UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Luiz Gustavo da Rocha Charamba

Providing Projective and Affine Invariance for Recognition by Multi-Angle-Scale Vision Transformer

Recife

2025

Luiz Gustavo da Rocha Charamba

Providing Projective and Affine Invariance for Recognition by Multi-Angle-Scale Vision Transformer

Ph.D. Thesis presented to the Centro de Informática of the Universidade Federal de Pernambuco in partial fulfillment of the requirements for the degree of Doctorate of Computer Science.

Concentration Area: Media and Interaction

Supervisor: Silvio de Barros Melo

Co-Supervisor: Nivan Roberto Ferreira Júnior

Recife

2025

.Catalogação de Publicação na Fonte. UFPE - Biblioteca Central

Charamba, Luiz Gustavo da Rocha.
Providing Projective and Affine Invariance for Recognition by Multi-Angle-Scale Vision Transformer / Luiz Gustavo da Rocha Charamba. - Recife, 2025.
130f.: il.

Tese (Doutorado) - Universidade Federal de Pernambuco, Centro de Informática, Programa de Pós-Graduação em Ciência da Computação, 2025.
Orientação: Silvio de Barros Melo.
Coorientação: Nivan Roberto Ferreira Júnior.
Inclui Referências e Apêndices.

1. Affine Invariance; 2. Projective Invariance; 3. Geometric Deep Learning; 4. Vision Transformer; 5. Computer Vision. I. Melo, Silvio de Barros. II. Ferreira Júnior, Nivan Roberto. III. Título.

UFPE-Biblioteca Central

Serviço Público Federal
Universidade Federal de Pernambuco
Centro de Informática
Pós-Graduação em Ciência da Computação

A tese de doutorado intitulada **"Providing Projective and Affine Invariance for Recognition by Multi-Angle-Scale Vision Transformer**", apresentada por **Luiz Gustavo da Rocha Charamba** em 28/08/2025, foi aprovada pela Banca Examinadora. Como exigência regimental para a obtenção do grau de doutor, o aluno deverá entregar a versão final do trabalho com as devidas correções propostas pelos examinadores, no prazo de 90 dias.

Recife, 28 de agosto de 2025

_____
Prof. Dr. Rafael Dueire Lins
Centro de Informática / UFPE

_____
Prof. Dr. Ricardo Martins de Abreu Silva
Centro de Informática / UFPE

_____
Prof. Dr. Manuel Menezes de Oliveira Neto
Instituto de Informática / UFRGS

_____
Prof. Dr. Sérgio de Carvalho Bezerra
Departamento de Computação Científica / UFPB

_____
Prof. Dr. João Marcelo Xavier Natário Teixeira
Departamento de Eletrônica e Sistemas / UFPE

As correções solicitadas na defesa foram aprovadas pelo presidente da banca em:

Recife, 14 de novembro de 2025.

Dedico este trabalho à minha filha Aurora, à minha esposa Raissa, e à minha mãe Maria das Graças.

AGRADECIMENTOS

Este trabalho só foi possível graças à ajuda de Deus e ao apoio de pessoas pelas quais serei eternamente grato.

Agradeço profundamente à minha amada esposa, Raissa, pelo amor, compreensão e constante apoio, especialmente nos momentos mais difíceis. Agradeço também por ter me presenteado com nossa amada filha, Aurora, que veio ao mundo neste ano para iluminar nossas vidas. Sua chegada tornou ainda mais especial o momento de defender esta tese, permitindo-me viver a alegria de compartilhar essa conquista com a sua presença graciosa.

À minha querida mãe, Maria das Graças da Rocha, expresso minha eterna gratidão por sua sabedoria, paciência e amor incondicional. Foi ela quem me ensinou a enfrentar os desafios da vida com coragem e fé, sempre torcendo por mim em cada etapa do caminho. Sem o seu exemplo e dedicação, nada disso teria sido possível.

Sou também profundamente grato ao meu orientador, professor Sílvio Melo, por sua confiança, orientação e inspiração ao longo desta jornada. Sua competência e humanidade despertaram em mim grande admiração e respeito, e sinto-me honrado por ter sido seu aluno de doutorado. Agradeço igualmente ao meu coorientador, professor Nivan Ferreira, pelas valiosas sugestões, críticas construtivas e contribuições que ajudaram a aprimorar este trabalho.

Aos meus amigos próximos e familiares, que sempre acreditaram em mim e me incentivaram com palavras de apoio e gestos de carinho, deixo meus sinceros agradecimentos.

Por fim, estendo minha gratidão a todas as pessoas que, de alguma forma, contribuíram para a realização desta tese, seja com palavras de incentivo, colaboração acadêmica ou simples gestos de amizade e solidariedade.

"All models are wrong, but some are useful." (BOX, 1976)

RESUMO

O reconhecimento de formas planares deformadas encontra aplicações em muitas áreas não relacionadas, tais como *marketing*, OCR e veículos autônomos. Um grande esforço tem sido dedicado a esse tema na literatura, baseado em abordagens geométricas diretas, embora com resultados ou desempenho limitados. Mais recentemente, várias abordagens de aprendizado de máquina foram propostas, mas com resultados satisfatórios apenas quando a deformação é, no máximo, uma transformação afim fraca. Esta tese apresenta o *Multi-Angle-Scale Vision Transformer*, MASViT, uma solução baseada em aprendizado profundo que supera os métodos do estado da arte no reconhecimento de imagens deformadas por afinidades e projetividades. Um ponto crucial em nossa proposta é a ausência de imagens deformadas durante a fase de treinamento. Nossa abordagem emprega filtros convolucionais 1D correspondentes a linhas retas que cruzam a forma no domínio polar, preservando a colinearidade, um invariante projetivo fundamental. As sequências angulares derivadas do domínio polar integram-se bem à arquitetura *Vision Transformer* (ViT), pois esses *patch embeddings* são geometricamente coerentes, aumentando a adequação ao codificador do *transformer*. Também introduzimos diversas técnicas de regularização para ampliar a capacidade de generalização do modelo. Para validar a abordagem, nós organizamos novos conjuntos de teste derivados do *German Traffic Sign Recognition Benchmark* (GTSRB). Por meio de extensos experimentos, demonstramos que essa abordagem supera os modelos do estado da arte, especialmente em cenários envolvendo imagens submetidas a severas deformações afins e projetivas.

Palavras-chaves: Invariância Afim; Invariância Projetiva; Aprendizagem Profunda Geométrica; Transformador de Visão; Visão Computacional.

ABSTRACT

The recognition of deformed planar shapes finds applications in many unrelated areas, such as marketing, OCR, and autonomous vehicles. An enormous effort has been devoted to this in the literature, based on direct geometric approaches, although with limited results or performance. More recently, many machine learning approaches have been proposed with satisfactory results only when the deformation is a weak affine at best. This thesis introduces the Multi-Angle-Scale Vision Transformer, MASViT, a deep-learning-based solution that outperforms state of the art methods in the recognition of affinely and projectively deformed images. A crucial point in our setting is the absence of deformed images during training phase. Our approach employs 1D convolutional filters corresponding to straight lines crossing the shape in the polar domain, preserving collinearity, a basic projective invariant. Angular sequences deriving from the polar domain integrate well with the Vision Transformer (ViT) architecture, as these patch embeddings are geometrically coherent, enhancing suitability for the transformer encoder. We also introduce several regularization techniques to boost the generalizability of model. To validate the approach, we curated new test datasets derived from the German Traffic Sign Recognition Benchmark (GTSRB). Through extensive experiments, we demonstrate that this approach surpasses state-of-the-art models, particularly when dealing with images subjected to severe affine and projective deformations.

Keywords: Affine Invariance; Projective Invariance; Geometric Deep Learning; Vision Transformer; Computer Vision.

SUMÁRIO

1 INTRODUCTION

In image recognition tasks, objects can suffer from numerous types of perturbation, including deformations due to geometric transformations that are quite common in everyday human visual perception and in the field of computer vision, which tries to mimic the successful properties of human vision to some extent. One of the most fundamental needs of computer vision applications is to be robust to the types of transformations produced by most cameras, such as rotations, scales, affinities and projectivities.

When it comes to using neural networks for image recognition tasks, this robustness is usually achieved by training models with large volumes of data with naturally deformed samples or by data augmentation. However, there is often a shortage of data or a need to explain the model.

Deep learning models with translation invariance already exist, such as CNNs, where convolutional filters operate on the image pixels in different positions; in essence, these operations are equivariant in the sense that translating the input values and then convolving is the same as first convolving and then translating the response. Subsampling operations, such as pooling, transform this equivariance into invariance, so that translating the input signal does not change the response. Due to this power of invariance to translations, these architectures have become the industry standard for computer vision applications.

Geometric Deep Learning is a subfield of deep learning that studies neural networks capable of operating in domains with complex geometric structures, such as graphs, surfaces, and symmetry groups (BRONSTEIN et al., 2021). In this context, symmetry refers to properties that remain invariant to certain transformations in an object. Exploiting these principles to design models that respect the geometry of the data, going beyond regular structures such as images and sequences, going beyond the traditional way of representing these data. Instead of relying exclusively on data augmentation, Geometric Deep Learning incorporates these symmetries directly into the architecture, creating networks that are intrinsically invariant or equivariant to various transformations. This results in more robust models, with better generalization, less need for data augmentation, and greater fidelity to the structure of the problem - in many cases surpassing traditional approaches such as CNNs, which are invariant only to translation.

1.1 MOTIVATION

Image recognition is a problem in computer vision that has received a lot of attention from the research community due to its application in areas such as traffic signs recognition for autonomous vehicles (HOUBEN et al., 2013), automatic extraction and recognition of logos (BIANCO et al., 2017), pose estimation for augmented reality (KENDALL; GRIMES; CIPOLLA, 2015), among others. One common challenge in this context is to effectively perform recognition of objects when the target objects have been distorted from camera projections. These projections can significantly alter an image's appearance and pose challenges for standard recognition systems. For these reasons, building image recognition models that are robust to different classes of projective transformations is an important research problem.

Convolutional Neural Networks (CNNs) are famous for their translation invariance, however, traditional CNNs often struggle when faced with affinely distorted images (LENC; VEDALDI, 2015). So, although CNN architectures are invariant to translations, many perturbations suffered by objects in the real world go far beyond simple translations.

More recent deep learning models based on Capsule Nets exhibited superior accuracy on affinely deformed images of the affNIST dataset (SABOUR; FROSST; HINTON, 2017; NETZER et al., 2011), being trained on the non-deformed images of the MNIST dataset (LECUN et al., 1998), compared to a CNN with a similar number of parameters. Since then, this and other architectures have advanced to provide geometric invariance for deep models, particularly on the MNIST family of datasets.

1.2 OBJECTIVE

Motivated by the need to enhance the robustness of deep architectures against image distortions in computer vision tasks, this work proposes the development of a solution based on a projectively invariant neural network. Since projective transformations are among the most general forms of geometric transformations, a model invariant to them is also inherently invariant to translations, rotations, scaling, and affine transformations. Furthermore, the approach aims to reduce the reliance on data augmentation techniques involving geometric transformations.

1.3 CONTRIBUTIONS

This thesis offers the following major contributions:

1. **New Vision Transformer architecture is proposed:** In this work, we introduced the Multi-Angle-Scale Vision Transformer (MASViT), a deep-learning-based solution that is robust in recognizing images distorted by projective transformations, being **trained on undeformed ones**.

2. **Extracts projective-invariant features using 1D convolutions in the polar domain:** MASViT is a new Vision Transformer (ViT) model that takes as input an image in the polar domain, processes it satisfying the collinearity invariance by using 1D convolutional filters, and passes a sequence of angular patch embeddings to the Transformer Encoder.

3. **Proposal of regularization techniques adapted to the polar domain:** Adapted regularization techniques were used, referred to as:

   - *Cyclic-angular shift:* Which aids in rotation invariance;
   - *Right-side padding:* Which helps in scaling invariance;
   - *Angle dropout:* Which is a dropout layer that randomly inactivates entire rows of the tensor data.

4. **Post-training enhancement mechanism:** MASViT allows for a post-trainning boost through the variation of the center of the polar domain conversion. This enables an improvement in the post-training accuracy. This boost mechanism allows improvements in the final accuracy without the need for complete retraining.

5. **Development of two new datasets for robustness assessment:** To evaluate our solution, we first develop two new datasets based on the German Traffic Sign Recognition Benchmark (GTSRB) (STALLKAMP et al., 2012). These datasets, named aff-GTSRB and proj-GTRSB, include images created from the original ones by applying these deformations (affine and projective, respectively) with different levels of severity. These datasets simulate realistic conditions commonly encountered in applications like traffic signal recognition for autonomous vehicles and OCR.

By incorporating varying levels of deformation severity, they provide a comprehensive framework for evaluating robust recognition models, reflecting scenarios where cameras and planar objects interact at diverse and challenging viewing angles in real-world situations.

6. **Comparative experimental evaluation with state-of-the-art methods:** We have compared MASViT with two other state-of-the-art GTSRB recognition models based on Spatial Transformers (ARCOS-GARCÍA; ALVAREZ-GARCIA; SORIA-MORILLO, 2018) and Capsule networks (CHEN et al., 2024), which their architectures are designed for robust recognition of distorted shapes. Our experiments show that MASViT significantly outperforms the previously mentioned methods when considering images that suffered affine and projective transformations.

1.4 SCIENTIFIC PRODUCTION

During this doctorate, it was possible to publish a related work, which consists of the development of a planar shape descriptor based on cross-ratio arrays for recognition tasks robust to severe projective deformation and occlusions. This work was published in 2021 in the journal *Computers & Graphics* (CHARAMBA; MELO; LIMA, 2021). A second paper, more closely related to the development of MASViT, was also submitted to the journal *Neural Computing & Applications*, the article has been accepted and is currently in the publication process. The articles developed during this doctorate are briefly summarized in the following.

1. Luiz G. Charamba, Silvio Melo, Ullayne de Lima, **Cross ratio arrays: A descriptor invariant to severe projective deformation and robust to occlusion for planar shape recognition**, *Computers & Graphics*, Elsevier, Volume 100, Pag. 54-65, 2021, ISSN 0097-8493, https://doi.org/10.1016/j.cag.2021.08.001.

2. Luiz G. Charamba, Nivan Ferreira, Silvio Melo, **Providing Projective and Affine Invariance for Recognition by Multi-Angle-Scale Vision Transformer**, *Neural Computing & Applications*, Springer Nature, 2024 (Status: Accepted).

1.5 ORGANIZATION OF THE THESIS

The thesis is organized as follows:

Chapter 2: Presents the theoretical foundation necessary for the development of this work. The fundamental concepts of geometric transformations applied to images are covered; the principles of neural networks, from classical architectures to Transformers; neural network learning mechanisms; in addition to the main evaluation metrics used for classifiers.

Chapter 3: Reviews the works related to the problem addressed, with emphasis on approaches based on CNNs, capsule networks and spatial transformers.

Chapter 4: Describes the methodology proposed for the development of MASViT, including the network architecture, the new regularization techniques introduced in this work — *cyclic-angular shift*, *right-side padding* and *angle dropout* — as well as the post-training improvement mechanism based on the variation of the polar center by max score.

Chapter 5: Details the experiments performed, including the description of the datasets used for training and validation, the experimental configurations and hyperparameters adopted, as well as the results obtained.

Chapter 6: Presents the conclusions of the work, highlighting the contributions achieved, and proposes possible directions for future work.

2 THEORETICAL BASIS

2.1 GEOMETRIC TRANSFORMATIONS BACKGROUND

Geometric Deep Learning is the area dedicated to making deep learning models more robust to data variations by exploring their symmetries. Symmetry is a certain property of an object or system unchanged when subjected to a spatial transformation, which can be smooth, continuous, or discrete. Symmetries are prevalent in many machine-learning tasks. In computer vision, a category of objects may exhibit certain symmetries that can be explored in object recognition problems, and developing neural networks that can be invariant or equivariant under these transformations is a goal of geometric deep learning (BRONSTEIN et al., 2017; BRONSTEIN et al., 2021; GERKEN et al., 2023). Usually, the focus in this area is the set of deformations produced by conventional cameras called projectivities, i.e., projective transformations. The invariance to these transformations is associated with well-known quantities and properties such as distances, angles, areas, collinearity of points, and incidence relations.

The taxonomy of projective transformations is illustrated in Table 1. Planar projective transformations are non-singular linear mappings in homogeneous coordinates, i.e., may be represented by non-singular homogeneous 3×3 matrices. This table shows the hierarchy of projective transformations according to their mathematical group classification (the product/inverse of an element in a group is an element of that group), along with their degrees of freedom (DoF) and main invariants. The largest group, formed by all projectivities, can be parametrized with eight degrees of freedom (the nine scalars in a 3×3 matrix minus one due to the matrix homogeneity). Their invariants include concurrency of lines, collinearity of points, intersection and tangency between curves, inflections, discontinuities and cusps in curves, and cross-ratio (ratio of ratio of lengths). The most general projectivity can transform a square into an arbitrary quadrilateral.

An important subgroup of the projectivities is composed of the *affinities*, which are characterized by matrices whose last line possesses entries with values 0, 0, and 1 (or a nonzero scalar). They are parametrized with six degrees of freedom and their invariants include, besides those of general projectivities, parallelism of lines, ratio of areas and lengths measured in parallel lines. The most general affinity can transform a square into any parallelogram. A notorious subgroup of the affinities is that of the *similarities*, cha-

Tabela 1 – Taxonomy of projectivities: The hierarchy of projective transformations according to their mathematical group classification. They are represented by homogeneous matrices, and their degrees of freedom are indicated. Their invariants are represented in the table by the following letters: (a)-concurrency of lines, (b)-collinearity of points, (c)-intersection between curves, (d)-tangency between curves, (e)-inflections of curves, (f)-discontinuities of curves, (g)-cusps in curves, (h) cross-ratio, (i)-parallelism of lines, (j)-ratio of areas, (k)-ratio of lengths measured in parallel lines, (l)-angle, (m)-ratio of lengths, (n)-length, and (o)-area.

Group	**Matrix**	**DoF**	**Minimum invariant properties**
Isometry	$\begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$	3	a, b, c, d, e, f, g, h, i, j, k, l, m, n, o.
Similarity	$\begin{bmatrix} \sigma\cos\theta & -\sigma\sin\theta & t_x \\ \sigma\sin\theta & \sigma\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$	4	a, b, c, d, e, f, g, h, i, j, k, l, m.
Affinity	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$	6	a, b, c, d, e, f, g, h, i, j, k.
Projectivity	$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$	8	a, b, c, d, e, f, g, h.

racterized by matrices whose leading 2×2 block corresponds to a scaled rotation, with angle θ and scale factor σ, and the last column includes the terms of translation t_x and t_y. They are, therefore, parametrized by four parameters, and their invariants include, besides those of the affinities: angles and ratio of lengths in arbitrary directions. The most general similarity can transform a square into an arbitrarily magnified, contracted, or rotated square. The family of isometries is the most restrictive projective group, parametrized with just three degrees of freedom: the angle of rotation θ and the translation terms t_x and t_y. Their invariants include those of the similarities, together with length and area. An isometry is a rigid transformation, preserving an object's shape and size, and can transform a square into a square of the same size, possibly rotated and translated (HARTLEY; ZISSERMAN, 2003). Fig. 1 shows a STOP sign subjected by these mentioned geometric transformations, illustrating the resulting deformations.

From their hierarchy of invariants, we can say that if the model is capable of recognizing samples distorted by projectivities, it will also be robust in recognizing samples distorted

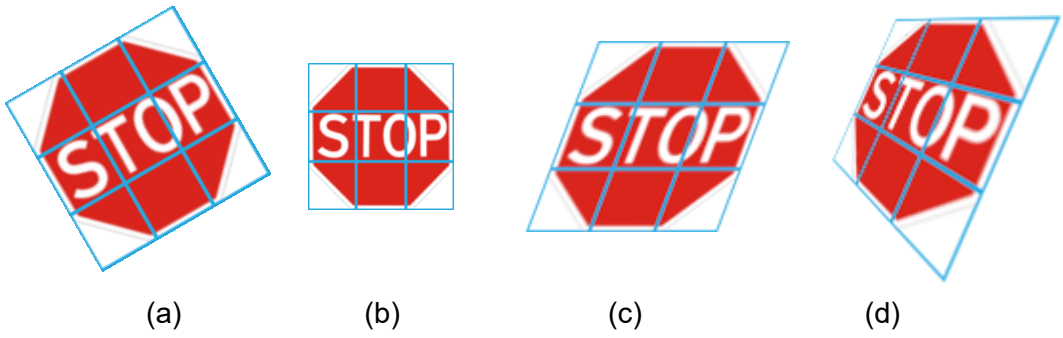


Figura 1 – A STOP sign undergoing four types of geometric transformations is illustrated: (a) **Rotation**, an isometric transformation, which preserves properties such as length and area; (b) **Scaling**, a similarity transformation, where proportions and angles remain intact; (c) **Shearing**, an affine transformation, which maintains parallelism of lines and preserves the ratio of lengths and ratio of areas; and (d) **Perspectivity**, a projective transformation, which preserves collinearity, concurrency, cross-ratio and other projective invariants. Source: The author.

by affinities, similarities, and isometries. The ability of deep learning models to handle different types of symmetries may depend on the nature of the geometric transformations inherent to the problem at hand.

2.2 ARTIFICIAL NEURAL NETWORKS

This section covers the theoretical foundations necessary to understand Artificial Neural Networks (ANN), including convolutional neural networks, also included are transformer architectures such as the Vision Transformer, both architectures are commonly used for image recognition applications.

