Exploring Image Segmentation Techniques

Abstract

In this report, we experiment with different deep learning approaches to investigate how different pre-training and feature extraction strategies affect segmentation performance.

1. Introduction

Image semantic segmentation involves partitioning an image into segments by classifying each pixel into a specific class. It plays a crucial role in computer vision tasks like object recognition, scene understanding, and medical analysis. We developed three segmentation models: UNet-based, autoencoder pre-training, and CLIP-based, using the same loss function and optimizer for fair comparison. This setup ensures that performance differences are due to the models' feature representation capabilities, not variations in training configuration. After evaluation, the best-performing model was refined into a prompt-based segmentation network with image point prompts.

1.1. Dataset Analysis

The dataset contains labeled images and corresponding ground-truth masks with pixel categories (background, cat, dog, and boundary which marks uncertain regions, and was merged into the background during training and ignored during testing). However, the dataset may not be balanced.

1.1.1 Dogs-to-Cats Ratio

To investigate this, we calculated the total number of cat images compared to dog images. This can lead to an unbalanced dataset where our segmentation model will have a bias towards predicting the majority class (dogs) for a pixel. This will also skew the performance metrics, where the model will perform well in predicting dog pixels but will have higher false negatives for predicting cat pixels. Any significant unbalance on the training dataset can reduce the model's performance when given a separate test dataset where there are equal or more cat images to dog images [1].

From the bar-plot, there are twice as many dog images to cat images. During the training dataset preparation phase, we used augmentations to further supplement the cat dataset size so that the cat to dog ratio is closer to 1:1.

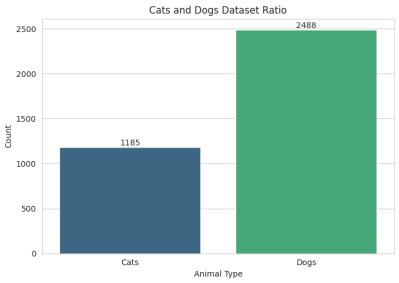


Figure 1. Plot of Cats vs Dogs Dataset Ratio

1.1.2 Counts per Category

We analyzed the sample counts per breed of cats and dogs. If some categories have too few samples, it can hinder the model's ability to classify them. Since the counts are relatively balanced across all breeds, no further data augmentation is necessary.

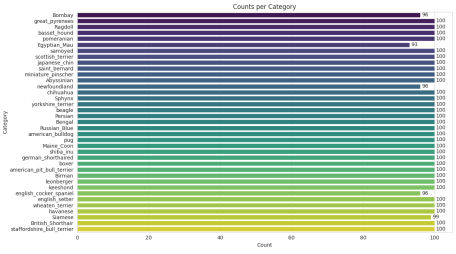


Figure 2. Plot of Counts per Categories

2. Image Augmentation

Image augmentation helps in feature robustness by exposing the model to a wide range of variations in the underlying image. This will aid the model in learning more generalizable representations of images containing similar objects.

2.1. Fixing the size of the Image

Based on our dataset, the images come in different dimensions. We need to fix the input image height and width in order for the model to predict a fixed dimension segmentation map. To achieve this, we explored resizing and cropping. Resizing downsamples the image to the specified width and height with a loss in pixel-level information. Cropping instead captures a part of the image and preserves the pixel-level information. However, random cropping requires multiple crops of the same image to fully capture the image context, and this also leads to complexity in managing the dataset. We chose to resize the images instead.

2.2. Flipping and Rotations

Flipping and rotating images help the model capture patterns that remain consistent under rotation, achieving rotation-equivariance. However, this may not always be helpful to the model if the images themselves are symmetrical in nature, such as nuclei objects [2]. In addition, equivariant U-Nets can achieve better performance in tasks for which local rotations in the image have an impact on the final image, but do not contain any information [2]. Since our dataset contains cats and dogs, and the rotation does not affect the ability to identify an object as a cat or dog, achieving rotation-equivariance should help improve the model's accuracy.

2.3. Elastic Transform

Elastic transform modifies images by applying local deformations that simulate physical distortions. The process begins by generating a grid that divides the image into smaller regions. Each point on the grid is then displaced by some random value. A Gaussian filter is usually applied to smooth out the displacements.

2.4. Color Jitter

Color Jitter modifies the brightness, contrast, saturation, and hue of an image. This is done by adding random values to the pixel intensity (brightness), multiplying random values to the pixel intensity (contrast), blending the image with a grayscale version (saturation) and adding a random offset to the hue component (hue). The combination of these modifications allows our model to generalize images with varying color characteristics.

3. UNet-based Segmentation

U-Net is a widely used architecture for image segmentation, featuring an encoder-decoder structure with skip connections that link corresponding layers to form its distinctive U-shape [3]. These connections help retain spatial information during downsampling and restore segmentation boundaries during upsampling, enhancing the model's ability to capture small objects and fine details.

