A VISUAL ENCODERS

In this section, we will describe the architectures of all end-to-end visual encoders, the image augmentations applied for end-to-end visual encoders, and detail the sources of the pre-trained encoders used in our study.

A.1 End-To-End Visual Encoders

Impala ResNet. The Impala ResNet architecture faithfully implements the visual encoder of the "large architecture" outlined by Espeholt et al. [8] consisting of a 3×3 convolution with stride 1, max pooling with 3×3 kernels and stride 2 followed by two residual blocks of two 3×3 convolutions with stride 1. This joint block is repeated three times with 16, 32, and 32 channels, respectively.

Custom ResNet. The architecture for our custom ResNet models is modelled after Liu et al. [20] and illustrated in detail in Figure 6.

ViT. Our ViT architectures are all based on the reference implementation at https://github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit.py. For all models, we use no dropout, and the following configurations are used across the considered ViT visual encoders:

Model name	Patch size	Num layers	Width	MLP dim	Num heads
ViT Tiny	16	12	192	768	3
Custom ViT	16	4	512	512	12

Table 6: Configurations of end-to-end ViT models.

The ViT Tiny architecture follows the suggested architecture of Steiner et al. [33]. In contrast, both custom ViT for 128×128 and 256×256 have notably fewer layers, wider dimensions of the attention layers and no increase of dimensions in the MLP projections. In our experiments, we found that such an architecture resulted in better online evaluation performance in CS:GO and Minecraft Dungeons.

Image augmentations. If image augmentations are applied during training, we randomly augment images after the down-scaling process. We implement all augmentations with the `torchvision` library and randomly sample augmentations during training. We apply the following augmentations as described by Baker et al. [1]:

- Change colour hue by a random factor between -0.2 and 0.2
- Change colour saturation with a random factor between 0.8 and 1.2
- Change brightness with a random factor between 0.8 and 1.2
- Change colour contrast with a random factor between 0.8 and 1.2
- Randomly rotate the image between -2 and 2 degrees
- Scale the image with a random factor between 0.98 and 1.02 in each dimension
- Apply a random shear to the image between -2 and 2 degrees
- Randomly translate the image between -2 and 2 pixels in both the x- and y-direction

A.2 Pre-Trained Visual Encoders

In this section, we will detail the sources for all pre-trained visual encoders considered in our evaluation.

OpenAI CLIP. For the visual encoders of OpenAI's CLIP models [27], we use the official interface at https://github.com/openai/CLIP. We use the following models from this repository: "RN50" (ResNet 50), "ViT-B/16", and "ViT-L/14". In preliminary experiments, we found the available larger ResNet models to provide no significant improvements in online evaluation performance and the ViT model with a larger patch size of 32 to perform worse than the chosen ViT models with patch sizes of 16 and 14.

DINOv2. For the DINOv2 pre-trained visual encoders [23], we use the official interface at https://github.com/facebookresearch/dinov2. Due to the computational cost, we do not evaluate the non-distilled ViT-G/14 checkpoint with 1.1 billion parameters.

FocalNet. For the FocalNet pre-trained visual encoders [40], we used the Hugging Face *timm* library (https://huggingface.co/docs/timm/index) to load the pre-trained models for its ease of use. We use the FocalNet models pre-trained on ImageNet-22K classification with 4 focal layers: "focalnet_large_fl4", "focalnet_xlarge_fl4", and "focalnet_huge_fl4".

Stable Diffusion. For the pre-trained stable diffusion 2.1 VAE encoder, we use the Hugging Face checkpoint of the model available at https://huggingface.co/stabilityai/sdxl-vae. This model can be accessed with the *diffusers* library. In contrast to other encoders, the VAE outputs a Gaussian distribution of embeddings rather than an individual embedding for a given image. We use the mode of the distribution of a given image as its embedding since (1) we want to keep the embeddings of the frozen encoder for a given image deterministic, and (2) we find the standard deviation to be neglectable for most inputs.

B DETAILS FOR VIDEO GAMES

Below, we provide more details for each video game we evaluate in, including details about the task, action space, dataset, and online evaluations.

B.1 Minecraft

Minecraft is a game that lets players create and explore a world made of breakable cubes. Players can gather resources, craft items and fight enemies in this open-world sandbox game. Minecraft is also a useful platform for AI research, where different learning algorithms can be tested and compared [15]. We use the MineRL [1, 11] environment, which connects Minecraft with Python and allows us to control the agents and the environment. We use MineRL version 1.0.2, which has been used for large-scale imitation learning experiments before [1], and which offers simpler mouse and keyboard input than previous MineRL versions [11].

Action space. In Minecraft, agents have two continuous actions corresponding to the x- and y-movement of the mouse to control the camera, and eight binary buttons. The buttons control the movement in four directions (forward, backward, rotate left, rotate right), interaction with items or objects, attacking (also used to destroy blocks needed to harvest wood), sprinting, and jumping.

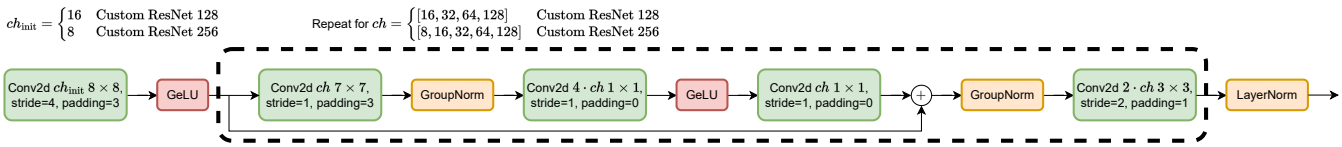


Figure 6: Illustration of the architecture of our custom ResNet visual encoders for 128×128 and 256×256 images.

Dataset. We use the Minecraft dataset released with the OpenAI VPT model [1] to select demonstrations of tree chopping. We choose the 6.13 version of the dataset and filter it to 40 minutes of human demonstrations that start from a fresh world and chop a tree within 1 minute. We also remove any erroneous files that remain after the filtering. The demonstrations include the image pixels seen by the human player at 640×360 resolution and the keyboard and mouse state at the same time, recorded at 20Hz. We also run the models at 20Hz.

Online evaluation. To evaluate our BC models, we use the "Treechop" task; after spawning to a new, randomly generated world, the player has to chop a single log of a tree within 1 minute. This is the first step to craft many of the items in Minecraft, and has been previously used to benchmark reinforcement learning algorithms [11]. See Figure 2b for a screenshot of the starting state. The agent observes the shown image pixels in first-person perspective, can move the player around and attack to chop trees. For reporting the performance of trained models, we rollout each model for 100 episodes with the same world seeds, and record the number of trees the player chopped. If the player chopped at least one tree within the first minute, the episode is counted as a success, otherwise it is counted as a failure (the timeout is set to 1 minute).

B.2 Counter-Strike: Global Offensive

CS:GO is a first-person shooter game designed for competitive, five versus five games. The core skill of the game is accurate aiming and handling the weapon recoil/sway as the weapon is fired. Previous work has used CS:GO as a benchmark to train and test behavioural cloning models [25], with best models able to outperform easier bots [26]. We incorporate experiments using CS:GO, as it offers visuals more similar to the real-world images that most pre-trained visual encoders were trained on, in contrast to our primary evaluation in Minecraft Dungeons and Minecraft (see Figure 2c).

Action space. We represent the mouse movement in x- and y-direction as two continuous values, and the left mouse click as a binary button.

Dataset. Following Pearce and Zhu [26], we use the "Clean aim train" dataset and setup. The controlled player is placed in the middle of an arena, and random enemies are spawned around them who try to reach the player. The player can not move; they can only aim and shoot (Figure 2c). The dataset contains 45 minutes of expert-human gameplay from one player, recorded at 16Hz.

Online evaluation. To evaluate models, we run each model for three rollouts of five minutes each in the "Clean aim train" environment at 16Hz, and report the average and standard deviation of the kills-per-minute.

B.3 Minecraft Dungeons

Minecraft Dungeons is an action-adventure role-playing video game with isometric camera view centered on the player. The player controls the movement and actions (including dodge roll, attack, use health potion, use items) of a single character which is kept in the center of the video frame (as seen in Figure 2a). The player has to complete diverse levels by following and completing several objectives. In our evaluation, we focus on the "Arch Haven" level of Minecraft Dungeons which contains fighting against several types of enemies and navigation across visually diverse terrain.

Action space. Agents have access to all effective controls in Minecraft Dungeons, including the x- and y-positions of both joysticks as four continuous values in the range $[-1, 1]$, the right trigger position (for shooting the bow), and ten buttons as binary actions. The most frequently used buttons during recordings control sword attacks, bow shooting, healing potions, and forward dodging.

