# Interpretable Image Classification

by

Anagha Joshi

(Under the Direction of Jaewoo Lee)

## Abstract

Deep neural networks are unparalleled tools in Machine Learning that provide near-human performance on image classification tasks [12]. However, their inability to explain their predictions is the primary barrier to adopting them despite their promising accuracy. Many model agnostic interpretation methods[26] have tried to explain the reasoning process behind such "black-box" deep learning models.

Prototype-based classification is a classical form of case-based reasoning [19] that claims to be a promising alternative to using black-box image classification models. The current state-of-the-art method among prototype-based classification is the ProtoPNet [7].

However, we've identified numerous flaws with this approach that stem from their design choice of using a deep CNN as the base architecture. Essentially, conventional CNNs [17][15] can perform well only with deeper layers. This means that to associate image pixels far away from each other, we are forced to ensure an adequate number of layers (depth) in our model, which adds to more complexity and resource utilization. By switching to a transformer-based architecture, we can eliminate this restriction of "depth" imposed by CNNs. Hence, we propose a transformer-based image classifier architecture that uses the self-attention mechanism to learn what parts of the image are important for the given visual task, which can help encode feature maps into high-level semantic features. By the new trend of using Transformers in computer vision tasks, this work investigates the impact that Transformers can cause in the field of Interpretable Machine Learning with respect to 1. Interpretability and 2. Performance and Resource Utilization.

Our results show that while Transformers show promising performance in reducing the complexity of the interpretable machine learning models, they hurt their interpretability.

INDEX WORDS: convolutional neural networks, CNNs, DNNs, interpretability, prototypes, classification, transformer, encoder, decoder, attention-mechanism, self-attention, multi-level attention.

# Interpretable Image Classification

by

Anagha Joshi

B.E., 2015
Pune, Maharashtra
India

A Thesis Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the
Degree

Master of Science

Athens, Georgia

2021

©2021
Anagha Joshi
All Rights Reserved

# Interpretable Image Classification

by

Anagha Joshi

Major Professor: Jaewoo Lee

Committee: John Miller
Sheng Li

Electronic Version Approved:

Ron Walcott
Vice Provost for Graduate Education and Dean of the Graduate School
The University of Georgia
August 2021

# Contents

**List of Figures** v

**List of Tables** vi

**1 Introduction** 1
- 1.1 Interpretability Methods . . . 1
- 1.2 Prototypical Learning . . . 2

**2 Related Work** 5
- 2.1 PrototypeNet: Case based reasoning through prototypes . . . 5
- 2.2 ProtoPNet: Prototypical Part Network . . . 6
- 2.3 DPNet: Dynamic Prototype Network . . . 6
- 2.4 CBC: Classification by Components . . . 7
- 2.5 ProtoPShare . . . 7
- 2.6 Neural Prototype Trees for Interpretable Fine-grained Image Recognition . . . 8
- 2.7 This Looks Like That, Because ... Explaining Prototypes for Interpretable Image Recognition . . . 8

**3 Background** 10
- 3.1 Prototypes . . . 10
- 3.2 Prototype based Learning . . . 11
- 3.3 Transformers in Computer Vision . . . 12
- 3.4 Why using Transformers may be a better alternative to CNNs 16

**4 Transformer-based Prototype Network TPNet** 18
- 4.1 Architecture . . . 18
- 4.2 Training Objective and Loss function . . . 20
- 4.3 Training Algorithm . . . 21

**5 Experiments and Results** **23**
5.1 Dataset . . . . . 23
5.2 Model Parameters . . . . . 23
5.3 Experimental Setup . . . . . 24
5.4 Results and Discussion . . . . . 24

**6 Conclusion** **29**
6.1 Conclusion . . . . . 29
6.2 Future Directions . . . . . 29

**Appendices** **30**

**A Appendix A** **30**
A.1 A modified Architecture . . . . . 30

**Bibliography** **35**

# List of Figures


1.1 ProtoPNet's Reasoning Process . . . . . 3

3.1 Self-attention module that takes Image feature maps as inputs. These maps are then multiplied with weight matrices $W^Q$, $W^K$, $W^V$ to transform them to the object queries namely Query, Key, and Value. . . . . 12
3.2 Multi head attention with multiple sets of Query/Key/Value weight matrices. . . . . 13
3.3 Basic Transformer architecture with an Encoder and Decoder which comprise of multiple Multi-head attention modules. . . 14
3.4 Shifted Window Approach as depicted by the authors of Swin Transformer . . . . . 15
3.5 Architecture of Swin-T Transformer . . . . . 16
3.6 Swin-T Transformer as our backbone (Equivalent of resnet18) 16
3.7 Both images are similar for a CNN as it does not consider the relative positioning of facial features . . . . . 17

4.1 TPNet Architecture. . . . . 22

5.1 System wide CPU usage by all three models . . . . . 26
5.2 GPU Percentage used by all three models . . . . . 27
5.3 Reasoning Process of ProtoPNet . . . . . 27
5.4 Reasoning Process of TPNet . . . . . 28

A.1 Modified Architecture with Hungarian matcher . . . . . 31
A.2 Hungarian unbalanced weighted assignment between prototypes and transformer output patch . . . . . 33

# List of Tables

5.1 Performance Comparison of our model against Baseline. . . . 25
5.2 Classification Performance of a transformer-based classifier versus a CNN-based classifier . . . . . 25

# Chapter 1

# Introduction

Convolutional neural networks (CNNs) [17] achieve human-level performance on many visual learning tasks such as image classification [12, 15, 31], objection detection [6, 38, 37]. Yet it becomes nearly impossible to judge the model's reasoning process due to their black-box nature which is especially important in critical domains with high stake decisions[27] such as legislation, law enforcement, medicine, etc. In response to this demand on interpreting automatic decision-making, a new field of explainable Artificial Intelligence (XAI) has emerged.

## 1.1 Interpretability Methods

Interpretability methods can be classified as either intrinsically interpretable (model-specific) or post-hoc (model-agnostic). Intrinsically intrepretable models to use their simple internal structure of the model, such as short decision trees, sparse linear models or the RuleFit algorithm.[10] [9][27] whereas post-hoc interpret-ablity methods learn an additional white-box model to explain the first black box model[26]. However, intrinsically interpretable models are not preferred over complex models such as deep convolutional neural networks because the latter provide improved performance.[17]

While posthoc methods grant users the ability to use complex models and provide visualization techniques [30] to reverse-engineer the reasoning process from the outcome, they still do not show the *actual* reasoning of the black-box model.

## 1.2 Prototypical Learning

Example-based explanation methods are an alternative to posthoc interpretability methods. The difference between example-based explanation methods and model-agnostic methods is that the example-based methods explain a model by selecting instances of the dataset and not by creating a visualization or summary of features.

Prototype-based classification falls under this category of example-based explanation methods and is a classical form of case-based reasoning [14]. Over time, the meaning of the word "prototype" has evolved from being like observation in the dataset to more generalized definitions like a combination of several observations or a latent representation learned in the feature space. The degree of closeness between an image and a prototype can either be measured in image space or the latent space, which the model learns. Similar images in the latent space are clustered closely together, while the dissimilar images are scattered far apart with respect to a distance metric. The central idea of Prototypical Part Network (ProtoPNet) [7] was to let the network learn a set of prototypical parts from the given image and score it based on its similarity to the learned prototypes. Hence their classification depends on how much a part of this image "looks like" a particular prototypical part learned by the network. ProtoPNet's reasoning process is shown in Figure 1.1, and it explains how their model reached a classification decision for the test image of a red-bellied woodpecker. For the given test image $x$ in Figure 1.1 (top), their model compares each patch of its latent features $f(x)$ with the learned prototype $p_j$ (visualized in the column "Prototypes) belonging to class k. This comparison generates a map of similarity scores for each prototype, which is then upsampled and overlaid on the original image to understand which part of the given test image was activated by that prototype. They call this map an "activation map." In Figure 1.1 (under the column "Activation Map"), we can see that the head region of the bird in the original image is most strongly activated by the first prototype wing region by the second and so on. This strongly activated region is also marked by a bounding box in the "Original image" column. In this example, we can interpret that their model finds a high similarity between the head of the given image and the prototypical head of the red-bellied woodpecker. The degree of this similarity is measured with a similarity score of 6.499. Similarly, the wing of the original image corresponds most strongly with the prototypical wing of the red-bellied woodpecker with a similarity score of 4.392. All these similarity scores are weighted and summed together to give a final score for the bird belonging to this class, and the whole process is repeated for all other classes. The

class with the highest number of points is predicted as the outcome, which in this case is the "red-bellied woodpecker".