3.1. Algorithm

3.1.1 Deep Learning Background

Deep learning utilizes artificial neural networks with multiple hierarchical layers. In these networks, each neuron calculates its value by computing a weighted sum of its inputs (adjusted by trainable bias terms), followed by a non-linear transformation through activation functions. The output of a single neuron in a layer can be described by:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{1}$$

$\mathbf{x}$ is the input vector, $\mathbf{W}$ is the weight matrix, $\mathbf{b}$ is the bias term, and f is the activation function.

These transformed signals are then propagated to subsequent layers. The network performs forward propagation to pass data through to the output layer, generating predictions. The loss function measures the discrepancy between predictions and ground truth.

To improve the model, deep learning uses backpropagation to adjust parameters. The gradients of the loss function with respect to the weights are calculated by applying the chain rule of derivatives, propagating the error backward from the output layer to the input layer. The gradient of the loss with respect to the weights can be written as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial y} \cdot \frac{\partial y}{\partial \mathbf{W}} \tag{2}$$

$\mathbf{W}$ is the weights, $\mathcal{L}$ is the loss function, $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$ is the gradient of $\mathcal{L}$ with respect to $\mathbf{W}$, $\frac{\partial \mathcal{L}}{\partial y}$ is the derivative of the loss with respect to the output y, $\frac{\partial y}{\partial \mathbf{W}}$ is the derivative of the output y with respect to $\mathbf{W}$.

Using the gradients, the weights are updated through gradient descent:

$$\mathbf{W} := \mathbf{W} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}} \tag{3}$$

η is the learning rate, $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$ is the gradient of the loss function with respect to the weights.

During training, the neural network iteratively applies forward and backward propagation to adjust the weights and biases, progressively refining its feature representations. This iterative process allows the network to gradually learn from simple features to more complex representations, thereby improving its performance. [4].

Convolutional Neural Networks (CNNs) are a powerful deep learning technique widely used in image processing. In the convolution operation, a set of learnable filters (also

called convolutional kernels) slide over input images, and each filter performs an element-wise multiplication with pixel values in the local patch it covers. The resulting values are then summed to produce a single output in the output feature map.

$$S(x,y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} I(x \cdot s + i,\ y \cdot s + j) \cdot K(i,j) \quad (4)$$

$S(x,y)$ is the value of the output feature map at position (x,y), $I(x \cdot s + i,\ y \cdot s + j)$ is the pixel value at position $(x \cdot s + i,\ y \cdot s + j)$ in the input image, and $K(i,j)$ is the filter weight at position (i,j). The stride s controls the step size with which the filter moves across the input.

By convolving the filter across the entire image with a predefined stride, the network generates a complete feature map that captures local patterns such as edges, textures, or color transitions. [5]. The key advantages of CNNs are weight sharing and local perception, where the same parameters are applied to the entire image, greatly reducing computational complexity and improving learning efficiency.

After convolution, pooling layers are typically applied to downsample the feature maps. Pooling reduces the spatial dimensions while retaining the most salient information, helping to lower computational cost and prevent overfitting. The most common form, max pooling, selects the maximum value within each patch of the feature map:

$$S(x,y) = \max_{0 \le i < k\ 0 \le j < k} I(x \cdot s + i, , y \cdot s + j) \quad (5)$$

$S(x,y)$ is the pooled output at position (x,y), $I(x \cdot s + i,\ y \cdot s + j)$ are the values in the local region of the input feature map, k is the kernel size, and s is the stride for pooling.

Through multiple layers of convolution and pooling, CNNs can progressively learn hierarchical features in images, ranging from low-level edges to high-level semantic concepts.

3.1.2 U-shape Architecture

Encoder In UNet, the image is first passed to the encoder of the model. The encoder aims to extract rich feature information from the image. It consists of multiple levels, with each level containing two convolutional layers followed by a max-pooling layer. The convolutional layer uses a small kernel size of 3x3. The reason for using small kernel size is to focus on pixel-level feature extraction, and they can be stacked to extract global patterns from the image. This also leads to a smaller number of trainable parameters estimated using the following formula. The input and output refers to channels.

$$\text{parameters} = (\text{kernel size} \times \text{input} + 1) \times \text{output} \quad (6)$$

The max-pooling layer extracts the most important features from the convolutional layer outputs, reducing the spatial dimensions of the feature maps. This helps the model in learning features at different spatial scales. and increases the model's translation invariance to small shifts. At each level of the encoder, the final convolution output before passing to the max-pooling layer, is saved as a skip-connection for the decoder.

Bottleneck The bottleneck layer is the bottom-most level. It has the smallest spatial dimensions but the highest number of feature channels. Composed of 2 convolutional layers, it serves as the transition point where the output from the encoder is passed to the decoder. This layer is crucial for segmentation performance, as too few feature channels can result in a lack of sufficient information for segmentation, while too many feature channels can cause overfitting due to poor generalization.

Decoder The decoder up-samples the bottleneck output to restore spatial details for precise boundary segmentation. UNet further improves this recovery and precision by concatenating the skip-connections from the encoder to the upsampled output. This combined output is then refined by passing through two convolutional layers. This is repeated for the same number of levels as the encoder.