Dataset. Before data collection, we pre-registered this study with our Institutional Review Board (IRB) who advised on the drafting of our participant instructions to ensure informed consent. After their approval, four players[4] played the "Arch Haven" level, and game frames at 1280×720 resolution, actions (joystick positions and button presses on a controller), and character position within the level were captured. The dataset includes a total of 139 recorded trajectories with more than eight hours of gameplay at 30Hz. Individual demonstrations vary between 160 and 380 seconds which corresponds to 4,800 and 11,400 recorded actions, respectively. We use 80% of the data for training and reserve 20% for validation. Each player was instructed to complete the level using a fixed character equipped with only the starting equipment of a sword and bow, and most players followed the immediate path towards level completion.

Online evaluation. To evaluate the quality of trained BC policies, we rollout the policy in the game with actions being queried at 10Hz (see Section E for details). These actions are then taken in the game using Xbox controller emulation software. Each rollout spawns the agent in the beginning of the "Arch Haven" level and queries actions until five minutes passed (3,000 actions) or the agent dies four times resulting in the level being failed. We run 20 rollouts per trained agent and report the progression throughout the level (Section D).

C TRAINING LOSS CURVES

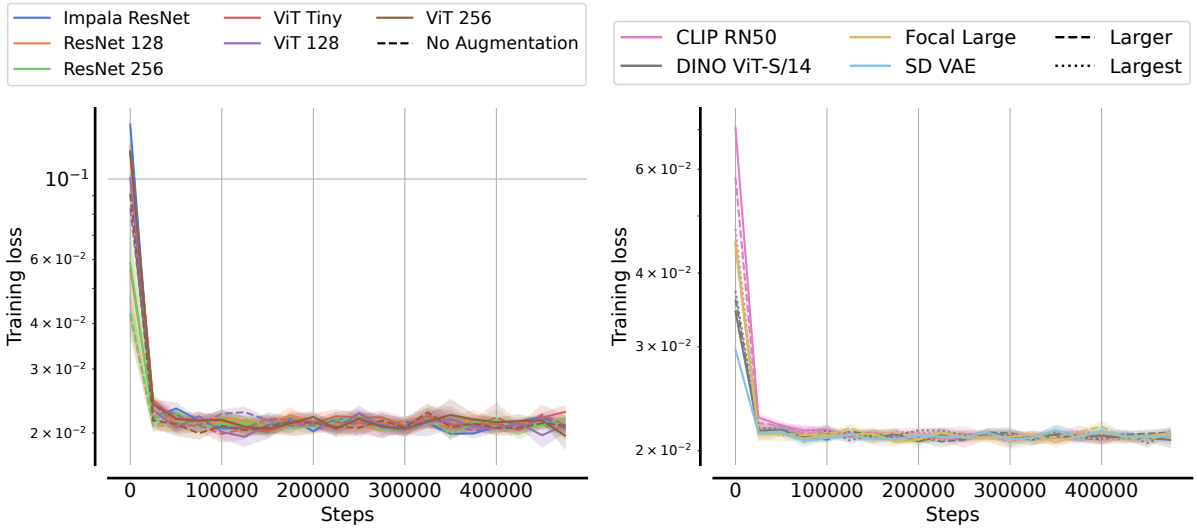
In this section, we learning curves for the training loss of all models across Minecraft Dungeons and Minecraft.

[4] 120 recordings were collected by one player with the remaining 19 recordings being roughly evenly split across the other three players.

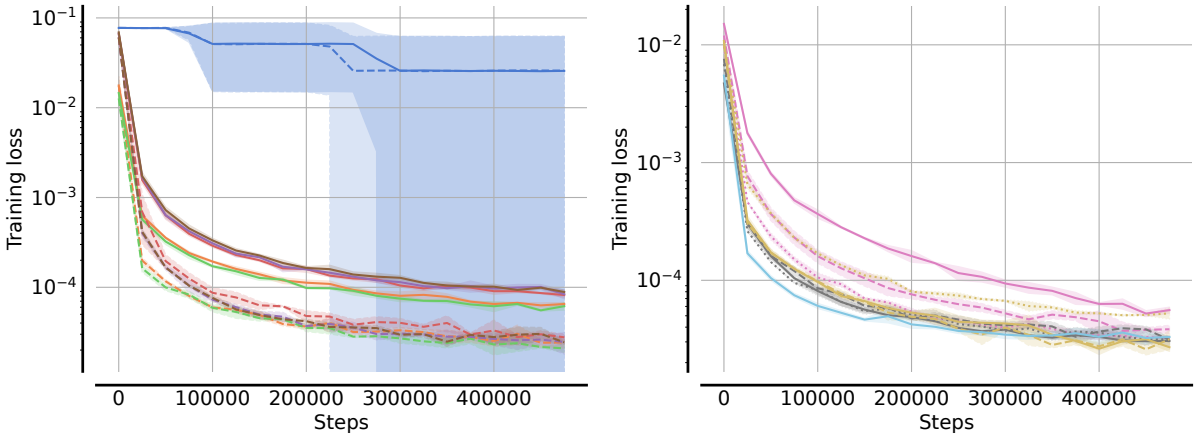
Minecraft. Figure 7a shows the training loss of end-to-end and pre-trained visual encoders in Minecraft. We find that the training loss for all end-to-end and pre-trained visual encoders in Minecraft plateaus after less than 50,000 training steps and all encoders appear to converge to a similar training loss. This might indicate that comparably short training might be sufficient given the small dataset in Minecraft, but we have observed that online rollout performance can increase even after training loss stagnates. Furthermore, this indicates that training loss is a poor indicator of online performance in Minecraft given we observed significant differences in online performance as seen in Table 3.

Counter-Strike: Global Offensive. Figure 7b shows the training loss for all models in CS:GO. In contrast to Minecraft, we can see that the training loss improves all throughout training for all trained models, indicating that the training task in CS:GO might be harder to learn compared to Minecraft. This is somewhat surprising given the comparable amount of demonstrations and lower dimensional action space of CS:GO compared to Minecraft. As expected, we can see that visual encoders trained end-to-end with image augmentations generally have a larger training loss despite improving online performance in most cases as seen in Table 3. Furthermore, we observe that the Impala ResNet models exhibit comparably high dispersion across three seeds, leading to large shading and stagnating training loss early in training. We hypothesise that this occurs due to the very large embeddings of the Impala encoders that make training a BC policy challenging. However, while the large dispersion across runs is also observed for the online rollout performance of Impala ResNet, in aggregate, we observe strong online performance of the Impala ResNet models in CS:GO (Table 3) despite the high training loss. Among pre-trained encoders, we find most models to converge to similar training loss values that are also comparable to end-to-end trained models.

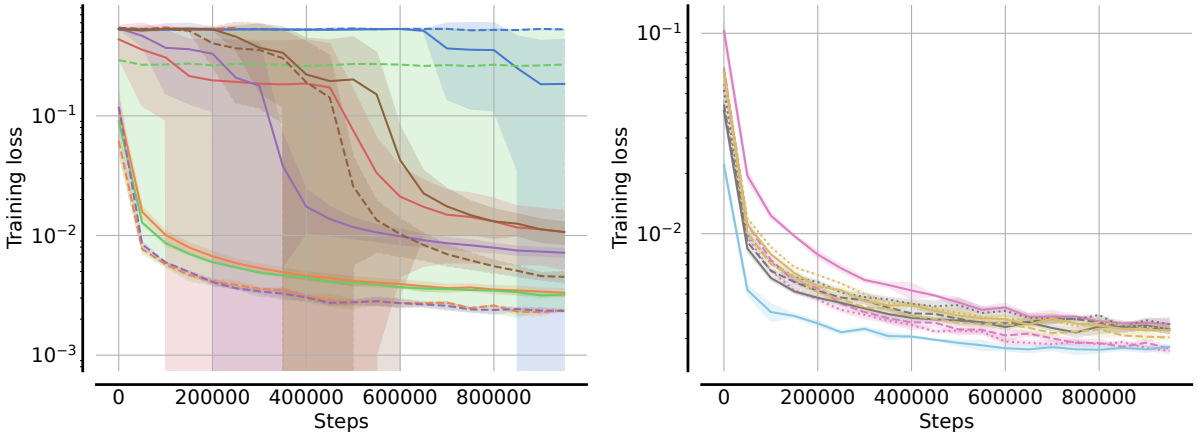
Minecraft Dungeons. Figure 7c shows the training loss for all models with end-to-end and pre-trained visual encoders in Minecraft Dungeons. Generally, we observe similar trends for Minecraft Dungeons as we observe for CS:GO with slowly converging training losses, image augmentations leading to higher training losses for end-to-end trained models, and the Impala ResNet models exhibiting high training loss that stagnates early in training and high dispersion across random seeds. Similarly, we observe that the training loss for the ResNet with 256×256 images trained without image augmentations stagnates early in training and exhibits high dispersion across seeds. Among the models with pre-trained visual encoders, the training loss appears comparable for most models. Only the reconstruction-based stable diffusion encoder and the CLIP ResNet50 models stand out with the lowest and highest training loss throughout training, respectively. Comparing the training loss of models with end-to-end and pre-trained visual encoders further shows that end-to-end encoders trained without image augmentation are capable of reaching lower losses. We hypothesise that this occurs since the end-to-end trained encoders are specialised to perform well on the exact training data the loss is computed over.



