Low-resolution Scene Context for Rapid Object Detection

Chen-An Fan

Supervisor: Dr. Kris Ehinger

University of Melbourne

Author Note

This research was undertaken using the LIEF HPC-GPGPU Facility hosted at the University of Melbourne. This Facility was established with the assistance of LIEF Grant LE170100200.

Declaration

I certify that

- *this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.*
- *the thesis is 8912 words in length (excluding text in images, table, bibliographies and appendices).*

Signature: Chen-An Fan, Nov. 2021

Acknowledgements

Firstly, I would like to express my deepest thanks to my supervisor, Dr. Kris Ehinger, for all the guidance, instructions, and mentoring throughout the whole project. She provides me with professional insights, gives me the freedom to choose and explore different approaches, and guides me to the correct course with patient and enthusiasm. Furthermore, I can feel my improvement during this semester and passion for computer vision through weekly discussions with her.

Secondly, I would also like to thank my research partner, Hee Won Kim, for all the discussions, resources sharing, and encouragement during this project. She provides valuable opinions during idea generations and implementation, and she gives me full support when we encountered setbacks.

Lastly, I would like to thank the University of Melbourne for providing my computational resources. This research was undertaken using the LIEF HPC-GPGPU Facility hosted at the University of Melbourne. This Facility was established with the assistance of LIEF Grant LE170100200.

Chen-An Fan

The University of Melbourne

November 2021

Abstract

Most of the SOTA CNNs based object detectors are object-centred. They focus on using the local features of the target objects directly for detection without understanding the scene. However, that is not the way how humans fast find the target in the environment. By referencing the past experience of the context, humans find the important regions to focus on before detailed searching for the local features of targets. This referencing in the human brain is called contextual priming. In this project, I will simulate the contextual priming mechanism in the human brain to machines by vision transformer architecture.

The attention mechanism in the vision transformer enables the model to capture the global information in the very early layers; hence, it can better understand the scene than the CNNs. Within the vision transformer, an input image is first divided into patches. Then each of the patches is projected into a low-dimension vector, which captures the low-resolution (coarse) features such as texture, edges, or blobs in that patch. Next, the features vector of each patch is fed into multiple transformer blocks to attend to the features in other patches. This architecture enables the model to first rapidly extract coarse features locally, then attend and refine them globally. It is similar to how humans understand the scene (e.g., finding surfaces and edges locally → guessing the scene based on the arrangement of the surfaces and edges).

In this project, I train vision transformer models to predict the likely locations for two classes of objects, which are “bottle” and “chair”. During the training, I mask the target objects to avoid the model learning the local features of the targets. As a result, my models reach 76.46% and 63.25% mean average precision on predicting the likely locations of bottle and chair. Furthermore, the detection speed is 18.06 FPS on Tesla P100-PCIE-12GB GPU. This indicates that a machine can locate objects rapidly using only the low-resolution scene context, just like the contextual priming mechanism in the human brain.

As my model only uses the scene context as features, my model can detect the likely locations of target objects even if the target objects are not in the scene. Hence, my model may be used on applications that need rapidly finding important regions to focus on within an environment. For example, in augmented reality applications, my model can be used to guide the users to put a virtual object at a reasonable place; in autonomous vehicle fields, my model can be used to rapidly understand the scene; moreover, my model could replace the region proposal network in two-step object detectors for generating fewer but accurate proposals.

Keywords: Contextual Priming, Scene Context, Vision Transformer, Rapid Object Detection

Table of contents

List of figures

List of tables

Chapter 1. Introduction

1.1. Motivation: The Priming Mechanism in Brain

What is the most efficient way to find a specific object in a scene? As a human, what will you do to search for a target in an environment rapidly? Research [1] proved that human uses prior experience to speed up the process of new information in the brain. For example, give an unseen indoor image to people and ask them to find bottles in it. People tend to start searching from horizontal surfaces like a desktop or a shelf unconsciously. They are not likely to search vertical surfaces such as walls because the prior experience tells people where the bottles are supposed to be. This demonstrates the use of scene context to narrow down the searching area for fast locating the target object. In neuroscience, the mechanism of referencing the prior experience about scene context is called contextual priming.

Priming is a mechanism that affects the processing of stimuli in the brain by referencing the prior experience of the same or related stimuli [1]. It can reduce the reaction time, introduce bias when generating responses, and lead to higher accuracy in identifying a degraded version of stimulus [1]. For example, it is hard for people to recognise the object in Figure 1 because the image is too blurry (degraded) and lacks enough object features. However, if other objects co-occurred in the same scene are also shown (see Figure 2 (a)), people can start having higher confidence to recognise the object. Furthermore, if the whole scene context is provided (see Figure 2 (b)), it will be obvious that the object in Figure 1 is pedestrians. This demonstrates the use of scene context to improve the recognition of a specific object.

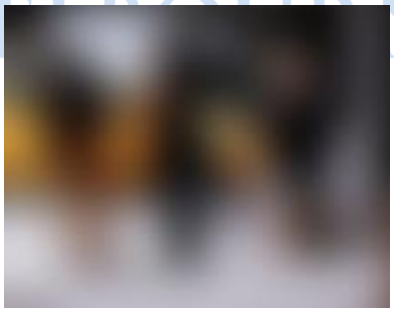
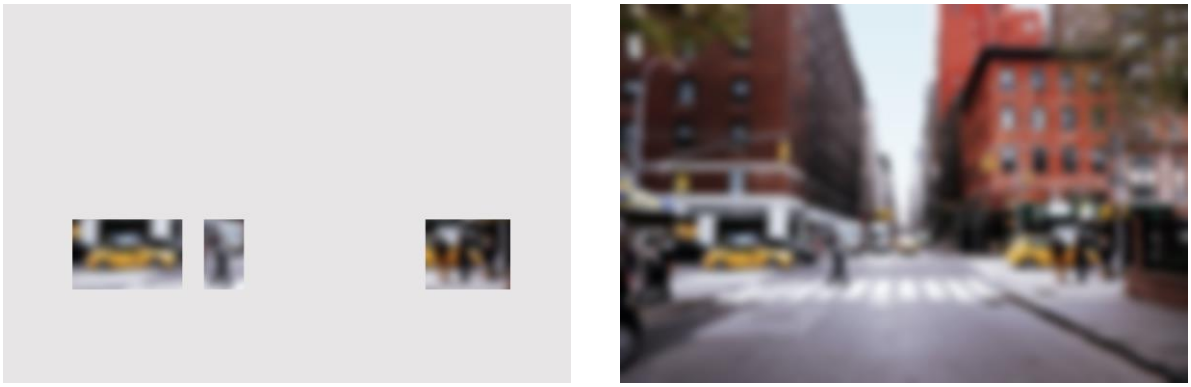


Figure 1. A degraded version of image

Chun and Jiang [2] pointed out that, in the real world, there exist relationships between the objects and the spatial configuration. Hence, when people want to recognise or locate a specific object by referencing the co-occurred objects or the whole scene context, the prior experience will introduce a strong bias about what the target object is or where the object target should be. This proves that referencing the visual context can guide spatial attention, therefore, narrow down the area for

searching the target object [2]. However, can a machine also have contextual priming to understand the scene?



(a) (b)

Figure 2. Contextual information – (a) co-occurred objects in the same scene; (b) the whole scene context

1.2. Goal

This project aims to simulate the biological contextual priming mechanism in the human brain to a machine. I aim to answer the research question, "Can a machine learn to detect all the possible locations of specific classes of objects in a scene by only using the contextual information?" My goal is to enable a machine to learn the spatial configuration of the environment (I called it low-resolution scene context) and purpose the likely regions of the target objects in the scene.

1.3. Contributions

Most of the SOTA object detectors [3-8] are object-centred. They mainly use the local visual features of objects (e.g., the shape, size, texture of the target object) with no or minor global spatial/contextual features to detect the target objects. These approaches may be problematic when the local visual features are not enough (e.g., the degraded image) for recognising the object. In this case, the object-centred object detectors may fail to find the target object. Moreover, all the object-centred object detectors cannot predict the possible locations of an object if the object is not in the scene.

In this project, I introduce a customised vision transformer model that can capture the global context and predict the possible location of the target object even the object is not in the scene. I take a patch approach that divides an input image into seven by seven patches and then feed those patches as a flatten sequence to the transformer like the natural language processing tasks. The model will encode every patch into a vector, and the self-attention mechanism in the transform will allow the model to have a global understanding of the scene. Hence, the model can capture the spatial arrangement of the scene and even the co-occurred objects in the scene. The model will predict the probability of the existence of target objects in each patch. During the training, I take a dropping approach that drops

all the patches with the target insides. This dropping approach can ensure the model never see the target object, therefore, never learn the local features of the object during training.

I trained the model for two object classes, which are “bottle” and “chair” classes. The data is from the COCO 2017 dataset [9]. Although the dataset is not perfect (see section 3.4), my model performs reasonable prediction as human expectation.

This project purposes the first model for possible locations prediction using only the scene context. It proves that machines can learn to use the scene context solely to locate an object, just like the contextual priming in the human brain.

For application, my model can be used as a potential replacement for the region proposal network in the two-stage object detectors (e.g. [4]) by generating much fewer and more accurate proposals. Moreover, my model may also be used in autonomous vehicles to highlight the possible locations of pedestrians or vehicles in the scene. Furthermore, this model may use in augmented reality (AR). When people want to place a virtual object in the real physical world in AR, this model can tell where the reasonable place is to put on. Lastly, my model can be further extended to improve the recognition of objects by referencing the scene context. The most important contribution is that this model enables the machine to reason the spatial information, the context.

1.4. Scope and Limitations

As we discussed in section 1.1, contextual priming has two uses. First, it can be used to narrow down the searching region for specific objects, in other words, to locate the objects. Second, contextual priming can be used to improve the recognition of a degraded version of an image. In this project, my model only covers the first uses by referencing the contextual information to predict the likely locations of objects. My model does not cover how to use contextual information to improve object recognition on degraded images. Nevertheless, it will not be hard to cover that further. The simplest way to do that is to multiply the confidence scores of object classes generated by the traditional object detector and the location probabilities generated by my model together. The product of the confidence score and the location probability can be used to judge if the recognition prediction is reasonable given the scene context, therefore, find the best class for the target object. Thus, this project set a foundation for machines to learn the scene context to stimulate the contextual priming; how to use the scene context is fully flexible depending on the downstream tasks.

1.5. Thesis Structure

The thesis is organised as follows. Chapter 2 will review the current SOTA object detectors and the vision transformer model proposed in recent years. Chapter 3 will discuss the implementation, including the data pre-processing, model architecture, the dropping trick in both training and test, the design of the loss function, the training, and the choice of evaluation metrics. Chapter 4 will show the training curves and prediction results of the bottle model and chair model. Furthermore, in chapter 5, I will evaluate my model's performance and discuss the reason behind the prediction results. Finally, chapter 6 will cover the limitations of my approach, the possible future works, and the conclusion.

Chapter 2. Literature review

Context priming is the mechanism that uses the prior experience of environment configuration to facilitate object detection. The previous study [10] indicated that humans might perceive the environment before looking for the features (i.e. the intrinsic information) of objects to find the target object. For example, when a human wants to find a pedestrian in an image of a street, he/she will usually first focus on the sidewalk area because he/she has prior experience knowing where a pedestrian is most likely to exist. This can avoid detailed scanning of the whole image to find target objects directly by their intrinsic features. Therefore, narrow down the searching area and facilitate further rapid object detection.

Moreover, when the intrinsic information of an object is not enough for detection, context priming play a crucial role [10]. For example, knowing the relative location and scale of the target object in the image can dominant the identification of the object's category (see Figure 3, the pedestrian on the crosswalk is actually a black triangle drawn by me, however, based on its scale and relative location in the scene, people may think it is a pedestrian).

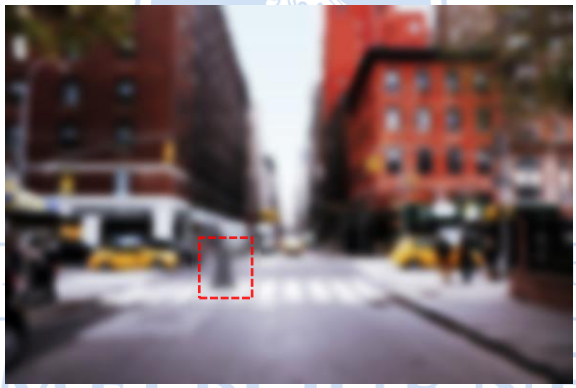


Figure 3. Artefact triangle in a street view image

A previous study [11] showed that low-level features statistics such as orientation and intensity contrast could represent the image's basic components such as edges, ridges, blobs and junctions. The low-level features, therefore, play a significant role to represent different scenes.

The previous work [10] used multiple Gabor filters with different orientations and scales to extract the context features of the entire image. Then, they [10] use principal component analysis (PCA) to reduce the dimension of context feature to extract the low-level statistics of the entire scene. Finally, they [10] train three separate Gaussian mixture models (GMMs) to predict the potential object categories (i.e. classes) in the scene, the likely region that contains those objects, and the possible scale (size) of the object given the scene (see Figure 4).

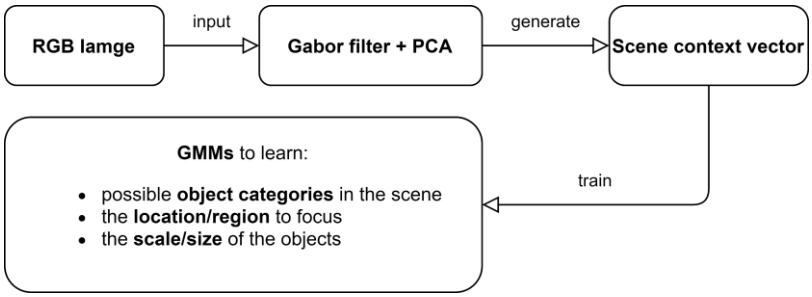


Figure 4. Contextual priming model proposed by Torralba [8]

However, Torralba's model [10] has some limitations. It is an early work that was implemented by using basic filters for features extraction and simple probability models (GMMs) for training. It did not exploit the power of the neural network; the GMMs is a probability model for clustering, so it cannot capture the interaction of features among different spatial locations in the image (e.g., GMMs can only know some regions are flat surface, but it cannot know the relationship among those flat surfaces). Therefore, it is not likely to capture co-occurred objects in the scene.

