

# School of Informatics



INFR11212  
Computer Vision coursework  
Image segmentation in pets classes

s2677266 s2748897  
April 2025

**Professor:** Laura Sevilla-Lara

# 1 Introduction

**Image segmentation** is a fundamental task in Computer Vision (CV) that involves partitioning an image into semantically meaningful regions. This consists in assigning labels to individual pixels, distinguishing between entire visualised objects, or combining both objectives (i.e., semantic, instance, and panoptic segmentation, respectively). Specifically, semantic segmentation performs granular-level classification with a set of object categories for all the relevant image’s pixels [1], aiming to simplify image representation under a structured informational schema [2]. This process is foundational in many CV pipelines, enriching data for higher-level tasks like object recognition, scene reconstruction, and medical analysis [3].

Advancements in **Machine Learning** (ML) have greatly improved segmentation performance; being able to learn hierarchical representations directly from data with no handcrafted features defined, Deep Learning (DL) models in particular leverage large-scale datasets and computational power to achieve superior accuracy and generalization [4]. This predominance clearly arises from the ability of Convolutional Neural Networks (CNNs) and transformer-based methods to capture complex spatial dependencies and semantic relationships within images [5]. Additionally, a successful strategy for Neural Network (NN) classifiers in this field is the use of pre-trained feature-extraction architectures, which help to leverage representational knowledge from the relevant dataset, or even from vast collections of images depicting objects beyond task’s focus. By providing robust embeddings — eventually tailored by fine-tuning — the segmentation performance is thus bolstered even when the available labelled data is scarce, the necessary training time is reduced, and generalization enhanced [6].

A leading architecture is the **UNet**, a fully convolutional network originally designed for biomedical segmentation. It employs an encoder-decoder structure: the contracting path captures spatial hierarchies through convolution and pooling, while the expanding one upsamples predictions and merge features via skip connections, preserving detail for pixel-level classification [7].

Similarly, **AutoEncoder** (AE) architectures consist of an encoder, that compresses input into a latent space, and a decoder; however, in this case, the de-contracting path tries to reconstruct the original input directly, instead of predicting any sort of classification already. AEs are unsupervised learning framework for feature extraction objectives: by pre-training such an architecture to minimize the input-recreation loss over a specific dataset, the embeddings flowing through the imposed bottleneck can work as wise summaries in the desired dimensionality of the instances [8, 9]. Variational or Denoising AEs produce robust representations too, modelling latent spaces as distributions for data augmentation and anomaly detection, or learning clean reconstruction from deliberately corrupted input to increase robustness to perturbations [10, 11].

Contrastive Language-Image Pretraining (**CLIP**) learns joint visual-text representations from large-scale image-text pairs, aligning images with descriptions rather than relying solely on pixel-level labels [12]. CLIP-based features generalize well across datasets, enabling zero- or few-shot segmentation and improving boundary detection through semantically rich embeddings [13].

Working in parallel with other techniques, **prompt-based segmentation** introduces minimal user input (e.g., points, boxes or scribbles) to guide the model in identifying specific regions [14]. Improving efficiency and adaptability thanks to user-provided cues which dynamically refine the output, this approach is widely adopted in strategies like the Segment Anything Model [15].

## 1.1 Objective

The aim of the present paper is to experiment disparate image segmentation techniques on a properly pre-processed and augmented images dataset (Sec. 2), exploring both traditional and modern DL approaches to this task. Specifically, the importance of pre-training and feature-

extraction strategies is investigated, comparing the performance of a UNet-based end-to-end NN (Sec. 3.1.1) to the one of Autoencoder-based (Sec. 3.1.2) and CLIP-bolstered (Sec. 3.1.3) architectures. The latter well-performing model is also experimented in a prompt-based version (Sec. 3.1.4), enriching the input format through a systematic annotation of the images with an informative rationale (Sec. 2.3). Evaluating their test performance and robustness to input perturbations (Sec. 3.3), these methods’ outcomes are then comparatively discussed (Sec. 4).

## 2 Data

A processed version of The Oxford-IIIT Pet Dataset [16] is given, already split into training and test subsets: a total of 3680 instances are for learning purpose, while other 3710 stay unseen for subsequent evaluation. Of the former set, a 20% is kept for validation purposes, while the remaining part is directly used for models optimization. Each image depicts a dog or a cat — rarely both, but with one of them only annotated — and comes associated with a ground-truth masks, indicating the pixels pertaining to each category (Appx. A.1.2). In particular, 4 classes are differentiated with these labels: cat, dog, animal’s outline, and background. A total of 37 breeds appear, each with about 100 instances both in the *Train/Val* and *Test* set, making the latter collection fair for evaluation of models based on the corresponding training set (Appx. A.1.1). Nevertheless, a strong class imbalance is noticeable, as the number of cat breeds is just 12 with respect to 25 dog ones; given the almost equal images population for each breed uniform across the two sets, both have less than  $\frac{1}{3}$  of cat instances (Tab. 1).

### 2.1 Resizing

The images and respective masks have different sizes, but in the *Train/Val* set most of their dimensions are near to 500x500 pixels, with a distribution that highly suggests this as a sensible resizing shape (Fig. 1), entailing minimal information loss [17]. For this reason, all the instances — ground-truth labels included — are initially converted to the aforementioned dimensions, which is useful to operate the subsequent preprocessing steps onto standardized tensors.

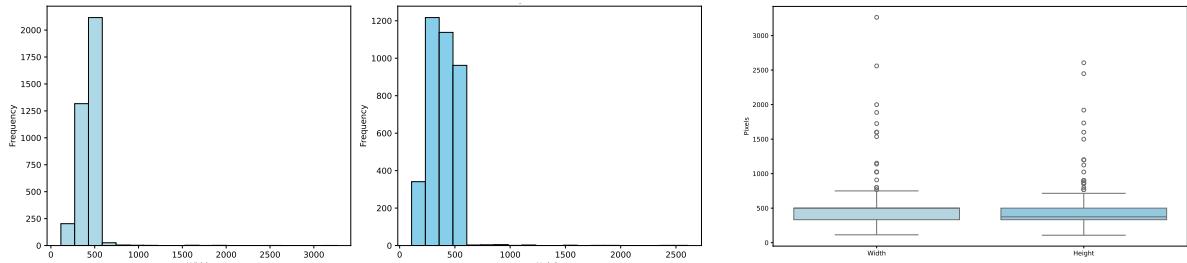


Figure 1: Images dimensions distribution in the training and validation set

### 2.2 Augmentation

Before performing conventional data augmentation, it is essential to address the issue of class imbalance: a significant asymmetry in the training set may cause the optimization process to prioritize the dominant classes, thereby neglecting the minimization of loss associated with predictions of the minority class. In this context, the considerably lower number of cat images in the *Train/Val* set is likely to result in inferior segmentation performance for cat pixels. This is because the model’s gradient descent process would be disproportionately influenced by the more prevalent dog images, leading it to better capture the patterns associated with dogs. To mitigate this issue, two primary strategies can be employed. The first involves assigning class-specific weights during loss computation (e.g., assigning a higher weight to cat pixels, potentially

	Train/Val	Test	Breeds
Cats	1188	1212	12
Dogs	2492	2498	25
Total	3680	3710	37

Table 1: Datasets’ classes counts and number of breeds per species

nearly double that of dog pixels) to compensate for the imbalance. The second approach is to rebalance the dataset by augmenting the minority class through derived images [18]. The latter method is adopted, to avoid the need for assigning weights to non-animal classes as well, which could inadvertently interfere with the model’s autonomous learning of those categories.

Moreover, due to cats’ natural behaviour, cat images frequently show the animal in particular postures and with its snout oriented at various angles relative to the camera and often in agile, upward-paws poses. This characteristic suggests that rotating all original training images by  $180^\circ$  could yield additional informative inputs that align with the natural variability of feline posture. Such an augmentation not only preserves the physiological plausibility of the visual representation — particularly for cats — but also contributes to learning direction-invariant anatomical features. While simple vertical flipping may suffice for augmenting dog images, this rotation-based augmentation may provide more substantial benefits for the model’s understanding of cat morphology. Therefore, the resized *Train/Val* images and their corresponding masks are included in the training set. In addition, cat-labeled instances are also added in their  $180^\circ$ -rotated versions. This preliminary augmentation results in a total of 2,376 cat images, effectively **resolving the class imbalance** without introducing any artificial noise.

Importantly, this was considered the only significant imbalance in the dataset. It is assumed that variations in breed within each species have a negligible impact on the overall performance, since the classification task operates at the species level, where physical traits remain consistently distinct and largely independent of breed. Although the segmentation task is based on pixel-level information, no further disequilibriums were identified: visual inspection suggests that, on average, cats and dogs occupy a similar amount of space within their respective images. Regarding other labels, the background class is expected to occupy a larger portion of pixels overall. However, this can be beneficial to Neural Networks, as the visual patterns in background regions are highly varied — spanning natural and indoor scenes depending on where each photo was taken, thus helping the model generalize better. In contrast, the *outline* class contains significantly fewer pixels than the others, but it is excluded from evaluation (Sec. 3.3) and retained in the target masks, being expected to aid the model to segment into well-defined regions. Finally, loss functions were used which account for per-image class imbalance (Sec. 3.1).

Following this setup, the dataset undergoes full **augmentation** procedures to further expand the number of training instances and enhance the model’s generalization capabilities. For each image, a new version is generated by probabilistically applying a series of transformations. The corresponding mask is augmented using geometric transformations only, ensuring that the label intensity and structure remain intact.

#### Geometric transformations on image and mask (application probability of 50%)

- Horizontal Flip: mirrors left-to-right
- Vertical Flip: mirrors top-to-bottom
- Random  $90^\circ$  Rotation: rotates by a random multiple of  $90^\circ$
- Affine Transformations: applies random scaling, translation, rotation, and shearing
- Elastic Transform: distorts using random elastic deformations
- Grid Distortion: applies a random grid deformation
- Optical Distortion: introduces random lens-like warping effects

#### Intensity transformations on image only (application probability of 30%)

- Gaussian Blur: applies a Gaussian blur with random kernel size between  $3 \times 3$  and  $7 \times 7$
- Motion Blur: simulates motion blur with random kernel size up to 5 pixels
- Random Brightness/Contrast: adjusts brightness and contrast randomly within  $\pm 20\%$
- Hue-Saturation-Value Shift: randomly modifies the hue, saturation, and brightness values
- Gaussian Noise: adds Gaussian noise with random variance in the range [10, 50]
- Multiplicative Noise: multiplies pixel values by a random factor in the range [0.9, 1.1]

### 2.3 Annotation

Finally, as a modification to the original database, for both the training and test images, one pixel pertaining to the centre of the depicted animal's nose is individuated and its coordinate over the 500x500 representation stored. After a trial with AI-based automatic annotation, producing poor-quality results, the above additional labelling procedure is carried manually, using a script to iteratively visualize one of the resized pictures and saving the position of a mouse click on it. For the really few cases with no nose displayed due to the animal's posture, the spatially nose-nearest visible point is indicated instead. The aim is preparing an additional informationally valid input to train a sensible prompt-enriched model (Sec. 3.1.4). In fact, the chosen pixel is deemed able to individuate the snout area, which is dense of species-characteristic visual features (e.g., muzzle, eyes and ears shape). Pointing this spot to the model would thus not only hint to the model the position of the animal, but also help recognizing felines from canids, through suggesting specifically relevant area [19]. To feed this information layer to the NNs, a one-dimensional heatmap is produced for each resized instance — to be conceived as the images' fourth channel — and eventually augmented geometrically like the respective mask in the *Train/Val* case. This greyscale 500x500 representation is built through a symmetric Gaussian kernel centred in the annotated pixel, represented by maximum brightness, which shades down to black with a sigma-parameter tuned to averagely allow the light area to extend over all the anatomies of interest (Fig. 2).

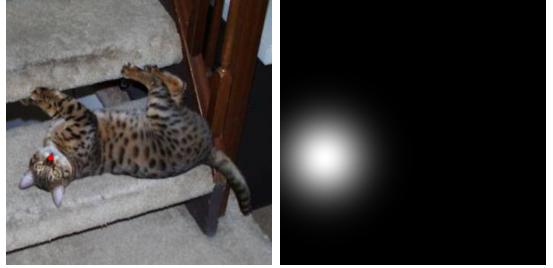


Figure 2: Annotated image and its heatmap

## 3 Experiments

With the presented background domain understanding (Sec. 1), this paper's specific segmentation task is addressed: the objective is to classify each pixel of the input colour (i.e., JPG-given in 3 channels) images into one of 4 labels, predicting the respective single-channel PNG-represented masks accurately, through efficient ML-models. The classes are (i) *Background*, (ii) *Cat*, (iii) *Dog*, and (iv) *Outline* of the animal. The original instances used for training are the augmented 500x500 ones, outcome of the described pre-processing (Sec. 2). Anyway, all the models use an input and output dimension of 256x256, due to computational limitations; nevertheless, the test procedures (Sec. 3.3) will be conducted resampling back to the original mask size.

### 3.1 Models Architectures and Training

For the segmentation task at issue, it is crucial to choose loss functions that not only maximize pixel accuracy but also handle class imbalances and overlapping regions effectively. To achieve this, the following loss types were experimented for NNs' gradient-descent optimization:

**Mean Square Error** MSE measures the average squared difference between predicted and target values:  $\mathcal{L}_{MSE} = \frac{1}{N} \sum_{i=1}^N (x_i - \hat{x}_i)^2$ , where  $x_i$  is the input pixel and  $\hat{x}_i$  the reconstructed pixel. It is used solely to pre-train the unsupervised AutoEncoder (Sec. 3.1.2) for learning compressed representations without labels [8].

**Cross-entropy Loss** Cross Entropy Loss focuses on pixel-wise classification by comparing predicted probabilities with ground truth labels, penalizing incorrect predictions. While effective for accuracy, it struggles with class imbalance, often biasing predictions toward the dominant background class. The loss is given by  $\mathcal{L}_{CE} = -\sum_{c=1}^C y_c \log(\hat{y}_c)$ , where  $C$  is the number of classes,  $y_c$  is the ground truth probability, and  $\hat{y}_c$  is the predicted probability for class  $c$ .

**Dice Loss** Dice Loss maximizes overlap between predicted and ground truth masks, making it robust against class imbalance. It prioritizes shape and boundary accuracy, ensuring well-defined object contours [20]. The loss is defined as  $\mathcal{L}_{Dice} = 1 - \frac{2\sum p_i g_i}{\sum p_i + \sum g_i}$ , where  $p_i$  and  $g_i$  are the predicted and ground truth values for pixel  $i$ , respectively.

**Combined Loss** Combining both losses balances pixel-wise accuracy with shape preservation:  $\mathcal{L}_{Combined} = \alpha \mathcal{L}_{CE} + \beta \mathcal{L}_{Dice}$ , where  $\alpha$  and  $\beta$  control their contributions. This hybrid approach ensures robust segmentation by mitigating class imbalance, preserving object boundaries [21].

These loss functions guide model training, evaluation, and comparison across architectures.

### 3.1.1 UNet-based end-to-end Neural Network

The implementation of U-Net is structured for image segmentation tasks, adhering to the classic encoder-decoder architecture to effectively capture both local and global context.

The encoder component comprises four convolutional blocks, each containing two convolutional layers followed by batch normalization and ReLU activation. Additionally, dropout with a rate of 0.2 is incorporated within each block to enhance robustness and mitigate overfitting. Max pooling layers with stride of 2 are employed to reduce dimensions and capture coarse features.

To facilitate efficient learning and ensure stable gradient propagation, batch normalization and ReLU activations are utilized throughout the convolutional blocks. This combination accelerates convergence and enhances the model's ability to learn complex patterns.

The decoder is designed to restore the spatial dimensions of the encoded features through transposed convolutions for upsampling. Each upsampling step is followed by a convolutional block analogous to those in the encoder. To preserve critical spatial details that may be lost during downsampling, skip connections are introduced, directly concatenating feature maps from the encoder to the corresponding decoder layers. This integration of contracting and expansive paths aids in maintaining high-frequency details essential for accurate segmentation.

The final output layer employs a  $1 \times 1$  convolution to map the learned features to the desired number of output classes, ensuring precise segmentation. By incorporating dropout, batch normalization, and skip connections, the U-Net model achieves a balance between generalization and fine-grained segmentation performance (Appx. A.2.1).

### 3.1.2 Autoencoder-based Decoder

**Pre-Training: Training the Encoder** - The autoencoder learns compact image representations by encoding and reconstructing inputs without labels (Appx. A.1.6). It consists of two components: an encoder that compresses the image and a decoder that reconstructs it.

- **Encoder:** A series of convolutional layers (64, 128, 256, 128 features) with max pooling reduce spatial resolution, while LeakyReLU activations capture low- and mid-level features (e.g., edges, textures, shapes). The output is a latent vector encoding image information.
- **Decoder:** Transposed convolutions upsample the latent vector to reconstruct the image. LeakyReLU activations follow each layer, with a final sigmoid output in [0,1]. Reconstruction loss is computed using Mean Squared Error (MSE), encouraging the network to capture meaningful features for downstream tasks like segmentation.

**Fine-Tuning: Fixed Encoder and Segmentation Decoder** - After pre-training, the encoder is paired with a segmentation decoder for pixel classification.

- **Fixed Encoder:** Its weights remain unchanged to preserve learned representations and prevent overfitting. Acting as feature extractor, it outputs compact image embeddings.
- **Segmentation Decoder:** Built with transposed convolutions and residual blocks, it upsamples latent features and maintains gradient flow via skip connections. The decoder

maps features to per-pixel class predictions, producing segmentation maps with categories like background, cat, dog, or outline.

Some experiments which will be discussed in later sections unfreeze this encoder and thus allow it to learn for some experiments as well (Appx. A.2.2).

### 3.1.3 CLIP-bolstered Decoder

The implementation of CLIP features for image segmentation leverages the pre-trained ViT-B-32 model to extract high-level semantic representations. These CLIP features are utilized alongside convolutional neural network (CNN)-based spatial features to construct a robust segmentation pipeline. The CLIP features are extracted directly from the input image using the model’s image encoder, normalized for consistency, and stored individually in a designated directory.

In the decoder architecture, the extracted 512-dimensional CLIP features are projected to 256 dimensions through a fully connected layer. Simultaneously, the input image is processed by a lightweight CNN encoder comprising two convolutional layers, ReLU activations, and max pooling operations to capture spatial details. The projected CLIP features are then concatenated with the CNN output along channel dimension, integrating semantic and spatial information.

The fused features are subsequently passed through a decoder network composed of transposed convolutions and upsampling layers, which reconstruct the segmentation map at the desired resolution. This combination of CLIP’s rich semantic understanding with the spatial precision of CNN-based features enables the model to generate context-aware accurate masks (Appx. A.2.3).

### 3.1.4 Prompt-enriched CLIP-bolstered Decoder

The proposed prompt-enriched CLIP-based decoder extends the previous model by incorporating additional spatial context through a location-specific heatmap. The primary objective of this architecture is to improve segmentation accuracy by directing the model’s attention toward specific anatomical features, specifically the nose of the animal. To achieve this, a heatmap corresponding to the nose location is generated and appended as a fourth channel to the original RGB image, resulting in a unified 4-channel input. These heatmaps were manually annotated to ensure the accuracy and reliability of the training data, as automated nose detection algorithms were found to be insufficiently precise.

As in the previous model, CLIP features are extracted using the pre-trained ViT-B-32 model and normalized to maintain consistency. These features are then concatenated with the augmented image, forming a fused input that integrates CLIP’s high-level semantic information with the spatial specificity provided by the heatmap. This combined representation is subsequently processed by the decoder network.

The decoder architecture retains the structure of the prior model, with modifications to the encoder to accommodate the 4-channel input. The network begins by projecting the 512-dimensional CLIP features to 256 dimensions using a fully connected layer. Concurrently, the augmented image (RGB + heatmap) is processed through a CNN encoder consisting of two convolutional layers with ReLU activations and max pooling operations. The projected CLIP features are reshaped and expanded to match the spatial dimensions of the CNN output, after which the two feature sets are concatenated along the channel axis (Appx. A.2.4).

The fused representation is then passed through the decoder module, which consists of transposed convolution layers and upsampling operations to reconstruct the segmentation map. By incorporating location-based heatmaps alongside CLIP’s semantic features, the model effectively integrates prompt-based training that aligns more closely with object-specific anatomical structures. This targeted fusion of contextual and semantic information enhances precision.

## 3.2 User Interface

The best-performing model was integrated into an interactive User Interface (UI), enabling direct user interaction and multiple forms of input (Appx. A.1.4 + demo attached).

For part (a), users are prompted to upload an image and select a point of interest by clicking on it. This interaction generates a heatmap, which is then processed by the prompt-based CLIP model described earlier. The segmentation result is subsequently displayed, allowing users to refine the segmentation process by providing explicit input.

For part (b), additional functionality was implemented to support multiple forms of prompts, including scribbling and bounding box inputs. Users can either draw directly on the uploaded image to guide the model toward specific shapes or define a bounding box around the region of interest. These features provide enhanced flexibility, enabling users to influence the segmentation outcome based on their intended focus.

By integrating point-based selection, freehand annotations, and bounding box inputs, the interface offers an intuitive and interactive experience, significantly improving the model’s segmentation capabilities.

## 3.3 Models Evaluation

The best-performing epochs are chosen by examining the training and validation loss curves (Appx. A.1.5), to ensure accuracy and generalization of the models.

We evaluate our models on three metrics: Intersection over Union (IoU) Score, Dice Score, and Pixel Accuracy.

**Intersection over Union (IoU) Score:** measures the overlap between the predicted and ground truth segmentation masks, calculated as the ratio of the intersection to the union:  $\text{IoU} = \frac{|A \cap B|}{|A \cup B|}$ , where  $A$  is the predicted segmentation mask and  $B$  is the ground truth mask for a given class [22].

**Dice Score:** evaluates the similarity between the predicted and ground truth segmentation masks. It is defined as  $\text{Dice} = \frac{2 \cdot |A \cap B|}{|A| + |B|}$ , where  $A$  and  $B$  are the predicted and ground truth masks, respectively [23].

**Pixel Accuracy:** metric computes the ratio of correctly predicted pixels to the total number of valid pixels:  $\text{Pixel Accuracy} = \frac{\text{Number of correct pixels}}{\text{Total number of valid pixels}}$  [24]

Note that the outline class, was used in the labels while training the models. However, it was omitted for the calculation of the above metrics.

Another point to note is that the segmentation output from the models was resized to match the Ground Truth label size of the image to ensure that there was no loss of information when calculating these scores.

### 3.3.1 Robustness Exploration

Subsequently, a robustness exploration is run on CLIP and its prompt-enriched counterpart, aiming at uncovering their performance’s dependence on images’ quality. Incremental levels of the following perturbation types were applied, tuning the specified parameter; the mean Dice score on *Test* set was used as metric, re-calculated at each gradual input worsening, comparing the predicted and gorund-truth masks directly at models’ output size to ignore resizing effects.

**Perturbations applied** on test images (or heatmaps, in the last case)

- a) Gaussian Noise: adds Gaussian-distributed noise to each pixel with increasing variance
- b) Gaussian Blurring: applies a 3x3 mask iteratively to approximate Gaussian blur
- c) Contrast Increase: multiplies pixel values by an increasing factor > 1, clamping at 255

- d) Contrast Decrease: multiplies pixel values by a decreasing factor  $< 1$
- e) Brightness Increase: adds an incremental offset to each pixel, clamping at 255
- f) Brightness Decrease: subtracts an incremental offset from each pixel, clamping at 0
- g) Square Occlusion: replaces a random square region of incremental size with black pixels
- h) Salt & Pepper Noise: randomly sets pixels to black or white with increasing probability
- i) Imprecise Prompt: shifts the heatmap in random direction by increasing number of pixels

## 4 Results

Model	Loss function	Epoch	Per-class IoU			Mean IoU	Mean Dice	Pixel Acc.(%)
			bkg	cat	dog			
<b>UNet</b>	Cross-entropy	39	.844	.488	.544	.6255	.5764	86.48
	Dice	45	.876	.498	.628	.6680	.6680	87.78
	C.-E.+Dice	22	.861	.485	.558	.6347	.5829	86.00
<b>Frozen</b>	Cross-entropy	92	.821	.374	.480	.5582	.5233	81.73
<b>AutoEncoder-based</b>	Dice	77	.800	.329	.421	.5168	.5098	76.79
	C.-E.+Dice	97	.821	.369	.485	.5581	.5230	81.06
<b>Fine-tuned</b>	Cross-entropy	54	.849	.458	.557	.6213	.5631	85.41
<b>AutoEncoder-based</b>	Dice	96	.837	.408	.503	.5825	.5438	81.46
	C.-E.+Dice	44	.857	.456	.573	.6287	.5636	85.24
<b>CLIP-bolstered</b>	Cross-entropy	100	.776	.654	.630	.6868	.7880	85.72
	Dice	61	.735	.598	.579	.6374	.7548	78.14
	C.-E.+Dice	14	.774	.649	.628	.6835	.7883	85.65
<b>Prompt-enriched</b>	Cross-entropy	31	.796	.677	.657	.7098	.8108	87.92
	Dice	87	.744	.627	.610	.6608	.7766	80.56
<b>CLIP-b.</b>	C.-E.+Dice	15	.782	.673	.649	.7013	.8032	85.04

Table 2: Segmentation Model Performance Metrics

By considering the all test results obtained (Tab. 2), we notice that the end-to-end approach gained already satisfactory results on the task at issue. In fact, based on properly pre-processed data and leveraging a sufficiently deep architecture, the **UNet** model reaches an IoU score over 0.6, with the background label predicted more accurately than animal ones, and an almost balanced performance on cat and dog classes.

A similar overall performance is noticeable for the fine-tuned **AutoEncoder-based** segmenter, which though present a less negligible gap between the animal classes' IoUs of about 0.1 for any loss function used. In general, this strategy did not outperform UNet models, probably due to its inherently high reliance on the input images' variability balance. In other words, the reconstructive AE might have suffered the cat instances, despite balanced in number through the preliminary 180°-rotated duplication (Sec. 2.2), were expressing less widely their species-relevant features with respect to dog ones, as the original dataset actually consists by less than  $\frac{1}{3}$  of cat images. Ultimately causing the subsequent segmentation model to be fed biased embeddings, this impeded learning is well exemplified by the frozen counterpart of the same model: when the AutoEncoder layers are not fine-tuned for classification, the leveraged latent representations can be not even partially corrected at training time to offset the specific task's labelling asymmetry, resulting in consistently lower scores.

On the other hand, the **CLIP-bolstered** model gains a coherently spread set of per-class IoU, with cats being discriminated even slightly better than dogs; a balance mirrored also by unparalleled Dice scores, despite standard pixel accuracies. Being pre-trained for computer vision purposes on a huge and inclusive set of instances, this feature-encoder manages indeed to provide highly valuable embeddings, which leveraged together with the dataset at issue allow to totally overcome its disequilibrium problem and reach the best performance. Preserving this quality, the mean IoU and Dice grow even more in the prompt-enriched case, demonstrating that the heatmap strategy works in hinting the animal-characteristic regions, making the learning quicker for the cross-entropy training too.

The CLIP-bolstered best-performing models undergo a **robustness exploration** too (Fig, 3), which unveils an higher sensibility for Salt & Pepper Noise in particular, while square occlusion seems to have relatively little influence on the segmentation output. Especially in the latter case, but consistently across all perturbation types, the prompt-aided method shows significantly better performances, with interestingly slower Dice scores reductions deriving from Gaussian Blurring and Brightness Decrease, as also in the case of Gaussian and S&P Noise applications from certain levels on. Furthermore, the heatmap-based model also demonstrated good robustness for small shifts, meaning that segmentation is comprehensibly impaired just when the prompt is clearly imprecise (i.e., indicating outside of the animal at all, rather than simply far from its muzzle).

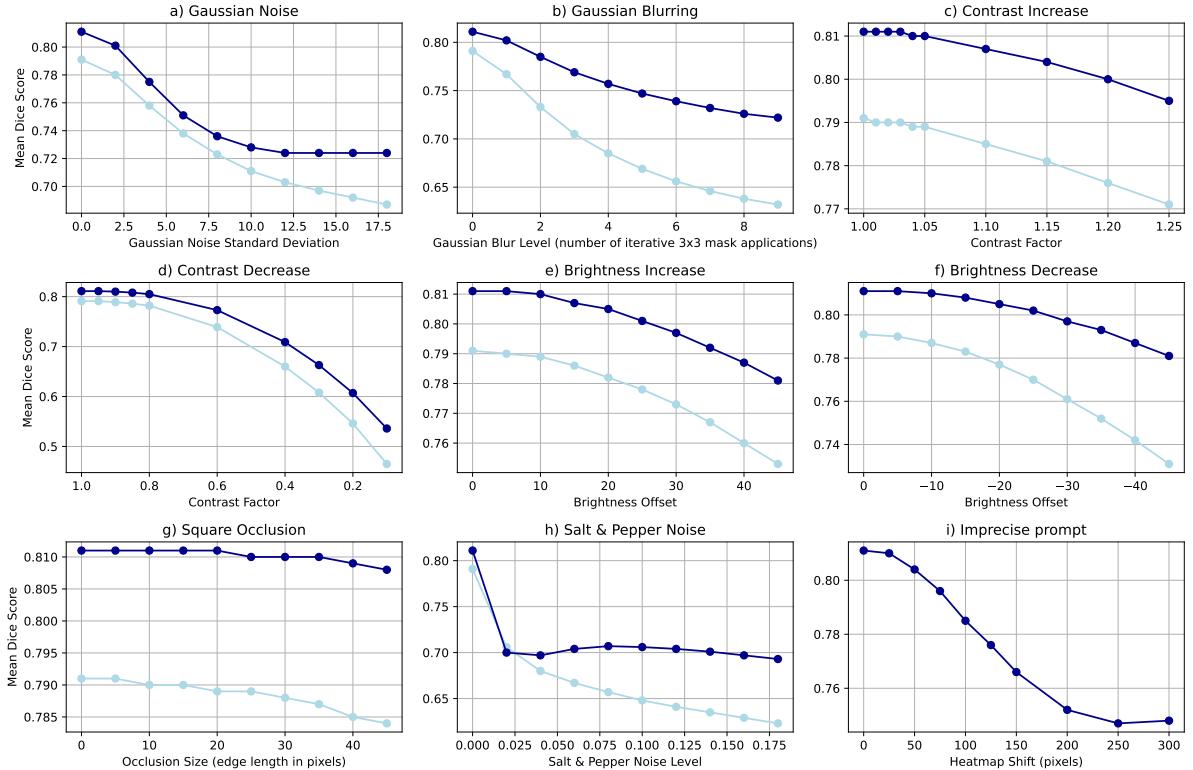


Figure 3: Comparative robustness exploration



## 5 Conclusion

The experimental objective was reached, individuating ML models which leverage widely pre-trained feature-extractors as better performing on limited datasets, fostering generalization and overcoming label asymmetries in the available instances. The effects of the latter problem appear more difficult to avoid by training embedders on the same dataset, as biased latent rep-

resentations would lead to poorer segmentation; in this case, even a plain end-to-end approach might reach better results.