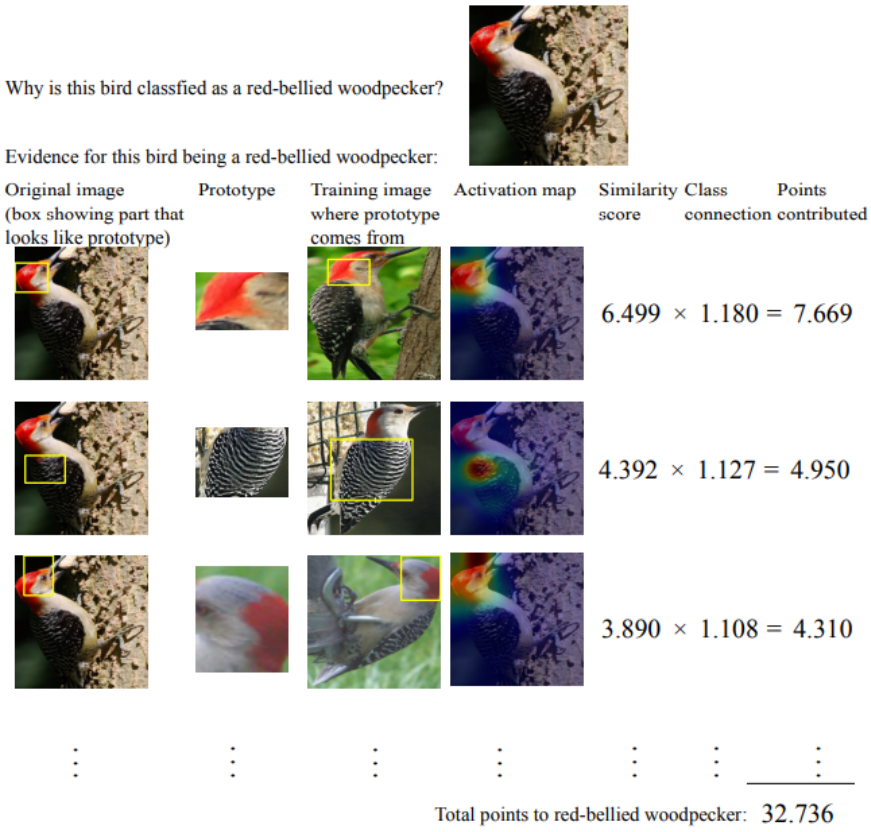


Figure 1.1: ProtoPNet's Reasoning Process

### 1.2.1 Limitations of the current approach

Current Prototypical Learning methods follow the paradigm of adding a prototype layer on top of a CNN-based architecture. We've identified several flaws in ProtoPNet's architecture that may stem from their design choice of using a CNN as their base architecture:

1. **CNN performance is dependent on depth:** CNNs perform best with deeper layers. This means that in order to associate image pixels that are far away from each other (receptive field size), we are forced to ensure an adequate number of layers (depth) in our model, which in turn adds to more complexity and resource utilization.

2. **Redundant input features:** Most of the current network architectures accept all the features as input, neglecting the fact that not all features are helpful in predicting the desired outcome. Instead, we could supply image example embeddings.

3. **CNNs cannot encode relative spatial information**. This means it is only concerned with detecting certain features and does not consider their positioning with respect to each other. Location of objects is not a significant characteristic in our application. But it may be important in some other visual tasks.

The above drawbacks associated with the use of CNN's can be mitigated by replacing them with a Transformer[35]. Transformers do not work with a local receptive field and can accept all the pixels of the input in one pass. Hence they can associate image pixels that are far away from each other without the restriction of depth. Moreover, we can also provide positional encoding to the transformers along with the input images if the location of objects in the image were a significant characteristic for our visual task.

To this end, we propose a transformer-based interpretable image classifier architecture with our model. By using transformers, we can avoid the restriction of "depth" that was earlier imposed by CNNs and thereby reduce the complexity of the model. Chapter 4 describes this architecture in more detail, and our code is available via https://github.com/anaghaj111/ExplainableAI/.

Chapter 2 describes the ProtoPNet model [7] and compares some of its contenders in the prototype learning domain. Chapter 5 shows our experimental setup, training method, and Chapter 6 compares our model's interpretability with existing approaches.

# Chapter 2

# Related Work

## 2.1 PrototypeNet: Case based reasoning through prototypes

1. The idea of a 'prototype layer' in a Deep Neural Network was first proposed in 2017 [19]. Prototype layer is an additional layer in the standard classifier DNN architecture that aids decision interpretation. Their network was a type of prototype classifier, in which observations are classified based on their closeness to a prototype observation within the data-set.In this work [19],prototypes are understood to be similar to training examples of the data-set and the set of prototypes is representative of the whole data-set.The architecture of Prototype based network consists primarily of two components:

   (a) An auto-encoder (comprises an encoder and a decoder)

   (b) A prototype classifier network

2. Drawbacks

   - Prototype based network worked great when used for classifying images of handwritten digits. However, when tried on more complex images such as natural images with varied backgrounds, the decoder failed to produce realistic prototype images.
   - Fine-grained comparisons between parts of images were a problem too since prototypes in Prototype based Network represented whole images.

## 2.2 ProtoPNet: Prototypical Part Network

1. Prototypical Part Network or ProtoPNet [7] was introduced to solve the problem of applying prototype based learning to natural images with various backgrounds. It also addressed the task of carrying out fine-grained comparisons between parts of images since prototypes in ProtoPNet represented parts of images and *not* whole images. The main idea here is to let the network learn a set of *prototypical parts* from the data and score the given image based on its similarity to the parts of learned prototypes.

2. Drawbacks

   (a) The number of prototypes is predetermined.

   (b) The prototypes are neither shared nor is the relationship between the prototypes explored.

   (c) This number of prototypes is fairly large owing to the fact that each of them is designated to one and only one class. Because of this, the interpretability is negatively influenced by the large size and added complexity.

   (d) During the training, the prototypes of different classes are pushed away from each other in the representation space. The prototypes with similar semantic can seem distant. This makes the predictions of the model unstable.

   (e) The prototypes are class dependent.

   (f) The training method does not penalize prototype vectors of the same class for overlapping with each other.

## 2.3 DPNet: Dynamic Prototype Network

1. Essentially DPNet [34] is an interpretable prototype-based neural network that is meant to capture the dynamic features, such as unnatural movements and temporal artifacts, and uses them to explain the prediction. Their definition of prototypes and training procedure is identical to that of ProtoPNet. However, DPNet uses a video dataset instead of an image dataset. Due to this the prototypes are visualised as videos instead of images.

2. Drawbacks

(a) The number of prototypes is predetermined.

(b) The prototypes are neither shared nor is the relationship between the prototypes explored.

(c) This number of prototypes is fairly large. Because of this, the interpretability is negatively influenced by the large size and added complexity.

## 2.4 CBC: Classification by Components

1. Main idea:This classification method [3] is inspired by BIEDERMAN's theory [2] which explains how humans recognize complex objects. Briefly, it states that humans identify complex objects by assuming that objects can be decomposed into general parts that function as structural primitives, called components. Classification of objects is therefore carried out by matching the extracted decomposition plan with a class Decomposition Plan (DP) for each potential object class. Basically, the class DPs describe which components are vital to be detected and which components are important to leave out for an object to belong to a specific class. Hence CBC classifies its input by applying positive, negative, and indefinite reasoning over an extracted DP. Their method claims to retain the advantages of Neural Nets like being end-to-end trainable on large datasets and yet achieving a high accuracy on complex tasks.

2. Drawbacks

   (a) They use a detection function in order to detect extracted DP is positive, negative or indefinite. It is still unclear as to what decides if a detection probability function is suitable for the given task or not. They write it off as an unanswered question.

   (b) Since the architecture is CNN based, it suffers from the same limitations that ProtoPNet does.

## 2.5 ProtoPShare

1. Main idea: This method [28] argues that prototypes should not be assigned to a specific class but should be shared among all the classes. That way we end up with fewer prototypes and do not necessarily compromise interpretability. It combines the idea of prototype parts to explain the reasoning behind its predictions.

2. Drawbacks

   (a) It might generalize some nuanced prototypes by categorizing them as the same prototype.

   (b) Still doesn't outperform ProtoPNet when it comes to accuracy in Prediction.

## 2.6 Neural Prototype Trees for Interpretable Fine-grained Image Recognition

1. Main idea: In their method [23], a ProtoTree or a Neural Prototype Tree hierarchically tracks a given image via a binary tree for interpretable image identification. Unlike ProtoPNet they refrain from multiplying weights with similarity scores. Instead, they just show something called a local explanation that depicts the routing of an image sample through this ProtTree. They too reject the idea of having class-specific prototypes.

2. Drawbacks

   (a) Their loss function just has the Cross Entropy. So they do not use any mechanism to prevent two prototypes from being similar like ProtoPNet, DPNet do.

## 2.7 This Looks Like That, Because ... Explaining Prototypes for Interpretable Image Recognition

1. Main idea: Their method [24] proposes to enhance the explanations provided by ProtoPNet. Their main argument is that while ProtoPNet points out the prototypes that the classification was based upon, it doesn't reveal "why" the model considers a prototype? In their work,[24] they augment prototypes with textual quantifiable data about visual characteristics regarded important by the classification model.

2. Drawbacks

   (a) The additional information that they provide along with prototypes are quantities associated with shape, hue, saturation, contrast, and texture. Contingent on the nature of the visual task and the

domain in which it is being used, these quantities may or may not be valuable to make or explain the decision. Hence this method cannot be universally applied.

Since most of the prototype-based learning techniques reviewed above are either similar to or built upon the ProtoPNet, we too choose it to be the base model for all our experiments along with a pre-trained ResNet50 that qualifies as a benchmark for image classifier. We make the choice of 50 layers or less owing to the limited GPU memory capacity that we have available.

# Chapter 3

# Background

## 3.1 Prototypes

Traditionally, a prototype was defined as an instance of data that is demonstrative of all the data. Over time, the meaning of the word "prototype" has evolved from being similar to an observation in the dataset to more generalized definitions like the combination of several observations or a latent representation learned in the feature space. In our work, we use the more generalized definition of prototypes as compared to the traditional one. Additionally, in order to make fine-grained comparisons, the prototypes in our network can be visualized as parts of images rather than the whole image.

### 3.1.1 Advantages of using prototypes for classification

1. **Prototypes can even improve humans' ability to classify:** In a user study conducted in 2016, [1][22], participants were given images which they were asked to visually match to one of two sets of images, where each set represented a class (e.g., two dog species). It was observed that the participants that performed best were shown prototypes instead of random images of a class.

2. **Model and Data agnostic**: Prototypes can work with any kind of data and any kind of machine learning model.