Recent state-of-the-art object detection models, no matter two-stage [3-5] or single-stage [6-8], all exploit the power of the deep neural network. Nevertheless, those two-stage region-based detectors [3-5] are all object-centred. They only use parts of the feature map that correspond to the region proposals to train the classifier. Therefore, they can only see the partial image and learn the local object's features. This makes them unable to capture global scene context well and performance poor when the image resolution is low. Instead, the single-stage [6-8] detectors use the whole feature map for learning. This makes YOLO [6] able to reason the global scene and all the elements in the entire image, leading to fewer false positives in background regions than region-based approaches [6]. However, YOLO [6] only use the output of the top-most convolutional layer as a feature map. The top layer of the deep neural network tends to be spatially coarse and unfavourable for precise localisation [12], hence, lead to lousy accuracy in detecting small objects in the scene.

To learn the global context better, YOLOv3 [8] and SSD [7] adopted a concept similar to the feature pyramid network [13] that uses multiple full feature maps at different scales for learning. Another approach, ION [14], added two additional 4-direction IRNNs [15] on top of the last convolutional layer to learn the context outside the region of interest (RoI). They all proved that incorporating scene context information can improve object detection accuracy and bounding box regression. However, they are still mainly object-centred and cannot capture global long-distance features.

Shrivastava and Gupta [16] augmented the Faster R-CNN [4] by adding a segmentation module. They [16] use the pixel-wise probability output of the semantic segmentation module as an additional feature map. Furthermore, they [16] treat this segmentation map as the contextual feature map to

represent the relationship between objects in the scene. I argue that this method is still object-centred; it still uses the local features of objects to segment objects first, then it could use the segmentation map to represent the spatial arrangement of the objects in the scene.

In this project, I would like to capture the scene context directly without detecting any objects in the scene in advance. Therefore, I decided to use the vision transformer (ViT) model architecture because it has been proved that the self-attention mechanism in ViT can integrate information globally (across the entire image), even in the lowest layer [17].

Vision transformer was first proposed by Dosovitskiy et al. [17] in 2020. They [17] adopted a self-attention-based architecture, the transformer [18], which is widely used in natural language processing (NLP) to replace the convolutional neural network (CNN) on an image classification task. Moreover, they [17] also proved that ViT requires much fewer computational resources to train than the CNNs due to the computational efficiency in transformers [18].

In ViT, an input image is first split and flattened into a sequence of patches. The image patches here are similar to the tokens (words) in NLP tasks. Then, the sequence of patches is fed into a trainable linear projection layer to project them into a sequence of vectors (these vectors are called "patch embedding vectors" in [17]). After that, a learnable embedding vector (called CLS embedding in [17]) is appended at the beginning of the sequence of patch embedding vectors. This learnable embedding vector is served as the holistic representation of the input image like the CLS token in BERT [19]. Next, learnable position embeddings are added to the patches embedding vectors to keep tracking the position of every patch in the original image. These position embeddings are important for ViT to keep the spatial relations between patches because ViT flattens the input image into a 1D sequence of patches at the first step. After adding the position embeddings, the result sequence of embedding vectors are fed into the transformer encoder [18] to generate the encoded output vectors.

In [17], only the encoded vector of the CLS embedding is taken and fed into an MLP to generate the class probabilities for the image classification task. As I mentioned before, the CLS embedding is served as the holistic representation of the whole input image due to the self-attention mechanism.

Experiments showed that ViT trained on mid-sized datasets perform worse than CNN because transformer lakes some of the inductive biases than CNN, such as the translation equivariance of images, due to the flattening step [17]. However, pre-trained ViT on a large dataset can solve this problem. ViT pre-trained on a large dataset can match or exceed the performance of the SOTA CNN-based image classifiers with a much cheaper computation cost (2-4 times cheaper than ResNets [20]) [17].

Researchers [21] made a comparison between ViT and ResNets [20]. They [21] point out that ViT can aggregate information from other spatial locations even in the earliest layers due to the self-attention mechanism. In the lower layers of ViT, some attention heads attend locally while others attend globally. While moving to higher layers of ViT, all attention heads attend globally [21]. This situation is very different from CNNs, which cannot capture global information in early layers due to the small receptive fields at lower layers. They [21] also observed that the higher layers of ViT preserved spatial location information more faithfully than CNNs because multiple pooling layers in CNNs may reduce the localisation.

To sum up, due to better spatial information preservation at higher layers and the earlier aggregation of global information, I decided to use ViT as my main model architecture to capture the scene context better. The attention nature of ViT makes it able to learn the relationship between patches at different spatial locations, making it capable of capturing the spatial configuration of the scene and even the co-occurred objects in the scene.

Chapter 3. Research methodology

3.1. Patch Approach

To represent the likely regions of objects, I decide to take a patch approach. The patch approach has some benefits. First, it is compatible with the vision transformer, which also crops input images into patches. Second, using the patches format to represent the probability map is intuitive. Here, every input image will be resized to square ($224 \times 224\ pixels$) and then divided into $7 \times 7\ patches$ (see Figure 5). Then, the prediction results of my model will be the probabilities of a specific object present in each patch (i.e., $P(object|patch)$ such as $P(bottle|patch[0,0])$). Hence, the final prediction will be the probability maps for every object class (see Figure 6).

Figure 5. 7x7 patches on image

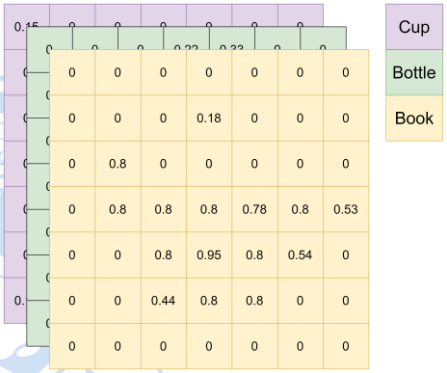


Figure 6. Probability maps

3.2. Model Architecture

As mentioned in chapter 2, I decided to use the vision transformer (ViT) [17] as my primary model due to its capability of capturing global context and maintaining spatial information faithfully. The ViT implemented by Dosovitskiy et al. [17] was for image classification tasks. They [17] appended an additional CLS embedding with the sequence of patch embeddings to represent the whole image and only used the encoded CLS embedding to predict the class of input image. Contrarily, my task is to predict the likely locations of objects, so every patch needs to output a probability vector to represent the probability of different classes of objects in that patch. Therefore, I modify the ViT architecture as Figure 8.

The three most important parameters needed to be set are the input image size, the patch size, and the embedding dimension. In order to match the available pre-trained weight online (from the PyTorch Image Models[22]), I choose the input image size to $224 \times 224\ pixels$ and the patch size to $32 \times 32\ pixels$, so there will be $7 \times 7\ patches$ in total. In this case, we will have 7×7 probability maps as output; it will not be too fine-grained nor too coarse. For the embedding dimension, I set it

to 768. This means that every patch ($32 \times 32\ pixels \times 3\ channels$) will be embedding into a one-dimension vector with 768 elements inside.

Before being fed into my model, an image is first resized to square ($224 \times 224\ pixel$) as the model take a fixed-size square image as input. Then, in the model, this input image will go through the following steps.

The first step is to crop the input image into a sequence of patches and linear project them into patch embedding vectors, as shown in Figure 8. This is done by one 2D convolution layer called the "patch embedding layer". In this convolution layer, I set the kernel size and stride equal to the patch size (32x32), and the number of output channels equals to the embedding dimension (768). Therefore, after this patch embedding layer, every patch will be projected to a patch embedding vector with a size equal to 1x768. Figure 7 shows the top 30 principal components of the learned filters in this patch embedding layer. Those filters are used to liner project each patch to lower dimensions. As we can observe in Figure 7, those filters are relatively simple; therefore, they capture the low-resolution (coarse) features in the patches. To see more filers, please see Appendix A: Filters of patch embedding layer.

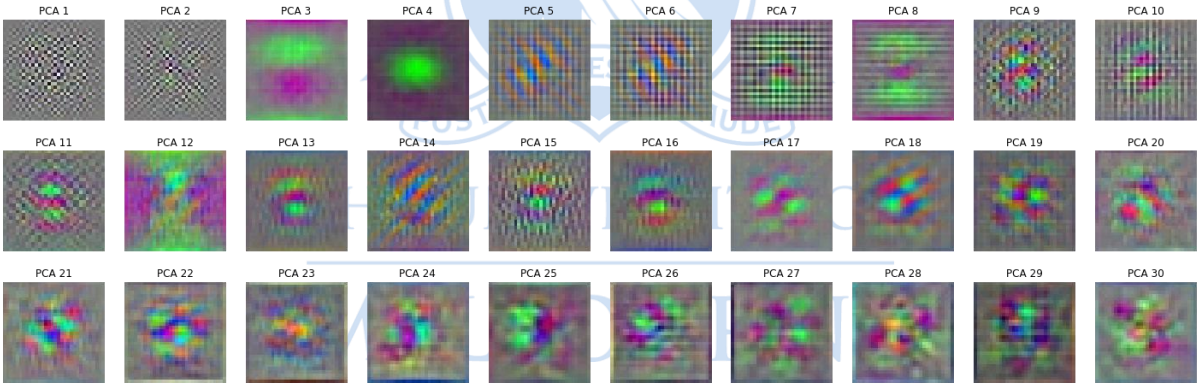


Figure 7. Top 30 principal components of the learned filters in patch embedding layer

The second step is adding a learnable position embedding vector to each patch embedding vector to let the transformers track the location of patches in the original image. Those learnable position embedding vectors are initialised as all zeros, and it will gradually learn to represent the location of patches during training [17]. Dosovitskiy et al. [17] proved that there is no need to handicraft more advanced position embedding as there is almost no difference in performance.

The third step is feeding each patch's embedding vector (position embedding + patch embedding) into the transformer encoder [18]. This transformer encoder is composed of stacking 12 transformer blocks. In each transformer block, there is one multi-head self-attention layer & one MLP layer, plus layer

norm & residual connection applied between layers as described in [17]. Thus, after the transformer encoder, we get an encoded vector for every patch.

Finally, the encoded vector of every patch is fed into a simple linear projection layer with a sigmoid function as the activation function. This layer will project the encoded vector into a classes probability vector representing the probabilities of different classes of objects in that patch.

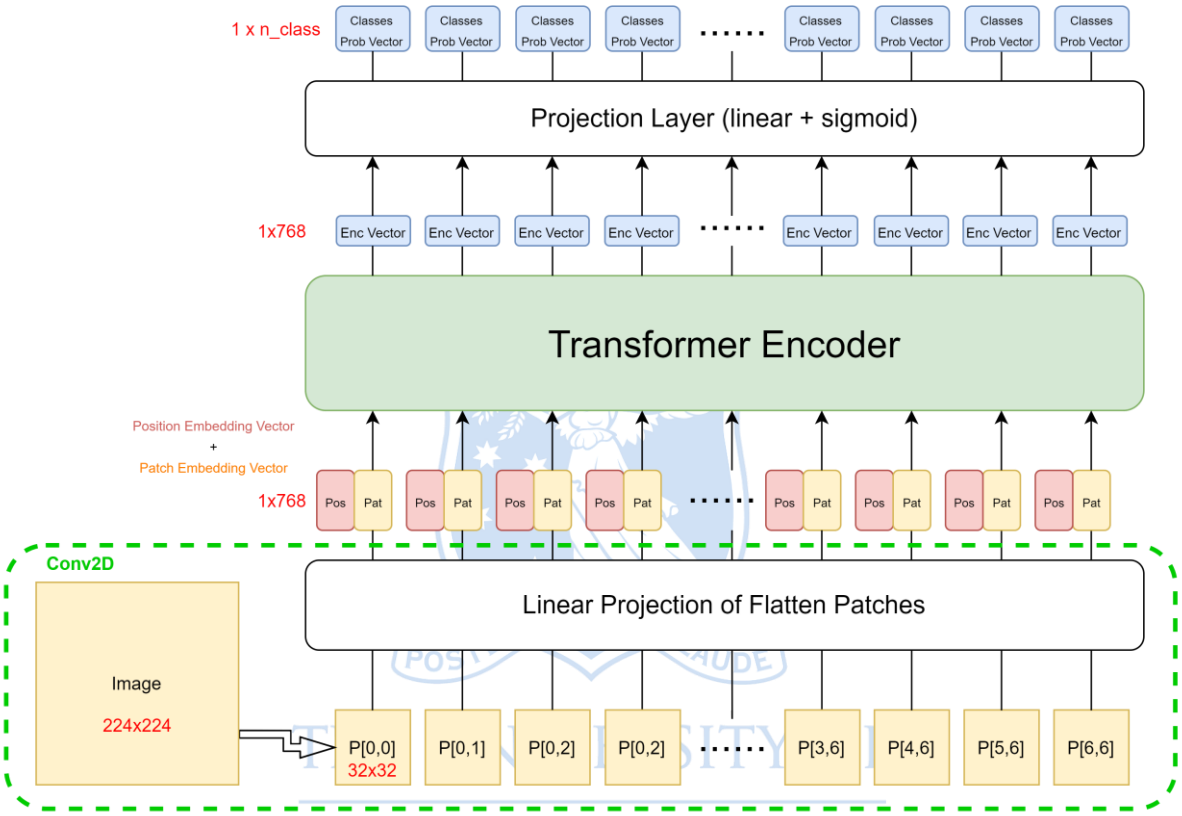


Figure 8. Model architecture

3.3. Data & Pre-processing

In this project, I use the COCO 2017 dataset [9] with bounding box (bbox) annotations. All images in the dataset will be first resized to square ($224 \times 224\ pixel$) by Lanczos interpolation. Then, the bounding boxes of images will be converted into patch representations using two thresholds (see Figure 9). The two thresholds are the *"large bbox threshold"* and *"small bbox threshold"*, which are defined as:

$$for\ large\ bbox: \frac{bbox\ area \cap patch\ area}{patch\ area} \geq 33\%$$

$$for\ small\ bbox: \frac{bbox\ area \cap patch\ area}{bbox\ area} \geq 25\%$$

The first threshold is defined as the percentage of patch area occupied by the bounding box. This threshold is suitable for large bounding boxes, which occupy much patch area (see the blue bbox in Figure 9). This threshold is set to 33%, which means that if a bounding box occupies 1/3 of the patch, that patch should be labelled as positive. I use 33% as the threshold because the patch size is large. By observing the data, I think occupied 1/3 of a large patch is enough to be defined as positive.

