A Comparative Study of Deep Learning Approaches for Semantic Segmentation on the Oxford-IIIT Pet Dataset

Rhodri Thomas Stanislas Castellana

April 10, 2025

Abstract

This project presents a comparative study of four deep learning approaches to semantic segmentation on the Oxford-IIIT Pet Dataset: U-Net, convolutional autoencoders, CLIP-based models, and prompt-guided segmentation. Each model is trained and evaluated under a consistent framework, with particular focus on robustness to real-world image perturbations and inference under point-based user prompts. We implement a lightweight decoder for CLIP and analyze its performance relative to more traditional CNN-based architectures. Our results show that CLIP outperforms other models in accuracy, IoU score, and Dice score, demonstrating strong generalization and robustness to noise, while U-Net produces the most precise boundaries. The autoencoder suffers from overfitting, despite fast convergence.

1 Introduction

Image segmentation has made dramatic leap forwards in the recent past with improvements in convolutional models ([9], [3], [17] [18]), the application of transformers to vision tasks [2], and even more recently the release of foundational vision models [1]. In this project we examined three modelling approaches to image segmentation - U-Net, autoencoders and CLIP (Contrastive Language-Image Pre-training). We analysed their performance and tested the robustness of the highest performing model by introducing a range of perturbations in the test data. We also built a point prompt based architecture with a GUI to test its impact on performance in predicting a mask. The dataset was a pre-prepared subset of cat and dog images and masks from the Oxford-IIIT Pet Dataset [13].

1.1 Models/ approaches used

1.1.1 U-Net

The U-Net architecture was originally proposed in 2015 by Ronneberger *et al.*[17] for pixel level classification in biomedical research. The best convolutional models for such tasks at the time used sliding windows over the image to generate patches. Individual models were then run for each patch. The ability of the U-Net architecture to capture local information and context in a single model made it much more efficient. The original U-Net model consisted of a contracting path (encoder) followed by a broadly symmetrical expanding path (decoder). The blocks in the contracting path comprised of two convolutional filters with rectified linear unit (ReLU) activation functions and then max pooling to double the features. The blocks in the expanding path comprised of upsampling, then two transposed convolutional filters with ReLU that ha;f the number of features each time. The final layer was a convolutional filter that reduces the dimensions to the required number of output channels. Key components of U-Net are the 'skip connections'. These connections project and concatenate the features maps from the block in the contracting path to the corresponding block in the expanding path. This allows valuable spatial information (higher frequency) [21] to be retained that could otherwise be lost in subsequent blocks in the contracting path.

1.1.2 Autoencoder

Autoencoders also comprise an encoder and decoder but are trained, initially at least, to recreate the original image [21]. The encoder outputs a key feature representation of the image in a compressed latent space. After passing through this 'bottleneck' the decoder aims to recreate the image from the compressed feature representation. The ability of autoencoders to learn the key features of an image enable them to form the basis for image compression, denoising, anomaly detection or generative tasks [4]. For image related tasks, full convolutional autoencoders perform well given their ability to capture localised information and context. For image segmentation tasks, the compressed features encoded from the input images (outputs from the trained encoder) are used to train a new decoder with the masks as the ground truth.

1.1.3 CLIP

The CLIP model published by Radford et al. in 2021 [15] uses two transformer based encoders (Figure 5: one for an input image and one for associated text. The model was trained on a dataset of 400 million images and accompanying text. The encoders embed the image and text into a shared embedding space. A contrastive loss function is used so that the model learns to co-locate corresponding images and text in this shared space and separate unrelated text and images. The CLIP model has been shown to perform well with no additional training (zero-shot approach) ver-

sus models specifically trained for classification tasks [15]. The ClipSeg model published in 2022 [11] built on CLIP to tackle image segmentation. In ClipSeg the CLIP encoders are used to create embeddings of image and / or text prompts that are combined with activations from layers in the encoder. Decoder blocks take the activation layers from the encoder as input, reduce the dimensions with a linear function, and condition the output based on the text and / or image prompt embeddings before applying multi-head attention [11]. Transposed convolutions are then applied to the outputs of these blocks to reconstruct the spatial information required. [14]. The projection of activations from mid layers in the encoder to the decoder is similar to skip connections in U-Net It provides the decoder with additional context. Within the decoder blocks the features are conditioned by applying feature wise linear modulation (FiLM). This is an affine transformation of the feature using a combination (element wise multiplication) of the image and text embeddings from their respective encoders.

1.1.4 Prompt-based segmentation

Prompt-based segmentation has taken off over the last few years with the release of foundational models for image and video segmentation (*e.g.* Segment Anything Model (SAM) [8] and Segment Anything Model 2 (SAM 2) [16]). These models have been trained on millions of images and billions of masks and are effective at segmenting images an videos into valid masks based on a given prompt. They are even effective on objects outside of the training data (zeroshot) and the pre-trained encoder and a simple decoder allows segmentation to occur in near real time. The models work by creating an embedding of the image (in the case of SAM) from the pre-trained encoder and combined this with a vector embedding of the prompt (*e.g.* a point, areas or bounding box or text) before passing through the decoder. The SAM decoder comprises of two modified transformer blocks that include self-attention and cross-attention between the image and prompt. The embedding are then upsampled and a multi-layer perceptron (MLP) and linear classifier are used to predict the probability of masks at the specified locations.

2 Preprocessing

2.1 Data Description and Cleansing

The preprepared dataset of cat and dog images from The Oxford-IIIT Pet Dataset [13] had been split into a training & validation dataset (TrainVal) of 3,680 image/mask pairs and a test dataset (Test) of 3,710.

1,185 of the TrainVal image/mask pairs are of a cat and 2,488 of a dog. No images/mask pairs contain both. The pixels in the masks are in one of four classes: background (0); cat (1); dog (2), and outline (255). The outline pixels have been changed (to background) as per instructions to leave three classes.

In seven of the TrainVal masks all pixels are classified as background despite the images containing cats and dogs. Given this represents only 0.2% of all TrainVal images we chose not to remove them. Notably, the Test dataset also contains such masks as highlighted below.

The images/masks pairs in the TrainVal data vary in size between 15,552 (W 144 x H 108) and 7,990,272 pixels (W 3,264 x H 2,448). The average is 171,954 pixels with a standard deviation of 197,930. The aspect ratios vary between 0.52 and 2.56.

The Test dataset comprises 1,203 cat and 2,491 dog image/mask pairs. No images/mask in the Test dataset contain both either. However, there are 16 masks with only background pixels in the Test dataset. The images/masks pairs in the Test data vary in size between 14,111 (W 137 x H 103) and 3,110,400 pixels (W 1,440 x H 2,160). The average is 177,745 pixels with a standard deviation of 101,215. The aspect ratios vary between 0.42 and 2.21.

The above comparison indicates that the TrainVal dataset represents a valid representation of the size and distribution of the Test data (also see Appendix).

2.2 Resizing

For the U-Net and autoencoder the image and masks were resized to dimensions of 256 x 256 x 3. The dimensions were chosen after visual inspection of the images (figure 1). The 256 x 256 dimensions balanced maintaining image detail with minimising the dataset size to enable efficient training with our compute power. Not only do bigger images need more FLOPS to train models in the same amount of time, but the data space increases exponentially from $\mathbb{R}^{256\times256\times3}$ to $\mathbb{R}^{512\times512\times3}$. For bigger data space, generally many more data points, image and mask pairs, are needed. Likewise we confirmed that the images with the largest change in aspect ratio remained recognisable as cats and dogs figure 2.

