Editing Neuron Hidden Activations for Specific Visual Properties in ViT

Junyang Sun
Symbolic Systems
sunnysun@stanford.edu

Bi Tian Yuan
Department of Physics
jackyuan@stanford.edu

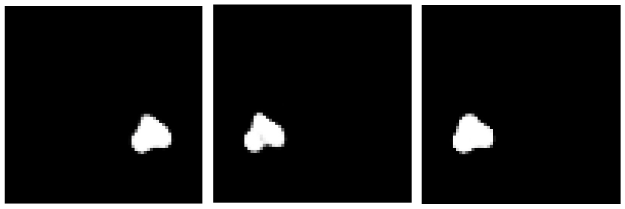

(a) Original (b) Target (c) Post Transform

Figure 1: An example of successfully changing the Location of the heart from bottom right to bottom left by manipulating latent space using approach 2.

Abstract

Neurons in Large-Language Models (LLMs) have been found to correspond to distinct high-level concepts, enabling researchers to map specific neurons to specific features. Inspired by these LLM interpretability research findings, we trained a Vision Transformer (ViT) to encode high-level vision properties, such as shape and x-location. We developed methods to directly edit the activations of the neural network to change desired properties, effectively "tricking" the neural network into producing a previously non-existent property. Our first approach utilizes non-linear models to map specific latent vectors to property labels and subsequently shifts the latent vectors in the desired direction. Our second approach employs a neural network to directly learn how to edit distinct neuron activations of the ViT, enabling the generation of new images with desired properties through the decoder. We demonstrate that properties such as location or shape can be modified successfully, though there is a risk of altering other properties. We hypothesize that this entanglement is partly due to the inherent characteristics of the transformer architecture and the loss function, which do not explicitly construct a manipulable latent space. Our findings highlight both the potential and the limitations of using ViT for property-specific image manipulation. These insights pave the way for future research aimed at disentangling complex features in latent spaces.

1. Introduction

The study of neural network activations and their correspondence to high-level concepts is a rapidly evolving field, with significant implications for understanding and improving model interpretability. Previous research by Anthropic has demonstrated the feasibility of constructing sparse dictionaries that map interpretable features to specific neurons [3], thereby facilitating a more intuitive understanding of the internal workings of neural networks.

Building on this foundation, our research aims to advance the understanding of Vision Transformers (ViTs) by developing a dictionary that details the necessary adjustments in neuron activations to alter a single property of an encoded image while maintaining all other properties constant. ViTs, known for their powerful representation capabilities, offer a unique opportunity to explore the manipulation of neuron activations due to their transformer-based architecture, which differs fundamentally from traditional convolutional neural networks [4].

Our work employs two novel approaches to learn a dictionary of modifications in neuron activations. The first approach combines both linear and non-linear models to establish a mapping between the latent vectors and the property labels [1]. By shifting the latent vectors in the favored direction, this method aims to achieve the desired modification in the image property. The use of linear models allows for straightforward interpretability, while non-linear models offer the flexibility to capture more complex relationships in the data.

The second approach utilizes a neural network to learn the latent vectors that would generate a new image with the desired property alteration. This method leverages the inherent ability of neural networks to capture complex patterns in the data, enabling precise adjustments in the latent space.

Through these methodologies, we have demonstrated that properties such as location or shape can be selectively modified. However, we also observed that some properties remain entangled, leading to unintended alterations. We hypothesize that this entanglement is partly due to the characteristics of the transformer architecture and the loss function, which do not explicitly construct a manipulable latent space.

Our findings highlight both the potential and limitations of using ViTs for property-specific image manipulation. By providing insights into the mechanisms of neuron activations within ViTs, our research contributes to the broader goal of enhancing model interpretability and control. This work paves the way for future research aimed at disentangling complex features in latent spaces, ultimately leading to more transparent and controllable neural network models.

1.1. Literature Review

1.1.1 Sparse Dictionary for LLM

In their groundbreaking study [3], ”Towards Monosemanticity: Decomposing Language Models With Dictionary Learning”, the research demonstrates a universal ”if and only if” relationship between specific neuron firings (features) and particular inputs, such as Arabic text or mathematical symbols. This relationship holds true across different models, indicating a form of universality in these feature-to-input mappings. The authors employ techniques like feature ablation and addition to further validate this relationship, showing that the presence or absence of certain features can add/subtract certain properties to the output of the model. We take inspiration from this model to build a dictionary that works not only on language features but on visual concepts.

1.1.2 Multimodal Neurons

In OpenAI’s study on multimodal neurons [5], they found that there are neurons in Contrastive Language-Image Pretraining (CLIP) responding to topics such as weather, seasons, actresses, etc. These have rich multimodality, in which a neuron would respond to a cartoon, text, or photo corresponding to the same concept. The CLIP model simultaneously embeds an image with ResNet and a text input with a Transformer language model using a contrastive loss. With these neurons corresponding to high-level concepts, this shows the natural result of aligning vision and text. Additionally, this paper shows the model vulnerable to ”typographic attack,” where adding adversarial text to images can cause them to be systematically misclassified. This paper showed that embeddings can be produced by ”word arithmetic” such as V(Img(”King”)) - V(Img(”Man”)) + V(Img(”Woman”)) = V(Img(”Queen”)). Many neurons are ”polysemantic,” meaning that they respond to multiple unrelated features. This was also confirmed in our previous research on VAE and disentanglement of features using dSprites, in which a single latent variable may be responsible for an entanglement of features that are not easily interpretable to humans.