The second threshold is defined as the percentage of a bounding box that falls inside a specific patch. This threshold is suitable for small bounding boxes, which typically cannot occupy the majority area of a patch (see the green bbox in Figure 9). For example, if a small bounding box falls at the centre of four patches, each patch will contain 25% of the bounding boxes. Hence, I set this "*small bbox threshold"* to 0.25 to label all four patches in this case as positive.

Any patch that satisfies one of the two thresholds will be labelled as positive (the ground truth likely region of the target object).

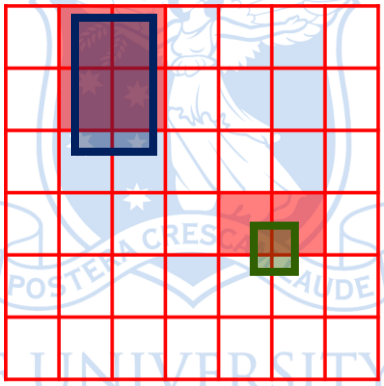


Figure 9. Annotation: bbox to patch representation

After converting the annotations into patch representation, I filter out images with more than 20 positive patches and images with no positive patches during training. The images with more than 20 positive patches (around 1.89% instances in the COCO dataset) tend to be close-shot images, so they will not have much contextual information to help the model learning. On the other hand, there will not be any positive patches in extreme cases where the target object is too tiny. Therefore, I also removed those images to prevent the model unlearns the scene context.

3.4. Ground Truth Enhancement

The COCO dataset is not perfect for my task. In the COCO dataset, only the target objects are labelled as ground truth. Nevertheless, my task is to find all the likely locations of the target objects. Take Figure 11 for example, the COCO dataset only labels the chair in the room as the ground truth; however, my task is to find all the possible locations to put a chair (can be thought as "where can

another chair be put in the room"). Ideally, the perfect ground truth of a chair in a room should be all the empty regions of the floor and the regions near a desk.

Unfortunately, there is no ideal dataset available, so I do a trick to improve the ground truth label slightly. I do not want the model to be penalised too much when it predicts a patch near the non-perfect COCO ground truth (the target object itself) as positive because the regions near the target objects are often also the likely locations to place that object. Hence, I apply a decay function to the COCO ground truth as the formula below.

$$Patch\ value\ =\ \max\left(\log_n(n-x),\ 0\right), \qquad x = distance\ to\ the\ nearest\ positive\ patch$$

The "n" in the decay function is a parameter to determine how far a neighbour patch could have a value larger than zero (see Figure 10). The value here means the probability of the target object present in the specific patch. This decay function has a blur effect similar to the Gaussian filter. However, it can keep the original positive patch's value as one while assigning decaying value to the neighbour patches based on the distance (see Figure 11 - right). Here, I set the "n" to 3. In that case, the patches that have a distance to the nearest positive patch larger or equal to 2 will have zero value. I choose "n" to be three because there are only 7x7 patches. If "n" is too large (e.g., n=4 or 5), more than 50% patches of the image would have some value (i.e., overextending the coco ground truth); this means that the target object can be placed almost everywhere in the scene, therefore, is not desirable to my task.

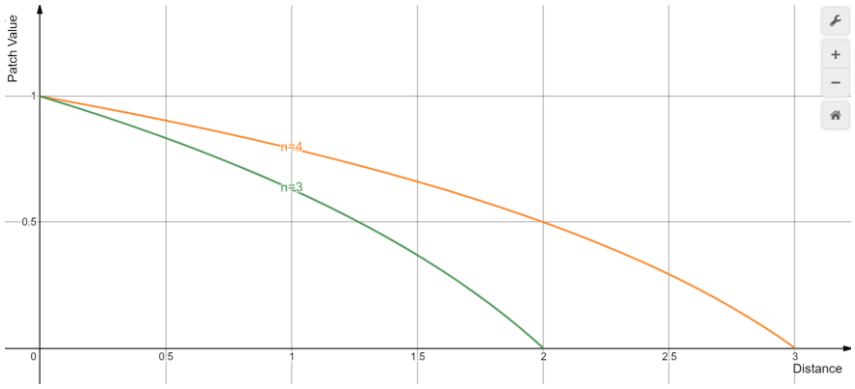


Figure 10. Decay function with different n

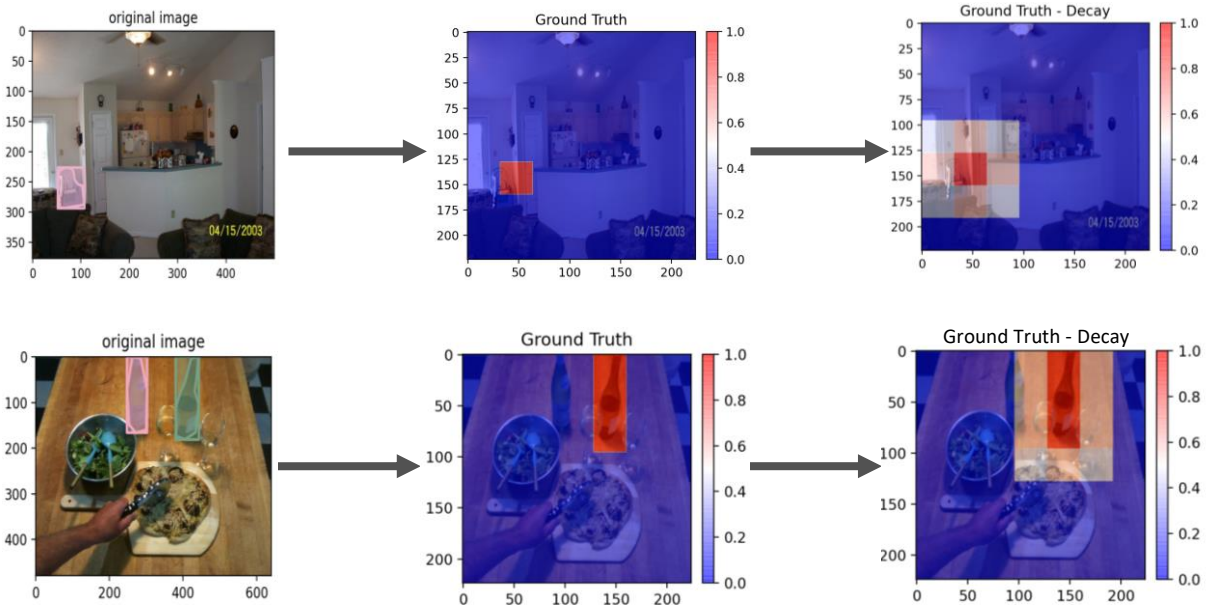


Figure 11. Ground truth enhancement

After applying the decay function to the COCO ground truth, I got a better annotation covering broader regions as the likely locations. In the following section, I will call this annotation the *"decay ground truth"*.

I have also tried to use a 3x3 Gaussian filter to blur the COCO ground truth; however, the Gaussian may make some isolated positive patches to have a patch value smaller than one (see Figure 12 centre). This will dilute the influence of the original COCO ground truth. Although there is some trick to normalising the Gaussian output to map the maximum patch value back to 1, the computation of decay ground truth is cheaper. Therefore, I decided to use the decay ground truth.

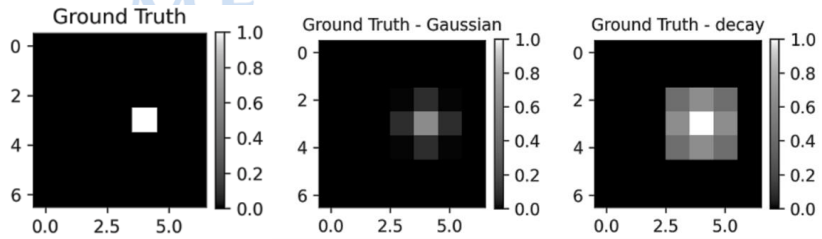


Figure 12. Comparison between Gaussian blur and decay ground truth

3.5. Fixation Data for Evaluation

As discussed in section 3.4, the COCO ground truth is not perfect. Therefore, although the decay function is applied on the ground truth to enhance the data, the enhanced ground truth is still not perfect. For example, the bottom right image of Figure 11 shows the enhanced ground truth of likely

locations of bottles that only cover a small region of the desktop; however, the perfect ground truth should cover the whole empty region of the desktop.

Therefore, I decided to use the COCO-Search18 fixation dataset [23, 24] as a better version of ground truth than the decay ground truth for evaluation. The COCO-Search18 fixation dataset [23, 24] consists of the eye movement tracking from ten human testers to find specific objects in images; hence, every image in the COCO-Search18 fixation dataset [23, 24] contains ten eye movement traces.

In total, there are 203 images for finding "chair" and 103 images for finding "bottle". The amount is not enough to train my ViT model; therefore, I only use the fixation data as the ground truth for evaluating the model performance. For training, I still use the decay ground truth.

People tend to find the target object by using contextual priming. Their eyes will fast scan through likely locations where they believe the object should be placed by referencing their prior experience (see Figure 13). Therefore, it is reasonable to use the human fixation data to serve as the ground truth possible locations of the target object.

Here, the ten fixations in each image are converted into patch representation by labelling all the patches that contain more than two fixation points (the points where the eye stops, see the red dots in Figure 13) as positive. The threshold for a patch to be labelled as positive is set to two fixation points to avoid error from unfocused people. Also, when counting the fixation points, the start points (the blue dots in Figure 13) are excluded because all human testers were asked to start the searching from the centre of an image. I call this annotation the *"fixation ground truth"*.

Figure 13. Fixations for searching bottle

Table 1 shows the result annotations after the conversion. The Fixation Ground Truth (no voting) in Table 1 shows the result when the threshold for a patch to be labelled as positive is set to one fixation point. It shows that some people are not focused enough; hence they look at some unreasonable

regions for bottles (e.g., the window). On the other hand, the Fixation Ground Truth (voting) in Table 1 shows the result when the threshold is set to 2 fixation points.

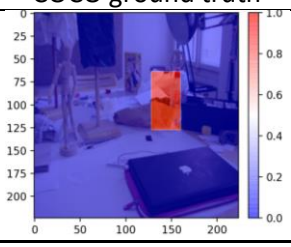
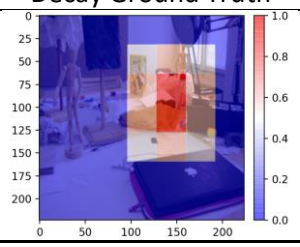
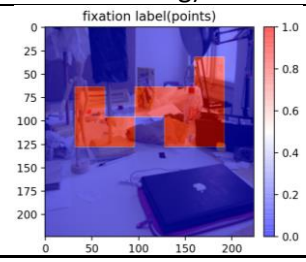
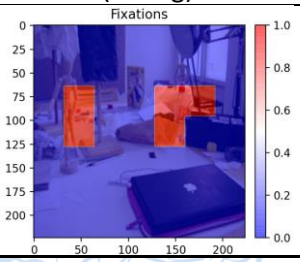
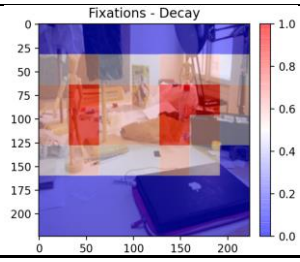
Original Image	COCO ground truth	Decay Ground Truth
		
Fixation Ground Truth (no voting)	**Fixation Ground Truth (voting)**	**Decayed Fixation Ground Truth**
		

Table 1. Ground truths comparison

Although the fixation ground truth is better than the COCO ground truth (see Table 1), it is still not perfect. There are two main limitations. First, people stop the scanning when they find the object; hence, the fixation ground truth may not cover all likely locations of the target objects. Second, people tend to ignore the empty regions because they know there is not any object there (their jobs are only finding the object, not finding all the likely locations). For example, in Figure 13, people did not look at the empty regions of the desktop. However, those empty regions of the desktop should be the possible locations of bottles for my task. Due to these two limitations, again, I apply the decay function on the fixation ground truth to further extend the probabilities to the neighbour patches of the positive patches (see Decayed Fixation Ground Truth in Table 1). I call this enhanced annotation as *"decayed fixation ground truth"*.

Again, even though the decayed fixation ground truth is better than the original COCO ground truth, it is still not perfect. The perfect ground truth should be tailored to my task by human annotators to cover all the possible locations of the target object in the scene.

3.6. 40% Dropping in Training

As my goal is only to use the scene context information to predict the likely locations of the target object, I do not want the model to learn the local feature of the target object. Therefore, during training, I perform 40% dropping. I drop all the patches that contain the target object (i.e., the ground truth positive patches) for each training image by filling zero to all pixels in those patches. This

dropping approach can ensure the model never sees the target object, therefore, never learns the local features of the target object during training.

However, suppose only the patches that contain the target object are dropped. In that case, it will be an obvious cue for the model to predict high probabilities to all dropped patches while predicting zero probabilities to all the other patches. Hence, instead of only dropping the ground truth positive patches, I also drop other random patches to reach a total of 40% of patches being dropped (see Figure 14, where all the patches containing a bottle and some random patches are dropped). Although the ground truths are always in some of the dropped patches, the model will not be told which dropped patches are the ground truth. Therefore, this method can force the model to learn and improve during the training.

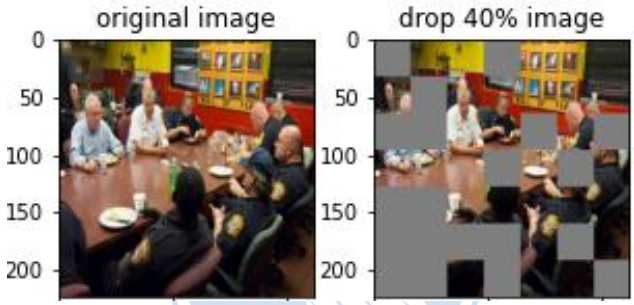


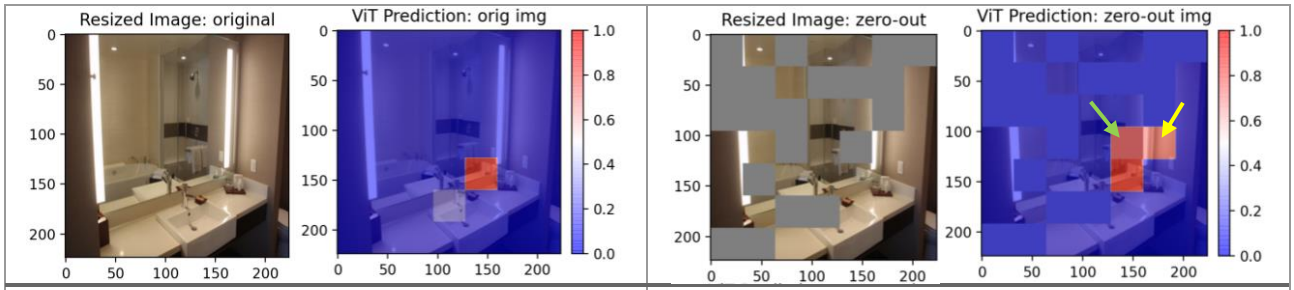
Figure 14. 40% dropping for bottles

3.7. 10 Times Dropping in Test

As mentioned in the previous section, the ground truths are always in some of the dropped patches during training. This encourages the model to learn to pick the ground truth from the dropped patches; therefore, the dropped patches' pattern would dominate the prediction results. In order to be consistent with the training, 40% dropping is also needed in test time. However, the 40% dropping should be entirely random during the test time because the model should not know where the ground truth positive patches are in advance.