(a) Minecraft



(b) Counter-Strike: Global Offensive



(c) Minecraft Dungeons

Figure 7: Training loss in log-scale for BC agents in Minecraft (top), Counter-Strike: Global Offensive (middle), and Minecraft Dungeons (bottom) with end-to-end trained (left) and pre-trained (right) visual encoders. We visualise the mean and standard deviation across three seeds after computing window-averaged training losses at twenty regular intervals throughout training.

D MINECRAFT DUNGEONS ARCH HAVEN LEVEL

To measure progress for the online evaluation in Minecraft Dungeons, we define boundaries of zones which determine the progression throughout the "Arch Haven" level we evaluate in. These zones and a heatmap showing the visited locations of the human demonstrations used for training are visualised in Figure 8. The heatmap also shows the path followed by most demonstrations towards completion of the level.

E MINECRAFT DUNGEONS ACTION FREQUENCY IN ONLINE EVALUATION

The visual encoders used in our evaluation have vastly different model sizes (see Table 1) and, thus, notably different computational cost at inference time. This is particularly challenging during online evaluation in Minecraft Dungeons, since there exists no programmatic interface to pause or slow down the game like in Minecraft and CS:GO. We attempt to take actions during evaluation at 10Hz to match the action selection frequency of the (processed) training data, in particular due to the recurrent architecture of our policy. However, we are unable to perfectly match this frequency for all visual encoders on the hardware used to conduct the evaluation (see Section F for specifications on the hardware used during training and online evaluation) despite using a more powerful GPU for pre-trained visual encoders due to their comparably large size.

Table 7 lists the average action frequencies of all models during online evaluation in Minecraft Dungeons across all runs conducted as part of our evaluation. We note that most end-to-end trained visual encoders enable fast inference achieving close to 10 Hz action frequency. The ViT Tiny model is the slowest model, likely due to its deeper 12 layers in comparison to the other end-to-end trained ViT models with 4 layers as shown in Table 6, but we are still able to take actions at more than 8.5Hz. For pre-trained visual encoders, we see comparably fast action frequencies for all CLIP and most DINOv2 models as. The largest DINOv2 and stable diffusion VAE have notably slower action frequencies, but the FocalNet models induced the highest inference cost. However, we highlight that we did not observe behaviour during online evaluation which would suggest that these models were significantly inhibited due to this discrepancy.

Table 7: Average action frequencies during online evaluation in Minecraft Dungeons across 60 runs per model (20 for each seed).

Model name	Action freq. (Hz)
Impala ResNet	9.83
ResNet 128	9.90
ResNet 256	9.81
ViT Tiny	8.63
ViT 128	9.90
ViT 256	9.46
Impala ResNet +Aug	9.78
ResNet 128 +Aug	9.67
ResNet 256 +Aug	9.62
ViT Tiny +Aug	8.77
ViT 128 +Aug	9.69
ViT 256 +Aug	9.63
CLIP ResNet50	9.85
CLIP ViT-B/16	9.84
CLIP ViT-L/14	9.71
DINOv2 ViT-S/14	9.81
DINOv2 ViT-B/14	9.81
DINOv2 ViT-L/14	7.93
FocalNet Large	8.00
FocalNet XLarge	6.13
FocalNet Huge	6.91
Stable Diffusion VAE	8.77

F TRAINING AND EVALUATION HARDWARE

All training runs have been completed using Azure compute using a mix of Nvidia 16GB V100s, 32GB V100s and A6000 GPUs.

Minecraft Dungeons. For Minecraft Dungeons, end-to-end training runs for Impala ResNet, custom ResNets (for 128×128 and 256×256 images) and custom ViT for 128×128 images without image augmentation have been done on four 16GB V100s for each run. Training runs for the same models with image augmentation have been run on one A6000 GPU (with 48GB of VRAM) for each run. Training the ViT Tiny and ViT model for 256×256 images needed more VRAMs, so these were trained on eight 16GB V100s for each run.

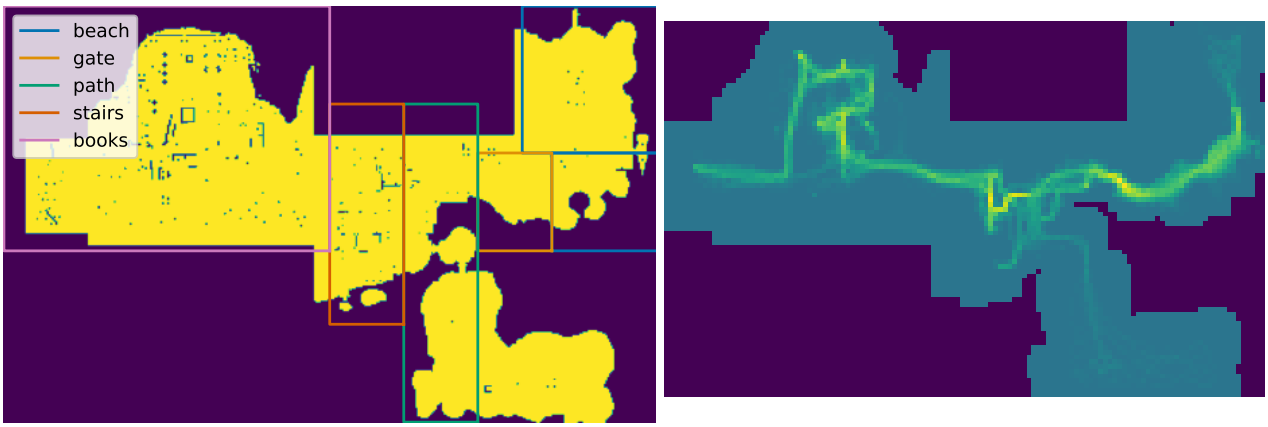
For training runs using pre-trained visual encoders, we computed the embeddings of all images in the Minecraft Dungeons dataset prior to training for more efficient training using A6000 GPUs. After, we were able to train each model using pre-trained visual encoders with four 16GB V100s for a single run.

To train models on half or a quarter of the training data for the third set of experiments, we used four 16GB V100s for a single run of any configuration.

Since the Minecraft Dungeons game is unable to run on Linux servers, we used Azure virtual machines running Windows 10 for the online evaluation. For evaluation of end-to-end trained models, we use a machine with two M60 GPUs, 24 CPU cores and 224GB of RAM. However, we noticed that this configuration was insufficient to evaluate models with larger pre-trained visual encoders at the desired 10Hz. Therefore, we used a configuration with one A10 GPU, 18 CPU cores and 220GB of RAM which was able to run the game and rollout the trained policy close to the desired 10Hz for all models.

Minecraft. The training hardware is similar to Minecraft Dungeons, with A6000s used for embedding/training with pretrained models, and 32GB V100s used to train the end-to-end models. Training pretrained models took considerably less time, with most models training within hours on a single A6000 GPU.

Minecraft evaluation was performed on remote Linux machines with A6000s, as MineRL is able to run on headless machines with virtual X buffers (`xvfb`). Each GPU had maximum of three rollouts



(a) Progression zones **(b) Human dataset heatmap**

Figure 8: (a) A visualisation of the boundaries of each progression zone in the "Arch Haven" level in Minecraft Dungeons used for online evaluations. (b) A heatmap visualising the visited locations of the human dataset of demonstrations within the "Arch Haven" level.

happening concurrently, with each rollout running at 3-9 frames per second, depending on the model size.

Counter-Strike: Global Offensive. Training was performed on the same hardware as with Minecraft experiments. For evaluation, we ran CS:GO on local Windows machines, equipped with either a GTX 1650Ti or a GTX 980 GPU, as per instructions in the original CS:GO paper [26]. We ran the game at lower speeds (and adjusted action rate accordingly) to allow models to predict actions in time to match the 16Hz action frequency.