1.1.3 Shifted Latent Vectors through Linear Regression

Belanec et al.’s work builds on current efforts to control outputs in generative models [1]. They introduce a novel approach by shifting latent features through building a linear relationship between latent vectors and property labels. In their research, a state-of-the-art Generative Adversarial Network (GAN), StyleGAN3, is used on the CelebA dataset [9], which contains a large number of celebrity faces, to alter facial features such as eyeglasses, black hair, and more.

Since the generator in a GAN samples from a latent space and generates images according to these latent variables, the authors trained a latent feature shifter that modifies the latent vectors in a direction learned by a linear regression model. They first trained a classifier on the generated images to distinguish whether specific binary labels of these facial features are present. Using those property labels, they identified a linear mapping between the latent vectors and the property labels. This step is crucial for understanding the relationship between the features and each property, thereby creating a mapping from the latent space to high-level, interpretable properties.

The shifted latent vector has demonstrated exceptional performance in feature modification. Inspired by their approach, we have adapted a similar methodology: training a classifier to identify the presence of a property and using a linear model to learn the corresponding mapping in the latent space.

1.1.4 Using LLM to explain another LLM

OpenAI conducted another study on LLMs [2], in which they defined an ”explanation score” that measures a language model’s ability to compress and reconstruct neuron activations using natural language. The procedures involve a subject model that we try to interpret, an explainer model that hypothesizes about the subject model’s behavior on neuron activations, and a simulator model that makes predictions based on the hypothesis to match the actual neuron activations. This paper found that GPT-4 or human explanations score poorly, and a typical neuron appeared polysemantic, confirming the result in the previous study. The study finds that there are neurons often explained poorly by the token-space explanations when compared with activation-based explanations. There are also pattern break neurons that is generally unable to be predicted by the explanation model because they’re context-sensitive. The inspiration we take from this paper is that, while at the task of interpreting neurons, we could use another similar model to explain and probe the hidden relationship between a feature and a neuron that may be otherwise difficult to see.

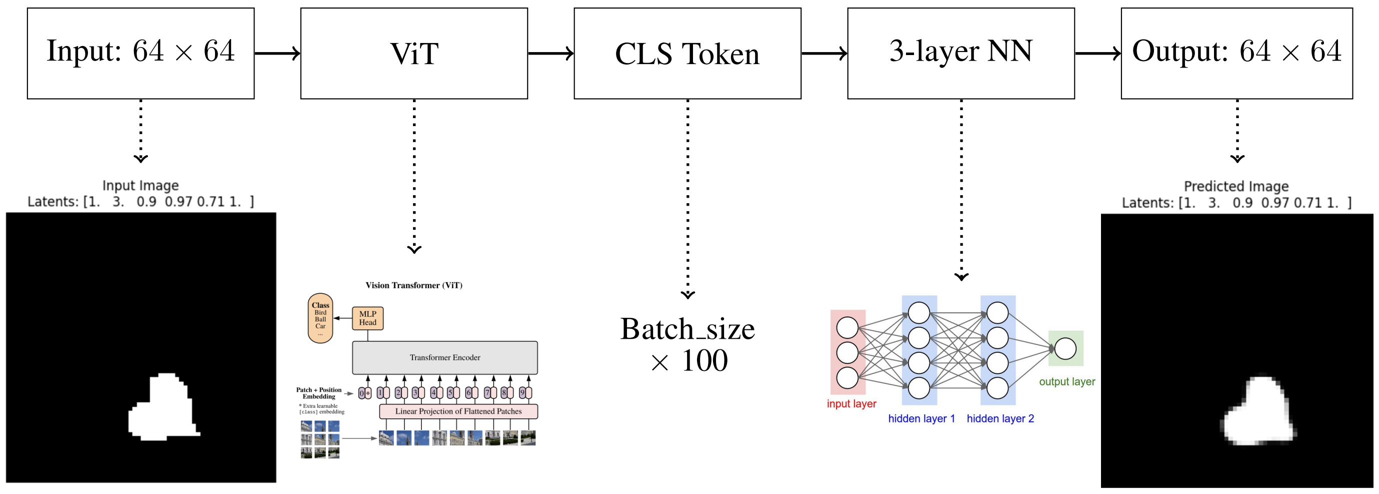


Figure 2: Model setup for reconstructing 64x64 d-sprite images using ViT and a 3-layer neural network.

2. Methods

2.1. Problem Setup

The first step is to create a Vision Transformer (ViT) (encoder) and decoder that is trained to reconstruct its input. For example, if the input is a square in the top left corner, the output should also be a square in the top left corner.

The key here is the creation of a small, lower-dimensional CLS token (embedding) between the ViT and the decoder. The CLS token acts as a throttle of information: the decoder only has access to this small embedding, but it must rely on it to reconstruct the original image.

After training a high-performing ViT encoder and decoder that reconstructs the image well enough, we freeze all the parameters and proceed to the second step.

The second step is to use different methods to create a dictionary. The dictionary takes as input the original property of the old image and the desired property information the user wants to add. It then returns a delta vector. When the old image is passed through the ViT encoder-decoder and the delta vector is applied to modify the CLS token output by the ViT, the modified CLS token is then passed to the original decoder. The decoder returns the image with only the specific property modified.

For example, let's say our objective is to modify the x-location of the object in the image. By transforming the neuron activations in the model in a high-dimensional space, we aim to learn the right transformation to "trick" the ViT into outputting a square in the top right corner, effectively changing only the x-location of the image.