One of the key challenges addressed was the class imbalance and label asymmetry in the dataset, which contained four distinct pixel-level labels: background, cat, dog, and outline. Models that relied on feature extractors pre-trained on large, diverse datasets (such as CLIP) were more robust to this imbalance, compared to those trained entirely end-to-end on the same small dataset. In contrast, models without such priors tended to develop biased latent representations, leading to poorer segmentation performance. Interestingly, the U-Net model successfully handled the presence of both a cat and a dog in the same image (Appx. A.1.3), accurately segmenting each. This shows the strength of neural networks in segmentation tasks.

The inclusion of the "outline" class, which serves as a boundary between animals and background (or between different animals), proved beneficial: though often predicted slightly thicker than annotated, it helped guide the model in cleanly separating adjacent regions, improving segmentation results.

Overall, Dice loss does not significantly help to reach better accuracy in the long run, probably because it focuses on counterbalancing the image-wise *background*'s higher pixel presence, while this class presents more complex and varied features to learn. However, in co-presence with cross-entropy, Dice ensures faster learning, making the architectures reach comparably satisfactory performances in a lower number of epochs.

In summary, architectures that integrate pre-trained vision-language models such as CLIP, paired with hybrid loss functions, were found to be the most effective for semantic segmentation under low-data conditions. Furthermore, the performance was enhanced by a prompt-based model that extended CLIP with annotated guidance, allowing the network to focus on salient image regions. This approach led to noticeable improvements in segmentation outputs.

## 5.1 Future work

To solve the class imbalance, having proper computational resources, label weights might be tuned through a grid search which can precisely offset this disequilibrium at loss computation time. These coefficients could in fact be trial-and-error experimented starting from values inversely proportional to the overall pixel-presence in the dataset; however, they might be tuned to find the best solution also considering the variability of features in each label. For instance, although the background has far more pixels in this case, it is represented by many different types of visual elements compared to the animals' areas, making its patterns harder to learn. A larger number of pixel-instances can aid learning up to a point, so the optimal ratio of background loss-weight to that of an animal class isn't simply expected to follow the inverse of their total pixel count ratio.

Additionally, experimenting with different combinations and relative weightings of Dice loss and cross-entropy loss presents another avenue for optimization. Such hybrid loss functions could allow better control over pixel-level precision and region-level coherence, particularly in boundary and minority-class regions. Furthermore, exploring transformer-based segmentation architectures, such as Segmenter or Mask2Former, may yield performance gains, especially when fine-tuned on tailored augmentations. Lastly, expanding the dataset through synthetic data generation, including techniques like GAN-based image synthesis or diffusion models, could alleviate data scarcity and improve generalization.

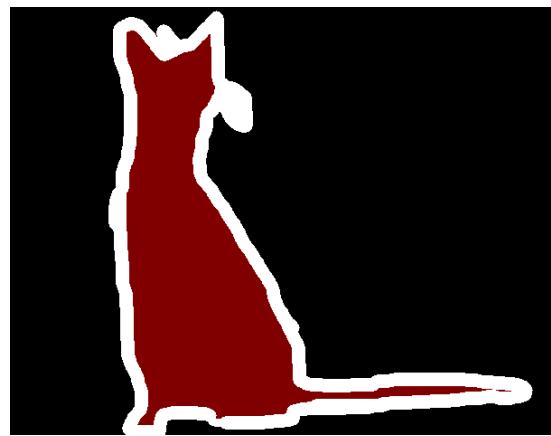
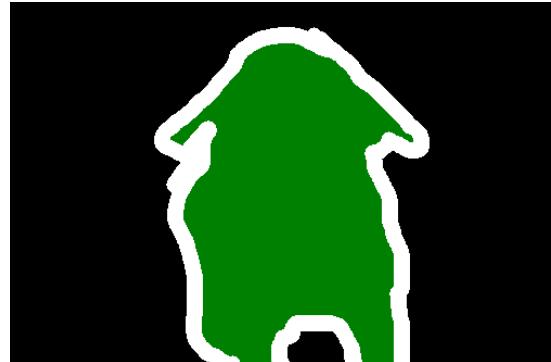
## A Appendices

### A.1 Additional material

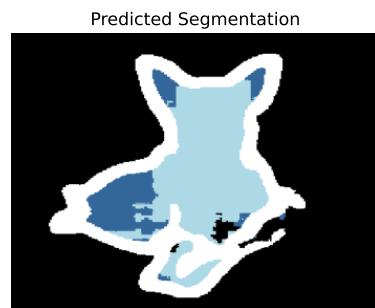
#### A.1.1 Counts of cats and dogs images by breed in the training/validation and test datasets

Species counts	Train/Val set	Test set
<b>Cats</b>		
Abyssinian	100	100
Bengal	100	100
Birman	100	100
Bombay	96	104
British Shorthair	100	100
Egyptian Mau	93	107
Maine Coon	100	100
Persian	100	100
Ragdoll	100	100
Russian Blue	100	100
Siamese	99	101
Sphynx	100	100
<b>Total cats</b>	1188	1212
<b>Dogs</b>		
American Bulldog	100	100
American Pit Bull Terrier	100	100
Basset Hound	100	100
Beagle	100	100
Boxer	100	100
Chihuahua	100	100
English Cocker Spaniel	96	104
English Setter	100	100
German Shorthaired	100	100
Great Pyrenees	100	100
Havanese	100	100
Japanese Chin	100	100
Keeshond	100	100
Leonberger	100	100
Miniature Pinscher	100	100
Newfoundland	96	104
Pomeranian	100	100
Pug	100	100
Saint Bernard	100	100
Samoyed	100	100
Scottish Terrier	100	99
Shiba Inu	100	100
Staffordshire Bull Terrier	100	91
Wheaten Terrier	100	100
Yorkshire Terrier	100	100
<b>Total dogs</b>	2492	2498

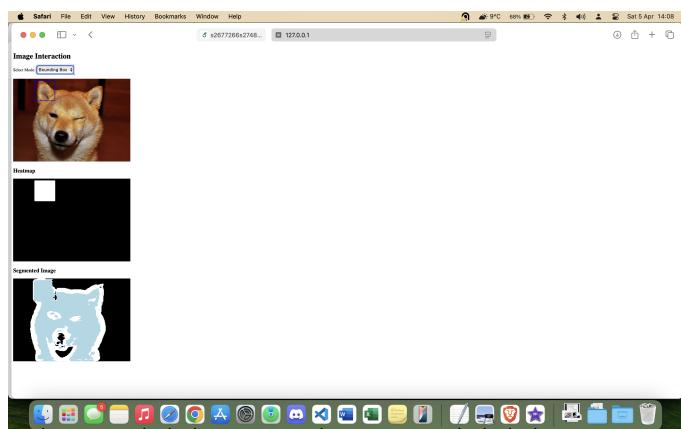
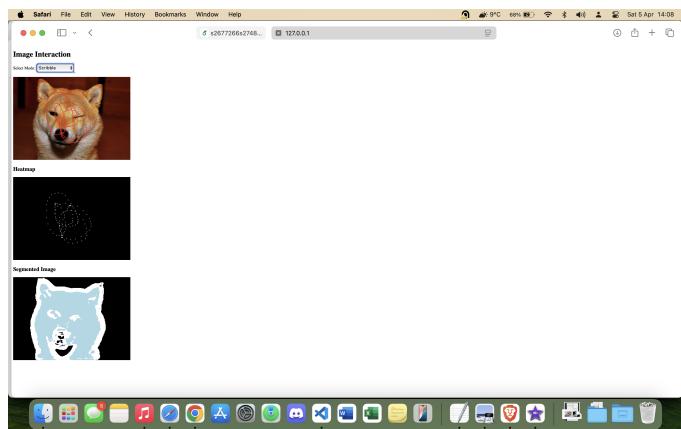
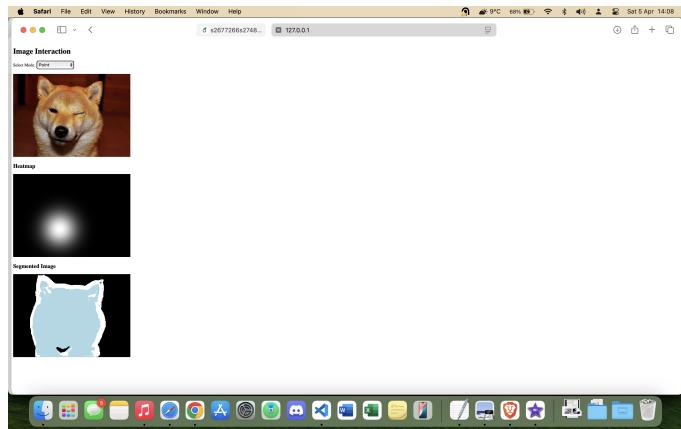
### A.1.2 Examples of cat- and dog-labelled instances



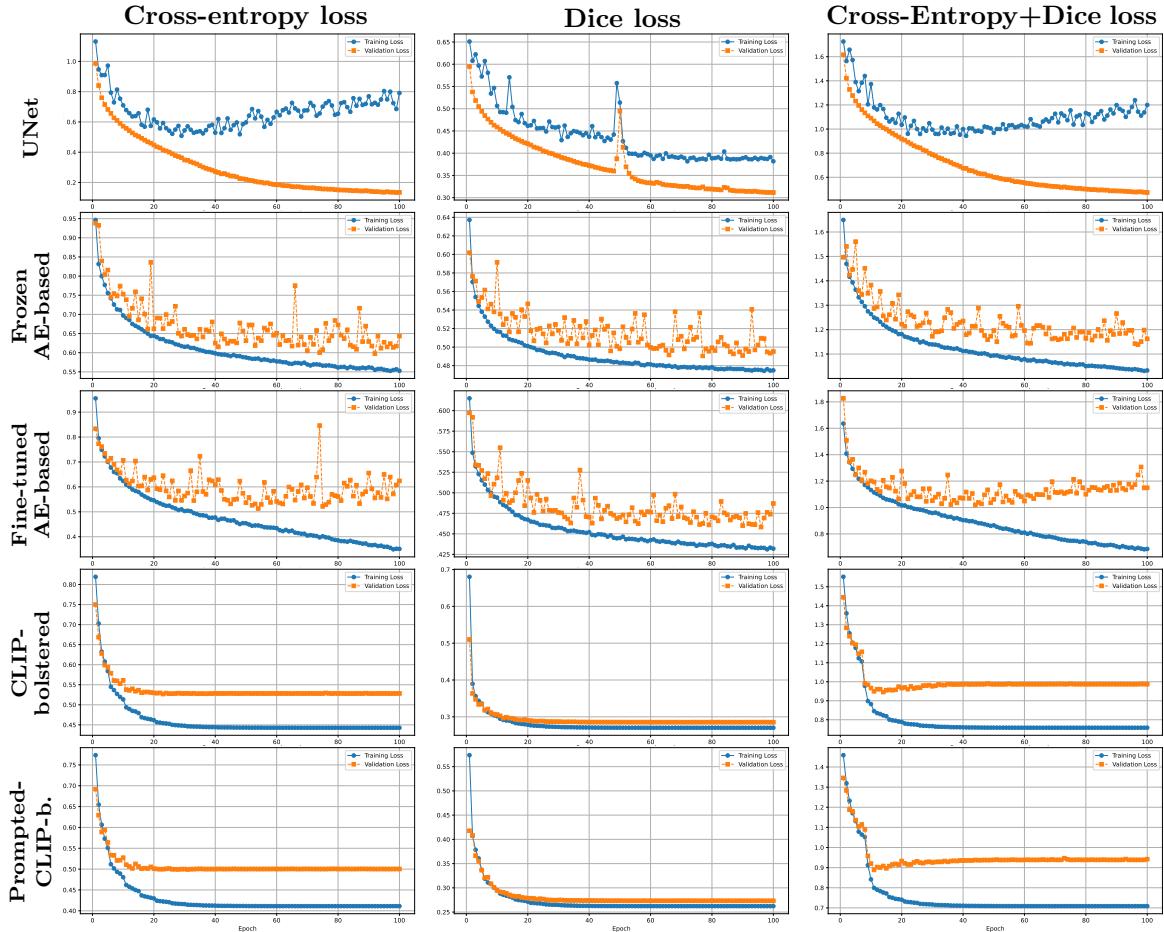
### A.1.3 UNet segmentation of an example test image containing both animals



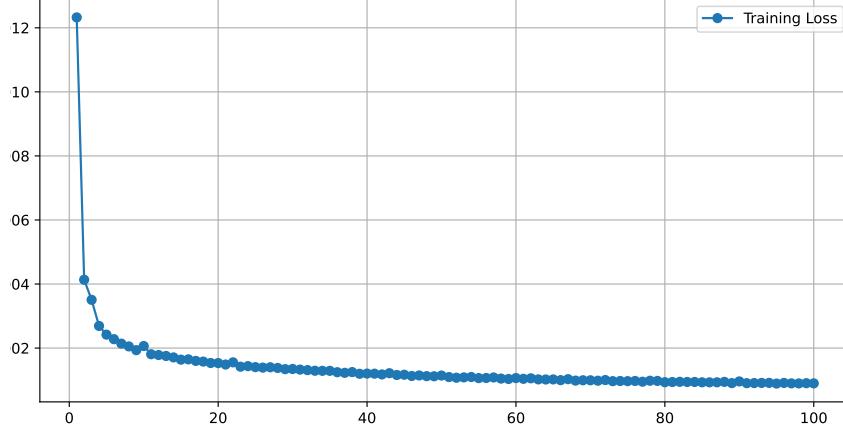
#### A.1.4 UI screenshots: point- scribble- and box-based examples



### A.1.5 Models learning: *Train* and *Val* loss curves by architecture and loss type



### A.1.6 Autoencoder pre-training: MSE reconstruction loss across epochs



### A.1.7 Prompt-enriched CLIP's robustness: Dice scores for significant perturbation levels

Transformation	(Parameter, Score)
a) Gaussian Noise	(0, 0.811), (4, 0.775), (8, 0.736), (12, 0.724), (18, 0.724)
b) Gaussian Blurring	(0, 0.811), (3, 0.769), (6, 0.739), (9, 0.722)
c) Contrast Increase	(1.00, 0.811), (1.02, 0.811), (1.05, 0.810), (1.15, 0.804), (1.25, 0.795)
d) Contrast Decrease	(1.00, 0.811), (0.90, 0.810), (0.80, 0.805), (0.40, 0.709), (0.10, 0.536)
e) Brightness Increase	(0, 0.811), (10, 0.810), (25, 0.801), (40, 0.787), (45, 0.781)
f) Brightness Decrease	(0, 0.811), (-10, 0.810), (-25, 0.802), (-40, 0.787), (-45, 0.781)
g) Square Occlusion	(0, 0.811), (15, 0.811), (30, 0.810), (45, 0.808)
h) Salt & Pepper Noise	(0.00, 0.811), (0.04, 0.697), (0.10, 0.706), (0.18, 0.693)
i) Imprecise Prompt	(0, 0.811), (50, 0.804), (100, 0.785), (200, 0.752), (300, 0.748)

## A.2 Model Architectures

### A.2.1 U-Net

```
UNet(  
    - Encoder:  
        - enc1: ConvBlock(3 -> 64)  
        - pool1: MaxPool2d(2)  
        - enc2: ConvBlock(64 -> 128)  
        - pool2: MaxPool2d(2)  
        - enc3: ConvBlock(128 -> 256)  
        - pool3: MaxPool2d(2)  
        - enc4: ConvBlock(256 -> 512)  
  
    - Decoder:  
        - up5: ConvTranspose2d(512 -> 256)  
        - dec5: ConvBlock(512 -> 256)  
        - up6: ConvTranspose2d(256 -> 128)  
        - dec6: ConvBlock(256 -> 128)  
        - up7: ConvTranspose2d(128 -> 64)  
        - dec7: ConvBlock(128 -> 64)  
  
    - Output:  
        - final_conv: Conv2d(64 -> num_classes, kernel_size=1)  
)
```

ConvBlock(in -> out) consists of:

- Conv2d(in, out, kernel=3, padding=1)
- BatchNorm2d(out)
- ReLU
- Dropout(0.2)
- Conv2d(out, out, kernel=3, padding=1)
- BatchNorm2d(out)
- ReLU
- Dropout(0.2)

### A.2.2 Autoencoder

Pre-Training of Encoder Architecture:

```
Autoencoder(  
    - Encoder:  
        - Conv2d(3 -> 64, kernel=3, stride=1, padding=1)  
        - LeakyReLU(0.2)  
        - MaxPool2d(kernel=2, stride=2)  
        - Conv2d(64 -> 128, kernel=3, stride=1, padding=1)  
        - LeakyReLU(0.2)  
        - MaxPool2d(kernel=2, stride=2)  
        - Conv2d(128 -> 256, kernel=3, stride=1, padding=1)  
        - LeakyReLU(0.2)
```

```

    - MaxPool2d(kernel=2, stride=2)
    - Conv2d(256 -> 128, kernel=3, stride=1, padding=1)
    - LeakyReLU(0.2)

    - Decoder:
        - ConvTranspose2d(128 -> 256, kernel=3, stride=2, padding=1, output_padding=1)
        - LeakyReLU(0.2)
        - ConvTranspose2d(256 -> 128, kernel=3, stride=2, padding=1, output_padding=1)
        - LeakyReLU(0.2)
        - ConvTranspose2d(128 -> 64, kernel=3, stride=2, padding=1, output_padding=1)
        - LeakyReLU(0.2)
        - Conv2d(64 -> 64, kernel=3, stride=1, padding=1)
        - LeakyReLU(0.2)
        - Conv2d(64 -> 3, kernel=3, stride=1, padding=1)
        - Sigmoid
)

```

Fine-Tuning with Frozen Encoder Architecture:

```

SegmentationDecoder(
    - Encoder: <frozen pre-trained encoder, not trained during decoder optimization>

    - Decoder:
        - ConvTranspose2d(128 -> 128, kernel=3, stride=2, padding=1, output_padding=1)
        - ResidualBlock(128 -> 128)
        - ConvTranspose2d(128 -> 64, kernel=3, stride=2, padding=1, output_padding=1)
        - ResidualBlock(64 -> 64)
        - ConvTranspose2d(64 -> 32, kernel=3, stride=2, padding=1, output_padding=1)
        - ResidualBlock(32 -> 32)
        - Conv2d(32 -> 32, kernel=3, padding=1)
        - BatchNorm2d(32)
        - ReLU
        - Conv2d(32 -> num_classes, kernel=1)
)

```

Each Residual Block consists of the following:

```

ResidualBlock(in_channels -> out_channels):
    - Conv2d(in_channels -> out_channels, kernel=3, padding=1)
    - BatchNorm2d(out_channels)
    - ReLU
    - Conv2d(out_channels -> out_channels, kernel=3, padding=1)
    - BatchNorm2d(out_channels)
    - Skip Connection (input + output)
    - ReLU

```

(Note: To allow the encoder to continue training,  
set the requires\_grad to True in the encoder part.)

### A.2.3 Clip-Bolstered

```
SegmentationDecoder(  
    - CLIP Projection:  
        - clip_fc: Linear(clip_feature_dim -> 256)  
  
    - CNN Encoder:  
        - Conv2d(3 -> 64, kernel=3, stride=1, padding=1)  
        - ReLU  
        - MaxPool2d(kernel=2)  
        - Conv2d(64 -> 128, kernel=3, stride=1, padding=1)  
        - ReLU  
  
    - Decoder:  
        - ConvTranspose2d(384 (128 + 256) -> 64, kernel=3, stride=1, padding=1)  
        - ReLU  
        - Upsample(to 256x256, mode='bilinear', align_corners=True)  
        - ConvTranspose2d(64 -> num_classes, kernel=3, stride=1, padding=1)  
)
```

### A.2.4 Prompt-Enriched-Clip-Bolstered

```
SegmentationDecoder(  
    - CLIP Projection:  
        - clip_fc: Linear(clip_feature_dim -> 256)  
  
    - CNN Encoder (input: RGB + heatmap, 4 channels):  
        - Conv2d(4 -> 64, kernel=3, stride=1, padding=1)  
        - ReLU  
        - MaxPool2d(kernel=2)  
        - Conv2d(64 -> 128, kernel=3, stride=1, padding=1)  
        - ReLU  
  
    - Decoder:  
        - ConvTranspose2d(384 -> 64, kernel=3, stride=1, padding=1)  
        - ReLU  
        - Upsample(to 256x256, mode='bilinear', align\_corners=True)  
        - ConvTranspose2d(64 -> num_classes, kernel=3, stride=1, padding=1)  
)
```

## A.3 Code

The entire code consisting of Data Preprocess, Model Architectures, Model Training, Model Testing, Robustness Exploration and UI

### A.3.1 data\_preprocessing.py: Code to resize and augment the images

```

1 # Authors: Sahil Mehul Bavishi (s2677266) and Matteo Spadaccia (s2748897)
2 # Subject: Computer Vision Coursework. Pre-Processing
3 # Date: 21.03.2025
4
5 """
6     """0. Preliminary code"""
7
8 # Setting code-behaviour variables
9 doResize = False      # if True, the input images' resizing is run, otherwise the saved outcomes are used
10 doAugment = False    # if True, the resized images' augmentation is run, otherwise the saved outcomes are used
11 genVISUALS = False   # set to False in order to avoid time-consuming visualizations' generation (images are instead
12             # displayed as pre-saved in 'Output/Visuals' folder)
13 dpi = 500            # dpi for .pdf-saved images visualization (with genVISUALS = False)
14
15 # Importing useful libraries
16 import os
17 from pathlib import Path
18 from pdf2image import convert_from_path # (also install poppler-utils)
19 from IPython.display import display
20 from collections import Counter
21 from tabulate import tabulate
22 import numpy as np
23 from PIL import Image
24 import seaborn as sns
25 import cv2
26 import albumentations as A
27 import matplotlib.pyplot as plt
28
29 # Loading input data
30 input_folder_trainval = 'Data/Input/TrainVal'
31 input_trainval = [f for f in os.listdir(input_folder_trainval+'color') if f.lower().endswith('.jpg', '.jpeg')]
32 input_trainval_labels = [f for f in os.listdir(input_folder_trainval+'label') if f.lower().endswith('.png')]
33
34 input_folder_test = 'Data/Input/Test'
35 input_test = [f for f in os.listdir(input_folder_test+'color') if f.lower().endswith('.jpg', '.jpeg')]
36 input_test_labels = [f for f in os.listdir(input_folder_test+'label') if f.lower().endswith('.png')]
37
38 output_folder = 'Data/Output'
39 output_folder_resized = os.path.join(output_folder, 'Resized')
40 output_folder_augmented = os.path.join(output_folder, 'Augmented')
41
42 output_folder_resized_color = os.path.join(output_folder_resized, 'color')
43 output_folder_resized_label = os.path.join(output_folder_resized, 'label')
44 output_folder_augmented_color = os.path.join(output_folder_augmented, 'color')
45 output_folder_augmented_label = os.path.join(output_folder_augmented, 'label')
46 output_folder_resized_heatmaps = os.path.join(output_folder_resized, 'heatmaps')
47 output_folder_augmented_heatmaps = os.path.join(output_folder_augmented, 'heatmaps')
48
49 """
50     """1. Dataset exploration"""
51
52 # Gauging image sizes
53 if genVISUALS or doResize:
54     widths = []
55     heights = []
56     for img_file in input_trainval:
57         with Image.open(os.path.join(input_folder_trainval+'color', img_file)) as img:
58             width, height = img.size
59             widths.append(width)
60             heights.append(height)
61
62 # Visualizing histogram distribution of dimensions
63 if genVISUALS or doResize:
64     plt.figure(figsize=(12, 5))
65     plt.subplot(1, 2, 1)
66     plt.hist(widths, bins=20, color='lightblue', edgecolor='black')
67     plt.xlabel("Width")
68     plt.ylabel("Frequency")
69     plt.title("Widths distribution")
70     plt.subplot(1, 2, 2)
71     plt.hist(heights, bins=20, color='skyblue', edgecolor='black')
72     plt.xlabel("Height")
73     plt.ylabel("Frequency")
74     plt.title("Heights distribution")
75     plt.tight_layout()
76     plt.savefig(Path('Data/Output/Visuals/dimensions_histogram.pdf'))
77     plt.show()
78 else:
79     for image in convert_from_path(Path('Data/Output/Visuals/dimensions_histogram.pdf'), dpi=dpi):
80         display(image)
81
82 # Visualizing boxplot distribution of dimensions
83 if genVISUALS or doResize:

```

```

83     plt.figure(figsize=(10, 6))
84     data = {"Width": widths, "Height": heights}
85     sns.boxplot(data=data, palette=['lightblue', 'skyblue'])
86     plt.title("Boxplot of Widths and Heights")
87     plt.ylabel("Pixels")
88     plt.tight_layout()
89     plt.savefig(Path('Data/Output/Visuals/dimensions_boxplot.pdf'))
90     plt.show()
91 else:
92     for image in convert_from_path(Path('Data/Output/Visuals/dimensions_boxplot.pdf'), dpi=dpi):
93         display(image)
94
95 # Printing stats
96 print(f"Input set size: {len(input_trainval)}\n")
97 if doResize:
98     min_width, min_height = np.min(widths), np.min(heights)
99     median_width, median_height = np.median(widths), np.median(heights)
100    mean_width, mean_height = np.mean(widths), np.mean(heights)
101    mode_width, mode_height = max(set(widths), key=widths.count), max(set(heights), key=heights.count)
102    q3_width, q3_height = np.percentile(widths, 75), np.percentile(heights, 75)
103    iqr_width = np.percentile(widths, 75) - np.percentile(widths, 25)
104    iqr_height = np.percentile(heights, 75) - np.percentile(heights, 25)
105    outlier_count_width = np.sum(widths > (q3_width + 1.5 * iqr_width))
106    outlier_count_height = np.sum(heights > (q3_height + 1.5 * iqr_height))
107    print(f"Min Size: {min_width}x{min_height}")
108    print(f"Median Size: {median_width}x{median_height}")
109    print(f"Mean Size: {mean_width:.2f}x{mean_height:.2f}")
110    print(f"Mode Size: {mode_width}x{mode_height}")
111    print(f"Q3 Size: {q3_width}x{q3_height}")
112    print(f"Outliers in width: {outlier_count_width}")
113    print(f"Outliers in height: {outlier_count_height}")
114 else:
115     print("Q3 width and height values (both 500 pixels) were chosen for resizing.")
116
117 # Counting dogs' and cats' species instances
118 species_counts = Counter(filename.rsplit('_', 1)[0] for filename in input_trainval)
119 test_species_counts = Counter(filename.rsplit('_', 1)[0] for filename in input_test)
120 cat_species = ["abyssinian", "bengal", "birman", "bombay", "british_shorthair", "egyptian_mau", "maine_coon", "persian",
121     "ragdoll", "russian_blue", "siamese", "sphynx"]
122 cat_counts = {species: count for species, count in species_counts.items() if species.lower() in cat_species}
123 dog_counts = {species: count for species, count in species_counts.items() if species.lower() not in cat_species}
124 total_cats, total_dogs = sum(cat_counts.values()), sum(dog_counts.values())
125 test_cat_counts = {species: count for species, count in test_species_counts.items() if species.lower() in cat_species}
126 test_dog_counts = {species: count for species, count in test_species_counts.items() if species.lower() not in
127     cat_species}
128 test_total_cats, test_total_dogs = sum(test_cat_counts.values()), sum(test_dog_counts.values())
129
130 # Displaying breeds' distribution
131 table_data = []
132 for species in sorted(cat_counts.keys()):
133     table_data.append([species, cat_counts[species], test_cat_counts[species]])
134 table_data.append(["TOT_CATS", total_cats, test_total_cats])
135 for species in sorted(dog_counts.keys()):
136     table_data.append([species, dog_counts[species], test_dog_counts.get(species, 0)])
137 print(tabulate(table_data, headers=["Species", "Train/Val_Count", "Test_Count"], tablefmt="grid"))
138
139 """
140 # Images' resizing
141
142 # Resizing images and labels (to Q3 width, Q3 height)
143 if doResize:
144     imgResize = (int(q3_width), int(q3_height))
145     widthsNP = np.array(widths)
146     heightsNP = np.array(heights)
147     i = 0
148     for img_file in input_trainval:
149         with Image.open(os.path.join(input_folder_trainval+'/color', img_file)) as img:
150             img_resized = img.resize(imgResize, Image.Resampling.LANCZOS)
151             if img_resized.mode == "RGBA":
152                 img_resized = img_resized.convert("RGB")
153             img_resized.save(os.path.join(output_folder_resized_color, img_file), format="JPEG")
154             i += 1
155     print(f"{i} images resized to {int(q3_width)}x{int(q3_height)} and saved in {output_folder_resized_color}.")
156
157     imgResize = (int(q3_width), int(q3_height))
158     widthsNP = np.array(widths)
159     heightsNP = np.array(heights)
160     i = 0
161     for img_file in input_trainval_labels:
162         with Image.open(os.path.join(input_folder_trainval+'/label', img_file)) as img:
163             img_resized = img.resize(imgResize, Image.Resampling.LANCZOS)
164             if img_resized.mode == "RGBA":
165                 img_resized = img_resized.convert("RGB")
166             img_resized.save(os.path.join(output_folder_resized_label, img_file), format="PNG")
167             i += 1
168     print(f"{i} labels resized to {int(q3_width)}x{int(q3_height)} and saved in {output_folder_resized_label}.")
169
170 else:
171     print("Using previously resized images and labels (500x500).")
172     imgResize = (500, 500)
173
174 """
175 # Augmenting dataset