Torchvision's V2 resize transform was used to change the dimensions. Bilinear interpolation was used for the images. It was sufficient to maintain image quality based on visual checks (*e.g.* Figure 1. Nearest neighbour was used for the masks to ensure pixel values remained in one of the three classes.

For CLIP we replicated the image and mask dimensions used in Lüddecke and Ecker (2022) [11]. The image input size is 352 x 352 and the masks inputs 224 x 224. We resized them separately and carried out the same visual checks as above.

Figure 1: Resized images to 256x256 - smallest in TrainVal (LHS) and Largest (RHS)

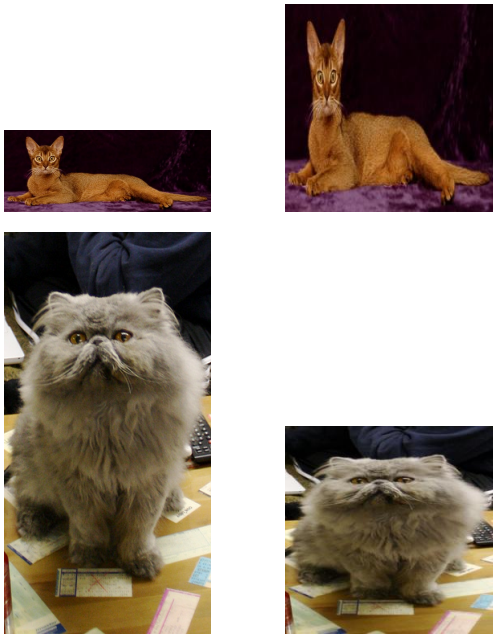

Figure 2: Original image (LHS) and resized image 256x256 (RHS). Top row: Highest aspect ratio image in TrainVal. Bottom row: Lowest aspect ratio image .

2.3 Augmentation

In order to increase the dataset size we augmented the initial 3,680 images/mask pairs in the TrainVal dataset with the same number again. Resulting in a total of 7,360 image/mask pairs that proved to provide an effective and efficient size for training our models as demonstrated in Section 4.

In order to do this we used the Torchvision V2 transformation functions. Transformations and parameters were selected that altered the image but within the bounds that the new images retained realistic representations of cats and dogs. We chose not to apply all of the transformation to every image but rather set probabilities to ensure a significant variety in the transformations applied for the new data. The following transformations were applied to both images and masks, and nearest neighbour interpolations was used unless stated otherwise (see Appendix for examples):

- Color jitter: Color jitter randomly alters the contrast, saturation, hue and brightness of the image. The factor parameter for all four components was set to 0.5 after a visual inspection of the impact of different values. The whole transformation was applied to 50% of the images. We wanted to retain the valuable information contained in the original colours in half of the augmented data. As color jitter, does not shift the image in any way, the original mask remains valid with no transformation required.
- Rotation: Rotation randomly rotates the image. The maximum rotation was set to 45 degrees as we believed it most likely that the vast majority of animals would be the correct way up in the image (confirmed this is true of TrainVal by visual inspection).
- Horizontal flip: This transformation flips the image horizontally with the set probability (50% in our augmentation process). We did not apply vertical flip as we concluded based on inspection of the TrainVal images that the vast majority of cat and dog images would be the correct way up.
- Elastic transformation: The elastic transformation applies a vector displacement to all pixels in the image. After visually testing with various parameters we set the strength and smoothness parameters at their default values of 50 and 5 respectively.

We considered other transformations such as cropping and additional colour changes but based on inspecting the outputs, we were satisfied that there was sufficient variety in the new dataset from those listed above.

2.4 Data processing for the prompt model

The data for the prompt model underwent the same standard augment + resize pre processing steps. We then created a mask with a red dot of radius 5 pixels, and applied a 9×9 Gaussian filter on the dot. Finally, we used alpha blending with parameter .75 to mix the red dot and image together. As all images have a single animal, finding the barycentre of an image is quite simple. Let $\Omega \subset \mathbb{N}^2$ be an image. The centre of mass or barycentre of an image is a pixel location $\mathbf{b} \in \mathbb{R}^2$ defined as follows:

$$\mathbf{b} = \frac{\sum_{\mathbf{x}\in\Omega} \mathbf{x} \cdot \mathbb{I}(\text{mask}(\mathbf{x}) = \text{cat} \vee \text{mask}(\mathbf{x}) = \text{dog})}{\sum_{\mathbf{x}\in\Omega} \mathbb{I}(\text{mask}(\mathbf{x}) = \text{cat} \vee \text{mask}(\mathbf{x}) = \text{dog})}. \quad (1)$$

We used `cv2`'s moments function (`cv2.moments`) to calculate the moments of black and white masks. If the shape in question is convex, then the barycentre is always in the shape. As most animals are not convex we filtered out the pictures where the barycentre was not on the animal. We also filtered out all the images with an all black mask. 14 images had an all background mask and 405 had a barycentre outside of the animal, discarding (i.e. removed from the training set) a total of 419 training images.

2.5 Post Preprocessing statistics

The pixel counts for both labels are shown below in table 1. Note that as stated before background pixels include border pixels.

Dataset	Dog Pixels ($\times 10^7$)	Cat Pixels ($\times 10^7$)	Background Pixels ($\times 10^8$)
Regular (256×256)	9.33 (19.4%)	4.50 (10.3%)	3.40 (70.4%)
CLIP (224×224)	7.12 (19.3%)	3.77 (10.2%)	2.60 (70.5%)

Table 1: Pixel distribution across different datasets after preprocessing.

3 Model design and implementation

The specific architecture for each model is detailed below. However the following were consistent for all the models :

- A training:validation split of 90:10 with the same random seed to ensure fair comparisons on performance.

- The batch size was set at 16. It has been shown that larger batch size decreases model performance [7]. Increasing the batch size allows more efficient GPU computation so a compromise of 16 was used.

- We used an Adam optimiser inline with common practice [20] in convolutional neural networks. The learning rate was set to $5 \cdot 10^{-4}$ for all models with a batch size of 16 and $1 \cdot 10^{-5}$ for models with a batch size of 1. It makes sense that a smaller batch size means a smaller step size as less importance should be attributed to a single data point due to the fact that it varies in some way from the average of all data points, and from the fact that smaller batch size means more iterations per epoch. The betas were 0.9 and 0.999 respectively. As Andrej Karpathy states in his blog: "adam is safe"[6].

- We also use the standard Softmax function [20] to turn the final activation layers into class likelihoods for classification. For an image $\Omega \subset \mathbb{N}^2$ pixel wise $\mathbf{x} \in \Omega$ softmax is defined as follows for $a_k(\mathbf{x})$ the output value of a NN for channel value $k \in \{0, 1, 2\}$ (the three possible classes) : $\text{Softmax}(a_k(\mathbf{x})) = \frac{\exp(a_k(\mathbf{x}))}{\sum_{j=0}^{2} a_j(\mathbf{x}))}$. Values pre softmax $a_k(\mathbf{x})\,,\ k \in \{0, 1, 2\}$ are called logits. Passing them through a softmax function ensures that $\forall \mathbf{x} \in \Omega\,, \quad \sum_{k \in K} \text{Softmax}(a_k(\mathbf{x})) = 1$, i.e. at each pixel in our output mask we have a valid probability density function that corresponds to classification probability.