This process is crucial due to the current lack of understanding of how ViTs represent visual concepts in their hidden layers. In both of our approaches, the CLS token (embeddings) are extracted from a ViT encoder-decoder model. This model processes the original image from the dataset through four transformer blocks to obtain the CLS token (embeddings) $Z \in \mathbb{R}^{100}$. These CLS token (embeddings) encapsulate compressed information that characterizes the image and are used to represent its features. The decoder, consisting of three linear layers, utilizes the latent vector Z to reconstruct the original image.

Our goal is to determine the specific ways to edit the CLS token (embeddings) so that our generated image possesses the desired properties.

2.2. ViT Model

The ViT model processes a 64×64 image with a depth of 4 and 6 heads [4]. We aimed to determine the minimum embedding size necessary for a 3-layer neural network (NN) to sufficiently reconstruct key properties of 64x64 d-sprite images, including location, rotation, size, and, most importantly, shape.

We set the embedding size to 10, and the 3-layer NN is able to capture everything else except for shape. On top of that, we experimented with various learning rates and even increased the neural network complexity from a 3-layer NN to a 6-layer NN. However, the decoder is still not able to capture the shape. This suggests that there is a limit to the amount of information an embedding size of 10 can convey to the decoder.

We found that an embedding size of 100 is the minimum to consistently and accurately recapture the shape property without significant learning difficulties.

We utilized binary cross-entropy loss due to the binary nature of the black-and-white images, allowing us to effectively penalize incorrect pixel predictions.

ViT Encoder-Decoder Model

- **Input:** Original image in 64×64.
- **Output:** Reconstructed image.
- **Function:** Encodes the image into a compressed hidden representation in $\mathbb{R}^{100}$ and decodes it back to reconstruct the image.

2.2.1 Contrastive Loss Function

For the ViT, we created a custom loss function that uses an alpha weighting of 0.9 for the Mean Squared Error (MSE) Loss and a β weighting of 0.1 for the Binary Cross-Entropy (BCE) Loss.

Furthermore, to maximize the distance between the embeddings of opposite classes so that our neural network encapsulate a better embedding space, we employed **contrastive loss** [8]. We define the two embedding vectors (each with shape `1 × 100`) that we want to push away from each other as much as possible. The d-sprite dataset gives each image a label as in `[color, shape, size, orientation, x_location, y_location]`. Each label is associated with one image, and one image is associated with one embedding. Therefore, we built a dictionary where the keys are the `1 × 6` property vector indicated above, and the value is its associated `1 × 100` embedding.

We chose to focus on maximizing the distances between embedding vectors that have different shapes since we noticed that: 1) during ViT reconstruction of the image, the shape was the most difficult feature to learn, and 2) in later attempts to shift the location of the image from one corner to the opposite corner, the shape often inadvertently changed as well, suggesting that shape information is quite entangled in the latent embedding space. As a result, we aimed to push the pairs of `[X, 1.0, X, X, X, X]` and `[X, 2.0, X, X, X, X]` as far from each other as possible using contrastive loss, and repeated this for shape pairs `[1, 3]`, `[2, 3]` during training as well. We experimented with various margins and settled on 5 to maximize the difference between these embeddings.

2.3. Approach 1: Linear & NonLinear Latent Shifter

Belanec et al.'s work builds on current efforts to control outputs in generative models [1]. They introduce a novel approach by shifting latent features through building a linear relationship between latent vectors and property labels. Inspired by this work, we employ a similar methodology using both linear and non-linear models to manipulate latent vectors extracted from a Vision Transformer (ViT) encoder-decoder model.

To shift the latent vectors, we first train a binary classifier on the reconstructed images to identify the presence of specific properties. This classifier provides the property labels Y. Using these labels, we establish a linear regression model to map the latent vectors Z to Y, learning the relationship between the features and the properties. The regression model provides coefficients β and an intercept ϵ that describe this mapping. After training the linear regression model, we shift the latent vectors Z by adding β in the direction of the desired property, scaled by a hyperpaprameter n.

The binary classifier used in our approach is designed to effectively extract and process features from input images. It consists of two convolutional layers with ReLU activation functions, each followed by a max pooling layer to reduce spatial dimensions. The output from the convolutional layers is flattened and passed through a fully connected layer with ReLU activation, followed by a final fully connected layer with a sigmoid activation function to produce a single output for binary classification.

The process involves the following steps:

1. **Latent Vector Extraction**: Extract latent vectors Z from the ViT encoder-decoder model for each image.
2. **Property Classification**: Train a binary classifier to predict property labels Y from the reconstructed images. In our case, we use only Hearts and Squares from the dataset and predict a binary label for the two.
3. **Linear Regression Training**: Train a linear regression model using Z and Y to learn the mapping and obtain coefficients β and intercept ϵ.
4. **Latent Vector Shifting**: Apply the learned mapping to shift the latent vectors in the direction of the desired property alteration to obtain Z_{new}.

$$Z_{new} = Z + \beta * n$$

We used both linear and non-linear approaches. The linear approach involves building a linear regression between the latent vectors Z and the property labels Y. We experimented with a linear regressor, a linear Support Vector Machine (SVM), a nonlinear SVM, and an ensemble model that takes in coefficients from both linear regression and the nonlinear SVM, weighted by a hyperparameter.

2.3.1 Linear Regression:

The linear regression model is defined as:

$$Y = Z\beta + \epsilon$$

where β represents the coefficients and ϵ is the intercept. The coefficients β are obtained by fitting the model using the training data (Z, Y) and minimizing the error ϵ.