An artificial neural network is a machine learning method used to solve regression, classification, and other problems, which presents a more simplified model based on the human brain structure and which obtains knowledge through learning.

Since 1943, a large number of models have been developed, some of which are quite realistic and of greater interest to neuroscientists. On the other hand, other scholars have delved deeper into the more abstract properties of neural networks, including distributed computing, tolerance to noisy inputs, and learning. (RUSSELL, 2010). Similar to biological neural networks, ANNs have processing units such as neurons, which have connections to other units, where they receive and send signals, just like in nerve synapses.

The simplified mathematical model of the neuron developed by (MCCULLOCH; PITTS,

1943) emits a signal at its output when a linear combination of its inputs exceeds some limits. This served as the basis for the Perceptron created in 1958 by (ROSENBLATT, 1958), a learning model consisting of sensory input units connected to a single layer of neurons. Rosenblatt stated that these networks can be trained to classify linearly separable patterns, functioning as a binary classifier. When dealing with non-linearly separable functions, the Perceptron cannot generate a hyperplane, and in the real world the data reported are linearly separable, making the Perceptron not very useful for problems of this nature. Figure 2 illustrates (a) sets of linearly separable patterns and (b) sets of non-linearly separable patterns.

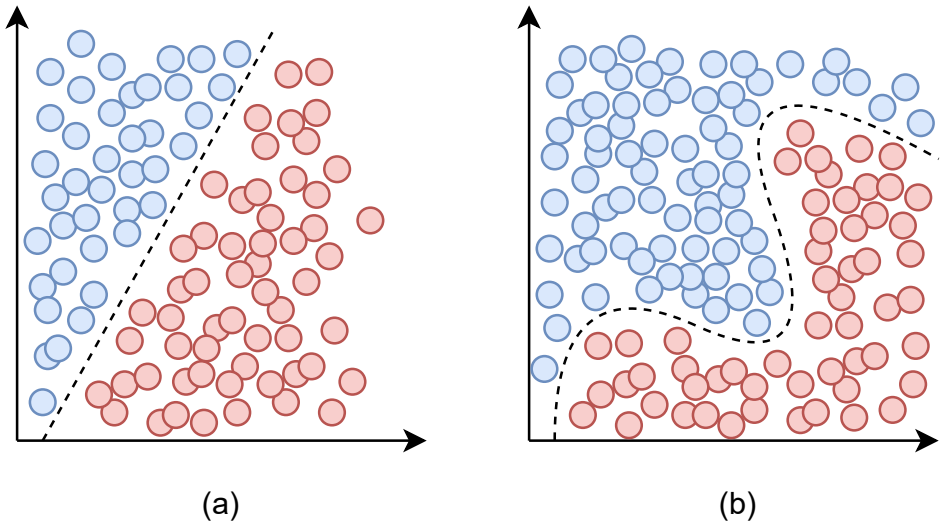


Figura 2 – Linearly separable pattern sets in **(a)** and non-linearly separable in **(b)**. The Perceptron is able to correctly separate the classes illustrated in **(a)**, but cannot do the same in **(b)**. Source: The author.

2.2.1 Multilayer Perceptron

With the need to make a neural network capable of solving non-linear problems, the architecture of the Multilayer Perceptron (MLP) emerged. This is a neural network similar to the Perceptron, but has more than one layer of neurons. In cases where it is not possible to separate the patterns in a single line, it is possible to separate the elements using more than one line with the use of the MLP.

2.2.1.1 Artificial Neuron

An artificial neuron is the basic processing unit of an artificial neural network, composed of input connections that contain synaptic weights and an output that can be understood as the inner product of an input vector x with a vector w, which represents the synaptic weights, added to a bias θ. In order for the output of this neuron to contain a non-linear behavior that has an activation threshold, imitating the behavior of the biological neuron, a non-linear function is included that makes a composition with the input and bias, which is called the activation function. Figure 3 illustrates this model.

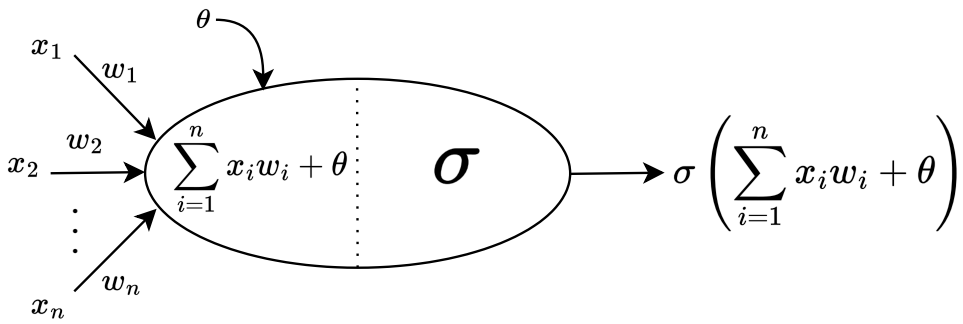


Figura 3 – Artificial neuron model: The neuron output signal is a composition of a nonlinear activation function σ with a sum of n inputs x_i multiplied by their respective synaptic weights w_i plus a bias θ. Source: The author.

2.2.1.2 Activation Functions

Activation functions determine the output of each neuron. Using linear functions as activation functions limits the power of ANNs to convex solutions; for non-convex solutions, non-linear functions are required. In other words, to learn complex patterns, it is necessary to use neurons that employ some type of non-linearity (BUDUMA; BUDUMA; PAPA, 2022).

The three main types of activation functions most commonly used in practice are:

1. Sigmoid
2. Hyperbolic Tangent
3. Rectified Linear

2.2.1.2.1 Sigmoid

The sigmoid activation function is expressed as follows:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2.1}$$

Intuitively, this means that when the sum of the weighted input values with the weights represented by the variable z is very small, the output of this function is very close to 0. When z is very large, the output is close to 1. The function takes the following S-shape, as shown in Figure 4.

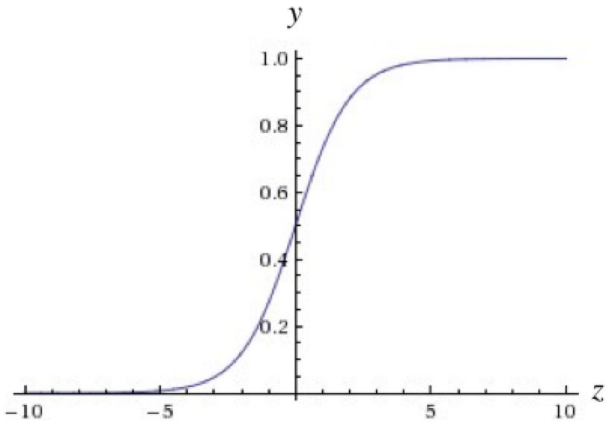


Figura 4 – Output of the sigmoid function as z varies. Source: Image taken from (BUDUMA; BUDUMA; PAPA, 2022).

2.2.1.2.2 Hyperbolic Tangent

The hyperbolic tangent activation function is expressed as follows in terms of z:

$$f(z) = Tanh(z) \tag{2.2}$$

The hyperbolic tangent (Tanh) activation function uses a type of nonlinearity very similar to the sigmoid, but instead of ranging from 0 to 1, the output ranges from -1 to 1, as illustrated in Figure 5. When S-shaped nonlinearities are used, the Tanh function is often preferred over the sigmoid function because it is centered at zero.

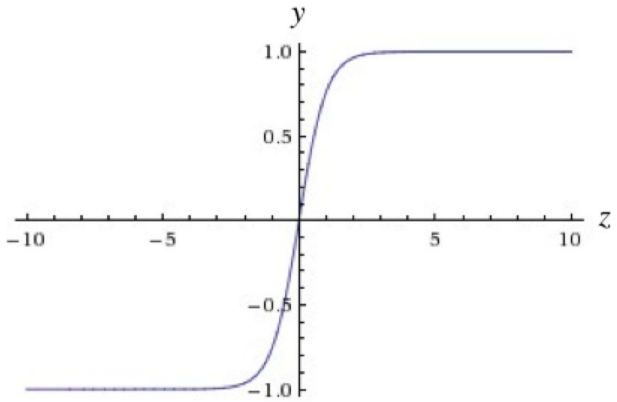


Figura 5 – Output of the Tanh function as z varies. Source: Image taken from (BUDUMA; BUDUMA; PAPA, 2022).

2.2.1.2.3 Rectified Linear

The rectified linear activation function is expressed as follows:

$$f(z) = max(0, z) \tag{2.3}$$

A different type of nonlinearity is used in this activation function, as illustrated in Figure 6.

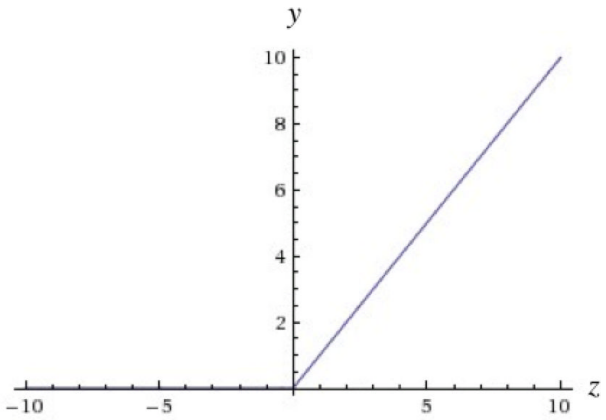


Figura 6 – Output of the rectified linear function as z varies. Source: Image taken from (BUDUMA; BUDUMA; PAPA, 2022).

Neurons that have rectified linear as their activation function in their outputs are called ReLU (Rectified Linear Units) and are easier to optimize because they are very similar to linear units. The difference is that in the ReLU output half of its domain is zero, which causes the derivatives to remain large whenever the activation function is operating in the active part ($z > 0$). The gradients are not only large, but also consistent. The second derivative of the function is 0 almost everywhere, and the first derivative of

the rectification operation is 1 everywhere the unit is active. This means that the direction of the gradient is much more useful for learning than it would be with activation functions that introduce second-order effects (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.2.1.3 Softmax Layer

For classification tasks, it is desirable that the network's output vector be a probability distribution over a set of mutually exclusive labels. That is, for K classes there would be K corresponding vectors, and it would be difficult for the network to recognize the objects with 100% confidence. The use of a probability distribution indicates how confident the predictions are. As a result, the output vector would have the following form:

$$[p_0, p_1, ...p_{K-1}], \tag{2.4}$$

where the sum of the probabilities of all outputs is expressed by:

$$\sum_{i=0}^{K-1} p_i = 1 \tag{2.5}$$

One way to achieve this behavior is to use a special output layer, called a Softmax layer. Unlike other types of layer, the output of a neuron in a softmax layer depends on the outputs of all other neurons in its layer. This is because the sum of all outputs must be equal to 1. This normalization can be achieved as follows:

$$y_i = \frac{e^{z_i}}{\sum_{j=0}^{K-1} e^{z_j}}, \tag{2.6}$$

where z_i is the value of the i-th neuron in the output layer before the normalization calculation, and K is the total number of classes.

A strong prediction would have a single vector component close to 1, while the remaining components would be close to 0. A weak prediction would have multiple possibilities, and the vector components would have close values, more or less equally likely (BUDUMA; BUDUMA; PAPA, 2022).

2.2.1.4 MLP Architecture

The MLP consists of three or more layers. Since an MLP is a fully connected network, each neuron in a layer connects its output to the input of other neurons in the next layer by multiplying their output values with the input synaptic weights of those other neurons. Figure 7 shows this architecture, also known as a feedforward architecture. The goal of a *feedforward* network is to approximate some function f' . For example, a classifier $y = f'(x)$ maps an input x to a category y. A feedforward network defines a mapping $y = f(x; w; \theta)$ and learns the values of the parameters that best approximate this function (GOODFELLOW; BENGIO; COURVILLE, 2016).

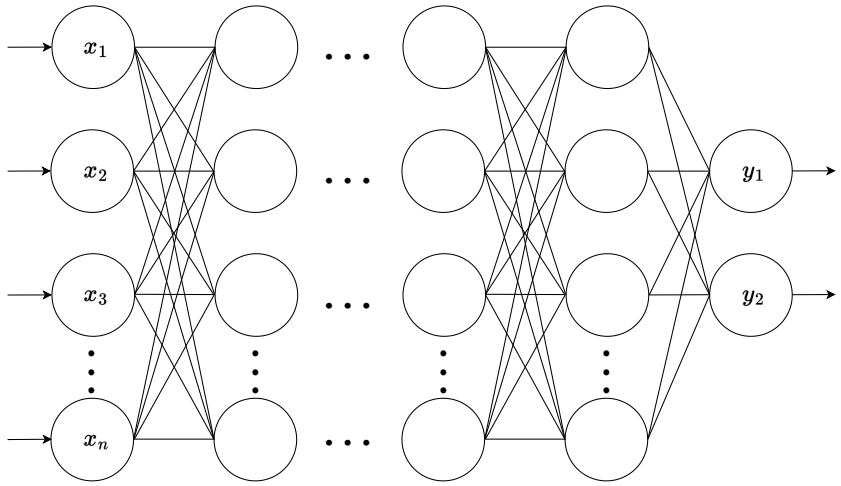


Figura 7 – Multilayer Perceptron Architecture: the first layer is responsible for receiving the input signals x_i, the last layer emits the output signals y_i, and the intermediate layers are also known as hidden layers. Source: The author.

2.2.2 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a specialized type of neural network architecture commonly used for signal and image processing, which can individually process sections of data at regular intervals, in a one-dimensional format, in the case of time series, or in a two-dimensional format, in the case of images, these are just a few examples of use.

This processing is done by filters (temporal or spatial, depending on the data domain) that slide over the data. These filters are also known as masks, kernels, templates and win-

dows, names inherited from the area of signal and image processing (GONZALEZ; WOODS, 2006).

In (LECUN et al., 1989), backpropagation was used to learn the coefficients of the convolution kernel directly from images of handwritten numbers.

The filters employ a mathematical operation called convolution, which is a type of linear operator. CNNs are simply neural networks that use convolution instead of multiplying the weight matrix with the data (input, or output from a previous layer) in at least one of their layers (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.2.2.1 Convolution

Convolution is defined as the integral of the product of one function by a translated and inverted copy of the second function. The convolution of two continuous functions f and g can be written as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau).g(t-\tau)d\tau, \tag{2.7}$$

where t is the independent variable and τ is the variable that defines the displacement. The discrete-domain version of the convolution is expressed as follows:

$$(f * g)[n] = \sum_{n=0}^{k} f[k].g[n-k], \tag{2.8}$$

where f and g are discrete functions that, in practice, are finite numerical sequences of equal or different sizes, and n is the independent variable and k is the variable that defines the displacement.

A version of convolution for discrete functions in two dimensions (widely used in digital image processing) is given as follows:

$$(f * w)[x, y] = \sum_{i=0}^{N} \sum_{j=0}^{M} f[i, j].w[x-i, y-j], \tag{2.9}$$

where f and w are discrete bounded functions of size M and N respectively, which vary with x and y, with i and j being the displacement variables. The practical effect of convolutions of this type can be expressed as feature extractors that can operate on the image generating new resulting images.

2.2.2.2 Convolutional Layers

Convolutional layers are responsible for extracting features from the input data. The process of extracting these features is done through convolutional filters (which are generally smaller than the data), where the filters traverse the input data in width, height and depth (channels), performing the convolution operation on the data.

With each input processing in the network training period, the filters are adjusted in such a way that they trigger when the input contains a certain detected feature that was learned by the filters, such as edges, colors, etc. In the following convolutional layers, the filters learn increasingly complex structures. In short, the more filters and convolutional layers, the more features it is possible to extract from the data.

2.2.2.3 Pooling Layers

Pooling layers aims to reduce the size of the data (only in the width and height dimensions). The best-known pooling operations are max-pooling and average-pooling. Max-pooling reduces the subparts of the data by the maximum value found in these areas; in an analogous way, average-pooling does this reduction using the average of the values of these areas, Figure 8 illustrates these operations.

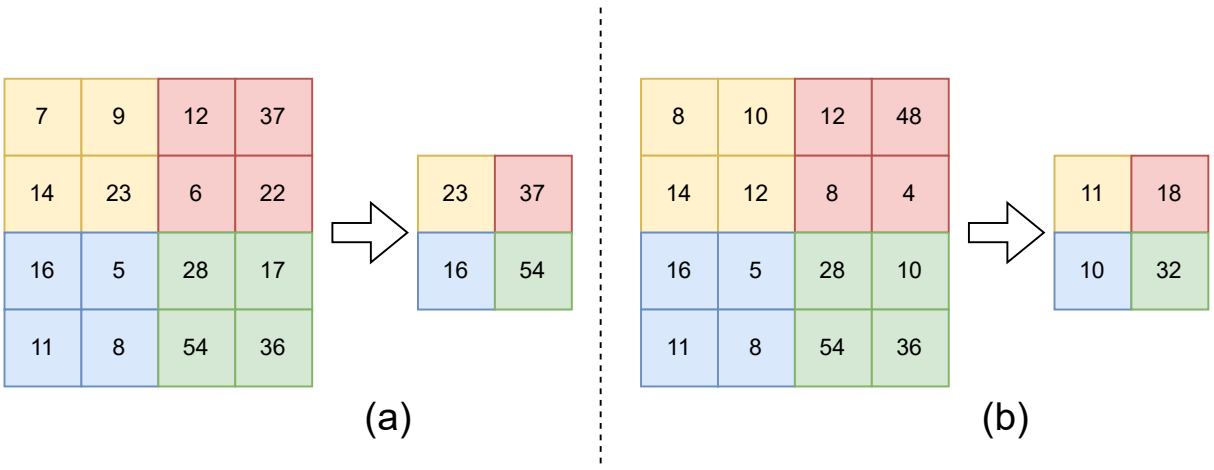


Figura 8 – Examples of masked pooling operations in the form 2×2: Max-pooling in (a) and Average-pooling in (b). Source: The author.

2.2.2.4 Fully-Connected Layers

Fully-connected (FC) layers are usually located at the end of the network. In these layers, the features extracted in the previous convolution layers are passed to these layers, which will act as the network's classifier. The architecture of a CNN (which aims to recognize images) can be basically divided between feature extraction and classification. The operation of FC layers is similar to that of MLP layers, with no distinction. Below, in Figure 9, a FC layer is shown receiving a vector with three inputs and two output neurons, and its calculations in matrix format.

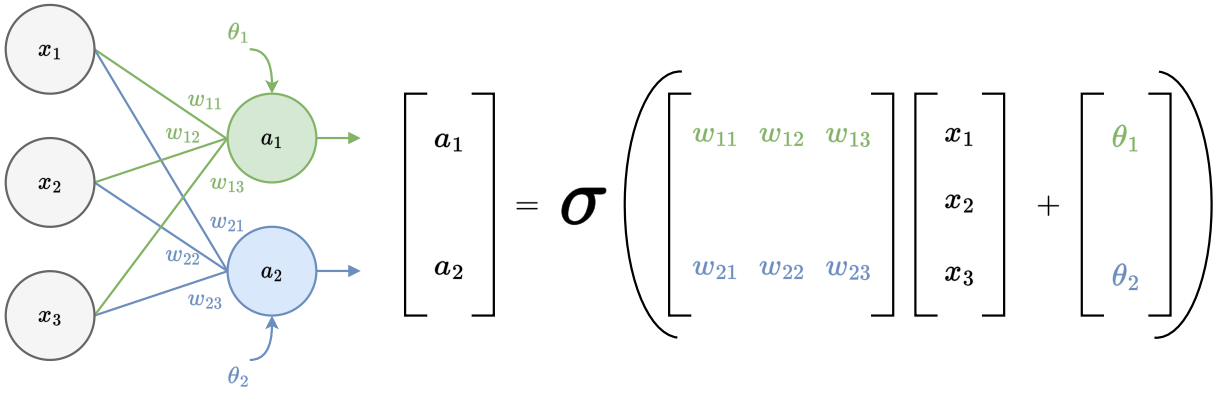


Figura 9 – A fully-connected layer with three inputs and two output neurons, where the input vector $(x_1, x_2, x_3)^T$ is multiplied by the weights w_{ij} through the weight matrix and added with the biases by the vector $(\theta_1, \theta_2)^T$, and the values a_1 and a_2 are the output responses of these neurons after the composition of the activation function σ with all this matrix calculation. Source: The author.

2.2.2.5 CNN Architecture

All of these layers are commonly used in CNN architectures. In short, convolutional layers act as feature extractors, pooling layers reduce data, and fully connected layers perform classification. An example of a CNN architecture comprising the three aforementioned layers (convolutional, pooling, and fully connected) is illustrated in Fig. 10. This architecture, known as LeNet, was the first convolutional neural network trained using backpropagation and was introduced by (LECUN et al., 1989).

In addition to the layers previously described, similar to MLP networks, CNN-based architectures also include activation functions, which are responsible for introducing non-linearity to the model, allowing it to learn more complex representations. Additionally,

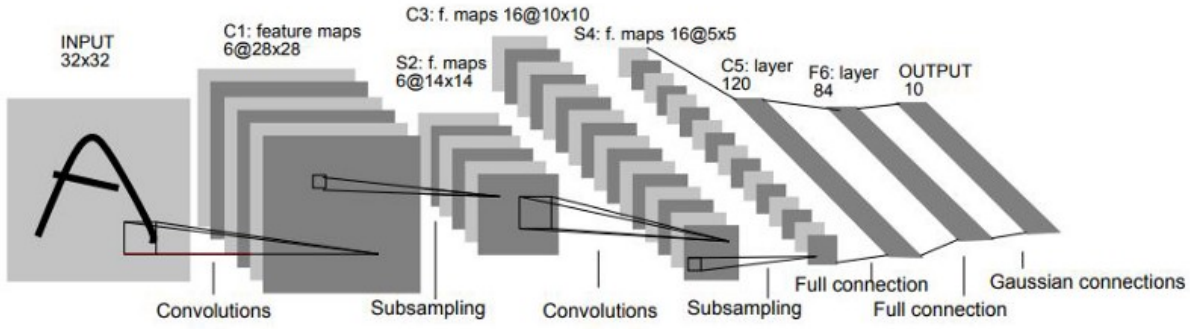


Figura 10 – Example of LeNet architecture. Source: Image taken from (LECUN et al., 1989).

when used for classification tasks, these networks usually end with a Softmax layer, which transforms the network outputs into probabilities associated with each class, reflecting the degree of confidence of the prediction.