- We use weighted Cross Entropy loss for pixel wise classification. Let $\mathbf{x} \in \Omega$ be a pixel location in our image, $a_k(\mathbf{x})\,, \forall k \in \{0, 1, 2\}$ the obtained logits at that position, $y_k = \mathbb{I}(\text{pixel } \mathbf{x} \text{ is of class } k)\,, \quad \forall k \in \{0, 1, 2\}$, and w_k the weight associated to channel k. We have the per pixel loss as

$$l(\mathbf{x}, \mathbf{y}) = \sum_{k \in \{0,1,2\}} -w_k \log\left[\text{Softmax}(a_k(\mathbf{x}))\right] \cdot y_k\,. \quad (2)$$

This leads to loss over an image as

$$L_\Omega = \sum_{\mathbf{x} \in \Omega} l(\mathbf{x}, \mathbf{y}). \quad (3)$$

And loss over a batch of size N as

$$L_{\Omega_1, \ldots, \Omega_N} = \frac{\sum_{n=1}^{N} \sum_{\mathbf{x} \in \Omega_n} w_{y_k(\mathbf{x})}\, l(\mathbf{x}, \mathbf{y})}{\sum_{n=1}^{N} \sum_{\mathbf{x} \in \Omega_n} w_{y_k(\mathbf{x})}} \quad (4)$$

where $w_{y_k(\mathbf{x})}$ is the weight for pixel $\mathbf{x}$ depending on its class k. As observed in section 2.5 the distribution per pixel class is far from uniform. Hence, we alter the importance of classification error for each class. For each class we scale its weight by 1/(%class presence in all images). In other words, classes with the least number of pixels have the highest coefficients, and common classes (i.e. background) have less of an impact on total loss. If this was not accounted for, the optimizer would hugely decrease the loss by classifying all pixels as background which would impact model performance.

- Note that the MSE loss is used for the "pretrained" encoder of the Autoencoder. The pretrained encoder must learn lower dimensional embedding of the input images. This is achieved by making the Autoencoder learn how to reconstruct an image from a condensed version of it. Given all images in our dataset $\Omega_i \in \mathbb{R}^{256 \times 256 \times 3}$ Our loss is as such: $\sum_{\Omega_i} ||\Omega_i - \hat{\Omega}_i||_2^2$. Here the Autoencoder does not output discrete classes, it outputs continuous pixel values along the 3 channels R,G, and B. It is therefore a **regression** problem. Altough MSE has its shortcomings (sensitive to outliers), it is simple to understand and it works well.

- Channel wise batch normalisation to deal with regularization + exploding gradient problem.

- We kept the order of magnitude of the number of trainable parameters in each model the same. CLIP has $\sim 3.2 \cdot 10^6$ parameters, U-Net 5.5 $\sim 10^6$ parameters, and the autoencoder $\sim 9.6 \cdot 10^6$ parameters. But this is not a fair comparison as the CLIP encoder has $\sim 3.0 \cdot 10^8$ pre-trained parameters which is around 100 times more than the other models. The challenge with CLIP is (a) to use the encoder's outputs in a way which allows the extraction of the maximum amount of useful information (b) interpret the outputs / embeddings in a way that is in line with the encoder architecture.

3.1 U-Net

Our U-Net architecture is shown in figure 3. It comprised of four convolutional contracting blocks before the bottleneck and then four corresponding transpose convolutional blocks. The features double within each contracting block (apart from the scaling of 3 (RGB) to 48 in the first block) and the spatial dimensions half after each block through max pooling. After each ReLU we applied channel-wise Batch Normalization to avoid parameter blow up. We used ReLU as it is computational efficient and was shown to be effective in the original U-Net architecture [17].

In the expanding path, the input into each block is the transposed convolution from the previous block concatenated with the output from the corresponding block in the contracting path (the skip connection). The blocks in the expansion path half the number of features and double the spatial dimensions (apart from the final block which returned three channels related to the segmentation classes). To obtain the final pixel class Softmax + Argmax must be applied pixel wise to the image. This is not directly done in the architecture as Pytorch's Cross Entropy loss applies these directly to the logits.

The four levels were chosen to ensure we had sufficient depth to effectively train a model whilst remaining efficient for our levels of compute.

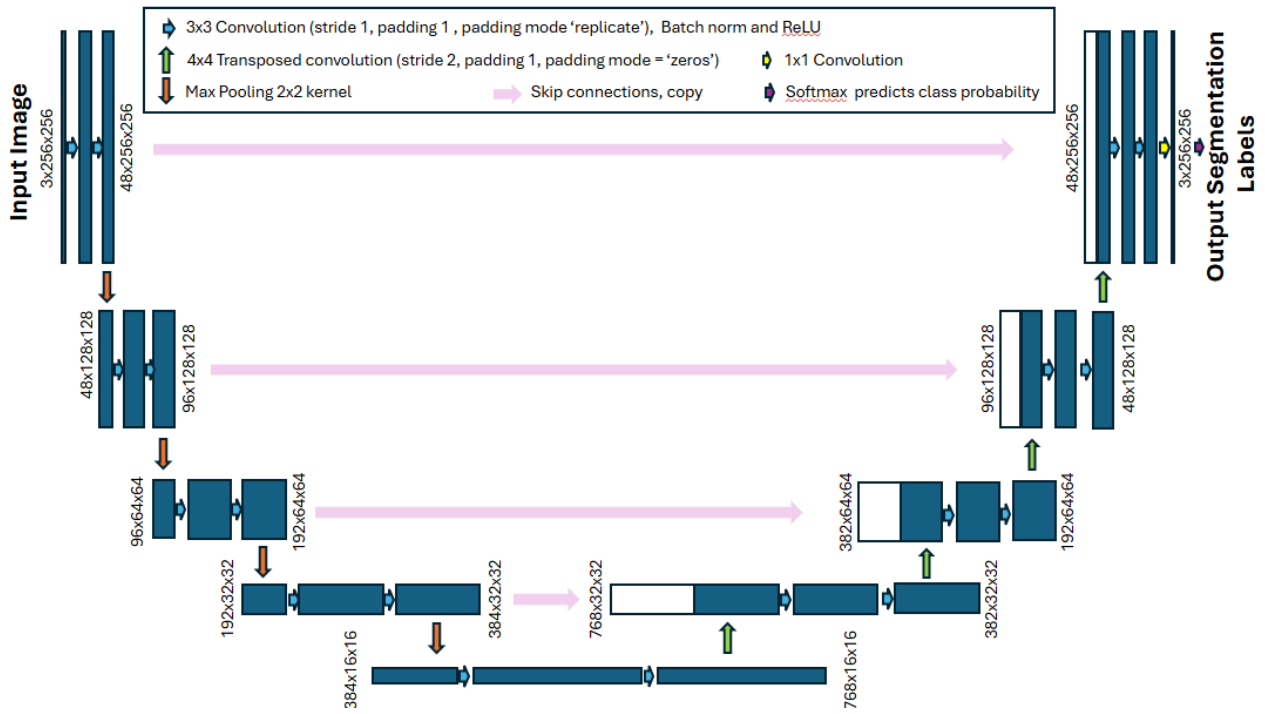

Figure 3: U-Net architecture for our model. Diagram based on Ronneberger *et al.*[17]

3.2 Autoencoder

We use a convolutional autoencoder as it effective in handling spatial data and efficient compared to other options (*e.g.* fully connected architecture). The architecture is shown in Figure 4. As per U-Net the model used four levels to balance the effectiveness of the model with the efficiency required given our compute. The encoder comprised of convolutional blocks followed by LeakyReLU and then max pooling to double the features and half the spatial dimensions each time (apart from the initial block which projects the 3 RGB input channels to 96 feature layers). The final step of the encoder flattens the feature map in to a vector and then applies a linear transformation to compress it from 49,152 to 384 features in the bottleneck. The decoder mirrors the encoder but in reverse with the upblocks comprising transpose convolutions with a stride of 2 and LeakyReLU that half the features and double the spatial dimensions. In the final upblock the features are reduced from 96 back to the 3 RGB channels to recreate the image (or the 3 class probabilities when used for mask prediction).