3. User can **choose the number** of prototypes

## 3.2 Prototype based Learning

In prototype-based learning algorithms [14, 25, 4, 13, 18], we introduce an "additional" task in addition to the original classification task. The additional task is to learn *prototypes*, a set of representative observations in the dataset, and the classification is based on the similarity between the given input and each prototype. The prototype classifier [18] combines an auto-encoder with a classification network. Let $f : \mathbb{R}^p \to \mathbb{R}^q$ and $g : \mathbb{R}^q \to \mathbb{R}^p$ denote the encoder and decoder in the autoencoder, respectively. A fully connected network, denoted by $h : \mathbb{R}^q \to \mathbb{R}^K$, takes the feature vectors extracted by the autoencoder as input and returns a distribution over classes.

- The network learns $m$ prototype vectors $\mathbf{p}_1, \ldots, \mathbf{p}_m \in \mathbb{R}^q$.
- Given a a feature vector $\mathbf{z} = f(\mathbf{x})$, the prototype vector $\mathbf{u} \in \mathbb{R}^m$ is given by:

$$\mathbf{u} = \mathsf{proto}(\mathbf{z}) = \left(\|\mathbf{z} - \mathbf{p}_1\|_2^2, \|\mathbf{z} - \mathbf{p}_2\|^2, \ldots, \|\mathbf{z} - \mathbf{p}_m\|_2^2\right)^\mathsf{T} .$$

- The network computes the weighted sum of distances and converts in into a class probability by applying the softmax function:

$$\mathbf{v} = \mathbf{W}\mathbf{u}\,, \quad \Pr[Y = k \mid X = \mathbf{x}] = \frac{\exp(v_k)}{\sum_{k'=1}^{K} \exp(v_{k'})}\,.$$

- The training of network is done by minimizing the cost function consisting of four components $\mathcal{L} = \mathcal{L}_{\text{cl}}(h \circ f, D) + \lambda R(g \circ f) + \lambda_1 R_1(\mathbf{p}, D) + \lambda_2 R_2(\mathbf{p}, D)$, where
  - $\mathcal{L}_{\text{cl}} = -\frac{1}{n}\sum_{i=1}^{n} \log((h \circ f)_{y_i}(\mathbf{x}_i))$ is the cross-entropy loss to penalize the misclassification,
  - $\mathcal{R}(g \circ f, D) = \frac{1}{n}\sum_{i=1}^{n} \|(g \circ f)(\mathbf{x}_i) - \mathbf{x}_i\|_2^2$ is the reconstruction error for the autoencoder,
  - $R_1(\mathbf{p}, D) = \frac{1}{m}\sum_{j=1}^{m} \min_{i \in [n]} \|\mathbf{p}_j - f(\mathbf{x}_i)\|_2^2$ enforces $\mathbf{p}_j$ is close to at least one observation (in the latent space), and
  - $R_2(\mathbf{p}, D) = \frac{1}{n}\sum_{i=1}^{n} \min_{j \in [m]} \|f(\mathbf{x}_i) - \mathbf{p}_j\|_2^2$ is the sum of squared distances between examples and prototypes.

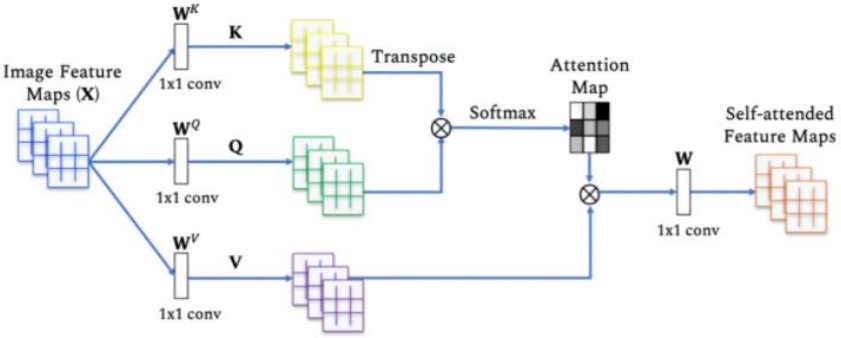


Figure 3.1: Self-attention module that takes Image feature maps as inputs. These maps are then multiplied with weight matrices $W^Q, W^K, W^V$ to transform them to the object queries namely Query, Key, and Value.

## 3.3 Transformers in Computer Vision

The transformer is a deep learning model that is used primarily in the field of natural language processing (NLP).[35]

Just like recurrent neural networks (RNNs),[29] Transformers are designed to handle sequential data, such as natural language, for tasks such as translation and text summarizing. However, unlike RNNs, Transformers do not require that the sequential data be processed in the order. For example, if the input data is a natural language sentence, the transformer does not need to process the beginning of it before the end. Due to this feature, the transformer allows for much more parallelization than RNNs and therefore reduced training times.[35] .

### 3.3.1 Self-Attention

One of the fundamental concepts behind the success of Transformers is the *self-attention* mechanism.

A self-attention module takes *n* inputs and return *n* outputs. The mechanism allows the inputs to interact with each other and figure out which input to pay more attention to. The outputs are aggregates of these interactions and attention scores.

Suppose we have a language input - *"The dog couldn't cross the lane because it was too tired."*. Here, "the" refers to "dog" and thus should have a good connection with it. To ensure this, the sequence of vectors **x** is encoded into an-

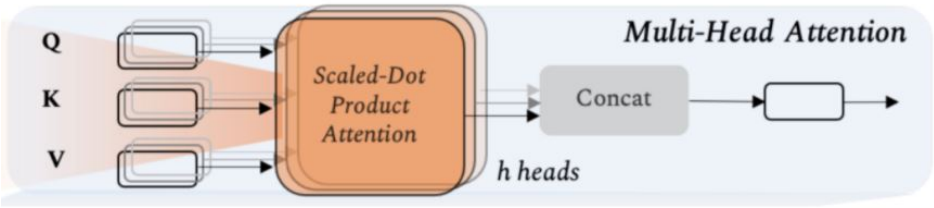


Figure 3.2: Multi head attention with multiple sets of Query/Key/Value weight matrices.

other sequence of vectors **z**. The first vector represents the word and the second one represents the original vector and its relationship around it. A pictorial depiction of how the self-attention mechanism works can be seen in 3.1. The operation of self-attention can also be described with the following equation:

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\intercal}{\sqrt{d}}\right)\mathbf{V} \tag{3.1}$$

### 3.3.2 Multi-Head Attention

The idea of the self-attention mechanism helps the model understand the relationship between each part of the input, and we extend the idea to multiple attention heads instead of just one. In other words, by adding multiple heads, we expand the model's ability to focus on different positions. Thereby providing multiple "representation subspaces." As seen in the Figure 3.2 Multi head attention entails multiple sets of Query/Key/Value weight matrices. The number of heads equals the number of such sets we end up with.

### 3.3.3 Basic Transformer

The basic transformer, as seen in Figure 3.3, is made up of "Encoder" layers and "Decoder" layers. Each of these "Encoder/Decoder" layers contains multiple multi-head attention blocks within them, followed by blocks handling the Addition and Normalization operations. This architecture was designed keeping NLP tasks in mind.[35] However, Transformers are now being applied to a broad range of computer vision tasks such as object detection[6][38][37][33] and image classification to name a few. [11]

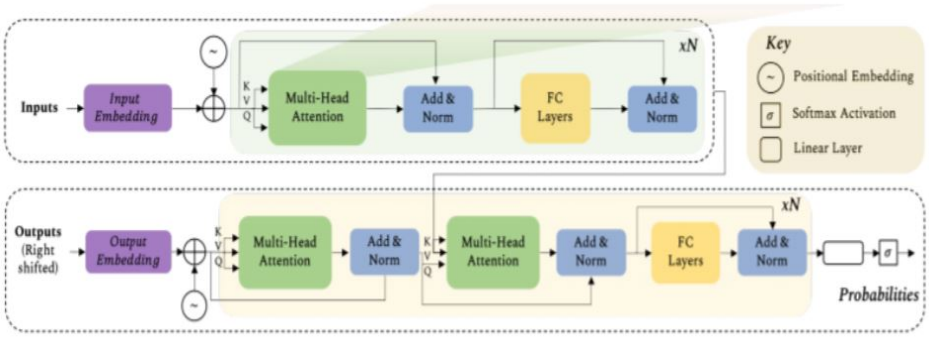


Figure 3.3: Basic Transformer architecture with an Encoder and Decoder which comprise of multiple Multi-head attention modules.

### 3.3.4 SWIN Transformer: A Hierarchical Vision Transformer using Shifted Windows

The Swin Transformer or otherwise known as the "Shifted Window" Transformer, was introduced by Microsoft this year[20]. This transformer is being widely adopted in the image [5] as well as video [21] recognition tasks. In their work, they claim that the Swin Transformer can replace the general-purpose CNN-based backbone and still yield similar, if not better, performance for the said visual task. [20]. The authors claim that. The main idea of the shifted window approach for calculating self-attention in the Swin Transformer architecture is shown in Figure 3.4. In layer l (on the left), a regular window partitioning scheme is implemented, and self-attention is calculated within each window. In the subsequent layer l + 1 (to the right), the window partitioning is shifted, causing new windows. The self-attention calculation in the new windows passes the boundaries of the preceding windows in layer l, delivering connections among them.