2.3.2 Support Vector Machine (SVM):

For the linear SVM, the optimization problem is:

$$\min_{\beta,b} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n} \max(0, 1 - y_i(\beta^T z_i + b))$$

where β is the normal vector, b is the bias, and C is the regularization parameter. The SVM model provides the coefficients β which define the direction of the maximum margin hyperplane.

2.3.3 Non-linear SVM with RBF Kernel:

The non-linear SVM with Radial Basis Function (RBF) kernel uses the kernel trick to map input features into a higher-dimensional space where a linear separator is found. The decision function is:

$$f(z) = \sum_{i=1}^{n} \alpha_i y_i K(z_i, z) + b$$

where α_i are the dual coefficients, $K(z_i, z)$ is the RBF kernel, and b is the bias term. The direction β is computed from the support vectors and their corresponding dual coefficients.

2.3.4 Ensemble Model:

To leverage the strengths of both linear and non-linear approaches, we implemented an ensemble model. This model combines the coefficients obtained from linear regression and the non-linear SVM. The coefficients from both models are normalized and then averaged, weighted by a hyperparameter. This weighting balances the contributions from the linear regression and the SVM to optimize the performance.

2.4. Approach 2: A Second Predictor Neural Network

The other Predictor model that we build consists of linear layers. It takes in two inputs that represent two sets of features in a pair of images. This pair of images differs in exactly one property. The output of the Predictor is a delta vector that is applied to the current hidden activations of the first input to be able to change that one property in the output of the Embedding model.

2.4.1 Predictor Network

Our model setup includes an `EmbeddingPredictor` neural network with three fully connected (FC) layers for predicting embeddings to modify image properties. The first FC layer has an input dimension of `input_dim` and an output of 256, followed by Batch Normalization and LeakyReLU activation (slope 0.2). The second FC layer takes 256 as input and outputs 128, also followed by Batch Normalization and LeakyReLU (slope 0.2). The final FC layer has an input of 128 and an output of `output_dim`.

The model is trained using GPU acceleration when available, automatically selecting between CUDA and MPS based on availability. The training process is configured to log results using Weights and Biases (wandb) for comprehensive experiment tracking. Additionally, a specific seed value is set to ensure reproducibility of the results.

We use the second NN to analyze and predict the modifications required in the hidden activations of the ViT CLS token in order to successfully modify the desired property. A successfully trained second NN is able to apply these modifications, resulting in reconstructed images with the specified property altered while other properties remain unchanged.

2.4.2 Combined Loss Function

After experimenting with a variety of loss functions such as the Mean Square Loss and Cross Entropy Loss, we designed a custom combined loss function that helps the predictor learn the task. The loss function is composed of three main components: the delta embedding loss, the reconstruction loss, and the triplet loss.

Our custom loss includes three types of different loss.

First, define the delta embedding loss:

$$\mathcal{L}_{\text{delta}} = \|\Delta e_{\text{pred}} - \Delta e_{\text{true}}\|^2 \tag{1}$$

where:

- Δe_{pred}: The delta embedding predicted by the Predictor model.
- $\Delta e_{\text{true}} = e_{\text{target}} - e_{\text{original}}$: The true delta embedding.

Next, define the reconstruction loss:

$$\mathcal{L}_{\text{recon}} = \|g_{\text{recon}}(e_{\text{original}} + \Delta e_{\text{pred}}) - x_{\text{target}}\|^2 \tag{2}$$

where:

- g_{recon}: The frozen reconstruction network.
- e_{original}: The embedding of the original image.
- x_{target}: The target image.

There is also a third loss component, the triplet loss [6], which is conceptually similar to contrastive loss. The goal is to push the predicted embedding closer to the ground truth embedding while pushing it further away from the original embedding. In our context:

- The `anchor_batch` is the original pre-transform embedding subtracted by the predicted delta vector (i.e., the predicted embedding).
- The `positive_batch` is the ground truth embedding post-transformation.
- The `negative_batch` is the original pre-transform embedding.

where:

- $d(a_i, p_i) = \|a_i - p_i\|_2$ is the Euclidean distance between the i-th anchor and positive embeddings.
- $d(a_i, n_i) = \|a_i - n_i\|_2$ is the Euclidean distance between the i-th anchor and negative embeddings.

margin $= 0.5$

$$\mathcal{L}_{\text{triplet}} = \frac{1}{batch} \sum_{i=1}^{batch\_size} \max\left(0, d(a_i, p_i) - d(a_i, n_i) + \text{margin}\right)$$

The total loss is then given by:

$$\mathcal{L} = \alpha \cdot \mathcal{L}_{\text{delta}} + \beta \cdot \mathcal{L}_{\text{recon}} + \gamma \cdot \mathcal{L}_{\text{triplet}} \tag{3}$$

where: α, β, and γ are weighting coefficients that balance the three loss components.

3. Dataset

For our project, we utilized the dSprites dataset [10], which consists of 2D shapes generated from six independent latent factors: color(1), shape(3), scale(6), rotation(40), and x $(32)$and y(32) positions. This dataset is specifically designed for disentanglement studies, making it an ideal choice for our work as it allows for easy separation and manipulation of variables in the latent space. Each image in the dataset is created by varying one latent factor at a time, providing a clear and controlled environment for our experiments. See Figure 4 for reference.

In our project, we focused on the shape and location properties. Shape is a particularly intuitive visual feature for humans to recognize and understand, making it a straightforward choice for our experiments. By concentrating on these properties, we aim to effectively demonstrate the potential and limitations of our approaches in manipulating specific visual features within the latent space.