Skip Connection Skip connections transfer intermediate feature maps from the encoder to the decoder before each upsampling step. Instead of relying solely on the compressed bottleneck representation, the decoder receives additional contextual information from earlier layers. This helps restore important spatial details for accurate segmentation.

3.1.3 Output

In the final convolutional layer at the top level, the model generates multi-class segmentation output. Each channel corresponds to one class, and the logits are assigned to each pixel. The output is then processed by the softmax activation function, which converts the logits into a probability distribution. This ensures that the sum of probabilities across all classes for each pixel equals 1. The class with the highest probability is chosen as the predicted label for each pixel, producing the final segmented image.

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad (7)$$

z_i is the raw score (logit) for the i-th class, e^{z_i} is the exponentiation of z_i. The denominator is the sum of the exponentials of all the raw scores, ensuring normalization.

3.2. Network Design

We used PyTorch to implement the UNet architecture and focused on modularity and flexibility.

3.2.1 Level

The 5-level UNet architecture is well demonstrated in segmentation tasks. Each level contains 2 consecutive 3×3 convolutional layers, followed by max pooling for downsampling. After bottleneck, transpose convolutions are used for upsampling restoration. Skip connections concatenate encoder features with corresponding decoder levels. [3].

3.2.2 Activation

Traditional activation functions like sigmoid and tanh have gradients that become very small (vanish) for inputs far from zero. In deep networks like U-Net which has a large number of layers, multiplying these small gradients during backpropagation can cause the vanishing gradient problem. For ReLU activation, it's function is given by:

$$f(x) = max(0, x) \tag{8}$$

The derivative of of the function when x $>$0 is 1. This means the activation function will have less impact on shrinking the gradient, and thus will not adversely affect the model training efficiency while retaining non-linearity.

3.2.3 Batch Normalization

We added a batch normalization layer after each convolutional layer to ensure that the output is normalized before passing to the activation. This is proven to stabilize training and speed up convergence [6].

4. Autoencoder-pretraining for Segmentation

In this approach, we first reconstruct the raw images using an autoencoder structure. After training the reconstruction network, we extract its fixed encoder's weights and add a segmentation network for the subsequent image segmentation task.

4.1. Algorithm

4.1.1 Theoretical Motivation

Theoretically speaking, pre-training the encoder of a segmentation model using an autoencoder reconstruction network offers several benefits:

Feature Compression By training the autoencoder to reconstruct input images, the encoder is forced to learn a compressed, low-dimensional representation that captures essential patterns such as edges, textures, and shapes. These representations are often transferable and beneficial for downstream tasks like segmentation. [7]

Feature Transfer Once the encoder is trained, its learned weights can be reused in the segmentation model. This type of transfer learning allows the network to start from a more informative initialization, rather than from random weights. [8]

Reduced Risk of Overfitting By freezing the pre-trained encoder and only training the segmentation decoder, the number of trainable parameters is significantly reduced. This is helpful when the labeled training data is limited, as it lowers the risk of overfitting. [9]

Smoother Optimization Trajectory A well-initialized encoder provides more stable and faster convergence in the later supervised training phase. This can lead to improved training dynamics and potentially better final performance. [9]

4.1.2 Pre-training via Image Reconstruction

The autoencoder is trained to reconstruct input images through a symmetrical encoder-decoder architecture.

Encoder The input image is first passed through the encoder, which consists of multiple layers. If the objective is only image reconstruction, convolutional layer would be sufficient. However, since our final aim is to pre-train the model for semantic segmentation, we include max-pooling layers after each convolutional layer. This addition helps to extract more robust features by inducing spatial hierarchy and promoting translation invariance—properties critical for robust semantic segmentation.

Bottleneck The compressed feature map, representing the most critical image information, is passed through the bottleneck layer. This layer holds the most significant, low-dimensional representation of the input image. By forcing the network to learn how to represent the image in this compressed space, the autoencoder encourages the extraction of essential features for downstream tasks like segmentation.

Decoder After passing through the bottleneck, the latent representation is fed into the decoder. The decoder uses

transposed convolution or upsampling layers to progressively reconstruct the image to its original resolution, aiming to recreate the input image based on the compressed features learned by the encoder.

Loss Function The reconstruction error is computed using a loss function, typically Mean Squared Error (MSE), which calculates the pixel-wise intensity difference between the input image and its reconstructed counterpart. Minimizing this error forces the model to learn a meaningful and compact representation of the input image.

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} \left(X_i - \hat{X}_i \right)^2 \tag{9}$$

X_i is the actual value (true value) of the i-th input pixel, $\hat{X}_i$ is the predicted value (the reconstructed value) for the i-th pixel by the model, N is the total number of pixels in the input image.

4.1.3 Segmentation with Frozen Pre-trained Encoder

After the encoder has been trained, the weights of the encoder are frozen to retain the learned features. A segmentation network is then added after the encoder.