**Overall architecture:**The figure 3.5 is the architecture of the Swin-T transformer as depicted in their introductory work[20]. First, a patch splitting module splits the given input RGB image into non-overlapping patches, just like ViT (Vision Transformer).[8] Then, each patch is regarded as a "token." The concatenation of the RGB pixel values becomes the feature of each patch. A linear embedding layer alters the dimension of this raw pixel-valued feature and thereby projects it to an arbitrarily chosen dimension 'C.' Numerous Transformer blocks with modified self-attention computation are applied on these patch tokens. (H/4 x W/4) the number of tokens is maintained throughout the

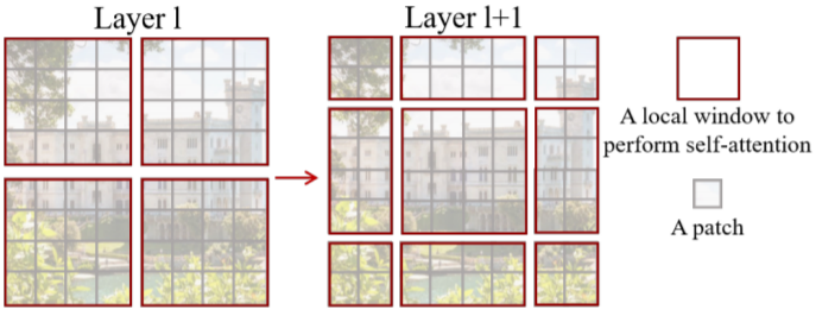


Figure 3.4: Shifted Window Approach as depicted by the authors of Swin Transformer

Transformer blocks, and along with the linear embedding is termed as "Stage 1". This number of tokens is further reduced by using patch merging layers as the depth of the network goes on increasing. After the first patch merging layer, the features of each group of 2 × 2 adjacent patches are concatenated, and a linear layer is applied to 4C dimensional features. This decreases the number of tokens by a multiple of 2×2 = 4, i.e., 2× downsampling of the resolution, and the output dimension is 2C. For further feature transformation, Swin Transformer blocks are applied subsequently, with the resolution maintained at H/8 x W/8. This decreases the number of tokens by a multiple of 2×2 = 4, i.e., 2× downsampling of the resolution, and the output dimension is 2C. For further feature transformation, Swin Transformer blocks are applied subsequently, with the resolution maintained at H/8 x W/8. This first block comprising of patch merging and feature transformation is referred to as "Stage 2". This procedure is repeated twice again, and referred to as "Stage 3" and "Stage 4", with output resolutions being H /16 × W/ 16 and H /32 × W /32 , correspondingly. All of these stages are said to jointly yield a hierarchical representation, having the same feature map resolutions like those of conventional convolutional networks. In our applications, we use the Swin-T as the backbone for feature extractions. Figure **??** shows that we use Stage 1 and Stage2 of the Swin-T as our backbone. A standard Swin Transformer block is developed by substituting the standard multi-head self-attention (MSA) module in a Transformer block with a module based on shifted windows (shown in **??**), with other layers kept unchanged. As depicted in Figure 3.6, a standard Swin Transformer block comprises of SW-MSA, i.e., a shifted window-based MSA module, followed by a 2-layer MLP (Multilayer Perceptron) with GELU nonlinearity in the middle. Next, before

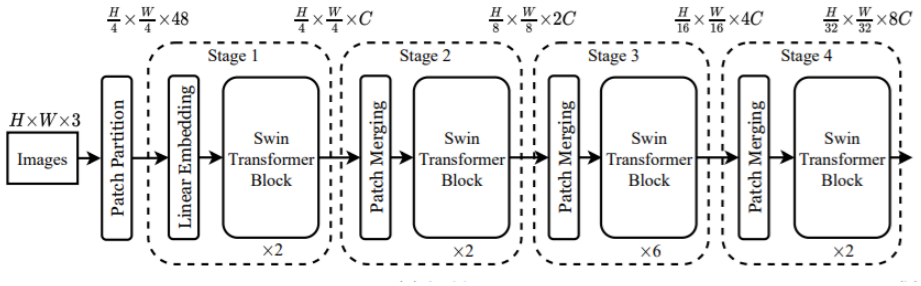


Figure 3.5: Architecture of Swin-T Transformer

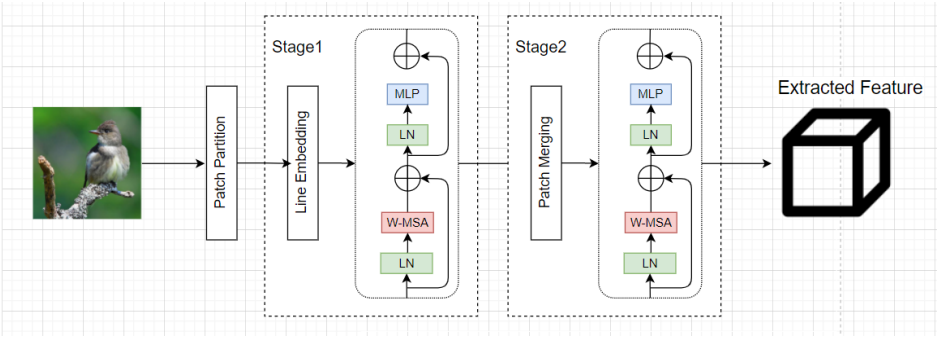


Figure 3.6: Swin-T Transformer as our backbone (Equivalent of resnet18)

each MSA module and each MLP, a LayerNorm (LN) layer is applied, and a residual connection is applied after each module.

## 3.4 Why using Transformers may be a better alternative to CNNs

Before we understand the architectural details of our model, it is important to know why we choose a Transformer backbone instead of a CNN. Convolutional Neural Networks have the following limitations:

1. **CNN performance is dependent on depth:** CNNs perform best with deeper layers. This means that in order to associate image pixels that are far away from each other (receptive field size), we are forced to ensure an adequate number of layers (depth) in the model, which in turn adds to more complexity and resource utilization. With Transformers, this restriction of ensuring enough depth is obsolete.

2. **Translation in-variance:** Translation in-variance means that if the same object in an image slightly changes its orientation or position, the CNN may not be able to recognize it. This is because CNN is only concerned with detecting certain features and does not consider their positioning with respect to each other. For example, consider the two faces shown in the figure 3.7. A CNN won't be able to distinguish between the face on the left and the distorted one on the right due to its inability to understand relative spatial positions. Since Transformers don't work with a local receptive field like CNNs, they have the ability to take into account the positioning of objects in the image as well.

3. Another major drawback of a CNN is their use of **pooling layers**. Pooling layers ignore a lot of meaningful information, such as the relation between the part and the whole. Consider the face detector in the figure 3.7 again. We know that we have to combine some features such as the mouth, 2 eyes, face oval, and a nose in order to say that is this is a face, but the CNN would only check if those 5 features are present with high probability. Pooling layers are absent in Transformers, so we don't lose meaningful information by switching to them.

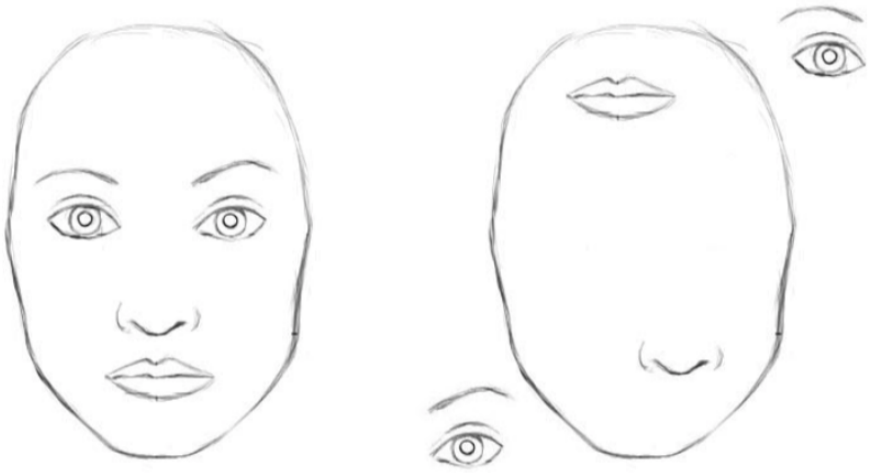

Figure 3.7: Both images are similar for a CNN as it does not consider the relative positioning of facial features

# Chapter 4

# Transformer-based Prototype Network TPNet

In this chapter, We discuss the architecture of TPNet. The main idea here is to let the network learn a set of *prototypical parts* from the data and score the given image based on its similarity to the parts of learned prototypes.

## 4.1 Architecture

Fig. 4.1 gives an overview of the architecture of our network that combines the power of attention mechanism along with the interpretability of prototypes. Our network consists of 3 main components: a Swin-T backbone to extract a compact feature representation, a transformer encoder, and decoder, a prototype classifier network.

- **Feature extraction:** We begin with a tensor $\mathbf{x} \in \mathbb{R}^{3\times H_0\times W_0}$ representing an RGB image. We then use a pre-trained resnet18 network to extract the meaningful feature vectors $f_{\mathsf{conv}}(\mathbf{x}) : \mathbb{R}^{3\times H_0\times W_0} \rightarrow R^{C\times H\times W}$ from this given image. To work with a decent number of features we further reduce the the channel dimension of $f_{\mathsf{conv}}(\mathbf{x})$ to from $C$ to $d$. Hence the result is a feature map $f(\mathbf{x}) : \mathbb{R}^{d\times H\times W}$.

- **Transformer Encoder:** Next, this feature map $f(\mathbf{x})$ is sent to the Transformer Encoder. Since the Transformer encoder expects a sequence as an input, we collapse the spatial dimensions of $f(\mathbf{x})$ to create another feature map $\mathbf{z}$ of size $d \times HW$. $\mathbf{z}$ is viewed as a sequence of $d$ features, each of which has the length of $HW$.

The self-attention mechanism in the transformer encoder learns the global pairwise interactions between $d$ feature maps. It transforms the input into a new $d \times HW$ sequence of embeddings $\mathbf{z}'$ where each element in the sequence is a *weighted* combination of (transformed) input elements. Each element in the transformer output $\mathbf{z}'$ represents an embedding of size $H \times W$, and there are $d$ such embeddings.