3.3 CLIP

We use the clip-vit-base-patch32 version of OpenAi's CLIP model [12] as it has been shown to be effective and efficient in computer vision tasks [2] . We used GitHub [10] [5] for code reference. To train our CLIP based model, each of the images in our training dataset is passed through the encoder of the pre-trained CLIP Visual Transformer model with weights frozen. The corresponding text for the image ' an image of a dog' or 'an image of a cat' is passed through the pre-trained CLIP Text Transformer with model weights frozen. The resulting tensors are combined by element wise multiplication in the shared embedding space. The decoder is made up of four blocks that take an input of the previous decoder block (if there is one) and activations from a hidden layer in the image encoder respectively (akin to skip connections). The encoder hidden layers used in our architecture are 8, 9, 10 and 11. In the decoder block, the number of features in the combined activations are reduced to 128 using a linear function and then combined text and image embeddings are used to apply an affine transformation (Feature wise linear transformation (FiLM)). This provides additional context to the model.

3.4 Prompt-based segmentation

Based on the results in table 2, we used our CLIP architecture for the prompt-based model. We retrained the decoder using the same training images but with a Gaussian-blurred red dot at the centre of the cat or dog (see Prompt Preprocessing in section 2). For testing we also added red dots at the centre points of the images (calculated in the same way) to replicate the section of a point by a user.

3.5 Approach to robustness exploration

To assess the robustness of our segmentation models, we implemented a set of common image perturbations that simulate real-world noise and distortions. Each perturbation modifies the input image before passing it through the model, allowing us to evaluate how performance degrades (or improves) under challenging conditions. The core implementation of each perturbation is listed below.