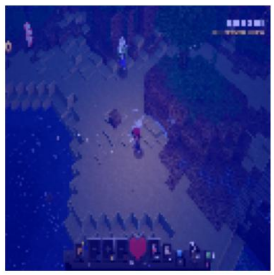
G GRAD-CAM VISUALISATIONS

To generate Grad-CAM [31] visualisations, we use the library available at https://github.com/jacobgil/pytorch-grad-cam. We use all actions of the policy trained on the embeddings of each visual encoder as the target concept to analyse, and visualise the average Grad-CAM plot across all actions. Following https://github.com/jacobgil/pytorch-grad-cam#chosing-the-target-layer, we use the activations of these layers within the visual encoders to compute visualisations for:

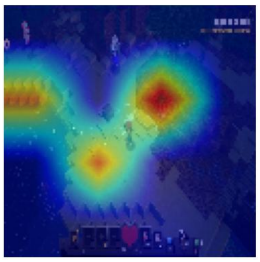
- ResNet: Activations across the last ResNet block
- ViT: Activations across the layer normalisation before the last attention block
- FocalNet: Activations across the layer normalisation before the last focal modulation block
- SD VAE: Activations across the last ResNet block within the mid-block of the encoder

(a) Original image

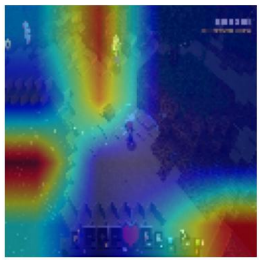
(b) Impala ResNet



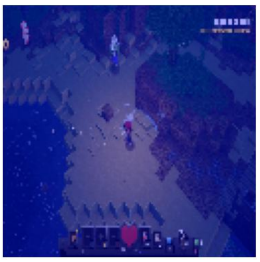
(c) Impala ResNet +Aug



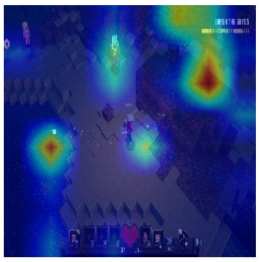
(d) ResNet 128



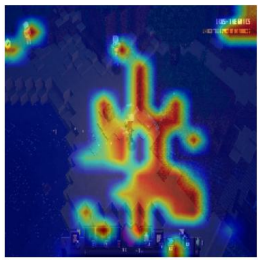
(e) ResNet 128 +Aug



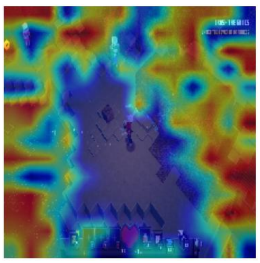
(f) ResNet 256



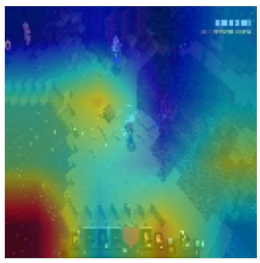
(g) ResNet 256 +Aug



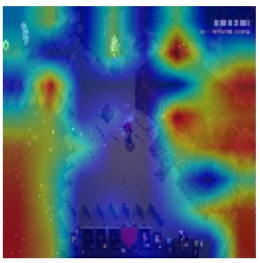
(h) ViT Tiny



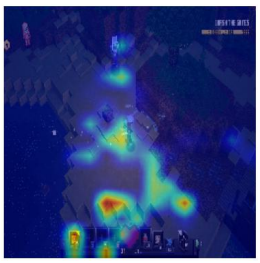
(i) ViT Tiny +Aug



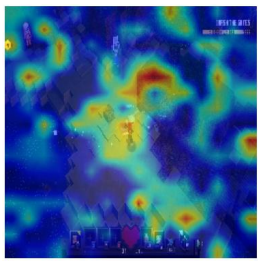
(j) ViT 128



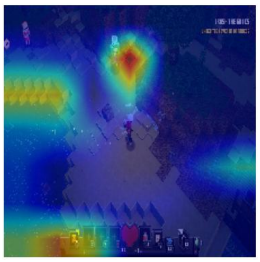
(k) ViT 128 +Aug



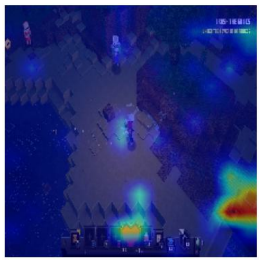
(l) ViT 256



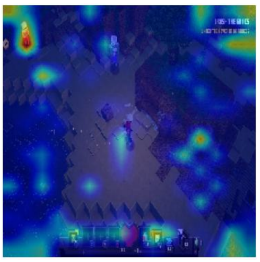
(m) ViT 256 +Aug



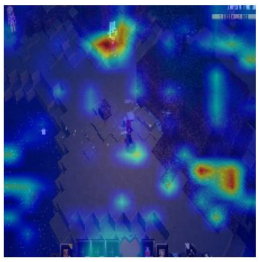
(n) CLIP RN50



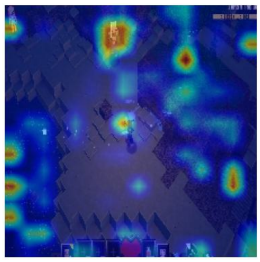
(o) CLIP ViT-B/16



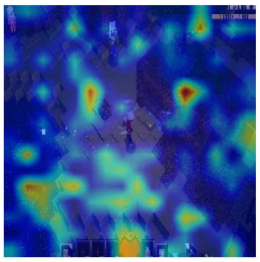
(p) CLIP ViT-L/14



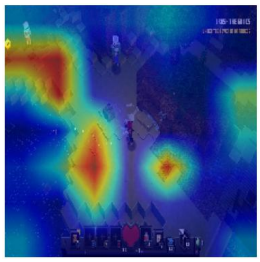
(q) DINOv2 ViT-S/14



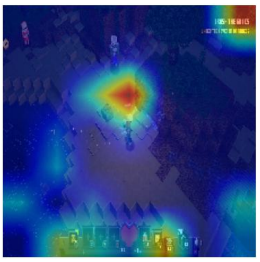
(r) DINOv2 ViT-B/14



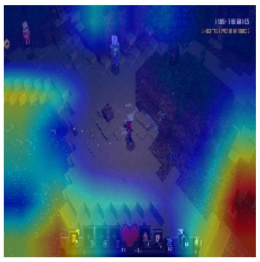
(s) DINOv2 ViT-L/14



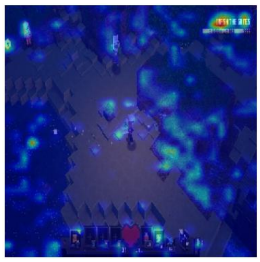
(t) Focal Large



(u) Focal XLarge



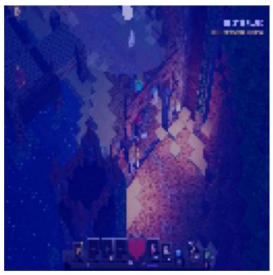
(v) Focal Huge



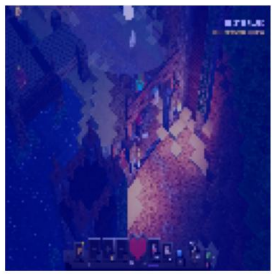
(w) SD VAE

Figure 9: Grad-Cam visualisations for all encoders (seed 0) in Minecraft Dungeons with policy action logits serving as the targets.