Table 2 shows the prediction results of “bottles” on the original test image (without dropping any patches) and on the test image with 40% patches randomly dropped. We can observe that both prediction results are not ideal. This is because my model not only learns the context information but also learns to observe the pattern of the dropped patches and assign high probabilities to some of the dropped patches and their neighbours. Hence, the model tends to predict low probabilities for almost all patches for the test image without dropping because it does not observe any dropped patches (see Table 2 left). On the other hand, for the test image with 40% random patch dropped, the model will select some of the dropped patches as the likely location by referencing the dropping patch pattern

and the scene context (see Table 2 right, the green arrow indicates the dropped patch selected by the model as the likely location and the yellow arrow indicates the neighbour patch that also is selected).



Prediction on the original test image	Prediction on 40% dropped test image

Table 2. Predictions on test image without or with 40% dropping

Since we observe that the prediction result is highly dominated by the pattern of dropped patches, only randomly dropping once for each test image is problematic. I need to repeat the randomly 40% dropping on a test image multiple times, make predictions on them, and average the prediction results as the final prediction to ensure every patch is dropped at least once and all patches are dropped equal times. This can ensure that every patch in the image has the same chance of being assigned probability by the model.

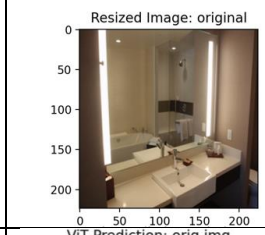
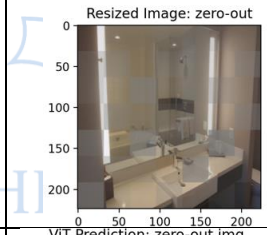
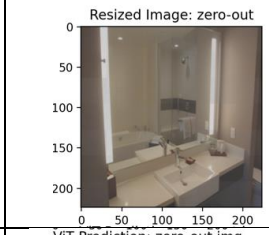
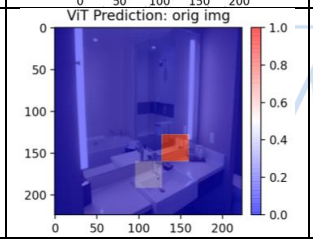
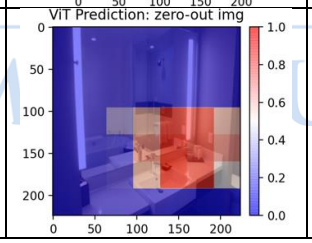
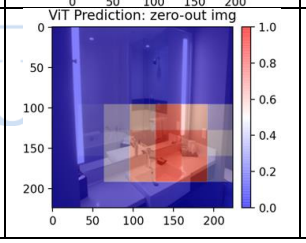
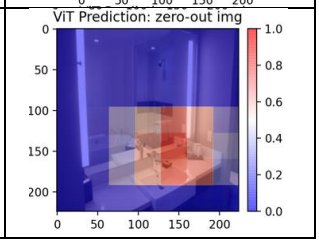
	Original image	Randomly drop 10 times	Randomly drop 20 times	Randomly drop 200 times
Average image				
Prediction				

Table 3. Dropping times comparison

Table 3 shows the comparison of randomly dropping different times on the test images. The first row is the average of n times randomly dropping test images. If a patch is greyer, it means that the patch is dropped more times. We can observe that the randomly dropping 10 and 20 times still show the grid pattern, which means that the patches inside them are not dropped evenly. I observe that we need at least 200 times random dropping to have every patch in the image be dropped approximate equal times. However, it is inefficient to drop and predict 200 times on a test image and then average their prediction results as the final output. Hence, I design a customised dropping strategy that only drops ten times and ensures every patch is dropped precisely four times. The customised dropping

strategy is shown in Figure 15. It contains ten different dropping patterns, and each pattern contains precisely 40% patches dropped (black means the patch is dropped). This strategy contains two sets of dropping patterns (see the first and second rows in Figure 15) instead of only using one of them to prevent the direction of the pattern from affecting the prediction results. The averaging prediction result is shown in Table 4; we can observe that the result is very similar to the averaging prediction results of random dropping 200 times in Table 3. This proves that this dropping pattern is reasonable.

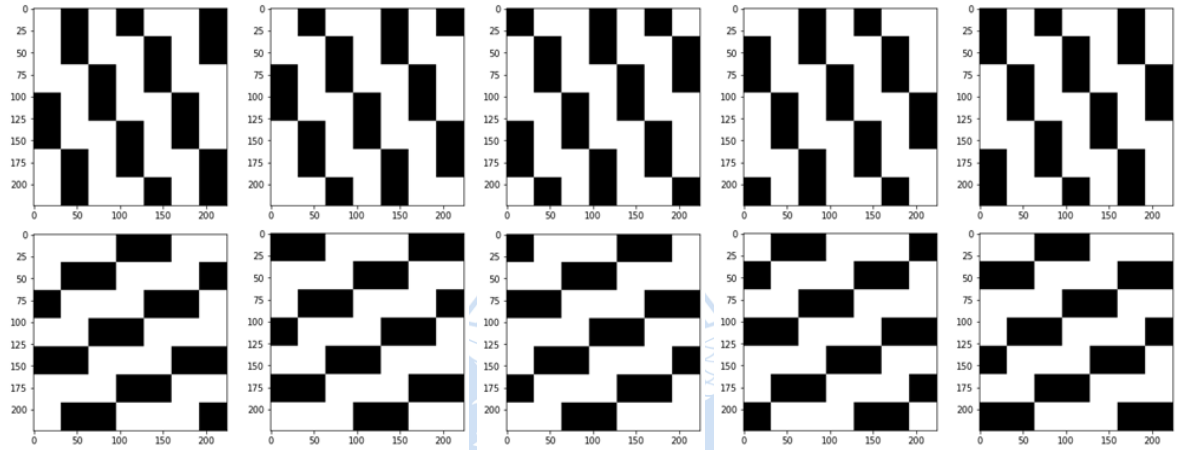


Figure 15. Customised dropping strategy

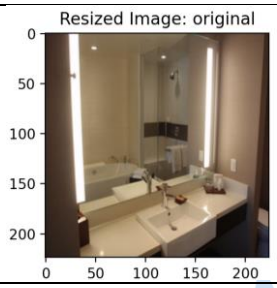
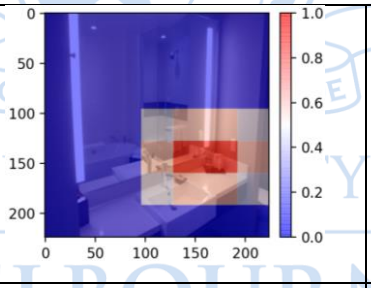
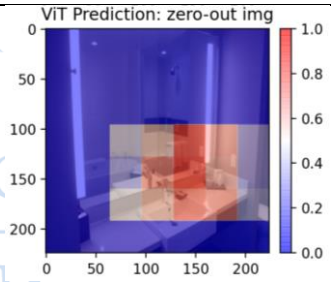
		
Original image	Decay ground truth	Average prediction results of customised 10 times dropping

Table 4. Prediction results by using customised dropping strategy

3.8. Evaluation Metrics

To evaluate the model's performance, I will use the decayed fixation ground truth data because it is the available closet data to the perfect ground truth (see section3.5). For the evaluation metrics, I decided to use mean average precision (mAP) as my primary metrics and the intersection over union (IoU) as the secondary metrics; I think mAP can best describe the performance on binary ground truth label. On the other hand, the IoU can demonstrate the ratio of overlapping between the prediction and ground truth; however, I think the IoU is less meaningful because the decayed fixation ground truth does not cover all the likely locations (see section 3.5). The decayed fixation ground truth is narrow than the perfect ground truth. Hence, using IoU to judge the model performance will

encourage the model to make a narrow prediction that is not desirable to my task as I want the model to detect all the likely locations.

3.9. Loss Function and Penalty

For the loss function, I decided to use the binary cross entropy (BCE) with logits loss [25] because my ground truth label is binary (positive means the patch is the likely location of the target object; negative means is not). Moreover, the binary cross entropy with logits loss in PyTorch [25] has an additional parameter called *"pos_weight"* to handle the imbalance between positive and negative labels. The loss function is shown below:

$$BCELoss = -\left[\boldsymbol{pos\_weight} \times y \times log\big(\sigma(x)\big) + (1-y) \times log\big(1 - \sigma(x)\big)\right]$$

$$y: the\ decay\ ground\ truth\ label.$$

$$x: predicted\ probabilities\ of\ all\ patches\ in\ the\ image\ \big(P(object|patch)\big).$$

As the COCO dataset only labels the target objects, the ratio of positive and negative patches in the dataset is very imbalanced. On average, there are 45.7 ground truth negative patches and 3.3 ground truth positive patches in an image. Therefore, I found that the setting of pos_weight is critical; it will determine how much should the model focus on the positive labels.

If the pos_weight is set to 1, the positive and negative labels have the same weight. However, because there are many more negative labels than positive ones, negative ones will dominate the learning. Therefore, the model will prevent the wrong prediction on the negative labels as much as possible; hence, it will predict low probabilities on all the patches.

On the other hand, if the pos_weight is set to a large value (e.g., 14), the model will weigh the positive labels too much. This will lead to the model assigning high probabilities to all the patches.

In theory, this pos_weight parameter should be equal to the negative and positive labels ratio, which are 13.84 (45.7/3.3) for our dataset. However, I had applied the decay function on the ground truth data as data enhancement (see section 3.4), so the actual influence of positive patches will be slightly more than 3.3. Hence, the pos_weight should be slightly smaller than 13.84. After multiple experiments, I found that setting pos_weight to 10 perform the best. Hence, the BCELoss function will be:

$$BCELoss = -\left[\mathbf{10} \times y \times log\big(\sigma(x)\big) + (1-y) \times log\big(1 - \sigma(x)\big)\right]$$

Still, only using BCELoss is not perfect enough. As I mentioned in section 3.6, the ground truths are always in some of the dropped patches during training, so the model may act lazy to predict high

probabilities on all the dropped patches. In contrast, predict low probabilities on all the other patches (see Figure 16). To prevent the model from capturing the pattern of dropped patches, I added an extra penalty term to the loss function. The penalty is defined as:

$$Penalty = 1 - MSE(prediction, patten\ of\ dropped\ patches)$$

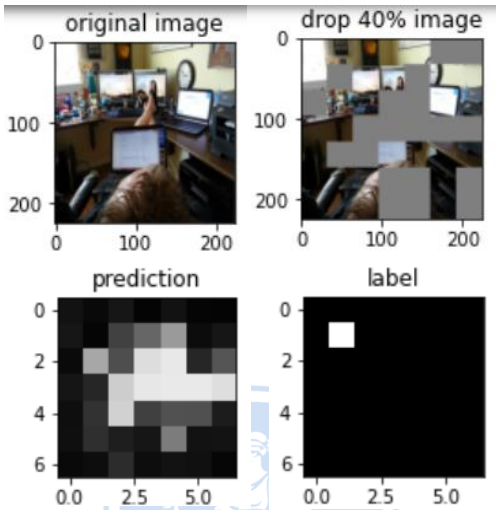


Figure 16. Prediction dominated by the pattern of dropped patches

The mean square error (MSE) measures the similarity between the prediction result and the pattern of dropped patches. Hence, if the prediction is too similar to the pattern of dropped patches, the penalty will be high.

I did experiments on different weights combinations of the BCELoss and the penalty and computed four evaluation metrics (mAP, recall, precision, IoU). Those four evaluation metrics are computed over the whole decayed fixation ground truth data for the target “bottle”.

The results are shown in Table 5. For visualisation, I plot the prediction result on the 40% dropped image to demonstrate how the pattern of dropped patches affects the prediction results.

Based on the mAP, the 90% BCELoss with 10% penalty performance was the best. Therefore, the final loss function is:

$$Loss = 0.9 \times BCELoss + 0.1 \times Penalty$$

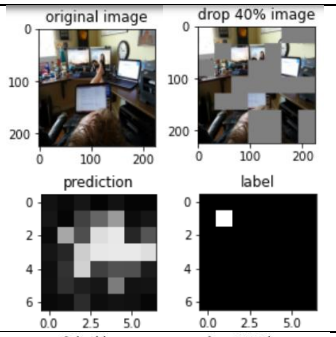
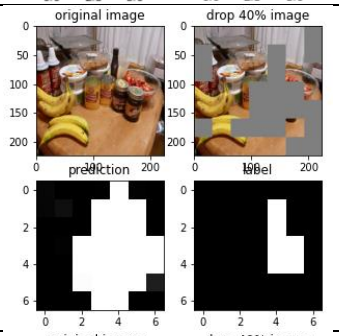
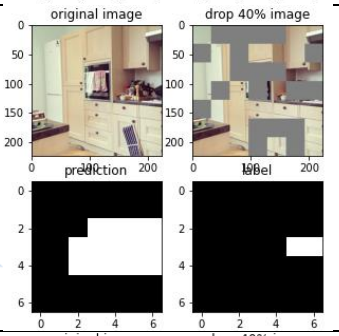
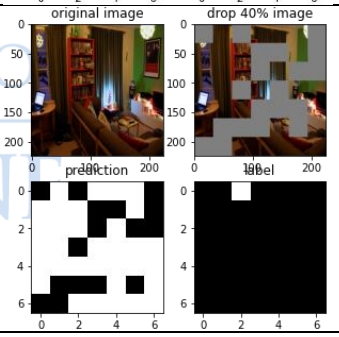
Loss Function	Findings and Performance	Visualisation
100% BCELoss	No penalty, the predictions are highly dominant by the pattern of dropped patches mAP = 0.7183 recall = 0.3041 precision = 0.8747 IoU = 0.416	
90% BCELoss + 10% Penalty	Similar performance to the 80/20 loss. mAP = 0.7646 recall = 0. 4961 precision = 0. 7961 IoU = 0. 5571	
80% BCELoss + 20% Penalty	Similar performance to the 90/10 loss. mAP = 0. 7485 recall = 0. 5734 precision = 0. 7427 IoU = 0. 5711	
70% BCELoss + 30% Penalty	Penalise too much; the model predicts zero probabilities to all the dropped patches. The four evaluation metrics are not discussed here because the prediction is clearly wrong.	