2.2.3 Vision Transformer

A Vision Transformer is a deep learning architecture that uses the principles of Transformers, which are commonly used in natural language processing (NLP), and has now been adapted for computer vision tasks. Instead of using convolutions as in traditional image models, ViT divides the image into patches and treats them as tokens, although a hybrid architecture using convolutional filters is also allowed according to the original ViT paper (DOSOVITSKIY et al., 2020) in the **Hybrid Architecture** section:

> *"As an alternative to raw image patches, the input sequence can be formed from feature maps of a CNN (LECUN et al., 1989). In this hybrid model, the patch embedding projection E (Eq. 1) is applied to patches extracted from a CNN feature map. As a special case, the patches can have spatial size* 1×1*, which means that the input sequence is obtained by simply flattening the spatial dimensions of the feature map and projecting to the Transformer dimension. The classification input embedding and position embeddings are added as described above."*

After processing the image patches (using convolutional or FC layers), this sequence of patches is concatenated with a trainable class token and added with trainable positioning parameters and are passed to a Transformer Encoder that will process this entire sequence

of tokens and return the first token that will be used in an MLP network that will classify the image.

In summary, a Vision Transformer operates with the following steps:

1. **Patch Division**: The input image is segmented into fixed-size patches, which are treated as individual tokens.

2. **Embedding**: Each patch is linearly flattened and further passed through a Fully-connected layers (or convolutional layers, in hybrid architecture) with a linear activation function transforming the patches into feature vectors (patch embeddings), to which it is concatenated with a learnable class token, whose final state captures the aggregated information for classification. Through attention layers, this token learns to summarize the global features of the image. To preserve the spatial structure of the patches, a one-dimensional positional embedding is added, since Transformers by themselves are permutation invariant and cannot infer the order of the input tokens.

3. **Transformer Encoding**: The sequence of tokens is processed by a Transformer encoder, which uses self-attention mechanisms to capture complex relationships between the different patches.

4. **Classification (or other tasks)**: For image classification tasks, a special classification token is added to the beginning of the sequence, and the corresponding output is used to predict the class of the image.

Figure 11 shows the architecture of ViT and these steps, from patch division to classification.

2.2.3.1 Transformer Encoder

The Transformer Encoder (TE) is an important ViT module, and was initially designed for NLP tasks (VASWANI et al., 2017). The architecture of TE is composed of several stacks of identical blocks. Each block contains a Multi-Head Attention layer, followed by a Feed-Forward Network. In both sublayers, there are residual connections, followed by a Layer Normalization. All sublayers, as well as the embedding layers of the model, maintain an output dimension equal to the input, as shown in Fig. 11.

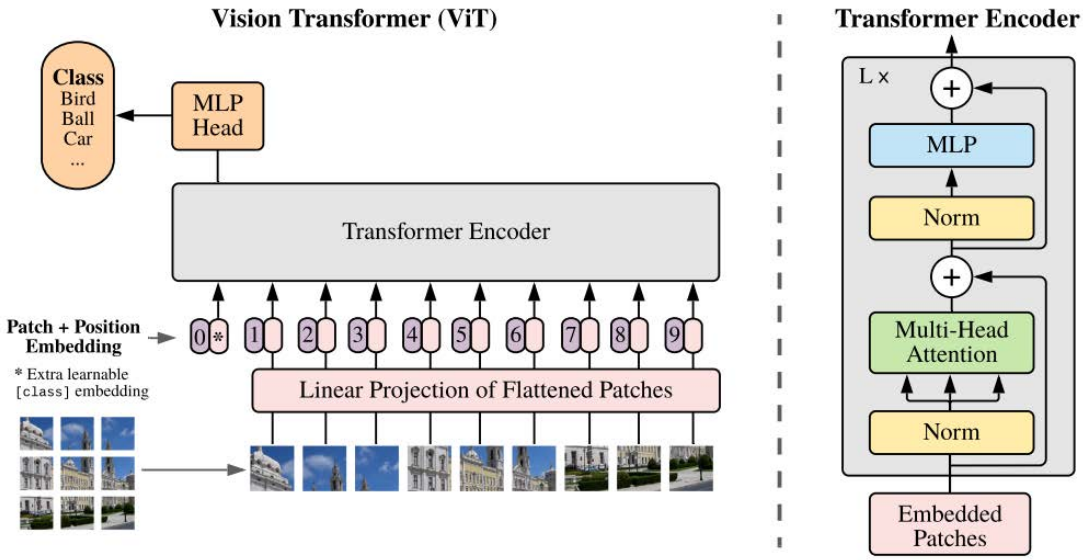


Figura 11 – Vision Transformer architecture and Transformer Encoder. Source: Image taken from (DOSOVITSKIY et al., 2020)

2.2.3.1.1 Multi-Head Attention

The Transformer Encoder includes a self-attention mechanism known as Multi-Head Attention. In this mechanism, each input embedding x (representing an image patch or a word, depending on the task) is linearly projected into three distinct vectors: query (q), key (k), and value (v), using learnable weight matrices. This process simulates a retrieval operation, where a query vector interacts with all keys (including its own) to determine the relevance of each input token. When generalizing to matrix form, the individual projections q, k, and v for each token are stacked to form the matrices Q, K, and V, respectively. The Multi-Head Attention mechanism performs this self-attention operation in parallel across multiple "heads", each using its own set of learned projection matrices Q, K, and V, these are concatenated and once again projected. This enables the model to capture diverse relationships and dependencies from different subspaces of the input representation. By attending to different positions and features simultaneously, the model can learn richer contextual representations (VASWANI et al., 2017). Fig. 12 illustrates this mechanism.

2.2.3.1.2 Scaled Dot-Product Attention

The main component of a Multi-Head Attention unit is the Scaled Dot-Product Attention. At first, the input vectors are duplicated three times and multiplied by weights

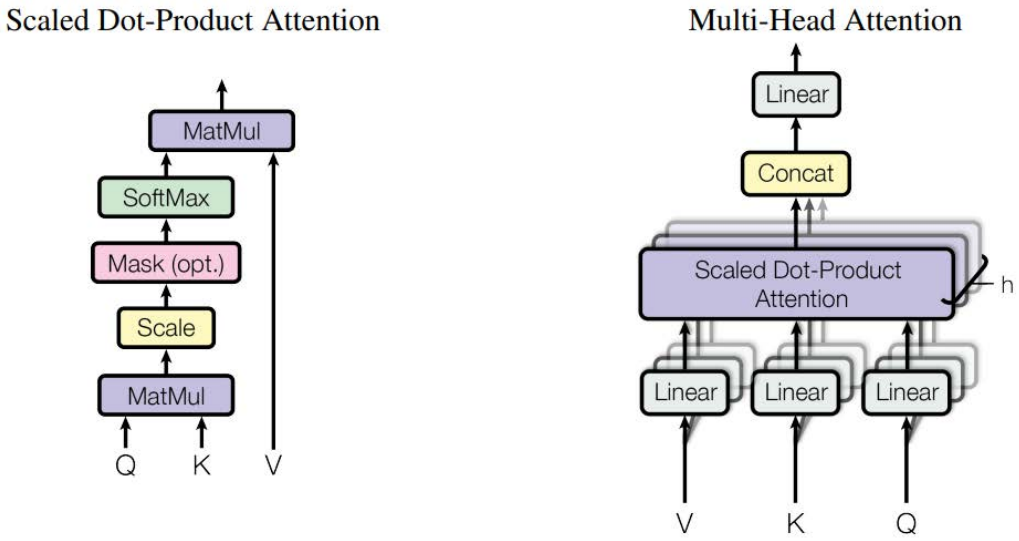


Figura 12 – Scaled Dot-Product and Multi-Head Attention mechanisms. Source: Image taken from (VASWANI et al., 2017)

W_q, W_k, and W_v, to get the **Queries** (Q), **Keys** (K), and **Values** (V) respectively. The **Queries** (Q) are then multiplied by the transposed **Keys** (K^T), and the result is divided by the square root of the dimension, $\sqrt{d_k}$, to avoid the vanishing gradient problem. The resulting matrix is subjected to a Softmax operation, which converts the scores into attention weights. These weights are subsequently multiplied by the **Values** (V) matrix, as described in the following equation.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.10}$$

2.3 LEARNING MECHANISM AND TRAINING

This section covers topics necessary to understand the ANN learning mechanism, such as: Error Functions, Gradient Descent and Backpropagation. And the process to train neural networks.

2.3.1 Error Functions

Artificial neural networks learn through the process of adjusting the weights of the inputs in each layer. To adjust these weights, it is necessary to define a metric of how close a result is to the expected one. This metric is called the error function. Some well-

known error functions are: Means Square Error (MSE), commonly used for regression problems, and cross-entropy, which is more commonly used for classification problems. In the following equations 2.11 and 2.12, the calculations of these cost functions are shown.

The MSE can be calculated as:

$$MSE(Y, Y') = \frac{1}{N} \sum_{i=1}^{N} (Y_i - Y_i')^2 , \tag{2.11}$$

where N is the number of training samples, Y_i and Y_i' represent the expected and predicted output of the i-th sample, respectively.

To calculate cross-entropy, we have:

$$CE(Y, Y') = -\frac{1}{N} \sum_{i=1}^{N} \Big(Y_i \log Y_i' + (1 - Y_i) \log(1 - Y_i') \Big), \tag{2.12}$$

where N is the number of training samples, Y_i and Y_i' represent the expected and predicted output of the i-th sample, respectively.

2.3.2 Gradient Descent and Backpropagation

For the network to achieve the desired result, it is necessary to adjust the weights, and the most widely used algorithm to perform this task is Backpropagation with Gradient Descent.

Gradient Descent is an iterative optimization algorithm that aims to find a local (or global) minimum of a function, in which each iteration takes the negative direction of the gradient. For training neural networks, gradient descent is used to minimize the error function in relation to the weight parameters w and biases θ.

Backpropagation was developed in (RUMELHART; HINTON; WILLIAMS, 1986) and is an algorithm that updates the network weights in order to optimize them during training. It calculates the gradient of the error function in relation to the network weights for a single or multiple input and output examples and performs it efficiently, unlike a direct calculation of the gradient in relation to each weight individually. This efficiency makes it feasible to use gradient methods for training multilayer networks, updating the weights to minimize losses.

In short, the weight values are changed by backpropagation iteratively, where the gradient is multiplied by a value known as **learning rate** that can be fixed throughout

training, or governed by a decay policy. The goal of gradient descent is to minimize the error function to the lowest possible error.

Figure 13 illustrates this iterative process, and equation 2.13 shows how the weights are updated using the sampling rate. For the purpose of didactic simplification, a case is being shown where there is only one weight variable w, which is an impractical situation when dealing with neural networks that usually have at least dozens of parameters to be optimized.

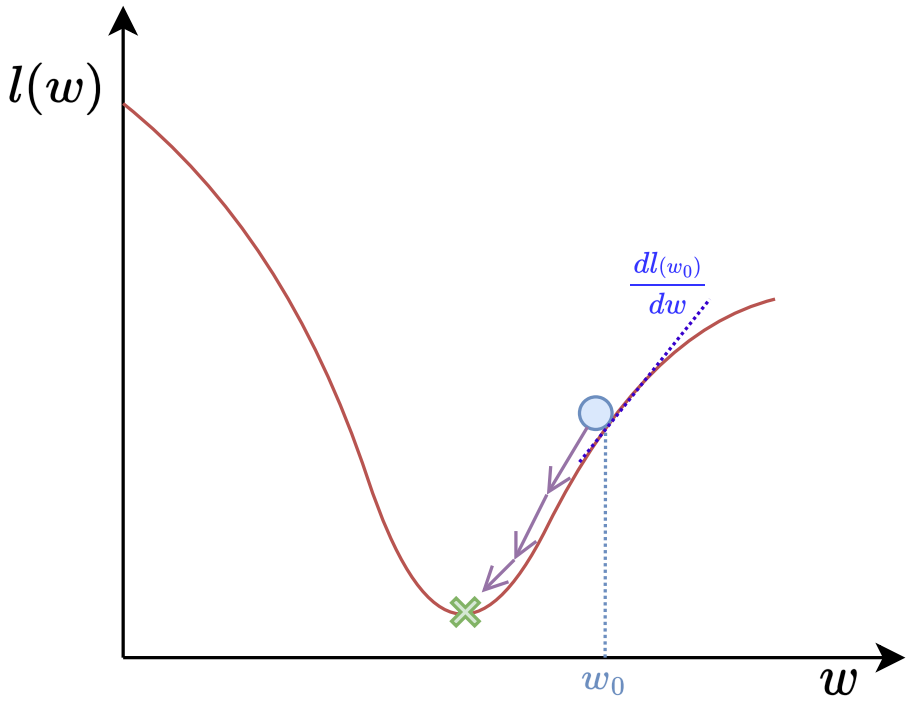


Figura 13 – The graph shows the error function l as a function of a weight variable w. The gradient descent algorithm starts from an initial value $l(w_0)$ and tries to minimize this error through the slope obtained by calculating the derivative at each instant of the iteration, getting closer and closer to the local minimum signaled by **x**. The blue dashed line, tangent to the error function curve, is the derivative at the initial weight value, $\frac{dl(w_0)}{dw}$. The arrows pointing to the minimum are the displacements obtained in each iteration of the algorithm, which have their value attenuated or amplified by the sampling rate. Source: The author.

$$w_{t+1} = w_t - \alpha \frac{dl(w_t)}{dw}, \tag{2.13}$$

where the next weight value w_{t+1} will be calculated from the weight variable in the current iteration w_t subtracted from the second term, consisting of the sampling rate α that multiplies the derivative term, where α plays the role of the ratio of the step sizes during the iterations.

2.3.3 Training

2.3.3.1 Epoch and Batch

The training of an ANN is iterative as already discussed, data samples are passed to the network, where each batch of data can be updated using the backpropagation algorithm. It is important to distinguish two concepts: Epoch and Batch.

- **Epoch**: An Epoch is an iteration where all samples were passed to an ANN during training, this process is repeated for several epochs.
- **Batch**: A Batch is a subset of the total data samples within an Epoch, where at the end of each Batch the weights are updated via backpropagation.

The batch size is varied, it is possible to use just a single sample per batch during training, or the entire dataset per batch, or an intermediate number of samples.

2.3.3.2 Underfitting and Overfitting

During training, it is recommended to use a dataset for training that is different from the validation and/or test dataset, in order to assess whether the network is able to generalize well beyond the training data. Two undesirable situations can occur related to training: underfitting and overfitting.

- **Underfitting**: It is a situation where the model cannot even have a good fit to the training data. This problem is easier to identify, being the worst case scenario.
- **Overfitting**: It is when the network can fit well to the training data, but cannot generalize well to the validation data.

The possible reasons why an ANN-based model is suffering from **underfitting** are listed below:

- Model with low complexity for the data: not very powerful for learning, it is possible to amplify the power of the model by increasing the number of parameters (more layers, more units per layer).

- Non-representative features: in this case, it may be that the features we are using to train the model are not representative (they are not related to each other or are not important for the model).
- Model with too many restriction parameters: the model becomes inflexible, restricted, and cannot adequately fit the data.

Some of the main causes of **overfitting** can be:

- Model too complex for the data: we can simplify our model by choosing a simpler model, with fewer parameters.
- Little training data: it may be necessary to collect more data to train the model, or use data augmentation.
- Noise in the training data: If there is any type of noise, extreme values or even incorrect values in the data, if the model learns from this type of data, it can lead to overfitting. Adequate preprocessing of the training data would be necessary.

2.3.3.3 Regularization

Regularization is a set of techniques used in machine learning and neural networks to improve a model's generalization, i.e., its ability to perform well on data not seen during training. It works by penalizing the complexity of the model or introducing controlled noise during training, reducing the risk of overfitting, when the model overfits the training data and fails to generalize.

In general, regularization seeks to find a balance between learning capacity and model simplicity, encouraging solutions that are flexible enough to capture patterns in the data, but not so complex that they memorize noise or particularities of the training set (GOODFELLOW; BENGIO; COURVILLE, 2016). Below are some classic examples of regularization techniques.

2.3.3.3.1 L1 and L2 Regularization

These techniques penalize large weight values by adding a regularization term to the loss function.

L1 regularization: adds the sum of the absolute values of the weights, encouraging sparsity in the model parameters (TIBSHIRANI, 1996):

$$\mathcal{L}_{\text{new}} = \mathcal{L}_{\text{original}} + \lambda \sum_i |w_i| \tag{2.14}$$

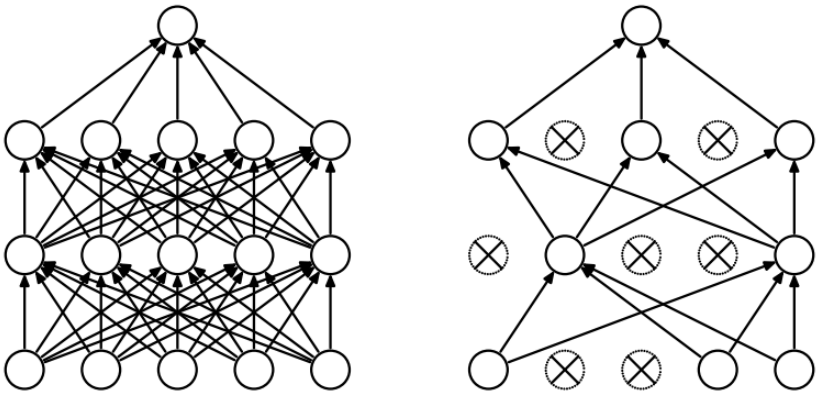
L2 regularization: adds the sum of the squared values of the weights, discouraging large weights but without driving many to zero (HOERL; KENNARD, 1970):

$$\mathcal{L}_{\text{new}} = \mathcal{L}_{\text{original}} + \lambda \sum_i w_i^2 \tag{2.15}$$

In both cases, $\mathcal{L}_{\text{original}}$ denotes the original loss function (e.g., cross-entropy or mean squared error), w_i are the model's parameters, and λ is a regularization coefficient that controls the strength of the penalty, and $\mathcal{L}_{\text{new}}$ corresponds to the modified loss after regularization is applied.

2.3.3.3.2 Dropout

Dropout randomly deactivates neurons during training, preventing co-adaptation of features and encouraging the network to learn redundant and robust representations that generalize better to unseen data (SRIVASTAVA et al., 2014).



(a) Standard Neural Net (b) After applying dropout.

Figura 14 – Dropout Neural Network: Standard neural network in (a) with two hidden layers. Thinned network after applying dropout in (b), crossed units represent dropped neurons. Source: Image taken from (SRIVASTAVA et al., 2014).

2.3.3.3.3 Data Augmentation

Data Augmentation generates new training samples from modifications to the original samples, through geometric transformations (e.g. rotations, translations, scales),

variation of brightness, contrast and color (pixel augmentation), and addition of noise (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

2.3.3.3.4 Batch Normalization

This regularization technique normalizes the output of each layer by adjusting and scaling the activations. Specifically, it standardizes the inputs to a layer for each mini-batch such that they have zero mean and unit variance. This normalization helps reduce internal covariate shift (the change in the distribution of layer inputs during training) thereby stabilizing and accelerating convergence (IOFFE; SZEGEDY, 2015).

2.3.3.3.5 Weight Decay

It is a regularization technique used directly in optimizers. It can be understood as a variant of L2 regularization, but applied directly in the weight update process, instead of as an explicit additional term in the loss function. The Weight Decay penalizes high values in the network weights, encouraging solutions with lower complexity and promoting better generalization. This is done by adding a fraction of the weights themselves to the gradient during the update, causing them to decay slowly over the course of training (LOSHCHILOV; HUTTER, 2019).

2.3.3.4 Weight Initialization

In deep neural networks, proper initialization of weights is essential to ensure successful training. Initializing weights with inappropriate values can cause two main problems:

- **Exploding gradients**: when gradients increase exponentially during backpropagation, leading to numerical instability.
- **Vanishing gradients**: when gradients tend to zero, preventing weights from being updated, especially in the initial layers of the network.

These problems affect signal propagation and error backpropagation, making learning difficult or even impossible. Initialization algorithms help preserve signal variance between layers, allowing for more stable and efficient training.

2.3.3.4.1 Weight Initialization Algorithms

It is listed below some common weight initialization algorithms.

- **Zero Initialization**: Initialize all weights to zero: In general, it is not recommended, as all units learn the same weights (the symmetry problem).
- **Random Initialization**: Weights are sampled from uniform or normal distributions with small variance. It can work for very shallow networks, but does not guarantee stability in deep networks.
- **Xavier (Glorot) Initialization** (GLOROT; BENGIO, 2010): Ideal for symmetric activation functions like tanh and sigmoid. Keeps the activation variance constant across layers.

2.4 EVALUATION OF CLASSIFIERS

This section presents some issues considered in the evaluation of classification models, such as the confusion matrix and its metrics.