- **Transformer Decoder:** Along with the Encoder output, another input passed to the Transformer decoder is obtained by reshaping the image $\mathbf{x} \in \mathbb{R}^{3 \times H_0 \times W_0}$ into a sequence of flattened 2D patches $\mathbf{x_p} \in \mathbb{R}^{N \times (P^2 \cdot 3)}$. $(H_0, W_0)$ is the resolution of the original image with 3 channels, $(P, P)$ is the height and width of each image patch, and $N = HW/P^2$ is the resulting number of patches. To keep a consistent embedding size of $d$ throughout all Transformer layers, we map the flattened patches to $d$ dimensions.

  We unflatten this Decoder output and reshape it back to its 3-dimensional size $\mathbf{z}'_\theta : R^{d \times H \times W}$ for further processing.

- **Prototype Classifier Network:** This portion of the network is responsible for handling the "additional" task of learning *prototypes*, a set of representative observations in the dataset, and the classification is based on the similarity between the given input and each prototype. Our network learns $m$ prototype vectors $\mathbf{p}_1, \ldots, \mathbf{p}_m \in R^d$ whose shape is $H_1 \times W_1 \times d$ where $H_1 < H$ and $W_1 < W$. The depth of each prototype is the same as that of the reshaped transformer output $\mathbf{z}'_\theta$, but the height and width of the prototypes are smaller. This gives the network the ability to make fine-grained comparisons between parts of images. In other words, each prototype represents some prototypical activation pattern in a patch of $\mathbf{z}'_\theta$ which will, in turn, represent a patch in the original pixel space. With this in mind, consider two sets $Z$ and $P$. $Z$ denotes the set of patches of $\mathbf{z}'_\theta$ while $P$ denotes the set of prototypes padded with $\phi$ (no class). Once we have a similarity score for each matched pair representing how strongly a prototypical part is present in some patch of the input image, the m similarity scores are multiplied by the weight matrix $w_h$ in the fully connected layer following the prototype layer to produce the output logits. These output logits are then normalized using softmax to deliver predicted probabilities for a given image belonging to various classes.

## 4.2 Training Objective and Loss function

Let $D = (x_i, y_i)_{i=1}^{N}$ be our training dataset, where $x_i \in \mathbb{R}^{3\times H_0\times W_0}$ is a tensor representing the $i^{th}$ training image and $y_i$ is the corresponding class label associated with this image tensor.

We aim to solve the optimization problem shown below:

$$L(D;\theta) = CE(D;\theta) + \lambda_c L_{clus}(D;\theta) + \lambda_d L_{sep}(D;\theta)$$

where $CE(.), L_{clus}(.), and L_{sep}(.)$ are the cross-entropy loss, clustering loss, and separation loss, respectively.

Here, $\theta$ are the trainable parameters for the network. Let $f(x_i)$, now denote the reshaped output of the Transformer decoder, $P$ denote the set of all prototypes such that $P = \{p_j\}_{j=1}^{m}$ where $p_j$ refers to the $j^{th}$ prototype in the set $P$ and $P_k$ denote the set of all prototypes that belong to class $k$.

1. **Cross Entropy:** This loss term penalizes misclassification and is given by:

$$CE(D;\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} -1[y_i = k]log(\hat{y_k})$$

2. **Clustering Loss:** encourages each training image to have some latent patch that is close to at least one prototype of its own class and is given by:

$$L_{clus}(D; p_1, p_2, \dots p_m) = \frac{1}{N}\sum_{i=1}^{N} min_{p\in P_{y_i}} min_{z\in patches(f(x_i))} \|z-p\|_2^2$$

3. **Separation Loss:** prevents each latent patch of the training image from getting too close to prototypes that aren't of their own class and is given by:

$$L_{sep}(D; p_1, p_2, \dots p_m) = -\frac{1}{N}\sum_{i=1}^{N} min_{p\notin P_{y_i}} min_{z\in patches(f(x_i))} \|z-p\|_2^2$$

## 4.3 Training Algorithm

**Algorithm 1:** Training algorithm

**Input:** training dataset $[X, Y]$

**Output** model trained on above dataset $model.pth$

**Data:** $CUB_200_2011$ $[X, Y]$

`/* Initialization */`

$w_{backbone} \leftarrow$ weights pre-trained on ImageNet `// pre-trained Resnet18 or Swin-T`

$\forall j$ : prototype $p_j \leftarrow \text{Uniform}([0,1]^{D \times H_1 \times W_1})$

criterion $\leftarrow L(D;\theta)$ `// Loss Function`

optimizer $\leftarrow$ Adam `// Optimizer function`

$N \leftarrow$ max training epochs

`/* Training */`

**while** *(NOT converge AND* $L_{clus}(D;\theta) < -L_{sep}(D;\theta)$*)* **do**

  **for** *epoch in range(N)* **do**

    **for** *image, labels in D* **do**

      optimizer gradients $\leftarrow 0$

      $w_{backbone} \leftarrow w_{backbone} - \eta_{backbone} \nabla_{backbone} L(D;\theta)$

      $w_{trans} \leftarrow w_{trans} - \eta_{trans} \nabla_{trans} L(D;\theta)$

    **foreach** *prototype* $p_j$ **do**

      $p_j \leftarrow argmin_{z \in \{\tilde{z}:\tilde{z} \in patches(f(x,y)) \forall (x,y) \in [X,Y]\}} \|z - p_j\|$

  **if** acc $\geq target_acc$ **then** save model.pth

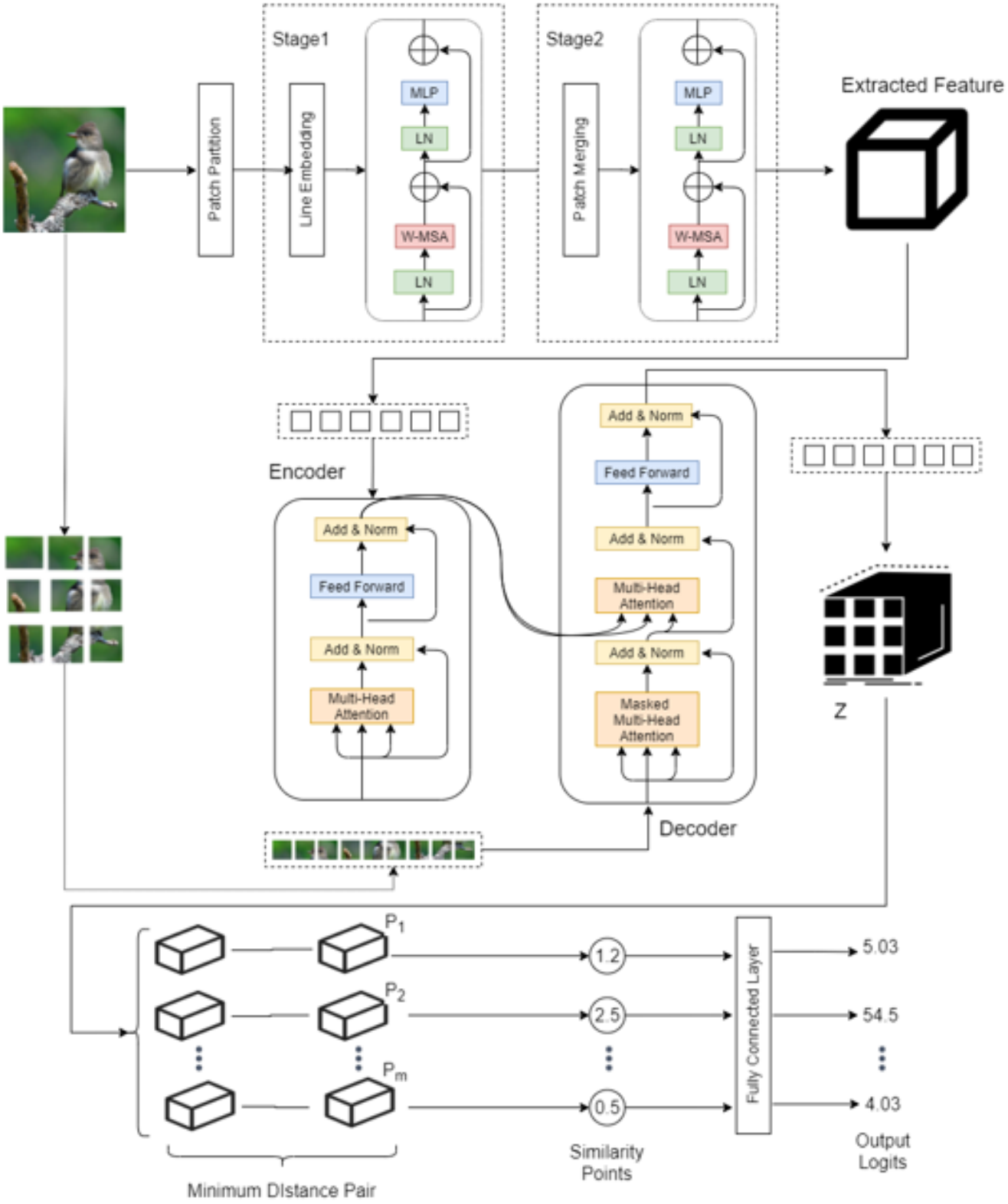


Figure 4.1: TPNet Architecture.

# Chapter 5

# Experiments and Results

## 5.1 Dataset

The dataset we used for our experiments is the Caltech-UCSD Birds dataset [36], which is a data set for bird species identification. This is also one of the datasets used by authors of ProtoPNet[7] for training their network. It contains 200 different classes with approximately 60 images per class. The data set provides a train-test split, leading to train with 5994 images and test with 5794 images.

To preprocess the training data, we apply the same data processing techniques as described in the original work[7]. We crop the images according to the bounding boxes provided with the data set and also further transformations, including rotation, distortion, shearing, and horizontal flipping. All the images are then resized to 224 x 224.