Table 5. Loss functions comparison (lr=0.000001)

3.10. Training

All my models in this project are trained using the pre-trained weights on the ImageNet dataset [26] as initialisation. Those pre-trained weights are from implementing the original vision transformer for an image classification task [17, 22]. Previous researches [17, 21] indicate that pre-trained weights trained on a large dataset are necessary to introduce image-specific inductive biases to ViT and make the ViT model reach the SOTA performance.

Chapter 4: Results

In this project, I train my model for two target objects which are “bottle” and “chair”. I will show the prediction result, the learning curve, and the performance of models for these two classes of objects. I will name the model as “Ratio of loss function _ class _ learning rate”. For example, the model “9010_bottle_e-6” means this model is trained with a 0.000001 learning rate, 90% BCELoss, 10% penalty, and it is for finding the likely locations of bottles in images.

4.1 Learning Curve

As discussed in section 3.9, the loss function with 9010 weights performs the best, and the loss function with 8020 weights performs the second. Hence, only these two combinations of weights are discussed here.

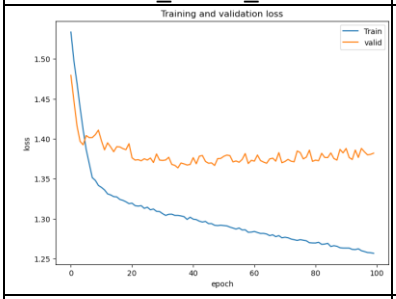
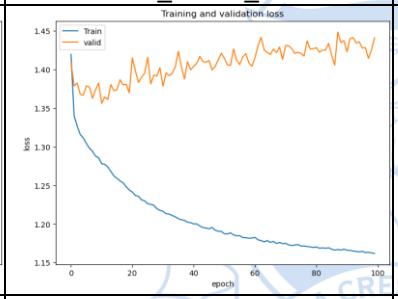
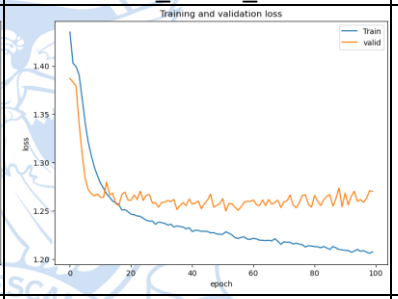
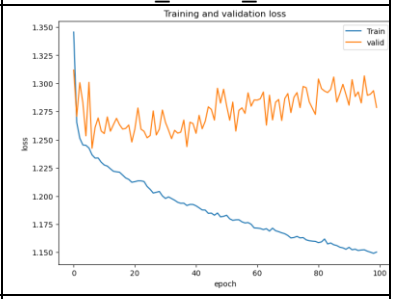
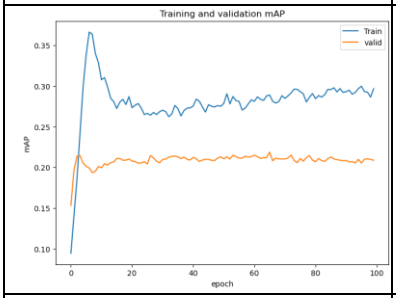
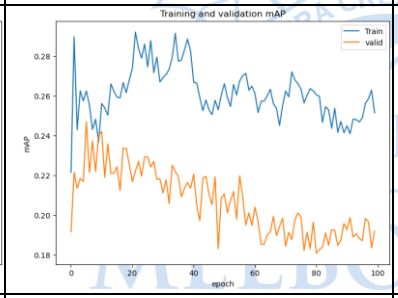
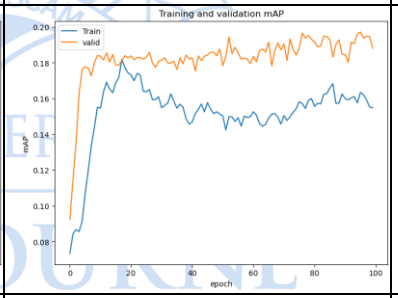
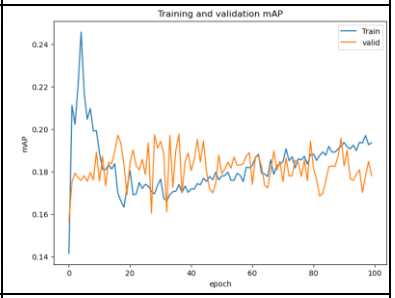
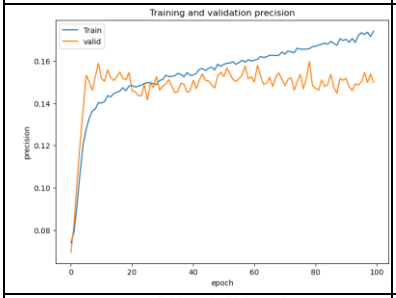
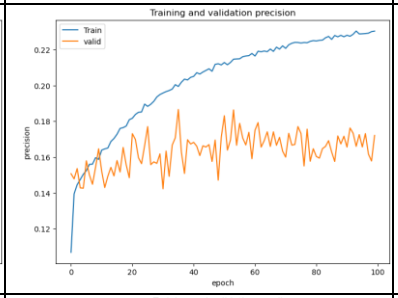
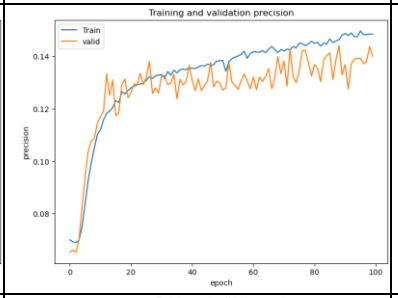
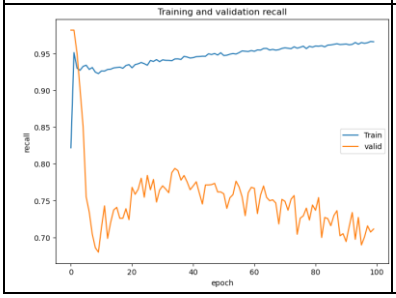
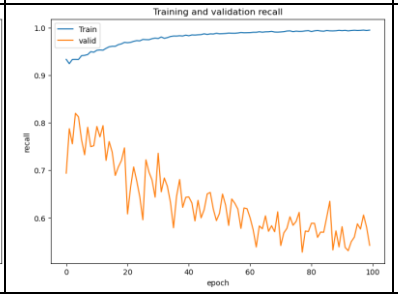
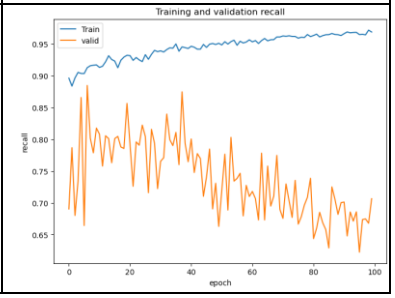
9010_bottle_e-6	9010_bottle_e-5	8020_bottle_e-6	8020_bottle_e-5
			
			
			
			

Table 6. Learning curve comparison for bottle models

Table 6 shows that all the four bottle models converge very fast, and the learning curve of the larger learning rate (e-5) fluctuate more. Moreover, based on the prediction results comparison in Table 7, we can conclude that the model with the e-6 learning rate performed better, and the 9010 model outperformed the 8020 model.

	Prediction result of likely locations of bottle			
Original Image	9010_bottle_e-6	9010_bottle_e-5	8020_bottle_e-6	8020_bottle_e-5

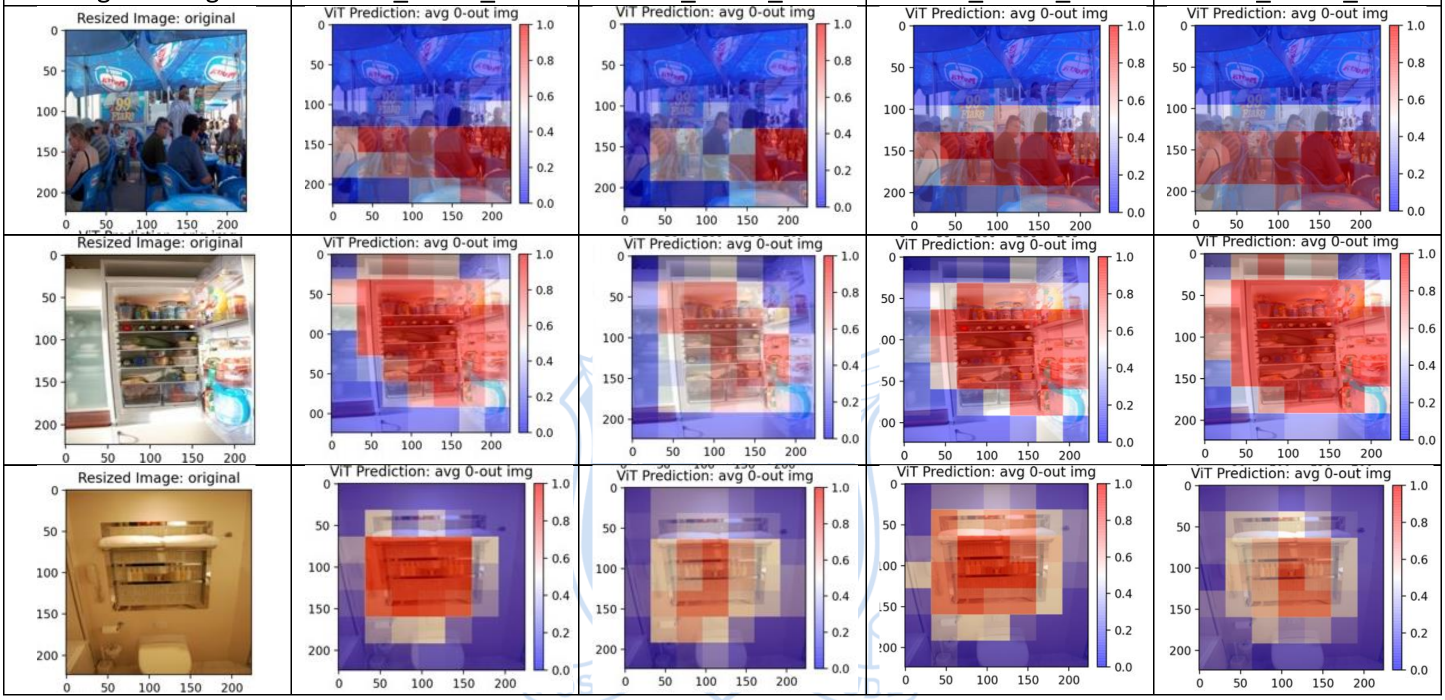


Table 7. Predictions comparison for bottle models

Table 8 shows the performance of the decayed fixation ground truth for different bottle models. Although the 8020_bottle_e-5 model has the best mAP, its prediction results are not the best based on human observation (see Table 7). This is again caused by the non-perfect annotations.

	9010_bottle_e-6	**9010_bottle_e-5**	**8020_bottle_e-6**	**8020_bottle_e-5**
mAP	**0.7646**	0.7377	0.7485	**0.7665**
Recall	0.4961	0.3858	0. 5734	0.5249
Precision	0.7961	0.8580	0.7427	0.7994
IoU	0.5571	0.5152	0. 5711	0.5724

Table 8. Evaluation performance comparison for bottle models

Again, Table 9 shows that the chair models also converge very fast, and the larger learning rate (e-5) cause the learning curve to fluctuate. Table 11 and Table 10 prove that the chair model with a small learning rate performs better.