```

```

176
177 # Defining augmentations
178 augmentation = A.Compose([
179
180     # Geometric transformations (applied to all)
181     A.HorizontalFlip(p=0.5),
182     A.VerticalFlip(p=0.5),
183     A.RandomRotate90(p=0.5),
184     A.Affine(scale=(0.9, 1.1), translate_percent=(0.1, 0.1), rotate=(-15, 15), shear=(-10, 10), p=0.5),
185     A.ElasticTransform(alpha=1, sigma=50, alpha_affine=50, p=0.5),
186     A.GridDistortion(num_steps=5, distort_limit=0.3, interpolation=0, p=0.5), # NN-interpolation
187     A.OpticalDistortion(distort_limit=0.3, shift_limit=0.3, interpolation=0, p=0.5), # Nearest Neighbor
188
189     # Intensity augmentations (applied to images only)
190     A.GaussianBlur(blur_limit=(3, 7), p=0.3),
191     A.MotionBlur(blur_limit=5, p=0.3),
192     A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.3),
193     A.HueSaturationValue(hue_shift_limit=20, sat_shift_limit=30, val_shift_limit=20, p=0.3),
194     A.GaussNoise(var_limit=(10.0, 50.0), p=0.3),
195     A.MultiplicativeNoise(multiplier=(0.9, 1.1), p=0.3)
196 ], additional_targets={'mask': 'mask', 'heatmap': 'mask'})
197
198 def is_cat(filename, cat_breeds = cat_species):
199     filename_lower = filename.lower()
200     return any(breed in filename_lower for breed in cat_breeds)
201
202 if doAugment:
203     i = 0
204     print('Augmenting the data...')
205     for img_file in os.listdir(output_folder_resized_color):
206         img_path = os.path.join(output_folder_resized_color, img_file)
207         mask_filename = os.path.splitext(img_file)[0] + ".png"
208         mask_path = os.path.join(output_folder_resized_label, mask_filename)
209         heatmap_filename = os.path.splitext(img_file)[0] + "_heatmap.png"
210         heatmap_path = os.path.join(output_folder_resized_heatmaps, heatmap_filename)
211
212         # Loading image, mask(grayscale) and heatmap
213         img = cv2.imread(img_path)
214         mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
215         heatmap = cv2.imread(heatmap_path, cv2.IMREAD_GRAYSCALE)
216
217         if img is None or mask is None or heatmap is None:
218             print(f"Skipping {img_file} as corresponding image or mask is missing!")
219             continue
220
221         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
222
223         # Saving original image, mask and heatmap
224         Image.fromarray(img).save(os.path.join(output_folder_augmented_color, f'orig_{img_file}'), "JPEG")
225         Image.fromarray(mask).save(os.path.join(output_folder_augmented_label, f'orig_{mask_filename}'), "PNG")
226         Image.fromarray(heatmap).save(os.path.join(output_folder_augmented_heatmaps, f'orig_{heatmap_filename}'), "PNG")
227
228         # Applying augmentation
229         augmented = augmentation(image=img, mask=mask, heatmap=heatmap)
230         aug_img, aug_mask, aug_heatmap = augmented['image'], augmented['mask'], augmented['heatmap']
231         aug_mask = np.round(aug_mask).astype(np.uint8)
232         aug_heatmap = np.round(aug_heatmap).astype(np.uint8)
233
234         # Saving augmented image and mask
235         Image.fromarray(aug_img).save(os.path.join(output_folder_augmented_color, f'aug_{i}_{img_file}'), "JPEG")
236         Image.fromarray(aug_mask).save(os.path.join(output_folder_augmented_label, f'aug_{i}_{mask_filename}'), "PNG")
237         Image.fromarray(aug_heatmap).save(os.path.join(output_folder_augmented_heatmaps, f'aug_{i}_{heatmap_filename}'), "PNG")
238
239         i += 1
240
241         # If image is a cat, rotate 180 degrees and augment further
242         if is_cat(img_file):
243             rot_img = cv2.rotate(img, cv2.ROTATE_180)
244             rot_mask = cv2.rotate(mask, cv2.ROTATE_180)
245             rot_heatmap = cv2.rotate(heatmap, cv2.ROTATE_180)
246
247             Image.fromarray(rot_img).save(os.path.join(output_folder_augmented_color, f'rot_{img_file}'), "JPEG")
248             Image.fromarray(rot_mask).save(os.path.join(output_folder_augmented_label, f'rot_{mask_filename}'), "PNG")
249             Image.fromarray(rot_heatmap).save(os.path.join(output_folder_augmented_heatmaps, f'rot_{heatmap_filename}'), "PNG")
250
251             rotated_augmented = augmentation(image=rot_img, mask=rot_mask, heatmap=rot_heatmap)
252             rot_aug_img, rot_aug_mask, rot_aug_heatmap = rotated_augmented['image'], rotated_augmented['mask'],
253                 rotated_augmented['heatmap']
254             rot_aug_mask = np.round(rot_aug_mask).astype(np.uint8)
255             rot_aug_heatmap = np.round(rot_aug_heatmap).astype(np.uint8)
256
257             Image.fromarray(rot_aug_img).save(os.path.join(output_folder_augmented_color, f'rot_aug_{i}_{img_file}'), "JPEG")
258             Image.fromarray(rot_aug_mask).save(os.path.join(output_folder_augmented_label, f'rot_aug_{i}_{mask_filename}'), "PNG")
259             Image.fromarray(rot_aug_heatmap).save(os.path.join(output_folder_augmented_heatmaps, f'rot_aug_{i}_{heatmap_filename}'), "PNG")
260
261             i += 1
262
263         print(f'{i} images and masks augmented, including originals, output saved in {output_folder_augmented_color} & {output_folder_augmented_label}')

```

```

263
264     else:
265         print("Using previously augmented data.")

```

### A.3.2 U-Net.py: Unet Architecture and Training

```

1 # Authors: Sahil Mehl Bavishi (s2677266) and Matteo Spadaccia (s2748897)
2 # Subject: Computer Vision Coursework. U-Net
3 # Date: 21.03.2025
4
5
6 # Global Constants
7 genVISUALS = False # set to False in order to avoid time-consuming visualizations' generation (images are instead
8 # displayed as pre-saved in 'Output/Visuals' folder)
9 dpi = 500 # dpi for .pdf-saved images visualization (with genVISUALS = False)
10 rePREPROC = False # if True, the input images' resizing and augmentation are run, otherwise the saved outcomes are
# used
11 random_seed = 42
12
13 # IMPORTS
14 import os
15 from pathlib import Path
16 from pdf2image import convert_from_path # (also install !apt-get install poppler-utils)
17 from IPython.display import display
18 import numpy as np
19 from PIL import Image
20 import seaborn as sns
21 import cv2
22 import tensorflow as tf
23 import albumentations as A
24 from sklearn.model_selection import train_test_split
25 import matplotlib.pyplot as plt
26 import torch
27 import torch.nn as nn
28 import torch.optim as optim
29 import torch.nn.functional as F
30 from torch.utils.data import DataLoader, Dataset
31 from torchvision import transforms
32 from torch.optim.lr_scheduler import StepLR
33 from tqdm import tqdm
34 import pickle as pkl
35
36
37 # Loading input data
38 input_folder_trainval = 'Data/Input/TrainVal'
39 input_trainval = [f for f in os.listdir(input_folder_trainval+'color') if f.lower().endswith('.jpg', '.jpeg')]
40 input_trainval_labels = [f for f in os.listdir(input_folder_trainval+'label') if f.lower().endswith('.png')]
41
42 input_folder_test = 'Data/Input/Test',
43 input_test = [f for f in os.listdir(input_folder_test+'color') if f.lower().endswith('.jpg', '.jpeg')]
44 input_test_labels = [f for f in os.listdir(input_folder_test+'label') if f.lower().endswith('.png')]
45
46 output_folder = 'Data/Output'
47 output_folder_resized = os.path.join(output_folder, 'Resized')
48 output_folder_augmented = os.path.join(output_folder, 'Augmented')
49
50 output_folder_resized_color = os.path.join(output_folder_resized, 'color')
51 output_folder_resized_label = os.path.join(output_folder_resized, 'label')
52 output_folder_augmented_color = os.path.join(output_folder_augmented, 'color')
53 output_folder_augmented_label = os.path.join(output_folder_augmented, 'label')
54
55 ##### 1. Dataset preprocessing and augmentation
56
57 The images are resized to the dimensions ( $H_{min}, W_{min}$ ), thus to take the Q3 height and width size
over all the images in the dataset; the instances will be processed in this format, then the output resized back
to the original dimensions...
58
59 Furthermore...(data augmentation)
60 """
61
62 # Printing stats
63 print(f"Input set size: {len(input_trainval)}\n")
64 print("Q3 width and height values (both 500 pixels) were chosen for resizing.")
65
66 ##### a) Resizing"""
67
68 # Resizing images (to Q3 width, Q3 height)
69 print("Using previously resized images and labels (500x500).")
70 imgResize = (256, 256)
71
72 ##### b) Augmenting dataset"""
73 print("Using previously augmented data.")
74
75
76 ##### c) Preparing datasets"""
77
78 # Preparing train, valid and test sets
79 validSize = 0.2
80 batchSize = 16
81 imgChannels = 3 ## this was 3

```

```

82     inputSize = (imgResize[0], imgResize[1], imgChannels)
83
84     import os
85     os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "expandable_segments:True"
86
87     # Define preprocessed size
88     preprocessedSize = imgResize # Desired size for input images and masks
89     inputSize = (128,128,3)
90
91     trainval_images = [os.path.join(output_folder_augmented_color, f) for f in os.listdir(output_folder_augmented_color)
92                         if f.endswith('.jpg')]
93     trainval_masks = [os.path.join(output_folder_augmented_label, f) for f in os.listdir(output_folder_augmented_label) if
94                         f.endswith('.png')]
95
96     train_images, val_images, train_masks, val_masks = train_test_split(trainval_images, trainval_masks, test_size=
97                         validSize, random_state=random_seed)
98
99
100    print(f"Train set size: {len(train_images)} ({((1-validSize)*100)%})")
101    print(f"Valid set size: {len(val_images)} ({(validSize)*100}%})")
102    print(f"\nTest set size: {len(test_images)}")
103    print(f"\nInput dimension: {preprocessedSize+imgChannels} # Adjusted for the preprocessed size")
104    print(f"\nBatches size: {batchSize}")
105
106    classesNum = 4 # number of output classes
107    epochsNum = 100 # number of training epochs
108    batchSize = 16
109
110
111    # Define the UNet architecture
112    class UNet(nn.Module):
113        def __init__(self, num_classes=classesNum):
114            super(UNet, self).__init__()
115
116            # Encoder
117            self.enc1 = self.conv_block(3, 64)
118            self.pool1 = nn.MaxPool2d(2)
119
120            self.enc2 = self.conv_block(64, 128)
121            self.pool2 = nn.MaxPool2d(2)
122
123            self.enc3 = self.conv_block(128, 256)
124            self.pool3 = nn.MaxPool2d(2)
125
126            self.enc4 = self.conv_block(256, 512)
127
128            # Decoder
129            self.up5 = nn.ConvTranspose2d(512, 256, kernel_size=2, stride=2)
130            self.dec5 = self.conv_block(512, 256)
131
132            self.up6 = nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2)
133            self.dec6 = self.conv_block(256, 128)
134
135            self.up7 = nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2)
136            self.dec7 = self.conv_block(128, 64)
137
138            self.final_conv = nn.Conv2d(64, num_classes, kernel_size=1) # output layer
139
140        def conv_block(self, in_channels, out_channels):
141            return nn.Sequential(
142                nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
143                nn.BatchNorm2d(out_channels),
144                nn.ReLU(inplace=True),
145                nn.Dropout(0.2),
146                nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1),
147                nn.BatchNorm2d(out_channels),
148                nn.ReLU(inplace=True),
149                nn.Dropout(0.2)
150            )
151
152        def forward(self, x):
153            # Encoder
154            c1 = self.enc1(x)
155            p1 = self.pool1(c1)
156
157            c2 = self.enc2(p1)
158            p2 = self.pool2(c2)
159
160            c3 = self.enc3(p2)
161            p3 = self.pool3(c3)
162
163            c4 = self.enc4(p3)
164
165            # Decoder
166            u5 = self.up5(c4)
167            u5 = torch.cat([u5, c3], dim=1)
168            c5 = self.dec5(u5)
169
170            u6 = self.up6(c5)
171            u6 = torch.cat([u6, c2], dim=1)

```

```

172     c6 = self.dec6(u6)
173
174     u7 = self.up7(c6)
175     u7 = torch.cat([u7, c1], dim=1)
176     c7 = self.dec7(u7)
177
178     outputs = self.final_conv(c7)
179
180     return outputs
181
182 # Define hyperparameters
183 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
184
185 # Initialize model, optimizer, and loss function
186 model = UNet(classesNum).to(device)
187 optimizer = optim.Adam(model.parameters(), lr=1e-3)
188 # criterion = nn.CrossEntropyLoss()
189
190 def dice_loss(pred, target, smooth=1e-6):
191     pred = torch.softmax(pred, dim=1)
192     target = torch.nn.functional.one_hot(target, num_classes=pred.shape[1]).permute(0, 3, 1, 2)
193     intersection = (pred * target).sum(dim=(2, 3))
194     union = pred.sum(dim=(2, 3)) + target.sum(dim=(2, 3))
195     dice = (2.0 * intersection + smooth) / (union + smooth)
196     return 1 - dice.mean()
197
198 def focal_loss(pred, target, alpha=0.8, gamma=2.0):
199     pred = torch.softmax(pred, dim=1)
200     target = torch.nn.functional.one_hot(target, num_classes=pred.shape[1]).permute(0, 3, 1, 2)
201     logpt = -torch.nn.functional.cross_entropy(pred, target, reduction='none')
202     pt = torch.exp(logpt)
203     focal_loss = -((1 - pt) ** gamma) * logpt
204     return focal_loss.mean()
205
206 # criterion = lambda outputs, masks: nn.CrossEntropyLoss()(outputs, masks) + dice_loss(outputs, masks)
207
208 # criterion = lambda outputs, masks: 0.7 * dice_loss(outputs, masks) + 0.3 * focal_loss(outputs, masks)
209
210 # Evaluation
211 def evaluate(model, test_loader):
212     model.eval()
213     correct = 0
214     total = 0
215     with torch.no_grad():
216         for images, masks in test_loader:
217             images, masks = images.to(device), masks.to(device)
218             outputs = model(images)
219             predictions = torch.argmax(outputs, dim=1)
220             correct += (predictions == masks).sum().item()
221             total += masks.numel()
222
223     accuracy = correct / total
224     print(f"Test Accuracy: {accuracy:.4f}")
225
226
227 class UNetDataset(Dataset):
228     def __init__(self, image_files, mask_files, transform=None, target_transform=None, max_images=100):
229         self.image_files = image_files # Limit to first 100 images
230         self.mask_files = mask_files # Limit to first 100 masks
231         self.transform = transform
232         self.target_transform = target_transform
233
234     def __len__(self):
235         return len(self.image_files)
236
237     def __getitem__(self, idx):
238         img_path = self.image_files[idx]
239         label_path = self.image_files[idx].replace('.jpg', '.png').replace('color', 'label')
240         image = Image.open(img_path).convert('RGB')
241         label = Image.open(label_path).convert('L') # Load label as grayscale
242
243         if self.transform:
244             image = self.transform(image)
245
246         label = label.resize(imgResize, Image.NEAREST)
247
248         # Convert label to tensor
249         label = torch.tensor(np.array(label), dtype=torch.long) # Shape: (H, W)
250
251         # Map pixel values to class indices
252         label = torch.where(label == 38, 1, label)
253         label = torch.where(label == 75, 2, label)
254         label = torch.where(label == 255, 3, label)
255         label = torch.where(label == 0, 0, label) # Ensure 0 stays as class 0
256
257         return image, label # No one-hot encoding
258
259 # Define transformations
260 image_transform = transforms.Compose([
261     transforms.Resize(imgResize),
262     transforms.ToTensor()
263 ])
264
265 # Load dataset with first 100 images

```

```

266 train_dataset = UNetDataset(train_images, train_masks, transform=image_transform, target_transform=None, max_images
267     =100)
268 train_loader = DataLoader(train_dataset, batch_size=batchSize, shuffle=True)
269 val_dataset = UNetDataset(val_images, val_masks, transform=image_transform, target_transform=None, max_images=100)
270 val_loader = DataLoader(val_dataset, batch_size=batchSize, shuffle=False)
271
272 test_dataset = UNetDataset(test_images, test_masks, transform=image_transform, target_transform=None, max_images=100)
273 test_loader = DataLoader(test_dataset, batch_size=batchSize, shuffle=False)
274
275 # Define the training function
276 def train_unet(model, train_loader, val_loader, epochs, model_save_path="/home/s2677266/CVis/Data/Output/
277     Models_DICEONLY/", device="cuda" if torch.cuda.is_available() else "cpu"):
278     model.to(device)
279
280     # criterion = lambda outputs, masks: 0.7 * dice_loss(outputs, masks) + 0.3 * focal_loss(outputs, masks)
281     # criterion = lambda outputs, masks: nn.CrossEntropyLoss()(outputs, masks) + dice_loss(outputs, masks)
282     criterion = lambda outputs, masks: dice_loss(outputs, masks)
283
284     # criterion = nn.CrossEntropyLoss()
285     optimizer = optim.Adam(model.parameters(), lr=0.001)
286
287     for epoch in range(epochs):
288         model.train()
289         epoch_loss = 0
290         for images, masks in tqdm(train_loader, desc=f"Epoch {epoch+1}/{epochs}"):
291             images, masks = images.to(device), masks.long().squeeze(1).to(device) # Ensure correct shape
292
293             optimizer.zero_grad()
294             outputs = model(images)
295             loss = criterion(outputs, masks)
296             loss.backward()
297             optimizer.step()
298             epoch_loss += loss.item()
299
300         print(f"Epoch {epoch+1}/{epochs}, Training Loss: {epoch_loss/len(train_loader):.4f}")
301
302     # Validation
303     model.eval()
304     val_loss = 0
305     with torch.no_grad():
306         for images, masks in val_loader:
307             images, masks = images.to(device), masks.long().squeeze(1).to(device)
308             outputs = model(images)
309             loss = criterion(outputs, masks)
310             val_loss += loss.item()
311
312     print(f"Epoch {epoch+1}/{epochs}, Validation Loss: {val_loss/len(val_loader):.4f}")
313
314     # Save model after each epoch
315     # epoch_model_path = f"{model_save_path}Better_Final_Outline_unet_model_epoch_{epoch+1}.pth" # with 32, 128
316     epoch_model_path = f"{model_save_path}UNET_DiceOnly_EPOCH_{epoch+1}.pth"
317     torch.save(model.state_dict(), epoch_model_path)
318     print(f"Model saved at {epoch_model_path}")
319
320     print("Training complete!")
321
322     # Define the testing function
323     def test_unet(model, test_loader, device="cuda" if torch.cuda.is_available() else "cpu"):
324         model.to(device)
325         model.eval()
326         test_loss = 0
327
328         # criterion = lambda outputs, masks: 0.7 * dice_loss(outputs, masks) + 0.3 * focal_loss(outputs, masks)
329         # criterion = nn.CrossEntropyLoss()
330         criterion = lambda outputs, masks: dice_loss(outputs, masks)
331
332         with torch.no_grad():
333             for images, masks in test_loader:
334                 images, masks = images.to(device), masks.long().squeeze(1).to(device)
335                 outputs = model(images)
336                 loss = criterion(outputs, masks)
337                 test_loss += loss.item()
338
339         print(f"Test Loss: {test_loss/len(test_loader):.4f}")
340
341     # Call the training function
342     train_unet(model, train_loader, val_loader, epochs=epochsNum)
343
344     # Evaluate on the test set
345     test_unet(model, test_loader)
346
347     print('Execution Completed')

```

### A.3.3 Autoencoder.py: Autoencoder Architecture and Training

```

1 # Authors: Sahil Mehl Bawishi (s2677266) and Matteo Spadaccia (s2748897)
2 # Subject: Computer Vision Coursework. Auto-Encoder
3 # Date: 21.03.2025
4

```

```

5 # Global Constants
6 genVISUALS = True # set to False in order to avoid time-consuming visualizations' generation (images are instead
7     # displayed as pre-saved in 'Output/Visuals' folder)
8 dpi = 500          # dpi for .pdf-saved images visualization (with genVISUALS = False)
9 rePREPROC = True   # if True, the input images' resizing and augmentation are run, otherwise the saved outcomes are
10    # used
11 random_seed = 42
12 trainEncoder = True
13
14 # IMPORTS
15 import os
16 from pathlib import Path
17 from pdf2image import convert_from_path # (also install !apt-get install poppler-utils)
18 from IPython.display import display
19 import numpy as np
20 from PIL import Image
21 import seaborn as sns
22 import cv2
23 import tensorflow as tf
24 import albumentations as A
25 from sklearn.model_selection import train_test_split
26 import matplotlib.pyplot as plt
27 import torch
28 import torch.nn as nn
29 import torch.optim as optim
30 import torch.nn.functional as F
31 from torch.utils.data import DataLoader, Dataset
32 import torchvision.transforms as transforms
33 from torch.optim.lr_scheduler import StepLR
34 from tqdm import tqdm
35 import pickle as pkl
36 from torch.cuda.amp import autocast, GradScaler
37
38 if torch.cuda.is_available():
39     print("using GPU")
40 else:
41     print("using CPU")
42
43 # Loading input data
44 input_folder_trainval = 'Data/Input/TrainVal'
45 input_trainval = [f for f in os.listdir(input_folder_trainval+'color') if f.lower().endswith('.jpg', '.jpeg')]
46 input_trainval_labels = [f for f in os.listdir(input_folder_trainval+'label') if f.lower().endswith('.png')]
47
48 input_folder_test = 'Data/Input/Test'
49 input_test = [f for f in os.listdir(input_folder_test+'color') if f.lower().endswith('.jpg', '.jpeg')]
50 input_test_labels = [f for f in os.listdir(input_folder_test+'label') if f.lower().endswith('.png')]
51
52 output_folder = 'Data/Output'
53 output_folder_resized = os.path.join(output_folder, 'Resized')
54 output_folder_augmented = os.path.join(output_folder, 'Augmented')
55
56 output_folder_resized_color = os.path.join(output_folder_resized, 'color')
57 output_folder_resized_label = os.path.join(output_folder_resized, 'label')
58 output_folder_augmented_color = os.path.join(output_folder_augmented, 'color')
59 output_folder_augmented_label = os.path.join(output_folder_augmented, 'label')
60
61 ##### 1. Dataset preprocessing and augmentation
62
63 The images are resized to the dimensions ( $H_{min}, W_{min}$ ), thus to take the Q3 height and width size
64 over all the images in the dataset; the instances will be processed in this format, then the output resized back
65 to the original dimensions...
66
67 Furthermore...(data augmentation)
68 """
69
70 # Printing stats
71 print(f"Input set size:{len(input_trainval)}\n")
72 print("Q3 width and height values (both 500 pixels) were chosen for resizing.")
73
74 ##### a) Resizing """
75 # Resizing images (to Q3 width, Q3 height)
76 print("Using previously resized images and labels (500x500).")
77 imgResize = (500, 500)
78
79 ##### b) Augmenting dataset """
80 print("Using previously augmented data.")
81
82 ##### c) Preparing datasets """
83
84 # Preparing train, valid and test sets
85 validSize = 0.2
86 batchSize = 4
87 imgChannels = 3 ## this was 3
88 inputSize = (imgResize[0], imgResize[1], imgChannels)
89 classesNum = 4 # number of output classes
90 epochsNum = 30 # number of training epochs
91 batchSize = 32
92
93 import os
94 os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "expandable_segments:True"
95 # Define preprocessed size

```

```

96     preprocessedSize = (256, 256) # Desired size for input images and masks
97     inputSize = (256, 256, 3)
98
99     trainval_images = [os.path.join(output_folder_augmented_color, f) for f in os.listdir(output_folder_augmented_color)
100         if f.endswith('.jpg')]
100    trainval_masks = [os.path.join(output_folder_augmented_label, f) for f in os.listdir(output_folder_augmented_label) if
101        f.endswith('.png')]
101
102    train_images, val_images, train_masks, val_masks = train_test_split(trainval_images, trainval_masks, test_size=
103        validSize, random_state=random_seed)
103
104    test_images = [os.path.join(input_folder_test, 'color', f) for f in os.listdir(os.path.join(input_folder_test, 'color'))
105        if f.endswith('.jpg')]
105    test_masks = [os.path.join(input_folder_test, 'label', f) for f in os.listdir(os.path.join(input_folder_test, 'label'))
106        if f.endswith('.png')]
106
107
108    print(f"Train set size: {len(train_images)} ({((1-validSize)*100)%})")
109    print(f"Valid set size: {len(val_images)} ({(validSize)*100}%})")
110    print(f"Test set size: {len(test_images)}")
111    print(f"\nInput dimension: {preprocessedSize+imgChannels,})") # Adjusted for the preprocessed size
112    print(f"Batch size: {batchSize}")
113
114
115
116
117
118 ##### DEFINING THE ENCODER PART OF THE AUTO-ENCODER (PRE TRAINING) #####
119
120    scaler = GradScaler()
121    # Define the Autoencoder
122    class Autoencoder(nn.Module):
123        def __init__(self):
124            super(Autoencoder, self).__init__()
125
126            # Encoder
127            self.encoder = nn.Sequential(
128                nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
129                nn.LeakyReLU(0.2),
130                nn.MaxPool2d(kernel_size=2, stride=2), # /2 size
131                nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
132                nn.LeakyReLU(0.2),
133                nn.MaxPool2d(kernel_size=2, stride=2), # /2 size
134                nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
135                nn.LeakyReLU(0.2),
136                nn.MaxPool2d(kernel_size=2, stride=2), # /2 size
137                nn.Conv2d(256, 128, kernel_size=3, stride=1, padding=1),
138                nn.LeakyReLU(0.2),
139            )
140
141            # Decoder
142            self.decoder = nn.Sequential(
143                nn.ConvTranspose2d(128, 256, kernel_size=3, stride=2, padding=1, output_padding=1), # x2 size
144                nn.LeakyReLU(0.2),
145                nn.ConvTranspose2d(256, 128, kernel_size=3, stride=2, padding=1, output_padding=1), # x2 size
146                nn.LeakyReLU(0.2),
147                nn.ConvTranspose2d(128, 64, kernel_size=3, stride=2, padding=1, output_padding=1), # x2 size
148                nn.LeakyReLU(0.2),
149                nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
150                nn.LeakyReLU(0.2),
151                nn.Conv2d(64, 3, kernel_size=3, stride=1, padding=1),
152                nn.Sigmoid()
153            )
154
155        def forward(self, x):
156            x = self.encoder(x)
157            x = self.decoder(x)
158            return x
159
160
161    # Below is the code to train the autoencoder
162
163    if trainEncoder == True:
164        class ImageDataset(Dataset):
165            def __init__(self, image_files, transform=None, max_images=100):
166                self.image_files = image_files
167                self.transform = transform
168
169            def __len__(self):
170                return len(self.image_files)
171
172            def __getitem__(self, idx):
173                img_path = self.image_files[idx]
174                image = Image.open(img_path).convert('RGB')
175                if self.transform:
176                    image = self.transform(image)
177                return image
178
179            # Transformations
180            transform = transforms.Compose([
181                transforms.Resize((256, 256)),
182                transforms.ToTensor(),
183                # transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]) # Normalize to [-1,1]
184            ])

```

```

186
187     # Load dataset with first 100 images
188     dataset = ImageDataset(image_files=train_images, transform=transform, max_images=100)
189     dataloader = DataLoader(dataset, batch_size=batchSize, shuffle=True)
190
191     # Device configuration
192     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
193     print(device)
194
195     # Initialize model, loss, and optimizer
196     autoencoder = Autoencoder().to(device)
197     criterion = nn.MSELoss()
198     optimizer = optim.Adam(autoencoder.parameters(), lr=1e-3)
199
200     def weights_init(m):
201         if isinstance(m, nn.Conv2d) or isinstance(m, nn.ConvTranspose2d):
202             nn.init.xavier_uniform_(m.weight)
203             if m.bias is not None:
204                 nn.init.zeros_(m.bias)
205
206     autoencoder.apply(weights_init)
207
208     # Training the Autoencoder
209     num_epochs = 100
210     for epoch in range(num_epochs):
211         print(f"Epoch [{epoch+1}/{num_epochs}] Running:")
212         torch.cuda.empty_cache()
213         epoch_loss = 0.0
214         for batch in dataloader:
215             batch = batch.to(device)
216             optimizer.zero_grad()
217
218             with autocast():
219                 outputs = autoencoder(batch)
220                 loss = criterion(outputs, batch)
221                 epoch_loss += loss.item()
222
223                 scaler.scale(loss).backward()
224                 scaler.step(optimizer)
225                 scaler.update()
226
227             print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss/len(dataloader)}")
228             torch.save(autoencoder.state_dict(), f'Data/Output/Models2/updated_complex_working_autoencoder_epoch_{epoch+1}.pth')
229
230
231 ##### NOW THAT THE ENCODER IS PRE-TRAINED, WE CAN CREATE A GOOD SEGMENTER #####
232
233     # Residual Block
234     class ResidualBlock(nn.Module):
235         def __init__(self, in_channels, out_channels):
236             super(ResidualBlock, self).__init__()
237             self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)
238             self.bn1 = nn.BatchNorm2d(out_channels)
239             self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)
240             self.bn2 = nn.BatchNorm2d(out_channels)
241             self.relu = nn.ReLU(inplace=True)
242
243         def forward(self, x):
244             residual = x
245             x = self.relu(self.bn1(self.conv1(x)))
246             x = self.bn2(self.conv2(x))
247             x += residual # skip connection
248             return self.relu(x)
249
250     # Enhanced Segmentation Decoder
251     class SegmentationDecoder(nn.Module):
252         def __init__(self, encoder):
253             super(SegmentationDecoder, self).__init__()
254
255             for param in encoder.parameters():
256                 param.requires_grad = False # set to False to FREEZE THE ENCODER
257
258             self.encoder = encoder
259
260             self.decoder = nn.Sequential(
261                 nn.ConvTranspose2d(128, 128, kernel_size=3, stride=2, padding=1, output_padding=1), # x2 size
262                 ResidualBlock(128, 128),
263                 nn.ConvTranspose2d(128, 64, kernel_size=3, stride=2, padding=1, output_padding=1), # x2 size
264                 ResidualBlock(64, 64),
265                 nn.ConvTranspose2d(64, 32, kernel_size=3, stride=2, padding=1, output_padding=1), # x2 size
266                 ResidualBlock(32, 32),
267                 nn.Conv2d(32, 32, kernel_size=3, padding=1),
268                 nn.BatchNorm2d(32),
269                 nn.ReLU(inplace=True),
270                 nn.Conv2d(32, classesNum, kernel_size=1)
271             )
272
273         def forward(self, x):
274             x = self.encoder(x)
275             x = self.decoder(x)
276             return x
277
278
279     # Custom Dataset for loading images and labels