## 5.2 Model Parameters

When forwarding the resized images through the backbone, the input image dimensions, $H = W = 224$ and $C = 3$, are transformed to the output dimensions $H_1 = 7$, $W_1 = 7$ and $C_1$ 256. Dimension $C_1$ is a hyper-parameter that determines the number of channels for the network output and the prototypes. We use 10 prototypes per class, leading to 2,000 prototypes in total. We conducted several experiments by varying the number of Transformer layers from 1 - 6. However, no significant impact of varying the layers was observed. So we decided to keep it stable at one layer. For both the Transformer Encoder and the Decoder, 8 attention heads were used.

## 5.3 Experimental Setup

The GPU we used for this experiment is NVIDIA CUDA Version 10.2. We evaluate and compare transformer-based interpretable models with the CNN based ProtoPNet to understand the effect of Transformers on the following:

1. Classification Performance versus resource utilization tradeoff.
2. Interpretability

### 5.3.1 Classification Performance versus resource utilization tradeoff

To evaluate the classification performance, we measure the accuracy of all the models after training them all for 100 epochs. To verify if Transformer based architectures actually reduce the resource consumption of the model we record the following:

1. CPU percent (a float indicating system wide performance measured using the python package 'psutil')
2. GPU Percentage (as shown by nvidia-watch-smi)

### 5.3.2 Interpretability

Ideally, our explanations are validated by comparing them to some ground truth. However, since we are opening up a "black box," this ground truth is not available. We, therefore, qualitatively analyze a selection of local explanations. We also compare the interpretability of our purely transformer-based network interpretable network with that of ProtoPNet to understand what impact the transformer has on interpretability.

## 5.4 Results and Discussion

### 5.4.1 Performance

After training the ProtoPNet with resnet18 as the backbone for 100 epochs, we recorded a test accuracy of $64.24\%$. After training the ProtoPNet with swinT as the backbone for 100 epochs, we recorded a test accuracy of $62.08\%$. After training our model with swinT as the backbone for 100 epochs, we got a test accuracy of $55.55\%$. However, when we used a pre-trained Resnet50 alone as

a classifier, it was only 22% accurate. These results are summarized in the table 5.1.

Table 5.1: Performance Comparison of our model against Baseline.

| Classification Method | Backbone | Training Epochs | Accuracy |
|---|---|---|---|
| ProtoPNet | resnet18 | 100 | 64.24% |
| Swin-ProtoPNet | resnet18 | 100 | 62.08% |
| TPNet (ours) | swinT | 100 | 55.55% |
| Pre-trained ResNet50 | None | 0 | 22% |

Furthermore, Table 5.2 shows that the precision, recall, and f1 score of a purely transformer-based prototype classifier network (Swin-ProtoPNet) is similar to that of the ProtoPNet.

Table 5.2: Classification Performance of a transformer-based classifier versus a CNN-based classifier

| Classification Method | Precision | Recall | F1 Score |
|---|---|---|---|
| ProtoPNet | 0.6699 | 0.6387 | 0.6539 |
| Swin-ProtoPNet | 0.6207 | 0.6207 | 0.6207 |

### 5.4.2 Resource utilization

Our training method uses both the CPU and the GPU, so to gauge the actual resource consumption, we need to factor in both the CPU and GPU usage. As seen in Figure 5.1, It is clear that both the transformer-based interpretable models use lesser system-wide CPU resources than ProtoPNet. CPU percent is 6.2 for Swin-ProtoPNet and 4 for TPNet. In comparison, it is 7.3 for ProtoPNet.

Furthermore, Figure 5.2 shows that Transformer based interpretable networks consume less GPU resources than ProtoPNet. Swin-ProtoPNet uses 70% and TPNEt uses 83% while ProtoPNet uses 87%.

From the results shown in Figure 5.1 and Figure 5.2, it is evident that transformer does free us from the restriction of "depth" that conventional CNNs imposed and thereby reduces the intensiveness of the model.

### 5.4.3 Interpretability:

We saw that the addition of Transformers reduces the intensiveness of the model without hurting the performance. However, they alter the interpretability of the model somewhat negatively. To evaluate this, we visualize the reasoning

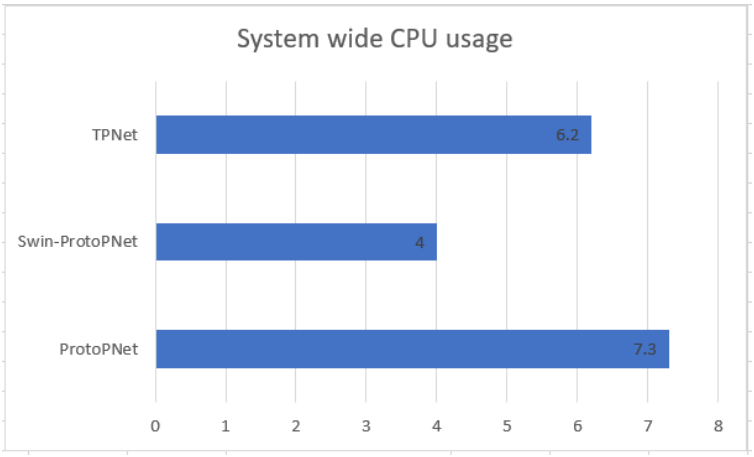


Figure 5.1: System wide CPU usage by all three models

process of ProtoPNet using a local example from the dataset. This visualization is similar to that shown by the authors of ProtoPNet [7] which shows the breakdown of the decision made by their model. To ensure a fair comparison, we use the same local example and make a similar visualization for TPNet. Figure 5.3 shows the reasoning process of ProtoPNet. As seen, this model draws fine-grained comparisons between prototypes across the dataset and the given image. In doing so, it identifies that parts of the image that are important to the decision making, and this importance is indicated by similarity scores. Suppose the total points calculated for a particular class are the highest, then that class is predicted. This map of interpretability is humanly understandable in a way that the user knows which parts were responsible for the decision. Figure 5.4 shows the reasoning process of TPNet. As seen, this model picks up whole images across the training data as prototypes. Although it arrives at the same decision as ProtoPNet did, there are marked differences in the activation maps of Figure 5.4 as compared to the activation maps in 5.3. The activation maps of Figure 5.4 are uniform and do not suggest the strong presence of any prototypical part. This means that the $L_2$ distances between each patch of the decoder output and each prototype were more or less similar. This pattern of receiving uniform activation maps is common for both the Transformer-based methods, which in turn suggests that we might need better distance metrics that can capture the distance between the patches of the decoder output and each prototype.