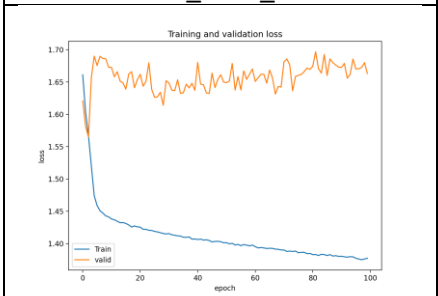
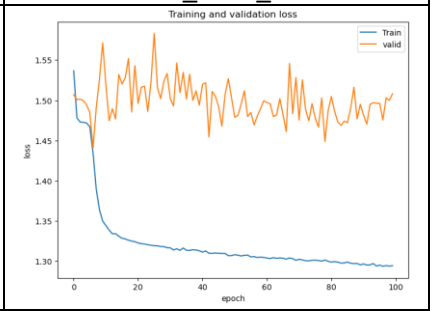
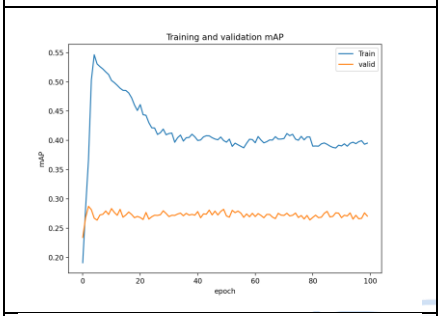
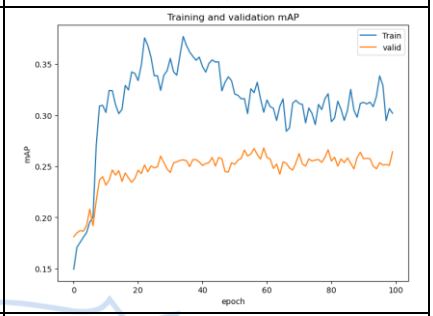
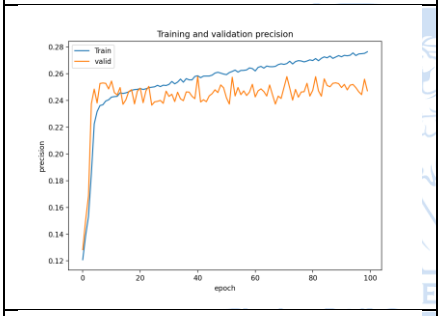
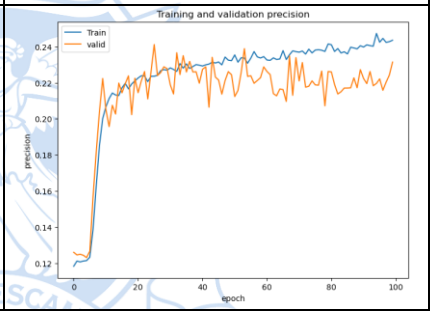
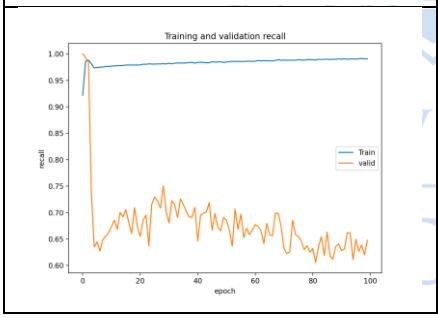
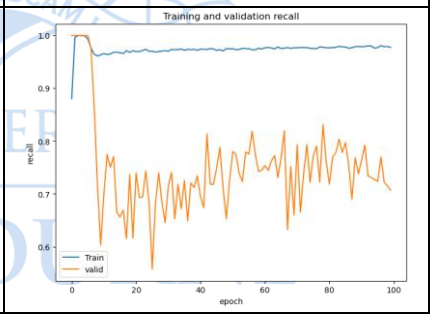
9010_chair_e-6	8020_chair_e-5
	
	
	
	

Table 9. Learning curve comparison for chair models

	9010_chair_e-6	**8020_chair_e-5**
mAP	**0.6325**	0.6101
Recall	0.5435	0.6154
Precision	0.6010	0.5655
IoU	0.5405	0.5371

Table 10. Evaluation performance comparison for chair models

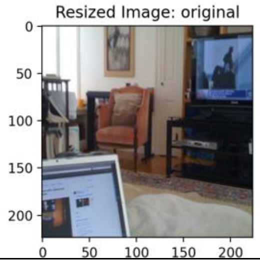
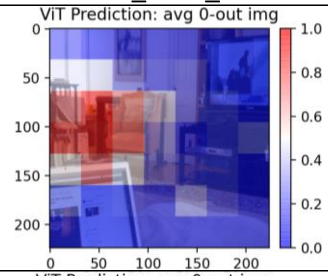
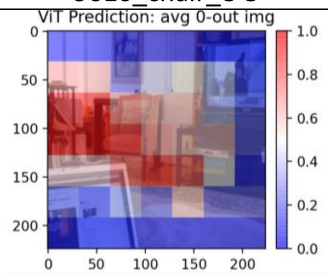
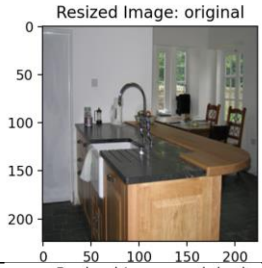
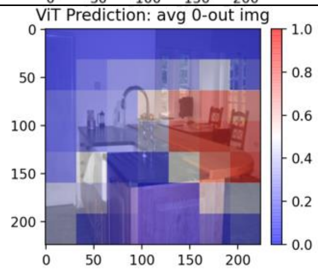
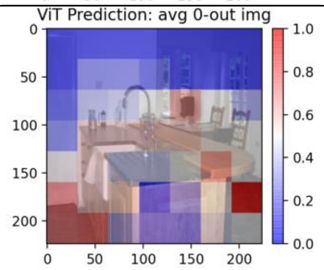
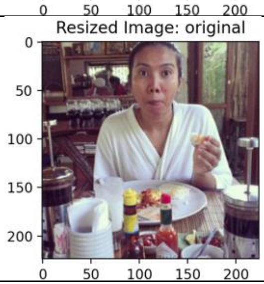
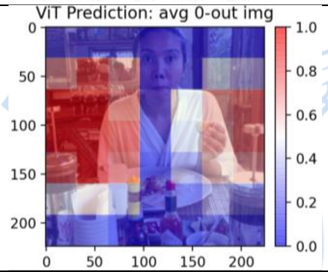
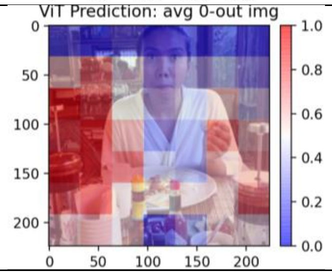
Original	Prediction result of likely locations of chair	
	9010_chair_e-6	9010_chair_e-5
		
		
		

Table 11. Predictions comparison for chair models

In this section, I observed that the 9010 models with smaller learning rates perform better for both bottle and chair models. Hence, in the following sections, I will only show the prediction results of the 9010_bottle_e-6 and 9010_chair_e-6 models.

4.2 "Bottle" Model Prediction Results

Table 12 shows the prediction results of model 9010_bottle_e-6 on the scene that have bottles inside. As we can observe, my model not only successfully find the bottles in the scene, but it can also find the likely locations where it is reasonable to place a bottle (e.g., desktop, shelf).

Table 13 shows the prediction results of bottles on images without any bottles inside. As we can see in the first row of Table 13, my model performs well in the indoor scenes and successfully finds reasonable locations such as desktops and tables. However, my model performs poorly on unfamiliar outdoor scenes (see the second row of Table 13) where there should not have any bottles.

Figure 17 shows an interesting finding that my bottle learned to capture the human hand as the likely location of bottles.

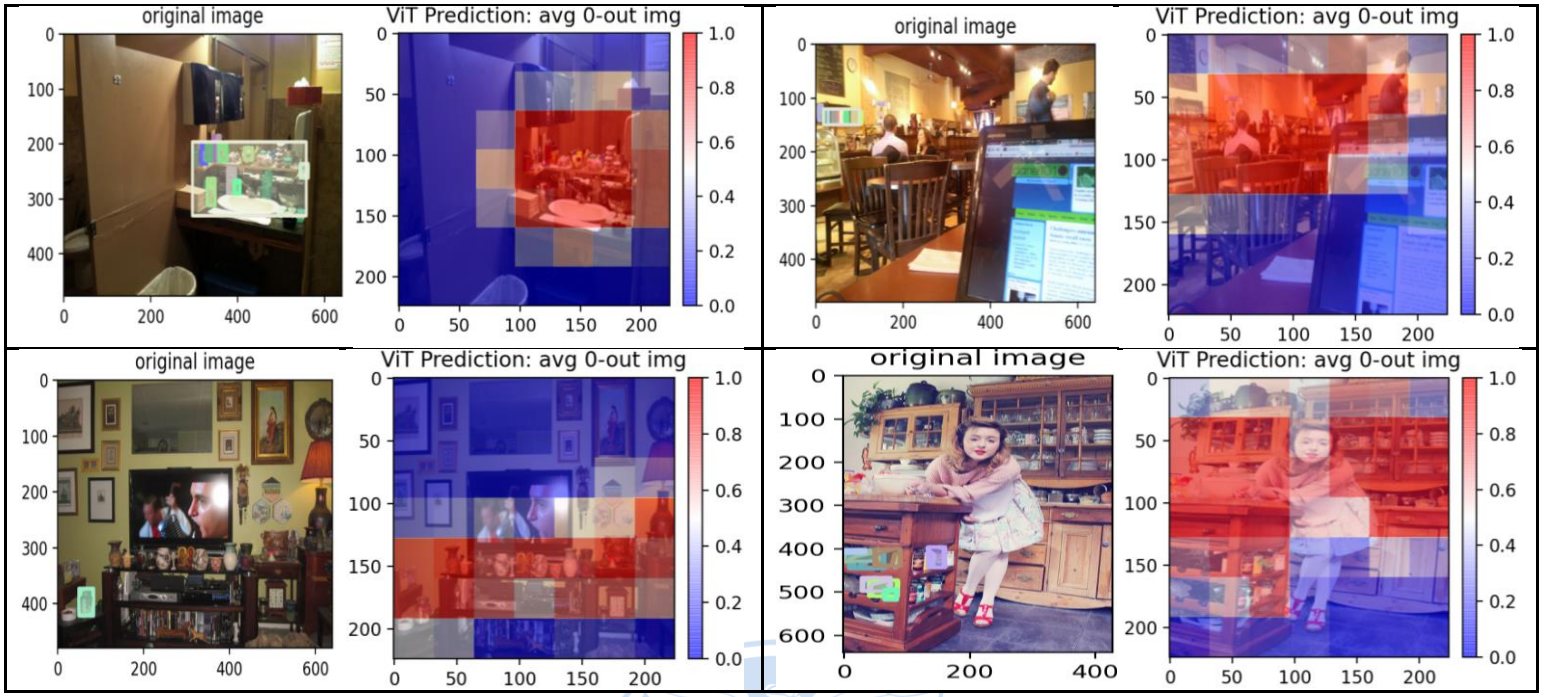


Table 12. Prediction on the scenes with bottles

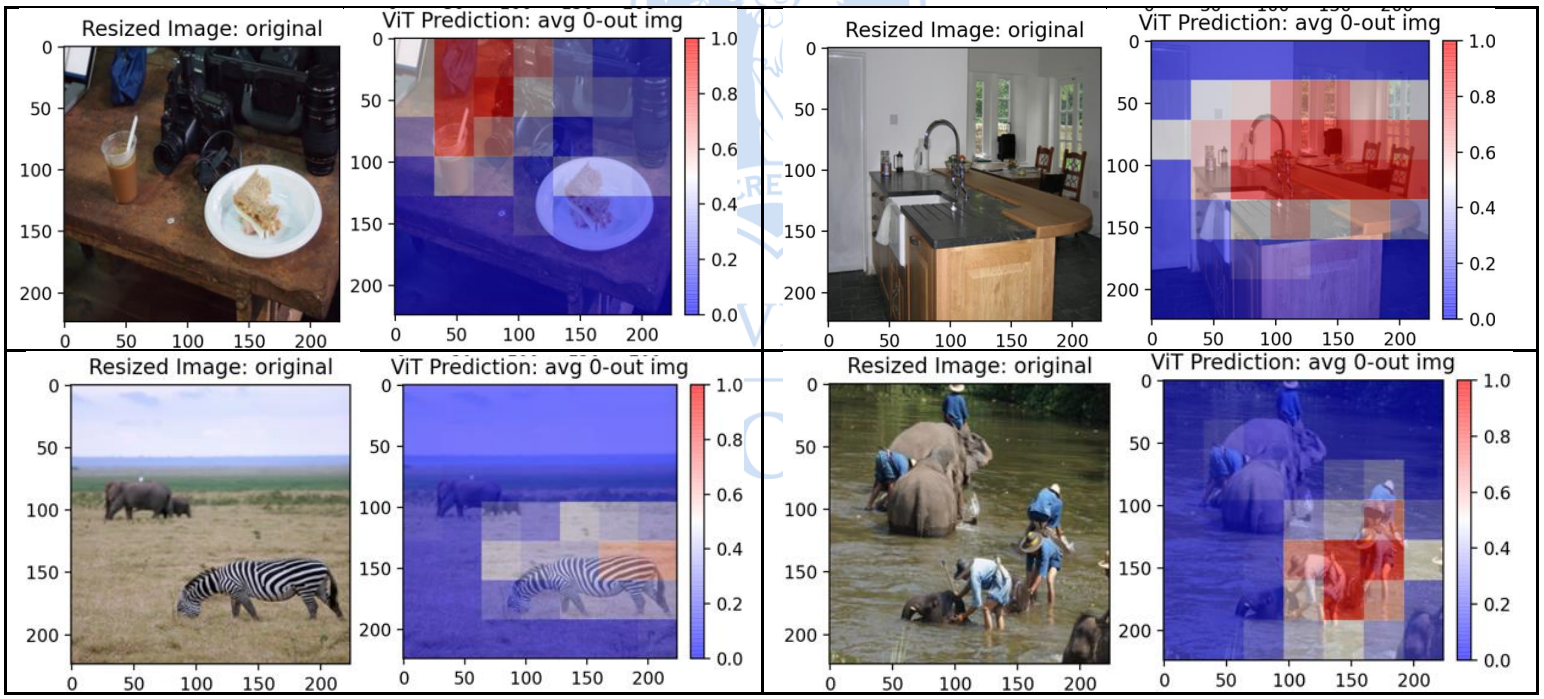


Table 13. Prediction on the scenes without bottles

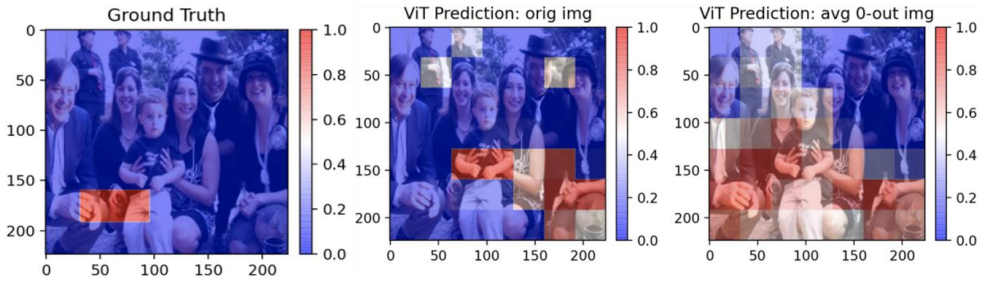


Figure 17. Bottle model captures human hands

4.3 “Chair” Model Prediction Results

Table 14 shows the prediction results of model 9010_chair_e-6 on the scenes that have chairs inside. As we can observe, this model can detect the likely locations no matter the indoor or outdoor scenes. The model finds chairs in the scene and finds all the possible locations of chairs (e.g., near a bed, near a desk, on an open surface, near the basketball court).

Table 15 shows the model labels the regions near desks and the edges of desks as the possible locations for chairs. This demonstrates that the model can understand the scene.

Table 16 shows the predictions on scenes without chairs inside. As observed in the first row of Table 16, this model performs poorly on those unfamiliar outdoor scenes. However, its prediction is not entirely unreasonable. This model highlights the open flat surfaces (e.g., grass, road) as possible locations to place a chair. The second row of Table 16 shows that the model predicts the regions near humans as the possible chair locations.