2.4.1 Confusion Matrix

The Confusion Matrix is a tool widely used to evaluate classification models, determining values that will be important for other metrics. The confusion matrix records the number of false positives, false negatives, true positives and true negatives. These concepts are briefly explained below:

- **True Positive** (TP): The class being searched for was predicted correctly.
- **True Negative** (TN): The class not being searched for was predicted correctly.
- **False Negative** (FN): The class not being searched for was predicted incorrectly.
- **False Positive** (FP): The class being searched for was predicted incorrectly.

The class of elements of interest during classification is treated as positive and the other class as negative. From these values, many important metrics for evaluating classifiers can be obtained. Table 2 and Fig. 15 help illustrate these concepts.

Tabela 2 – Confusion Matrix

		Predict Value	
		Yes	*No*
Real Value	*Yes*	True Positive (TP)	False Negative (FN)
	No	False Positive (FP)	True Negative (TN)

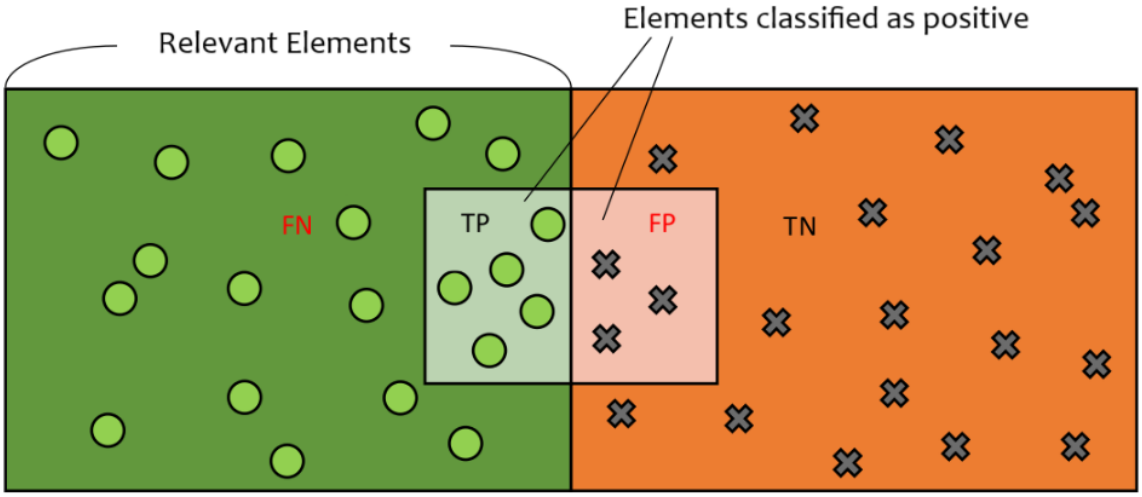


Figura 15 – Example of Classification. Source: The author.

2.4.2 Evaluation Metrics

The classification metrics extracted from the confusion matrix are:

- Accuracy
- Precision
- Recall
- Specificity
- F1-Score

2.4.2.1 Accuracy

It evaluates the percentage of hits of the classifier, obtained by the ratio between the number of hits (TP + TN) and the total number of samples (TP + FP + FN + TN), as shown in Fig. 16.

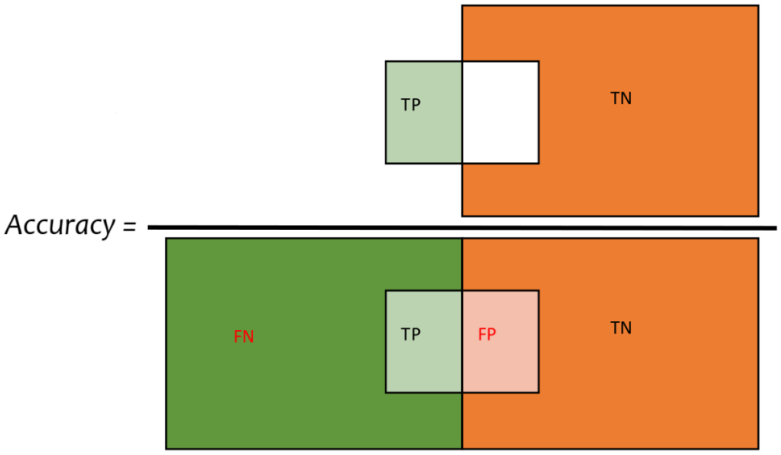


Figura 16 – $Accuracy = \frac{(TP+TN)}{(TP+FP+FN+TN)}$. Source: The author.

2.4.2.2 Precision

It evaluates how many selected elements are relevant by the classifier, it is obtained by the ratio of the number of true positives (TP) to the sum of all values detected as positive by the model (TP + FP), as shown in Fig. 17.

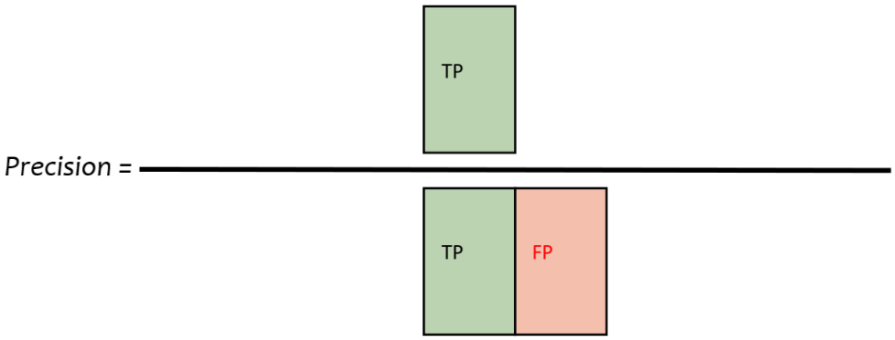


Figura 17 – $Precision = \frac{TP}{(TP+FP)}$. Source: The author.

2.4.2.3 Recall

It evaluates how many relevant elements were selected by the classifier, it is obtained by the ratio of true positives (TP) to all elements labeled as positive, relevant (TP + FN), as shown in Fig. 18.

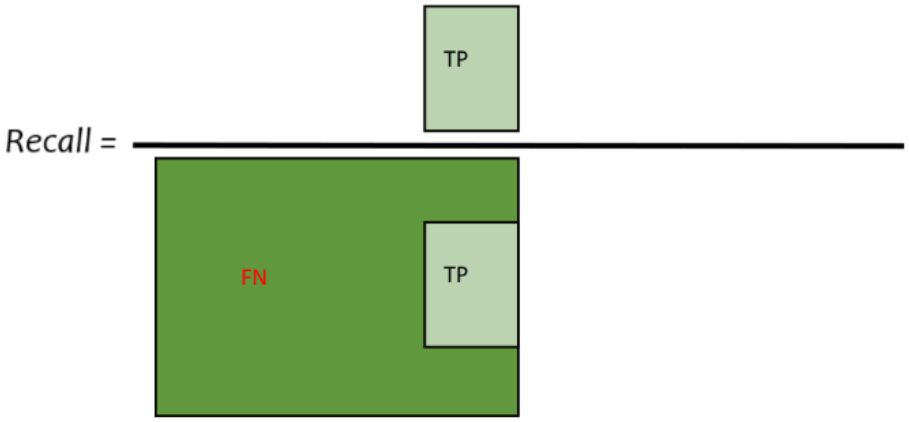


Figura 18 – $Recall = \frac{TP}{(TP+FN)}$. Source: The author.

2.4.2.4 Specificity

It evaluates the ability of the classifier to detect negative results, obtained by the ratio of true negatives (TN) to the total number of samples labeled as negative (TN + FP), as shown in Fig. 19.

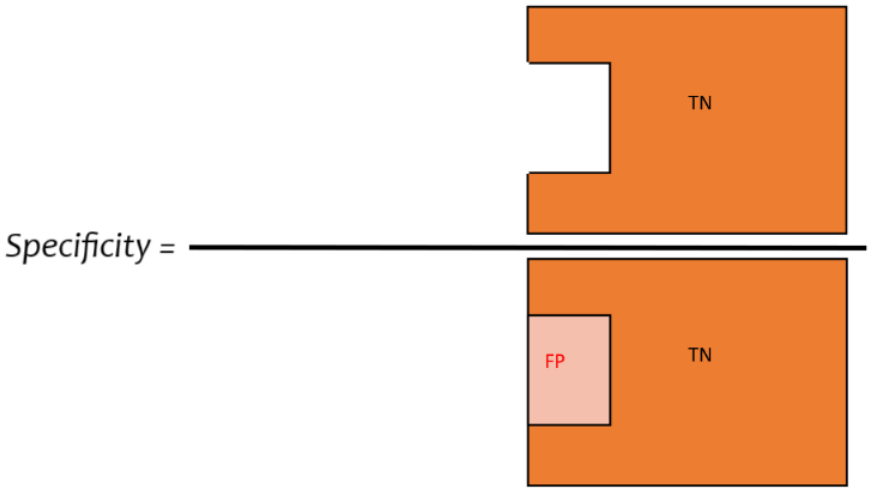


Figura 19 – $Specificity = \frac{TN}{(TN+FP)}$. Source: The author.

2.4.2.5 F1-Score

It is a harmonic mean calculated based on precision and recall. It is obtained based on the equation:

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{2.16}$$

2.5 DISCUSSION

This chapter presents a concise overview of the hierarchy of geometric transformations and their associated geometric invariants. It also introduces the operating principles of Artificial Neural Networks, covering the evolution from basic artificial neurons to advanced architectures such as MLPs, CNNs, and ViTs, along with their respective layers and operations. In addition, the chapter addresses the learning mechanisms underlying neural networks. Finally, it details the classification evaluation metrics that will be used to assess the model developed in this work.

3 RELATED WORKS

This Chapter presents different techniques for description and classification of planar shapes, divided between two major approaches: Planar Shape Recognition by Handcrafted Shape Descriptors and by Artificial Neural Networks. A historical review is made and works more directly related to the recognition of these shapes with robustness are also shown.

3.1 PLANAR SHAPE RECOGNITION BY SHAPE DESCRIPTORS

This Section reviews and discusses some works related to the description and classification of planar shapes; classical works from the literature and more recent works directly linked to the problem of recognizing planar objects with projective deformations are mentioned.

Some research works on shape descriptors, such as (ZHANG; LU, 2004), (LONCARIC, 1998), classify various descriptors by the following properties:

- Contour-Based × Region-Based
- Global × Structural (Local)
- Spatial Domain × Transform Domain

Region-based methods use information from the interior of the object, such as color and/or texture; unlike contour-based methods, which consider only the edges of the object, neglecting information from the interior of the shape (KAZMI; YOU; ZHANG, 2013). Both Global and Structural methods are subdivided into contour-based and region-based approaches. There are several approaches that propose to describe shape characteristics for planar objects that are invariant to rotation, translation and scale and even more general transformations such as *affinities* and *projectivities*. An example of a classical descriptor invariant to scale and Euclidean transformations is the Generalized Hough Transform (BALLARD, 1981), an algorithm that uses object edge information, such as gradient vectors, to define a mapping in a space of parameter accumulators referring to rotation angle θ, location (x, y) of the centroid and scale S of the object, constituting a descriptor invariant to *similarity*. Fourier descriptors are used in the recognition of shapes

invariant to *affinities*, these descriptors work with information in the frequency domain of a closed planar curve, lower frequency components are responsible for approximating the shape of the curve, while higher frequencies describe the finer details of the shape to be represented.

Typical approaches for symbol (planar shape) recognition are texture, color, and shape. In the past decades, significant efforts have been devoted to texture features, among which the SIFT descriptor (LOWE, 2004) and its variants, e.g., SURF (BAY et al., 2008) and PCA-SIFT (KE; SUKTHANKAR, 2004), have gained great success. These descriptors are mainly derived based on local texture information (JIA et al., 2014). *Scale-Invariant Feature Transform* (SIFT) is a local affine-invariant descriptor that describes a local region around a *key point*. A SIFT descriptor is robust to occlusion and does not require segmentation. The comparative evaluation of region-based local descriptors tested in (MIKOLAJCZYK; SCHMID, 2005) showed that the SIFT descriptor performs significantly better than other descriptors in the same category. In (ROUTRAY; RAY; MISHRA, 2017), some descriptors based on intensity information are explored, where tests are carried out with the SIFT, SURF and HOG descriptors in noisy images. SIFT proved to be the best, obtaining the maximum number of feature points and associated points.

However, there are numerous practical scenarios in image recognition where richer textures are not always available, but only distinct geometric features, especially for the recognition of artificial symbols, such as characters, traffic signs and logos. More recent studies in symbol recognition (planar shapes) (LI; TAN, 2010) have shown that a SIFT descriptor has low discrimination while contour-based descriptors discriminate better (JIA et al., 2014). In mid-2010, a global contour-based shape descriptor was released, which makes use of rectilinear trajectories that start from points on the edges of a convex polygon that surrounds the object; such trajectories intersect with the edges of the object forming collinear points; only four of these points are chosen to calculate a cross-ratio value for each trajectory; as seen in section 2.1, the cross-ratio is the most fundamental projective invariant; this descriptor proposed by Li and Tan (LI; TAN, 2010) groups several cross-ratio values into spectra. The method was dubbed CRS, for *Cross Ratio Spectrum* (CRS). CRS has been successful in representing simple planar shapes, such as characters, that have suffered severe projective deformations. A representation for planar objects invariant to projective transformations, based on approximations of parts of the object's contour in pairs of conics, was presented in (SRESTASATHIERN; YILMAZ, 2011). The method proved

to be quite robust, but was only tested on planar shapes that did not present internal structures, only the silhouette. In (LUO et al., 2013) a method similar to CRS was proposed in the use of trajectories that cut the object to calculate a new geometric invariant called the characteristic number (*characteristic number*, CN) and a new shape descriptor based on CN values calculated from collinear points was presented, with the advantage of using more collinear points to obtain a single CN value, increasing the descriptive power including more information to calculate the invariant. A compact shape descriptor called *Hierarchical Characteristic Number Context* (HCNC) was shown in (JIA et al., 2016) and was robust to perspective deformations. This descriptor makes use of the same invariant already mentioned in (LUO et al., 2013), the characteristic number (CN); the descriptor is built on a coarse-to-fine strategy that combines the global geometry given by projective invariants and local contextual information in a hierarchical feature descriptor.

In (CHARAMBA; MELO; LIMA, 2021), we proposed a method that extracts projective invariant features through the casting of rays emitted by different points of a convex hull that surrounds the object, these rays cut the planar shape generating collinear edge points where it is possible to calculate cross-ratio values that can be stored in vectors linked to each ray, generating a hierarchical descriptor that can be used to calculate a dissimilarity distance based on the *matching* of rays by these cross-ratio vectors, *Cross-ratio Arrays* (CRA) and backprojection error, with this distance obtained between descriptors it is possible to classify the projectively deformed planar shapes, following a nearest neighbor approach. Comparative experiments showed that CRA was able to outperform HCNC on a dataset containing planar shapes that have rich interior structure, such as logos and traffic signs, and that it maintained a good degree of robustness for recognition as the degree of projective deformation became more severe.

3.2 PLANAR SHAPE RECOGNITION BY NEURAL NETWORKS

Significant advances have been made using capsule networks to handle this invariance issue. For example, (HINTON; KRIZHEVSKY; WANG, 2011) addressed this gap by proposing Capsule Nets, which take advantage of the neuron grouping models' better handling of variations concerning position, scale, lighting, and orientation. The Routing-CapsNet exhibited high robustness on the affinely deformed images of the affNIST dataset (NETZER et al., 2011), being trained on the non-deformed images of the MNIST dataset (LECUN

et al., 1998), outperforming equivalent CNNs by (SABOUR; FROSST; HINTON, 2017). Other Capsule Nets variations emerged like GE-CapsNet by (LENSSEN; FEY; LIBUSCHEWSKI, 2018), showing proven equivariance and invariance properties in tests with the MNIST and affNIST datasets. Aff-CapsNets, introduced by (GU; TRESP, 2020), are robust to affine transformations, achieving high accuracy being affNIST-tested/MNIST-trained. The Capsule Nets' routing algorithm was modified by (RIBEIRO; LEONTIDIS; KOLLIAS, 2020) to incorporate global context, resulting in a 97.69% accuracy on the affNIST dataset after training on the original MNIST dataset. Additionally, (MACDONALD; RAMASINGHE; LUCEY, 2022) introduced a mathematical framework for convolutional neural networks over Lie groups, which was tested on datasets such as affNIST and homNIST, achieving accuracy rates of 95.08% and 95.71%, respectively, also using MNIST for training.

Another type of neural network architecture that provides invariance to some geometric transformations is the Spatial Transformer (ST). In their original paper, (JADERBERG et al., 2015) proposed a mechanism for neural networks to learn the transformation parameters from the input data. ST consists of three main components: the *localization network*, predicting transformation parameters from input features; the *grid generator*, creating a sampling grid; and the *sampler*, applying the grid to the input feature map to produce the transformed output. The Polar Transformer Network (PTN), designed by (ESTEVES et al., 2018b), combines ideas from the ST and canonical coordinate representations. The result is a network invariant to translation and equivariant to rotation and scale. The PTN is trained end-to-end and is composed of three distinct stages: A polar origin predictor, a transformer module, and a classifier. The Equivariant Transformer (ET) network was proposed by (TAI; BAILIS; VALIANT, 2019) which, similarly to ST, builds in prior knowledge on the continuous transformation invariances of its input domain. By encapsulating equivariant functions within an image-to-image mapping, unlike traditional STs, Equivariant Transformers incorporate additional structure in the functions used to predict transformations.

A considerable effort has been devoted to advancing techniques based on CNNs (Convolutional Neural Networks). (KUMAR; SHARMA; GOECKE, 2020) proposed a method to address rotations by augmenting feature maps rather than data. (MARCOS; VOLPI; TUIA, 2016) introduced a scheme for explicitly learning rotation-invariant by rotatable filters to avoid data augmentation in texture recognition. (NOORD; POSTMA, 2017) developed a multi-scale method using an ensemble of networks specializing in different image resoluti-

ons to learn scale-variant and scale-invariant features. However, these approaches attempt to enhance CNN robustness through ensembles, feature map augmentations, and filter modifications, which are computationally expensive and introduce additional complexity. Despite these efforts, achieving true invariance to simple geometric transformations, such as rotations and scaling, remains a challenge for these models. They lack robustness to nonlinear geometric changes, impairing performance in tasks involving severe deformations. (COHEN; WELLING, 2016) highlighted that CNNs fail to guarantee invariance or equivariance to transformations like rotations and scaling, proposing group-equivariant convolutional networks to address this limitation. (ESTEVES et al., 2018a) demonstrated that traditional CNNs perform poorly in tasks involving 3D transformations like rotations, relying heavily on data augmentation to improve results. Furthermore, (MARCOS et al., 2017) showed that while augmentation may partially mitigate the issue, CNNs still lack true equivariance to geometric transformations. (AZULAY; WEISS, 2019) revealed that convolutional architectures do not guarantee even the simplest transformation invariance, as they ignore the classical sampling theorem, even small translations or rescalings of the input image can significantly alter the network's prediction, and data augmentation alone cannot achieve true invariance, these findings were supported through evaluations of well-known architectures, including ResNet, VGG, InceptionResNet, and DenseNet; although convolutional filters are translation-equivariant, CNNs as a whole do not inherently guarantee translation invariance by default as explained by (BISCIONE; BOWERS, 2021). These studies emphasize the limitations of CNNs in handling even basic geometric transformations, underlining the need for specialized architectures to tackle these challenges effectively. This limitation makes CNNs less suitable for problems requiring robustness to complex geometric deformations like affinities and projectivities, justifying the exploration of alternatives like Capsule Networks, Spatial Transformers, and the approach proposed in this work.

3.3 DISCUSSION

This chapter presents a bibliographic review of classical approaches to shape description, aiming to describe them with characteristics that are invariant to various geometric transformations, from the simplest, such as rotation and scale, to the most complex, such as affinities and projectivities, for classification tasks. The same was done for more mo-

dern approaches, with the use of deep learning, where most of the most recent advances are based on spatial transformers and capsule networks and their experiments carried out for the recognition of images of handwritten digits deformed by affine and projective transformations.

It was also seen that for image recognition tasks, CNNs are popularly used, which extract features from images through their convolutional filters, which are basically sliding windows that perform linear operations in a weighted manner on the pixels as they pass through the image. A technical limitation of these filters is that they can only provide translation invariance.

In this work, we propose using ViT to process images in the polar domain. ViT, a Transformer-based model, treats images as sequences of patches, leveraging multi-head attention to capture relationships and learn hierarchical representations as demonstrated in (DOSOVITSKIY et al., 2020). Unlike previous works, our approach addresses the recognition of deformed images from undeformed GTSRB images under various severity levels of affine and projective deformation.

4 DEVELOPMENT

The solution proposed in this work is based on exploring collinearity, which is a projective invariant, through the use of 1D convolutional filters in the initial layers of images in the polar domain. This process generates patch embeddings that feed into a Transformer Encoder to classify the GTRSB images deformed by affinities and projectivities. This chapter provides a detailed explanation of the proposed ViT model architecture, the regularization techniques and the other decisions made to achieve the goals of this work.

4.1 MULTI-ANGLE-SCALE VISION TRANSFORMER

Our approach begins by converting the images into the polar domain as a preprocessing phase. Subsequently, it augments the data for MASViT's training by applying certain transformations in the polar domain. Post-training, the model can be tested and its performance is further improved using Max Score. This chapter discusses these steps in detail along with the model architecture.