```

```

280 class SegmentationDataset(Dataset):
281     def __init__(self, image_files, label_files, transform=None):
282         self.image_files = image_files
283         self.label_files = label_files
284         self.transform = transform
285
286     def __len__(self):
287         return len(self.image_files)
288
289     def __getitem__(self, idx):
290         img_path = self.image_files[idx]
291         label_path = self.image_files[idx].replace('.jpg', '.png').replace('color', 'label')
292         image = Image.open(img_path).convert('RGB')
293         label = Image.open(label_path).convert('L')
294
295         if self.transform:
296             image = self.transform(image)
297
298         label = label.resize((256, 256), Image.NEAREST)
299
300         # Convert label to tensor
301         label = torch.tensor(np.array(label), dtype=torch.long)
302
303         # Map pixel values to class indices
304         label = torch.where(label == 38, 1, label)
305         label = torch.where(label == 75, 2, label)
306         label = torch.where(label == 255, 3, label)
307         label = torch.where(label == 0, 0, label)
308
309         return image, label
310
311     # Transformations
312     transform = transforms.Compose([
313         transforms.Resize((256, 256)),
314         transforms.ToTensor(),
315     ])
316
317
318     # Load segmentation dataset with first 100 images
319     segmentation_dataset = SegmentationDataset(
320         image_files=train_images,
321         label_files=train_masks,
322         transform=transform
323     )
324
325     val_segmentation_dataset = SegmentationDataset(
326         image_files=val_images,
327         label_files=val_masks,
328         transform=transform
329     )
330
331     segmentation_dataloader = DataLoader(segmentation_dataset, batch_size=batchSize, shuffle=True)
332     val_segmentation_dataloader = DataLoader(val_segmentation_dataset, batch_size=batchSize, shuffle=True)
333
334     # Load the pre-trained autoencoder
335     autoencoder = Autoencoder()
336     autoencoder.load_state_dict(torch.load('Data/Output/Models2/updated_complex_working_autoencoder_epoch_95.pth'))
337
338     print("Loaded Autoencoder: Data/Output/Models2/updated_complex_working_autoencoder_epoch_95")
339
340     print("Saving decoder at Data/Output/Models3/Not_Frozen_Val_Segmentation_95_Decoder_DICE_epoch_")
341     print("Frozen")
342     print("segmentation_criterion = lambda outputs, masks: nn.CrossEntropyLoss()(outputs, masks) + dice_loss(outputs, masks)")
343
344     # Initialize the segmentation model
345     # segmentation_model = SegmentationDecoder(autoencoder.encoder)
346
347     # Define loss and optimizer for segmentation
348     segmentation_model = SegmentationDecoder(autoencoder.encoder)
349     # class_weights = torch.tensor([1.0, 1.0, 1.0, 1.0]) # Adjust these values as needed
350     # segmentation_criterion = nn.CrossEntropyLoss()(weight=class_weights)
351
352     def dice_loss(pred, target, smooth=1e-6):
353         pred = torch.softmax(pred, dim=1)
354         target = torch.nn.functional.one_hot(target, num_classes=pred.shape[1]).permute(0, 3, 1, 2)
355         intersection = (pred * target).sum(dim=(2, 3))
356         union = pred.sum(dim=(2, 3)) + target.sum(dim=(2, 3))
357         dice = (2.0 * intersection + smooth) / (union + smooth)
358         return 1 - dice.mean()
359
360     # segmentation_criterion = lambda outputs, masks: nn.CrossEntropyLoss()(outputs, masks) + dice_loss(outputs, masks)
361     # segmentation_criterion = nn.CrossEntropyLoss()
362     # Example of class weights (higher weight for less frequent classes)
363     segmentation_optimizer = optim.Adam(segmentation_model.parameters(), lr=1e-3, weight_decay=1e-4)
364
365     segmentation_criterion = lambda outputs, masks: dice_loss(outputs, masks)
366
367     # Training the Segmentation Model
368
369     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
370     segmentation_model.to(device)
371     # scheduler = torch.optim.lr_scheduler.StepLR(segmentation_optimizer, step_size=3, gamma=0.5)
372     # scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(segmentation_optimizer, T_max=10, eta_min=1e-5)
373

```

```

374     scaler = torch.cuda.amp.GradScaler()
375
376     num_epochs = 100
377     for epoch in tqdm(range(num_epochs), desc="Training Progress"):
378         print(f'Epoch [{epoch+1}/{num_epochs}] Running:')
379         segmentation_model.train()
380         epoch_loss = 0
381
382         for images, labels in segmentation_dataloader:
383             images, labels = images.to(device), labels.to(device)
384
385             segmentation_optimizer.zero_grad()
386
387             with autocast():
388                 outputs = segmentation_model(images)
389                 loss = segmentation_criterion(outputs, labels)
390                 epoch_loss += loss.item()
391             scaler.scale(loss).backward()
392             scaler.step(segmentation_optimizer)
393             scaler.update()
394
395         print(f'Epoch [{epoch+1}/{num_epochs}], Training Loss: {epoch_loss/len(segmentation_dataloader)}')
396
397         segmentation_model.eval()
398         val_loss = 0
399         with torch.no_grad():
400             for images, labels in val_segmentation_dataloader:
401                 images, labels = images.to(device), labels.to(device)
402                 outputs = segmentation_model(images)
403                 loss = segmentation_criterion(outputs, labels)
404                 val_loss += loss.item()
405
406         print(f'Epoch [{epoch+1}/{num_epochs}], Validation Loss: {val_loss/len(val_segmentation_dataloader)}')
407         print(f'Saved Model at Data/Output/Models_DICEONLY/Frozen_Val_Segmentation_95_Decoder_DiceOnly_epoch{epoch+1}.pth')
408         torch.save(segmentation_model.state_dict(), f'Data/Output/Models_DICEONLY/Frozen_Val_Segmentation_95_Decoder_DiceOnly_epoch{epoch+1}.pth')

```

### A.3.4 Clip.py: Clip Architecture and Training

```

1 # Authors: Sahil Memul Bavishi (s2677266) and Matteo Spadaccia (s2748897)
2 # Subject: Computer Vision Coursework. Clip
3 # Date: 21.03.2025
4
5 """
6     0. Preliminary code"""
7
8 # Setting code-behaviour variables
9 extractFeatures = True # if True, the CLIP features are extracted, otherwise the saved ones are used
10
11 preprocessedSize = (256, 256) # Desired size to read input images and masks in
12 imgChannels = 3
13 classesNum = 4
14
15 validSize = 0.2
16 epochsNum = 100
17 batchSize = 16
18 learningRate = 1e-3
19
20 random_seed = 42
21
22 # Importing useful libraries
23 import os
24 import numpy as np
25 from PIL import Image
26 from sklearn.model_selection import train_test_split
27 import torch
28 import torch.nn as nn
29 import torch.optim as optim
30 from torch.utils.data import DataLoader, Dataset
31 from torchvision import transforms
32 from tqdm import tqdm
33 import open_clip
34
35 # Loading input data
36 input_folder_trainval = 'Data/Input/TrainVal'
37 input_trainval = [f for f in os.listdir(input_folder_trainval+'color') if f.lower().endswith('.jpg', '.jpeg')]
38 input_trainval_labels = [f for f in os.listdir(input_folder_trainval+'label') if f.lower().endswith('.png')]
39
40 input_folder_test = 'Data/Input/Test'
41 input_test = [f for f in os.listdir(input_folder_test+'color') if f.lower().endswith('.jpg', '.jpeg')]
42 input_test_labels = [f for f in os.listdir(input_folder_test+'label') if f.lower().endswith('.png')]
43
44 output_folder = 'Data/Output'
45 output_folder_resized = os.path.join(output_folder, 'Resized')
46 output_folder_augmented = os.path.join(output_folder, 'Augmented')
47
48 output_folder_resized_color = os.path.join(output_folder_resized, 'color')
49 output_folder_resized_label = os.path.join(output_folder_resized, 'label')
50 output_folder_augmented_color = os.path.join(output_folder_augmented, 'color')
51 output_folder_augmented_label = os.path.join(output_folder_augmented, 'label')

```

```

52 # Preparing train, valid and test sets
53 os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "expandable_segments=True"
54 trainval_images = [os.path.join(output_folder_augmented_color, f) for f in os.listdir(output_folder_augmented_color)
55     if f.endswith('.jpg')]
56 trainval_masks = [os.path.join(output_folder_augmented_label, f) for f in os.listdir(output_folder_augmented_label) if
57     f.endswith('.png')]
58 train_images, val_images, train_masks, val_masks = train_test_split(trainval_images, trainval_masks, test_size=
59     validSize, random_state=random_seed)
60 test_images = [os.path.join(input_folder_test, 'color', f) for f in os.listdir(os.path.join(input_folder_test, 'color'))
61     if f.endswith('.jpg')]
62 test_masks = [os.path.join(input_folder_test, 'label', f) for f in os.listdir(os.path.join(input_folder_test, 'label'))
63     if f.endswith('.png')]
64
65 print(f"Train set size: {len(train_images)} ({((1-validSize)*100)%})")
66 print(f"Valid set size: {len(val_images)} ({(validSize)*100}%})")
67 print(f"Test set size: {len(test_images)}")
68 print(f"\nInput dimension: {preprocessedSize*imgChannels}")
69 print(f"Batch size: {batchSize}")
70
71 """1. CLIP features extraction"""
72
73 # Loading CLIP model
74 device = "cuda" if torch.cuda.is_available() else "cpu"
75 model, preprocess_train, preprocess_val = open_clip.create_model_and_transforms("ViT-B-32", pretrained="openai")
76 model = model.to(device)
77 tokenizer = open_clip.get_tokenizer("ViT-B-32")
78
79 # Preparing features saving directory
80 clip_features_dir = "Data/Output/Models/clipTrainFeatures"
81 os.makedirs(clip_features_dir, exist_ok=True)
82
83 if extractFeatures:
84     # Extract and save CLIP features
85     def extract_clip_features(image_path):
86         image = Image.open(image_path).convert("RGB")
87         image_tensor = preprocess_val(image).unsqueeze(0).to(device) # Use open_clip's preprocessing
88         with torch.no_grad():
89             features = model.encode_image(image_tensor) # Extract image features
90             features = features / features.norm(p=2, dim=-1, keepdim=True) # Normalize
91         return features.cpu()
92
93 clip_features_trainval = {}
94 print("Extracting Clip Features")
95 for image_name in trainval_images:
96     image_path = image_name
97     clip_features = extract_clip_features(image_path)
98
99     # Save each feature tensor separately
100    feature_path = os.path.join(clip_features_dir, f"{image_name.split('/')[-1].split('.')[0]}.pt")
101    torch.save(clip_features, feature_path)
102
103    clip_features_trainval[image_name] = feature_path
104
105 """2. CLIP-based segmentation"""
106
107 # Loading CLIP features
108 clip_features_trainval = {
109     image_name: torch.load(os.path.join(clip_features_dir, f"{image_name.split('/')[-1].split('.')[0]}.pt"))
110     for image_name in trainval_images
111 }
112 print(f"Loaded {len(clip_features_trainval)} CLIP feature tensors.")
113 print("Saving model to this location: Data/Output/Models2/CLIP_CROSS2_epoch_")
114
115 class SegmentationDataset(Dataset):
116     def __init__(self, image_files, label_files, clip_features_dict, transform=None):
117         self.image_files = image_files
118         self.label_files = label_files
119         self.clip_features_dict = clip_features_dict
120         self.transform = transform
121
122     def __len__(self):
123         return len(self.image_files)
124
125     def __getitem__(self, idx):
126         img_path = self.image_files[idx]
127         label_path = img_path.replace('.jpg', '.png').replace('color', 'label')
128
129         # Loading image
130         image = Image.open(img_path).convert("RGB")
131         if self.transform:
132             image = self.transform(image)
133         else:
134             image = transforms.ToTensor()(image)
135
136         # Loading CLIP feature tensor
137         image_name = img_path
138         clip_feature = self.clip_features_dict[image_name]
139
140         # Loading and processing label
141         label = Image.open(label_path).convert('L')

```

```

142     label = label.resize(preprocessedSize, Image.NEAREST)
143     label = torch.tensor(np.array(label), dtype=torch.long)
144
145     # Converting label classes
146     label = torch.where(label == 38, 1, label)
147     label = torch.where(label == 75, 2, label)
148     label = torch.where(label == 255, 3, label)
149     label = torch.where(label == 0, 0, label)
150
151     return image, clip_feature, label
152
153     # Defining Transformations
154     transform = transforms.Compose([
155         transforms.Resize(preprocessedSize, interpolation=Image.NEAREST),
156         transforms.ToTensor()
157     ])
158
159     # Loading Dataset & DataLoader
160     segmentation_dataset = SegmentationDataset(
161         image_files=train_images,
162         label_files=train_masks,
163         clip_features_dict=clip_features_trainval,
164         transform=None
165     )
166
167     segmentation_dataloader = DataLoader(segmentation_dataset, batch_size=batchSize, shuffle=True)
168
169     val_segmentation_dataset = SegmentationDataset(
170         image_files=val_images,
171         label_files=val_masks,
172         clip_features_dict=clip_features_trainval,
173         transform=None
174     )
175
176     val_segmentation_dataloader = DataLoader(val_segmentation_dataset, batch_size=batchSize, shuffle=True)
177
178     class SegmentationDecoder(nn.Module):
179         def __init__(self, clip_feature_dim=512, num_classes=classesNum):
180             super(SegmentationDecoder, self).__init__()
181
182             # CLIP features
183             self.clip_fc = nn.Linear(clip_feature_dim, 256)
184
185             # CNN Feature Extractor
186             self.encoder = nn.Sequential(
187                 nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
188                 nn.ReLU(),
189                 nn.MaxPool2d(2),
190                 nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
191                 nn.ReLU(),
192             )
193
194             # Decoder
195             self.decoder = nn.Sequential(
196                 nn.ConvTranspose2d(384, 64, kernel_size=3, stride=1, padding=1),
197                 nn.ReLU(),
198                 nn.Upsample(size=(256, 256), mode='bilinear', align_corners=True),
199                 nn.ConvTranspose2d(64, num_classes, kernel_size=3, stride=1, padding=1),
200             )
201
202         def forward(self, image, clip_features):
203             cnn_features = self.encoder(image)
204             clip_features = self.clip_fc(clip_features).view(clip_features.shape[0], 256, 1, 1)
205             clip_features = clip_features.expand(-1, -1, cnn_features.shape[2], cnn_features.shape[3])
206             fusion = torch.cat([cnn_features, clip_features], dim=1)
207             segmentation_output = self.decoder(fusion)
208             return segmentation_output
209
210
211         def dice_loss(pred, target, smooth=1e-6):
212             pred = torch.softmax(pred, dim=1)
213             target = torch.nn.functional.one_hot(target, num_classes=pred.shape[1]).permute(0, 3, 1, 2)
214             intersection = (pred * target).sum(dim=(2, 3))
215             union = pred.sum(dim=(2, 3)) + target.sum(dim=(2, 3))
216             dice = (2.0 * intersection + smooth) / (union + smooth)
217             return 1 - dice.mean()
218
219     # Initialize Model, Loss, Optimizer
220     segmentation_model = SegmentationDecoder().to(device)
221     # segmentation_criterion = nn.CrossEntropyLoss()
222     segmentation_criterion = lambda outputs, masks: dice_loss(outputs, masks)
223     # segmentation_criterion = lambda outputs, masks: nn.CrossEntropyLoss()(outputs, masks) + dice_loss(outputs, masks)
224
225     print("Loss: segmentation_criterion = nn.CrossEntropyLoss()")
226
227     segmentation_optimizer = optim.Adam(segmentation_model.parameters(), lr=learningRate)
228     scheduler = torch.optim.lr_scheduler.StepLR(segmentation_optimizer, step_size=5, gamma=0.5)
229     scaler = torch.cuda.amp.GradScaler()
230
231     # Training model
232     for epoch in tqdm(range(epochsNum), desc="Training Progress"):
233         segmentation_model.train()
234         running_loss = 0.0
235
236         print(f"Epoch [{epoch+1}/{epochsNum}] Running:")

```

```

237
238     for images, clip_features, labels in segmentation_dataloader:
239         images, clip_features, labels = images.to(device), clip_features.to(device), labels.to(device)
240
241         segmentation_optimizer.zero_grad()
242
243         # Mixed precision training
244         with torch.cuda.amp.autocast():
245             outputs = segmentation_model(images, clip_features)
246             loss = segmentation_criterion(outputs, labels)
247
248         # Scaling loss and backpropagating
249         scaler.scale(loss).backward()
250         scaler.step(segmentation_optimizer)
251         scaler.update()
252
253         running_loss += loss.item()
254
255     segmentation_model.eval()
256     val_loss = 0
257     with torch.no_grad():
258         for images, clip_features, labels in val_segmentation_dataloader:
259             images, clip_features, labels = images.to(device), clip_features.to(device), labels.to(device)
260             outputs = segmentation_model(images, clip_features)
261             loss = segmentation_criterion(outputs, labels)
262             val_loss += loss.item()
263
264     print(f'Epoch [{epoch+1}/{epochsNum}], Validation Loss: {val_loss/len(val_segmentation_dataloader)}')
265
266     scheduler.step()
267
268     avg_loss = running_loss / len(segmentation_dataloader)
269
270     print(f'Epoch [{epoch+1}/{epochsNum}], Training Loss: {avg_loss:.4f}')
271     print(f'Saved here: Data/Output/Models_DICEONLY/CLIP_DiceOnly_epoch_{epoch+1}.pth')
272     torch.save(segmentation_model.state_dict(), f'Data/Output/Models_DICEONLY/CLIP_DiceOnly_epoch_{epoch+1}.pth')
273
274 print("Training complete!")

```

### A.3.5 annotation.py: Annotating the Data for Prompted-Clip Training

```

1 import cv2
2 import os
3 import pickle
4
5 # Define paths
6 image_dir = "Data/Output/Resized/color"
7 annotations_dir = "Data/Output/Resized/annotations"
8
9 # Ensure the annotations directory exists
10 os.makedirs(annotations_dir, exist_ok=True)
11
12 # Specify the range of images
13 start = 1
14 end = 3680
15
16 # Generate the output file name for coordinates
17 output_file = os.path.join(annotations_dir, f"annotations_{start}_to_{end}.pkl")
18
19 # Get all images in the directory and sort them
20 image_files = sorted([f for f in os.listdir(image_dir) if f.endswith(("jpg", ".png"))])
21 image_files = image_files[start - 1:end]
22
23 # Dictionary to store annotations
24 annotations = {}
25 clicked = False # Flag to track click status
26 point = None # Store clicked point
27
28 # Mouse click event callback function
29 def click_event(event, x, y, flags, param):
30     global clicked, point
31     if event == cv2.EVENT_LBUTTONDOWN:
32         point = (x, y)
33         clicked = True # Set flag
34
35 # Loop through images
36 for image_name in image_files:
37     current_image = image_name
38     image_path = os.path.join(image_dir, image_name)
39
40     img = cv2.imread(image_path)
41     if img is None:
42         print(f"Error loading {image_name}, skipping...")
43         continue
44
45     img = cv2.resize(img, (500, 500))
46     clicked = False
47     point = None
48
49     # Display image and wait for a click
50     cv2.imshow("Annotate Nose - Click on the nose", img)

```

```

51     cv2.setMouseCallback("Annotate_Nose---Click on the nose", click_event)
52
53     while not clicked:
54         cv2.waitKey(1)
55
56     cv2.destroyAllWindows()
57
58     if point:
59         annotations[current_image] = point
60
61         # Draw point on image
62         cv2.circle(img, point, 5, (0, 0, 255), -1) # Red dot
63
64         # Save annotated image
65         annotated_path = os.path.join(annotations_dir, f"annotated_{image_name}")
66         cv2.imwrite(annotated_path, img)
67         print(f"Saved annotated image: {annotated_path}")
68
69     # Save annotations to a .pkl file
70     with open(output_file, "wb") as f:
71         pickle.dump(annotations, f)
72
73     print(f"Annotations saved to {output_file}")
74     cv2.destroyAllWindows()

```

### A.3.6 buildHeatmaps.py: Building Heatmaps for Prompted-Clip Training

```

1 import os
2 import pickle
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 # Parameters
7 merged_annotations_file = "Data/Output/test_annotations.pkl"
8 output_dir = "Data/Input/Test/heatmaps/"
9 os.makedirs(output_dir, exist_ok=True)
10
11 def generate_gaussian_heatmap(size, center, intensity=1.0, sigma=100):
12     x0, y0 = center
13     x = np.arange(size[0])
14     y = np.arange(size[1])
15     X, Y = np.meshgrid(x, y)
16     heatmap = intensity * np.exp(-((X - x0)**2 + (Y - y0)**2) / (2 * sigma**2))
17     return heatmap
18
19 # Load merged annotations
20 with open(merged_annotations_file, "rb") as f:
21     annotations = pickle.load(f)
22
23 # Generate heatmaps
24 for img_name, (x, y) in annotations.items():
25     heatmap = generate_gaussian_heatmap((500, 500), (x, y), intensity=1.0, sigma=50)
26     plt.imsave(os.path.join(output_dir, f"{img_name.split('.')[0]}_heatmap.png"), heatmap, cmap='gray')
27
28 print(f"Generated {len(annotations)} heatmaps in {output_dir}")

```

### A.3.7 Prompt-Clip.py: Prompt-Clip Architecture and Training

```

1 # Authors: Sahil Mehul Bavishi (s2677266) and Matteo Spadaccia (s2748897)
2 # Subject: Computer Vision Coursework. Clip
3 # Date: 21.03.2025
4
5 """
6     0. Preliminary code"""
7
8 # Setting code-behaviour variables
9 extractFeatures = True # if True, the CLIP features are extracted, otherwise the saved ones are used
10
11 preprocessedSize = (256, 256) # Desired size to read input images and masks in
12 imgChannels = 3
13 classesNum = 4
14
15 validSize = 0.2
16 epochsNum = 100
17 batchSize = 16
18 learningRate = 1e-3
19
20 random_seed = 42
21
22 # Importing useful libraries
23 import os
24 import numpy as np
25 from PIL import Image
26 from sklearn.model_selection import train_test_split
27 import torch
28 import torch.nn as nn

```

```

29 import torch.optim as optim
30 from torch.utils.data import DataLoader, Dataset
31 from torchvision import transforms
32 from tqdm import tqdm
33 import open_clip
34
35 # Loading input data
36 input_folder_trainval = 'Data/Input/TrainVal'
37 input_trainval = [f for f in os.listdir(input_folder_trainval+'color') if f.lower().endswith('.jpg', '.jpeg')]
38 input_trainval_labels = [f for f in os.listdir(input_folder_trainval+'label') if f.lower().endswith('.png')]
39
40 input_folder_test = 'Data/Input/Test'
41 input_test = [f for f in os.listdir(input_folder_test+'color') if f.lower().endswith('.jpg', '.jpeg')]
42 input_test_labels = [f for f in os.listdir(input_folder_test+'label') if f.lower().endswith('.png')]
43
44 output_folder = 'Data/Output'
45 output_folder_resized = os.path.join(output_folder, 'Resized')
46 output_folder_augmented = os.path.join(output_folder, 'Augmented')
47
48 output_folder_resized_color = os.path.join(output_folder_resized, 'color')
49 output_folder_resized_label = os.path.join(output_folder_resized, 'label')
50
51 output_folder_augmented_color = os.path.join(output_folder_augmented, 'color')
52 output_folder_augmented_label = os.path.join(output_folder_augmented, 'label')
53
54 # Preparing train, valid and test sets
55 os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "expandable_segments=True"
56 trainval_images = [os.path.join(output_folder_augmented_color, f) for f in os.listdir(output_folder_augmented_color)
57     if f.endswith('.jpg')]
58 trainval_masks = [os.path.join(output_folder_augmented_label, f) for f in os.listdir(output_folder_augmented_label) if
59     f.endswith('.png')]
60 train_images, val_images, train_masks, val_masks = train_test_split(trainval_images, trainval_masks, test_size=
61     validSize, random_state=random_seed)
62 test_images = [os.path.join(input_folder_test, 'color', f) for f in os.listdir(os.path.join(input_folder_test, 'color'))
63     if f.endswith('.jpg')]
64 test_masks = [os.path.join(input_folder_test, 'label', f) for f in os.listdir(os.path.join(input_folder_test, 'label'))
65     if f.endswith('.png')]
66
67 print(f"Train set size: {len(train_images)} ({((1-validSize)*100)%})")
68 print(f"Valid set size: {len(val_images)} ({(validSize)*100}%)")
69 print(f"\nTest set size: {len(test_images)}")
70 print(f"\nInput dimension: {preprocessedSize+(imgChannels,)}")
71 print(f"Batch size: {batchSize}")
72
73 """1. CLIP features extraction"""
74
75 # Loading CLIP model
76 device = "cuda" if torch.cuda.is_available() else "cpu"
77 model, preprocess_train, preprocess_val = open_clip.create_model_and_transforms("ViT-B-32", pretrained="openai")
78 model = model.to(device)
79 tokenizer = open_clip.get_tokenizer("ViT-B-32")
80
81 # Preparing features saving directory
82 clip_features_dir = "Data/Output/Models/ResOnlyClipTrainFeatures3"
83 os.makedirs(clip_features_dir, exist_ok=True)
84
85 if extractFeatures:
86     # Extract and save CLIP features
87     def extract_clip_features(image_path):
88         image = Image.open(image_path).convert("RGB")
89         image_tensor = preprocess_val(image).unsqueeze(0).to(device) # Use open_clip's preprocessing
90         with torch.no_grad():
91             features = model.encode_image(image_tensor) # Extract image features
92             features = features / features.norm(p=2, dim=-1, keepdim=True) # Normalize
93             return features.cpu()
94
95     clip_features_trainval = {}
96     print("Extracting Clip Features")
97     for image_name in trainval_images:
98         image_path = image_name
99         clip_features = extract_clip_features(image_path)
100
101        # Save each feature tensor separately
102        feature_path = os.path.join(clip_features_dir, f"{image_name.split('/')[-1].split('.')[0]}.pt")
103        torch.save(clip_features, feature_path)
104
105    clip_features_trainval[image_name] = feature_path
106
107 """2. CLIP-based segmentation"""
108
109 # Loading CLIP features
110 clip_features_trainval = {
111     image_name: torch.load(os.path.join(clip_features_dir, f"{image_name.split('/')[-1].split('.')[0]}.pt"))
112     for image_name in trainval_images
113 }
114 print(f"Loaded {len(clip_features_trainval)} CLIP feature tensors.")
115
116 class SegmentationDataset(Dataset):
117     def __init__(self, image_files, label_files, clip_features_dict, transform=None):
118         self.image_files = image_files

```

```

119     self.label_files = label_files
120     self.clip_features_dict = clip_features_dict
121     self.transform = transform
122
123     def __len__(self):
124         return len(self.image_files)
125
126     def __getitem__(self, idx):
127         img_path = self.image_files[idx]
128         label_path = img_path.replace('.jpg', '.png').replace('color', 'label')
129
130         # Compute heatmap path: assuming heatmaps are stored in a folder "heatmaps"
131         # located in the same parent folder as "color"
132         base_name = os.path.splitext(os.path.basename(img_path))[0]
133         color_dir = os.path.dirname(img_path)
134         parent_dir = os.path.dirname(color_dir)
135         heatmap_dir = os.path.join(parent_dir, "heatmaps")
136         heatmap_path = os.path.join(heatmap_dir, base_name + "_heatmap.png")
137
138         # Loading image
139         image = Image.open(img_path).convert("RGB")
140         if self.transform:
141             image = self.transform(image)
142         else:
143             image = transforms.ToTensor()(image)
144
145         # Loading heatmap and converting to 1-channel tensor
146         heatmap = Image.open(heatmap_path).convert('L')
147         heatmap = transforms.ToTensor()(heatmap)
148
149         # Concatenating image and heatmap to form a 4-channel input
150         image = torch.cat([image, heatmap], dim=0)
151
152         # Loading CLIP feature tensor
153         image_name = img_path
154         clip_feature = self.clip_features_dict[image_name] # Shape: (512,)
155
156         # Loading and processing label
157         label = Image.open(label_path).convert('L')
158         label = label.resize(preprocessedSize, Image.NEAREST)
159         label = torch.tensor(np.array(label), dtype=torch.long)
160
161         # Converting label classes
162         label = torch.where(label == 38, 1, label)
163         label = torch.where(label == 75, 2, label)
164         label = torch.where(label == 255, 3, label)
165         label = torch.where(label == 0, 0, label)
166
167         return image, clip_feature, label
168
169     # Defining Transformations
170     transform = transforms.Compose([
171         transforms.Resize(preprocessedSize, interpolation=Image.NEAREST),
172         transforms.ToTensor()
173     ])
174
175     # Loading Dataset & DataLoader
176     segmentation_dataset = SegmentationDataset(
177         image_files=train_images,
178         label_files=train_masks,
179         clip_features_dict=clip_features_trainval,
180         transform=None
181     )
182     segmentation_dataloader = DataLoader(segmentation_dataset, batch_size=batchSize, shuffle=True)
183
184     val_segmentation_dataset = SegmentationDataset(
185         image_files=val_images,
186         label_files=val_masks,
187         clip_features_dict=clip_features_trainval,
188         transform=None
189     )
190
191     val_segmentation_dataloader = DataLoader(val_segmentation_dataset, batch_size=batchSize, shuffle=True)
192
193     class SegmentationDecoder(nn.Module):
194         def __init__(self, clip_feature_dim=512, num_classes=classesNum):
195             super(SegmentationDecoder, self).__init__()
196
197             # Project CLIP features
198             self.clip_fc = nn.Linear(clip_feature_dim, 256)
199
200             # CNN Feature Extractor (with 4 input channels: RGB + Heatmap)
201             self.encoder = nn.Sequential(
202                 nn.Conv2d(4, 64, kernel_size=3, stride=1, padding=1),
203                 nn.ReLU(),
204                 nn.MaxPool2d(2),
205                 nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
206                 nn.ReLU(),
207             )
208
209             # Decoder
210             self.decoder = nn.Sequential(
211                 nn.ConvTranspose2d(384, 64, kernel_size=3, stride=1, padding=1),
212                 nn.ReLU(),
213                 nn.Upsample(size=(256, 256), mode='bilinear', align_corners=True),

```

```

214         nn.ConvTranspose2d(64, num_classes, kernel_size=3, stride=1, padding=1),
215     )
216
217     def forward(self, image, clip_features):
218         cnn_features = self.encoder(image)
219         clip_features = self.clip_fc(clip_features).view(clip_features.shape[0], 256, 1, 1)
220         clip_features = clip_features.expand(-1, -1, cnn_features.shape[2], cnn_features.shape[3])
221         fusion = torch.cat([cnn_features, clip_features], dim=1)
222         segmentation_output = self.decoder(fusion)
223         return segmentation_output
224
225
226     def dice_loss(pred, target, smooth=1e-6):
227         pred = torch.softmax(pred, dim=1)
228         target = torch.nn.functional.one_hot(target, num_classes=pred.shape[1]).permute(0, 3, 1, 2)
229         intersection = (pred * target).sum(dim=(2, 3))
230         union = pred.sum(dim=(2, 3)) + target.sum(dim=(2, 3))
231         dice = (2.0 * intersection + smooth) / (union + smooth)
232         return 1 - dice.mean()
233
234     # Initialize Model, Loss, Optimizer
235     segmentation_model = SegmentationDecoder().to(device)
236     # segmentation_criterion = nn.CrossEntropyLoss()
237     # segmentation_criterion = lambda outputs, masks: nn.CrossEntropyLoss()(outputs, masks) + dice_loss(outputs, masks)
238
239     segmentation_criterion = lambda outputs, masks: dice_loss(outputs, masks)
240
241     print("Loss: segmentation_criterion = lambda outputs, masks: nn.CrossEntropyLoss()(outputs, masks) + dice_loss(outputs, masks)")
242
243     segmentation_optimizer = optim.Adam(segmentation_model.parameters(), lr=learningRate)
244     scheduler = torch.optim.lr_scheduler.StepLR(segmentation_optimizer, step_size=5, gamma=0.5)
245     scaler = torch.cuda.amp.GradScaler()
246
247     # Training model
248     for epoch in tqdm(range(epochsNum), desc="Training Progress"):
249         segmentation_model.train()
250         running_loss = 0.0
251
252         print(f"Epoch [{epoch+1}/{epochsNum}] Running:")
253
254         for images, clip_features, labels in segmentation_dataloader:
255             images, clip_features, labels = images.to(device), clip_features.to(device), labels.to(device)
256
257             segmentation_optimizer.zero_grad()
258
259             # Mixed precision training
260             with torch.cuda.amp.autocast():
261                 outputs = segmentation_model(images, clip_features)
262                 loss = segmentation_criterion(outputs, labels)
263
264             # Scaling loss and backpropagating
265             scaler.scale(loss).backward()
266             scaler.step(segmentation_optimizer)
267             scaler.update()
268
269             running_loss += loss.item()
270
271         segmentation_model.eval()
272         val_loss = 0
273         with torch.no_grad():
274             for images, clip_features, labels in val_segmentation_dataloader:
275                 images, clip_features, labels = images.to(device), clip_features.to(device), labels.to(device)
276                 outputs = segmentation_model(images, clip_features)
277                 loss = segmentation_criterion(outputs, labels)
278                 val_loss += loss.item()
279
280         print(f'Epoch [{epoch+1}/{epochsNum}], Validation Loss: {val_loss / len(val_segmentation_dataloader)}')
281
282         scheduler.step()
283
284         avg_loss = running_loss / len(segmentation_dataloader)
285         print(f'Epoch [{epoch+1}/{epochsNum}], Training Loss: {avg_loss:.4f}')
286         print(f"Saved here: Data/Output/Models_DICEONLY/PROMPTED_CLIP_DiceOnly_epoch_{epoch+1}.pth")
287         torch.save(segmentation_model.state_dict(), f'Data/Output/Models_DICEONLY/PROMPTED_CLIP_DiceOnly_epoch_{epoch+1}.pth')
288
289     print("Training complete!")