4. Results

4.1. ViT

We employed the ViT implementation in PyTorch[1]. The model was trained for 100 epochs, with a learning rate of

[1] https://github.com/lucidrains/vit-pytorch?tab=readme-ov-file

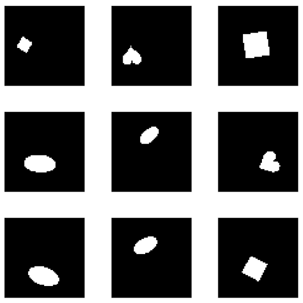


Figure 3: A selection of images in dsprites.

5×10^{-5}. We used a batch size of 64, lr_scheduler of 0.01, and the Adam optimizer. The training process involved monitoring the loss and accuracy over time to ensure proper convergence.

Figure 5 shows the training and validation loss curves over the 100 epochs. The training loss decreased steadily, indicating that the model was learning effectively to recontstruct the original input image. The validation loss also decreased, although with some fluctuations, which is typical in neural network training.

At the end of the training cycle, the final average training loss was 0.0697 and the final average validation loss was 0.0689. These values indicate that the model has generalized the reconstruction abilities to the validation set.

4.2. CNN Classifier

We trained both a binary classifier and a multiclass classifier for different purposes in our study. The binary classifier was designed to distinguish between hearts and squares, addressing the challenge in approach one where identifying the opposite label for squares was problematic due to the binary nature of the approach. This allowed us to focus on binary labels and experiment within that simplified context.

The CNN classifier was trained for 5 epochs with a learning rate of 1×10^{-3} using the Adam optimizer and a batch size of 64. The binary cross-entropy loss function was used to optimize the model for the binary classifier, and categorical cross-entropy was used for the multiclass classifier.

During training, we monitored the loss and accuracy to ensure proper convergence. Figures 6 and 7 show the training loss curves over the 5 epochs for the binary and multiclass classifiers, respectively. The training loss steadily decreased in both cases, indicating that the models were effectively learning the features necessary for classification.

These results indicate that both CNN classifiers successfully generalized to the validation set and were effective in distinguishing between the different shapes in the dataset.

As seen in 6 and 7, the higher training loss observed

Model	Training Accuracy (%)	Evaluation Accuracy (%)
Binary Classifier	99.97	99.96
Multiclass Classifier	99.15	99.14

Table 1: Training and Evaluation Accuracy for both CNN Classifier Models

for the multiclass classifier compared to the binary classifier is expected due to the nature of the loss functions used. The binary classifier utilizes binary cross-entropy (BCE) loss, which is specifically designed for binary classification tasks and tends to converge more quickly and efficiently when distinguishing between two classes. In contrast, the multiclass classifier employs categorical cross-entropy loss, which is more complex as it must handle multiple classes simultaneously. This increased complexity inherently leads to higher training loss values, as the model must learn to differentiate between a greater number of classes, resulting in a more challenging optimization process.

4.3. Approach 1 - Linear Regression & SVM

We experimented with various hyperparameters for the linear regression and SVM models to optimize the accuracy of property modifications. The training procedures and results for each model are detailed below.

4.3.1 Hyperparameter Tuning

Hyperparameter tuning was critical for optimizing the performance of the SVM models. We used grid search to experiment with different values of the regularization parameter C and the scalar factor n that determines the amount of change applied to the latent vectors. For instance, each of the models required a different set of hyperparameters and performed differently on the modification. The SVM with an RBF kernel was trained with $C = 10$, and the scalar factor for shifting the latent vectors was set to 16.

4.3.2 Modification Accuracy

The accuracies of property modifications for the different models are summarized in Table 2. We observed that the non-linear SVM model achieved the highest accuracy among the SVM variants. The linear regression model also performed well but was slightly less accurate compared to the non-linear SVM.

The modification accuracies for different scalar values were logged, and the visualizations of original and shifted images were generated to evaluate the effectiveness of the property modifications. These visualizations confirmed that the modifications were generally successful, with the intended properties altered while other properties remained unchanged.

4.3.3 Modification Success

From visualizing the images before and after modification, we see that the shapes are being distorted as we increase the scaling factor for the amount of shift applied onto the original latent vectors. However, the changes in shape are not really systematic and are rather a distortion on top of the original shape, regardless of the direction of the change (i.e. from a heart to a square or from a square to a heart).

4.4. Approach 2 - Neural Network Predictor

We decided to focus on modulating the specific property of x_location from $x = 0$ to $x = 31$, and vice versa. There are a total of $3(shapes) * 6(sizes) * 40(rotation) * 32(Y - values) = 23040$ training pairs.

The training curve can be found in Figure 8. The results were quite intriguing: In the following example, we correctly shifted the large square in the bottom right corner to the bottom left corner, while preserving correctly its size, rotation, and y-location. The only thing that got distorted was the shape, which we have earlier mentioned was the most complex property of them all.

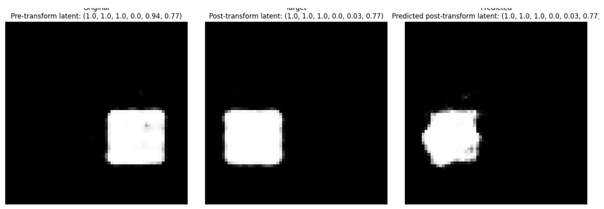

Pre-transform latent: (1.0, 1.0, 1.0, 0.0, 0.94, 0.77)
Post-transform latent: (1.0, 1.0, 1.0, 0.0, 0.03, 0.77)
Predicted post-transform latent: (1.0, 1.0, 1.0, 0.0, 0.03, 0.77)

Figure 4: Example of "tricking" the decoder to change its x-location property while keeping the other properties constant

However, most of the times we failed to preserve the shapes while transforming the x-location. More examples can be found in Figures 13 14 15 16.