Table 17 indicates an interesting finding that the chair learned to predict the human shoulders and the sitting peoples as the likely locations of chairs.

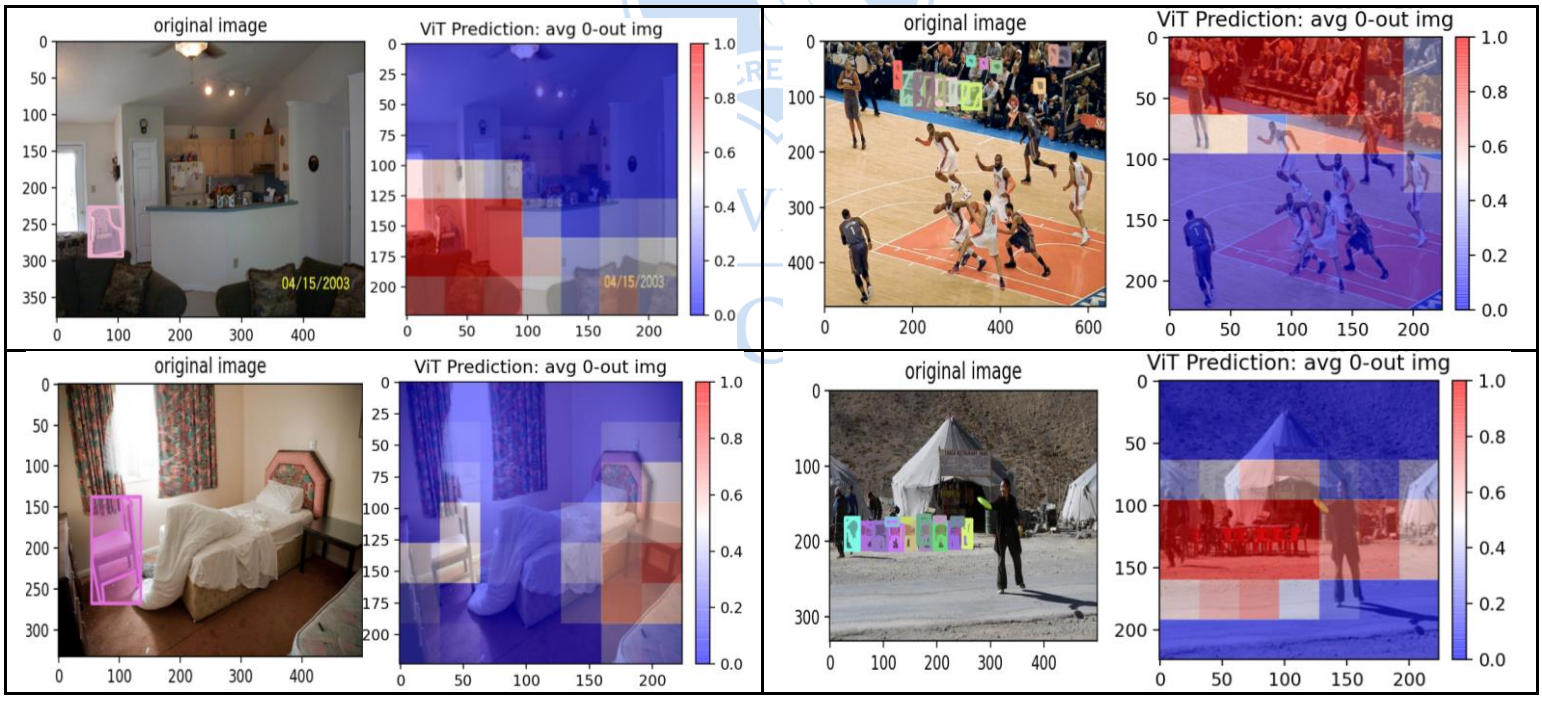


Table 14. Predictions on scenes with chairs

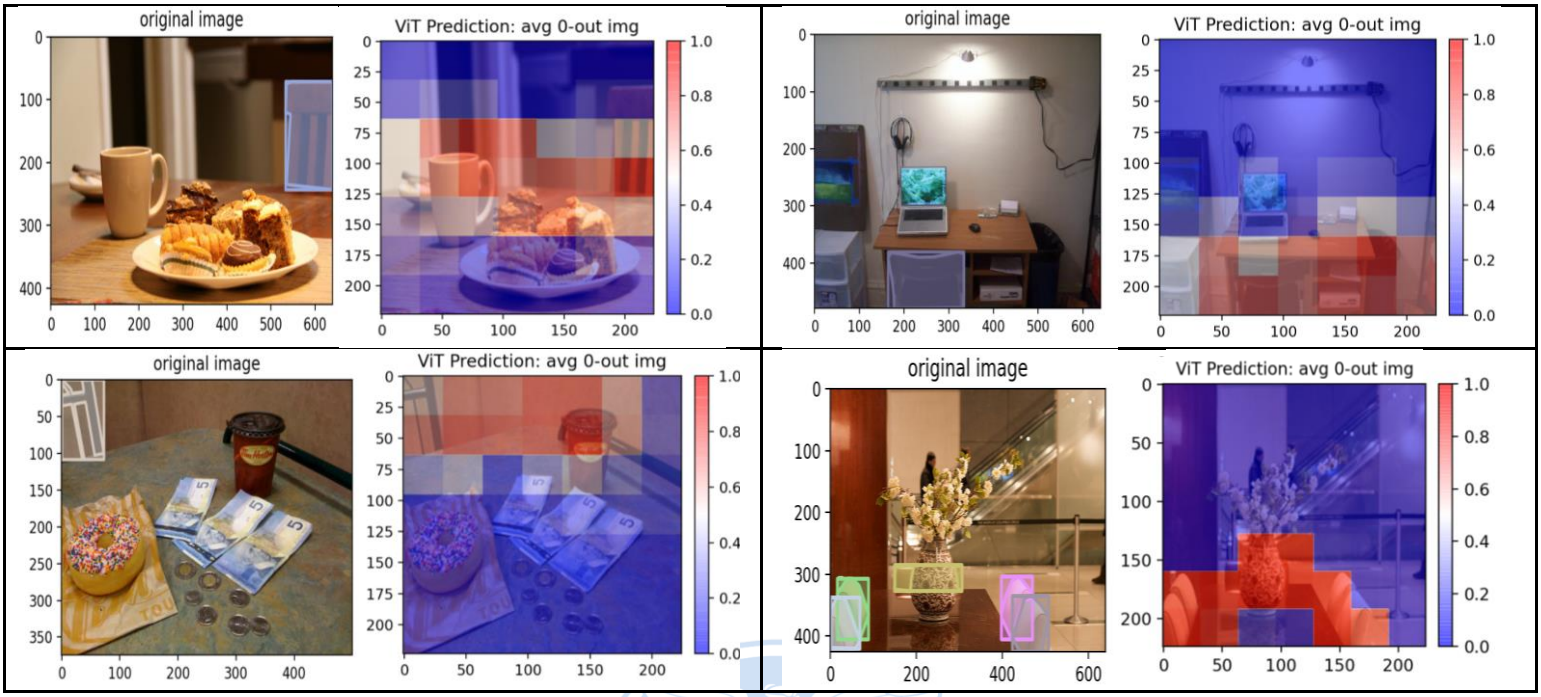


Table 15. Predictions of chairs – find the edge of tables

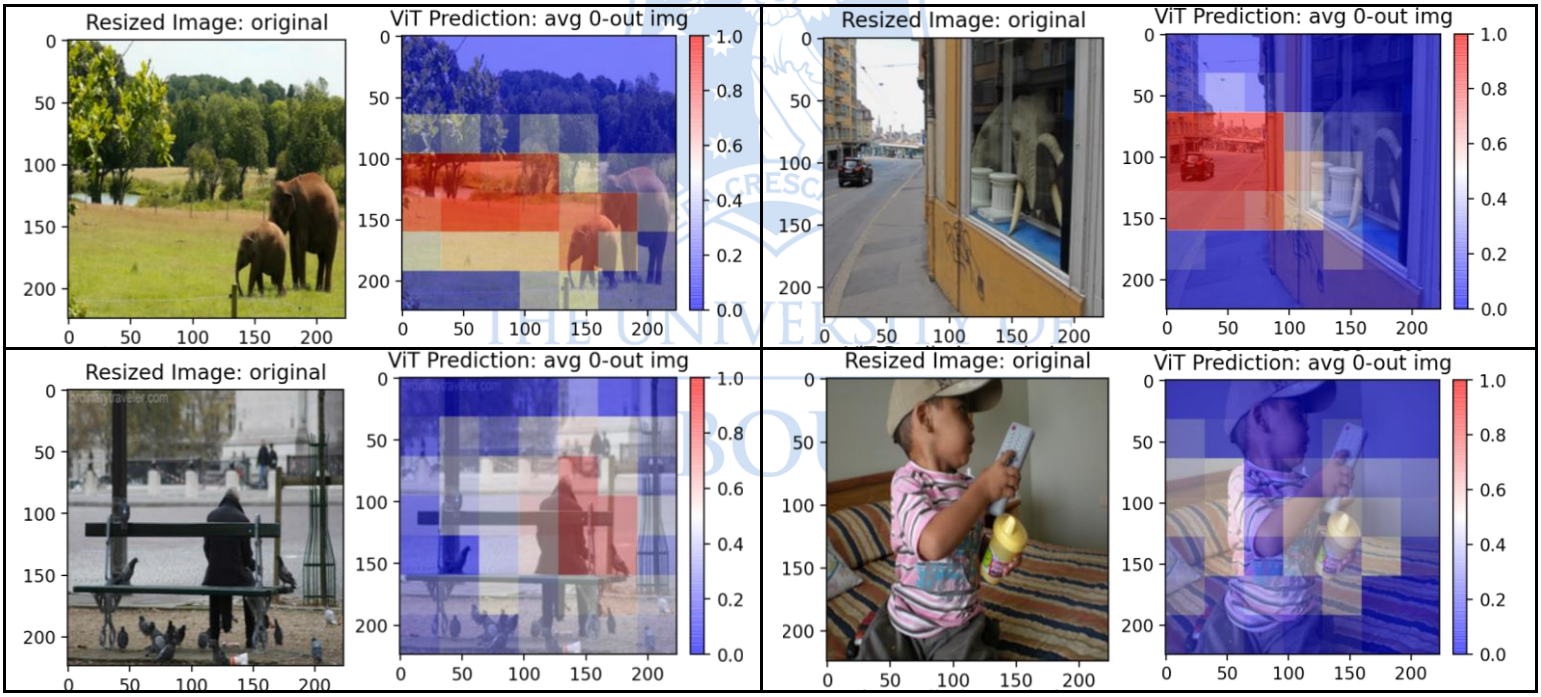


Table 16. Prediction on scenes without chairs

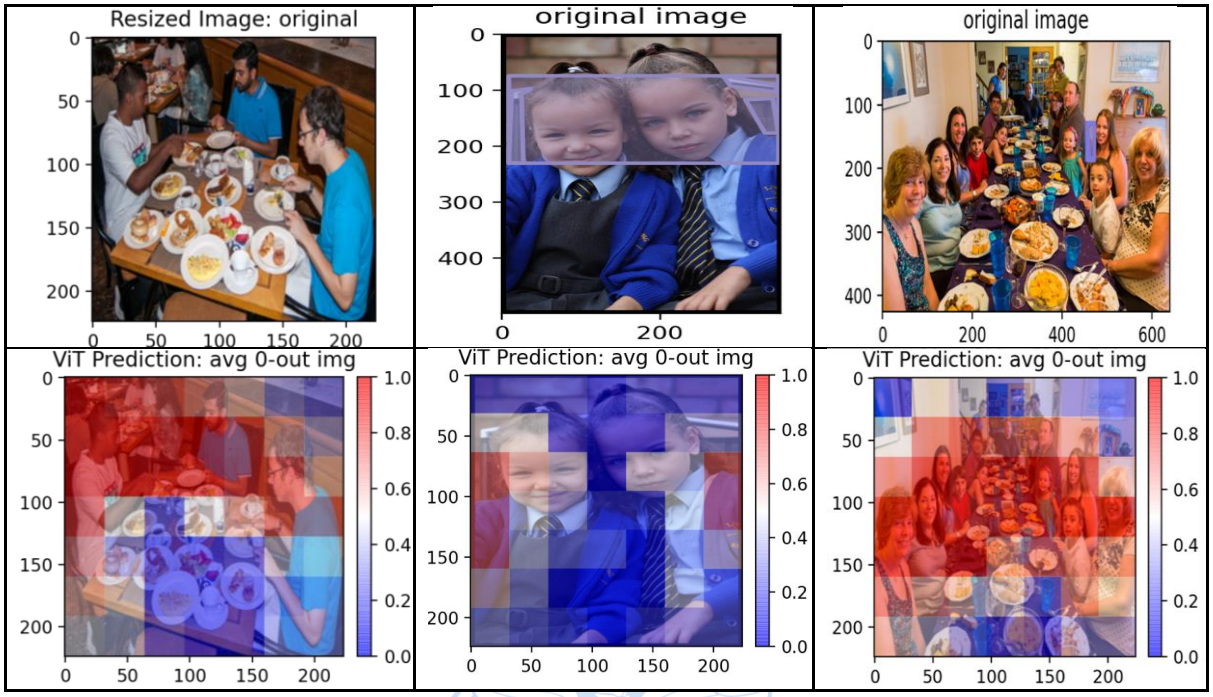


Table 17. Chair model captures human shoulders and sitting people

4.4 Prediction Speed

This model spent 736.54 sec on making predictions for 13300 images on Tesla P100-PCIE-12GB GPU. This means its prediction speed is 18.06 FPS. Notice that a single prediction for one image actually run the model 10 times because the model implements ten times dropping and averages their prediction results during inference (see section 3.7). Hence, if we do not implement the ten times dropping and only run the model once per image, the prediction speed will be 180.6 FPS.

The prediction time mentioned above includes the time spent on loading images from the dataset. Therefore, if we exclude the reading time, the prediction speed could be faster.

Chapter 5: Discussion and Analysis

5.1 Learning Progress

This section will discuss how my vision transformer model learns and why it learns/converges so fast. According to the training curves (see Table 6 and Table 9), both the bottle and chair models converge within the first 20 epochs.