4.1.1 Polar Domain

The conversion of an image from the Euclidean domain into the polar domain can be done by remapping points from the Cartesian coordinate system (x, y) to the polar coordinate system (ρ, θ) as follows:

$$\rho = \sqrt{(x - x_c)^2 + (y - y_c)^2}, \tag{4.1}$$

$$\theta = arctan\left(\frac{y - y_c}{x - x_c}\right) \tag{4.2}$$

where the center of the transform in the image is (x_c, y_c), θ is the angle between the vector $(x - x_c, y - y_c)$ and the horizontal line, and ρ is the size of the vector. For training, we are using the image's center as the polar conversion origin. Some works demonstrate the benefits of rotation invariance for recognition by neural networks using images in polar and log-polar domains (ESTEVES et al., 2018b), (AMORIM et al., 2018), (REMMELZWAAL; MISHRA; ELLIS, 2020), and it is guaranteed by the preservation of collinearity, which is a

more fundamental invariant, thanks to the 1D patches that are processed by convolutional filters with format $1 \times K$ (ignoring the channel dimension), where K is the horizontal size of the filter. These filters, working in the polar domain, are more powerful than the traditional squared filters in the Euclidean domain, as they make the filters not only translation equivariant, but also rotation equivariant. Patterns identified by a filter at different angles tend to result in the same correlation because changing the angle just requires moving the filter to a different row in the polar domain.

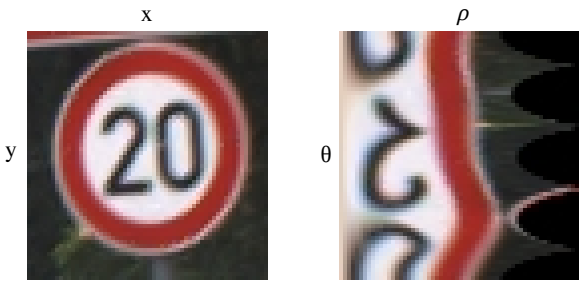


Figura 20 – A GTSRB sample in Euclidean and Polar domains. Source: The author.

Two potential issues may arise from the mapping between Euclidean and polar domains: numerical errors due to pixel value truncation, and the fact that equidistant points can be mapped to the same ρ value. Regarding the first, the truncation introduces only minor and uniform approximations that do not significantly affect collinearity. Since the method relies on global consistency rather than absolute pointwise precision, the overall impact on the results is negligible. Concerning the second, although equidistant points along the central line collapse to the same ρ, this does not pose a practical problem, as such points are geometrically equivalent with respect to the center. Furthermore, the implementation addresses this case deterministically, ensuring no ambiguity in the representation.

4.1.2 MASViT Architecture

Similarly to any ViT architecture, the MASViT uses a Transformer Encoder that receives as input a 1D sequence of token embeddings, each one corresponding to a W-pixel-wide patch of a polar image with H such patches, i.e., its dimensions are $H \times W \times C$ (C channels), with the angles varying in H dimension. That means that each patch has $1 \times W \times C$ format, and is processed by 1D convolutional filters in several layers, generating

a sequence of embeddings representing a natural angle sequence.

This process is illustrated in Fig. 21: The model is fed with 48×48-RGB polar images partitioned in 48 1×48-RGB rows. These are processed initially by a set of four convolutional layers and a pooling layer (oriented horizontally), the latter set with kernel format $1 \times W$, which generates $48 \times 1 \times 512$-tensor data. Next, it follows a set of four convolutional layers and a pooling layer with kernel format $H \times 1$ (oriented vertically). This last set of layers was designed to reduce the height dimension of the tensor data, delivering to the Transformer Encoder a sequence of 9 embeddings instead of 49, as it turns out a smaller sequence of embeddings generalizes better, according to the results obtained in our training phase. Between the layers, there are also activation functions (ReLU) and batch normalization layers, which are better detailed in Table 3.

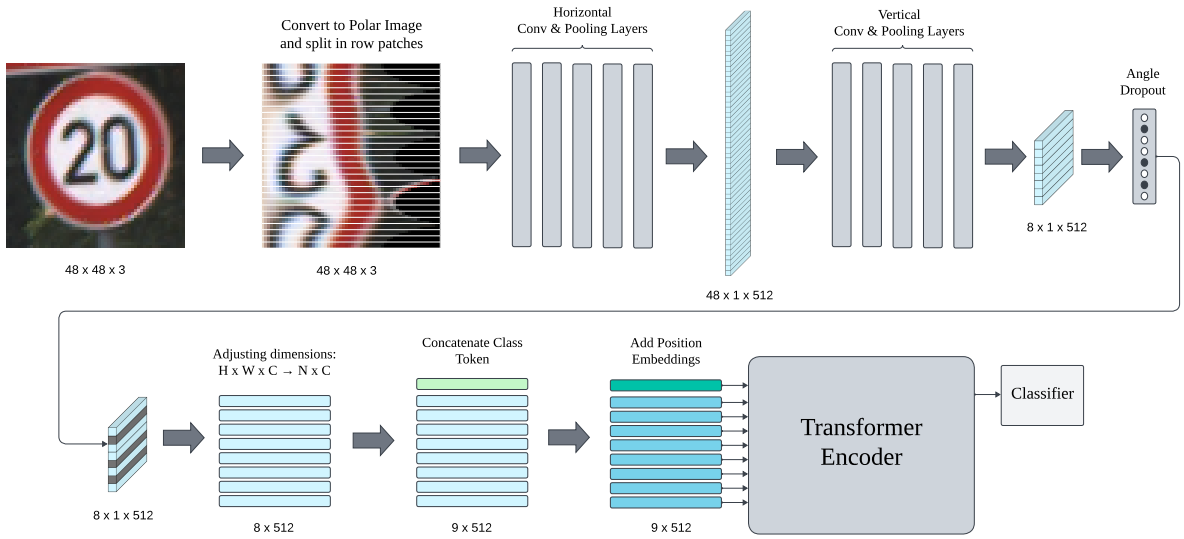


Figura 21 – An overview of the MASViT architecture. Source: The author.

4.1.2.1 Horizontal and Vertical Layers

Table 3 describes the Horizontal and Vertical layers of MASViT. The Horizontal layers contain convolutional and max-pooling layers with kernels of size $(1, w)$, similarly the Vertical layers contain same layers with kernels of size $(h, 1)$. Each convolutional layer is followed by a batch normalization layer and a ReLU activation function. ReLU is widely preferred because of its simplicity, its non-saturating behavior for positive inputs, and its effectiveness in mitigating the vanishing gradient problem.

Tabela 3 – Horizontal and Vertical Layers Settings

Layer	Kernel	Stride	In Channel	Out Channel
Conv2D	(1,1)	(1,1)	3	64
BatchNorm2D	–	–	64	64
ReLU	–	–	64	64
Conv2D	(1,3)	(1,1)	64	128
BatchNorm2D	–	–	128	128
ReLU	–	–	128	128
Conv2D	(1,5)	(1,1)	128	256
BatchNorm2D	–	–	256	256
ReLU	–	–	256	256
MaxPool2D	(1,7)	(1,7)	256	256
Conv2D	(1,6)	(1,6)	256	512
BatchNorm2D	–	–	512	512
ReLU	–	–	512	512
Conv2D	(3,1)	(1,1)	512	512
BatchNorm2D	–	–	512	512
ReLU	–	–	512	512
Conv2D	(3,1)	(1,1)	512	512
BatchNorm2D	–	–	512	512
ReLU	–	–	512	512
Conv2D	(3,1)	(1,1)	512	512
BatchNorm2D	–	–	512	512
ReLU	–	–	512	512
Conv2D	(3,1)	(1,1)	512	512
BatchNorm2D	–	–	512	512
ReLU	–	–	512	512
MaxPool2D	(5,1)	(5,1)	512	512

4.1.2.2 Angle Dropout

The Angle Dropout layer is a regularization technique introduced in this work. This module, which follows the vertical convolutional/pooling layers, implements a specific type of dropout that randomly selects certain angles or segments of the input tensors along the height dimension and sets them to zero based on a dropout probability p (we use $p = 0.5$). This can be useful for introducing robustness or regularization during the training of Transformer models, especially in scenarios where the sequence data structure

has a dimension that can be interpreted as meaningful "angles"or "segments".

Note that the angle dropout and traditional dropout are not the same operation, the angle dropout is a custom implementation of dropout that zeros out elements along the height dimension of an input tensor. Unlike traditional dropout, which zeros elements independently, angle dropout specifically regularizes along one dimension. Compared to spatial dropout, angle dropout introduces a distinct geometric inductive bias. Although it may superficially resemble masking along lines as in spatial dropout, angle dropout operates on oblique directions rather than axis-aligned ones. This angular masking effectively simulates geometric distortions in feature space, something standard spatial dropout does not capture. As a result, angle dropout not only provides regularization, but also functions as a structured form of feature-level augmentation, better aligning the model with the geometric variations encountered in real-world data. Angle Dropout pseudocode is shown in Algorithm 1, where B is the batch size dimension.

Algorithm 1 Angle Dropout - Pseudocode

Require: Input Tensor x with shape (B, C, H, W),
number of angles $A = \texttt{num\_angles}$,
dropout probability p

Ensure: Tensor with dropout applied along the height

if training mode **then**
  Get $(B, C, H, W) \leftarrow \texttt{shape}(x)$
  Create mask `mask_angle` with shape $(B, 1, A, 1)$
    where each input has probability p of being zero
  Repeat `mask_angle` along channels:
    $\texttt{mask\_angle} \leftarrow \texttt{repeat}(1, C, 1, 1)$
  Expand mask along width:
    $\texttt{mask\_data} \leftarrow \texttt{expand}(B, C, A, W)$
  Repeat blocks of height to cover H:
    $\texttt{mask\_data} \leftarrow \texttt{repeat}(1, 1, 1, H/A)$
    $\texttt{mask\_data} \leftarrow \texttt{reshape}(B, C, H, W)$
  Apply mask:
    $x \leftarrow x \cdot \texttt{mask\_data}$
end if
return x

4.1.2.3 Class Token and Positional Embeddings

Similarly to what the authors use in the original ViT paper (DOSOVITSKIY et al., 2020), we use a class token and positional embeddings, the class token is a learnable

special vector inserted into the input of the Vision Transformer that, after processing, serves as a global representation of the image and is used for the final classification. After the patch embeddings are generated and their dimensions adjusted from $H \times W \times C$ to $N \times C$, where N is the embeddings sequence length; a class token tensor of size 1×512 is concatenated with a sequence of embeddings in the first position to perform classification during processing within the Transformer Encoder; these resulting patch embeddings are summed with positional embeddings tensor of the same size (9×512) to retain spatial information. These extra learnable parameters work together to help the ViT models provide complete image information.

4.1.2.4 Transformer Encoder and Classifier settings

Our Transformer Encoder (TE) supports patches of size 512 for a sequence of embeddings with 9 elements. The TE contains 256 attention heads, MLPs with 512 neurons with Gaussian error linear units (GELU) as activation functions (HENDRYCKS; GIMPEL, 2016), and MLP dropout set to 50%. The number of transformer layers is eight. Following processing within the TE, the processed class token feeds into the Classifier, which consists of a Layer Norm followed by a Linear layer with an input size of 512 and an output size of 43 (number of GTSRB classes).

4.2 DATA AUGMENTATION IN POLAR DOMAIN

Although we are processing the image with convolutional filters in a one-dimensional format, satisfying the collinearity invariance, it is still necessary to use data augmentation concerning scaling, as convolutional filters are equivariant to translation, but not to scale. There is also a need for cyclic variation in polar domain of the sequences passed to the TE, as this module does not present cyclic invariance properties. This operation would be equivalent to a rotation in an Euclidean space. The purpose of this unconventional data augmentation method is to emulate a rotation and a scale, but using fewer transformation parameters and variation values.

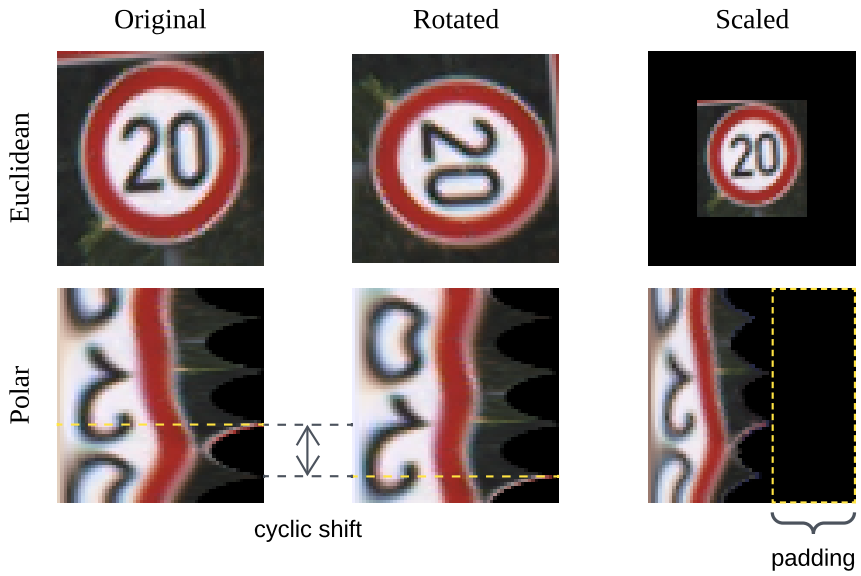


Figura 22 – Rotation in the Euclidean domain causes a cyclic shift in the θ axis in the Polar domain, and a change of scale in the Euclidean domain causes a change of scale in the ρ axis in the Polar domain. A similar effect can be obtained by padding blanks on the right side of the polar image and re-dimensioning it back to the original size. Source: The author.

4.2.1 Cyclic-Angular Shifting.

This operation performs a circular shift vertically in the input image, which means that it varies angles cyclically in polar domain. It is equivalent to a rotation in the Euclidean domain, as shown in Fig. 22. It generates a random integer h_s, between 0 and $h-1$, where h is the height of the image, and shifts the content of the polar image cyclically along the height dimension by h_s. This Cyclic-Shift operation uses only one integer parameter (h_s) which, in our case, $h_s \in \{0, 1, ..., 47\}$.

4.2.2 Right-Side Padding

This operation pads blanks on the right side of the polar images, with different padding sizes randomly chosen (w_p). After padding, the function resizes the image back to its original dimensions. This is equivalent to scaling in the Euclidean domain as shown in Fig. 22. This operation resizes the ρ variable in the polar coordinates, i.e., it operates horizontally in the polar image, which is equivalent to a radial resize in the Euclidean domain. The Right-Side Padding in the polar image uses only one integer parameter (w_p) which, in our case, $w_p \in \{0, 1, ..., 40\}$.

Performing data augmentation with fewer parameters that take discrete values is, whe-

never applicable, preferable if compared to using transformations that have more degrees of freedom, each one allowing a much wider range of variations. In this case, isometries, similarities, affinities, and projectivities have respectively three, four, six, and eight continuously varying parameters, most of them unlimited. This adopted data augmentation in the polar domain, allied to a model that explores the collinearity invariance, reduces it to just two parameters with limited discrete ranges.

4.3 POST TRAINING BOOST BY MAX SCORE

We present a novel technique designed to enhance the accuracy of our post-training model. As discussed earlier, our model takes polar images as input, requiring a specified center of transformation, (x_c, y_c). During training, we use the center of the original image for this purpose. However, to further explore model inference, we can generate polar images using different transformation centers, referred to as *centroids*.

This approach allows us to create multiple polar images from a single $H \times W$ image in the Euclidean domain, through centering the polar conversion in various centroids. This multitude of polar images (specifically 48×48 images) is subsequently fed into the model for inference. Each polar image generated from a unique centroid yields a score vector, with each entry in the vector related to a class in the model's classification (totaling 43 classes). Consequently, this results in 43 score maps for each original Euclidean domain image, Fig.23 show this process.

The significance of this technique lies in its ability to provide diverse perspectives on the same scene, enabling the identification of regions of high scores within each classification map. Ultimately, by analyzing these score maps, we can select the point of maximum score across the 43 maps as the optimal model inference for the given scene. This approach enhances the robustness and accuracy of our model by leveraging multiple viewpoints derived from a single input image.

This multi-centroid approach is more robust than just a single polar image with a fixed centroid, as deformed objects (ex.: traffic signs) hardly have their original centers coinciding with the center of their images. By training the model with polar images whose conversion is centered in different centroids, it is expected that the max-score centroid will be the closest to the actual center of the object, regardless of the deformation suffered by the input image. Therefore, there are two approaches: One that uses a single polar image

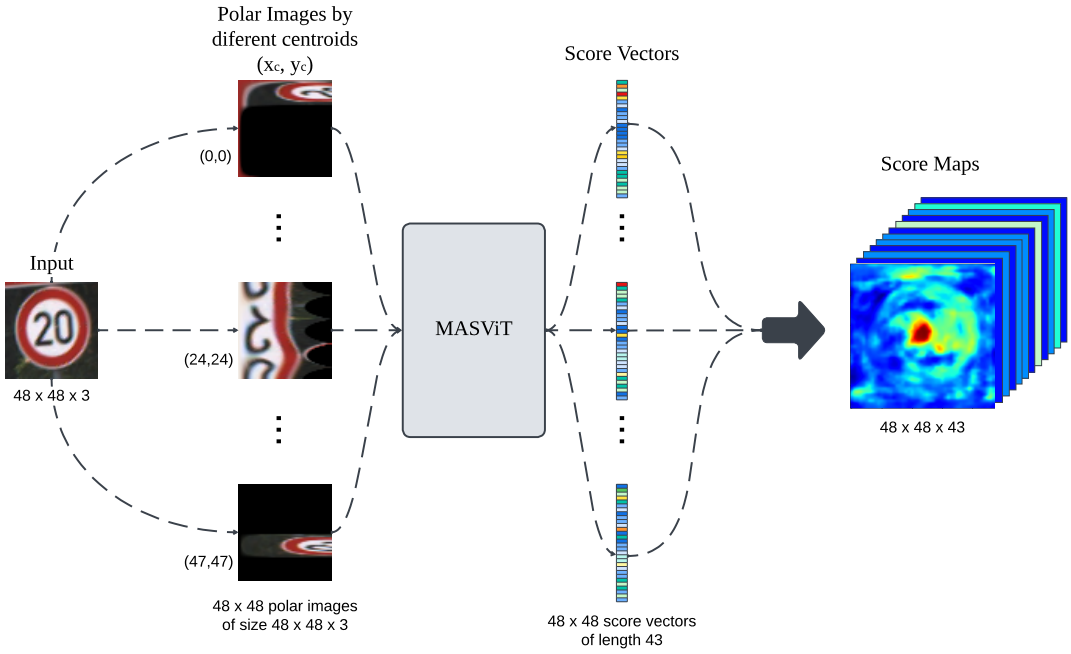


Figura 23 – Score Maps Generation. Source: The author.

through a fixed centroid, where a single score vector returned by the model is verified; and the one that makes inferences based on the maximum score by analyzing the score maps generated by spatial exploration based on different centroids.

4.4 DISCUSSION

In this chapter, we presented the development of the proposed approach, which allows a Vision Transformer to offer invariance to affine and projective transformations in classification tasks. The network receives as input images transformed to the polar domain, which, combined with the use of one-dimensional convolutional filters, facilitates the extraction of features invariant to projections, such as collinearity.

The architecture of MASViT was detailed, addressing the creation of patch embeddings through unidimensional convolutional and pooling layers, the introduction of the Angle Dropout module, and the use of class tokens and positional embeddings. All this processing is then passed to the Transformer Encoder, followed by a final classification step, as is common in ViTs aimed at classification tasks. We adopted a dropout rate of 50% for both the Angle Dropout and the standard Dropout, as this value is a widely accepted default for fully connected layers. It provides a good balance between underfitting and overfitting, encouraging robust feature learning and effectively reducing overfitting,

as demonstrated in the original Dropout paper (SRIVASTAVA et al., 2014). Implementation details of MASViT and the Angle Dropout module are provided in Appendix A. The code was developed in Python using the PyTorch framework.

The use of data augmentation in the polar domain was also discussed, in which two techniques aimed at scale and rotation variations were proposed. These techniques benefit from the fact that they operate in discrete domains, which reduces the number of variation parameters. It is important to highlight that this work did not use data augmentation based on affine or projective geometric transformations. Instead, it adopted simpler transformations, emulated through the Cyclic-Angular Shifting and Right-Side Padding techniques.

Finally, a post-training boost mechanism was introduced, based on the variation of the center of the conversion to the polar domain. This procedure consists of selecting the version of the image whose classification output obtained the highest score, promoting accuracy improvements without the need for reprocessing or complete retraining of the network.

5 EXPERIMENTS

In this chapter, we detail the experiments conducted, including descriptions of the datasets used for training, validation, and testing. The two test datasets consist of images distorted by affine and projective transformations at varying levels of severity. We also present the experimental setup and hyperparameters adopted, along with the results obtained using two MASViT configurations: fixed-centroid and max-score. Finally, we compare the results against two state-of-the-art models based on capsule networks and spatial transformers.

5.1 DATASETS

In this study, we utilized the GTSRB introduced by (STALLKAMP et al., 2012) for training and validation. The original GTSRB Train set comprised 39,209 samples for training, and the original GTSRB Test set comprised 12,630 samples for validation. To evaluate MASViT's robustness for recognizing affine and projective deformations across different severity levels, we introduced aff-GTSRB and proj-GTSRB datasets groups, each representing distinct deformation categories.