The segmentation network acts as a decoder that mirrors the encoder's structure. It progressively upsamples the feature maps back to the original image resolution to produce a pixel-wise segmentation mask. To retain sufficient model capacity and learn hierarchical semantic features, multiple upsampling and convolution layers are used instead of a single prediction layer. Transposed convolutions followed by convolutions are commonly used to restore spatial details.

4.2. Network Design

4.2.1 Autoencoder

The encoder processes the input image (256×256×3) through three convolutional blocks, each with a 3×3 convolution (padding=1), batch normalization, ReLU activation, and 2×2 max-pooling (stride=2), reducing the spatial size to 32×32 and increasing channels from 3 to 256. The bottleneck output (32×32×256) captures high-level compressed features. In the decoder, three transposed convolutions with ReLU progressively upsample the features. The final output (32×32×256) is restored to the original size (256×256×3) via a 3×3 convolution followed by a sigmoid activation.

4.2.2 Segmentation Network

After pretraining, the encoder is frozen to extract compact features. The segmentation network takes the encoder output (32×32×256) and refines it through four convolutional layers. The first three use 3×3 convolutions (padding=1) with ReLU, reducing channels from 256 to 32. A final 1×1 convolution maps to the number of classes. The output is upsampled to the original image size 256×256 via bilinear interpolation to generate the final pixel-wise segmentation map.

5. Contrastive Language-Image Pre-Training (CLIP) for Segmentation

CLIP is a neural network trained on a variety of (image, text) pairs [10]. OpenAI trained CLIP on a huge dataset of 400M text-image pairs which allows it to build a strong understanding on image content in general [10]. In this approach, we aim to leverage CLIP's feature extraction capabilities to improve our segmentation performance. There are 2 architectures for CLIP, and the version we used utilizes vision transformer to process the image.

5.1. Algorithm

5.1.1 Vision Transformer Background

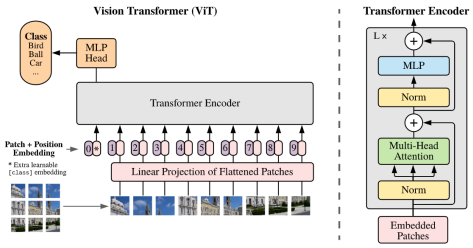


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

Figure 3. Vision Transformer [11]

The vision transformer aims to transfer the use of transformer architecture in natural language processing (NLP) into image recognition. Similar to how NLP transformers treat each word as a token, vision transformers treat images as a sequence of small patches. The transformer architecture allows the model to learn a global representation of the image content, and also the relationship between different patches. This could allow the model to better grasp object segments that may appear in different patches of the image, giving it a better understanding on the whole object representation.

When presented on large datasets, vision transformers prove to perform comparatively better than state-of-the-art Convolutional Neural Networks (CNN) while requiring fewer computational resources for training [11].

5.1.2 CLIP Architecture

CLIP employs a dual-encoder architecture designed to learn joint representations of images and text.

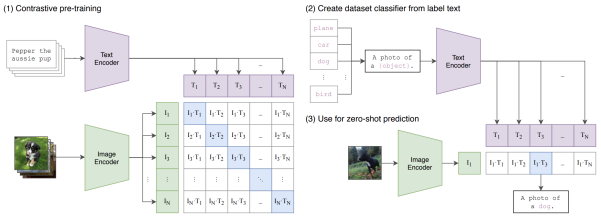


Figure 1. Summary of our approach. While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes.

Figure 4. CLIP's architecture

Contrastive Learning CLIP utilizes both the vision transformer and NLP transformer to generate individual embeddings for both the image and text. By training a model that learns to predict which captions go with which image, the model weights become a latent space shared by both image and text [10]. These learned representations by contrasting positive pairs (an image and its corresponding text) with negative pairs (an image and unrelated text) are also known as contrastive learning. Similar images and texts will appear to be closer to each other in that latent space based on cosine similarity. This can be calculated as follows:

$$\text{cosine\_similarity}(I, T) = \frac{I \cdot T}{\|I\|_2 \|T\|_2} \tag{10}$$

I represents the embedding vector produced by the Image Encoder for a given image. T represents the embedding vector produced by the Text Encoder for a given text description. The cosine similarity is used as the contrastive loss in order for the model to learn image and text pairs. The final output is the provided text with the highest similarity score to the provided image.

5.1.3 Adapting CLIP for Segmentation

Image Encoder Within the CLIP architecture, before the image embedding is projected together with the text embedding in the shared latent space, we want to use the image embedding that has been generated by the vision transformer as it contains the image features. Since semantic segmentation requires spatial information in order to classify individual pixels, the individual patch embeddings retain this spatial correspondence (before being flattened into a sequence), and thus can be transformed back into it's spatial dimensions.

Decoder In the decoder, it follows a similar architecture to UNet's decoder without the use of skip connections. This is because CLIP's vision transformer directly provides the final image embeddings and there aren't any intermediate steps which we can be used as skip connections. The reshaped embeddings are repeatedly up-sampled and convolved to increase spatial resolution and allow for learning how individual pixels should be classified.