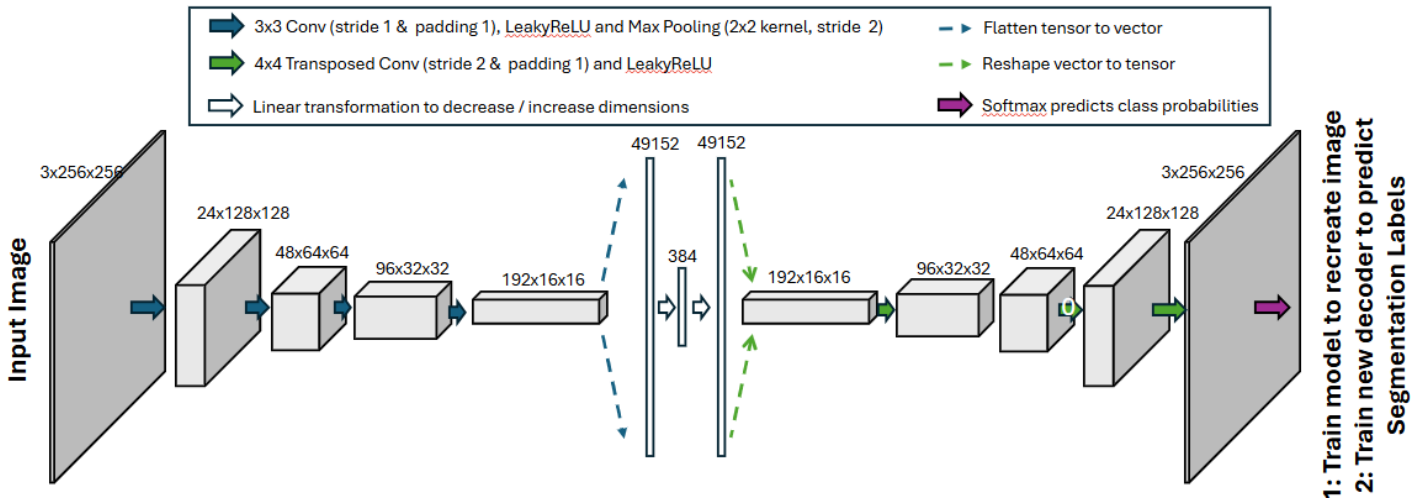

Figure 4: Autoencoder architecture used in our model

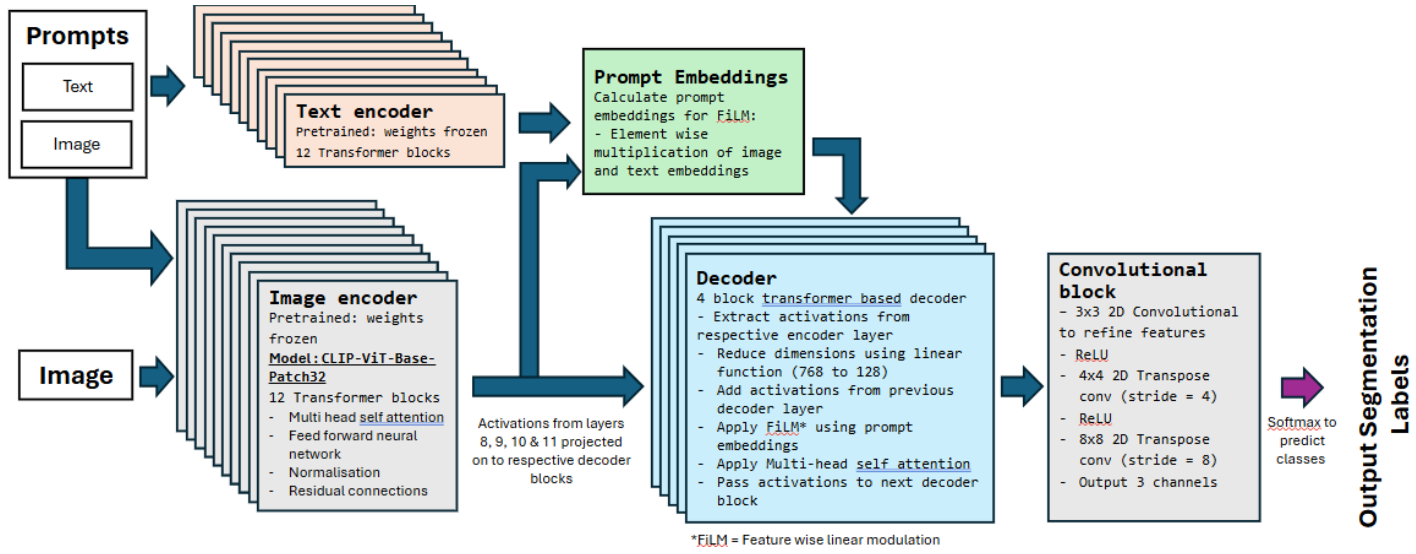

Figure 5: Clip Segmentation architecture used in our model

- **Gaussian Noise** simulates sensor noise by adding normally distributed noise to each pixel: `perturb_image = img + np.random.normal(0, sigma, size=img.shape)`
- **Gaussian Blur** simulates defocus or motion blur by convolving the image with a Gaussian kernel: `perturb_image[..., chn] = convolve2d(input_img[..., chn], blur_filter, ''same'')`
- **Salt and Pepper Noise** simulates dead pixels or data corruption by randomly setting pixels to black or white: `perturb_image = 255*random_noise(input_img, mode='s&p', amount=amount)`
- **Contrast Change** simulates exposure differences by scaling pixel intensities:`perturb_image = (input_img.astype(np.int16)*factor).clip(0, 255)`
- **Brightness Change** simulates lighting variation by adding a constant to all pixels: `perturb_image = (input_img.astype(np.int16) + factor).clip(0, 255)`
- **Occlusion** simulates partial object obstruction by zeroing out a square region at a random location: `perturb_image[x:x+w, y:y+h] = 0`

3.6 User Interface (UI) design

The UI allows the user to select an image and draw a red dot at a prompt point for mask prediction. The new image (with dot) is resized to 352x352 (using bilinear interpolation) and run through our CLIP model. The predicted mask is then displayed as an overlay of the image for visual analysis.

3.7 Performance metrics

To compare the performance of the models we use three key metrics. Note that in the following $A, B \in \{0,1\}^{w \times h}$.

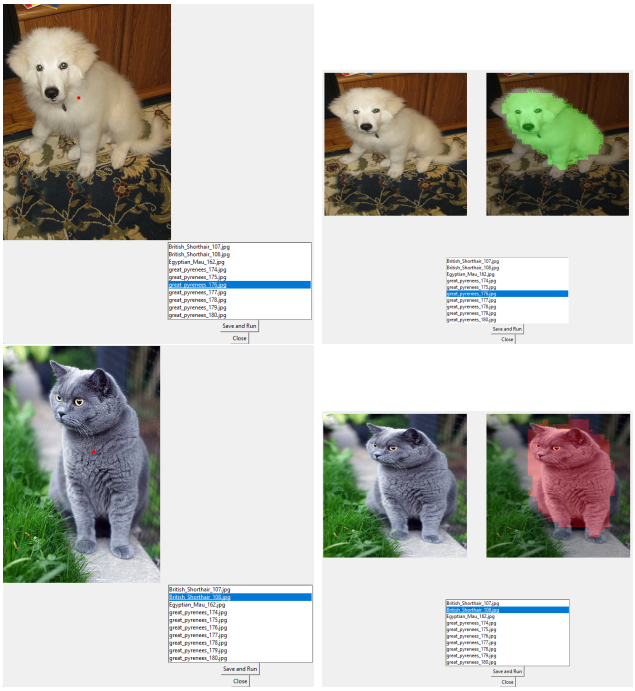

Figure 6: UI screenshots: selecting prompt point (LHS) and returning mask prediction (RHS).

- Intersection over Union (Jaccard Index): calculated as the count of pixels where the predicted and ground truth masks overlap (intersection) divided by the total number of unique pixels covered by the two masks (union).

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\text{sum}(A \wedge B)}{\text{sum}(A \vee B)} \tag{5}$$

  As masks have three output channels in one-hot encoding then $IoU = \frac{1}{3}(J(y_1,\hat{y}_1) + J(y_2,\hat{y}_2) + J(y_3,\hat{y}_3))$ or the average of each channel's Jaccard scores. $0 \leq IoU \leq 1$. IoU is maximal when $IoU = 1 \iff |A \cap B| = |A \cup B|$ meaning the two images are identical. When $IoU = 0 \iff |A \cap B| = 0$, which occurs when no pixels in the masks are 1 at the same locations.

- Dice Coefficient (F1 Score): calculated by doubling the number of pixels where the predicted and ground truth masks overlap divided by the sum of the number of non-zero pixels in the ground truth mask and the number of non-zero pixels in the predicted mask.

$$D(A,B) = \frac{2|A \cap B|}{|A| + |B|} = \frac{2 \cdot \text{sum}(A \wedge B)}{\text{sum}(A) + \text{sum}(B)} \tag{6}$$

  . Again, dice score for a 3 channel image is defined as the average dice score over the 3 channels: $Dice = \frac{1}{3}(D(y_1,\hat{y}_1) + D(y_2,\hat{y}_2) + D(y_3,\hat{y}_3))$. Via a bit of algebra we find that $J = \frac{D}{2-D}$ where J is the Jaccard index, and D the Dice score. Using the fact that $\frac{D}{2-D} \leq D\,,\ \forall D \in [0,1]$ we obtain $J \leq D$. We therefore have that the Dice index is always greater than or equal to the IoU score.

- Pixel Accuracy: calculated as the correctly predicted pixels (True Positives and True Negatives) divided by the total number of pixels. It provides an intuitive way to understand how well the model is predicting the right outcome but it is highly sensitive to when the classes are imbalanced. Notably for this project, we need to use it with care due to the the presence of a much higher percentage of background pixels.

4 Performance of the models

4.1 Performance

We look at the three base models performance metrics in two cases. In the first case we compare NN output with downsized mask. While in the second case, we up-sample masks to the dimension of original image. This is to observe how negatively up-sampling affects results. Note that output masks have 3 discrete possible values: `Red=(128,0,0)`, `Green=(0,128,0)`, `Black=(0,0,0)`. This implies that when up-sampling the outputted NN masks we cannot use bilinear/cubic interpolation, but we must use nearest neighbors interpolation: `torchvision.transforms.InterpolationMode.NEAREST`. This will lead to a more rugged/pixilated output masks which in turn we hypothesize will decrease performance.

Model	Accuracy	IoU score	Dice score
U-Net	85.9%	47.6%	53.8%
Autoencoder	70.8%	35.7%	43.9%
CLIP	92.8%	55.2%	60.0%
Prompt	91.2%	53.3%	58.8%

Table 2: Performance on test set on downsized masks. For CLIP the masks are of dimensions 224×224 and other models 256×256

Model	Accuracy	IoU score	Dice score
U-Net	85.9%	47.6%	53.9%
Autoencoder	70.8%	35.7%	43.9%
CLIP	92.9%	55.3%	60.0%
Prompt	91.3%	53.2%	58.9%

Table 3: Performance on test set on original sized masks. All NN output masks are up-scaled back to dimensions of corresponding original mask. Nearest Neighbours was used to make sure no continuous values are introduced.