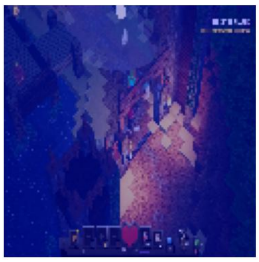
(a) Original image



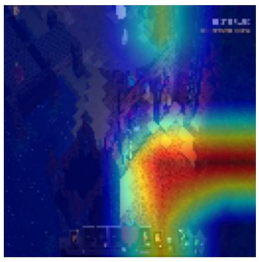
(b) Impala ResNet



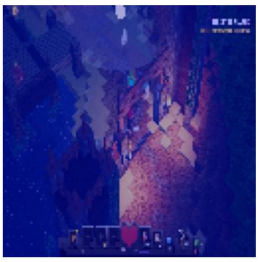
(c) Impala ResNet +Aug



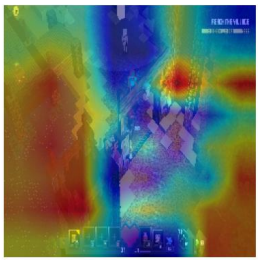
(d) ResNet 128



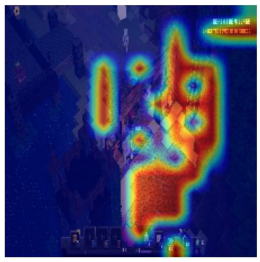
(e) ResNet 128 +Aug



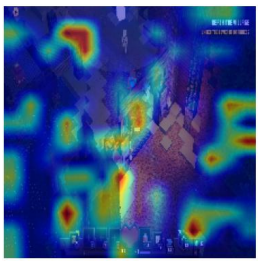
(f) ResNet 256



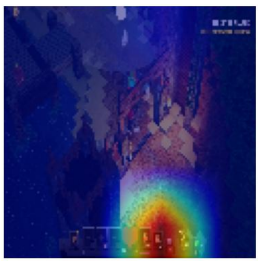
(g) ResNet 256 +Aug



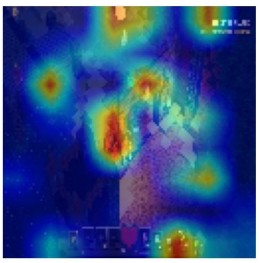
(h) ViT Tiny



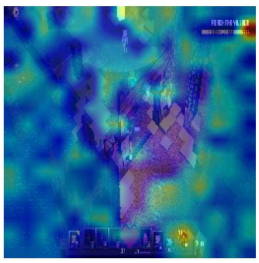
(i) ViT Tiny +Aug



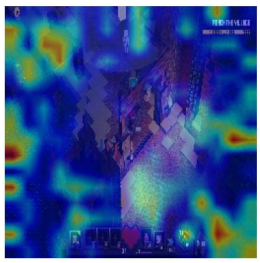
(j) ViT 128



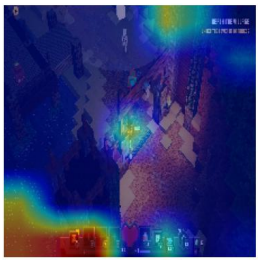
(k) ViT 128 +Aug



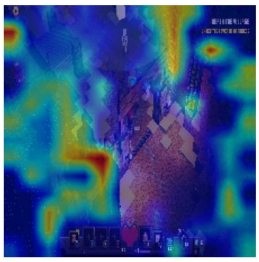
(l) ViT 256



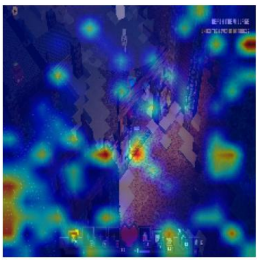
(m) ViT 256 +Aug



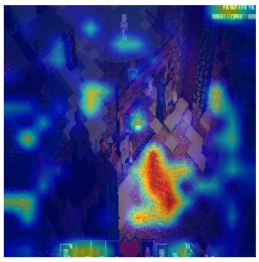
(n) CLIP RN50



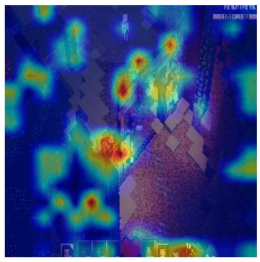
(o) CLIP ViT-B/16



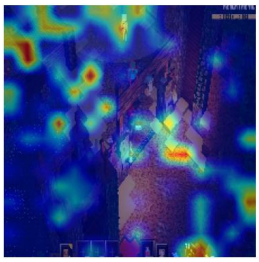
(p) CLIP ViT-L/14



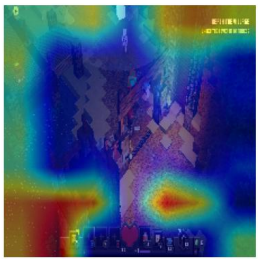
(q) DINOv2 ViT-S/14



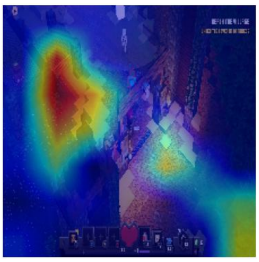
(r) DINOv2 ViT-B/14



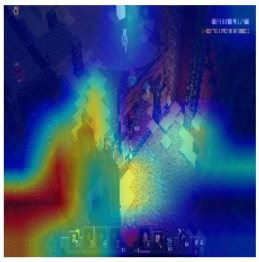
(s) DINOv2 ViT-L/14



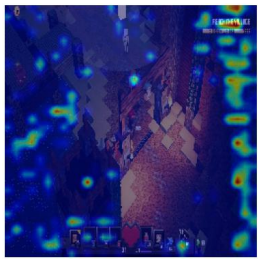
(t) Focal Large



(u) Focal XLarge

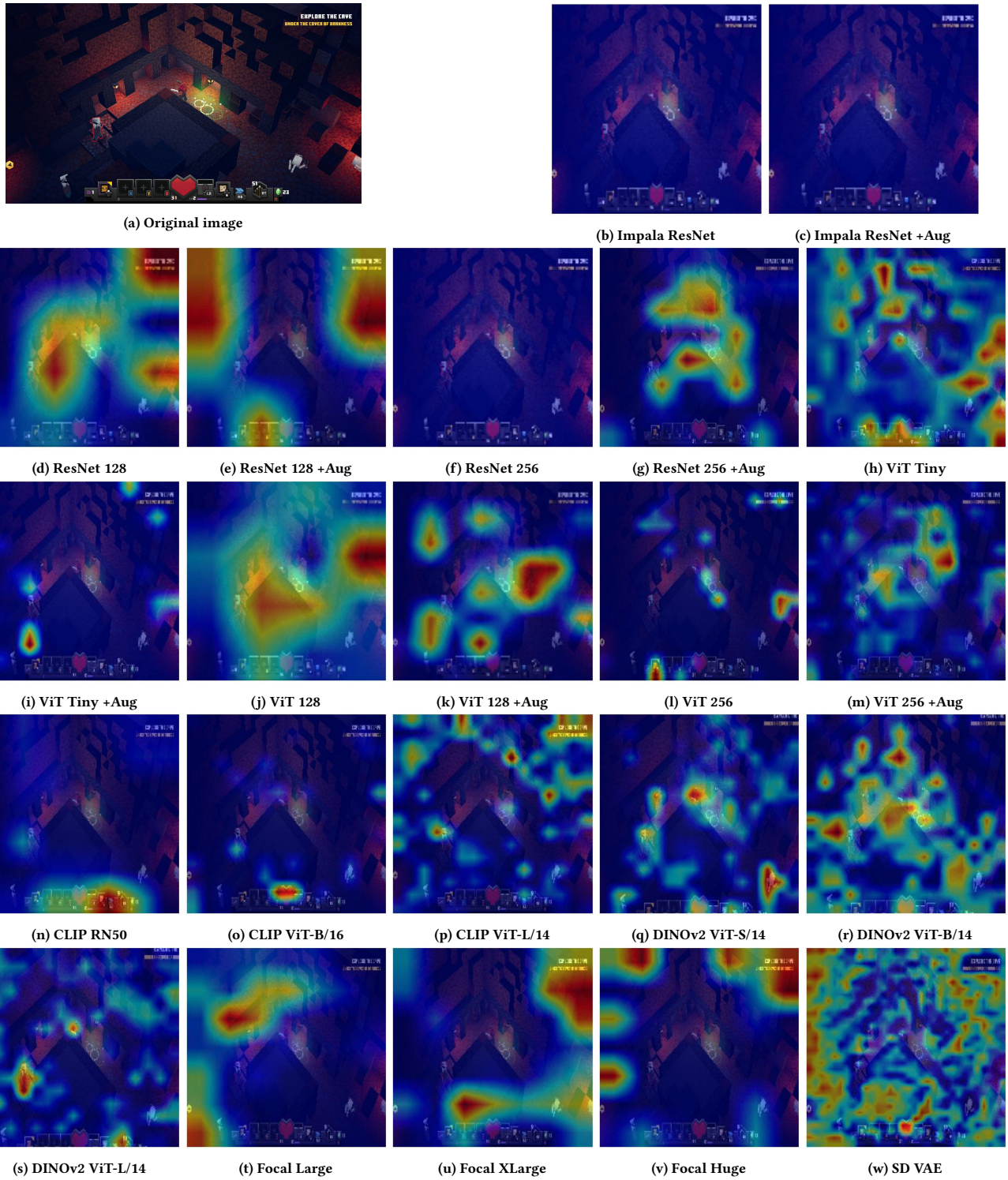


(v) Focal Huge



(w) SD VAE

Figure 10: Grad-Cam visualisations for all encoders (seed 0) in Minecraft Dungeons with policy action logits serving as the targets.



