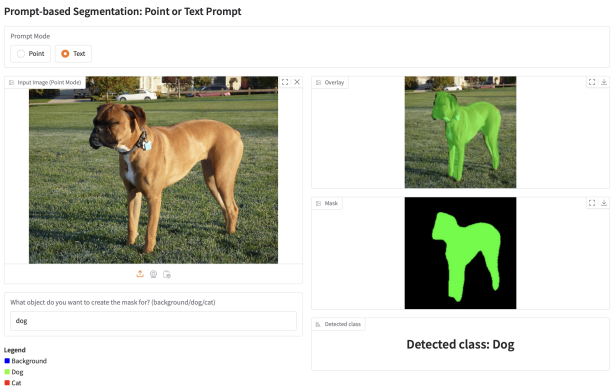

Figure 15. Gradio app interface in text-prompt mode. The user enters a class name, "dog" in this case, and the model segments the corresponding region.

The app was built using `Gradio` (Abid et al., 2019), a Python library specifically designed for the development of user interfaces for machine learning models. By default, the app operates in point-prompt mode, as shown in Figure 14, where users can simply click on a region of the image to indicate the object of interest. The model then uses the selected point to segment the corresponding object and displays the resulting mask and overlay.

As an extended feature, we also implemented text-prompt mode, allowing users to type a class name as an alternative form of input. This is illustrated in Figure 15. When a valid text prompt is entered, the app generates the corresponding segmentation.

5. Robustness Exploration

To assess the robustness of the segmentation models, we applied eight different types of perturbations to the test set, each with increasing levels of intensity, simulating real-

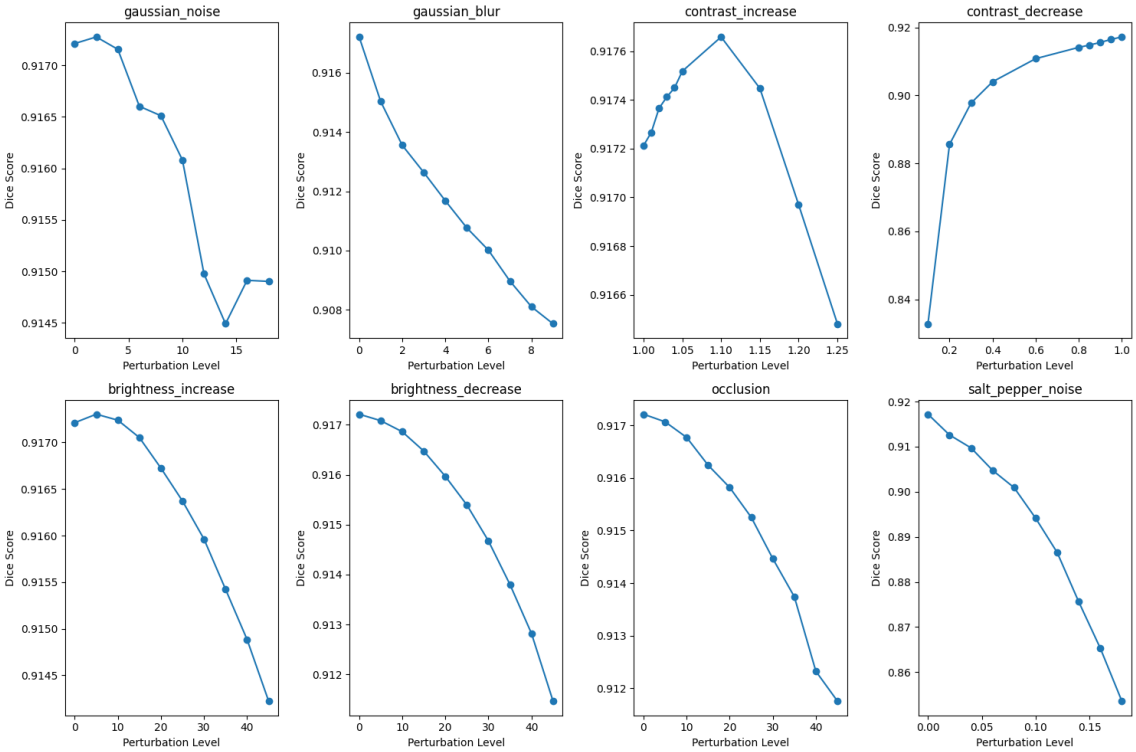

Figure 16. Mean Dice Score vs. Perturbation Strength on the Prompt-Based Segmentation Model.

world noise or distortions. The performance under perturbations was evaluated using Dice score, IoU, and pixel accuracy without retraining the model.

- **Gaussian Pixel Noise**: This algorithm adds noise by adding random values from a zero-mean Gaussian distribution to each pixel. The standard deviation of the distribution is progressively increased, intensifying the noise level. After the noise was added, we clipped the values to the range [0,255] in order to ensure valid image intensities.
- **Gaussian Blurring**: We apply a 3x3 kernel to the image, smoothing it by averaging pixel values with their neighbours. Each additional blur iteration reduces sharpness, simulating an out-of-focus effect or a motion blur. To do this, we use the OpenCV (Pulli et al., 2012) function `cv2.GaussianBlur()` iteratively.
- **Contrast Increase/Decrease**: We multiply each pixel's intensity by a scalar factor. For a factor greater than 1, the contrast is increased by exaggerating the difference between dark and light regions. For a factor less than 1, the contrast is decreased, compressing the range of intensities. We then clipped the results to the range [0,255].
- **Brightness Increase/Decrease**: We add or subtract a constant from every pixel value. Increasing brightness makes the image uniformly lighter while decreasing brightness darkens it. Clipping the values ensures that they remain within the acceptable range.
- **Occlusion**: We insert a black square (a block of zero-valued pixels) of varying sizes (from 5 to 45 pixels) at a random location in the image. As a result, it models a scenario where parts of the scene might be obstructed by another object. As the size of the occluding block increases, the model will lose spatial context.
- **Salt and Pepper Noise**: We introduce impulse noise by randomly replacing a proportion of the pixels with extreme values, either 0 for *'pepper'* or 255 for *'salt'*. This represents severe pixel corruptions, distorting texture and edge information.