Also, note that the test set has a few incorrectly sampled images. Either the mask is only background, or at times the mask is inverted (i.e. animal is labelled as background, and background is labelled as an animal)

4.1.1 U-Net

The U-Net training results are shown in figure 7. Looking at the loss we can see that it stagnates at around 0.175. As it does not reach close to 0 we conclude that a more complex U-Net model with more parameters would be needed

to reach a lower loss, and, therefore probably higher performance metrics. U-Net is the model with the biggest difference in training / test set and evaluation / test set IoU score. We rule out the possibility that hyperparameters were tuned to the validation set (as we did not do that). In the last 10 epochs we start to see a difference of about 10% between train and validation IoU. Train IoU is at around 80%, validation 70% and test 54%. This is a sign of overfitting, and more steps towards regularization could have been taken to counter this. We use batch norm, but could have replaced that by dropout layers that have been shown to be good regularizers [19]. Another option could be to undergo an early stop at around epoch 15 for training, but it makes more sense to train a bigger model with more generalisation. Another possible explanation to this big difference is the fact that training and validation sets contained augmented data. This leads to a drift in the distribution of both datasets compared to the test set. It is therefore possible that data augmentation hurts performance. More specifically, we believe that color jitter is the biggest culprit. It is probable that the NNs differentiate between cats and dogs by animal texture/color. Some of our augmented images contain unrealistic blue and purple cats. Another interesting observation is the bumpy accuracy and IoU validation score. It is initially pretty big in the first 10 iterations, but does level off.

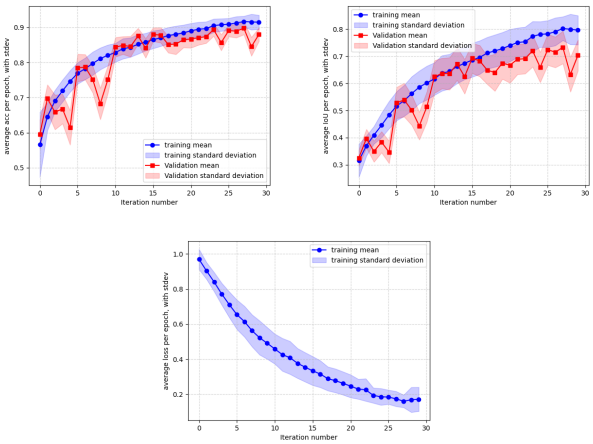

Figure 7: U-Net validation metrics over epochs: Accuracy (left), IoU (right), and Loss (below).

4.1.2 Autoencoder

Figure 8 displays how the MSE loss of the reconstruction autoencoder levels off very rapidly, to the point where it resembles an exponential decay. It is for that reason that we stopped training the reconstruction autoencoder after 10 epochs. The autoencoder was the fastest model to train per epoch, allowing us to run it for approximately 50 epochs. This efficiency likely stems from its high parameter count, which enables rapid convergence during training. However, due to time constraints, we did not compute validation metrics after each epoch, as doing so would have significantly increased computation time. Despite strong training performance—with a final IoU of around 80%—the test set IoU dropped drastically to 40%, indicating substantial overfitting. This suggests that the model could benefit from stronger regularization techniques or extended training with intermediate validation checks to improve generalization.

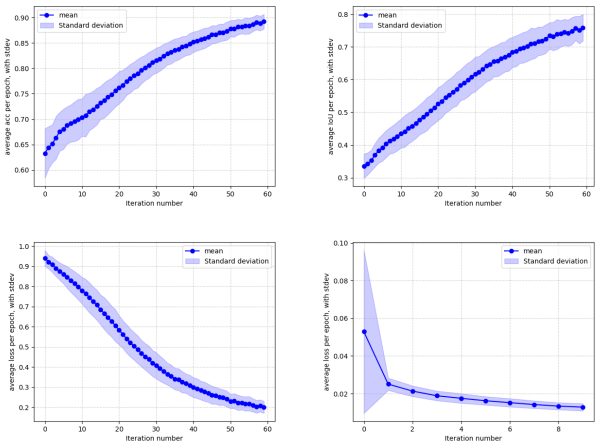

Figure 8: Autoencoder validation metrics over epochs: Accuracy (left), IoU (right), and below CE Loss (left) for segmentation and MSE Loss (right) for reconstruction).

4.1.3 CLIP and Prompt Model

CLIP results are shown in figure 9 Over the course of just five training epochs, we observe the evolution of cross-entropy loss, training and validation accuracy, as well as training and validation IoU. A striking observation is that performance plateaus after only a single epoch. This rapid convergence can be attributed to the powerful pre-trained CLIP encoder used, coupled with a lightweight decoder. Interestingly, the cross-entropy loss does not reach zero but instead stabilizes between 0.150 and 0.200. This suggests that the model is confident in its predictions without being overly confident or overfitting to noise. Similarly to U-Net, this indicates that a bigger more complex model (decoder in this case) could be tested to decrease the loss. Furthermore, the validation metrics remain consistently close to their training counterparts, indicating strong generalization. In fact, validation accuracy and IoU slightly exceed training metrics across all epochs because validation is evaluated after training on the complete dataset for that epoch.

Given that the Prompt Model is based off of the same CLIP architecture it is unsurprising to see similar results to CLIP. In fact, performance is marginally worse as a result of the point prompt being added to the image. It is possible that the pretrained CLIP encoder is so effective given its enormous training set that the red dot marginally impacts its effectiveness. However, we propose further work to examine whether randomising the location of the dot helps performance or adding it as a 4th channel. Another option would be to combine the image and prompt point features in the embedding space before input in to the decoder rather than on the input image.

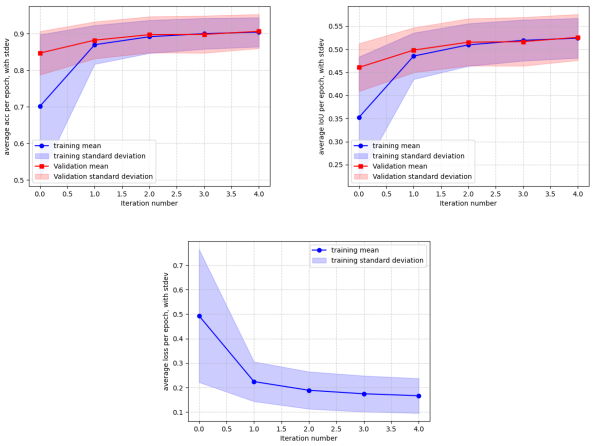

Figure 9: CLIP validation metrics over epochs: Accuracy (left), IoU (right), and Loss (below).

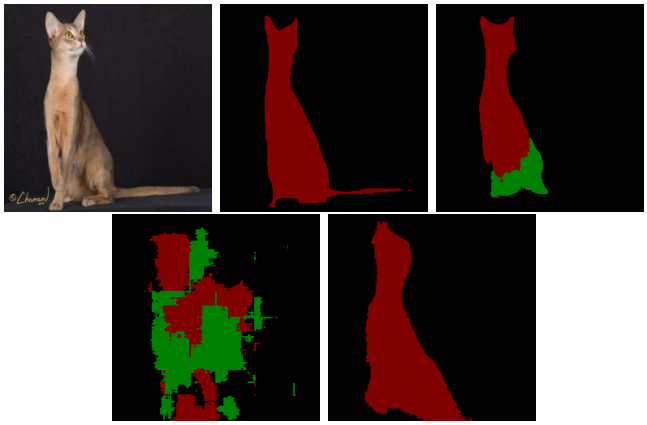

Figure 10: Difference in prediction. First two images in 1st row are the data image and ground truth label respectfully. After that we have U-Net prediction followed by autoencoder and then CLIP.

4.2 Model outputs

We thought it would be insightful to examine the model outputs that have low IoU to understand how and where the model goes wrong.

We decided to start by taking a random example as shown in figure 10. The image and mask in question are those for `Abyssinian_2.jpg` in the `Test` directory.