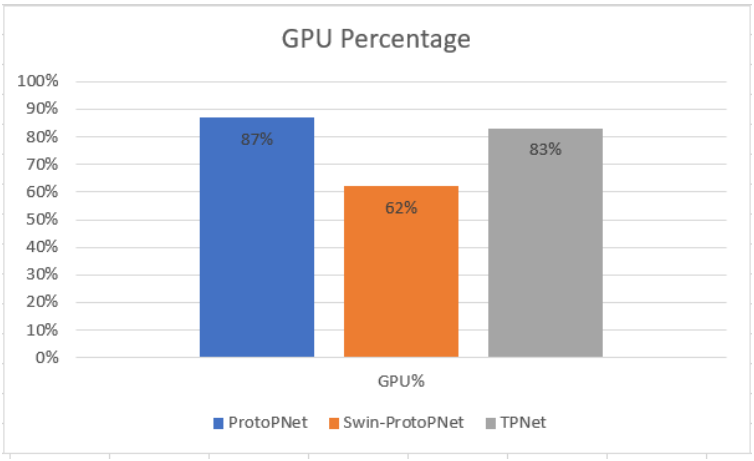


Figure 5.2: GPU Percentage used by all three models

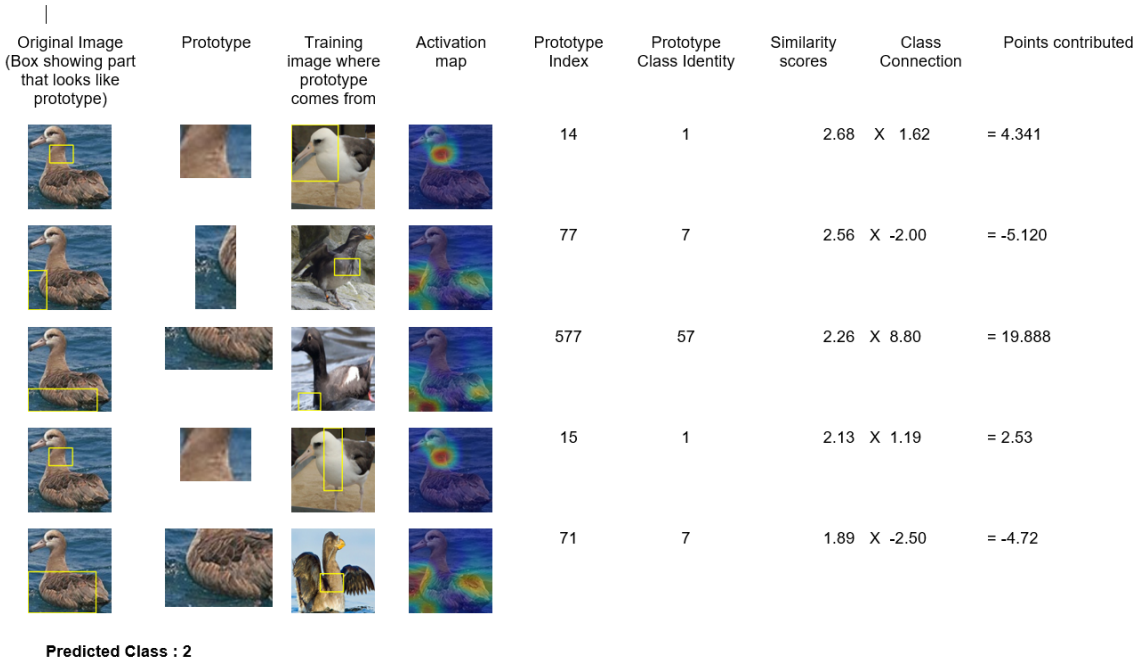


Figure 5.3: Reasoning Process of ProtoPNet

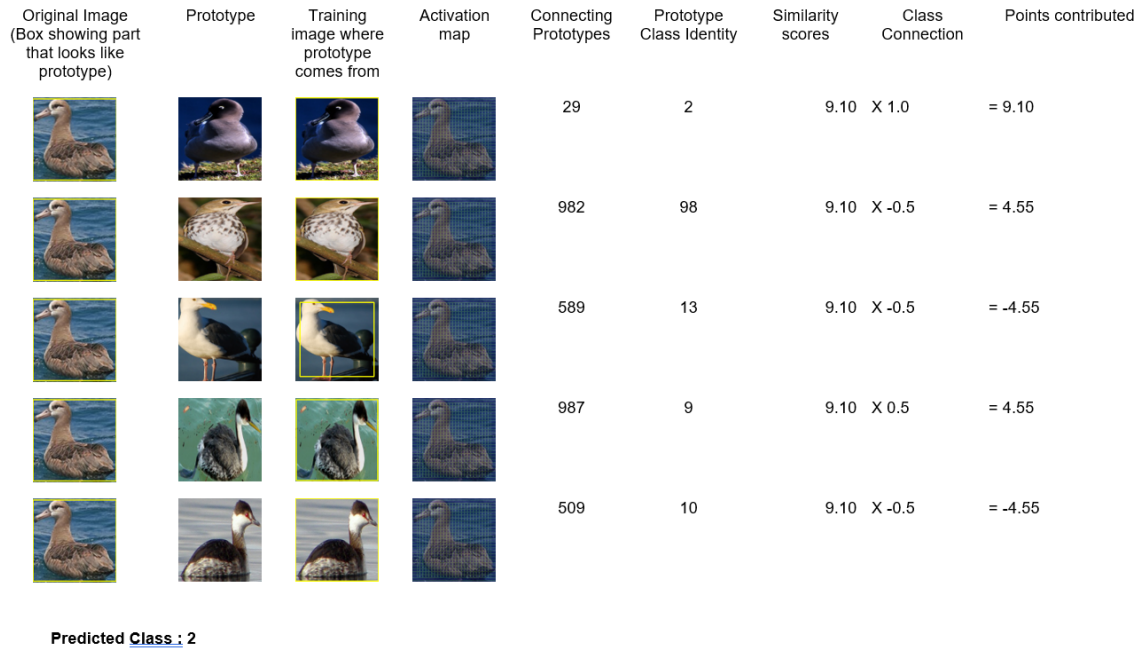


Figure 5.4: Reasoning Process of TPNet

# Chapter 6

# Conclusion

## 6.1 Conclusion

We addressed the limitations of ProtoPNet, such as depth restriction due to the use of CNNs, and excessive prototypes. We also demonstrated how the use of transformers in prototype learning could liberate us from the restriction of depth, thereby making the model less intensive. However, we discovered that simply the inclusion of one transformer block leads to the problem of receiving uniform activation maps that negatively impacts interpretability.

The solution to this could be exploring distance metrics other than $L_2$ distance that can capture the difference between the patches of the decoder output and each prototype.

## 6.2 Future Directions

We've identified the following improvements in the current architecture that might make ProtoPNet more accurate while still maintaining its interpretability:

1. We can use the Hungarian algorithm [16] to uniquely match the patches of extracted features with prototypes instead of distance maps. This might help avoid redundant prototypes. (More details are provided in Appendix A)

2. Alternatively, we could use a better distance metric to compare the patches of extracted features with the prototypes such as Wasserstein's distance. [32]

# Appendix A

# Appendix A

In the "Conclusions" chapter, we proposed an improvement over the current network architecture where we can use the Hungarian algorithm [16] to uniquely the decoder output patches with prototypes instead of distance maps which might help avoid the redundant prototypes. The details of such an architecture are laid down in the following section.

## A.1 A modified Architecture

Fig. A.1 gives an overview of the alternative architecture that uses the Hungarian algorithm in our network. As shown before this Network too combines the power of an attention mechanism along with the interpret-ability of prototypes. Our modified network still consists 3 main components: a CNN backbone to extract a compact feature representation, an encoder transformer, a prototype classifier network.

- **Feature extraction:** We begin with a tensor $\mathbf{x} \in \mathbb{R}^{3\times H_0\times W_0}$ representing an RGB image. We then use a CNN $f_\theta : \mathbb{R}^{3\times H_0\times W_0} \to \mathbb{R}^{C\times H\times W}$ (such as ResNet or VGG) to extract feature vectors from the given images. To work with a decent number of features, we reduce the channel dimension of $f_\theta(\mathbf{x})$ by adding two additional $1 \times 1$ convolutional layers (initialized using Kaiming uniform initialization [HZRS15]), creating a new feature map $\mathbf{z}_0 \in \mathbb{R}^{d\times H\times W}$. We use the RELU activation function for all our convolution layers except the last one, for which we use the sigmoid activation function.

- **Encoder Transformer:** Since the encoder expects a sequence as an input, we collapse the spatial dimensions of $\mathbf{z}_0$ and create a feature map $\mathbf{z}$

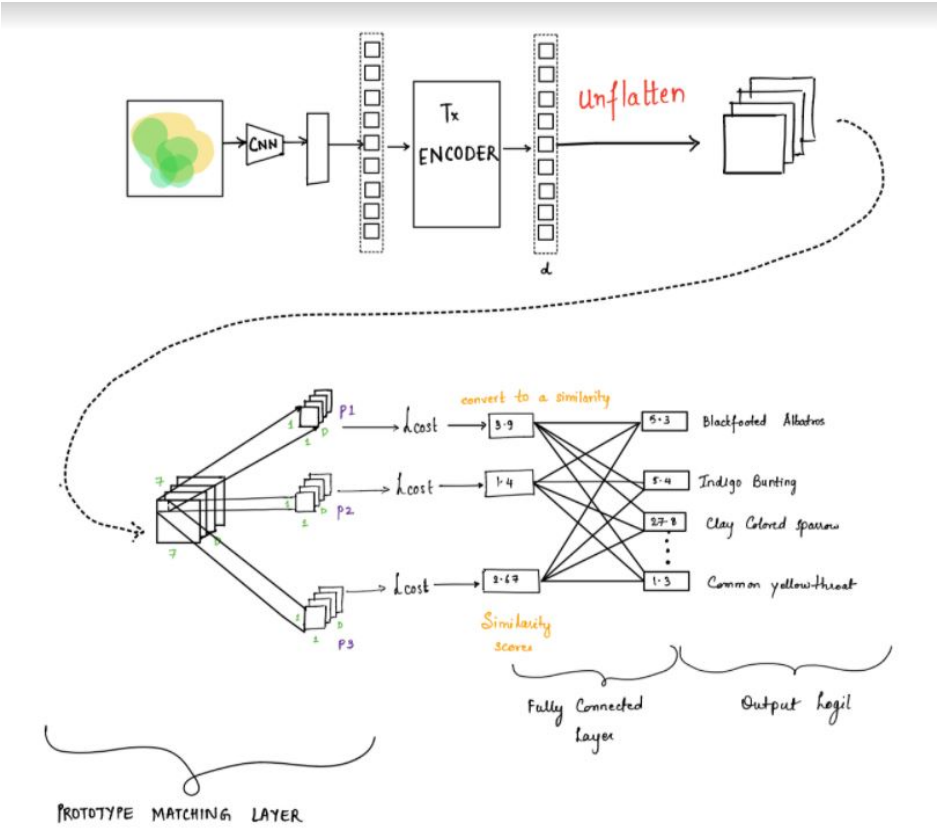


Figure A.1: Modified Architecture with Hungarian matcher

of size $d \times HW$. $\mathbf{z}$ is viewed as a sequence of $d$ features, each of which has the length of $HW$.

The self-attention mechanism in the transformer encoder learns the global pairwise interactions between $d$ feature maps and transforms the input into a new $d \times HW$ sequence of embeddings $\mathbf{z}'$ where each element in the sequence is a *weighted* combination of (transformed) input elements. Each element in the transformer encoder output $\mathbf{z}'$ represents an embedding of size $d$ and there are $H \times W$ such embeddings. We unflatten this output and reshape it back to its 3 dimensional size $\mathbf{z}' : \mathbb{R}^{d \times HW} \rightarrow \mathbf{z}'_\theta : \mathbb{R}^{d \times H \times W}$ for further processing.

- **Prototype Classifier Network:** This portion of the network is responsible for handling the "additional" task of learning *prototypes*, a set of representative observations in the dataset, and the classification is based on the similarity between the given input and each prototype. Our network learns $m$ prototype vectors $\mathbf{p}_1, \ldots, \mathbf{p}_m \in \mathbb{R}^d$ whose shape is $H_1 \times W_1 \times d$ where $H_1 < H$ and $W_1 < W$. The depth of each prototype is the same as that of the reshaped transformer output $\mathbf{z}'_\theta$, but the height and width of the prototypes are smaller than $H$ and $W$. This gives the network the ability to make fine-grained comparisons between parts of images. In other words, each prototype represents some prototypical activation pattern in a patch of $\mathbf{z}'_\theta$ which will, in turn, represent a patch in the original pixel space. With this in mind, consider two sets Z and P. Z denotes the set of patches of $\mathbf{z}'_\theta$ while P denotes the set of prototypes padded with $\phi$ (no class).