The generation of these test datasets with incremental deformation levels followed the methodology outlined in (LI; TAN, 2010) and (CHARAMBA; MELO; LIMA, 2021), which consists of each planar shape being captured by a camera whose projection plane orientation is controlled by adjustments in the elevation (el) and azimuth (az) angles. These adjustments produce deformations with different levels of severity. The datasets aff-GTSRB and proj-GTSRB were derived similarly, with each dataset subgroup stemming from the original GTSRB Test samples and categorized according to four elevation angles: $90°, 60°, 45°$, and $30°$. Each subgroup was further subdivided into eight partitions according to the following azimuth angle variations: $0°, 15°, 45°, 60°, 90°, 120°, 190°$, and $275°$. Consequently, each new dataset produced a total of $12,630 \times 8 = 101,040$ images derived from the original test samples. Figure 24 illustrates examples of the new samples generated from an original test traffic sign (50) with varied elevation and azimuth angles within the aff-GTSRB and proj-GTSRB groups.

The distinction between the two dataset groups lies in the type of camera employed.

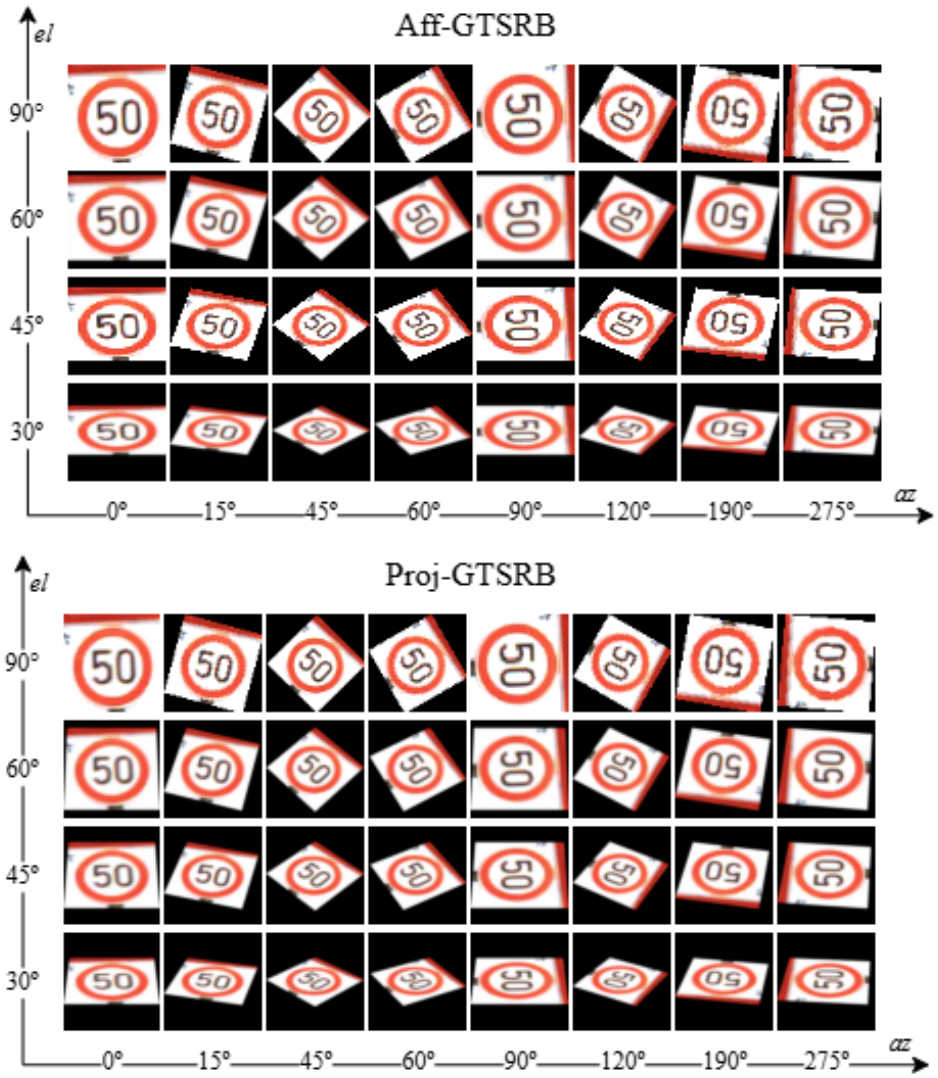


Figura 24 – Aff-GTSRB and Proj-GTSRB: The new datasets were obtained from original GTSRB Test samples by view point angle variations (elevation and azimuth); and the two groups differ in terms of camera projection (orthographic and perspective). Source: The author.

The aff-GTSRB dataset utilizes orthographic projection from an affine camera, whereas the proj-GTSRB dataset employs conventional perspective projection. One notable difference is that samples from the first group maintain parallel lines, whereas in the projective version, this is not the case.

5.2 TRAINING

We trained MASViT using the original GTSRB Train samples converted to the polar domain centered at (24,24) and applied data augmentation techniques within this domain that correspond to varying scale and rotation severity in Euclidean space, as explained

in section 4.2. Therefore, there are no affine or projective deformations in this data augmentation. Additionally, we performed pixel-level augmentation by randomly adjusting the brightness, contrast, and saturation of the images.

We used the Adam optimizer, known for its ability to handle sparse gradients and accelerate convergence (RUDER, 2016). The initial learning rate was set to 10^{-4} with an exponential decay factor of 0.99. These choices were based on the work of (KINGMA; BA, 2015), where the Adam optimizer was introduced, suggesting that typical learning rates range from 10^{-3} to 10^{-4}. The authors also claim that exponential decay helps adjust the learning rate over time, improving convergence stability.

We used Cross Entropy as the loss function, which is effective for classification tasks according to (GOODFELLOW; BENGIO; COURVILLE, 2016). Samples were fed into the model in batches of size 16 over 300 epochs. We chose small batches because it has been observed in practice that while larger batches can be efficient for training, they often result in poor generalization, as demonstrated in (KESKAR et al., 2016) and (HOFFER; HUBARA; SOUDRY, 2017).

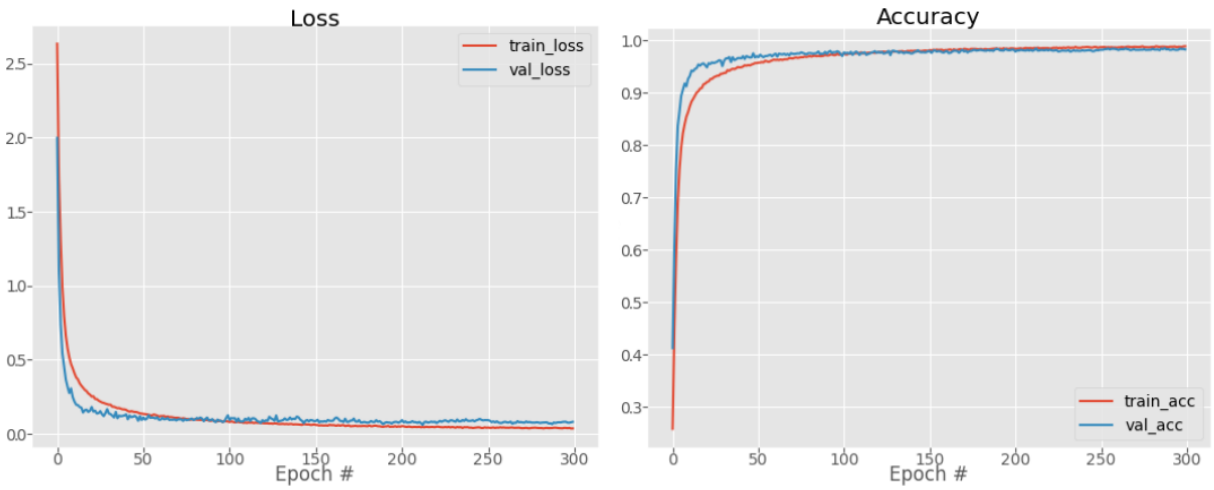


Figura 25 – Loss and Accuracy curves for training (in red) and validation (in blue) data. Source: The author.

During training, we employed the original GTSRB Test dataset for validation purposes, monitoring both accuracy and loss values. The model achieved a validation accuracy of 98.31% in the final epoch. Fig. 25 depicts the plots illustrating the curves of loss and classification accuracy values.

5.3 TESTING

By comparing MASViT to other state-of-the-art GTSRB classifier models, we assessed its performance against models like the one by Arcos-García et al. (ARCOS-GARCÍA; ALVAREZ-GARCIA; SORIA-MORILLO, 2018), which utilizes a CNN that alternates convolutional and Spatial Transformer modules, achieving an accuracy of 99.71%; and the one by Chen et al. (CHEN et al., 2024), based on capsules that use a global routing mechanism, boasting the highest accuracy of the original test samples at 99.96% in their experiments. Additionally, we tested MASViT for GTSRB Test also, using two distinct approaches: Fixed Centroid and Max score, as discussed in Section 4.3. The resulting accuracy values were 98.31% and 97.95%, respectively.

Regarding the Max Score approach, there is an assumption we took advantage of to improve the performance: The centers of the planar shapes in the GTSRB data are not far from the center of the images. Thus, we constrained the positioning of the tested centroids to a neighborhood of the image center. We used a 7×7 grid to generate 49 polar images per Euclidean image instead of all 48×48 pixels. This was an important assumption given the multiplicative cost each new polar domain imposes on our method.

Working with the results from the GTSRB test samples as a starting point, we conducted tests with aff-GTSRB and proj-GTSRB datasets. Table 4 displays the accuracy values of all tested models for the two groups of datasets, subdivided according to the elevation angle. It is important to note that at $el = 90^{o}$, it represents a frontal view, where the camera projection plane is parallel to the planar shape, resulting in no shape transformation other than rotations. Therefore, the aff-GTSRB $el = 90^{o}$ and proj-GTSRB $el = 90^{o}$ datasets are identical, thus producing identical results, as observed in the two columns with this elevation angle.

As the elevation angle decreases, deformations become more severe, negatively impacting the performance of the different tested methods. For elevation angles below 90^{o}, a difference between the two dataset groups begins to be noticed, with the negative impact in the performance being slightly greater in the proj-GTSRB datasets, due to the higher generality of the projective transformations.

Analyzing the different methods, the models proposed in (ARCOS-GARCÍA; ALVAREZ-GARCIA; SORIA-MORILLO, 2018) and (CHEN et al., 2024) did not perform well, despite achieving excellent results when confronted with original GTSRB Test samples. Even

for the mildest case of transformation (el =90º), the results were significantly below expectations. At this elevation, images are distorted only by azimuth variations, resulting in simple rotations. Although designed to handle spatial transformations, Capsule Networks and Spatial Transformers still do not guarantee more severe transformations without having seen such levels of deformations during training. Capsule Nets can encode pose information and generalize better than standard CNNs, but their local part–whole relationships tend to break under strong distortions, especially those not observed during training. Spatial Transformers, in turn, learn global alignment functions and can normalize moderate variations, but they only compensate for transformations that are present in the training data, offering no intrinsic invariance. In both cases, robustness depends on encountering representative distortions during learning. MASViT overcomes these limitations by leveraging polar-domain representations and 1D convolutional filters that preserve the collinearity, a projective invariant, thus enabling generalization even to unseen affine and projective deformations.

About the two MASViT approaches, Fixed Centroid and Max Score, in the original GTSRB Test and el =90º (for aff-GTSRB and proj-GTSRB) the Fixed Centroid approach had better performance than Max score in these situations, but when the elevation angle is decreased the Max score approach becomes a better approach, although the Max Score technique searches a 7×7 neighborhood around the image center, its lower accuracy compared to the Fixed Centroid is due to sensitivity to local fluctuations or spurious activations. Max Score selects the position with the highest score value, which can occasionally misinterpret local maxima, whereas the Fixed Centroid relies on a stable geometric center, which in datasets like original GTSRB and its deformed variants in elevation 90° is typically well aligned with front-view traffic signs, providing more consistent and accurate results in these situations. In the most severe deformation (el =30°), where the biggest difference between the approaches occurs, the Max Score overcomes the Fixed Centroid by approximately 10 percentage points in the two transformation groups.

By employing the Max Score approach, the score maps can be visualized, offering deeper insights into the underlying behavior and decision process of the method. Fig. 26 illustrates several test images (six for each group) at different elevation angles and randomly chosen azimuth angles of different categories, along with their score maps and an overlapped representation of these images. It can be observed that the maximum value tends to concentrate at the center of the signs in most cases. Examples of mismatches can

Tabela 4 – Accuracy of Methods in percentage values.
(*) (ARCOS-GARCÍA; ALVAREZ-GARCIA; SORIA-MORILLO, 2018)
(**)(CHEN et al., 2024)

	Aff-GTSRB				**Proj-GTSRB**			
Methods	el=90º	el=60º	el=45º	el=30º	el=90º	el=60º	el=45º	el=30º
MASViT (fixed centroid)	**97.27**	95.81	93.16	66.71	**97.27**	95.58	91.79	64.04
MASViT (max score)	97.11	**96.00**	**94.49**	**76.68**	97.11	**95.67**	**93.76**	**73.93**
CNN + 3 Spatial Transformers *	38.46	35.39	32.38	21.47	38.46	34.76	29.75	19.79
Global Routing CapsNet **	33.40	29.48	25.47	12.32	33.40	28.68	22.34	11.06

also be observed in Fig.27: the score maps for the true and predicted classes are illustrated. In the case of the "turn left" traffic sign, the true score map presents maximum activation concentrated near the center of the sign; however, the predicted class shows a stronger and more diffuse activation in another region of its score map, ultimately leading to misclassification.

In addition to Table 4, which only shows the accuracy, there are several tables in the Appendix B that show in detail the precision, recall and F1-score per class of the two MASViT approaches, for all datasets of test. These tables are in **Classification Reports**, Section B.1. Analyzing these tables, it becomes evident that certain classes, such as the first and 15th, experience a more pronounced decline in recall compared to other categories as distortions increase (e.g., elevation = 30°). This low recall can be primarily attributed to the class imbalance in the GTSRB dataset. Several classes are substantially underrepresented in the training set, which restricts the model's ability to generalize under severe deformations. For instance, the 15th category, class 14 (STOP), has only 780 training samples, and class 0 just 210, whereas high-recall classes such as class 2 have more than 2000. This disparity likely explains the poorer performance of minority classes, particularly under challenging conditions such as elevation equals to 30°. Moreover, the scarcity of training samples may have reduced the effectiveness of the model's regularization and generalization mechanisms, further amplifying the decline in recall for these underrepresented classes.

In addition to the Classification Reports, there are also several tables describing the Confusion Matrices of the test experiments exhibited in Section C.

And in Section D.1, **Computing Infrastructures for Experiments**, of Appendix D, the hardware specifications used for all experiments (training and testing) carried out in this work are presented, along with the corresponding execution times, and number of parameters and inference times for all models.

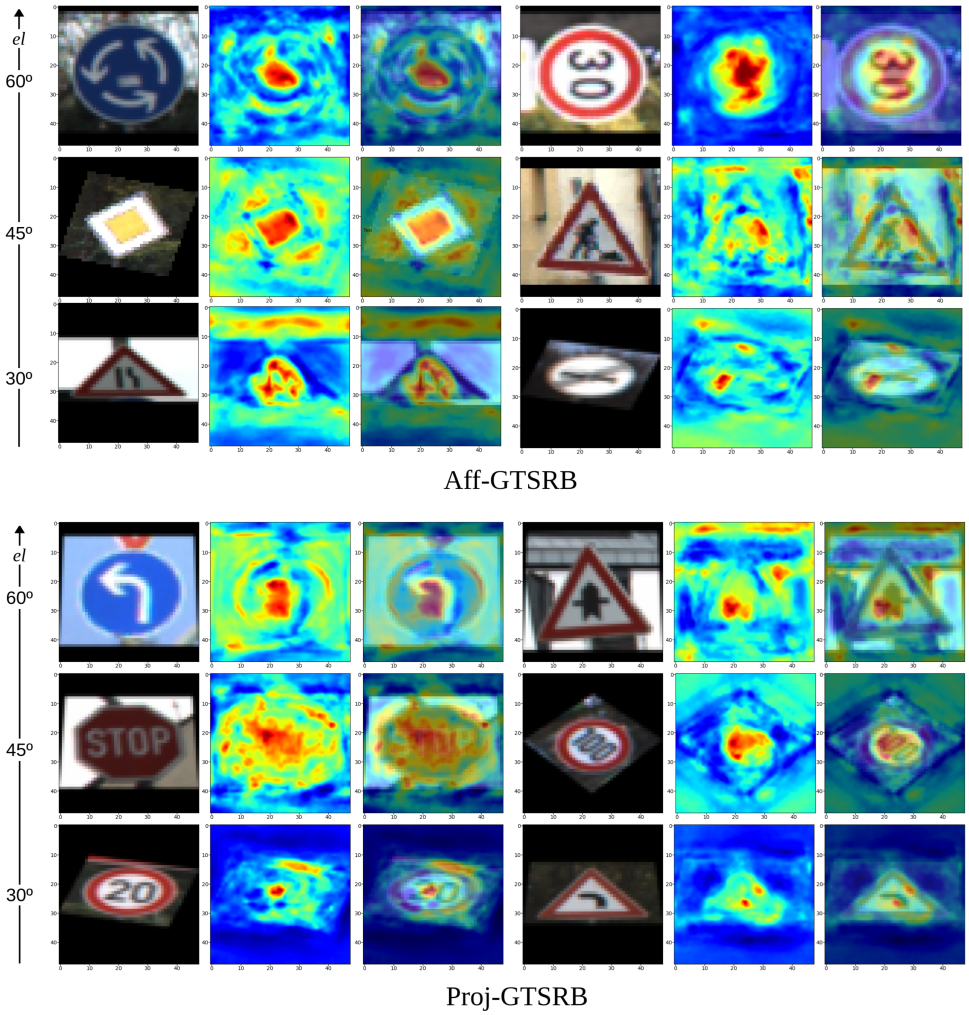


Figura 26 – Some samples from the datasets (Aff-GTSRB and Proj-GTSRB) at different elevation and azimuth angles, next to each test image there is a score map specific to the sign category and an overlaid representation of the map over the test image. Source: The author.

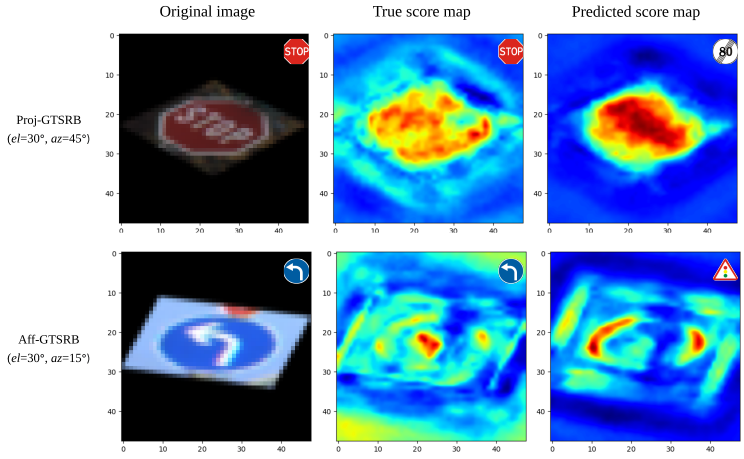


Figura 27 – Mismatch examples and their corresponding score maps. The left column presents a sample from each test dataset group (Aff-GTSRB and Proj-GTSRB) at an elevation of 30°; the middle column displays the true score maps; while the right column presents the predicted ones. Each image and score map includes a traffic sign icon in the upper-right corner, corresponding to its class.

5.4 DISCUSSION

To evaluate MASViT's robustness to geometric deformations in this study, the aff-GTSRB and proj-GTSRB test sets were introduced, derived from the original GTSRB by applying affine and projective transformations. These were generated using controlled variations in elevation and azimuth angles, simulating increasing deformation severity. While aff-GTSRB uses orthographic projection, proj-GTSRB introduces more complex distortions via perspective projection. This setup allows a systematic assessment of model performance under realistic, challenging conditions.

The training procedure adopted prioritized robustness to rotation and scale by performing data augmentation directly in the polar domain, which naturally encodes such transformations as translations. This decision enabled MASViT to learn invariant features without the need for explicit affine or projective augmentations. The use of pixel-level adjustments, such as brightness, contrast, and saturation, contributed to enhancing the model's generalization under different lighting conditions, while the small batch training strategy favored better generalization.

Despite not achieving top-tier accuracy on the original GTSRB Test dataset compared to capsule- and spatial transformer-based models, MASViT demonstrated remarkable resilience under geometric distortions, especially in the presence of affine and projective transformations. When tested on aff-GTSRB and proj-GTSRB datasets, MASViT main-

tained competitive performance where other models significantly degraded. This result highlights MASViT's core strength: its robustness to shape-preserving and perspective deformations, which is crucial for real-world traffic sign recognition scenarios where camera viewpoints may vary widely.

A key insight emerged from the comparison between the two post-training strategies: Fixed Centroid and Max Score. The Fixed Centroid approach performed better under ideal or near-ideal conditions, such as the original GTSRB and $el = 90°$ datasets, where signs are centered and minimally distorted. However, in more challenging scenarios with increased deformation (lower elevation angles), the Max Score strategy proved to be significantly more effective. This adaptive behavior underscores the importance of allowing the model to explore multiple centroids, especially when signs are displaced or severely deformed due to camera perspective.

6 CONCLUSION AND FUTURE WORK

This work introduced an innovative deep-learning-based solution, the Multi-Angle-Scale Vision Transformer, designed to enhance robustness in image recognition tasks influenced by projective and affine transformations. It is based upon a ViT architecture that leverages collinearity as a geometric invariant to affinities and projectivities across different angles and scales within images. Additionally, modifications to the transformer architecture and two new GTSRB-based datasets were introduced, offering new references for future work.