5.2. Network Design

Choice of CLIP Model The encoder uses the provided CLIP vision model and processor from HuggingFace, specifically (openai/clip-vit-base-patch32). We chose to use the vision transformer variant over the ResNet variant because vision transformer has been shown to perform on par or even better than state-of-the-art CNNs with fewer computational resources [11]. A patch with size of 32x32 should be a good balance as it satisfies our computational resource constraint. The smaller patch size variants such as 16x16 can lead to much longer input sequences for the transformer (196 patches for a standard 224x224 input vs. 49 patches for the 32x32 variant). This quadratic increase in sequence length raises the computational and memory requirements significantly, potentially exceeding our allocated resources.

Reshaping of Patch Embeddings The provided image patch embeddings include a [CLS] token that provides a summary embedding of the image content. However, the [CLS] token lacks the spatial correspondence needed by the downstream decoder. Therefore, we excluded the [CLS] token and utilize only the sequence of patch embeddings. These patch embeddings are reshaped back into their spatial grid arrangement allowing us to create feature maps (768x7x7).

We also added a linear projection layer to convolve the original image embedding vector length of 768 to 1024, as this conforms to the bottleneck size that is used in our previous UNET architecture. This ensures a much cleaner comparison between the effectiveness of CLIP's feature extraction capabilities and the usage of skip connections.

Decoder The decoder emulates the same architecture as the UNET decoder. Due to the smaller image dimension in the bottleneck (7x7), the output segmentation map needs to be resized to the dimensions of the original input image dimension.

6. Training Process

6.1. Configuration

Batch Size The batch size is determined based on our GPU memory constraint, time constraint and model perfor-

mance. Larger batch sizes requires larger GPU memory allocation, increase training speed, which can lead to poorer performance [12]. As we prioritize the comparison across models over each individual model accuracy (but enough to pass the IOU baseline 0.33), we chose a moderate batch size of 64 that balances between performance and training speed.

Learning Rate As U-Net needs to be trained from scratch, it is given a slightly larger learning rate (1e-4) than the pre-trained models (1e-5) which are used to fine-tune the segmentation network. This is because the initial weights of U-Net are expected to be far away from the optimal solution.

Number of Epochs We defined the number of epochs to be large (50) because we implemented an early stopping mechanism explained in 6.2.

6.2. Model Training

6.2.1 Validation and Early Stopping

To prevent the possibility of overfitting our model, we randomly split the initial dataset into training and validation with a 9:1 ratio. We chose random split because it is simpler to implement. The drawback of random splitting is that it will certainly lead to a bias in the model if the dataset is imbalanced [13], but since we supplemented more cat data points using augmentation to achieve a 1:1 cat to dog ratio, this reduces the disadvantage of random splitting. The ratio of the split was chosen based on the rough standard for train-validation splits [13].

Using the validation dataset to check the validation loss after every epoch, we implemented an early stopping mechanism that considers 2 parameters: delta and patience. Delta represents the threshold to identify if a model's performance is stagnating from training. It uses the difference between the validation score for the current epoch and the previous epoch. Patience considers how many consecutive times the model's training stagnates, and terminates the model training once the patience is reached.

6.2.2 Loss Function

Loss function dictates how the loss between the input image and target segmentation mask should be calculated. Since we are predicting classes for each pixel, loss functions that are appropriate for class predictions should be selected.

Cross-Entropy Loss

$$\text{Cross-Entropy Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,j}\log(p_{i,j}) \quad (11)$$

$y_{i,j}$ is the true label for the i-th sample and j-th class, where $y_{i,j} = 1$ if the i-th sample belongs to class j, and $y_{i,j} = 0$ otherwise. $p_{i,j}$ is the predicted probability for the i-th sample belonging to class j, C is the number of the classes.

Cross-Entropy Loss compares the predicted probability distribution and the true probability distribution. The model aims to minimize this difference to align the predicted probability distribution as close as possible to the true probability distribution.

Dice Loss

$$\text{Dice Loss}_j = 1 - \frac{2\sum_i y_{i,j}p_{i,j}}{\sum_i y_{i,j} + \sum_i p_{i,j}} \quad (12)$$

$$\text{Overall Dice Loss} = \frac{1}{C}\sum_{c=1}^{C}\text{Dice Loss}_j \quad (13)$$

$y_{i,j}$ is the true label for the i-th pixel in class j, $p_{i,j}$ is the predicted probability for the i-th pixel in class j, C is the number of the classes.

The dice loss uses the dice coefficient to calculate the overlap between the predicted mask and the ground truth mask. Increasing overlap will also increase the dice coefficient, so the loss is calculated to be 1 - dice coefficient as per definition of loss function.

Experimenting with both loss functions, we discovered that cross-entropy loss provided a better convergence than Dice loss. During the early training stage, the Dice loss relies on the global sum across all pixels which vary significantly to changes in the predicted mask while cross-entropy calculates the loss of each pixel independently. we chose to use cross-entropy loss due to it's stability during early training.