```

### A.3.8 performance\_evaluation.py: Evaluating Models on Test Set

```

1  # Authors: Sahil Mehl Bawishi (s2677266) and Matteo Spadaccia (s2748897)
2  # Subject: Computer Vision Coursework. Performance evaluation
3  # Date: 21.03.2025
4
5
6  """O. Preliminary code"""
7  print("PERFORMANCE-EVALUATION:")
8
9  # Setting code-behaviour variables
10 classNum = 4

```

```

11 excludedClasses = [3]
12 class_to_pixel = {0: 0, 1: 38, 2: 75, 3: 255}
13 pixel_to_class = {v: k for k, v in class_to_pixel.items()}
14
15 image_dir = 'Data/Input/Test/color/'
16 label_dir = 'Data/Input/Test/label/'
17 heatmap_dir = 'Data/Input/Test/heatmaps/'
18
19 # (to skip testing of a model, set any relevant desired epoch to 0)
20 models_dir = 'Data/Output/Models_DICEONLY/'
21 unet_epoch = 0 # CROSS : 29, DICE: 22 path: /home/s2677266/CVis/Data/Output/Models/UNET_CROSS_2_EPOCH_
22 autoencoder_epoch = 0 # 95
23 segmentation_decoder_epoch = 0 #42
24 clip_decoder_epoch = 0 #42
25 prompted_clip_decoder_epoch = 0 #42
26
27 pathName = f'Data/Output/Models_DICEONLY/Not_Frozen_Val_Segmentation_95_Decoder_DiceOnly_epoch_96.pth'
28
29 # Importing useful libraries
30 import os
31 import torch
32 import torch.nn as nn
33 import torch.nn.functional as F
34 import numpy as np
35 from PIL import Image
36 from torchvision import transforms
37 from tqdm import tqdm
38 import open_clip
39
40 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
41 image_files = [f for f in os.listdir(image_dir) if f.endswith('.jpg', '.png']]]
42
43 # Defining data loading
44 transform = transforms.Compose([
45     transforms.Resize((256, 256)),
46     transforms.ToTensor(),
47 ])
48 def load_image_and_label(image_path, label_path):
49     """Load an image and its corresponding label, apply necessary transformations."""
50     image = Image.open(image_path).convert('RGB')
51     label = Image.open(label_path).convert('L') # label(grayscale)
52     input_tensor = transform(image).unsqueeze(0).to(device)
53     label_array = np.array(label)
54     return input_tensor, label_array
55
56 # Defining IoU function
57 def compute_iou(pred, target, num_classes=classNum, exclude_classes=[]):
58     """
59         Compute Intersection over Union (IoU) for multi-class segmentation using accumulated dataset-wide counts,
60         with the option to exclude specific classes from the computation.
61
62         :param pred: Predicted segmentation map (H, W), with class indices.
63         :param target: Ground truth segmentation map (H, W), with class indices.
64         :param num_classes: Number of classes in segmentation.
65         :param exclude_classes: List of class indices to exclude from IoU calculation.
66         :return: IoU scores per class and mean IoU.
67     """
68
69     intersection_counts = np.zeros(num_classes)
70     union_counts = np.zeros(num_classes)
71
72     for cls in range(num_classes):
73         if cls in exclude_classes:
74             continue
75
76         pred_inds = pred == cls
77         target_inds = target == cls
78
79         intersection_counts[cls] += torch.logical_and(pred_inds, target_inds).sum().item()
80         union_counts[cls] += torch.logical_or(pred_inds, target_inds).sum().item()
81
82     iou_per_class = [
83         intersection_counts[cls] / union_counts[cls] if union_counts[cls] > 0 else float('nan')
84         for cls in range(num_classes)
85         if cls not in exclude_classes
86     ]
87
88     mean_iou = np.nanmean(iou_per_class) if iou_per_class else float('nan')
89
90     return iou_per_class, mean_iou
91
92 # Defining Dice coefficient function
93 def compute_dice_coefficient(pred, target, num_classes=classNum, exclude_classes=[]):
94     """
95         Compute Dice Coefficient for multi-class segmentation,
96         with the option to exclude specific classes from the computation.
97
98         :param pred: Predicted segmentation map (H, W), with class indices.
99         :param target: Ground truth segmentation map (H, W), with class indices.
100        :param num_classes: Number of classes in segmentation.
101        :param exclude_classes: List of class indices to exclude from Dice calculation.
102        :return: Dice scores per class and mean Dice score.
103    """
104
105     dice_per_class = []

```

```

106     for cls in range(num_classes):
107         if cls in exclude_classes:
108             continue
109
110         pred_inds = pred == cls
111         target_inds = target == cls
112         intersection = 2.0 * torch.logical_and(pred_inds, target_inds).sum().item()
113         union = pred_inds.sum().item() + target_inds.sum().item()
114
115         if union == 0:
116             dice_per_class.append(float('nan'))
117         else:
118             dice_per_class.append(intersection / union)
119
120     mean_dice = np.nanmean(dice_per_class) if dice_per_class else float('nan')
121
122     return dice_per_class, mean_dice
123
124 # Defining pixel accuracy function
125 def compute_pixel_accuracy(pred, target, exclude_classes=[]):
126     """
127     Compute Pixel Accuracy,
128     with the option to exclude specific classes from the computation.
129
130     :param pred: Predicted segmentation map (H, W), with class indices.
131     :param target: Ground truth segmentation map (H, W), with class indices.
132     :param exclude_classes: List of class indices to exclude from accuracy calculation.
133     :return: Pixel accuracy score.
134     """
135
136     exclude_mask = torch.zeros_like(target, dtype=torch.bool)
137     for cls in exclude_classes:
138         exclude_mask |= (target == cls)
139
140     valid_pixels = ~exclude_mask
141     correct = (pred[valid_pixels] == target[valid_pixels]).sum().item()
142     total = valid_pixels.sum().item()
143
144     return correct / total if total > 0 else float('nan')
145
146
147 """1. UNet testing"""
148 if unet_epoch!=0:
149     print("\nTesting UNet...")
150
151 # Loading model
152 class UNet(nn.Module):
153     def __init__(self, num_classes=classNames):
154         super(UNet, self).__init__()
155
156         # Encoder
157         self.enc1 = self.conv_block(3, 64)
158         self.pool1 = nn.MaxPool2d(2)
159
160         self.enc2 = self.conv_block(64, 128)
161         self.pool2 = nn.MaxPool2d(2)
162
163         self.enc3 = self.conv_block(128, 256)
164         self.pool3 = nn.MaxPool2d(2)
165
166         self.enc4 = self.conv_block(256, 512)
167
168         # Decoder
169         self.up5 = nn.ConvTranspose2d(512, 256, kernel_size=2, stride=2)
170         self.dec5 = self.conv_block(512, 256)
171
172         self.up6 = nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2)
173         self.dec6 = self.conv_block(256, 128)
174
175         self.up7 = nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2)
176         self.dec7 = self.conv_block(128, 64)
177
178         self.final_conv = nn.Conv2d(64, num_classes, kernel_size=1) # output layer
179
180     def conv_block(self, in_channels, out_channels):
181         return nn.Sequential(
182             nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
183             nn.BatchNorm2d(out_channels),
184             nn.ReLU(inplace=True),
185             nn.Dropout(0.2),
186             nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1),
187             nn.BatchNorm2d(out_channels),
188             nn.ReLU(inplace=True),
189             nn.Dropout(0.2)
190         )
191
192     def forward(self, x):
193         # Encoder
194         c1 = self.enc1(x)
195         p1 = self.pool1(c1)
196
197         c2 = self.enc2(p1)
198         p2 = self.pool2(c2)
199
200         c3 = self.enc3(p2)

```

```

201     p3 = self.pool3(c3)
202
203     c4 = self.enc4(p3)
204
205     # Decoder
206     u5 = self.up5(c4)
207     u5 = torch.cat([u5, c3], dim=1)
208     c5 = self.dec5(u5)
209
210     u6 = self.up6(c5)
211     u6 = torch.cat([u6, c2], dim=1)
212     c6 = self.dec6(u6)
213
214     u7 = self.up7(c6)
215     u7 = torch.cat([u7, c1], dim=1)
216     c7 = self.dec7(u7)
217
218     outputs = self.final_conv(c7)
219
220     return outputs
221
222 model = UNet().to(device)
223 model.load_state_dict(torch.load(f'/home/s2677266/CVis/Data/Output/Models_DICEONLY/UNET_DiceOnly_EPOCH_88.pth',
224                         map_location=device))
225 model.eval()
226
227 print("(model loaded)")
228
229 # Evaluating image by image
230 all_intersection = np.zeros(3)
231 all_union = np.zeros(3)
232 all_dice = []
233 all_pixel_accs = []
234 for image_file in tqdm(image_files, desc="Processing images"):
235     image_path = os.path.join(image_dir, image_file)
236     label_path = os.path.join(label_dir, image_file.replace(".jpg", ".png"))
237     input_tensor, label_image = load_image_and_label(image_path, label_path)
238
239     # Inference
240     with torch.no_grad():
241         output_tensor = model(input_tensor)
242         output_tensor = F.softmax(output_tensor, dim=1) # softmax to get predicted class indices
243         predicted_classes = torch.argmax(output_tensor, dim=1).squeeze(0).cpu()
244         predicted_classes = F.interpolate( # resizing predicted mask to original label size
245             predicted_classes.unsqueeze(0).unsqueeze(0).float(),
246             size=(label_image.shape[0], label_image.shape[1]),
247             mode='nearest',
248         ).squeeze(0).squeeze(0).to(dtype=torch.uint8)
249         label_tensor = torch.from_numpy(label_image).to(dtype=torch.uint8) # conversion of label image to class
250         indices
251         label_class_map = torch.zeros_like(label_tensor, dtype=torch.uint8)
252         for pixel_value, class_idx in pixel_to_class.items():
253             label_class_map[label_tensor == pixel_value] = class_idx
254
255         # Computing metrics
256         iou_per_class, _ = compute_iou(predicted_classes, label_class_map, exclude_classes=excludedClasses)
257         _, mean_dice = compute_dice_coefficient(predicted_classes, label_class_map, exclude_classes = excludedClasses)
258         pixel_acc = compute_pixel_accuracy(predicted_classes, label_class_map, exclude_classes = excludedClasses)
259
260         for cls in range(classNum):
261             if cls in excludedClasses:
262                 continue
263             if not np.isnan(iou_per_class[cls]):
264                 all_intersection[cls] += torch.logical_and(predicted_classes == cls, label_class_map == cls).sum().item()
265                 all_union[cls] += torch.logical_or(predicted_classes == cls, label_class_map == cls).sum().item()
266         all_dice.append(mean_dice)
267         all_pixel_accs.append(pixel_acc)
268
269     # Displaying overall scores
270     final_iou_per_class = [
271         all_intersection[cls] / all_union[cls] if all_union[cls] > 0 else float('nan')
272         for cls in range(classNum) if cls not in exludedClasses
273     ]
274     overall_mean_iou = np.nanmean(final_iou_per_class)
275     overall_mean_dice = np.nanmean(all_dice)
276     overall_pixel_accuracy = np.mean(all_pixel_accs)
277
278     print(f"Per-Class IoU: {final_iou_per_class}")
279     print(f"Overall Mean IoU: {overall_mean_iou:.4f}")
280     print(f"Overall Mean Dice Coefficient: {overall_mean_dice:.4f}")
281     print(f"Overall Pixel Accuracy: {overall_pixel_accuracy:.4f}")
282
283 """2. Autoencoder testing"""
284 if autoencoder_epoch*segmentation_decoder_epoch!=0:
285     print("\nTesting Autoencoder...")
286
287     # Loading model
288     class Autoencoder(nn.Module):
289         def __init__(self):
290             super(Autoencoder, self).__init__()
291
292         # Encoder

```

```

293     self.encoder = nn.Sequential(
294         nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
295         nn.LeakyReLU(0.2),
296         nn.MaxPool2d(kernel_size=2, stride=2), # /2 size
297         nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
298         nn.LeakyReLU(0.2),
299         nn.MaxPool2d(kernel_size=2, stride=2), # /2 size
300         nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
301         nn.LeakyReLU(0.2),
302         nn.MaxPool2d(kernel_size=2, stride=2), # /2 size
303         nn.Conv2d(256, 128, kernel_size=3, stride=1, padding=1),
304         nn.LeakyReLU(0.2),
305     )
306
307     # Decoder
308     self.decoder = nn.Sequential(
309         nn.ConvTranspose2d(128, 256, kernel_size=3, stride=2, padding=1, output_padding=1), # x2 size
310         nn.LeakyReLU(0.2),
311         nn.ConvTranspose2d(256, 128, kernel_size=3, stride=2, padding=1, output_padding=1), # x2 size
312         nn.LeakyReLU(0.2),
313         nn.ConvTranspose2d(128, 64, kernel_size=3, stride=2, padding=1, output_padding=1), # x2 size
314         nn.LeakyReLU(0.2),
315         nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
316         nn.LeakyReLU(0.2),
317         nn.Conv2d(64, 3, kernel_size=3, stride=1, padding=1), # output layer
318         nn.Sigmoid()
319     )
320
321     def forward(self, x):
322         x = self.encoder(x)
323         x = self.decoder(x)
324         return x
325
326 class ResidualBlock(nn.Module):
327     def __init__(self, in_channels, out_channels):
328         super(ResidualBlock, self).__init__()
329         self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)
330         self.bn1 = nn.BatchNorm2d(out_channels)
331         self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)
332         self.bn2 = nn.BatchNorm2d(out_channels)
333         self.relu = nn.ReLU(inplace=True)
334
335     def forward(self, x):
336         residual = x
337         x = self.relu(self.bn1(self.conv1(x)))
338         x = self.bn2(self.conv2(x))
339         x += residual # skip connection
340         return self.relu(x)
341
342 class SegmentationDecoder(nn.Module):
343     def __init__(self, encoder):
344         super(SegmentationDecoder, self).__init__()
345
346         for param in encoder.parameters():
347             param.requires_grad = True
348
349         self.encoder = encoder
350
351         # Decoder (with residual blocks and skip connections)
352         self.decoder = nn.Sequential(
353             nn.ConvTranspose2d(128, 128, kernel_size=3, stride=2, padding=1, output_padding=1), # x2 size
354             ResidualBlock(128, 128),
355             nn.ConvTranspose2d(128, 64, kernel_size=3, stride=2, padding=1, output_padding=1), # x2 size
356             ResidualBlock(64, 64),
357             nn.ConvTranspose2d(64, 32, kernel_size=3, stride=2, padding=1, output_padding=1), # x2 size
358             ResidualBlock(32, 32),
359             nn.Conv2d(32, 32, kernel_size=3, padding=1),
360             nn.BatchNorm2d(32),
361             nn.ReLU(inplace=True),
362             nn.Conv2d(32, 4, kernel_size=1)
363         )
364     def forward(self, x):
365         x = self.encoder(x)
366         x = self.decoder(x)
367         return x
368
369     autoencoder = Autoencoder().to(device)
370     autoencoder.load_state_dict(torch.load(models_dir+f'updated_complex_working_autoencoder_epoch_{autoencoder_epoch}.pth', map_location=device))
371     autoencoder.eval()
372
373     # Loading segmentation model
374     segmentation_model = SegmentationDecoder(autoencoder.encoder).to(device)
375     segmentation_model.load_state_dict(torch.load(pathName, map_location=device))
376     segmentation_model.eval()
377
378     print("(model loaded)")
379
380     # Evaluating image by image
381     all_intersection = np.zeros(3)
382     all_union = np.zeros(3)
383     all_dice = []
384     all_pixel_accs = []
385     for image_file in tqdm(image_files, desc="Processing images"):
386         image_path = os.path.join(image_dir, image_file)

```

```

387     label_path = os.path.join(label_dir, image_file.replace(".jpg", ".png"))
388     input_tensor, label_image = load_image_and_label(image_path, label_path)
389
390     # Inference
391     with torch.no_grad():
392         output_tensor = segmentation_model(input_tensor)
393         output_tensor = F.softmax(output_tensor, dim=1)
394         predicted_classes = torch.argmax(output_tensor, dim=1).squeeze(0).cpu()
395         predicted_classes = F.interpolate(predicted_classes, # resizing predicted mask to original label size
396             predicted_classes.unsqueeze(0).unsqueeze(0).float(),
397             size=(label_image.shape[0], label_image.shape[1]),
398             mode='nearest',
399         ).squeeze(0).squeeze(0).to(dtype=torch.uint8)
400         label_tensor = torch.tensor(label_image, dtype=torch.uint8)
401         label_class_map = torch.zeros_like(label_tensor)
402         for pixel_value, class_idx in pixel_to_class.items():
403             label_class_map[label_tensor == pixel_value] = class_idx
404
405     # Computing metrics
406     iou_per_class, _ = compute_iou(predicted_classes, label_class_map, exclude_classes=excludedClasses)
407     _, mean_dice = compute_dice_coefficient(predicted_classes, label_class_map, exclude_classes = excludedClasses)
408     pixel_acc = compute_pixel_accuracy(predicted_classes, label_class_map, exclude_classes = excludedClasses)
409
410     for cls in range(classNum):
411         if cls in excludedClasses:
412             continue
413         if not np.isnan(iou_per_class[cls]):
414             all_intersection[cls] += torch.logical_and(predicted_classes == cls, label_class_map == cls).sum().item()
415             all_union[cls] += torch.logical_or(predicted_classes == cls, label_class_map == cls).sum().item()
416
417     all_dice.append(mean_dice)
418     all_pixel_accs.append(pixel_acc)
419
420     # Displaying overall scores
421     final_iou_per_class = [
422         all_intersection[cls] / all_union[cls] if all_union[cls] > 0 else float('nan')
423         for cls in range(classNum) if cls not in exludedClasses
424     ]
425     overall_mean_iou = np.nanmean(final_iou_per_class)
426     overall_mean_dice = np.nanmean(all_dice)
427     overall_pixel_accuracy = np.mean(all_pixel_accs)
428
429     print(f"Per-Class IoU: {final_iou_per_class}")
430     print(f"Overall Mean IoU: {overall_mean_iou:.4f}")
431     print(f"Overall Mean Dice Coefficient: {overall_mean_dice:.4f}")
432     print(f"Overall Pixel Accuracy: {overall_pixel_accuracy:.4f}")
433
434
435     """3. CLIP testing"""
436     if clip_decoder_epoch!=0:
437         print("\nTesting CLIP...")
438
439     # Loading model
440     class ClipDecoder(nn.Module):
441         def __init__(self, clip_feature_dim=512, num_classes=classNum):
442             super(ClipDecoder, self).__init__()
443
444             # Project CLIP features
445             self.clip_fc = nn.Linear(clip_feature_dim, 256)
446
447             # CNN Feature Extractor
448             self.encoder = nn.Sequential(
449                 nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
450                 nn.ReLU(),
451                 nn.MaxPool2d(2),
452                 nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
453                 nn.ReLU(),
454             )
455
456             # Decoder
457             self.decoder = nn.Sequential(
458                 nn.ConvTranspose2d(384, 64, kernel_size=3, stride=1, padding=1),
459                 nn.ReLU(),
460                 nn.Upsample(size=(256, 256), mode='bilinear', align_corners=True),
461                 nn.ConvTranspose2d(64, num_classes, kernel_size=3, stride=1, padding=1),
462             )
463
464         def forward(self, image, clip_features):
465             cnn_features = self.encoder(image)
466             clip_features = self.clip_fc(clip_features).view(clip_features.shape[0], 256, 1, 1)
467             clip_features = clip_features.expand(-1, -1, cnn_features.shape[2], cnn_features.shape[3])
468             fusion = torch.cat([cnn_features, clip_features], dim=1)
469             segmentation_output = self.decoder(fusion)
470             return segmentation_output
471
472     model, preprocess_train, preprocess_val = open_clip.create_model_and_transforms("ViT-B-32", pretrained="openai")
473     model = model.to(device)
474     tokenizer = open_clip.get_tokenizer("ViT-B-32")
475
476     # Loading segmentation model
477     clip_segmentation_model = ClipDecoder().to(device)
478     # clip_segmentation_model.load_state_dict(torch.load(models_dir+f'New_clip_segmentation_model_epoch_{clip_decoder_epoch}.pth', map_location=device))

```

```

479     clip_segmentation_model.load_state_dict(torch.load(models_dir+f'CLIP_DiceOnly_epoch_{clip_decoder_epoch}.pth',
480                                         map_location=device))
481     clip_segmentation_model.eval()
482
483     print("(model loaded)")
484
485     # Evaluating image by image
486     all_intersection = {cls: 0 for cls in range(classNum)}
487     all_union = {cls: 0 for cls in range(classNum)}
488     all_dice = []
489     all_pixel_accs = []
490
491     for image_file in tqdm(image_files, desc="Processing images"):
492         image_path = os.path.join(image_dir, image_file)
493         label_path = os.path.join(label_dir, image_file.replace('.jpg', '.png'))
494         _, label_image = load_image_and_label(image_path, label_path)
495
496         # CLIP feature extraction
497         image_clip = Image.open(image_path).convert("RGB")
498         image_clip_tensor = preprocess_val(image_clip).unsqueeze(0).to(device)
499         with torch.no_grad():
500             clip_features = model.encode_image(image_clip_tensor)
501             clip_features = clip_features / clip_features.norm(dim=-1, keepdim=True)
502
503         # Inference
504         image_seg = Image.open(image_path).convert("RGB")
505         image_tensor = transforms.ToTensor()(image_seg).unsqueeze(0).to(device)
506         with torch.no_grad():
507             output_tensor = clip_segmentation_model(image_tensor, clip_features)
508             pred_mask = output_tensor.squeeze(0) # output conversion to predicted segmentation mask
509             pred_mask = torch.argmax(pred_mask, dim=0).cpu()
510             pred_mask = F.interpolate( # resizing predicted mask to original label size
511                 pred_mask.unsqueeze(0).unsqueeze(0).float(),
512                 size=(label_image.shape[0], label_image.shape[1]),
513                 mode='nearest',
514             ).squeeze(0).squeeze(0).to(dtype=torch.uint8)
515             label_tensor = torch.from_numpy(label_image).to(dtype=torch.uint8)
516             gt_mask_class = torch.zeros_like(label_tensor, dtype=torch.uint8)
517             for pixel_value, class_idx in pixel_to_class.items():
518                 gt_mask_class[label_tensor == pixel_value] = class_idx
519
520             # Computing metrics
521             iou_per_class, _ = compute_iou(pred_mask, gt_mask_class, classNum, exclude_classes=excludedClasses)
522             _, mean_dice = compute_dice_coefficient(pred_mask, gt_mask_class, exclude_classes=excludedClasses)
523             mean_dice = float(mean_dice)
524             pixel_acc = compute_pixel_accuracy(pred_mask, gt_mask_class, exclude_classes=excludedClasses)
525
526             for cls in range(classNum):
527                 if cls in excludedClasses:
528                     continue
529                 if not np.isnan(iou_per_class[cls]):
530                     all_intersection[cls] += torch.logical_and(pred_mask == cls, gt_mask_class == cls).sum().item()
531                     all_union[cls] += torch.logical_or(pred_mask == cls, gt_mask_class == cls).sum().item()
532
533             all_dice.append(mean_dice)
534             all_pixel_accs.append(pixel_acc)
535
536             # Displaying overall scores
537             final_iou_per_class = [
538                 all_intersection[cls] / all_union[cls] if all_union[cls] > 0 else float('nan')
539                 for cls in range(classNum) if cls not in excludedClasses
540             ]
541             overall_mean_iou = np.nanmean(final_iou_per_class)
542             overall_mean_dice = np.nanmean(all_dice)
543             overall_pixel_accuracy = np.mean(all_pixel_accs)
544
545             print(f"IoU_per_class:{final_iou_per_class}")
546             print(f"Overall_Mean_IoU:{overall_mean_iou:.4f}")
547             print(f"Overall_Mean_Dice_Coefficient:{overall_mean_dice:.4f}")
548             print(f"Overall_Pixel_Accuracy:{overall_pixel_accuracy:.4f}")
549
550             """4. Prompted CLIP testing"""
551             if prompted_clip_decoder_epoch!=0:
552                 print("\nTesting_prompted_CLIP...")
553
554             # Loading model
555             class PromptedClipDecoder(nn.Module):
556                 def __init__(self, clip_feature_dim=512, num_classes=classNum):
557                     super(PromptedClipDecoder, self).__init__()
558
559                     # Project CLIP features
560                     self.clip_fc = nn.Linear(clip_feature_dim, 256)
561
562                     # CNN Feature Extractor (with 4 input channels: RGB + Heatmap)
563                     self.encoder = nn.Sequential(
564                         nn.Conv2d(4, 64, kernel_size=3, stride=1, padding=1),
565                         nn.ReLU(),
566                         nn.MaxPool2d(2),
567                         nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
568                         nn.ReLU(),
569                     )
570
571                     # Decoder
572                     self.decoder = nn.Sequential(
573                         nn.ConvTranspose2d(384, 64, kernel_size=3, stride=1, padding=1),