Our evaluation of MASViT against state-of-the-art GTSRB classifier models showcased its robust performance across various deformation scenarios. While the concurrent models demonstrated high accuracy on original GTSRB test samples, their performance significantly declined when exposed to deformed images, particularly those with severe deformations, it is important to highlight that these models, based on Spatial Transformers and Capsules, they have architectures that are also designed for invariance to geometric transformations, to enhance the recognition of distorted shapes; however, they underperformed. Our analysis revealed that MASViT, employing both Fixed Centroid and Max Score approaches, exhibited promising adaptability to deformation challenges. Notably, the Fixed Centroid approach showed slightly superior performance on original GTSRB test samples and mild deformations, while the Max Score approach was proved more effective as the severity of deformations increased, outperforming the Fixed Centroid with a large difference in the most severe deformation scenarios, at the cost of a longer processing time. In summary, the Max Score method, while more flexible, is sensitive to local fluctuations and may misinterpret spurious activations, while the stable geometric center of Fixed Centroid aligns better with the traffic signals in front, producing more consistent accuracy in front view samples.

In summary, MASViT demonstrates robustness in handling diverse deformation cases, with a training phase based on non-deformed images, with respect to affinities and projectivities. This showcases its potential for real-world applications where image recognition must contend with varying degrees of affine and projective distortions. The concentration of maximum values in the center of the traffic signs, as observed in score maps, further underscores the efficacy of MASViT in capturing relevant features despite deformation

challenges, highlighting its significance in advancing the field of image recognition under adverse conditions.

In terms of application scope, MASViT is a model built upon the ViT architecture and therefore inherits both its strengths and limitations. While the original ViT was primarily designed for image classification, MASViT was also initially developed with this goal in mind. Nevertheless, as ViT variants have been successfully extended beyond classification through architectural modifications, MASViT could similarly be adapted for tasks such as object detection and semantic segmentation, albeit requiring additional adjustments. Regarding the datasets used in this thesis, although MASViT was evaluated exclusively on GTSRB and its geometrically deformed variants (aff-GTSRB and proj-GTSRB), its applicability extends beyond traffic sign recognition to the broader recognition of planar shapes, including characters, logos, and other visual patterns.

MASViT opens up a range of possibilities for future works including an investigation into methods for incorporating scale invariance during patch embedding via convolutional filters. This would eliminate the necessity for scaling variations in the data. Proposing a new Transformer Encoder that is intrinsically cyclically invariant removes the need for circular shifts in the data. Although using a single centroid during training is sufficient to achieve a robustness to recognize deformed samples, future studies can explore the impact of varying the number of centroids on performance. Another room for improvement may be the refinement of the Max Score approach to enable the identification of better regions beyond just the maximum score point, enhancing robustness. This score maps approach can also be used to infer the projectivity matrix that produced the image deformation: At least four MASViT networks can be trained with images in polar domains based on different centroids. Given an image to be tested, its distinct maximum points on the corresponding score maps can be taken as the key points' correspondences to the used centroids, allowing us to infer the projectivity matrix entries. This may be used in several applications, including those that benefit from using a traffic sign as a base to infer other planar shapes present in the scene.

REFERENCES

AMORIM, M.; BORTOLOTI, F.; CIARELLI, P. M.; OLIVEIRA, E. de; SOUZA, A. F. de. Analysing rotation-invariance of a log-polar transformation in convolutional neural networks. In: IEEE. *2018 International Joint Conference on Neural Networks (IJCNN)*. [S.l.], 2018. p. 1–6.

ARCOS-GARCÍA, Á.; ALVAREZ-GARCIA, J. A.; SORIA-MORILLO, L. M. Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods. *Neural Networks*, Elsevier, v. 99, p. 158–165, 2018.

AZULAY, A.; WEISS, Y. Why do deep convolutional networks generalize so poorly to small image transformations? *Journal of Machine Learning Research*, v. 20, n. 184, p. 1–25, 2019.

BALLARD, D. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, v. 13, n. 2, p. 111 – 122, 1981. ISSN 0031-3203. Available at: <http://www.sciencedirect.com/science/article/pii/0031320381900091>.

BAY, H.; ESS, A.; TUYTELAARS, T.; GOOL, L. V. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, v. 110, n. 3, p. 346 – 359, 2008. ISSN 1077-3142. Similarity Matching in Computer Vision and Multimedia. Available at: <http://www.sciencedirect.com/science/article/pii/S1077314207001555>.

BIANCO, S.; BUZZELLI, M.; MAZZINI, D.; SCHETTINI, R. Deep learning for logo recognition. *Neurocomputing*, Elsevier, v. 245, p. 23–30, 2017.

BISCIONE, V.; BOWERS, J. S. Convolutional neural networks are not invariant to translation, but they can learn to be. *Journal of Machine Learning Research*, v. 22, n. 229, p. 1–28, 2021.

BOX, G. E. Science and statistics. *Journal of the American Statistical Association*, Taylor & Francis, v. 71, n. 356, p. 791–799, 1976.

BRONSTEIN, M. M.; BRUNA, J.; COHEN, T.; VELIČKOVIĆ, P. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

BRONSTEIN, M. M.; BRUNA, J.; LECUN, Y.; SZLAM, A.; VANDERGHEYNST, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, IEEE, v. 34, n. 4, p. 18–42, 2017.

BUDUMA, N.; BUDUMA, N.; PAPA, J. *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms*. [S.l.]: "O'Reilly Media, Inc.", 2022.

CHARAMBA, L. G.; MELO, S.; LIMA, U. de. Cross ratio arrays: A descriptor invariant to severe projective deformation and robust to occlusion for planar shape recognition. *Computers & Graphics*, Elsevier, v. 100, p. 54–65, 2021.

CHEN, R.; SHEN, H.; ZHAO, Z.-Q.; YANG, Y.; ZHANG, Z. Global routing between capsules. *Pattern Recognition*, Elsevier, v. 148, p. 110142, 2024.

COHEN, T.; WELLING, M. Group equivariant convolutional networks. In: PMLR. *International conference on machine learning*. [S.l.], 2016. p. 2990–2999.

DOSOVITSKIY, A.; BEYER, L.; KOLESNIKOV, A.; WEISSENBORN, D.; ZHAI, X.; UNTERTHINER, T.; DEHGHANI, M.; MINDERER, M.; HEIGOLD, G.; GELLY, S. et al. An image is worth 16x16 words: Transformers for image recognition at scale. In: *International Conference on Learning Representations*. [S.l.: s.n.], 2020.

ESTEVES, C.; ALLEN-BLANCHETTE, C.; MAKADIA, A.; DANIILIDIS, K. Learning so (3) equivariant representations with spherical cnns. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. [S.l.: s.n.], 2018. p. 52–68.

ESTEVES, C.; ALLEN-BLANCHETTE, C.; ZHOU, X.; DANIILIDIS, K. Polar transformer networks. In: *6th International Conference on Learning Representations, ICLR 2018*. [S.l.: s.n.], 2018.

GERKEN, J. E.; ARONSSON, J.; CARLSSON, O.; LINANDER, H.; OHLSSON, F.; PETERSSON, C.; PERSSON, D. Geometric deep learning and equivariant neural networks. *Artificial Intelligence Review*, Springer, v. 56, n. 12, p. 14605–14662, 2023.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: JMLR WORKSHOP AND CONFERENCE PROCEEDINGS. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. [S.l.], 2010. p. 249–256.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN 013168728X.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. [S.l.]: MIT press, 2016.

GU, J.; TRESP, V. Improving the robustness of capsule networks to image affine transformations. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2020. p. 7285–7293.

HARTLEY, R.; ZISSERMAN, A. *Multiple view geometry in computer vision*. UK: Cambridge university press, 2003.

HENDRYCKS, D.; GIMPEL, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

HINTON, G. E.; KRIZHEVSKY, A.; WANG, S. D. Transforming auto-encoders. In: SPRINGER. *Artificial Neural Networks and Machine Learning–ICANN 2011: 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I 21*. [S.l.], 2011. p. 44–51.

HOERL, A. E.; KENNARD, R. W. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, Taylor & Francis, v. 12, n. 1, p. 55–67, 1970.

HOFFER, E.; HUBARA, I.; SOUDRY, D. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *Advances in neural information processing systems*, v. 30, 2017.

HOUBEN, S.; STALLKAMP, J.; SALMEN, J.; SCHLIPSING, M.; IGEL, C. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In: IEEE. *The 2013 international joint conference on neural networks (IJCNN)*. [S.l.], 2013. p. 1–8.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: PMLR. *International conference on machine learning*. [S.l.], 2015. p. 448–456.

JADERBERG, M.; SIMONYAN, K.; ZISSERMAN, A. et al. Spatial transformer networks. *Advances in neural information processing systems*, v. 28, 2015.

JIA; FAN, X.; LIU, Y.; LI, H.; LUO, Z.; GUO, H. Hierarchical projective invariant contexts for shape recognition. *Pattern Recognition*, v. 52, p. 358 – 374, 2016. ISSN 0031-3203.

JIA, Q.; FAN, X.; LUO, Z.; LIU, Y.; GUO, H. A new geometric descriptor for symbols with affine deformations. *Pattern Recognition Letters*, v. 40, p. 128 – 135, 2014. ISSN 0167-8655. Available at: <http://www.sciencedirect.com/science/article/pii/S0167865513004261>.

KAZMI, I. K.; YOU, L.; ZHANG, J. J. A survey of 2d and 3d shape descriptors. In: *2013 10th International Conference Computer Graphics, Imaging and Visualization*. [S.l.: s.n.], 2013. p. 1–10.

KE, Y.; SUKTHANKAR, R. Pca-sift: A more distinctive representation for local image descriptors. In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Washington, DC, USA: IEEE Computer Society, 2004. (CVPR'04), p. 506–513. Available at: <http://dl.acm.org/citation.cfm?id=1896300.1896374>.

KENDALL, A.; GRIMES, M.; CIPOLLA, R. Posenet: A convolutional network for real-time 6-dof camera relocalization. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2015. p. 2938–2946.

KESKAR, N. S.; MUDIGERE, D.; NOCEDAL, J.; SMELYANSKIY, M.; TANG, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. In: *International Conference on Learning Representations*. [S.l.: s.n.], 2016.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. [s.n.], 2015. Available at: <http://arxiv.org/abs/1412.6980>.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2012. v. 25, p. 1097–1105.

KUMAR, D.; SHARMA, D.; GOECKE, R. Feature map augmentation to improve rotation invariance in convolutional neural networks. In: SPRINGER. *Advanced Concepts for Intelligent Vision Systems: 20th International Conference, ACIVS 2020, Auckland, New Zealand, February 10–14, 2020, Proceedings 20*. [S.l.], 2020. p. 348–359.

LECUN, Y.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKEL, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation*, MIT Press, v. 1, n. 4, p. 541–551, 1989.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, IEEE, v. 86, n. 11, p. 2278–2324, 1998. Available at: <http://yann.lecun.com/exdb/mnist/>.

LENC, K.; VEDALDI, A. Understanding image representations by measuring their equivariance and equivalence. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2015. p. 991–999.

LENSSEN, J. E.; FEY, M.; LIBUSCHEWSKI, P. Group equivariant capsule networks. *Advances in neural information processing systems*, v. 31, 2018.

LI, L.; TAN, C. L. Recognizing planar symbols with severe perspective deformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 32, n. 4, p. 755–762, April 2010. ISSN 0162-8828.

LONCARIC, S. A survey of shape analysis techniques. *Pattern Recognition*, v. 31, p. 983–1001, 1998.

LOSHCHILOV, I.; HUTTER, F. Decoupled weight decay regularization. In: *International Conference on Learning Representations*. [s.n.], 2019. Available at: <https://openreview.net/forum?id=Bkg6RiCqY7>.

LOWE, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, v. 60, n. 2, p. 91–110, Nov 2004. ISSN 1573-1405. Available at: <https://doi.org/10.1023/B:VISI.0000029664.99615.94>.

LUO, Z.; LUO, D.; FAN, X.; ZHOU, X.; JIA, Q. A shape descriptor based on new projective invariants. In: *2013 IEEE International Conference on Image Processing*. [S.l.: s.n.], 2013. p. 2862–2866. ISSN 1522-4880.

MACDONALD, L. E.; RAMASINGHE, S.; LUCEY, S. Enabling equivariance for arbitrary lie groups. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2022. p. 8183–8192.

MARCOS, D.; VOLPI, M.; KOMODAKIS, N.; TUIA, D. Rotation equivariant vector field networks. In: *Proceedings of the IEEE International Conference on Computer Vision*. [S.l.: s.n.], 2017. p. 5048–5057.

MARCOS, D.; VOLPI, M.; TUIA, D. Learning rotation invariant convolutional filters for texture classification. In: IEEE. *2016 23rd International Conference on Pattern Recognition (ICPR)*. [S.l.], 2016. p. 2012–2017.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, n. 4, p. 115–133, 1943.

MIKOLAJCZYK, K.; SCHMID, C. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 27, n. 10, p. 1615–1630, Oct 2005. ISSN 0162-8828.

NETZER, Y.; WANG, T.; COATES, A.; BISSACCO, A.; WU, B.; NG, A. Y. Reading digits in natural images with unsupervised feature learning. In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. [S.l.: s.n.], 2011.

NOORD, N. V.; POSTMA, E. Learning scale-variant and scale-invariant features for deep image classification. *Pattern Recognition*, Elsevier, v. 61, p. 583–592, 2017.

REMMELZWAAL, L. A.; MISHRA, A. K.; ELLIS, G. F. Human eye inspired log-polar pre-processing for neural networks. In: IEEE. *2020 International SAUPEC/RobMech/-PRASA Conference*. [S.l.], 2020. p. 1–6.

RIBEIRO, F. D. S.; LEONTIDIS, G.; KOLLIAS, S. Introducing routing uncertainty in capsule networks. *Advances in Neural Information Processing Systems*, v. 33, p. 6490–6502, 2020.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958.

ROUTRAY, S.; RAY, A. K.; MISHRA, C. Analysis of various image feature extraction methods against noisy image: Sift, surf and hog. In: *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. [S.l.: s.n.], 2017. p. 1–5.

RUDER, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group UK London, v. 323, n. 6088, p. 533–536, 1986.

RUSSELL, P. N. Artificial intelligence: A modern approach by stuart. *Russell and Peter Norvig contributing writers, Ernest Davis...[et al.]*, 2010.

SABOUR, S.; FROSST, N.; HINTON, G. E. Dynamic routing between capsules. *Advances in neural information processing systems*, v. 30, 2017.

SRESTASATHIERN, P.; YILMAZ, A. Planar shape representation and matching under projective transformation. *Computer Vision and Image Understanding*, v. 115, n. 11, p. 1525 – 1535, 2011. ISSN 1077-3142. Available at: <http: //www.sciencedirect.com/science/article/pii/S107731421100169X>.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014.

STALLKAMP, J.; SCHLIPSING, M.; SALMEN, J.; IGEL, C. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, Elsevier, v. 32, p. 323–332, 2012.

TAI, K. S.; BAILIS, P.; VALIANT, G. Equivariant transformer networks. In: PMLR. *International Conference on Machine Learning*. [S.l.], 2019. p. 6086–6095.

TIBSHIRANI, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, Wiley, v. 58, n. 1, p. 267–288, 1996.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, Ł.; POLOSUKHIN, I. Attention is all you need. *Advances in neural information processing systems*, v. 30, 2017.

ZHANG, D.; LU, G. Review of shape representation and description techniques. *Pattern Recognition*, v. 37, n. 1, p. 1 – 19, 2004. ISSN 0031-3203. Available at: <http://www.sciencedirect.com/science/article/pii/S0031320303002759>.

APÊNDICE A – APPENDIX

A.1 IMPLEMENTATION

This section presents the code implementation of the MASViT network and the Angle Dropout module to provide a clearer understanding of the architecture. The implementation was developed in Python 3 using the PyTorch framework. Alternatively, the full code can also be found in the following GitHub repository: github.com/Charamba/MASViT.

A.1.1 MASViT Implementation

The MASViT network was implemented as a Python class that inherits from `nn.Module`, PyTorch's base class for neural network modules. The class defines an initialization method (`__init__`) where several arguments are specified. These include the convolutional layer parameters: `input_channel`, `hidden_channel` (1, 2, and 3), and `output_channel`, which define the number of channels for the vertical and horizontal 1D convolutions. In addition, the Transformer Encoder settings are defined through arguments such as:

- `num_patches`: Number of input patches embeddings;
- `num_heads`: Number of attention heads in the multi-head attention mechanism;
- `mlp_size`: Dimension of the MLP hidden layer;
- `mlp_dropout`: Dropout probability applied within the MLP layers;
- `embedding_dropout`: Dropout probability applied to the input patches embeddings;
- `num_transformer_layers`: Number of encoder sub-layers (Transformer blocks).

The class also takes arguments for `angle_dropout` and `num_classes`. Still within the `__init__` method, the network weights are initialized using the Xavier algorithm (GLOROT; BENGIO, 2010) with uniform distribution, through the method `init_weights_xavier`. Finally, the `forward` method implements the entire data flow of the MASViT architecture as described in Section 4.1.2: it includes the creation of patch embeddings, application of the Angle Dropout module, concatenation of the class token, addition of positional embeddings, and forwarding the sequence to the TE, followed by classification using the

final MLP applied to the class token. The following presents the full code implementation of the MASViT architecture:

```
import torch
import torch.nn as nn
from torch import relu
from angle_dropout import AngleDropout


class MultiAngleScaleVisionTransformer(nn.Module):
  """
  A Multi-Angle-Scale Vision Transformer implementation for image
    classification tasks.

  Args:
      input_channel (int): Number of input channels in the images.
      hidden_channel_1 (int): Number of output channels for the first
    convolutional layer.
      hidden_channel_2 (int): Number of output channels for the second
    convolutional layer.
      hidden_channel_3 (int): Number of output channels for the third
    convolutional layer.
      output_channel (int): Number of output channels for the final
    convolutional layer.
      num_patches (int): Number of patches to divide the input image into
    .
      num_heads (int): Number of attention heads in the Transformer
    encoder.
      mlp_size (int): Size of the feedforward layer in the Transformer
    encoder.
      mlp_dropout (float): Dropout rate for the MLP layers in the
    Transformer encoder.
      embedding_dropout (float): Dropout rate for the embedding layer.
      angle_dropout (float): Dropout rate for the angle dropout layer.
      num_transformer_layers (int): Number of Transformer encoder layers.
      num_classes (int): Number of output classes for classification.

  Attributes:
      conv1, conv2, conv3, conv4 (nn.Conv2d): Convolutional layers for
    patch embedding.
```

```
    vert_conv1, vert_conv2, vert_conv3, vert_conv4 (nn.Conv2d):
  Vertical convolutional layers for feature extraction.
    flatten (nn.Flatten): Layer to flatten the feature maps.
    class_embedding (nn.Parameter): Learnable class token embedding.
    position_embedding (nn.Parameter): Learnable position embedding.
    transformer_encoder (nn.TransformerEncoder): Transformer encoder
  composed of multiple layers.
    classifier (nn.Sequential): Classification head consisting of a
  layer normalization and a linear layer.
    bn_conv1, bn_conv2, bn_conv3, bn_conv4 (nn.BatchNorm2d): Batch
  normalization layers for the convolutional layers.
    bn_vert_conv1, bn_vert_conv2, bn_vert_conv3, bn_vert_conv4 (nn.
  BatchNorm2d): Batch normalization layers for the vertical
  convolutional layers.
    embedding_dropout (nn.Dropout): Dropout layer for embeddings.
    angle_dropout (AngleDropout): Dropout layer that applies angle-
  based dropout.
  """

  def __init__(self,
         input_channel,
         hidden_channel_1,
         hidden_channel_2,
         hidden_channel_3,
         output_channel,
         num_patches,
         num_heads,
         mlp_size,
         mlp_dropout,
         embedding_dropout,
         angle_dropout,
         num_transformer_layers,
         num_classes):
    super().__init__()

    # Convolutional layers
    self.conv1 = nn.Conv2d(
      input_channel, hidden_channel_1, kernel_size=(1, 1))
    self.conv2 = nn.Conv2d(
```

```
      hidden_channel_1, hidden_channel_2, kernel_size=(1, 3))
61    self.conv3 = nn.Conv2d(
      hidden_channel_2, hidden_channel_3, kernel_size=(1, 5))
63    self.conv4 = nn.Conv2d(
      hidden_channel_3, output_channel, kernel_size=(1, 6), stride=(1, 6)
    )
65
      self.vert_conv1 = nn.Conv2d(
67      output_channel, output_channel, kernel_size=(3, 1))
      self.vert_conv2 = nn.Conv2d(
69      output_channel, output_channel, kernel_size=(3, 1))
      self.vert_conv3 = nn.Conv2d(
71      output_channel, output_channel, kernel_size=(3, 1))
      self.vert_conv4 = nn.Conv2d(
73      output_channel, output_channel, kernel_size=(3, 1))

75    # only flatten the feature map dimensions into a single vector
      self.flatten = nn.Flatten(start_dim=2,
77                  end_dim=3)

79    self.num_patches = num_patches
      self.embedding_dim = output_channel
81
      # Class token embedding
83    self.class_embedding = nn.Parameter(data=torch.randn(1, 1, self.
    embedding_dim),
                      requires_grad=True)
85
      # Position token embedding
87    self.position_embedding = nn.Parameter(data=torch.randn(1, self.
    num_patches+1, self.embedding_dim),
                          requires_grad=True)
89    # Transformer Encoder
      encoder_layer = nn.TransformerEncoderLayer(d_model=self.embedding_dim
    ,
91                            nhead=num_heads,
                              dim_feedforward=mlp_size,
93                            dropout=mlp_dropout,
                              activation="gelu",
```

```
                            batch_first=True,
                            norm_first=True)

    self.transformer_encoder = nn.TransformerEncoder(
      encoder_layer, num_layers=num_transformer_layers)

    # Classifier head
    self.classifier = nn.Sequential(
      nn.LayerNorm(normalized_shape=self.embedding_dim),
      nn.Linear(in_features=self.embedding_dim,
            out_features=num_classes))

    # Batch Normalization layers
    self.bn_conv1 = nn.BatchNorm2d(hidden_channel_1)
    self.bn_conv2 = nn.BatchNorm2d(hidden_channel_2)
    self.bn_conv3 = nn.BatchNorm2d(hidden_channel_3)
    self.bn_conv4 = nn.BatchNorm2d(output_channel)

    self.bn_vert_conv1 = nn.BatchNorm2d(output_channel)
    self.bn_vert_conv2 = nn.BatchNorm2d(output_channel)
    self.bn_vert_conv3 = nn.BatchNorm2d(output_channel)
    self.bn_vert_conv4 = nn.BatchNorm2d(output_channel)

    # Dropout
    self.embedding_dropout = nn.Dropout(embedding_dropout)

    self.angle_dropout = AngleDropout(num_patches, angle_dropout)

    # Initializing weights
    self.apply(self.init_weights_xavier)

def init_weights_xavier(self, module):
    """
    Initialize the weights of the module using Xavier uniform
    distribution.

    Args:
      module (nn.Module): The module to initialize.
    """
```

```
133     if isinstance(module, nn.Linear):
            nn.init.xavier_uniform_(module.weight)
135         module.bias.data.fill_(0.01)

137   def forward(self, x):
        """
139     Forward pass of the model.

141     Args:
          x (torch.Tensor): Input tensor of shape (batch_size, input_channel,
        height, width).
143
        Returns:
145       torch.Tensor: Output tensor of shape (batch_size, num_classes)
       containing the class scores.
        """
147     # Create class token embedding and expand it to match the batch size
        batch_size = x.shape[0]
149     class_token = self.class_embedding.expand(batch_size, -1, -1)

151     # Conv Layers (create patch embedding)
        x = relu(self.bn_conv1(self.conv1(x)))      # (48,48) --> (48,48)
153     x = relu(self.bn_conv2(self.conv2(x)))      # (48,48) --> (48,46)
        x = relu(self.bn_conv3(self.conv3(x)))      # (48,46) --> (48,42)
155     x = nn.MaxPool2d((1, 7), stride=(1, 7))(x)  # (48,42) --> (48,6)
        x = relu(self.bn_conv4(self.conv4(x)))      # (48,6)  --> (48,1)
157
        x = relu(self.bn_vert_conv1(self.vert_conv1(x)))  # (48,1) --> (46,1)
159     x = relu(self.bn_vert_conv2(self.vert_conv2(x)))  # (46,1) --> (44,1)
        x = relu(self.bn_vert_conv3(self.vert_conv3(x)))  # (44,1) --> (42,1)
161     x = relu(self.bn_vert_conv4(self.vert_conv4(x)))  # (42,1) --> (40,1)
        x = nn.MaxPool2d((5, 1), stride=(5, 1))(x)        # (40,1) --> (8,1)
163
        # Performs angle dropout
165     x = self.angle_dropout(x)

167     # Adjust so the embedding is on the final dimension:
        # [batch_size, row_patches*C, N] -> [batch_size, N, row_patches*C]
169     x = self.flatten(x)
```

```
    x = x.permute(0, 2, 1)

    # Concat class embedding and patch embedding
    x = torch.cat((class_token, x), dim=1)

    # Add position embedding to patch embedding
    x = self.position_embedding + x

    # Run embedding dropout (optional)
    x = self.embedding_dropout(x)

    # Pass patch, position and class embedding through transformer
    encoder layers
    x = self.transformer_encoder(x)

    # Put 0 index logit through classifier
    # run on each sample in a batch at 0 index
    x = self.classifier(x[:, 0])

    return x
```