(a) Original image

(b) Impala ResNet

(c) Impala ResNet +Aug

(d) ResNet 128

(e) ResNet 128 +Aug

(f) ResNet 256

(g) ResNet 256 +Aug

(h) ViT Tiny

(i) ViT Tiny +Aug

(j) ViT 128

(k) ViT 128 +Aug

(l) ViT 256

(m) ViT 256 +Aug

(n) CLIP RN50

(o) CLIP ViT-B/16

(p) CLIP ViT-L/14

(q) DINOv2 ViT-S/14

(r) DINOv2 ViT-B/14

(s) DINOv2 ViT-L/14

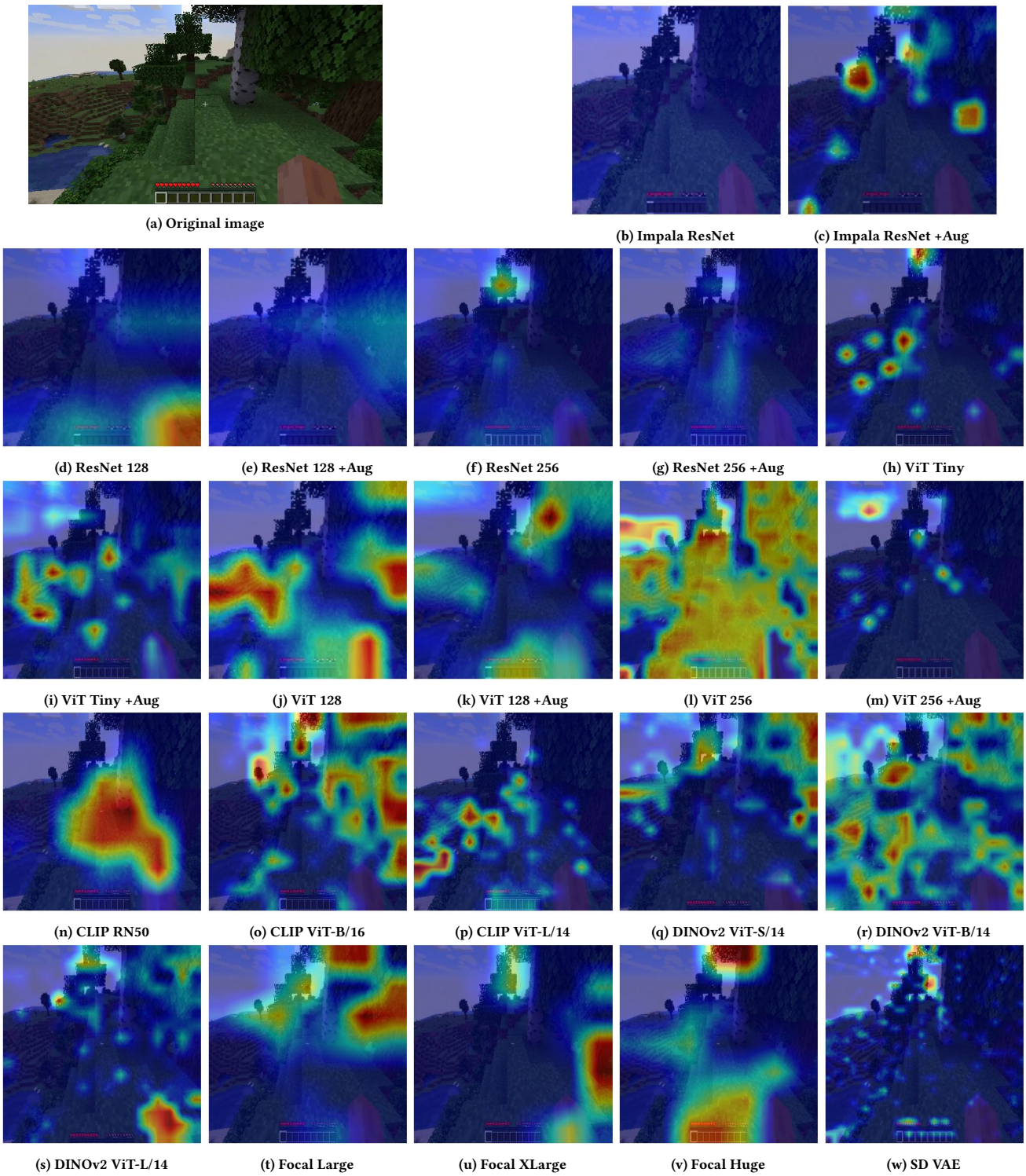
(t) Focal Large

(u) Focal XLarge

(v) Focal Huge

(w) SD VAE

Figure 11: Grad-Cam visualisations for all encoders (seed 0) in Minecraft Dungeons with policy action logits serving as the targets.



(a) Original image

(b) Impala ResNet

(c) Impala ResNet +Aug

(d) ResNet 128

(e) ResNet 128 +Aug

(f) ResNet 256

(g) ResNet 256 +Aug

(h) ViT Tiny

(i) ViT Tiny +Aug

(j) ViT 128

(k) ViT 128 +Aug

(l) ViT 256

(m) ViT 256 +Aug

(n) CLIP RN50

(o) CLIP ViT-B/16

(p) CLIP ViT-L/14

(q) DINOv2 ViT-S/14

(r) DINOv2 ViT-B/14

(s) DINOv2 ViT-L/14

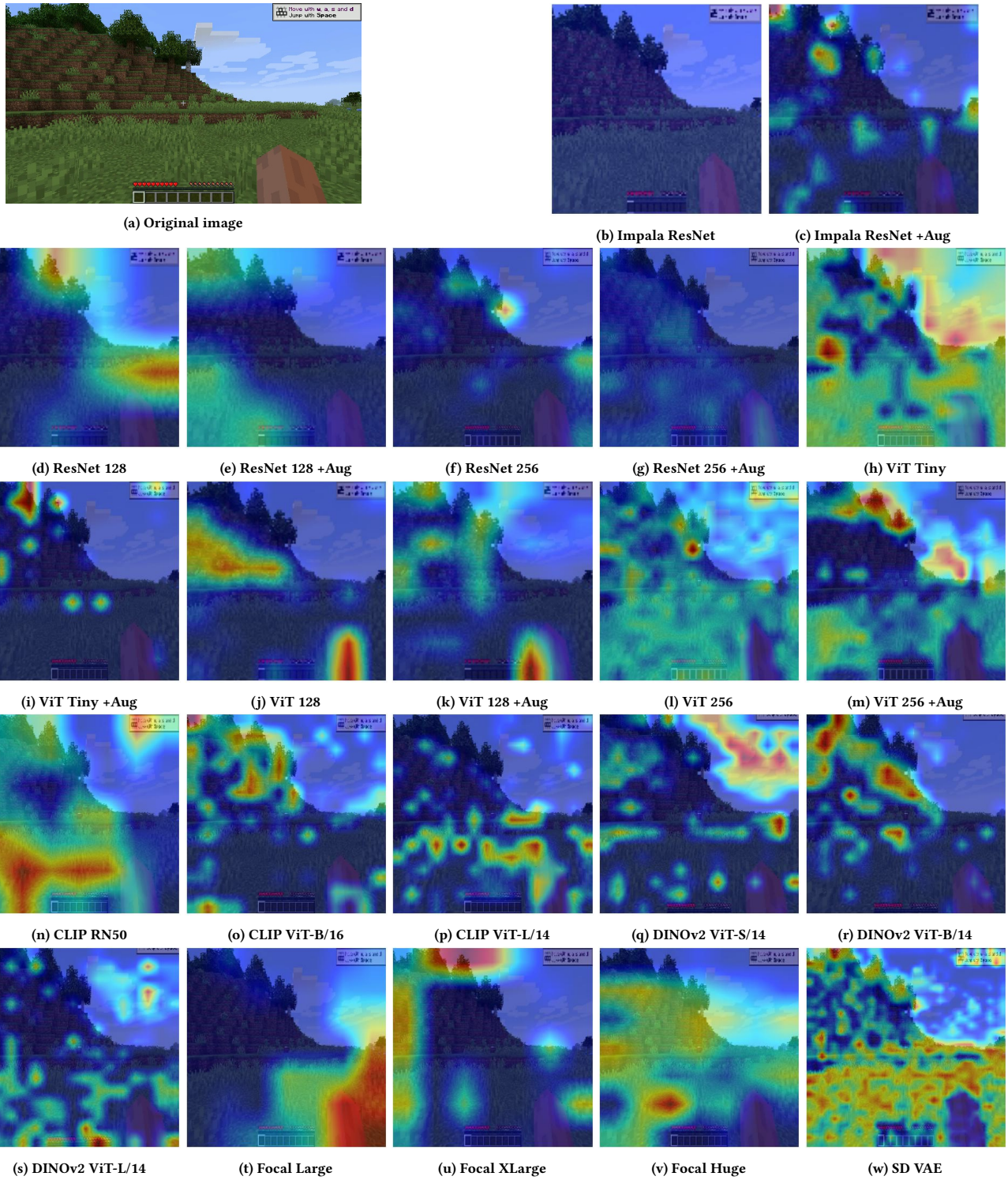
(t) Focal Large

(u) Focal XLarge

(v) Focal Huge

(w) SD VAE

Figure 12: Grad-Cam visualisations for all encoders (seed 0) in Minecraft with policy action logits serving as the targets.