```

```

573     nn.ReLU(),
574     nn.Upsample(size=(256, 256), mode='bilinear', align_corners=True),
575     nn.ConvTranspose2d(64, num_classes, kernel_size=3, stride=1, padding=1),
576 )
577
578 def forward(self, image, clip_features):
579     cnn_features = self.encoder(image)
580     clip_features = self.clip_fc(clip_features).view(clip_features.shape[0], 256, 1, 1)
581     clip_features = clip_features.expand(-1, -1, cnn_features.shape[2], cnn_features.shape[3])
582     fusion = torch.cat([cnn_features, clip_features], dim=1)
583     segmentation_output = self.decoder(fusion)
584     return segmentation_output
585
586 model, preprocess_train, preprocess_val = open_clip.create_model_and_transforms("ViT-B-32", pretrained="openai")
587 model = model.to(device)
588 tokenizer = open_clip.get_tokenizer("ViT-B-32")
589
590 # Loading segmentation model
591 prompted_clip_segmentation_model = PromptedClipDecoder().to(device)
592 prompted_clip_segmentation_model.load_state_dict(torch.load(models_dir+f'PROMPTED_CLIP_DiceOnly_epoch_{epoch}.pth',
593                                         map_location=device))
594 prompted_clip_segmentation_model.eval()
595
596 print("(model loaded)")
597
598 # Evaluating image by image
599 all_intersection = {cls: 0 for cls in range(classNum)}
600 all_union = {cls: 0 for cls in range(classNum)}
601 all_dice = []
602 all_pixel_accs = []
603 for image_file in tqdm(image_files, desc="Processing images"):
604     image_path = os.path.join(image_dir, image_file)
605     label_path = os.path.join(label_dir, image_file.replace('.jpg', '.png'))
606     heatmap_path = os.path.join(heatmap_dir, image_file.split('.')[0]+'_heatmap.png')
607     _, label_image = load_image_and_label(image_path, label_path)
608
609     # CLIP feature extraction
610     image_clip = Image.open(image_path).convert("RGB")
611     image_clip_tensor = preprocess_val(image_clip).unsqueeze(0).to(device)
612     with torch.no_grad():
613         clip_features = model.encode_image(image_clip_tensor)
614     clip_features = clip_features / clip_features.norm(dim=-1, keepdim=True)
615
616     # Inference
617     image_seg = Image.open(image_path).convert("RGB").resize((500, 500))
618     heatmap = Image.open(heatmap_path).convert('L')
619     heatmap = transforms.ToTensor()(heatmap).unsqueeze(0).to(device)
620     image_tensor = transforms.ToTensor()(image_seg).unsqueeze(0).to(device)
621     image_tensor = torch.cat([image_tensor, heatmap], dim=1)
622     with torch.no_grad():
623         output_tensor = prompted_clip_segmentation_model(image_tensor, clip_features)
624     pred_mask = output_tensor.squeeze(0) # output conversion to predicted segmentation mask
625     pred_mask = torch.argmax(pred_mask, dim=0).cpu()
626     pred_mask = F.interpolate( # resizing predicted mask to original label size
627         pred_mask.unsqueeze(0).unsqueeze(0).float(),
628         size=(label_image.shape[0], label_image.shape[1]),
629         mode='nearest',
630     ).squeeze(0).squeeze(0).to(dtype=torch.uint8)
631     label_tensor = torch.from_numpy(label_image).to(dtype=torch.uint8)
632     gt_mask_class = torch.zeros_like(label_tensor, dtype=torch.uint8)
633     for pixel_value, class_idx in pixel_to_class.items():
634         gt_mask_class[label_tensor == pixel_value] = class_idx
635
636     # Computing metrics
637     iou_per_class, _ = compute_iou(pred_mask, gt_mask_class, classNum, exclude_classes=excludedClasses)
638     _, mean_dice = compute_dice_coefficient(pred_mask, gt_mask_class, exclude_classes=excludedClasses)
639     mean_dice = float(mean_dice)
640     pixel_acc = compute_pixel_accuracy(pred_mask, gt_mask_class, exclude_classes=excludedClasses)
641
642     for cls in range(classNum):
643         if cls in excludedClasses:
644             continue
645         if not np.isnan(iou_per_class[cls]):
646             all_intersection[cls] += torch.logical_and(pred_mask == cls, gt_mask_class == cls).sum().item()
647             all_union[cls] += torch.logical_or(pred_mask == cls, gt_mask_class == cls).sum().item()
648
649     all_dice.append(mean_dice)
650     all_pixel_accs.append(pixel_acc)
651
652     # Displaying overall scores
653     final_iou_per_class = [
654         all_intersection[cls] / all_union[cls] if all_union[cls] > 0 else float('nan')
655         for cls in range(classNum) if cls not in excludedClasses
656     ]
657     overall_mean_iou = np.nanmean(final_iou_per_class)
658     overall_mean_dice = np.nanmean(all_dice)
659     overall_pixel_accuracy = np.mean(all_pixel_accs)
660
661     print(f"IoU_per_class:{final_iou_per_class}")
662     print(f"Overall_Mean_IoU:{overall_mean_iou:.4f}")
663     print(f"Overall_Mean_Dice_Coefficient:{overall_mean_dice:.4f}")
664     print(f"Overall_Pixel_Accuracy:{overall_pixel_accuracy:.4f}")

```

### A.3.9 robustness\_exploration.py: Robustness Exploration

```

1 # Authors: Sahil Mehl Bavishi (s2677266) and Matteo Spadaccia (s2748897)
2 # Subject: Computer Vision Coursework. Robustness exploration on the prompted CLIP model
3 # Date: 31.03.2025
4
5
6 """0. Preliminary code"""
7
8 print("ROBUSTNESS\u20D7EXPLORATION\u20D7WITH\u20D7DIFFERENT\u20D7PERTURBATION\u20D7TYPES\u20D7(PROMPTED\u20D7CLIP):")
9
10 # Setting code-behaviour variables
11 classNum = 4
12 pixel_to_class = {0: 0, 38: 1, 75: 2, 255: 3}
13 excludedClasses = [3]      # classes excludeed from metric calculation (3 not to consider the outline)
14 maxImages = -1             # number of input images to load from the defined set (0 to consider them all)
15 digits = 3                 # digits to approximate decimal results in, when printed
16
17 # Defining paths
18 model_dir = f'Data/Output/Models2/PROMPTED_CLIP_DICE2_epoch_42.pth'
19 images_dir = 'Data/Input/Test/color/'
20 masks_dir = 'Data/Input/Test/label/'
21 heatmaps_dir = 'Data/Input/Test/heatmaps/'
22 output_plots_dir = 'Data/Output/Visuals/Prompted_CLIP_Robustness/'
23
24 # Importing useful libraries
25 import os
26 import numpy as np
27 import torch
28 import torch.nn as nn
29 from torchvision import transforms
30 from PIL import Image
31 import open_clip
32 import matplotlib.pyplot as plt
33 import cv2
34 from skimage.util import random_noise
35
36 device = "cuda" if torch.cuda.is_available() else "cpu"
37
38 image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]
39
40 # Loading pre-trained prompted CLIP model
41 CLIP_model, preprocess_train, preprocess_val = open_clip.create_model_and_transforms("ViT-B-32", pretrained="openai")
42 CLIP_model = CLIP_model.to(device)
43 tokenizer = open_clip.get_tokenizer("ViT-B-32")
44
45 class PromptedClipDecoder(nn.Module):
46     def __init__(self, clip_feature_dim=512, num_classes=classNum):
47         super(PromptedClipDecoder, self).__init__()
48
49         # Project CLIP features
50         self.clip_fc = nn.Linear(clip_feature_dim, 256)
51
52         # CNN Feature Extractor (with 4 input channels: RGB + Heatmap)
53         self.encoder = nn.Sequential(
54             nn.Conv2d(4, 64, kernel_size=3, stride=1, padding=1),
55             nn.ReLU(),
56             nn.MaxPool2d(2),
57             nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
58             nn.ReLU(),
59         )
60
61         # Decoder
62         self.decoder = nn.Sequential(
63             nn.ConvTranspose2d(384, 64, kernel_size=3, stride=1, padding=1),
64             nn.ReLU(),
65             nn.Upsample(size=(256, 256), mode='bilinear', align_corners=True),
66             nn.ConvTranspose2d(64, num_classes, kernel_size=3, stride=1, padding=1),
67         )
68
69     def forward(self, image, clip_features):
70         cnn_features = self.encoder(image)
71         clip_features = self.clip_fc(clip_features).view(clip_features.shape[0], 256, 1, 1)
72         clip_features = clip_features.expand(-1, -1, cnn_features.shape[2], cnn_features.shape[3])
73         fusion = torch.cat([cnn_features, clip_features], dim=1)
74         segmentation_output = self.decoder(fusion)
75         return segmentation_output
76
77 # Loading segmentation model
78 model = PromptedClipDecoder().to(device)
79 model.load_state_dict(torch.load(model_dir, map_location=device))
80 model.eval()
81
82 def compute_dice_coefficient(pred, target, num_classes=classNum, exclude_classes=[]):
83     """
84     Compute Dice Coefficient for multi-class segmentation with the option to exclude specific classes.
85
86     :param pred: Predicted segmentation map (H, W), with class indices.
87     :param target: Ground truth segmentation map (H, W), with class indices.
88     :param num_classes: Number of classes in segmentation.
89     :param exclude_classes: List of class indices to exclude from Dice calculation.
90     """
91
92

```

```

93     dice_per_class = []
94     for cls in range(num_classes):
95         if cls in exclude_classes:
96             continue
97
98     pred_inds = pred == cls
99     target_inds = target == cls
100    intersection = 2.0 * torch.logical_and(pred_inds, target_inds).sum().item()
101    union = pred_inds.sum().item() + target_inds.sum().item()
102
103    if union == 0:
104        dice_per_class.append(float('nan'))
105    else:
106        dice_per_class.append(intersection / union)
107
108    mean_dice = np.nanmean(dice_per_class) if dice_per_class else float('nan')
109
110    return dice_per_class, mean_dice
111
112
113 """a. Gaussian noise"""
114
115 def apply_gaussian_noise(image, std_dev):
116     """Applies Gaussian noise to an image with a given standard deviation."""
117     noise = np.random.normal(0, std_dev, image.shape)
118     noisy_image = np.clip(image + noise, 0, 255).astype(np.uint8)
119     return noisy_image
120
121 def evaluate_gaussian_noise(model, images_dir, masks_dir, std_dev_levels, device, save_as:str="a_gaussian_noise"):
122     """Evaluates the model with Gaussian noise perturbation and plots Dice score vs. noise level."""
123     image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]
124     dice_scores = []
125
126     for std_dev in std_dev_levels:
127         total_dice = []
128         counter = 0
129         for img_file in image_files:
130             counter += 1
131             if counter == maxImages:
132                 break
133             img_path = os.path.join(images_dir, img_file)
134             mask_path = os.path.join(masks_dir, img_file.replace('.jpg', '.png'))
135             heatmap_path = os.path.join(heatmaps_dir, img_file.split('.')[0]+'_heatmap.png')
136
137             image = np.array(Image.open(img_path).convert("RGB").resize((500, 500)))
138             noisy_image = apply_gaussian_noise(image, std_dev)
139             image_tensor = transforms.ToTensor()(Image.fromarray(noisy_image)).unsqueeze(0).to(device)
140             heatmap = Image.open(heatmap_path).convert('L').resize((500, 500))
141             heatmap = transforms.ToTensor()(heatmap).unsqueeze(0).to(device)
142             image_tensor = torch.cat([image_tensor, heatmap], dim=1)
143
144             gt_mask = Image.open(mask_path).convert("L")
145             gt_mask = gt_mask.resize((256, 256), Image.NEAREST)
146             label_array = np.array(gt_mask)
147
148             label_tensor = torch.from_numpy(label_array).to(dtype=torch.uint8)
149             gt_mask_class = torch.zeros_like(label_tensor, dtype=torch.uint8)
150
151             for pixel_value, class_idx in pixel_to_class.items():
152                 gt_mask_class[label_tensor == pixel_value] = class_idx
153
154             image_clip_tensor = preprocess_val(Image.open(img_path).convert("RGB")).unsqueeze(0).to(device)
155
156             with torch.no_grad():
157                 clip_features = CLIP_model.encode_image(image_clip_tensor)
158                 clip_features = clip_features / clip_features.norm(dim=-1, keepdim=True)
159                 model_output = model(image_tensor, clip_features)
160                 pred_mask = model_output.squeeze(0)
161                 pred_mask = torch.argmax(pred_mask, dim=0).cpu()
162
163             _, mean_dice = compute_dice_coefficient(pred_mask, gt_mask_class, exclude_classes = excludedClasses)
164
165             total_dice.append(mean_dice)
166
167             dice_scores.append(np.mean(total_dice))
168
169     plt.figure()
170     plt.plot(std_dev_levels, dice_scores, marker='o')
171     plt.xlabel("Gaussian Noise Standard Deviation")
172     plt.ylabel("Mean Dice Score")
173     plt.title("Gaussian Noise")
174     plt.grid()
175     plt.savefig(output_plots_dir + save_as + ".pdf", format="pdf")
176
177     return dice_scores
178
179 print("\n\na. Gaussian Noise...", end=' ')
180 std_dev_levels = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
181 dice_scores = evaluate_gaussian_noise(model, images_dir, masks_dir, std_dev_levels, device)
182 print("Done!")
183 print("*49")
184 print("Just deviations", " ".join(["{}" for value in std_dev_levels]), "|")
185 print("meanDiceScore", " ".join(["{}" for d in dice_scores]), "|")
186 print("*49,\n")
187

```

```

188     """b. Gaussian blurring"""
189
190     def apply_gaussian_blur(image, ksize):
191         """Applies Gaussian blur to an image with a given kernel size."""
192         if ksize % 2 == 0:
193             ksize += 1
194
195         kernel = np.array([
196             [1/16, 2/16, 1/16],
197             [2/16, 4/16, 2/16],
198             [1/16, 2/16, 1/16]
199         ])
200         blurred_image = cv2.filter2D(image, -1, kernel)
201         return blurred_image
202
203     def evaluate_gaussian_blur(model, images_dir, masks_dir, max.blur.level, device, save_as:str="b_gaussian_blurring"):
204         """Evaluates the model with Gaussian blurring perturbation and plots Dice score vs. blur level."""
205         image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]
206         dice_scores = []
207
208         for blur_level in range(max.blur.level + 1):
209             total_dice = []
210
211             counter = 0
212             for img_file in image_files:
213                 counter += 1
214                 if counter == maxImages:
215                     break
216
217                 img_path = os.path.join(images_dir, img_file)
218                 mask_path = os.path.join(masks_dir, img_file.replace('.jpg', '.png'))
219                 heatmap_path = os.path.join(heatmaps_dir, img_file.split('.')[0]+'_heatmap.png')
220
221                 blurred_image = np.array(Image.open(img_path).convert("RGB").resize((500, 500)))
222                 for i in range(blur_level):
223                     blurred_image = apply_gaussian_blur(blurred_image, 3)
224
225                 image_tensor = transforms.ToTensor()(Image.fromarray(blurred_image)).unsqueeze(0).to(device)
226                 heatmap = Image.open(heatmap_path).convert('L').resize((500, 500))
227                 heatmap = transforms.ToTensor()(heatmap).unsqueeze(0).to(device)
228                 image_tensor = torch.cat([image_tensor, heatmap], dim=1)
229
230                 gt_mask = Image.open(mask_path).convert("L")
231                 gt_mask = gt_mask.resize(256, 256), Image.NEAREST)
232                 label_array = np.array(gt_mask)
233
234                 label_tensor = torch.from_numpy(label_array).to(dtype=torch.uint8)
235                 gt_mask_class = torch.zeros_like(label_tensor, dtype=torch.uint8)
236
237                 for pixel_value, class_idx in pixel_to_class.items():
238                     gt_mask_class[label_tensor == pixel_value] = class_idx
239
240                 image_clip_tensor = preprocess_val(Image.open(img_path).convert("RGB")).unsqueeze(0).to(device)
241
242                 with torch.no_grad():
243                     clip_features = CLIP_model.encode_image(image_clip_tensor)
244                     clip_features = clip_features / clip_features.norm(dim=-1, keepdim=True)
245                     model_output = model(image_tensor, clip_features)
246                     pred_mask = model_output.squeeze(0)
247                     pred_mask = torch.argmax(pred_mask, dim=0).cpu()
248
249                     mean_dice = compute_dice_coefficient(pred_mask, gt_mask_class, exclude_classes = excludedClasses)
250                     total_dice.append(mean_dice)
251
252             dice_scores.append(np.mean(total_dice))
253
254             plt.figure()
255             plt.plot(range(max.blur.level + 1), dice_scores, marker='o')
256             plt.xlabel("Gaussian Blur Level (number of iterative 3x3 mask applications)")
257             plt.ylabel("Mean Dice Score")
258             plt.title("Gaussian Blurring")
259             plt.grid()
260             plt.savefig(output_plots_dir + save_as + ".pdf", format="pdf")
261
262     print("\nb)\u00d7Gaussian\u00d7Blurring...", end=' ')
263     max.blur.level = 9
264     dice_scores = evaluate_gaussian_blur(model, images_dir, masks_dir, max.blur.level, device)
265     print("Done!")
266     print("*" * 49)
267     print("\u00d7blurring\u00d7steps\u00d7=", "\u00d7".join([f"\u00d7{value:{digits+2}}\u00d7" for value in list(range(max.blur.level+1))]), "\u00d7")
268     print("\u00d7mean\u00d7Dice\u00d7score\u00d7=", "\u00d7".join([f"\u00d7{d:{digits+2}}.{digits}f\u00d7" for d in dice_scores]), "\u00d7")
269     print("*" * 49, "\n")
270
271     """c. Contrast increase"""
272
273     def apply_contrast_change(image, contrast_factor):
274         """Applies contrast increase to an image by multiplying pixel values by a contrast factor."""
275         contrast_image = image * contrast_factor
276         contrast_image = np.clip(contrast_image, 0, 255).astype(np.uint8)
277         return contrast_image
278
279     def evaluate_contrast_change(model, images_dir, masks_dir, contrast_factors, device, save_as:str="c_contrast_increase",
280                                 invert_axis:bool=False):
281         """Evaluates the model with increased contrast perturbation and plots Dice score vs. contrast factor."""

```

```

282     image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]
283     dice_scores = []
284
285     for contrast_factor in contrast_factors:
286         total_dice = []
287
288         counter = 0
289         for img_file in image_files:
290             counter += 1
291             if counter == maxImages:
292                 break
293             img_path = os.path.join(images_dir, img_file)
294             mask_path = os.path.join(masks_dir, img_file.replace('.jpg', '.png'))
295             heatmap_path = os.path.join(heatmaps_dir, img_file.split('.')[0] + '_heatmap.png')
296
297             image = np.array(Image.open(img_path).convert("RGB").resize((500, 500)))
298             contrast_image = apply_contrast_change(image, contrast_factor)
299             image_tensor = transforms.ToTensor()(Image.fromarray(contrast_image)).unsqueeze(0).to(device)
300             heatmap = Image.open(heatmap_path).convert('L').resize((500, 500))
301             heatmap = transforms.ToTensor()(heatmap).unsqueeze(0).to(device)
302             image_tensor = torch.cat([image_tensor, heatmap], dim=1)
303
304             gt_mask = Image.open(mask_path).convert("L")
305             gt_mask = gt_mask.resize((256, 256), Image.NEAREST)
306             label_array = np.array(gt_mask)
307
308             label_tensor = torch.from_numpy(label_array).to(dtype=torch.uint8)
309             gt_mask_class = torch.zeros_like(label_tensor, dtype=torch.uint8)
310
311             for pixel_value, class_idx in pixel_to_class.items():
312                 gt_mask_class[label_tensor == pixel_value] = class_idx
313
314             image_clip_tensor = preprocess_val(Image.open(img_path).convert("RGB")).unsqueeze(0).to(device)
315
316             with torch.no_grad():
317                 clip_features = CLIP_model.encode_image(image_clip_tensor)
318                 clip_features = clip_features / clip_features.norm(dim=-1, keepdim=True)
319                 model_output = model(image_tensor, clip_features)
320                 pred_mask = model_output.squeeze(0)
321                 pred_mask = torch.argmax(pred_mask, dim=0).cpu()
322
323             _, mean_dice = compute_dice_coefficient(pred_mask, gt_mask_class, exclude_classes = excludedClasses)
324             total_dice.append(mean_dice)
325
326             dice_scores.append(np.mean(total_dice))
327
328     plt.figure()
329     plt.plot(contrast_factors, dice_scores, marker='o')
330     plt.xlabel("Contrast Factor")
331     plt.ylabel("Mean Dice Score")
332     plt.title("Contrast Decrease" if invert_axis else "Contrast Increase")
333     plt.grid()
334     if invert_axis: plt.gca().invert_xaxis()
335     plt.savefig(output_plots_dir + save_as + ".pdf", format="pdf")
336
337     return dice_scores
338
339 print("\n\n)c. Contrast Increase...", end=' ')
340 contrast_factors = [1.0, 1.01, 1.02, 1.03, 1.04, 1.05, 1.1, 1.15, 1.2, 1.25]
341 dice_scores = evaluate_contrast_change(model, images_dir, masks_dir, contrast_factors, device)
342 print("Done!")
343 print("-" * 49)
344 print("|\u03bccontrast\u03bcfactor\u03bc=", "\u03bc|\u03bc".join([f"\u03bc{value:{{digits+2}}}" for value in contrast_factors]), "|")
345 print("|\u03bcmean\u03bcDice\u03bcScore\u03bc=", "\u03bc|\u03bc".join([f"\u03bc{d:{{digits+2}}}.{{digits}f}" for d in dice_scores]), "|")
346 print("-" * 49, "\n")
347
348 """
349     d. Contrast decrease"""
350     # (using contrast increase functions)
351
352 print("\n)d. Contrast Decrease...", end=' ')
353 contrast_factors = [1.0, 0.95, 0.90, 0.85, 0.80, 0.60, 0.40, 0.30, 0.20, 0.10]
354 dice_scores = evaluate_contrast_change(model, images_dir, masks_dir, contrast_factors, device, save_as="d_contrast_decrease", invert_axis=True)
355 print("Done!")
356 print("-" * 49)
357 print("|\u03bccontrast\u03bcfactor\u03bc\u03bc=", "\u03bc|\u03bc".join([f"\u03bc{value:{{digits+2}}}" for value in contrast_factors]), "|")
358 print("|\u03bcmean\u03bcDice\u03bcScore\u03bc\u03bc=", "\u03bc|\u03bc".join([f"\u03bc{d:{{digits+2}}}.{{digits}f}" for d in dice_scores]), "|")
359 print("-" * 49, "\n")
360
361 """
362     e. Brightness Increase"""
363
364 def apply_brightness_change(image, brightness_offset):
365     """Applies brightness change by adding an offset to pixel values."""
366     bright_image = image.astype(np.int16) + brightness_offset
367     bright_image = np.clip(bright_image, 0, 255).astype(np.uint8)
368     return bright_image
369
370 def evaluate_brightness_change(model, images_dir, masks_dir, brightness_offsets, device, save_as:str="e_brightness_increase", invert_axis:bool=False):
371     """Evaluates the model with brightness perturbation and plots Dice score vs. brightness offset."""
372     image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]
373     dice_scores = []

```

```

375     for brightness_offset in brightness_offsets:
376         total_dice = []
377
378         counter = 0
379         for img_file in image_files:
380             counter += 1
381             if counter == maxImages:
382                 break
383             img_path = os.path.join(images_dir, img_file)
384             mask_path = os.path.join(masks_dir, img_file.replace('.jpg', '.png'))
385             heatmap_path = os.path.join(heatmaps_dir, img_file.split('.')[0]+'_heatmap.png')
386
387             image = np.array(Image.open(img_path).convert("RGB").resize((500, 500)))
388             bright_image = apply_brightness_change(image, brightness_offset)
389             image_tensor = transforms.ToTensor()(Image.fromarray(bright_image)).unsqueeze(0).to(device)
390             heatmap = Image.open(heatmap_path).convert('L').resize((500, 500))
391             heatmap = transforms.ToTensor()(heatmap).unsqueeze(0).to(device)
392             image_tensor = torch.cat([image_tensor, heatmap], dim=1)
393
394             gt_mask = Image.open(mask_path).convert("L")
395             gt_mask = gt_mask.resize((256, 256), Image.NEAREST)
396             label_array = np.array(gt_mask)
397
398             label_tensor = torch.from_numpy(label_array).to(dtype=torch.uint8)
399             gt_mask_class = torch.zeros_like(label_tensor, dtype=torch.uint8)
400
401             for pixel_value, class_idx in pixel_to_class.items():
402                 gt_mask_class[label_tensor == pixel_value] = class_idx
403
404             image_clip_tensor = preprocess_val(Image.open(img_path).convert("RGB")).unsqueeze(0).to(device)
405
406             with torch.no_grad():
407                 clip_features = CLIP_model.encode_image(image_clip_tensor)
408                 clip_features = clip_features / clip_features.norm(dim=-1, keepdim=True)
409                 model_output = model(image_tensor, clip_features)
410                 pred_mask = model_output.squeeze(0)
411                 pred_mask = torch.argmax(pred_mask, dim=0).cpu()
412
413                 _, mean_dice = compute_dice_coefficient(pred_mask, gt_mask_class, exclude_classes=excludedClasses)
414                 total_dice.append(mean_dice)
415
416             dice_scores.append(np.mean(total_dice))
417
418     plt.figure()
419     plt.plot(brightness_offsets, dice_scores, marker='o')
420     plt.xlabel("Brightness_Offset")
421     plt.ylabel("Mean_Dice_Score")
422     plt.title("BrightnessDecrease" if invert_axis else "BrightnessIncrease")
423     plt.grid()
424     if invert_axis: plt.gca().invert_xaxis()
425     plt.savefig(output_plots_dir + save_as + ".pdf", format="pdf")
426
427     return dice_scores
428
429 print("\nne) Brightness_Increase...", end=' ')
430 brightness_offsets = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
431 dice_scores = evaluate_brightness_change(model, images_dir, masks_dir, brightness_offsets, device)
432 print("Done!")
433 print("*"*49)
434 print("|bright_offset=", "|".join([f"value:{digits+2}" for value in brightness_offsets]), "|")
435 print("|meanDiceScore=", "|".join([f"d:{digits+2}.{digits}f" for d in dice_scores]), "|")
436 print("*"*49, "\n")
437
438 """
439 f. Brightness Decrease"""
440 # (using brightness increase functions)
441
442 print("\n|f. BrightnessDecrease...", end=' ')
443 brightness_offsets = [0, -5, -10, -15, -20, -25, -30, -35, -40, -45]
444 dice_scores = evaluate_brightness_change(model, images_dir, masks_dir, brightness_offsets, device, save_as="f_brightness_decrease", invert_axis=True)
445 print("Done!")
446 print("*"*49)
447 print("|bright_offset=", "|".join([f"value:{digits+2}" for value in brightness_offsets]), "|")
448 print("|meanDiceScore=", "|".join([f"d:{digits+2}.{digits}f" for d in dice_scores]), "|")
449 print("*"*49, "\n")
450
451
452 """
453 g. Square occlusion"""
454
455 def apply_occlusion(image, square_size):
456     """Applies occlusion by replacing a random square region with black pixels."""
457     h, w, _ = image.shape
458     if square_size > 0:
459         x = np.random.randint(0, w - square_size + 1)
460         y = np.random.randint(0, h - square_size + 1)
461         image[y:y+square_size, x:x+square_size] = 0
462     return image
463
464 def evaluate_occlusion(model, images_dir, masks_dir, occlusion_sizes, device, save_as:str="g_square_occlusion"):
465     """Evaluates the model under increasing occlusion perturbation."""
466     image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]
467     dice_scores = []
468
469     for square_size in occlusion_sizes:

```

```

469     total_dice = []
470
471     counter = 0
472     for img_file in image_files:
473         counter += 1
474         if counter == maxImages:
475             break
476         img_path = os.path.join(images_dir, img_file)
477         mask_path = os.path.join(masks_dir, img_file.replace('.jpg', '.png'))
478         heatmap_path = os.path.join(heatmaps_dir, img_file.split('.')[0]+'_heatmap.png')
479
480         image = np.array(Image.open(img_path).convert("RGB").resize((500, 500)))
481         occluded_image = apply_occlusion(image, square_size)
482         image_tensor = transforms.ToTensor()(Image.fromarray(occluded_image)).unsqueeze(0).to(device)
483         heatmap = Image.open(heatmap_path).convert('L').resize((500, 500))
484         heatmap = transforms.ToTensor()(heatmap).unsqueeze(0).to(device)
485         image_tensor = torch.cat([image_tensor, heatmap], dim=1)
486
487         gt_mask = Image.open(mask_path).convert("L")
488         gt_mask = gt_mask.resize((256, 256), Image.NEAREST)
489         label_array = np.array(gt_mask)
490
491         label_tensor = torch.from_numpy(label_array).to(dtype=torch.uint8)
492         gt_mask_class = torch.zeros_like(label_tensor, dtype=torch.uint8)
493
494         for pixel_value, class_idx in pixel_to_class.items():
495             gt_mask_class[label_tensor == pixel_value] = class_idx
496
497         image_clip_tensor = preprocess_val(Image.open(img_path).convert("RGB")).unsqueeze(0).to(device)
498
499         with torch.no_grad():
500             clip_features = CLIP_model.encode_image(image_clip_tensor)
501             clip_features = clip_features / clip_features.norm(dim=-1, keepdim=True)
502             model_output = model(image_tensor, clip_features)
503             pred_mask = model_output.squeeze(0)
504             pred_mask = torch.argmax(pred_mask, dim=0).cpu()
505
506             _, mean_dice = compute_dice_coefficient(pred_mask, gt_mask_class, exclude_classes=excludedClasses)
507             total_dice.append(mean_dice)
508
509         dice_scores.append(np.mean(total_dice))
510
511     plt.figure()
512     plt.plot(occlusion_sizes, dice_scores, marker='o')
513     plt.xlabel("Occlusion Size (edge length in pixels)")
514     plt.ylabel("Mean Dice Score")
515     plt.title("Square Occlusion")
516     plt.grid()
517     plt.savefig(output_plots_dir + save_as + ".pdf", format="pdf")
518     return dice_scores
519
520 print("\nng) Square Occlusion...", end=' ')
521 occlusion_sizes = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
522 dice_scores = evaluate_occlusion(model, images_dir, masks_dir, occlusion_sizes, device)
523 print("Done!")
524 print(" " * 49)
525 print("|_oc|_square|size| =", "|_|".join([f"{value:{digits+2}}" for value in occlusion_sizes]), "|")
526 print("|_mean|Dice|score| =", "|_|".join([f"{d:{digits+2}.{digits}f}" for d in dice_scores]), "|")
527 print(" " * 49, "\n")
528
529 """
530     h. Salt&Pepper Noise"""
531
532 def apply_salt_and_pepper_noise(image, noise_amount):
533     """Applies salt and pepper noise to an image."""
534     noisy_image = random_noise(image, mode='s&p', amount=noise_amount)
535     noisy_image = (noisy_image * 255).astype(np.uint8)
536     return noisy_image
537
538 def evaluate_salt_and_pepper_noise(model, images_dir, masks_dir, noise_levels, device, save_as:str="h_SaP_noise"):
539     """Evaluates the model with salt-and-pepper noise perturbation and plots Dice score vs. noise level."""
540     image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]
541     dice_scores = []
542
543     for noise_amount in noise_levels:
544         total_dice = []
545
546         counter = 0
547         for img_file in image_files:
548             counter += 1
549             if counter == maxImages:
550                 break
551             img_path = os.path.join(images_dir, img_file)
552             mask_path = os.path.join(masks_dir, img_file.replace('.jpg', '.png'))
553             heatmap_path = os.path.join(heatmaps_dir, img_file.split('.')[0]+'_heatmap.png')
554
555             image = np.array(Image.open(img_path).convert("RGB").resize((500, 500)))
556             noisy_image = apply_salt_and_pepper_noise(image, noise_amount)
557             image_tensor = transforms.ToTensor()(Image.fromarray(noisy_image)).unsqueeze(0).to(device)
558             heatmap = Image.open(heatmap_path).convert('L').resize((500, 500))
559             heatmap = transforms.ToTensor()(heatmap).unsqueeze(0).to(device)
560             image_tensor = torch.cat([image_tensor, heatmap], dim=1)
561
562             gt_mask = Image.open(mask_path).convert("L")
563             gt_mask = gt_mask.resize((256, 256), Image.NEAREST)