  - **Hungarian unbalanced weighted assignment problem:** We treat the problem of matching transformer output patches with prototypes as an unbalanced weighted assignment problem since the number of patches and the number of prototypes are not equal. Hungarian algorithm provides a way for solving such unbalanced weighted assignment problem by adding dummy rows or columns with cost 0, thereby making it a square matrix. Fig. A.2 shows the Qualification matrix that represents L2 distances between each patch of the transformer output and each prototype. Eventually, the Hungarian algorithm yields an optimal solution of one assignment between each patch of the transformer output and its corresponding matched prototype with an optimal matching cost (L2 distance in our case). Once we have the optimal assignment, we convert each distance/ cost to a similarity score. To convert dis-

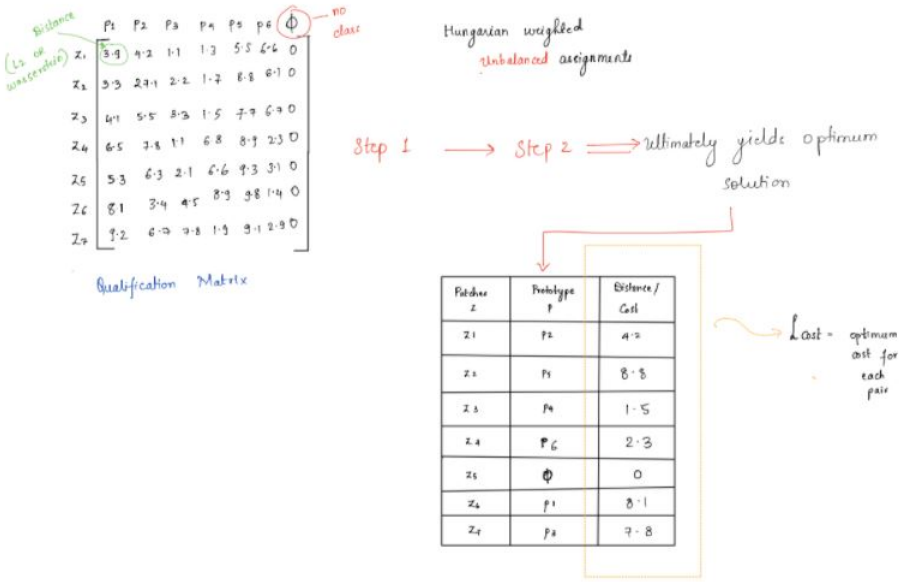


Figure A.2: Hungarian unbalanced weighted assignment between prototypes and transformer output patch

tance measure to similarity measure, we need to first normalize the L2 distance l to [0 1], by using $\mathbf{l_{norm} = l/max(l)}$. Then the similarity measure s is given by: $\mathbf{s = 1 - l_{norm}}$. where s is in the range [0 1], with 1 denotes highest similarity (the items in comparison are identical), and 0 denotes lowest similarity (largest distance).

Once we have a similarity score for each matched pair representing how strongly a prototypical part is present in some patch of the input image, the m similarity scores are multiplied by the weight matrix $w_h$ in the fully connected layer following the prototype layer to produce the output logits. These output logits are then normalized using softmax to deliver predicted probabilities for a given image belonging to various classes. The algorithm shown in Algorithm 1 demonstrates the entire training process divided into three stages. In stage 1, the optimization problem we aim to solve is =:

$$L(D;\theta) = CE(D;\theta) + \lambda_c L_{clus}(D;\theta) + \lambda_d L_{sep}(D;\theta)$$

where $CE(.), L_{clus}(.), and L_{sep}(.)$ are the cross-entropy loss, clustering loss, and separation loss, respectively. Here, $\theta$ are the trainable parameters

for the network. And $f(x_i)$, now denote the reshaped output of the Transformer Encoder.

This optimization is convex because we fix all the parameters from the convolution and prototype layers.

# Bibliography


[1] Rajiv Khanna Been Kim and Sanmi Koyejo. "Examples are not Enough, Learn to Criticize! Criticism for Interpretability". In: *Advances in Neural Information Processing Systems*. 2016.

[2] I. Biederman. "Recognition-by-Components: A Theory of Human Image Understanding". In: *Psychological Review* 94 (1987), pp. 115–147.

[3] Irving Biederman. "Recognition-by-components: A theory of human image understanding". In: *Psychological Review* 94 (1987), pp. 115–147.

[4] Jacob Bien and Robert Tibshirani. "Prototype selection for interpretable classification". In: *The Annals of Applied Statistics* (2011), pp. 2403–2424.

[5] Hu Cao et al. *Swin-Unet: Unet-like Pure Transformer for Medical Image Segmentation*. 2021. arXiv: 2105.05537 [eess.IV].

[6] Nicolas Carion et al. *End-to-End Object Detection with Transformers*. 2020. arXiv: 2005.12872 [cs.CV].

[7] Chaofan Chen et al. "This looks like that: deep learning for interpretable image recognition". In: *Advances in neural information processing systems*. 2019, pp. 8930–8941.

[8] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=YicbFdNTTy.

[9] Mengnan Du, Ninghao Liu, and Xia Hu. "Techniques for Interpretable Machine Learning". In: *CoRR* abs/1808.00033 (2018). arXiv: 1808.00033. URL: http://arxiv.org/abs/1808.00033.

[10] Jerome H. Friedman and Bogdan E. Popescu. "Predictive learning via rule ensembles". In: *The Annals of Applied Statistics* 2.3 (Sept. 2008). ISSN: 1932-6157. DOI: 10.1214/07-aoas148. URL: http://dx.doi.org/10.1214/07-AOAS148.

[11] Kai Han et al. *A Survey on Visual Transformer*. 2021. arXiv: 2012.12556 [cs.CV].

[12] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[13] Been Kim, Cynthia Rudin, and Julie A Shah. "The bayesian case model: A generative approach for case-based reasoning and prototype classification". In: *Advances in neural information processing systems*. 2014, pp. 1952–1960.

[14] Janet L Kolodner. "An introduction to case-based reasoning". In: *Artificial intelligence review* 6.1 (1992), pp. 3–34.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[16] H. Kuhn. "The Hungarian Method for the Assignment Problem". In: *Naval Research Logistic Quarterly* 2 (May 2012).

[17] Yann LeCun et al. "Handwritten Digit Recognition with a Back-Propagation Network". In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1990. URL: https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

[18] O. Li et al. "Deep Learning for Case-based Reasoning through Prototypes: A Neural Network that Explains its Predictions". In: *AAAI*. 2018.

[19] Oscar Li et al. *Deep Learning for Case-Based Reasoning through Prototypes: A Neural Network that Explains Its Predictions*. 2017. arXiv: 1710.04806 [cs.AI].

[20] Ze Liu et al. "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows". In: *CoRR* abs/2103.14030 (2021). arXiv: 2103.14030. URL: https://arxiv.org/abs/2103.14030.

[21] Ze Liu et al. "Video Swin Transformer". In: *CoRR* abs/2106.13230 (2021). arXiv: 2106.13230. URL: https://arxiv.org/abs/2106.13230.

[22] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. *Interpretable Machine Learning – A Brief History, State-of-the-Art and Challenges*. 2020. arXiv: 2010.09337 [stat.ML].

[23] Meike Nauta, Ron van Bree, and Christin Seifert. *Neural Prototype Trees for Interpretable Fine-grained Image Recognition*. 2020. arXiv: 2012.02046 [cs.CV].

[24] Meike Nauta et al. *This Looks Like That, Because ... Explaining Prototypes for Interpretable Image Recognition*. 2020. arXiv: 2011.02863 [cs.CV].

[25] Carey E Priebe et al. "Classification using class cover catch digraphs". In: *Journal of classification* 20.1 (2003), pp. 003–023.

[26] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *CoRR* abs/1602.04938 (2016). arXiv: 1602.04938. URL: http://arxiv.org/abs/1602.04938.

[27] Cynthia Rudin. *Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead*. 2019. arXiv: 1811.10154 [stat.ML].

[28] Dawid Rymarczyk et al. "ProtoPShare: Prototype Sharing for Interpretable Image Classification and Similarity Discovery". In: *CoRR* abs/2011.14340 (2020). arXiv: 2011.14340. URL: https://arxiv.org/abs/2011.14340.

[29] Alex Sherstinsky. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network". In: *Physica D: Nonlinear Phenomena* 404 (Mar. 2020), p. 132306. ISSN: 0167-2789. DOI: 10.1016/j.physd.2019.132306. URL: http://dx.doi.org/10.1016/j.physd.2019.132306.

[30] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. *Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps*. 2014. arXiv: 1312.6034 [cs.CV].

[31] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[32] Michael Snow and Jan Van lent. *Monge's Optimal Transport Distance for Image Classification*. 2018. arXiv: 1612.00181 [cs.CV].

[33] Zhiqing Sun et al. *Rethinking Transformer-based Set Prediction for Object Detection*. 2020. arXiv: 2011.10881 [cs.CV].

[34] Loc Trinh et al. "Interpretable and Trustworthy Deepfake Detection via Dynamic Prototypes". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. Jan. 2021, pp. 1973–1983.

[35] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[36] C. Wah et al. *The Caltech-UCSD Birds-200-2011 Dataset*. Tech. rep. CNS-TR-2011-001. California Institute of Technology, 2011.

[37] Minghang Zheng et al. *End-to-End Object Detection with Adaptive Clustering Transformer*. 2020. arXiv: 2011.09315 [cs.CV].

[38] Xizhou Zhu et al. *Deformable DETR: Deformable Transformers for End-to-End Object Detection*. 2020. arXiv: 2010.04159 [cs.CV].