(a) Original image

(b) Impala ResNet

(c) Impala ResNet +Aug

(d) ResNet 128

(e) ResNet 128 +Aug

(f) ResNet 256

(g) ResNet 256 +Aug

(h) ViT Tiny

(i) ViT Tiny +Aug

(j) ViT 128

(k) ViT 128 +Aug

(l) ViT 256

(m) ViT 256 +Aug

(n) CLIP RN50

(o) CLIP ViT-B/16

(p) CLIP ViT-L/14

(q) DINOv2 ViT-S/14

(r) DINOv2 ViT-B/14

(s) DINOv2 ViT-L/14

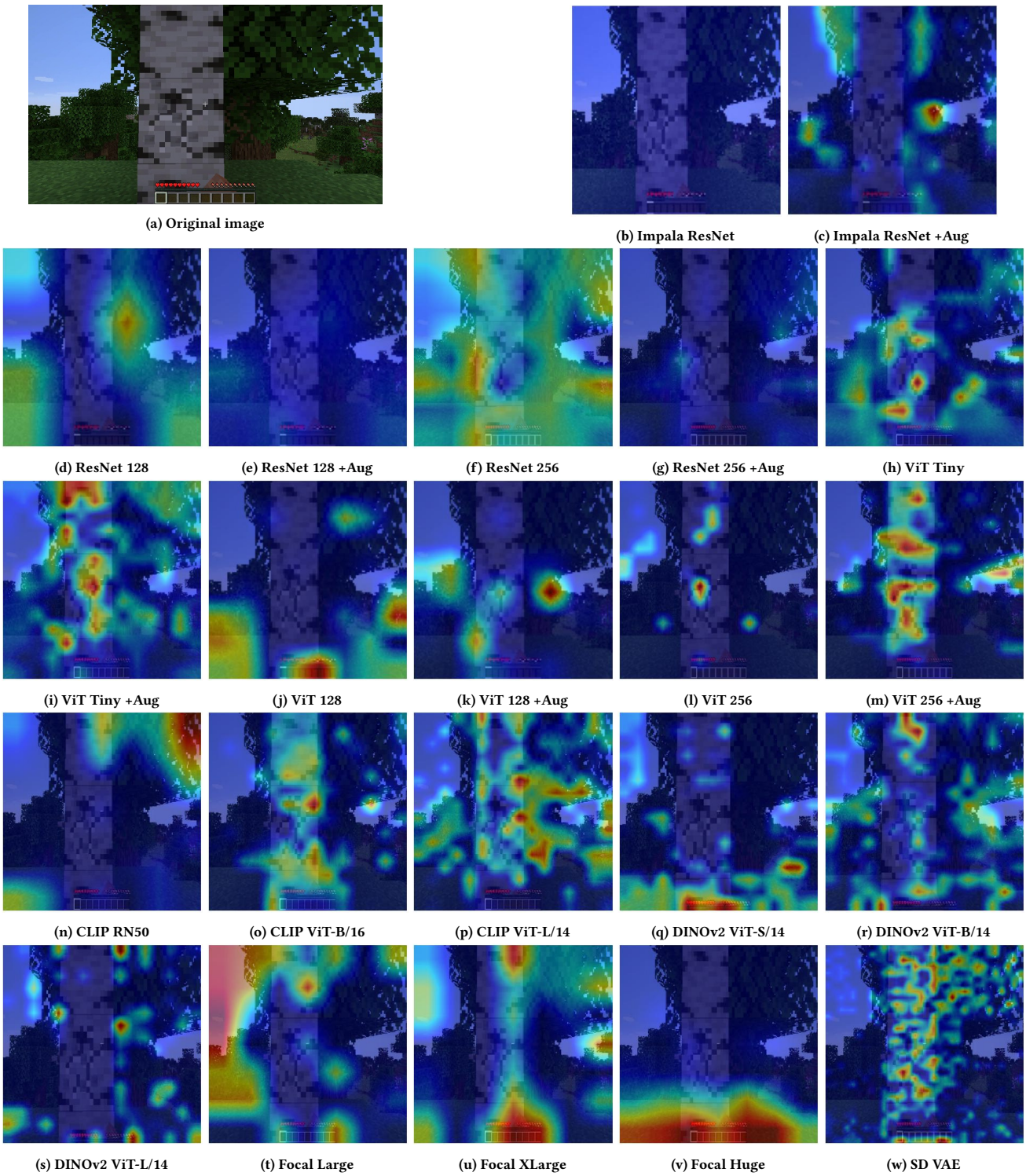
(t) Focal Large

(u) Focal XLarge

(v) Focal Huge

(w) SD VAE

Figure 13: Grad-Cam visualisations for all encoders (seed 0) in Minecraft with policy action logits serving as the targets.



(a) Original image

(b) Impala ResNet

(c) Impala ResNet +Aug

(d) ResNet 128

(e) ResNet 128 +Aug

(f) ResNet 256

(g) ResNet 256 +Aug

(h) ViT Tiny

(i) ViT Tiny +Aug

(j) ViT 128

(k) ViT 128 +Aug

(l) ViT 256

(m) ViT 256 +Aug

(n) CLIP RN50

(o) CLIP ViT-B/16

(p) CLIP ViT-L/14

(q) DINOv2 ViT-S/14

(r) DINOv2 ViT-B/14

(s) DINOv2 ViT-L/14

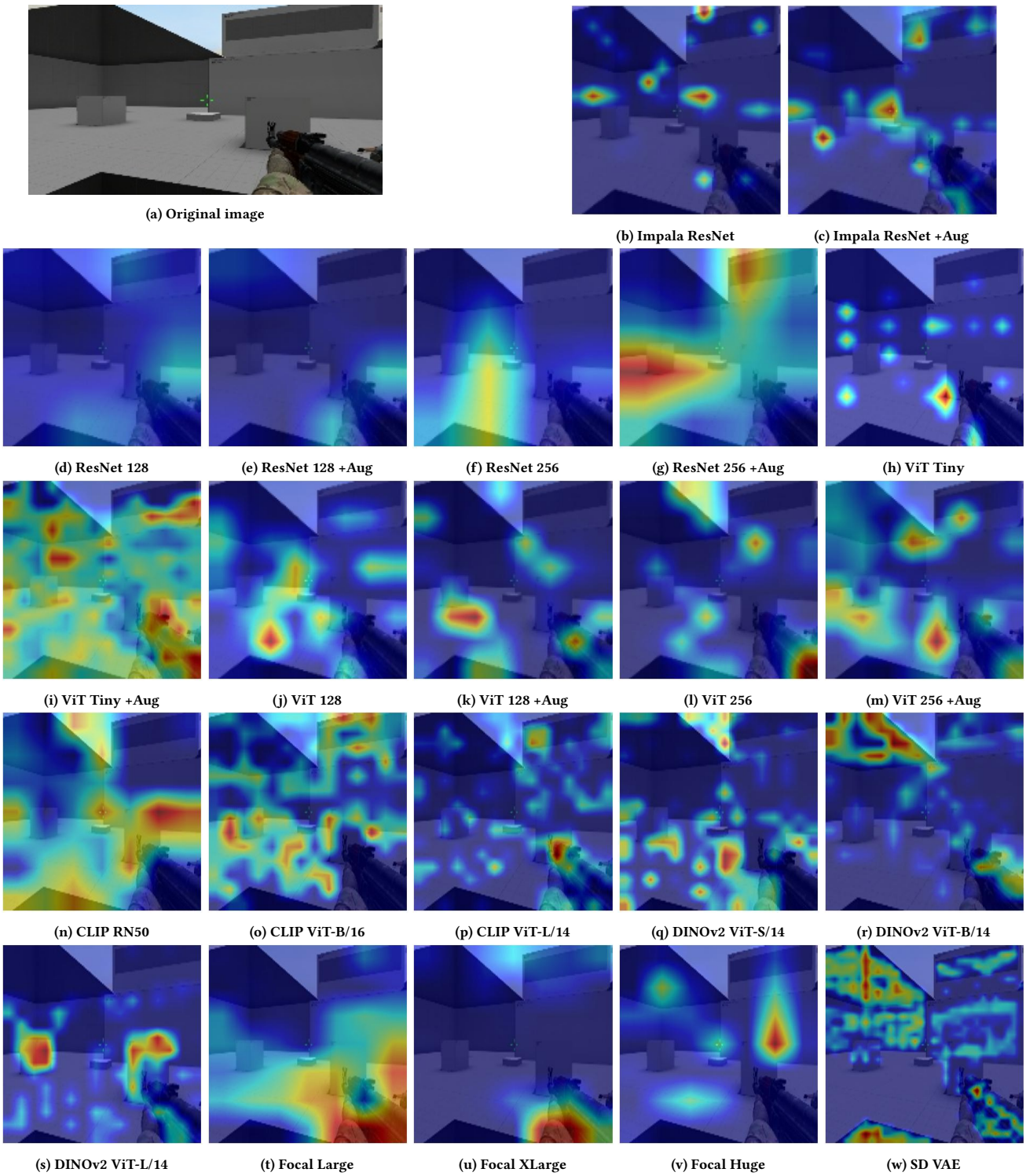
(t) Focal Large

(u) Focal XLarge

(v) Focal Huge

(w) SD VAE

Figure 14: Grad-Cam visualisations for all encoders (seed 0) in Minecraft with policy action logits serving as the targets.