6.2.3 Optimizer

The choice of our optimizer is Adam. It has been demonstrated to perform well in image segmentation tasks [14]. Adam combines the advantages of the Momentum and RMSprop optimization algorithms. Compared to stochastic gradient descent, it computes an adaptive learning rate using the first and second moments of the gradient, allowing it to perform well with noisy gradients.

7. Evaluation

7.1. Intersection over Union (IoU)

IoU is used to measure the overlap between the predicted area and the true area.

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}} \quad (14)$$

Intersection is the area of overlap between the predicted region and the ground truth region. Union is the total area covered by both the predicted region and the ground truth region.

7.2. Dice coefficient (Dice)

The Dice coefficient calculates the similarity by measuring the ratio of the intersection to the sum of the areas of the two regions. The value of the Dice coefficient ranges from 0 to 1, where a higher value indicates a greater overlap between the prediction and the ground truth.

$$\text{Dice} = \frac{2|A \cap B|}{|A| + |B|} \tag{15}$$

A and B represent the predicted region and the ground truth region, respectively, with $|A \cap B|$ denoting the size of the intersection between the predicted and ground truth regions. $|A|$ and $|B|$ represent the total number of pixels in the predicted region and the ground truth region, respectively.

7.3. Accuracy

Accuracy measures the proportion of correctly classified pixels (both foreground and background) out of the total number of pixels. The higher the accuracy, the better the model's overall performance in correctly classifying pixels.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{16}$$

For a specified class c, TP is number of pixels correctly predicted as belonging to class c, TN is the number of pixels correctly predicted as not belonging to class c, FP is the number of pixels incorrectly predicted to class c, FN is the number of pixels belonging to class c but misclassified as other classes.

7.4. Performance

Table 1. Comparison of Segmentation Model Performance (Mean IoU)

Model	IoU	Dice	Accuracy
UNET	36	45	82
Autoencoder	23	30	77
CLIP	55	61	94

8. Discussion

8.1. Impact of Network Architecture on Model Performance

The U-Net model achieved moderate performance, effectively balancing spatial detail preservation and semantic feature learning through its hierarchical architecture. Compared to traditional Fully Convolutional Networks (FCN), U-Net addresses the issue of losing low-level features in deeper layers by using skip connections. These connections directly fuse shallow spatial information (from early encoder layers) with deep semantic representations (from decoder layers), enabling simultaneous recovery of fine-grained details and high-level contextual understanding. However, U-Net is trained on a limited number of image samples, which can make it difficult to be proficient in generalizing features and ignoring noise.

Figure 5. UNet: An Example of Mask Prediction

The Autoencoder-pretrained segmentation model performed relatively poorly. While the pretraining step allowed the model to learn some basic feature representations, it struggled with details, such as the ears and tail in the cat segmentation task, and much of the cat's body was also swallowed by the background. This limitation arises from the fact that autoencoders are designed to focus on reconstructing global structures, rather than emphasizing local boundaries, which are crucial for precise segmentation. As a result, although the model was able to capture the most basic body structure of the cat, it lacked the spatial precision and boundary sensitivity required to predict finer details, highlighting the need for better localization in the pretraining phase. For future improvement, we may try using loss functions like Structural Similarity Index Loss (SSIM Loss) or Boundary-Aware Loss during image reconstruction pretraining, as they are better suited for preserving structural details and enhancing the model's sensitivity to boundaries.

Figure 6. Autoencoder Pre-trained Model: An Example of Mask Prediction

The CLIP-based model demonstrated superior performance. The strong performance of it can be attributed to the richness of the pre-trained knowledge, enabling the model to generalize better to unseen data and complex segmentation tasks.

However, it still failed to accurately segment fine details such as the cat's extended paws and ears. This may due to the low spatial resolution in CLIP's visual encoder (e.g., 32×32 patch size in ViT-B/32 [11]), which limits the model's ability to capture small and fine-grained structures. In addition, CLIP tends to focus on global semantics (e.g., recognizing "a cat") rather than precise localization of parts, leading to a semantic bias toward most prominent features like cat's face. Furthermore, its web-scraped training data may underrepresent uncommon poses, such as stretched limbs, resulting in limited generalization to such cases.

Additionally, CLIP may misclassify parts of the background as the cat, especially when background areas share similar dark colors with the cat's fur, leading to segmentation errors at the darker border areas of the background.

Figure 7. CLIP Segmentation Model: An Example of Mask Prediction

8.2. Key Challenges and Future Improvement

8.2.1 Class Imbalance at the Pixel Level

In a single image, the background often dominates in pixel count, leading to class imbalance and biasing the model toward predicting the background. This results in higher evaluation scores for the background compared to minority classes like cats and dogs. To mitigate this, future work could replace default cross-entropy loss with class-balanced alternatives such as weighted cross-entropy, focal loss, or a hybrid of Dice loss and cross-entropy to improve performance on underrepresented classes [15].