Let us go back to the filters in the patch embedding layer (see Figure 7) and the model architecture (see Figure 8). As Figure 7 shows, the filters learned in the patch embedding layer (one convolution layer) are simple, which means they capture the low-resolution (coarse) features in the patches. Those low-resolution features of every patch will then be fed into transformer blocks to learn to attend features from other patches. Hence, the learning logic of my model is first extracting low-resolution (coarse) features by simple CNN inside the patch itself and then improving the extracted features by attending features from other patches.

As the model only needs to learn coarse features locally (within the patch itself) and attend other patches globally, I think the learning is simpler than the learning in a deep CNN. Moreover, my model uses the pre-trained weights trained on a large dataset (the ImageNet [26] dataset). The pre-trained weights introduce enough image-specific inductive bias [17, 21]; therefore, much simplifies the learning for my task. I believe the two reasons mentioned above is why my model converges so fast.

Table 18 shows the prediction results at different epochs. It demonstrates that the model converges within the first 20 epochs.

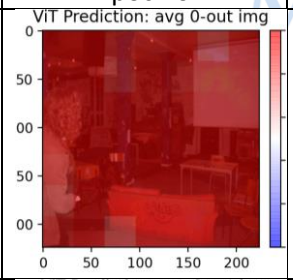
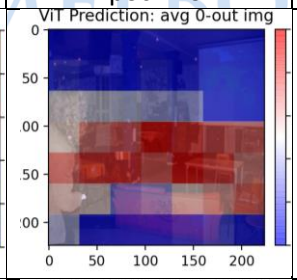
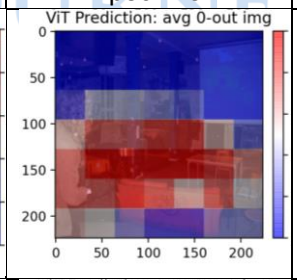
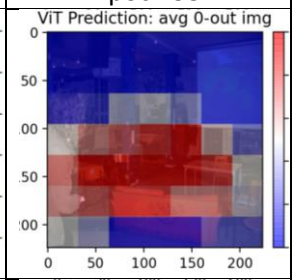
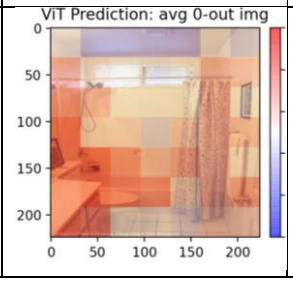
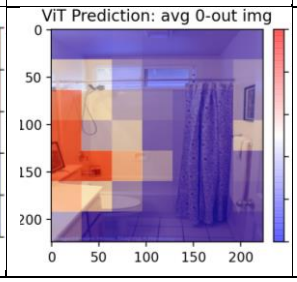
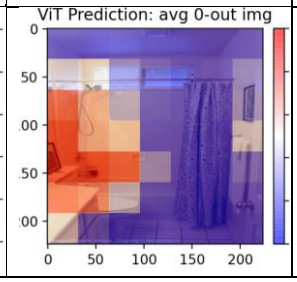
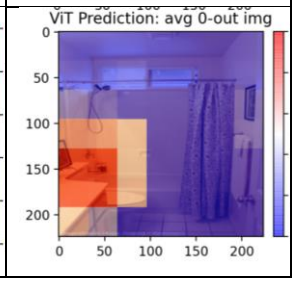
	Epoch 0	Epoch 10	Epoch 20	Epoch 95
Chair Model				
Bottle Model				

Table 18. Learning progress

5.2 Model Performance

Section 1.2 shows that the bottle model performs well in indoor scenes no matter if there are any bottles in the scenes (see Table 12 and the first row of Table 13). We can observe that the bottle model can understand the scene by finding the surface, desktop, or shelf.

However, the bottle model performs poorly in wild outdoor scenes that it is unfamiliar with (see the second row of Table 13) because my model never saw those scenes during training. Even though the scene is entirely unrelated to bottles, the model still tries to find a plain surface or region near a human as the possible locations to place a bottle.

Figure 17 shows that the bottle model can detect human hands as the likely locations of bottles because, in the COCO dataset [9], there are many images of humans holding bottles. This proves that my model uses the global context and the local features of co-occurred objects in the scene to make a decision.

Section 1.3 shows that the chair model performs well on both indoor and outdoor scenes (see Table 14). It can find the region near a bed, a desk, a tent, or the basketball court by referencing the global structure of the scene. Table 15 shows some close-shot images which have less global scene information and demonstrates that my model still can find the reasonable likely location of bottles. This proves that my model can reference the local features of co-occurred objects in the scene and use it to predict the likely location of chairs. My model can understand the relationship between table and chair, find the table and know that chairs have high probabilities to be placed near a table.

Table 16 shows that my model performs worse on the scene without any chairs. Even if the scene is entirely unrelated to chairs, my model still tries to find flat surfaces (e.g., the grass and the road) and regions near a human as the likely locations to place a chair.

Table 17 shows that my chair model captures the local features of human shoulders & sitting people, and labels them as the likely locations of chairs because there are many images of humans sitting on chairs in the COCO dataset. This again proves that my model can use the co-occurred local features in the scene to make a prediction.

To sum up, my models are able to understand the low-resolution scene context (i.e., the structure of the scene such as understanding where a flat surface is; the co-occurred local features in the scene such as the features of humans), but it does not have a high-level understanding of what the scene is (i.e., it cannot know the scene is wild grassland, indoor room or outdoor park).

Chapter 6: Conclusion and Recommendations

In this project, I aim to build a context-centred model that can use only the low-resolution (coarse) scene context to predict the likely locations of target objects. Using the attention mechanism in the vision transformer [17], I prove that the low-resolution context information can be used to locate an object rapidly without the need of an object-centred detector. Hence, I confirm that machines can simulate the contextual priming mechanism in the human brain. Overall, my bottle and chair models reach 76.46% and 63.25% mean average precision on the COCO-Search-18 fixation dataset [23] with 18.06 FPS detection speed.

However, this project has three limitations. First, since my model can only understand the low-resolution scene context such as the spatial information, the structure of the scene, and the co-occurred local features in the scene, it can not have a high-level understanding of what the scene is. Hence, when the model sees a totally unrelated scene, it will still try to predict some possible locations based on the low-resolution contextual information. Second, there is not a perfect ground truth annotation available for my task. Therefore, the numerical evaluation of the model performance may not be precise. Third, currently, the bottle and chair models are trained separately due to the time limitation. Ideally, the model should be able to make predictions for multiple classes of objects to prevent the need of multiple models and the redundant scene features extraction in multiple models.

To solve the first limitation, I can append an additional CLS token at the front of the sequence of patches to serve as the holistic representation of the scene (the scene prior). For more details, please see Appendix B: Future Work – introducing scene prior by CLS token. To resolve the second limitation, I will need human-labelled ground truth annotations that can faithfully represent target objects' possible locations. For the third limitation, my model can handle multiple classes of objects; I only need more time to adjust the 40% dropping approach. Also, the third limitation must be solved after the scene prior is implemented to prevent wrong predictions when combining the outdoor class of target objects and the indoor class of target objects.

In conclusion, this project proves a machine can have contextual priming and is able to rapidly detect the likely location of target objects even the target objects are not in the scene. This model can be used to replace the region proposal network in the two-step object detectors; used in applications such as augmented reality to guide a user to place a virtual object reasonably; used in autonomous vehicles to fast understand the environment; or used in any applications that need rapidly finding the important regions to focus within an environment.

References

[1] R. N. Henson, "Neuroimaging studies of priming," *Progress in neurobiology,* vol. 70, no. 1, pp. 53-81, 2003.

[2] M. M. Chun and Y. Jiang, "Contextual cueing: Implicit learning and memory of visual context guides spatial attention," *Cognitive psychology,* vol. 36, no. 1, pp. 28-71, 1998.

[3] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440-1448.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems,* vol. 28, pp. 91-99, 2015.

[5] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," in *Advances in neural information processing systems*, 2016, pp. 379-387.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779-788.

[7] W. Liu *et al.*, "Ssd: Single shot multibox detector," in *European conference on computer vision*, 2016: Springer, pp. 21-37.

[8] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767,* 2018.

[9] T.-Y. Lin *et al.*, "Microsoft coco: Common objects in context," in *European conference on computer vision*, 2014: Springer, pp. 740-755.

[10] A. Torralba, "Contextual priming for object detection," *International journal of computer vision,* vol. 53, no. 2, pp. 169-191, 2003.

[11] A. Torralba and P. Sinha, "Statistical context priming for object detection," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, 2001, vol. 1: IEEE, pp. 763-770.

[12] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Hypercolumns for object segmentation and fine-grained localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 447-456.

[13] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117-2125.

[14] S. Bell, C. L. Zitnick, K. Bala, and R. Girshick, "Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2874-2883.

[15] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *arXiv preprint arXiv:1504.00941,* 2015.

[16] A. Shrivastava and A. Gupta, "Contextual priming and feedback for faster r-cnn," in *European conference on computer vision*, 2016: Springer, pp. 330-348.

[17] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929,* 2020.

[18] A. Vaswani *et al.*, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998-6008.

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805,* 2018.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770-778.

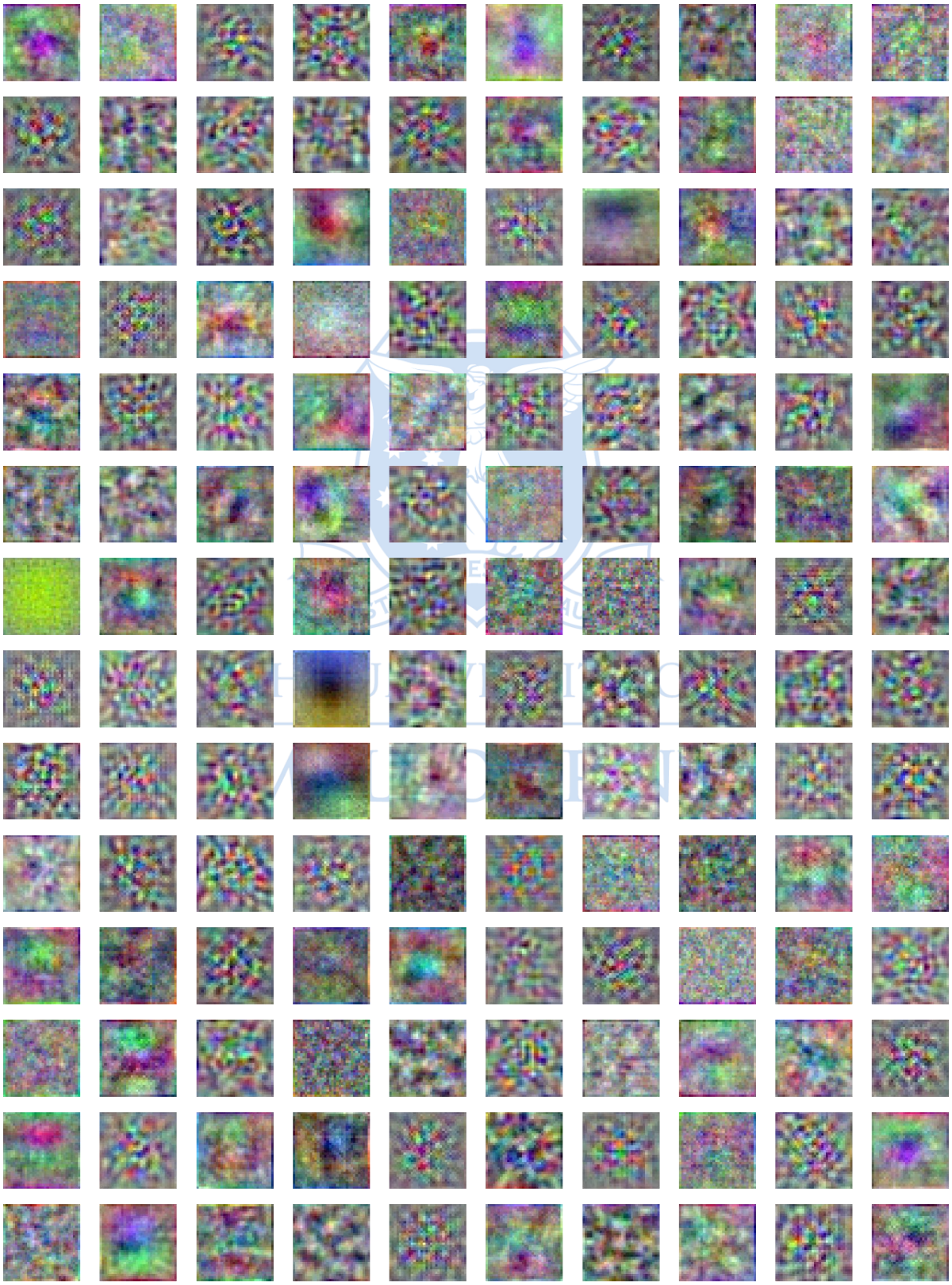
[21] M. Raghu, T. Unterthiner, S. Kornblith, C. Zhang, and A. Dosovitskiy, "Do Vision Transformers See Like Convolutional Neural Networks?," *arXiv preprint arXiv:2108.08810,* 2021.

[22] R. Wightman, "PyTorch Image Models," *GitHub repository,* 2019, doi: 10.5281/zenodo.4414861. GitHub.
[23] Y. Chen, Z. Yang, S. Ahn, D. Samaras, M. Hoai, and G. Zelinsky, "COCO-Search18 fixation dataset for predicting goal-directed attention control," *Scientific reports,* vol. 11, no. 1, pp. 1-11, 2021.
[24] Z. Yang *et al.*, "Predicting goal-directed human attention using inverse reinforcement learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 193-202.
[25] "BCEWithLogitsLoss — PyTorch 1.10.0 documentation." Pytorch.org. https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html (accessed.
[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, 2009: Ieee, pp. 248-255.

Appendices

Appendix A: Filters of patch embedding layer

This appendix lists 140 out of 768 filters in the patch embedding layer of the 9010_chair_e-6 model.



Appendix B: Future Work – introducing scene prior by CLS token

To further improve the model, we can append an additional CLS token at the front of the sequence of patches to serve as the holistic representation of the scene (see the figure below). Then, this token will be fed into the transformer to generate the scene prior vector. After that, the scene prior vector is fed into an independent project layer to generate the scene prior. Finally, the overall prediction will be the scene prior to multiplying with the probability map.

If the scene is completely unrelated to the target object, it will have a very low scene prior. Hence, even if the model still predicts high probabilities for some patches based on the low-resolution scene information, the final output will be very low. This can prevent the model make unreasonable predictions on unfamiliar scenes.