```

```

564     label_array = np.array(gt_mask)
565
566     label_tensor = torch.from_numpy(label_array).to(dtype=torch.uint8)
567     gt_mask_class = torch.zeros_like(label_tensor, dtype=torch.uint8)
568
569     for pixel_value, class_idx in pixel_to_class.items():
570         gt_mask_class[label_tensor == pixel_value] = class_idx
571
572     image_clip_tensor = preprocess_val(Image.open(img_path).convert("RGB")).unsqueeze(0).to(device)
573
574     with torch.no_grad():
575         clip_features = CLIP_model.encode_image(image_clip_tensor)
576         clip_features = clip_features / clip_features.norm(dim=-1, keepdim=True)
577         model_output = model(image_tensor, clip_features)
578         pred_mask = model_output.squeeze(0)
579         pred_mask = torch.argmax(pred_mask, dim=0).cpu()
580
581     _, mean_dice = compute_dice_coefficient(pred_mask, gt_mask_class, exclude_classes=excludedClasses)
582     total_dice.append(mean_dice)
583
584     dice_scores.append(np.mean(total_dice))
585
586 plt.figure()
587 plt.plot(noise_levels, dice_scores, marker='o')
588 plt.xlabel("Salt & Pepper Noise Level")
589 plt.ylabel("Mean Dice Score")
590 plt.title("Salt & Pepper Noise")
591 plt.grid()
592 plt.savefig(output_plots_dir + save_as + ".pdf", format="pdf")
593
594 return dice_scores
595
596 print("\n\nSalt & Pepper Noise...")
597 noise_levels = [0.00, 0.02, 0.04, 0.06, 0.08, 0.10, 0.12, 0.14, 0.16, 0.18]
598 dice_scores = evaluate_salt_and_pepper_noise(model, images_dir, masks_dir, noise_levels, device)
599 print("Done!")
600 print("-" * 49)
601 print("S&P noise level=", " ".join([f"value:{value:2d}" for value in noise_levels]), "|")
602 print("mean Dice score=", " ".join([f"d:{dice_scores[d]:2d}" for d in range(len(dice_scores))]), "|")
603 print("-" * 49, "\n")
604
605 """
606     i. Imprecise prompt """
607
608 def shift_heatmap(heatmap, distance):
609     """Shifts the heatmap by a given distance in a random direction."""
610     h, w = heatmap.shape
611     angle = np.random.uniform(0, 2 * np.pi) # Random angle in radians
612     dx = int(distance * np.cos(angle)) # X shift
613     dy = int(distance * np.sin(angle)) # Y shift
614
615     translation_matrix = np.float32([[1, 0, dx], [0, 1, dy]])
616     shifted_heatmap = cv2.warpAffine(heatmap, translation_matrix, (w, h), borderValue=0)
617     return shifted_heatmap
618
619 def evaluate_heatmap_shift(model, images_dir, masks_dir, shift_distances, device, save_as="i_heatmap_shift"):
620     """Evaluates the model with shifting heatmaps and plots Dice score vs. shift distance."""
621     image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]
622     dice_scores = []
623
624     for distance in shift_distances:
625         total_dice = []
626
627         counter = 0
628         for img_file in image_files:
629             counter += 1
630             if counter == maxImages:
631                 break
632             img_path = os.path.join(images_dir, img_file)
633             mask_path = os.path.join(masks_dir, img_file.replace('.jpg', '.png'))
634             heatmap_path = os.path.join(heatmaps_dir, img_file.split('.')[0] + '_heatmap.png')
635
636             image = np.array(Image.open(img_path).convert("RGB").resize((500, 500)))
637             image_tensor = transforms.ToTensor()(Image.fromarray(image)).unsqueeze(0).to(device)
638
639             heatmap = np.array(Image.open(heatmap_path).convert('L').resize((500, 500)))
640             shifted_heatmap = shift_heatmap(heatmap, distance)
641             heatmap_tensor = transforms.ToTensor()(Image.fromarray(shifted_heatmap)).unsqueeze(0).to(device)
642
643             image_tensor = torch.cat([image_tensor, heatmap_tensor], dim=1)
644
645             gt_mask = Image.open(mask_path).convert("L").resize((256, 256), Image.NEAREST)
646             label_array = np.array(gt_mask)
647
648             label_tensor = torch.from_numpy(label_array).to(dtype=torch.uint8)
649             gt_mask_class = torch.zeros_like(label_tensor, dtype=torch.uint8)
650
651             for pixel_value, class_idx in pixel_to_class.items():
652                 gt_mask_class[label_tensor == pixel_value] = class_idx
653
654             image_clip_tensor = preprocess_val(Image.open(img_path).convert("RGB")).unsqueeze(0).to(device)
655
656             with torch.no_grad():
657                 clip_features = CLIP_model.encode_image(image_clip_tensor)
658                 clip_features = clip_features / clip_features.norm(dim=-1, keepdim=True)

```

```

659     model_output = model(image_tensor, clip_features)
660     pred_mask = model_output.squeeze(0)
661     pred_mask = torch.argmax(pred_mask, dim=0).cpu()
662
663     _, mean_dice = compute_dice_coefficient(pred_mask, gt_mask_class, exclude_classes=excludedClasses)
664     total_dice.append(mean_dice)
665
666     dice_scores.append(np.mean(total_dice))
667
668     plt.figure()
669     plt.plot(shift_distances, dice_scores, marker='o')
670     plt.xlabel("Heatmap Shift (pixels)")
671     plt.ylabel("Mean Dice Score")
672     plt.title("Imprecise prompt")
673     plt.grid()
674     plt.savefig(output_plots_dir + save_as + ".pdf", format="pdf")
675
676     return dice_scores
677
678 print("\n ni) Imprecise prompt...", end=' ')
679 shift_distances = [0, 25, 50, 75, 100, 125, 150, 200, 250, 300]
680 dice_scores = evaluate_heatmap_shift(model, images_dir, masks_dir, shift_distances, device)
681 print("Done!")
682 print("*" * 49)
683 print(" heatmap shift = ", " ".join([f"{value:{digits+2}}" for value in shift_distances]), "|")
684 print("mean Dice score = ", " ".join([f"{d:{digits+2}.{digits}f}" for d in dice_scores]), "|")
685 print("*" * 49, "\n")

```

### A.3.10 robustness\_exploration\_prompt\_clip.py: Robustness-Exploration for Prompt Clip

```

1 # Authors: Sahil Memul Bavishi (s2677266) and Matteo Spadaccia (s2748897)
2 # Subject: Computer Vision Coursework. Robustness exploration on the prompted CLIP model
3 # Date: 31.03.2025
4
5 """0. Preliminary code"""
6
7 print("ROBUSTNESS_EXPLORATION_WITH_DIFFERENT_PERTURBATION_TYPES(PROMPTED_CLIP):")
8
9 # Setting code-behaviour variables
10 classNum = 4
11 pixel_to_class = {0: 0, 38: 1, 75: 2, 255: 3}
12 excludedClasses = [3] # classes excluded from metric calculation (3 not to consider the outline)
13 maxImages = -1 # number of input images to load from the defined set (0 to consider them all)
14 digits = 3 # digits to approximate decimal results in, when printed
15
16 # Defining paths
17 model_dir = f'Data/Output/Models2/PROMPTED_CLIP_DICE2_epoch_42.pth'
18 images_dir = 'Data/Input/Test/color/'
19 masks_dir = 'Data/Input/Test/label/'
20 heatmaps_dir = 'Data/Input/Test/heatmaps/'
21 output_plots_dir = 'Data/Output/Visuals/Prompted_CLIP_Robustness/'
22
23 # Importing useful libraries
24 import os
25 import numpy as np
26 import torch
27 import torch.nn as nn
28 from torchvision import transforms
29 from PIL import Image
30 import open_clip
31 import matplotlib.pyplot as plt
32 import cv2
33 from skimage.util import random_noise
34
35 device = "cuda" if torch.cuda.is_available() else "cpu"
36
37 image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]
38
39 # Loading pre-trained prompted CLIP model
40 CLIP_model, preprocess_train, preprocess_val = open_clip.create_model_and_transforms("ViT-B-32", pretrained="openai")
41 CLIP_model = CLIP_model.to(device)
42 tokenizer = open_clip.get_tokenizer("ViT-B-32")
43
44 class PromptedClipDecoder(nn.Module):
45     def __init__(self, clip_feature_dim=512, num_classes=classNum):
46         super(PromptedClipDecoder, self).__init__()
47
48         # Project CLIP features
49         self.clip_fc = nn.Linear(clip_feature_dim, 256)
50
51         # CNN Feature Extractor (with 4 input channels: RGB + Heatmap)
52         self.encoder = nn.Sequential(
53             nn.Conv2d(4, 64, kernel_size=3, stride=1, padding=1),
54             nn.ReLU(),
55             nn.MaxPool2d(2),
56             nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
57             nn.ReLU(),
58         )
59
60         # Decoder
61

```

```

62     self.decoder = nn.Sequential(
63         nn.ConvTranspose2d(384, 64, kernel_size=3, stride=1, padding=1),
64         nn.ReLU(),
65         nn.Upsample(size=(256, 256), mode='bilinear', align_corners=True),
66         nn.ConvTranspose2d(64, num_classes, kernel_size=3, stride=1, padding=1),
67     )
68
69     def forward(self, image, clip_features):
70         cnn_features = self.encoder(image)
71         clip_features = self.clip_fc(clip_features).view(clip_features.shape[0], 256, 1, 1)
72         clip_features = clip_features.expand(-1, -1, cnn_features.shape[2], cnn_features.shape[3])
73         fusion = torch.cat([cnn_features, clip_features], dim=1)
74         segmentation_output = self.decoder(fusion)
75         return segmentation_output
76
77     # Loading segmentation model
78     model = PromptedClipDecoder().to(device)
79     model.load_state_dict(torch.load(model_dir, map_location=device))
80     model.eval()
81
82     def compute_dice_coefficient(pred, target, num_classes=classNum, exclude_classes=[]):
83         """
84             Compute Dice Coefficient for multi-class segmentation with the option to exclude specific classes.
85
86             :param pred: Predicted segmentation map (H, W), with class indices.
87             :param target: Ground truth segmentation map (H, W), with class indices.
88             :param num_classes: Number of classes in segmentation.
89             :param exclude_classes: List of class indices to exclude from Dice calculation.
90             :return: Dice scores per class and mean Dice score.
91         """
92
93         dice_per_class = []
94         for cls in range(num_classes):
95             if cls in exclude_classes:
96                 continue
97
98             pred_inds = pred == cls
99             target_inds = target == cls
100            intersection = 2.0 * torch.logical_and(pred_inds, target_inds).sum().item()
101            union = pred_inds.sum().item() + target_inds.sum().item()
102
103            if union == 0:
104                dice_per_class.append(float('nan'))
105            else:
106                dice_per_class.append(intersection / union)
107
108            mean_dice = np.nanmean(dice_per_class) if dice_per_class else float('nan')
109
110        return dice_per_class, mean_dice
111
112
113    """a. Gaussian noise"""
114
115    def apply_gaussian_noise(image, std_dev):
116        """Applies Gaussian noise to an image with a given standard deviation."""
117        noise = np.random.normal(0, std_dev, image.shape)
118        noisy_image = np.clip(image + noise, 0, 255).astype(np.uint8)
119        return noisy_image
120
121    def evaluate_gaussian_noise(model, images_dir, masks_dir, std_dev_levels, device, save_as:str="a_gaussian_noise"):
122        """Evaluates the model with Gaussian noise perturbation and plots Dice score vs. noise level."""
123        image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]
124        dice_scores = []
125
126        for std_dev in std_dev_levels:
127            total_dice = []
128            counter = 0
129            for img_file in image_files:
130                counter += 1
131                if counter == maxImages:
132                    break
133                img_path = os.path.join(images_dir, img_file)
134                mask_path = os.path.join(masks_dir, img_file.replace('.jpg', '.png'))
135                heatmap_path = os.path.join(heatmaps_dir, img_file.split('.')[0]+'_heatmap.png')
136
137                image = np.array(Image.open(img_path).convert("RGB").resize((500, 500)))
138                noisy_image = apply_gaussian_noise(image, std_dev)
139                image_tensor = transforms.ToTensor()(Image.fromarray(noisy_image)).unsqueeze(0).to(device)
140                heatmap = Image.open(heatmap_path).convert('L').resize((500, 500))
141                heatmap = transforms.ToTensor()(heatmap).unsqueeze(0).to(device)
142                image_tensor = torch.cat([image_tensor, heatmap], dim=1)
143
144                gt_mask = Image.open(mask_path).convert("L")
145                gt_mask = gt_mask.resize((256, 256), Image.NEAREST)
146                label_array = np.array(gt_mask)
147
148                label_tensor = torch.from_numpy(label_array).to(dtype=torch.uint8)
149                gt_mask_class = torch.zeros_like(label_tensor, dtype=torch.uint8)
150
151                for pixel_value, class_idx in pixel_to_class.items():
152                    gt_mask_class[label_tensor == pixel_value] = class_idx
153
154                image_clip_tensor = preprocess_val(Image.open(img_path).convert("RGB")).unsqueeze(0).to(device)
155
156                with torch.no_grad():

```

```

157     clip_features = CLIP_model.encode_image(image_clip_tensor)
158     clip_features = clip_features / clip_features.norm(dim=-1, keepdim=True)
159     model_output = model(image_tensor, clip_features)
160     pred_mask = model_output.squeeze(0)
161     pred_mask = torch.argmax(pred_mask, dim=0).cpu()
162
163     _, mean_dice = compute_dice_coefficient(pred_mask, gt_mask_class, exclude_classes = excludedClasses)
164
165     total_dice.append(mean_dice)
166
167     dice_scores.append(np.mean(total_dice))
168
169 plt.figure()
170 plt.plot(std_dev_levels, dice_scores, marker='o')
171 plt.xlabel("Gaussian Noise Standard Deviation")
172 plt.ylabel("Mean Dice Score")
173 plt.title("Gaussian Noise")
174 plt.grid()
175 plt.savefig(output_plots_dir + save_as + ".pdf", format="pdf")
176
177 return dice_scores
178
179 print("\nna) Gaussian Noise...", end=' ')
180 std_dev_levels = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
181 dice_scores = evaluate_gaussian_noise(model, images_dir, masks_dir, std_dev_levels, device)
182 print("Done!")
183 print(" "*49)
184 print(" | std_deviations = ", " | ".join([f"value:{d:d+2}" for value in std_dev_levels]), "|")
185 print(" | mean_Dice_Score = ", " | ".join([f"d:{d:d+2}.f" for d in dice_scores]), "|")
186 print(" "*49, "\n")
187
188 """
189 b. Gaussian blurring """
190
191 def apply_gaussian_blur(image, ksize):
192     """Applies Gaussian blur to an image with a given kernel size."""
193     if ksize % 2 == 0:
194         ksize += 1
195     kernel = np.array([
196         [1/16, 2/16, 1/16],
197         [2/16, 4/16, 2/16],
198         [1/16, 2/16, 1/16]
199     ])
200     blurred_image = cv2.filter2D(image, -1, kernel)
201     return blurred_image
202
203 def evaluate_gaussian_blur(model, images_dir, masks_dir, max.blur_level, device, save_as:str="b_gaussian_blurring"):
204     """Evaluates the model with Gaussian blurring perturbation and plots Dice score vs. blur level."""
205     image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]
206     dice_scores = []
207
208     for blur_level in range(max.blur_level + 1):
209         total_dice = []
210
211         counter = 0
212         for img_file in image_files:
213             counter += 1
214             if counter == maxImages:
215                 break
216             img_path = os.path.join(images_dir, img_file)
217             mask_path = os.path.join(masks_dir, img_file.replace('.jpg', '.png'))
218             heatmap_path = os.path.join(heatmaps_dir, img_file.split('.')[0]+'_heatmap.png')
219
220             blurred_image = np.array(Image.open(img_path).convert("RGB").resize((500, 500)))
221             for i in range(blur_level):
222                 blurred_image = apply_gaussian_blur(blurred_image, 3)
223             image_tensor = transforms.ToTensor()(Image.fromarray(blurred_image)).unsqueeze(0).to(device)
224             heatmap = Image.open(heatmap_path).convert('L').resize((500, 500))
225             heatmap = transforms.ToTensor()(heatmap).unsqueeze(0).to(device)
226             image_tensor = torch.cat([image_tensor, heatmap], dim=1)
227
228             gt_mask = Image.open(mask_path).convert("L")
229             gt_mask = gt_mask.resize((256, 256), Image.NEAREST)
230             label_array = np.array(gt_mask)
231
232             label_tensor = torch.from_numpy(label_array).to(dtype=torch.uint8)
233             gt_mask_class = torch.zeros_like(label_tensor, dtype=torch.uint8)
234
235             for pixel_value, class_idx in pixel_to_class.items():
236                 gt_mask_class[label_tensor == pixel_value] = class_idx
237
238             image_clip_tensor = preprocess_val(Image.open(img_path).convert("RGB")).unsqueeze(0).to(device)
239
240             with torch.no_grad():
241                 clip_features = CLIP_model.encode_image(image_clip_tensor)
242                 clip_features = clip_features / clip_features.norm(dim=-1, keepdim=True)
243                 model_output = model(image_tensor, clip_features)
244                 pred_mask = model_output.squeeze(0)
245                 pred_mask = torch.argmax(pred_mask, dim=0).cpu()
246
247                 _, mean_dice = compute_dice_coefficient(pred_mask, gt_mask_class, exclude_classes = excludedClasses)
248                 total_dice.append(mean_dice)
249
250     dice_scores.append(np.mean(total_dice))

```

```

252     plt.figure()
253     plt.plot(range(max_blur_level + 1), dice_scores, marker='o')
254     plt.xlabel("GaussianBlurLevel(number of iterative 3x3 mask applications)")
255     plt.ylabel("MeanDiceScore")
256     plt.title("GaussianBlurring")
257     plt.grid()
258     plt.savefig(output_plots_dir + save_as + ".pdf", format="pdf")
259
260     return dice_scores
261
262 print("\nb) GaussianBlurring...", end=' ')
263 max.blur.level = 9
264 dice_scores = evaluate_gaussian_blur(model, images_dir, masks_dir, max.blur.level, device)
265 print("Done!")
266 print("*" * 49)
267 print(f"blurring steps={", " ".join([f"value:{value:2d}" for value in list(range(max.blur.level+1))]), "|")
268 print(f"mean Dice score={", " ".join([f"d:{dice_scores[d]:2d}" for d in dice_scores]), "|")
269 print("*" * 49, "\n")
270
271 """
272     c. Contrast increase"""
273
274 def apply_contrast_change(image, contrast_factor):
275     """Applies contrast increase to an image by multiplying pixel values by a contrast factor."""
276     contrast_image = image * contrast_factor
277     contrast_image = np.clip(contrast_image, 0, 255).astype(np.uint8)
278     return contrast_image
279
280 def evaluate_contrast_change(model, images_dir, masks_dir, contrast_factors, device, save_as:str="c_contrast_increase",
281     , invert_axis:bool=False):
282     """Evaluates the model with increased contrast perturbation and plots Dice score vs. contrast factor."""
283     image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]
284     dice_scores = []
285
286     for contrast_factor in contrast_factors:
287         total_dice = []
288
289         counter = 0
290         for img_file in image_files:
291             counter += 1
292             if counter == maxImages:
293                 break
294             img_path = os.path.join(images_dir, img_file)
295             mask_path = os.path.join(masks_dir, img_file.replace('.jpg', '.png'))
296             heatmap_path = os.path.join(heatmaps_dir, img_file.split('.')[0]+'_heatmap.png')
297
298             image = np.array(Image.open(img_path).convert("RGB").resize((500, 500)))
299             contrast_image = apply_contrast_change(image, contrast_factor)
300             image_tensor = transforms.ToTensor()(Image.fromarray(contrast_image)).unsqueeze(0).to(device)
301             heatmap = Image.open(heatmap_path).convert('L').resize((500, 500))
302             heatmap = transforms.ToTensor()(heatmap).unsqueeze(0).to(device)
303             image_tensor = torch.cat([image_tensor, heatmap], dim=1)
304
305             gt_mask = Image.open(mask_path).convert("L")
306             gt_mask = gt_mask.resize((256, 256), Image.NEAREST)
307             label_array = np.array(gt_mask)
308
309             label_tensor = torch.from_numpy(label_array).to(dtype=torch.uint8)
310             gt_mask_class = torch.zeros_like(label_tensor, dtype=torch.uint8)
311
312             for pixel_value, class_idx in pixel_to_class.items():
313                 gt_mask_class[label_tensor == pixel_value] = class_idx
314
315             image_clip_tensor = preprocess_val(Image.open(img_path).convert("RGB")).unsqueeze(0).to(device)
316
317             with torch.no_grad():
318                 clip_features = CLIP_model.encode_image(image_clip_tensor)
319                 clip_features = clip_features / clip_features.norm(dim=-1, keepdim=True)
320                 model_output = model(image_tensor, clip_features)
321                 pred_mask = model_output.squeeze(0)
322                 pred_mask = torch.argmax(pred_mask, dim=0).cpu()
323
324                 _, mean_dice = compute_dice_coefficient(pred_mask, gt_mask_class, exclude_classes = excludedClasses)
325                 total_dice.append(mean_dice)
326
327             dice_scores.append(np.mean(total_dice))
328
329     plt.figure()
330     plt.plot(contrast_factors, dice_scores, marker='o')
331     plt.xlabel("ContrastFactor")
332     plt.ylabel("MeanDiceScore")
333     plt.title("ContrastDecrease" if invert_axis else "ContrastIncrease")
334     plt.grid()
335     if invert_axis: plt.gca().invert_xaxis()
336     plt.savefig(output_plots_dir + save_as + ".pdf", format="pdf")
337
338     return dice_scores
339
340 print("\nc) Contrast Increase...", end=' ')
341 contrast_factors = [1.0, 1.01, 1.02, 1.03, 1.04, 1.05, 1.1, 1.15, 1.2, 1.25]
342 dice_scores = evaluate_contrast_change(model, images_dir, masks_dir, contrast_factors, device)
343 print("Done!")
344 print("*" * 49)
345 print(f"contrast factor={", " ".join([f"value:{value:2d}" for value in contrast_factors]), "|")
346 print(f"mean Dice score={", " ".join([f"d:{dice_scores[d]:2d}" for d in dice_scores]), "|")

```

```

346 | print("—" * 49, "\n")
347 |
348 |
349 """d. Contrast decrease"""
350 # (using contrast increase functions)
351
352 print("\nd) ContrastDecrease...", end=' ')
353 contrast_factors = [1.0, 0.95, 0.9, 0.85, 0.8, 0.6, 0.4, 0.3, 0.2, 0.1]
354 dice_scores = evaluate_contrast_change(model, images_dir, masks_dir, contrast_factors, device, save_as="d_contrast_decrease", invert_axis=True)
355 print("Done!")
356 print("—" * 49)
357 print("l)contrastfactor=", "l".join([f"value:{digits+2}" for value in contrast_factors]), "|")
358 print("l)meanDiceScore=", "l".join([f"d:{digits}f" for d in dice_scores]), "|")
359 print("—" * 49, "\n")
360
361
362 """e. Brightness Increase"""
363
364 def apply_brightness_change(image, brightness_offset):
365     """Applies brightness change by adding an offset to pixel values."""
366     bright_image = image.astype(np.int16) + brightness_offset
367     bright_image = np.clip(bright_image, 0, 255).astype(np.uint8)
368     return bright_image
369
370 def evaluate_brightness_change(model, images_dir, masks_dir, brightness_offsets, device, save_as:str="e_brightness_increase", invert_axis:bool=False):
371     """Evaluates the model with brightness perturbation and plots Dice score vs. brightness offset."""
372     image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]
373     dice_scores = []
374
375     for brightness_offset in brightness_offsets:
376         total_dice = []
377
378         counter = 0
379         for img_file in image_files:
380             counter += 1
381             if counter == maxImages:
382                 break
383             img_path = os.path.join(images_dir, img_file)
384             mask_path = os.path.join(masks_dir, img_file.replace('.jpg', '.png'))
385             heatmap_path = os.path.join(heatmaps_dir, img_file.split('.')[0]+'_heatmap.png')
386
387             image = np.array(Image.open(img_path).convert("RGB").resize((500, 500)))
388             bright_image = apply_brightness_change(image, brightness_offset)
389             image_tensor = transforms.ToTensor()(Image.fromarray(bright_image)).unsqueeze(0).to(device)
390             heatmap = Image.open(heatmap_path).convert('L').resize((500, 500))
391             heatmap = transforms.ToTensor()(heatmap).unsqueeze(0).to(device)
392             image_tensor = torch.cat([image_tensor, heatmap], dim=1)
393
394             gt_mask = Image.open(mask_path).convert("L")
395             gt_mask = gt_mask.resize((256, 256), Image.NEAREST)
396             label_array = np.array(gt_mask)
397
398             label_tensor = torch.from_numpy(label_array).to(dtype=torch.uint8)
399             gt_mask_class = torch.zeros_like(label_tensor, dtype=torch.uint8)
400
401             for pixel_value, class_idx in pixel_to_class.items():
402                 gt_mask_class[label_tensor == pixel_value] = class_idx
403
404             image_clip_tensor = preprocess_val(Image.open(img_path).convert("RGB")).unsqueeze(0).to(device)
405
406             with torch.no_grad():
407                 clip_features = CLIP_model.encode_image(image_clip_tensor)
408                 clip_features = clip_features / clip_features.norm(dim=-1, keepdim=True)
409                 model_output = model(image_tensor, clip_features)
410                 pred_mask = model_output.squeeze(0)
411                 pred_mask = torch.argmax(pred_mask, dim=0).cpu()
412
413             _, mean_dice = compute_dice_coefficient(pred_mask, gt_mask_class, exclude_classes=excludedClasses)
414             total_dice.append(mean_dice)
415
416             dice_scores.append(np.mean(total_dice))
417
418             plt.figure()
419             plt.plot(brightness_offsets, dice_scores, marker='o')
420             plt.xlabel("Brightness_Offset")
421             plt.ylabel("Mean_Dice_Score")
422             plt.title("BrightnessDecrease" if invert_axis else "BrightnessIncrease")
423             plt.grid()
424             if invert_axis: plt.gca().invert_xaxis()
425             plt.savefig(output_plots_dir + save_as + ".pdf", format="pdf")
426
427             return dice_scores
428
429 print("\ne)BrightnessIncrease...", end=' ')
430 brightness_offsets = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
431 dice_scores = evaluate_brightness_change(model, images_dir, masks_dir, brightness_offsets, device)
432 print("Done!")
433 print("—" * 49)
434 print("l)bright_offset=", "l".join([f"value:{digits+2}" for value in brightness_offsets]), "|")
435 print("l)meanDiceScore=", "l".join([f"d:{digits}f" for d in dice_scores]), "|")
436 print("—" * 49, "\n")

```

```

439 """f. Brightness Decrease"""
440 # (using brightness increase functions)
441
442 print("\n\nf. BrightnessDecrease...", end=' ')
443 brightness_offsets = [0, -5, -10, -15, -20, -25, -30, -35, -40, -45]
444 dice_scores = evaluate_brightness_change(model, images_dir, masks_dir, brightness_offsets, device, save_as="f_brightness_decrease", invert_axis=True)
445 print("Done!")
446 print("-" * 49)
447 print(f"bright_offset={brightness_offsets}, ".join([f"value:{v:2d}" for v in brightness_offsets]), "|")
448 print(f"meanDiceScore={dice_scores}, ".join([f"{{d:{v:2d}}.{v:2d}}" for d in dice_scores]), "|")
449 print("-" * 49, "\n")
450
451
452 """g. Square occlusion"""
453
454 def apply_occlusion(image, square_size):
455     """Applies occlusion by replacing a random square region with black pixels."""
456     h, w, _ = image.shape
457     if square_size > 0:
458         x = np.random.randint(0, w - square_size + 1)
459         y = np.random.randint(0, h - square_size + 1)
460         image[y:y+square_size, x:x+square_size] = 0
461     return image
462
463 def evaluate_occlusion(model, images_dir, masks_dir, occlusion_sizes, device, save_as:str="g_square_occlusion"):
464     """Evaluates the model under increasing occlusion perturbation."""
465     image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]
466     dice_scores = []
467
468     for square_size in occlusion_sizes:
469         total_dice = []
470
471         counter = 0
472         for img_file in image_files:
473             counter += 1
474             if counter == maxImages:
475                 break
476             img_path = os.path.join(images_dir, img_file)
477             mask_path = os.path.join(masks_dir, img_file.replace('.jpg', '.png'))
478             heatmap_path = os.path.join(heatmaps_dir, img_file.split('.')[0] + '_heatmap.png')
479
480             image = np.array(Image.open(img_path).convert("RGB").resize((500, 500)))
481             occluded_image = apply_occlusion(image, square_size)
482             image_tensor = transforms.ToTensor()(Image.fromarray(occluded_image)).unsqueeze(0).to(device)
483             heatmap = Image.open(heatmap_path).convert('L').resize((500, 500))
484             heatmap = transforms.ToTensor()(heatmap).unsqueeze(0).to(device)
485             image_tensor = torch.cat([image_tensor, heatmap], dim=1)
486
487             gt_mask = Image.open(mask_path).convert("L")
488             gt_mask = gt_mask.resize((256, 256), Image.NEAREST)
489             label_array = np.array(gt_mask)
490
491             label_tensor = torch.from_numpy(label_array).to(dtype=torch.uint8)
492             gt_mask_class = torch.zeros_like(label_tensor, dtype=torch.uint8)
493
494             for pixel_value, class_idx in pixel_to_class.items():
495                 gt_mask_class[label_tensor == pixel_value] = class_idx
496
497             image_clip_tensor = preprocess_val(Image.open(img_path).convert("RGB")).unsqueeze(0).to(device)
498
499             with torch.no_grad():
500                 clip_features = CLIP_model.encode_image(image_clip_tensor)
501                 clip_features = clip_features / clip_features.norm(dim=-1, keepdim=True)
502                 model_output = model(image_tensor, clip_features)
503                 pred_mask = model_output.squeeze(0)
504                 pred_mask = torch.argmax(pred_mask, dim=0).cpu()
505
506                 _, mean_dice = compute_dice_coefficient(pred_mask, gt_mask_class, exclude_classes=excludedClasses)
507                 total_dice.append(mean_dice)
508
509             dice_scores.append(np.mean(total_dice))
510
511             plt.figure()
512             plt.plot(occlusion_sizes, dice_scores, marker='o')
513             plt.xlabel("OcclusionSize_(edge_length_in_pixels)")
514             plt.ylabel("MeanDiceScore")
515             plt.title("Square Occlusion")
516             plt.grid()
517             plt.savefig(output_plots_dir + save_as + ".pdf", format="pdf")
518         return dice_scores
519
520 print("\nng. SquareOcclusion...", end=' ')
521 occlusion_sizes = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
522 dice_scores = evaluate_occlusion(model, images_dir, masks_dir, occlusion_sizes, device)
523 print("Done!")
524 print("-" * 49)
525 print(f"oc_size={occlusion_sizes}, ".join([f"value:{v:2d}" for v in occlusion_sizes]), "|")
526 print(f"meanDiceScore={dice_scores}, ".join([f"{{d:{v:2d}}.{v:2d}}" for d in dice_scores]), "|")
527 print("-" * 49, "\n")
528
529
530 """h. Salt&Pepper Noise"""
531
532 def apply_salt_and_pepper_noise(image, noise_amount):