(a) Original image

(b) Impala ResNet

(c) Impala ResNet +Aug

(d) ResNet 128

(e) ResNet 128 +Aug

(f) ResNet 256

(g) ResNet 256 +Aug

(h) ViT Tiny

(i) ViT Tiny +Aug

(j) ViT 128

(k) ViT 128 +Aug

(l) ViT 256

(m) ViT 256 +Aug

(n) CLIP RN50

(o) CLIP ViT-B/16

(p) CLIP ViT-L/14

(q) DINOv2 ViT-S/14

(r) DINOv2 ViT-B/14

(s) DINOv2 ViT-L/14

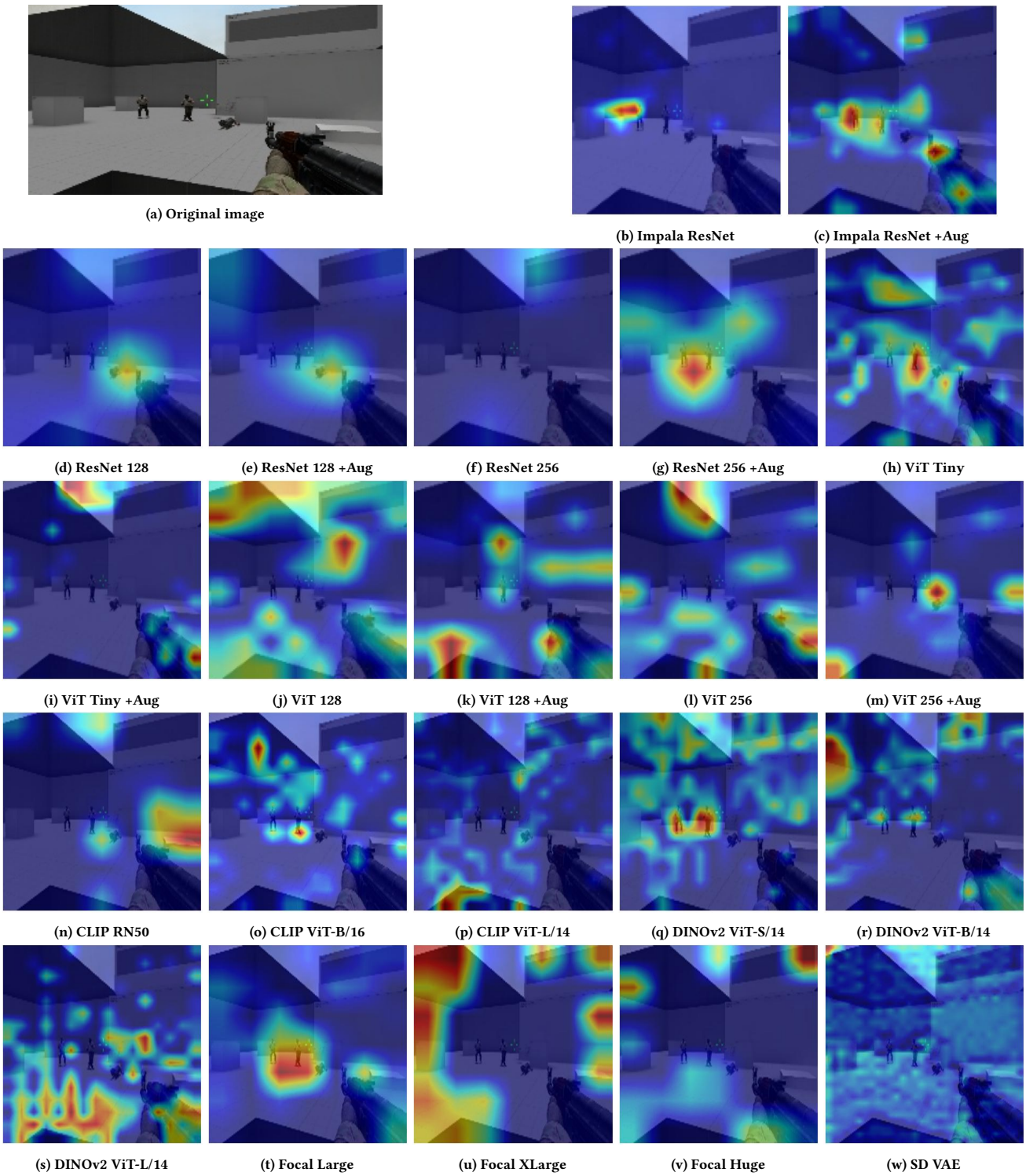
(t) Focal Large

(u) Focal XLarge

(v) Focal Huge

(w) SD VAE

Figure 15: Grad-Cam visualisations for all encoders (seed 0) in Counter Strike with policy action logits serving as the targets.



(a) Original image

(b) Impala ResNet

(c) Impala ResNet +Aug

(d) ResNet 128

(e) ResNet 128 +Aug

(f) ResNet 256

(g) ResNet 256 +Aug

(h) ViT Tiny

(i) ViT Tiny +Aug

(j) ViT 128

(k) ViT 128 +Aug

(l) ViT 256

(m) ViT 256 +Aug

(n) CLIP RN50

(o) CLIP ViT-B/16

(p) CLIP ViT-L/14

(q) DINOv2 ViT-S/14

(r) DINOv2 ViT-B/14

(s) DINOv2 ViT-L/14

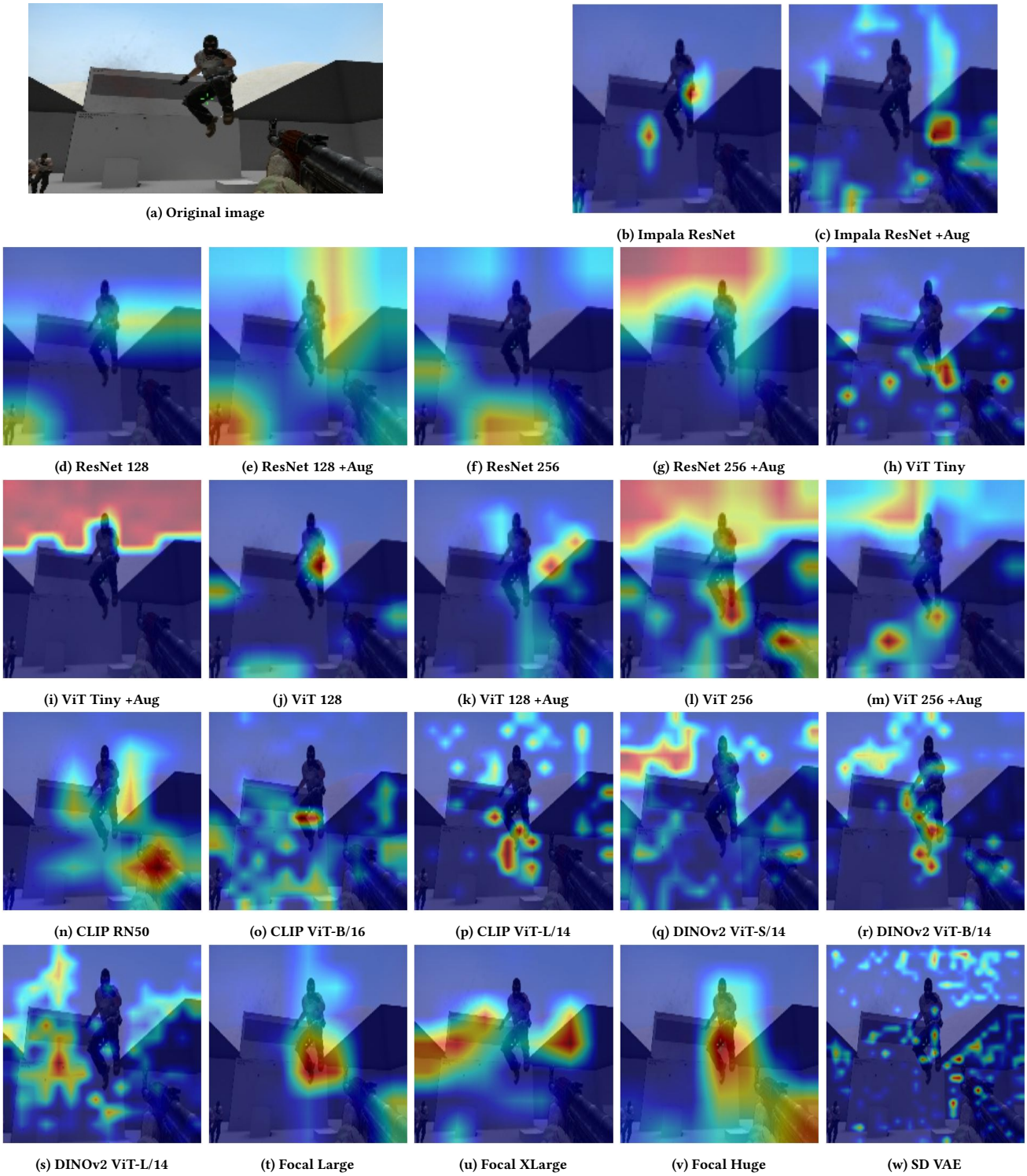
(t) Focal Large

(u) Focal XLarge

(v) Focal Huge

(w) SD VAE

Figure 16: Grad-Cam visualisations for all encoders (seed 0) in Counter Strike with policy action logits serving as the targets.



(a) Original image

(b) Impala ResNet

(c) Impala ResNet +Aug

(d) ResNet 128

(e) ResNet 128 +Aug

(f) ResNet 256

(g) ResNet 256 +Aug

(h) ViT Tiny

(i) ViT Tiny +Aug

(j) ViT 128

(k) ViT 128 +Aug

(l) ViT 256

(m) ViT 256 +Aug

(n) CLIP RN50

(o) CLIP ViT-B/16

(p) CLIP ViT-L/14

(q) DINOv2 ViT-S/14

(r) DINOv2 ViT-B/14

(s) DINOv2 ViT-L/14

(t) Focal Large

(u) Focal XLarge

(v) Focal Huge

(w) SD VAE

Figure 17: Grad-Cam visualisations for all encoders (seed 0) in Counter Strike with policy action logits serving as the targets.