5. Discussion

5.1. Shifting Latent Vectors through SVM

Quantitatively, the SVM approach demonstrated higher accuracy than other models in approach 1 for modifying the target properties, as evidenced by classifier-based evaluation. However, it is equally important to consider the qualitative results. Upon visual inspection, we observed that the shifted images often exhibited noticeable changes corresponding to the target properties. For instance, when shifting towards the 'square' label, the resulting images displayed more angular features, even if they were not perfect squares to the human eye (see Figures 9 and 12). Similarly, when shifting towards the 'heart' label, the resulting images often had an inward opening, suggesting the presence of the characteristic crevice of a heart shape (see Figure 11).

5.2. Entangled Shapes, Problems, and Solutions

Both methods often produced shapes without clear boundaries and were accompanied by floating white pixels nearby. These qualitative observations highlight the entanglement of properties, indicating that we are not yet able to completely isolate and modify a single property effectively. The first SVM-based approach not only failed to learn the shape accurately but also ”split” the shape into many smaller pieces, creating unnatural images. Despite better successes in the second approach of using a neural network to change the x-location, modifications often introduced unintended artifacts or failed to achieve a clear representation of the target shape. Nonetheless, it was slightly better than the first SVM-based approach.

In the future, this entangled shaped issue can be resolved by introducing a fourth loss created by a classifier that predicts whether the generated shape has a correct and natrual shape to encourage the second NN model to learn embeddings that produce more natural shapes.

Furthermore, both methods suffered from the problem of ”vanishing shapes.” Generated shapes became extremely tiny and eventually reduced to just a few dots while maintaining the correct location.

We have addressed this problem in the second method by introducing a second reconstructive loss, $\mathcal{L}_{\text{recon}}$, which completely eliminated this problem.

In extreme cases where the scaling factor was set to a high value, the images resulted in very distorted shapes. Since the classifier is not as robust as a pre-trained one, it tended to classify these distorted shapes almost exclusively as hearts. This underscores the importance of balancing quantitative metrics with qualitative assessments to gain a comprehensive understanding of the model's performance and limitations.

5.3. Explicit Latent Space through SVM

Inspired by [1]'s success in manipulating latent spaces and facial features, we trained a classifier and applied linear regression and other nonlinear models to the latent vectors. However, we could not achieve the same level of manipulation. We hypothesize this is due to their use of StyleGAN, which explicitly constructs a disentangled latent space for sampling. In contrast, our ViT does not build such an explicit latent space, making it difficult to establish a linear relationship between the latent space and visual feature labels.

5.4. Evaluating Modification for Shapes

To measure the success of our modifications, we employed a multiclass classifier trained to identify the three different shapes in the dSprites dataset for both methods. This classifier was used to evaluate the reconstructed images, with the target label set to the desired shape. The success of the modification was determined by the classifier's ability to correctly identify the target shape. While the reconstructed images from the decoder may not perfectly resemble a square or a heart to human eyes, successfully fooling the classifier serves as an important proxy for measuring the effectiveness of the modifications. It worked with some success, with the results presented in 2.

However, we recognize limitations in this approach. Despite the classifier being properly trained and evaluated on the validation set, it is possible that while our methods have changed the target shape to our desired shape, the target shape would not appear ”natural” to a human observer.

Model	**Train (%)**	**Validation (%)**
Linear Regression ($n = 8$)	34.19	34.20
Linear SVM ($n = 12$)	38.39	38.8
Non-linear SVM ($n = 6$)	49.38	49.28
Ensemble ($n = 6$)	48.11	47.76
Neural Network	**50.08**	49.11

Table 2: Modification Success of shape by different models. n stands for the scaling factor when the shift is applied.

6. Future Works

6.1. GAN Approach

To disentangle shapes in the embedding `cls_token`, we will use a GAN-like approach. If the shape produced by editing the latent `cls_token` does not match the intended shape, we penalize the second NN model. This is achieved by using a multi-class classifier to classify the images generated using the predicted embeddings, and penalize the wrong classification of shapes to encourage the NN model to create embeddings that generate explicit and correct shapes.

6.2. Larger ViT Model

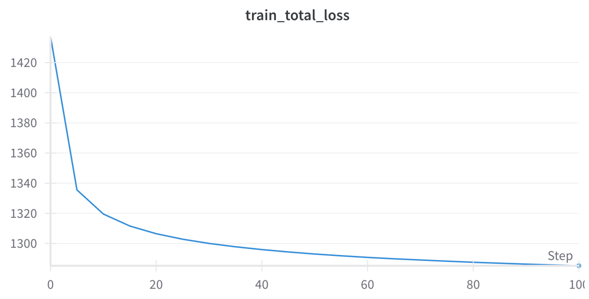
We limited our `cls_token` embedding size to 100 to throttle information, preventing the decoder from accessing all the information from the ViT. By increasing the embedding size to 300, the model may better disentangle shape representation in the `cls_token` embedding, improving our ability to modify only the `x_location` while keeping the shape constant.