Starting with U-Net, we have observed that generally U-Net is very good at finding a clear and smooth boundary for cats and dogs. As we can see, the boundary is smoother than the ground truth mask itself as in the latter border pixels were replaced by background pixels. One of the issues with our U-Net model is that it confuses some pixels as being dog (green) rather than cat (red). This confusion seems to be somewhat smooth and continuous. More specifically, parts that are classified as dog are clustered together with a relatively smooth boundary.

As observed in table 2 the autoencoder has the worst results. In this example we see multiple issues: the background/animal boundary is not smooth and accurate, the model can't really seem to differentiate between cat/dog, there are seemingly random floating pixel conglomerates that are classified as an animal but are not that at all.

Looking at CLIP's output, its winning advantage is that it distinguishes perfectly between cat and dog. In all masks this is the case! The main issue with CLIP that limits its performance is the inability to reconstruct clear boundaries. There are two main possibilities explaining that. The first one is that the decoder is not complex enough. The decoder has a total of $\sim 3.2 \cdot 10^6$ parameters where $\sim 2.4 \cdot 10^6$ of these parameters are in the transformers and $3 \cdot 10^5$ are in the convolutional up-sampler. It is possible that a more complex up-sampler leads to a more accurate boundary. The second possibility is that the embedding simply does not contain the spatial / boundary information to allow the decoder to recreate the border of an animal, making it guess an approximate border. In that case, increasing the complexity of the up-scaler would change the **preciseness** of the boundary but not the **accuracy**.

4.2.1 U-Net

The example in figure 11 is really interesting although highly inaccurate. A striking observation is that the NN has identified the fluffy and hairy pillow as an animal. This is probably due to its shape, texture, and color. Second, the NN has segmented background between the cat and pillow to seperate the animal entities even though in the image there is no background between them. Third, it has misclassified the cat as a dog, and the pillow as a cat. We would actually argue that the pillow looks more like a dog than a cat due to its hair. Finally, another observation reinforces the fact that U-Net is the best NN at delimiting animals (compared to our other two NNs).

Figure 11: Image (left), Ground Truth (center), U-Net predicion (right)

4.2.2 Autoencoder

The Autoencoder segmented relatively well. We can observe some of its artifacts in figure 12. First, the boundary is pretty accurate but lacks smoothness. Second, there are little holes inside the image that do not seem to occur from U-Net or CLIP predictions. We believe that these might both be a consequence of the fully connected layers at the bottleneck of the encoder. Although fully connected layers conserve globality, this comes at the cost of a generalised locality parameters. Both other networks, probably learned that when there is no sharp boundary around a pixel and that all surrounding pixels are of one type it is very probably that the pixel we are trying to classify is of the same type.

Figure 12: Image (left), Ground truth (center), Autoencoder prediction (right)

4.2.3 CLIP and Prompt model

Figure 13: Image (left), Ground truth (center), CLIP prediction (right)

In figure 13 we look at the archetypal output of our CLIP model. As previously stated, on one hand, it does not make a single error pixel wise between cats and dogs. On the other hand, it has difficulty finding a clear boundary between different class pixels. Unsurprisingly the Prompt model behaves similarly, being effective at predicting cat or dog but struggling with boundaries (see 6.

5 Robustness exploration

As the graphs show in figure 14 the perturbations do not shift the test Dice score of 60% 2 by more than 0.1%. We can therefore conclude that the model is robust to image perturbations. This is most likely due to the nature of the encoder that has been trained on an immense dataset

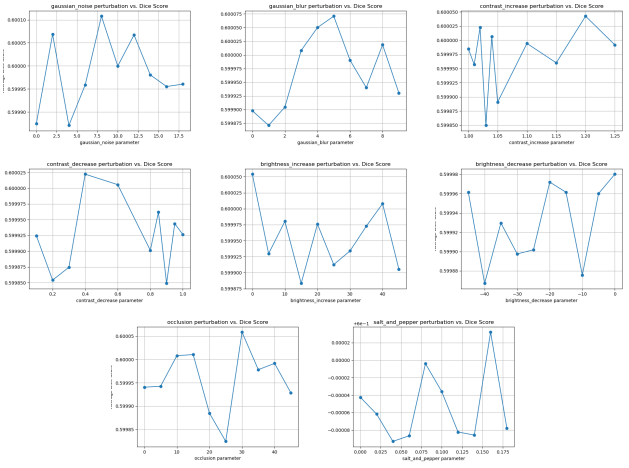

Figure 14: Resulting Dice Score of ClIPSeg model with perturbations.

Figure 15: Perturbed Images. (1) Gaussian noise $\sigma = 18$, (2) Gaussian blur filter passed 9 times (3) Contrast increase by factor of 1.25 (4) Contrast decrease by factor 0.1 (5) Brightness increase by intensity value 45 (6) Brightness decrease by intensity value 45 (7) Occlusion of a box of dimension 40 pixels (8) Salt and pepper noise with value 0.18 (9) Ground truth mask (10) Original image

6 Conclusion and future work

This project provided a comparison of segmentation models, highlighting the strengths and limitations of U-Net, autoencoders, and CLIP-based approaches. While are satisfied with the current results, there are several directions for improvement.

From a training perspective, future experiments could benefit from less aggressive data augmentation strategies, better class balance between cats and dogs (both in number of images and foreground pixel distribution), and the removal of image-mask pairs containing only background pixels, which may otherwise distort learning signals.

For the autoencoder model, introducing a convolutional bottleneck rather than fully connected might help preserve some spatial information. Also applying regularization techniques could help reduce overfitting and improve generalization. In the case of U-Net, increasing model capacity and adding dropout or weight decay may further enhance its performance, particularly in challenging test conditions.

With CLIP, future work could focus on increasing decoder complexity to better match the granularity of U-Net's decoder, potentially leading to improved boundary precision. Additionally, a deeper investigation into skip connections is warranted. The CLIPSeg paper recommends using layers 8 through 11 of the visual transformer, yet the representational content of earlier layers remains less explored. Understanding the semantic and spatial characteristics of different CLIP layers may provide valuable insights into how best to fuse them for dense prediction tasks.

Overall, these avenues point toward a more robust, accurate, and interpretable segmentation pipeline, combining the strengths of pretrained transformers with task-specific decoding strategies.

References

[1] Muhammad Awais et al. *Foundational Models Defining a New Era in Vision: A Survey and Outlook*. 2023. arXiv: 2307.13721 [cs.CV]. URL: https://arxiv.org/abs/2307.13721.

[2] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV]. URL: https://arxiv.org/abs/2010.11929.

[3] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[4] IBM. *What is an autoencoder?* 2025. URL: https://www.ibm.com/think/topics/autoencoder.

[5] Anuj K. *My-ClipSeg-using-CLIP*. 2023. URL: https://github.com/Anuj-N/My-ClipSeg-using-CLIP/blob/master/README.md.

[6] Andrej Karpathy. *Andrej Karpathy blog: A Recipe for Training Neural Networks*. Accessed: 01.04.2025. 2019. URL: https://karpathy.github.io/2019/04/25/recipe/.

[7] Nitish Shirish Keskar et al. *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*. 2017. URL: https://arxiv.org/abs/1609.04836.