8.2.2 Handling the Boundary Class

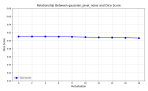
The original masks include a boundary class, marking uncertain regions between the foreground (cats or dogs) and the background. In our setup, we merged this class into the background during training and ignored it during testing. While this reduced the number of classes to predict, it increased class imbalance by further inflating the background pixel count. Future improvements could involve treating the boundary as void, object or a separate class to try better handling these regions.

8.2.3 Thin or Pointed Objects

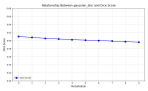
Thin or pointed objects features are difficult to learn due to their small representation, making them easily overshadowed by the background. Low resolution and multi-scale issues, such as detail loss from increased network depth and pooling layers reducing spatial resolution, further hinder the capture of these features. Future improvements could include using higher-resolution networks, introducing more advanced multi-scale techniques, or employing attention mechanisms to better focus on fine details and enhance segmentation performance.

8.3. Perturbation Analysis

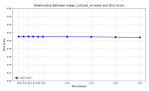
Different perturbations are applied in increasing and decreasing intensity to the images to evaluate the robustness of the CLIP segmentation model.



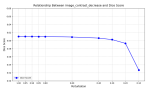
(a) Gaussian Pixel Noise



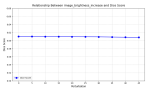
(b) Gaussian Blurring



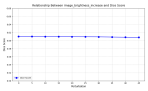
(c) Image Contrast Increase



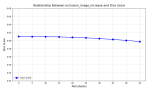
(d) Image Contrast Decrease



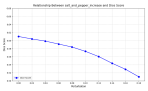
(e) Image Brightness Increase



(f) Image Brightness Decrease



(g) Occlusion of the Image Increase



(h) Salt and Pepper Noise

8.3.1 Gaussian Pixel Noise

This perturbation adds noise to each pixel.

$$I'(x, y) = I(x, y) + N(x, y) \tag{17}$$

The noise N is determined by picking a random number taken from the Gaussian normal distribution. The model performed well because the perturbation does not remove underlying edges in the image.

8.3.2 Gaussian Blur

This perturbation convolves the image, with each convolution increasing the standard deviation of the approximate gaussian blur. This is due to the equation of combined variance.

$$\sigma_{total}^2 = \sigma_1^2 + \sigma_2^2 \tag{18}$$

The model performed worse than gaussian pixel noise due to removing high frequency details such as edges by using a weighted averaging of neighboring pixels. However, performance did not degrade much because of prior augmentations in training set.

8.3.3 Image Contrast

Increasing contrast makes the difference between segments in the image larger, while decreasing contrast compresses the range of pixel intensities, which increases the difficulty for the model to detect edges and textures. The model suffered in performance for decreasing contrast while it remained relatively robust to increasing contrast.

8.3.4 Image Brightness

Changing image brightness is done by adding a constant value to all pixels. This does not affect the relative differences between pixels, and thus the model performance remained relatively robust.

8.3.5 Occlusion

Occlusion removes a portion of the image by replacing the pixels with 0s. The model remained robust because the global context of the image is well captured by the vision transformer.

8.3.6 Salt and Pepper Noise

Salt and pepper replaces random pixels with black and white, which disrupts the vision transformer capability to operate on pixel-level information, thus leading to a decline in model performance.

9. Prompt Based Segmentation with CLIP

Using our previous highest performing segmentation model (CLIP), we adapt the model architecture to also support an additional point input. The prompt-based segmentation will predict the object where the point lies on.

9.1. Dataset Preparation

We generated multiple points from the same image that correspond to different segmentation masks. Since the images mostly consist of a single object and a background, we randomly sample a point within the object, and another point within the background. The ground truth pixel class will correspond to the object which the point lies on. The generated point is converted into a gaussian heatmap so that it conveys the spatial positioning of the point. The gaussian heatmap is chosen with a small standard deviation as we want to have sharp localized peaks for precise spatial positioning of the clicks.

9.2. Algorithm

The adaption to the clip segmentation model lies in taking account of the additional generated point heatmap.

Encoder The point heatmap is passed into CLIP to generate the image embedding. This is because we hypothesize that the vision transformer will be more effective in extracting the spatial information from the point heatmap compared to a custom encoder used for handling the point heatmap which requires training from scratch. We chose to concatenate both image and point heatmap features together to achieve a representation that includes the context of the image and the point.

Decoder To accommodate the increase in number of feature maps (1536), the decoder needs to be modified to support up-sampling with an input channel of (1536). The rest of the decoder behave similarly to the CLIP segmentation decoder mentioned previously.

9.3. Network Design

we removed the linear projection layer that originally converts the feature maps to have the same size as the UNet implementation. This is because the current number of feature maps after concatenating the point heatmap exceeds the size which UNet implements. We want to preserve the richer semantics that exist in this combined representation.