6.3. Expanding the Scope of Modifiable Properties

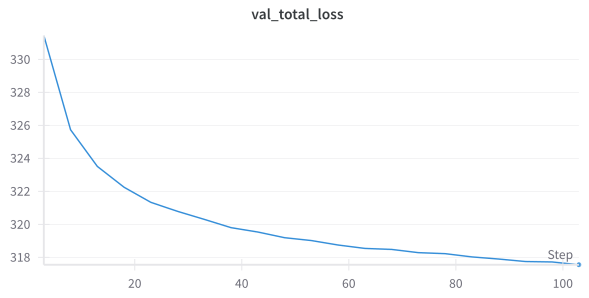
Our current work primarily focuses on modifying location and shape properties. Expanding the scope to include other visual attributes such as color, texture, and complex object features could provide a more comprehensive understanding of the capabilities and limitations of ViTs in this context. Utilizing datasets like CLEVR [7], which offers a range of object and scene properties, and CelebA [9], which includes detailed facial attributes, will be particularly beneficial for this purpose.

7. Appendix

7.1. Figures



(a) Total Training Loss Curve for ViT



(b) Total Validation Loss Curve for ViT

Figure 5: Training and Validation Loss Curves for ViT

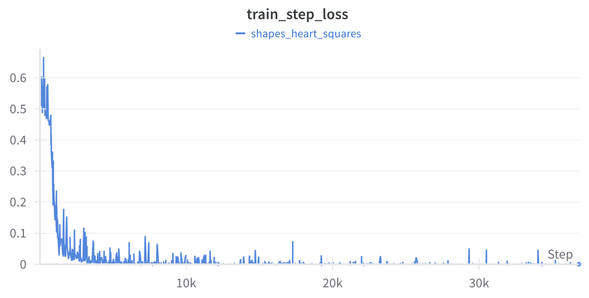


Figure 6: Binary CNN Training Loss Curves

8. Contributions and Acknowledgements

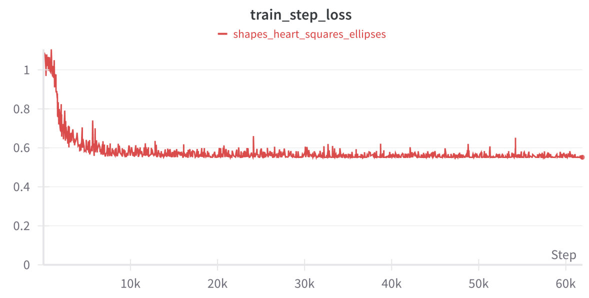


Figure 7: Multiclass CNN Training Loss Curves

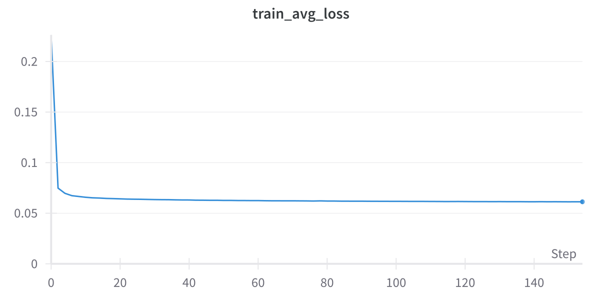


Figure 8: NN Prediction of Delta Embeddings Training Loss Curves

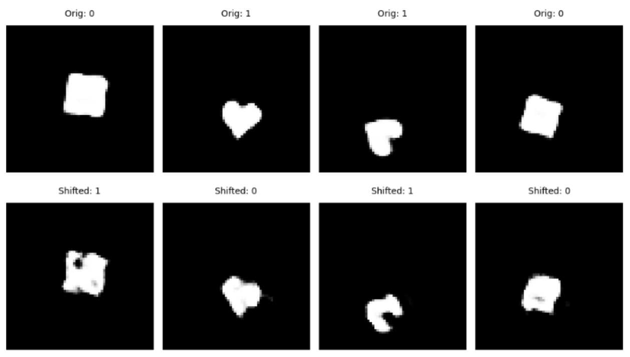


Figure 9: Examples of modified images using the Linear Regression approach.

References

[1] R. Belanec, P. Lacko, and K. Malinovská. Controlling the output of a generative model by latent feature vector shifting. In *2023 World Symposium on Digital Intelligence for Systems and Machines (DISA)*. IEEE, Sept. 2023. 1, 2, 4, 8

[2] S. Bills, N. Cammarata, D. Mossing, H. Tillman, L. Gao, G. Goh, I. Sutskever, J. Leike, J. Wu, and W. Saunders. Language models can explain neurons in language models. https://openaipublic.blob.core.windows.net/neuron-explainer/paper/index.html, 2023. 2

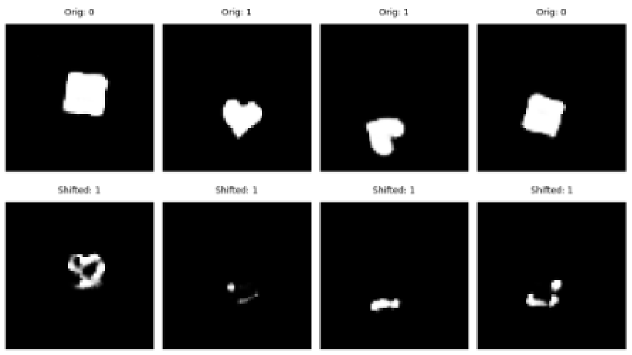


Figure 10: Examples of modified images using the Linear SVM approach.

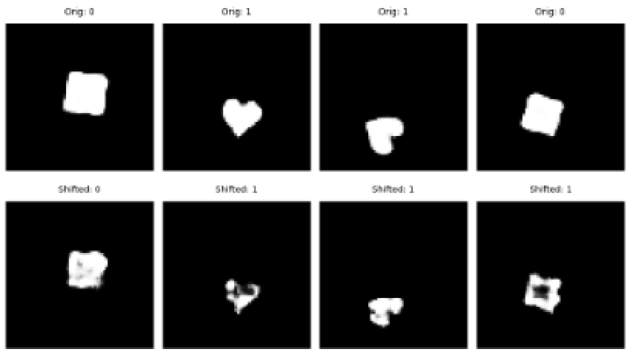


Figure 11: Examples of modified images using the Non-linear SVM approach.