```

```

533     """Applies salt and pepper noise to an image."""
534     noisy_image = random_noise(image, mode='s&p', amount=noise_amount)
535     noisy_image = (noisy_image * 255).astype(np.uint8)
536     return noisy_image
537
538 def evaluate_salt_and_pepper_noise(model, images_dir, masks_dir, noise_levels, device, save_as:str="h_SaP_noise"):
539     """Evaluates the model with salt-and-pepper noise perturbation and plots Dice score vs. noise level."""
540     image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]
541     dice_scores = []
542
543     for noise_amount in noise_levels:
544         total_dice = []
545
546         counter = 0
547         for img_file in image_files:
548             counter += 1
549             if counter == maxImages:
550                 break
551             img_path = os.path.join(images_dir, img_file)
552             mask_path = os.path.join(masks_dir, img_file.replace('.jpg', '.png'))
553             heatmap_path = os.path.join(heatmaps_dir, img_file.split('.')[0]+'_heatmap.png')
554
555             image = np.array(Image.open(img_path).convert("RGB")).resize((500, 500))
556             noisy_image = apply_salt_and_pepper_noise(image, noise_amount)
557             image_tensor = transforms.ToTensor()(Image.fromarray(noisy_image)).unsqueeze(0).to(device)
558             heatmap = Image.open(heatmap_path).convert('L').resize((500, 500))
559             heatmap = transforms.ToTensor()(heatmap).unsqueeze(0).to(device)
560             image_tensor = torch.cat([image_tensor, heatmap], dim=1)
561
562             gt_mask = Image.open(mask_path).convert("L")
563             gt_mask = gt_mask.resize((256, 256), Image.NEAREST)
564             label_array = np.array(gt_mask)
565
566             label_tensor = torch.from_numpy(label_array).to(dtype=torch.uint8)
567             gt_mask_class = torch.zeros_like(label_tensor, dtype=torch.uint8)
568
569             for pixel_value, class_idx in pixel_to_class.items():
570                 gt_mask_class[label_tensor == pixel_value] = class_idx
571
572             image_clip_tensor = preprocess_val(Image.open(img_path).convert("RGB")).unsqueeze(0).to(device)
573
574             with torch.no_grad():
575                 clip_features = CLIP_model.encode_image(image_clip_tensor)
576                 clip_features = clip_features / clip_features.norm(dim=-1, keepdim=True)
577                 model_output = model(image_tensor, clip_features)
578                 pred_mask = model_output.squeeze(0)
579                 pred_mask = torch.argmax(pred_mask, dim=0).cpu()
580
581             _, mean_dice = compute_dice_coefficient(pred_mask, gt_mask_class, exclude_classes=excludedClasses)
582             total_dice.append(mean_dice)
583
584             dice_scores.append(np.mean(total_dice))
585
586             plt.figure()
587             plt.plot(noise_levels, dice_scores, marker='o')
588             plt.xlabel("Salt & Pepper Noise Level")
589             plt.ylabel("Mean Dice Score")
590             plt.title("Salt & Pepper Noise")
591             plt.grid()
592             plt.savefig(output_plots_dir + save_as + ".pdf", format="pdf")
593
594     return dice_scores
595
596 print("\n\nh) Salt & Pepper Noise...")
597 noise_levels = [0.0, 0.02, 0.04, 0.06, 0.08, 0.1, 0.12, 0.14, 0.16, 0.18]
598 dice_scores = evaluate_salt_and_pepper_noise(model, images_dir, masks_dir, noise_levels, device)
599 print("Done!")
600 print("-" * 49)
601 print(f"S&P noise level = {noise_levels}, mean Dice score = {dice_scores}")
602 print(f"mean Dice score = {np.mean(dice_scores)}")
603 print("-" * 49, "\n")
604
605 """
606     Imprecise prompt"""
607
608 def shift_heatmap(heatmap, distance):
609     """Shifts the heatmap by a given distance in a random direction."""
610     h, w = heatmap.shape
611     angle = np.random.uniform(0, 2 * np.pi) # Random angle in radians
612     dx = int(distance * np.cos(angle)) # X shift
613     dy = int(distance * np.sin(angle)) # Y shift
614
615     translation_matrix = np.float32([[1, 0, dx], [0, 1, dy]])
616     shifted_heatmap = cv2.warpAffine(heatmap, translation_matrix, (w, h), borderValue=0)
617     return shifted_heatmap
618
619 def evaluate_heatmap_shift(model, images_dir, masks_dir, shift_distances, device, save_as="i_heatmap_shift"):
620     """Evaluates the model with shifting heatmaps and plots Dice score vs. shift distance."""
621     image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]
622     dice_scores = []
623
624     for distance in shift_distances:
625         total_dice = []
626
627         counter = 0

```

```

628     for img_file in image_files:
629         counter += 1
630         if counter == maxImages:
631             break
632         img_path = os.path.join(images_dir, img_file)
633         mask_path = os.path.join(masks_dir, img_file.replace('.jpg', '.png'))
634         heatmap_path = os.path.join(heatmaps_dir, img_file.split('.')[0] + '_heatmap.png')
635
636         image = np.array(Image.open(img_path).convert("RGB").resize((500, 500)))
637         image_tensor = transforms.ToTensor()(Image.fromarray(image)).unsqueeze(0).to(device)
638
639         heatmap = np.array(Image.open(heatmap_path).convert('L').resize((500, 500)))
640         shifted_heatmap = shift_heatmap(heatmap, distance)
641         heatmap_tensor = transforms.ToTensor()(Image.fromarray(shifted_heatmap)).unsqueeze(0).to(device)
642
643         image_tensor = torch.cat([image_tensor, heatmap_tensor], dim=1)
644
645         gt_mask = Image.open(mask_path).convert("L").resize((256, 256), Image.NEAREST)
646         label_array = np.array(gt_mask)
647
648         label_tensor = torch.from_numpy(label_array).to(dtype=torch.uint8)
649         gt_mask_class = torch.zeros_like(label_tensor, dtype=torch.uint8)
650
651         for pixel_value, class_idx in pixel_to_class.items():
652             gt_mask_class[label_tensor == pixel_value] = class_idx
653
654         image_clip_tensor = preprocess_val(Image.open(img_path).convert("RGB")).unsqueeze(0).to(device)
655
656         with torch.no_grad():
657             clip_features = CLIP_model.encode_image(image_clip_tensor)
658             clip_features = clip_features / clip_features.norm(dim=-1, keepdim=True)
659             model_output = model(image_tensor, clip_features)
660             pred_mask = model_output.squeeze(0)
661             pred_mask = torch.argmax(pred_mask, dim=0).cpu()
662
663             _, mean_dice = compute_dice_coefficient(pred_mask, gt_mask_class, exclude_classes=excludedClasses)
664             total_dice.append(mean_dice)
665
666         dice_scores.append(np.mean(total_dice))
667
668     plt.figure()
669     plt.plot(shift_distances, dice_scores, marker='o')
670     plt.xlabel("Heatmap_Shift_(pixels)")
671     plt.ylabel("Mean_Dice_Score")
672     plt.title("Imprecise_prompt")
673     plt.grid()
674     plt.savefig(output_plots_dir + save_as + ".pdf", format="pdf")
675
676     return dice_scores
677
678 print("\n ni ) Imprecise prompt... ", end=' ')
679 shift_distances = [0, 25, 50, 75, 100, 125, 150, 200, 250, 300]
680 dice_scores = evaluate_heatmap_shift(model, images_dir, masks_dir, shift_distances, device)
681 print("Done!")
682 print("-" * 49)
683 print(" | heatmap shift = ", " | ".join([f"value:{value:2d}" for value in shift_distances]), "|")
684 print(" | mean Dice score = ", " | ".join([f"d:{dice_scores[d]:2d}" for d in range(len(dice_scores))]), "|")
685 print("-" * 49, "\n")

```

### A.3.11 flaskApp.py: Backend for UI

```

1  from flask import Flask, render_template, request, send_from_directory, jsonify
2  import os
3  from PIL import Image
4  import torch
5  import numpy as np
6  import torchvision.transforms as transforms
7  import streamlit as st
8  import torch
9  import torch.nn as nn
10 from torchvision import transforms
11 from PIL import Image
12 import open_clip
13 import numpy as np
14 import matplotlib.pyplot as plt
15
16
17
18 app = Flask(__name__)
19 app.config['UPLOAD_FOLDER'] = 'static'
20 classesNum = 4
21 def load_models():
22     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
23     clip_model, _, preprocess_val = open_clip.create_model_and_transforms("ViT-B-32", pretrained="openai")
24     clip_model = clip_model.to(device)
25
26     class ClipDecoder(nn.Module):
27         def __init__(self, clip_feature_dim=512, num_classes=classesNum):
28             super(ClipDecoder, self).__init__()
29
30             # Project CLIP features

```

```

31         self.clip_fc = nn.Linear(clip_feature_dim, 256)
32
33     # CNN Feature Extractor (Now 4 input channels: RGB + Heatmap)
34     self.encoder = nn.Sequential(
35         nn.Conv2d(4, 64, kernel_size=3, stride=1, padding=1),
36         nn.ReLU(),
37         nn.MaxPool2d(2),
38         nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
39         nn.ReLU(),
40     )
41
42     # Decoder
43     self.decoder = nn.Sequential(
44         nn.ConvTranspose2d(384, 64, kernel_size=3, stride=1, padding=1),
45         nn.ReLU(),
46         nn.Upsample(size=(256, 256), mode='bilinear', align_corners=True),
47         nn.ConvTranspose2d(64, num_classes, kernel_size=3, stride=1, padding=1),
48     )
49
50     def forward(self, image, clip_features):
51         cnn_features = self.encoder(image)
52         clip_features = self.clip_fc(clip_features).view(clip_features.shape[0], 256, 1, 1)
53         clip_features = clip_features.expand(-1, -1, cnn_features.shape[2], cnn_features.shape[3])
54         fusion = torch.cat([cnn_features, clip_features], dim=1)
55         segmentation_output = self.decoder(fusion)
56
57         return segmentation_output
58
59     model_path = 'Data/Output/Models/PROMPTED_CLIP_CROSS_epoch_44.pth',
60     segmentation_model = ClipDecoder().to(device)
61     segmentation_model.load_state_dict(torch.load(model_path, map_location=device))
62     segmentation_model.eval()
63     return clip_model, preprocess_val, segmentation_model, device
64
65     clip_model, preprocess_val, model, device = load_models()
66
67     # Extract CLIP features
68     def extract_clip_features(image):
69         image = preprocess_val(image).unsqueeze(0).to(device)
70         with torch.no_grad():
71             features = clip_model.encode_image(image)
72             features = features / features.norm(p=2, dim=-1, keepdim=True)
73         return features.to(device)
74
75     def generate_gaussian_heatmap(size, center, intensity=1.0, sigma=50):
76         x0, y0 = center
77         x = np.arange(size[0])
78         y = np.arange(size[1])
79         X, Y = np.meshgrid(x, y)
80         heatmap = intensity * np.exp(-((X - x0)**2 + (Y - y0)**2) / (2 * sigma**2))
81         return heatmap
82
83     def segment_image(image, heatmap):
84         clip_features = extract_clip_features(image)
85         heatmap_resized = Image.fromarray((heatmap * 255).astype(np.uint8)).resize((256, 256))
86         heatmap = np.array(heatmap_resized) / 255.0 # Convert back to float array
87
88         # Resize and normalize input image
89         transform = transforms.Compose([
90             transforms.Resize((256, 256)),
91             transforms.ToTensor(),
92         ])
93
94         input_tensor = transform(image).to(device).to(torch.float32)
95         heatmap = torch.tensor(heatmap, dtype=torch.float32).unsqueeze(0).to(device)
96         input_tensor = torch.cat([input_tensor, heatmap], dim=0).unsqueeze(0)
97
98         with torch.no_grad():
99             output = model(input_tensor, clip_features)
100            output = torch.argmax(output.squeeze(), dim=0).cpu().numpy()
101
102    return output
103
104    def visualize_segmentation(segmented_output, original_size):
105        # Define a color palette for 4 classes (0, 1, 2, 3)
106        color_map = {
107            0: [0, 0, 0],
108            1: [51, 102, 153],
109            2: [173, 216, 230],
110            3: [255, 255, 255]
111        }
112
113        # Create a color image based on the segmented output
114        colored_output = np.zeros((segmented_output.shape[0], segmented_output.shape[1], 3), dtype=np.uint8)
115        for label, color in color_map.items():
116            colored_output[segmented_output == label] = color
117
118        # Convert to a PIL image
119        segmented_image = Image.fromarray(colored_output)
120        segmented_image = segmented_image.resize(original_size, resample=Image.NEAREST)
121
122        # Save the image
123        segmented_image_path = os.path.join(app.config['UPLOAD_FOLDER'], 'segmented.png')
124        segmented_image.save(segmented_image_path)
125
126    return segmented_image_path

```

```

126
127     @app.route('/')
128     def home():
129         return render_template('index.html')
130
131     @app.route('/upload', methods=['POST'])
132     def upload():
133         try:
134             image = request.files['image']
135             if image:
136                 image_path = os.path.join(app.config['UPLOAD_FOLDER'], image.filename)
137                 image.save(image_path)
138                 return render_template('image_display.html', image_path=image_path)
139             else:
140                 return "No image received"
141         except Exception as e:
142             return f"Error: {str(e)}"
143
144     @app.route('/process_click', methods=['POST'])
145     def process_click():
146         try:
147             data = request.json
148             x, y = float(data['x']), float(data['y'])
149             image_path = data['image_path']
150             print(f"Clicked here {x}, {y}")
151             image = Image.open(image_path).convert("RGB")
152             image_size = image.size
153             print(image_size)
154             heatmap = generate_gaussian_heatmap(image_size, (x, y), intensity=1.0, sigma=50)
155             heatmap_path = os.path.join(app.config['UPLOAD_FOLDER'], 'heatmap.png')
156
157             plt.imsave(heatmap_path, heatmap, cmap='gray')
158
159             # print("Came here")
160             segmented_output = segment_image(image, heatmap)
161             segmented_image_path = visualize_segmentation(segmented_output, image_size)
162             print('Reached here')
163             return jsonify({'heatmap': heatmap_path, 'segmented_image': segmented_image_path})
164         except Exception as e:
165             return jsonify({'error': str(e)})
166
167     @app.route('/process_scribble', methods=['POST'])
168     def process_scribble():
169         try:
170             data = request.json
171             points = [(float(p['x']), float(p['y'])) for p in data['points']]
172             image_path = data['image_path']
173
174             print(f"Scribble points: {points}")
175
176             image = Image.open(image_path).convert("RGB")
177             image_size = image.size
178
179             # Generate heatmap from multiple points
180             heatmap = np.zeros(image_size[:-1]) # Create empty heatmap
181             for x, y in points:
182                 heatmap += generate_gaussian_heatmap(image_size, (x, y), intensity=1.0, sigma=1)
183
184             heatmap = np.clip(heatmap, 0, 1) # Normalize
185
186             heatmap_path = os.path.join(app.config['UPLOAD_FOLDER'], 'heatmap.png')
187             plt.imsave(heatmap_path, heatmap, cmap='gray')
188
189             segmented_output = segment_image(image, heatmap)
190             segmented_image_path = visualize_segmentation(segmented_output, image_size)
191
192             return jsonify({'heatmap': heatmap_path, 'segmented_image': segmented_image_path})
193
194         except Exception as e:
195             return jsonify({'error': str(e)})
196
197     @app.route('/process_bounding_box', methods=['POST'])
198     def process_bounding_box():
199         try:
200             data = request.json
201             start_x, start_y = float(data['startX']), float(data['startY'])
202             end_x, end_y = float(data['endX']), float(data['endY'])
203             image_path = data['image_path']
204
205             print(f"Bounding Box: ({start_x}, {start_y}) to ({end_x}, {end_y})")
206
207             image = Image.open(image_path).convert("RGB")
208             image_size = image.size
209
210             # Ensure proper ordering of coordinates
211             x_min, x_max = min(start_x, end_x), max(start_x, end_x)
212             y_min, y_max = min(start_y, end_y), max(start_y, end_y)
213
214             # Generate all points inside the bounding box
215             points = [(x, y) for x in range(int(x_min), int(x_max)) for y in range(int(y_min), int(y_max))]
216
217             heatmap = np.zeros(image_size[:-1]) # Empty heatmap
218             for x, y in points:
219                 heatmap += generate_gaussian_heatmap(image_size, (x, y), intensity=1.0, sigma=1)
220

```

```

221     heatmap = np.clip(heatmap, 0, 1) # Normalize
222
223     heatmap_path = os.path.join(app.config['UPLOAD_FOLDER'], 'heatmap.png')
224     plt.imsave(heatmap_path, heatmap, cmap='gray')
225
226     segmented_output = segment_image(image, heatmap)
227     segmented_image_path = visualize_segmentation(segmented_output, image_size)
228
229     return jsonify({'heatmap': heatmap_path, 'segmented_image': segmented_image_path})
230
231 except Exception as e:
232     return jsonify({'error': str(e)}), 500
233
234
235 @app.route('/static/<path:filename>')
236 def static_files(filename):
237     return send_from_directory(app.config['UPLOAD_FOLDER'], filename)
238
239 if __name__ == '__main__':
240     app.run(debug=True)

```

### A.3.12 index.html: Frontend Landing Page

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Image Segmentation</title>
7      <link href="https://fonts.googleapis.com/css2?family=Merrriweather:wght@300;400;700&display=swap" rel="stylesheet">
8      <style>
9          body {
10              font-family: 'Merrriweather', serif;
11              background-color: #eef2f7;
12              margin: 0;
13              padding: 20px;
14              display: flex;
15              justify-content: center;
16              align-items: center;
17              flex-direction: column;
18              min-height: 100vh;
19          }
20          h1, h2 {
21              color: #2c3e50;
22              margin-bottom: 15px;
23              text-align: center;
24          }
25          form {
26              background-color: #ffffff;
27              padding: 25px;
28              border-radius: 12px;
29              box-shadow: 0 6px 12px rgba(0, 0, 0, 0.1);
30              margin-bottom: 25px;
31              width: 100%;
32              max-width: 400px;
33              text-align: center;
34          }
35          label {
36              font-weight: bold;
37              color: #34495e;
38          }
39          input[type="file"] {
40              margin: 10px 0;
41              padding: 8px;
42              border: 1px solid #ccc;
43              border-radius: 6px;
44              width: 100%;
45          }
46          button {
47              background-color: #3498db;
48              color: white;
49              padding: 12px 18px;
50              border: none;
51              border-radius: 8px;
52              cursor: pointer;
53              font-size: 16px;
54              transition: background-color 0.3s ease-in-out;
55          }
56          button:hover {
57              background-color: #2980b9;
58          }
59          .image-container {
60              text-align: center;
61              margin: 20px 0;
62          }
63          img {
64              max-width: 100%;
65              border-radius: 12px;
66              box-shadow: 0 6px 12px rgba(0, 0, 0, 0.1);
67              cursor: crosshair;

```

```

68         }
69     .segmented-title {
70         margin-top: 30px;
71         font-size: 20px;
72         font-weight: bold;
73     }
74 
```

`</style>`

```

75 <script>
76     function sendClick(event) {
77         let img = document.getElementById("uploadedImage");
78         let rect = img.getBoundingClientRect();
79         let x = event.clientX - rect.left;
80         let y = event.clientY - rect.top;
81         let imagePath = img.getAttribute("src");

82         fetch('/process_click', {
83             method: 'POST',
84             headers: { 'Content-Type': 'application/json' },
85             body: JSON.stringify({ x: x, y: y, image_path: imagePath })
86         })
87         .then(response => response.json())
88         .then(data => {
89             if (data.segmented_image) {
90                 document.getElementById("segmentedResult").src = data.segmented_image;
91             } else {
92                 alert("Error: " + data.error);
93             }
94         });
95     }
96 
```

`</script>`

```

97 </head>
98 <body>
99     <h1>Image Segmentation Tool</h1>
100    <form action="/upload" method="POST" enctype="multipart/form-data">
101        <label for="image">Select an image:</label>
102        <input type="file" name="image" required>
103        <button type="submit">Upload</button>
104    </form>

105    <h2>Click on the image to select coordinates</h2>
106    <div class="image-container">
107        
108    </div>

109    <h2 class="segmented-title">Segmented Image</h2>
110    <div class="image-container">
111        <img id="segmentedResult" src="">
112    </div>
113 </body>
114 </html>

```

### A.3.13 image\_display.html: Frontend Interaction and Result Page

```

1  <!DOCTYPE html>
2  <html lang="en">
3      <head>
4          <meta charset="UTF-8">
5          <title>Image Interaction</title>
6          <style>
7              canvas {
8                  position: absolute;
9                  top: 0;
10                 left: 0;
11                 pointer-events: none;
12             }
13         </style>
14         <script>
15             let mode = "point";
16             let drawing = false;
17             let strokePoints = [];
18             let startX, startY, endX, endY;
19             let canvas, ctx;
20
21             function changeMode() {
22                 mode = document.getElementById("modeSelect").value;
23             }
24
25             function sendClick(event) {
26                 if (mode !== "point") return;
27                 let img = document.getElementById("uploadedImage");
28                 let rect = img.getBoundingClientRect();
29                 let x = (event.clientX - rect.left).toFixed(2);
30                 let y = (event.clientY - rect.top).toFixed(2);
31                 let imagePath = img.getAttribute("src");
32
33                 fetch('/process_click', {
34                     method: 'POST',
35                     headers: { 'Content-Type': 'application/json' },
36                     body: JSON.stringify({ x: x, y: y, image_path: imagePath })
37                 })
38                 .then(response => response.json())

```

```

39         .then(updateResults);
40     }
41
42     window.onload = function() {
43         let img = document.getElementById("uploadedImage");
44         canvas = document.createElement("canvas");
45         canvas.width = img.width;
46         canvas.height = img.height;
47         canvas.style.position = "absolute";
48         canvas.style.top = img.offsetTop + "px";
49         canvas.style.left = img.offsetLeft + "px";
50         document.body.appendChild(canvas);
51         ctx = canvas.getContext("2d");
52     };
53
54     function startDrawing(event) {
55         event.preventDefault(); // Prevent image movement
56         let img = document.getElementById("uploadedImage");
57         let rect = img.getBoundingClientRect();
58         startX = event.clientX - rect.left;
59         startY = event.clientY - rect.top;
60         drawing = true;
61         strokePoints = [];
62         ctx.clearRect(0, 0, canvas.width, canvas.height);
63     }
64
65     function draw(event) {
66         event.preventDefault(); // Prevent dragging behavior
67         if (!drawing) return;
68         let img = document.getElementById("uploadedImage");
69         let rect = img.getBoundingClientRect();
70         let x = event.clientX - rect.left;
71         let y = event.clientY - rect.top;
72
73         if (mode === "scribble") {
74             addPoint(event, true);
75         } else if (mode === "bounding_box") {
76             ctx.clearRect(0, 0, canvas.width, canvas.height);
77             ctx.strokeStyle = "blue";
78             ctx.lineWidth = 2;
79             ctx.strokeRect(startX, startY, x - startX, y - startY);
80             endX = x;
81             endY = y;
82         }
83     }
84
85
86     function stopDrawing(event) {
87         if (!drawing) return;
88         drawing = false;
89         if (mode === "scribble" && strokePoints.length > 0) {
90             sendStroke();
91         } else if (mode === "bounding_box") {
92             sendBoundingBox();
93         }
94     }
95
96     function addPoint(event, drawLine) {
97         let img = document.getElementById("uploadedImage");
98         let rect = img.getBoundingClientRect();
99         let x = event.clientX - rect.left;
100        let y = event.clientY - rect.top;
101        strokePoints.push({ x: x.toFixed(2), y: y.toFixed(2) });
102
103        if (drawLine && strokePoints.length > 1) {
104            let lastPoint = strokePoints[strokePoints.length - 2];
105            ctx.beginPath();
106            ctx.moveTo(lastPoint.x, lastPoint.y);
107            ctx.lineTo(x, y);
108            ctx.strokeStyle = "red";
109            ctx.lineWidth = 2;
110            ctx.stroke();
111        }
112    }
113
114    function sendStroke() {
115        let img = document.getElementById("uploadedImage");
116        let imagePath = img.getAttribute("src");
117
118        fetch('/process_scribble', {
119            method: 'POST',
120            headers: { 'Content-Type': 'application/json' },
121            body: JSON.stringify({ points: strokePoints, image_path: imagePath })
122        })
123        .then(response => response.json())
124        .then(updateResults);
125    }
126
127    function sendBoundingBox() {
128        let img = document.getElementById("uploadedImage");
129        let imagePath = img.getAttribute("src");
130
131        fetch('/process_bounding_box', {
132            method: 'POST',
133            headers: { 'Content-Type': 'application/json' },

```

```

134         body: JSON.stringify({ startX: startX.toFixed(2), startY: startY.toFixed(2), endX: endX.toFixed(2),
135             endY: endY.toFixed(2), imagePath: imagePath })
136     })
137     .then(response => response.json())
138     .then(updateResults);
139 }
140
141     function updateResults(data) {
142         if (data.segmentedImage) {
143             document.getElementById("segmentedResult").src = data.segmentedImage;
144         }
145         if (data.heatmap) {
146             document.getElementById("heatmapResult").src = data.heatmap;
147         }
148     }
149 </script>
150 </head>
151 <body>
152     <h1>Image Interaction</h1>
153     <label for="modeSelect">Select Mode:</label>
154     <select id="modeSelect" onchange="changeMode()">
155         <option value="point">Point</option>
156         <option value="scribble">Scribble</option>
157         <option value="bounding_box">Bounding Box</option>
158     </select>
159     <br><br>
160     
167
168     <h2>Heatmap</h2>
169     <img id="heatmapResult" src="" style="max-width:100%;">
170
171     <h2>Segmented Image</h2>
172     <img id="segmentedResult" src="" style="max-width:100%;">
173 </body>
174 </html>

```

#### A.4 Contribution statement

For the production of the present study, the workload was equally distributed between two students, with clearly devided responsibilities and continuous development checks in the form of weekly meetings. Hereafter, the specific distribution of tasks.

S2677266: dataset loading, augmentation, AutoEncoder, CLIP, data annotation, performance evaluation, UI backend, report drafting

S2748897: dataset loading, resizing, UNet, data annotation, prompt-enriched CLIP, robustness exploration, UI frontend, report drafting

## References

- [1] Alberto Garcia-Garcia, Sergio Orts-Escalano, Sergiu Oprea, Victor Villena-Martinez, and Jose Garcia-Rodriguez. A review on deep learning techniques applied to semantic segmentation. *arXiv preprint arXiv:1704.06857*, 2017.
- [2] Akanksha Bali and Shailendra Narayan Singh. A review on the strategies and techniques of image segmentation. In *2015 Fifth international conference on advanced computing & communication technologies*, pages 113–120. IEEE, 2015.
- [3] Hui Zhang, Jason E Fritts, and Sally A Goldman. Image segmentation evaluation: A survey of unsupervised methods. *computer vision and image understanding*, 110(2):260–280, 2008.
- [4] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3523–3542, 2022. Available at: DOI: 10.1109/TPAMI.2021.3059968.
- [5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.
- [6] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2018.
- [7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5–9, 2015, proceedings, part III* 18, pages 234–241. Springer, 2015.
- [8] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.
- [9] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- [10] Diederik P Kingma, Max Welling, et al. Auto-encoding variational bayes, 2013.
- [11] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.
- [12] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Clip: Contrastive language-image pretraining. Website, 2021. <https://openai.com/index/clip/>.
- [13] Ziqin Zhou, Yinjie Lei, Bowen Zhang, Lingqiao Liu, and Yifan Liu. Zegclip: Towards adapting clip for zero-shot semantic segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11175–11185, 2023.
- [14] Zdravko Marinov, Paul F Jäger, Jan Egger, Jens Kleesiek, and Rainer Stiefelhagen. Deep interactive segmentation of medical images: A systematic review and taxonomy. *IEEE transactions on pattern analysis and machine intelligence*, 2024.

- [15] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Sam: Segment anything model. Website, 2023. <https://segment-anything.com/>.
- [16] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar. Oxford-iiit pet dataset. Website, 2012. <https://www.robots.ox.ac.uk/~vgg/data/pets/>.
- [17] Roboflow. You might be resizing your images incorrectly. *Roboflow Blog*, 2023. Accessed: 2025-04-04.
- [18] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.
- [19] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3498–3505, 2012.
- [20] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 fourth international conference on 3D vision (3DV)*, pages 565–571. Ieee, 2016.
- [21] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Matthias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, et al. Attention u-net: Learning where to look for the pancreas. *arXiv preprint arXiv:1804.03999*, 2018.
- [22] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111:98–136, 2015.
- [23] Lee R Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.
- [24] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.