9.4. Performance

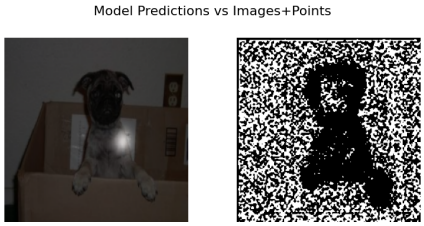


Figure 9. Point CLIP Segmentation: Example of Mask Prediction

Table 2. Prompt-Based Segmentation Performance

Metric	Score (%)
IOU	37
Dice	53
Accuracy	56

The CLIP segmentation had a drop in IOU after integrating with an extra point prompt input likely due to the increase in complexity and not enough training. To improve performance, we can perhaps provide larger training epochs to improve the model's understanding of the extra point prompt.

10. Appendix

The folder structures are divided as follow:

10.1. Dataset

1. **augmentation.py** Augmentation definitions on the dataset.

2. **dataset_prep.py** Dataset preparation script to be ran before training. Helps with reconstructing the dataset directory to be split by cats and dogs directory. It takes the base directory as an CLI argument.

3. **exploration.py** Data exploration script that plots the dataset distribution of cats and dogs, and the different breeds.

4. **pet_heatmap.py** The point heatmap pet dataset used for point clip segmentation.

5. **pet.py** The pet dataset used for training and testing unet, autoencoder, clip.

10.2. Model

Each model follows the directory structure:

1. **model.py** Contains model architecture definition.

2. **train.py** Contains the training loop including the optimizer, loss functions and configuration parameters.

3. **test.py** Contains test loop to test the IoU, dice and accuracy score.

10.3. Utilities

1. **dataset_prep.py** This contains the functions used for generating data point prompts and segmentation masks for the prompt-based segmentation model, plotting heatmaps and saving masks to image files.

2. **dataset.py** Contain functions that convert the mask pixels to classes, and pixel to class mapping.

3. **helper.py** Contain functions to load and save model checkpoints.

4. **metric.py** Contain metrics such as iou, dice and accuracy calculations. The validation loop is also placed here.

5. **train.py** Contain early stopping and training log definitions.

References

[1] Ali Mirzaeli. Inbalanced classification. *https://www.linkedin.com/pulse/imbalanced-classification-ali-mirzaei*, 31 Mar 2021. 1

[2] Bruno Dumas Benoît Frénay Robin Ghyselinck, Valentin Delchevalerie. On the effectiveness of rotation-equivariance in u-net: A benchmark for image segmentation. *https://arxiv.org/html/2412.09182v1S6*, 12 Dec 2024. 2

[3] Thomas Brox Olaf Ronneberger, Philipp Fischer. U-net: Convolutional networks for biomedical image segmentation. *https://arxiv.org/pdf/1505.04597*, 18 May 2015. 2, 4

[4] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. 2

[5] Keiron O'shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015. 3

[6] Christian Szegedy Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 11 Feb 2015. 4

[7] Hisashi Shimodaira. Improving prediction accuracy of semantic segmentation methods using convolutional autoencoder based pre-processing layers. *arXiv preprint arXiv:2404.12718*, 2024. 4

[8] Jonas Dippel, Matthias Lenga, Thomas Goerttler, Klaus Obermayer, and Johannes Höhne. Transfer learning for segmentation problems: Choose the right encoder and skip the decoder. *arXiv preprint arXiv:2207.14508*, 2022. 4

[9] Davood Karimi, Simon K Warfield, and Ali Gholipour. Transfer learning in medical image segmentation: New insights from analysis of the dynamics of model parameters and learned representations. *Artificial intelligence in medicine*, 116:102078, 2021. 4

[10] Chris Hallacy Aditya Ramesh Gabriel Goh Sandhini Agarwal Girish Sastry Amanda Askell Pamela Mishkin Jack Clark Gretchen Krueger Ilya Sutskever Alec Radford, Jong Wook Kim. Learning transferable visual models from natural language supervision. *https://arxiv.org/pdf/2103.00020*, 26 Feb 2021. 5, 6

[11] Alexander Kolesnikov Dirk Weissenborn Xiaohua Zhai Thomas Unterthiner Mostafa Dehghani Matthias Minderer Georg Heigold Sylvain Gelly Jakob Uszkoreit Neil Houlsby Alexey Dosovitskiy, Lucas Beyer. An image is worth 16x16 words: Transformers for image recognition at scale. *https://arxiv.org/pdf/2010.11929*, 3 Jun 2021. 5, 6, 9

[12] Jorge Nocedal Mikhail Smelyanskiy Ping Tak Peter Tang Nitish Shirish Keskar, Dheevatsa Mudigere. On large-batch training for deep learning: Generalization gap and sharp minima. *https://arxiv.org/pdf/1609.04836*, 9 Feb 2017. 7

[13] Nikolaj Buhl. Training, validation, test split for machine learning datasets. *https://encord.com/blog/train-val-test-split*, 19 Nov 2024. 7

[14] M Sultan Zia Kaleem Arshid Kebin Jia Zaka Ur Rehman Atif Mehmood Muhammad Yaqub, Jinchao Feng. State-of-the-art cnn optimizer for brain

tumor segmentation in magnetic resonance images. *https://pmc.ncbi.nlm.nih.gov/articles/PMC7407771*, 03 Jul 2020. 7

[15] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017. 9