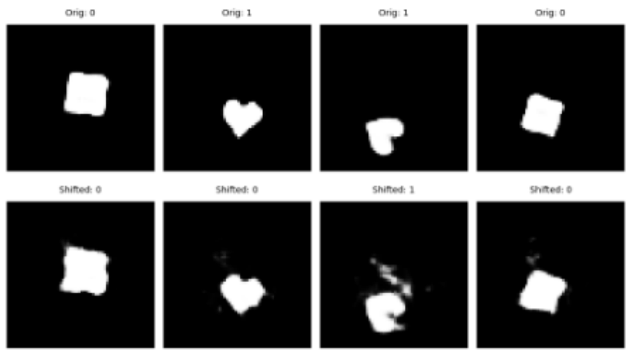


Figure 12: Examples of modified images using the Ensemble approach.

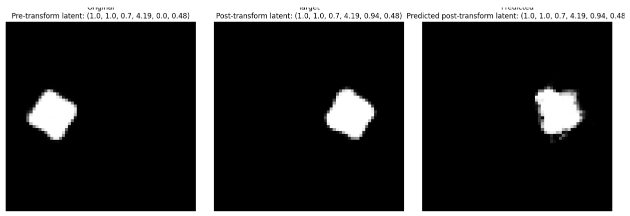

Pre-transform latent: (1.0, 1.0, 0.7, 4.19, 0.0, 0.48)
Post-transform latent: (1.0, 1.0, 0.7, 4.19, 0.94, 0.48)
Predicted post-transform latent: (1.0, 1.0, 0.7, 4.19, 0.94, 0.48

Figure 13: Examples of modified images using the NN approach.

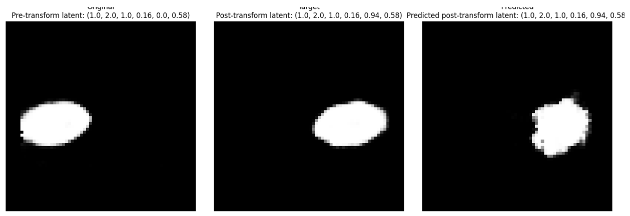

Pre-transform latent: (1.0, 2.0, 1.0, 0.16, 0.0, 0.58)
Post-transform latent: (1.0, 2.0, 1.0, 0.16, 0.94, 0.58)
Predicted post-transform latent: (1.0, 2.0, 1.0, 0.16, 0.94, 0.58

Figure 14: Examples of modified images using the NN approach.

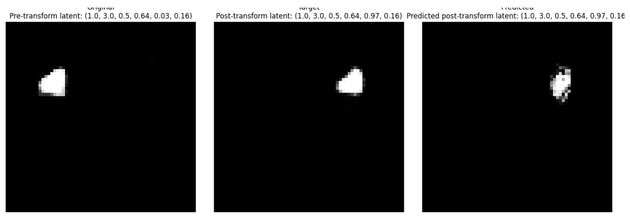

Pre-transform latent: (1.0, 3.0, 0.5, 0.64, 0.03, 0.16)
Post-transform latent: (1.0, 3.0, 0.5, 0.64, 0.97, 0.16)
Predicted post-transform latent: (1.0, 3.0, 0.5, 0.64, 0.97, 0.16

Figure 15: Examples of modified images using the NN approach.

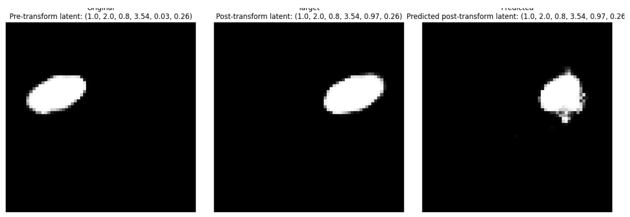

Pre-transform latent: (1.0, 2.0, 0.8, 3.54, 0.03, 0.26)
Post-transform latent: (1.0, 2.0, 0.8, 3.54, 0.97, 0.26)
Predicted post-transform latent: (1.0, 2.0, 0.8, 3.54, 0.97, 0.26

Figure 16: Examples of modified images using the NN approach.

[3] T. Bricken, A. Templeton, J. Batson, B. Chen, A. Jermyn, T. Conerly, N. Turner, C. Anil, C. Denison, A. Askell, R. Lasenby, Y. Wu, S. Kravec, N. Schiefer, T. Maxwell, N. Joseph, Z. Hatfield-Dodds, A. Tamkin, K. Nguyen, B. McLean, J. E. Burke, T. Hume, S. Carter, T. Henighan, and C. Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html. 1, 2

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. 1, 3

[5] G. Goh, N. C. †, C. V. †, S. Carter, M. Petrov, L. Schubert, A. Radford, and C. Olah. Multimodal neurons in artificial neural networks. *Distill*, 2021. https://distill.pub/2021/multimodal-neurons. 2

[6] E. Hoffer and N. Ailon. Deep metric learning using triplet network, 2018. 5

[7] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. B. Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. *CoRR*, abs/1612.06890, 2016. 8

[8] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan. Supervised contrastive learning. *CoRR*, abs/2004.11362, 2020. 4

[9] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015. 2, 9

[10] L. Matthey, I. Higgins, D. Hassabis, and A. Lerchner. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017. 6