Table 4 shows the prompt-based segmentation model outperformed the remaining models; therefore, we applied the perturbations here. Figure 16 illustrates the results, showing that it is robust. Minor perturbations such as light Gaussian noise and brightness adjustments maintain performance. As perturbation intensity increases–particularly with contrast increase and heavy salt and pepper noise– the Dice score drops more significantly. However, the Dice score never drops below 0.8, indicating the robustness of the model to different perturbations.

Method	Mean IoU	Dice Score	Pixel Accuracy
UNet	0.7278	0.7765	0.9088
Autoencoder	0.4399	0.5113	0.8028
CLIP Features	0.8323	0.9078	0.9258
Prompt-based	0.8915	0.9352	0.9354

Table 5. Performance comparison of the segmentation methods on the test set.

6. Challenges

We also encountered several well-known challenges associated with image segmentation throughout this project. One of the most prominent issues was dealing with ambiguous boundaries, where there are fuzzy transitions between the objects and the background. This is particularly problematic in our case, as we are segmenting animals, and their fur often blends into the surrounding environment, such as a blanket or a carpet. These soft boundaries make it difficult for the model to detect clear edges, ultimately impacting performance metrics like IoU.

Another significant challenge is the presence of small or thin structures. Features such as ears, tails or paws often occupy only a few pixels and are easily blurred out or completely missed in the predicted segmentation masks.

We also encountered the common issue of class imbalance. In segmentation masks, background pixels dominate compared to the foreground classes like “cat” or “dog”. As a result, the model tends to become biased toward predicting background regions, leading to under-segmentation of the actual objects of interest.

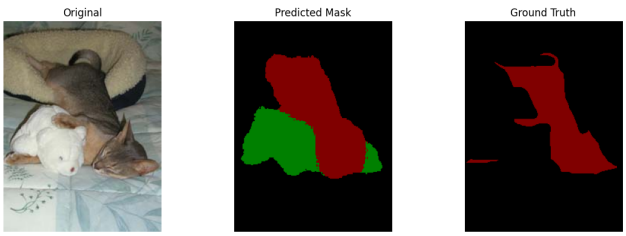

Figure 17. Failure case from the CLIP-based segmentation model. The model incorrectly segments a stuffed toy as a real pet.

In addition, visual variability posed a challenge. Animals appear in different poses, scales, light conditions, and even occlusions (e.g., partially behind furniture or other objects). This variability makes generalisation difficult.

Finally, we observed instances of confusing objects, where the model mistakenly predicted toys, such as stuffed animals, as real pets. This can be observed in Figure 17 and it is likely due to their visual similarity in shape, highlighting the difficulty of semantic understanding in segmentation tasks that rely uniquely on visual features.

7. Conclusion

After a thorough evaluation of different approaches to performing image segmentation, the results suggest that the best-performing model is the prompt-based segmentation leveraging CLIP features, as seen in Table 5. Among the first three examined models, including UNet and autoencoder-based architectures, the CLIP-based segmentation model consistently demonstrated superior performance. While CLIP may not always outperform task-specific architectures in fine-grained tasks, such as distinguishing between specific dog or cats breeds, its generalisation capabilities makes it particularly effective in our three-class segmentation. By incorporating prompt-based inputs into this architecture, we further enhanced its effectiveness, achieving a final mean IoU of 0.8915.

This improvement can be attributed to the rich semantic representations learned by CLIP during its pre-training on large-scale image-text pairs, which allow the model to generalise well across diverse visual settings. The addition of prompt-based conditioning further refines the model’s focus, helping it to more precisely localise the target object, particularly in complex scenes.

Nonetheless, it remains evident that even this top-performing model does not completely overcome all challenges. Issues such as handling high visual variability or dealing with semantically ambiguous objects still pose difficulties. Further improvements could be explored by incorporating refinement networks to better capture fine details and ambiguous boundaries. Additionally, training on more diverse and extensive datasets could help the model generalise even more effectively.

Overall, the results demonstrate that prompt-based segmentation using CLIP features is a powerful and flexible approach for image segmentation tasks. Its success highlights the value of combining powerful pre-trained representations with flexible prompt-guided inputs. The developed interface further shows the practical potential of such models in real-world applications.

A. Team Responsibilities

The work was divided fairly between the two of us:

- B177280: developed autoencoder and prompt-based segmentation models.
- B176913: developed UNet and CLIP models.

We worked together after we developed each model to make sure they were correct and then divided the report fairly.

References

Abubakar Abid, Ali Abdalla, Ali Abid, Dawood Khan, Abdulrahman Alfozan, and James Zou. Gradio: Hassle-free sharing and testing of ml models in the wild. *arXiv preprint arXiv:1906.02569*, 2019.

Alexander Buslaev, Vladimir I Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A Kalinin. Albumentations: fast and flexible image augmentations. *Information*, 11(2):125, 2020.

Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and C V Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3498–3505. IEEE, 2012.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Kari Pulli, Alexander Baksheev, Kirill Kornyakov, and Victor Eruhimov. Real-time computer vision with opencv. *Communications of the ACM*, 55(6):61–69, 2012. doi: 10.1145/2184319.2184337.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.