[8] Alexander Kirillov et al. *Segment Anything*. 2023. arXiv: 2304.02643 [cs.CV]. URL: https://arxiv.org/abs/2304.02643.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E.Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems* 25 (2012), pp. 1106–1114. URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[10] Timo Lüddecke. *clipseg*. 2022. URL: https://github.com/timojl/clipseg.

[11] Timo Lüddecke and Alexander S. Ecker. *Image Segmentation Using Text and Image Prompts*. 2022. arXiv: 2112.10003 [cs.CV]. URL: https://arxiv.org/abs/2112.10003.

[12] OpenAI. *Model Card: CLIP*. Jan. 2021. URL: https://huggingface.co/openai/clip-vit-base-patch32.

[13] Omkar M. Parkhi et al. *The Oxford-IIIT Pet Dataset*. 2025. URL: https://www.robots.ox.ac.uk/~vgg/data/pets/.

[14] Ethan Perez et al. *FiLM: Visual Reasoning with a General Conditioning Layer*. 2017. arXiv: 1709.07871 [cs.CV]. URL: https://arxiv.org/abs/1709.07871.

[15] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV]. URL: https://arxiv.org/abs/2103.00020.

[16] Nikhila Ravi et al. *SAM 2: Segment Anything in Images and Videos*. 2024. arXiv: 2408.00714 [cs.CV]. URL: https://arxiv.org/abs/2408.00714.

[17] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV]. URL: https://arxiv.org/abs/1505.04597.

[18] Laura Sevilla. *Lecture 1: Introduction to Computer Vision*. 2025.

[19] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[20] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Jan. 2022. ISBN: 978-3-030-34371-2. DOI: 10.1007/978-3-030-34372-9.

[21] Antonio Torralba, Phillip Isola, and William T. Freeman. *Foundations of Computer Vision*. MIT Press, 2022. ISBN: 9780262048972. DOI: 10.1007/978-3-030-34372-9.

A Code

See accompanying file: *S2751476 – S2756547_code.zip* .

B Contributions

We approached this projected collaboratively throughout, with joint planning sessions, regular discussions on progress and reviews of final work. We iterated on each others written and code contributions to improve the final outcomes. Hence it is difficult to assign specific sections to either individual. In terms of initiating work:

- S2751476 initiated the work on data discovery, data processing, CLIP, the prompt-based model and UI, and robustness inputs.
- S2756547 initiated the work on U-Net, autoencoders, running the models, evaluation metrics and performance comparisons including robustness outputs.

However, both individuals had significant input to the all sections. Overall we feel that we contributed equally in terms of time and, quantity and quality of contribution.

C Data Discovery

Charts showing the dimensions of the images in the TrainVal dataet and Test dataset are shown in figure

D Data Augmentation Examples

Examples of the output images from our augmentation process shown in figure 17.

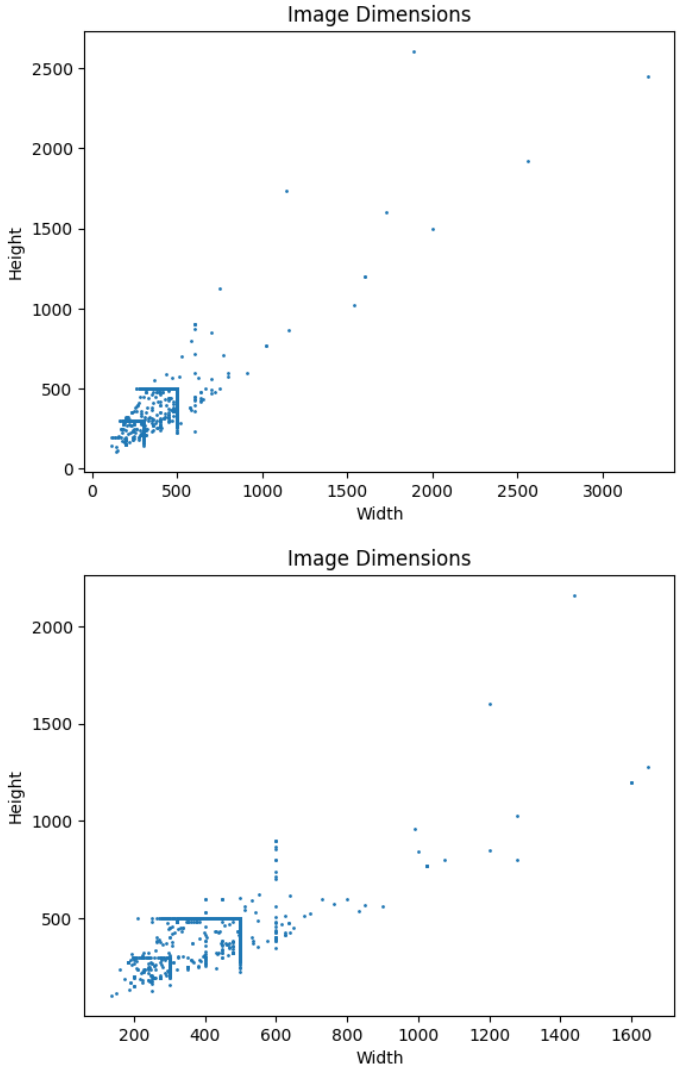

Figure 16: Image dimensions by pixels of the TrainVal dataset above and the Test dataset below

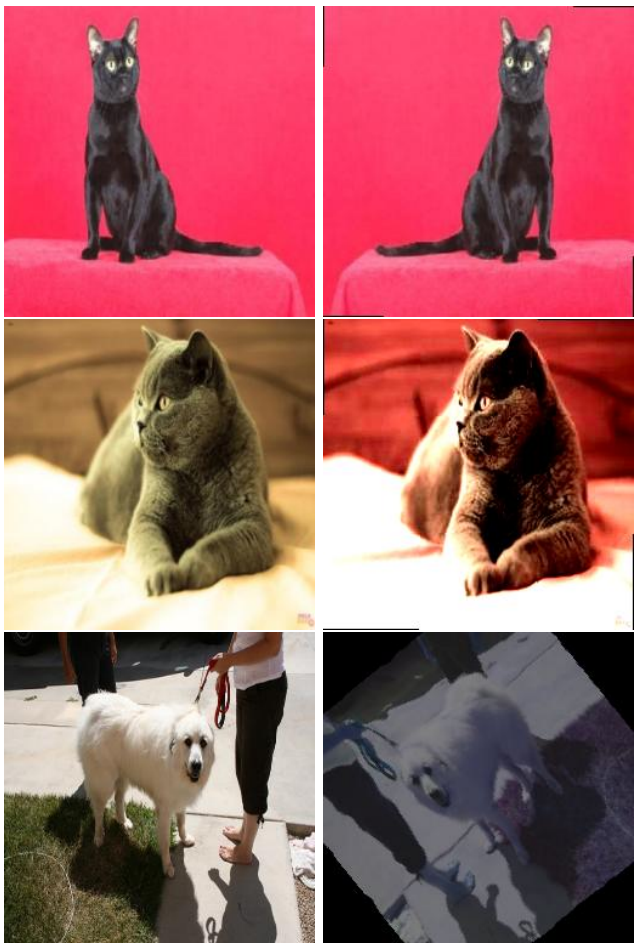

Figure 17: Examples of data augmentation with original image on the right and the augmented image on the left. Top row: reflection. Second row: color jitter. Third row: rotation and color jitter.