A.1.2 Angle Dropout Implementation

The Angle Dropout module was implemented as a Python class that inherits from `nn.Module`, PyTorch's base class for neural network modules. The class defines an initialization method (`__init__`) where two arguments are specified: `num_angles` (number of angles) and `p` (dropout probability). Finally, the `forward` method implements the Angle Dropout algorithm described in Alg. 1. The following presents the full code implementation of the Angle Dropout:

```
import torch
import torch.nn.functional as F
from torch import nn, Tensor

class AngleDropout(nn.Module):
  """
  Applies a dropout that randomly selects certain rows of the
  input tensors (B,C,H,W) along the height dimension and sets
```

```
    them to zero based on a dropout probability p

    Args:
      num_angles (int): Number of angles (rows) of the input tensor
      p (float): Probability of dropout
    """

    def __init__(self, num_angles: int, p: float = 0.5):
      super().__init__()
      self.num_angles = num_angles
      self.p = p  # probability

    def forward(self, x: Tensor) -> Tensor:
      """
      Forward pass of the AngleDropout layer.

      Args:
        x (Tensor): Input tensor of shape (B, C, H, W).

      Returns:
        Tensor: Output tensor with dropout applied along the height
     dimension.
      """
      if self.training:
        b, c, h, w = x.shape
        # dropout angles randomly
        mask_angle = F.dropout(torch.ones(b, 1, self.num_angles, 1).cuda(),
       p=self.p, training=self.training)
        # repeat in c dimensions
        mask_angle = mask_angle.repeat(1, c, 1, 1)

        mask_data = mask_angle.expand(b, c, self.num_angles, w)

        mask_data = mask_data.repeat(1, 1, 1, int(
          h/self.num_angles)).reshape(b, c, h, w)
        x = mask_data * x

      return x
```

APÊNDICE B – APPENDIX

B.1 CLASSIFICATION REPORTS

We present all classification metrics for each test dataset, encompassing Precision, Recall, F1-score per class, and the Accuracy of all MASViT experiments utilizing two methodologies: fixed centroid and max score, spanning across several tables. There are a total of 16 tables: two for GTSRB Test, two for elevation angle 90° (Aff-GTSRB / Proj-GTSRB), six for Aff-GTSRB with elevation angles 30°, 45°, and 60°, and finally, six more for Proj-GTSRB with elevation angles 30°, 45°, and 60°.

B.1.1 GTSRB Test

The tables 5 and 6 show the report of tests in GTSRB Test dataset using fixed centroid and max score approaches, respectively.

Tabela 5 – GTSRB Test - MASViT (fixed centroid)

Class	Precision	Recall	F1-score	Support
	1.0000	1.0000	1.0000	60
	0.9874	0.9778	0.9826	720
	0.9720	0.9985	0.9851	660
	0.9858	0.9952	0.9905	420
	0.9970	0.9623	0.9794	690
	0.9863	0.9986	0.9924	720
	1.0000	1.0000	1.0000	270
	1.0000	0.9905	0.9952	210
	1.0000	0.9933	0.9967	150
	0.9972	1.0000	0.9986	360
	0.9846	0.9846	0.9846	390

Continued on next page

Tabela 5 – continuation

Class	Precision	Recall	F1-score	Support
	0.9677	1.0000	0.9836	60
	0.9842	0.9987	0.9914	750
	0.9375	1.0000	0.9677	90
	0.9750	0.8667	0.9176	90
	0.9916	0.9833	0.9874	120
	0.9867	0.9867	0.9867	150
	1.0000	0.9667	0.9831	90
	0.9979	0.9792	0.9884	480
	0.9871	0.8500	0.9134	180
	1.0000	0.9833	0.9916	60
	0.9932	0.9800	0.9866	150
	0.9677	1.0000	0.9836	90
	0.9932	0.9667	0.9797	450
	0.9730	0.9600	0.9664	150
	1.0000	0.9852	0.9925	270
	0.9836	1.0000	0.9917	60
	0.9857	0.9857	0.9857	210
	1.0000	0.9917	0.9958	120
	0.9923	0.9974	0.9949	390
	0.9737	0.9250	0.9487	120
	0.9032	0.9333	0.9180	60
	0.9477	0.9986	0.9725	690
	0.9855	0.7556	0.8553	90
	0.9984	0.9758	0.9870	660

Continued on next page

Tabela 5 – continuation

Class	Precision	Recall	F1-score	Support
	0.8713	0.9778	0.9215	90
	0.8824	1.0000	0.9375	60
	0.9468	0.9889	0.9674	90
	0.9617	0.9968	0.9790	630
	0.9797	0.9667	0.9732	150
	0.9933	0.9933	0.9933	450
	0.9955	0.9933	0.9944	450
	0.9856	1.0000	0.9928	480
Accuracy			**0.9831**	12630
Macro avg	0.9779	0.9741	0.9752	12630
Weighted avg	0.9835	0.9831	0.9829	12630

Tabela 6 – GTSRB Test - MASViT (max score)

Class	Precision	Recall	F1-score	Support
	1.0000	1.0000	1.0000	60
	0.9916	0.9819	0.9867	720
	0.9836	1.0000	0.9917	660
	0.9976	0.9952	0.9964	420
	0.9955	0.9638	0.9794	690
	0.9903	0.9958	0.9931	720
	0.9926	1.0000	0.9963	270
	1.0000	0.9905	0.9952	210
	0.9551	0.9933	0.9739	150

Continued on next page

Tabela 6 – continuation

Class	Precision	Recall	F1-score	Support
	0.9972	0.9972	0.9972	360
	0.9719	0.9744	0.9731	390
	0.8955	1.0000	0.9449	60
	0.9907	0.9907	0.9907	750
	0.9091	1.0000	0.9524	90
	1.0000	0.6667	0.8000	90
	0.9917	0.9917	0.9917	120
	0.9868	1.0000	0.9934	150
	0.9885	0.9556	0.9718	90
	0.9958	0.9896	0.9927	480
	0.9770	0.9444	0.9605	180
	0.9831	0.9667	0.9748	60
	1.0000	0.9800	0.9899	150
	0.9783	1.0000	0.9890	90
	0.9714	0.9800	0.9757	450
	0.9931	0.9533	0.9728	150
	1.0000	1.0000	1.0000	270
	0.9677	1.0000	0.9836	60
	0.9673	0.9857	0.9764	210
	0.7532	0.9917	0.8561	120
	0.9717	0.9692	0.9705	390
	1.0000	0.9417	0.9700	120
	1.0000	0.9833	0.9916	60
	0.9442	0.9565	0.9503	690

Continued on next page

Tabela 6 – continuation

Class	Precision	Recall	F1-score	Support
	0.9825	0.6222	0.7619	90
	0.9985	0.9848	0.9916	660
	0.8544	0.9778	0.9119	90
	0.7945	0.9667	0.8722	60
	0.8878	0.9667	0.9255	90
	0.9873	0.9905	0.9889	630
	1.0000	0.9267	0.9619	150
	0.9912	0.9978	0.9945	450
	0.9912	0.9978	0.9945	450
	0.9938	1.0000	0.9969	480
Accuracy			**0.9795**	12630
Macro avg	0.9679	0.9667	0.9647	12630
Weighted avg	0.9808	0.9795	0.9793	12630

B.1.2 Aff-GTSRB / Proj-GTSRB (el=90°)

In elevation angle equals to 90°, there aren't not difference between affinity and projectivity. So, there are here two classification report tables, using the two MASViT approaches: fixed centroid in Table 7 and max score in Table 8.

Tabela 7 – Aff-GTSRB / Proj-GTSRB (el=90°) - MASViT (fixed centroid)

Class	Precision	Recall	F1-score	Support
	0.9957	0.9542	0.9745	480

Continued on next page

Tabela 7 – continuation

Class	Precision	Recall	F1-score	Support
	0.9710	0.9821	0.9765	5760
	0.9794	0.9973	0.9883	5280
	0.9702	0.9872	0.9786	3360
	0.9940	0.9609	0.9772	5520
	0.9559	0.9938	0.9745	5760
	0.9727	0.9903	0.9814	2160
	1.0000	0.8780	0.9350	1680
	0.9992	0.9892	0.9941	1200
	0.9968	0.9691	0.9827	2880
	0.9865	0.9622	0.9742	3120
	0.9467	1.0000	0.9726	480
	0.9712	0.9952	0.9830	6000
	0.8997	0.9972	0.9460	720
	0.9547	0.7903	0.8647	720
	0.9924	0.9583	0.9751	960
	0.9386	0.9808	0.9593	1200
	0.9697	0.9792	0.9744	720
	0.9900	0.9552	0.9723	3840
	0.9741	0.8896	0.9299	1440
	0.9488	0.8875	0.9171	480
	0.9507	0.9800	0.9651	1200
	0.9676	0.9944	0.9808	720
	0.9878	0.9661	0.9768	3600
	0.9591	0.9575	0.9583	1200

Continued on next page

Tabela 7 – continuation

Class	Precision	Recall	F1-score	Support
	0.9920	0.9815	0.9867	2160
	0.9856	1.0000	0.9928	480
	0.9923	0.9179	0.9536	1680
	0.9695	0.9917	0.9804	960
	0.9709	0.9946	0.9826	3120
	0.9605	0.9365	0.9483	960
	0.9134	0.9667	0.9393	480
	0.9376	0.9746	0.9558	5520
	0.9683	0.7639	0.8540	720
	0.9973	0.9616	0.9791	5280
	0.8014	0.9861	0.8842	720
	0.8762	0.9729	0.9220	480
	0.9413	0.9583	0.9498	720
	0.9668	0.9891	0.9778	5040
	0.9905	0.9592	0.9746	1200
	0.9846	0.9919	0.9882	3600
	0.9925	0.9861	0.9893	3600
	0.9748	0.9992	0.9869	3840
Accuracy			**0.9727**	101040
Macro avg	0.9648	0.9611	0.9618	101040
Weighted avg	0.9735	0.9727	0.9726	101040

Tabela 8 – Aff-GTSRB / Proj-GTSRB (el=90°) - MASViT (max score)

Class	Precision	Recall	F1-score	Support
	0.9958	0.9812	0.9885	480
	0.9805	0.9861	0.9833	5760
	0.9899	1.0000	0.9949	5280
	0.9939	0.9771	0.9854	3360
	0.9948	0.9641	0.9792	5520
	0.9686	0.9965	0.9824	5760
	0.9724	0.9963	0.9842	2160
	1.0000	0.9054	0.9503	1680
	0.9835	0.9958	0.9896	1200
	0.9965	0.9785	0.9874	2880
	0.9617	0.9571	0.9594	3120
	0.9619	1.0000	0.9806	480
	0.9833	0.9825	0.9829	6000
	0.8955	1.0000	0.9449	720
	0.9468	0.7917	0.8623	720
	0.9979	0.9823	0.9900	960
	0.9615	1.0000	0.9804	1200
	0.9804	0.9736	0.9770	720
	0.9966	0.9810	0.9887	3840
	0.9744	0.9771	0.9757	1440
	0.9799	0.8146	0.8896	480
	0.9949	0.9733	0.9840	1200
	0.9559	0.9931	0.9741	720

Continued on next page

Tabela 8 – continuation

Class	Precision	Recall	F1-score	Support
	0.9248	0.9869	0.9549	3600
	0.9597	0.9525	0.9561	1200
	0.9995	0.9954	0.9974	2160
	0.9836	1.0000	0.9917	480
	0.8787	0.9833	0.9281	1680
	0.6990	0.9990	0.8225	960
	0.9747	0.9510	0.9627	3120
	0.9955	0.9156	0.9539	960
	0.9875	0.9875	0.9875	480
	0.9451	0.9098	0.9271	5520
	0.9346	0.6750	0.7839	720
	0.9969	0.9752	0.9859	5280
	0.8410	0.9847	0.9072	720
	0.7689	0.9979	0.8685	480
	0.9249	0.9583	0.9413	720
	0.9913	0.9454	0.9678	5040
	0.9957	0.9617	0.9784	1200
	0.9863	0.9972	0.9917	3600
	0.9903	0.9953	0.9928	3600
	0.9927	0.9979	0.9953	3840
Accuracy			**0.9711**	101040
Macro avg	0.9590	0.9623	0.9584	101040
Weighted avg	0.9731	0.9711	0.9712	101040

B.1.3 Aff-GTSRB el={60°, 45°, 30°}

The tables 9, 10, 11, 12, 13 and 14 show the report of tests in aff-GTSRB datasets for elevation degrees 60°, 45° and 30°, alternating between fixed centroid and max score approaches.

Tabela 9 – Aff-GTSRB (el=60°) - MASViT (fixed centroid)

Class	Precision	Recall	F1-score	Support
	0.9655	0.7583	0.8495	480
	0.9438	0.9599	0.9518	5760
	0.9643	0.9968	0.9803	5280
	0.9793	0.9708	0.9750	3360
	0.9979	0.9612	0.9792	5520
	0.9485	0.9969	0.9721	5760
	0.9894	0.9917	0.9905	2160
	1.0000	0.8750	0.9333	1680
	0.9958	0.9808	0.9882	1200
	0.9972	0.9889	0.9930	2880
	0.9868	0.9321	0.9586	3120
	0.9462	0.9896	0.9674	480
	0.9483	0.9790	0.9634	6000
	0.8043	0.9931	0.8888	720
	0.9370	0.6819	0.7894	720
	0.9903	0.9531	0.9713	960
	0.9667	0.9692	0.9680	1200
	0.9670	0.9361	0.9513	720
	0.9729	0.9732	0.9731	3840

Continued on next page

Tabela 9 – continuation

Class	Precision	Recall	F1-score	Support
	0.9522	0.8708	0.9097	1440
	0.8873	0.7542	0.8153	480
	0.9410	0.9708	0.9557	1200
	0.9505	0.9611	0.9558	720
	0.9844	0.9306	0.9567	3600
	0.9224	0.9117	0.9170	1200
	0.9716	0.9662	0.9689	2160
	0.9231	1.0000	0.9600	480
	0.9633	0.9518	0.9575	1680
	0.8989	0.9542	0.9257	960
	0.9855	0.9612	0.9732	3120
	0.8830	0.8885	0.8858	960
	0.8781	0.8104	0.8429	480
	0.9132	0.9817	0.9462	5520
	0.9284	0.6125	0.7381	720
	0.9912	0.9350	0.9623	5280
	0.8690	0.9861	0.9239	720
	0.7780	0.9417	0.8520	480
	0.8537	0.9889	0.9163	720
	0.9142	0.9833	0.9475	5040
	0.9871	0.8942	0.9383	1200
	0.9756	0.9872	0.9814	3600
	0.9763	0.9444	0.9601	2400

Continued on next page

Tabela 9 – continuation

Class	Precision	Recall	F1-score	Support
	0.9325	0.9458	0.9391	720
Accuracy			**0.9581**	101040
Macro avg	0.9445	0.9321	0.9356	101040
Weighted avg	0.9594	0.9581	0.9576	101040

Tabela 10 – Aff-GTSRB (el=60°) - MASViT (max score)

Class	Precision	Recall	F1-score	Support
	0.9855	0.9896	0.9875	480
	0.9838	0.9814	0.9826	5760
	0.9780	1.0000	0.9889	5280
	0.9974	0.9095	0.9514	3360
	0.9985	0.9563	0.9770	5520
	0.9581	0.9972	0.9773	5760
	0.9953	0.9907	0.9930	2160
	1.0000	0.8780	0.9350	1680
	0.9975	0.9883	0.9929	1200
	0.9914	0.9962	0.9938	2880
	0.9895	0.9397	0.9640	3120
	0.9776	1.0000	0.9887	480
	0.9866	0.9810	0.9838	6000
	0.7059	1.0000	0.8276	720
	0.9499	0.7903	0.8628	720
	0.9947	0.9750	0.9847	960

Continued on next page

Tabela 10 – continuation

Class	Precision	Recall	F1-score	Support
	0.9463	0.9992	0.9720	1200
	0.9659	0.9847	0.9752	720
	0.9918	0.9797	0.9857	3840
	0.9843	0.9590	0.9715	1440
	0.9870	0.7937	0.8799	480
	0.9816	0.9800	0.9808	1200
	0.9571	0.9917	0.9741	720
	0.9615	0.9844	0.9728	3600
	0.9128	0.9425	0.9274	1200
	0.9977	0.9921	0.9949	2160
	0.9658	1.0000	0.9826	480
	0.6852	0.9887	0.8095	1680
	0.5596	0.9823	0.7130	960
	0.9774	0.9426	0.9597	3120
	0.9891	0.8500	0.9143	960
	0.9936	0.9771	0.9853	480
	0.9492	0.7755	0.8536	5520
	0.9545	0.7861	0.8621	720
	0.9988	0.9797	0.9892	5280
	0.8375	0.9736	0.9004	720
	0.6907	0.9771	0.8093	480
	0.8285	0.9931	0.9033	720
	0.9841	0.9720	0.9780	5040
	0.9937	0.9150	0.9527	1200

Continued on next page

Tabela 10 – continuation

Class	Precision	Recall	F1-score	Support
	0.9768	0.9958	0.9862	3600
	0.9895	0.9958	0.9927	3600
	0.9922	0.9896	0.9909	3840
Accuracy			**0.9600**	101040
Macro avg	0.9428	0.9552	0.9444	101040
Weighted avg	0.9670	0.9600	0.9611	101040

Tabela 11 – Aff-GTSRB (el=45°) - MASViT (fixed centroid)

Class	Precision	Recall	F1-score	Support
	0.9746	0.5604	0.7116	480
	0.9059	0.9243	0.9150	5760
	0.9742	0.9860	0.9800	5280
	0.9442	0.9518	0.9480	3360
	0.9948	0.9625	0.9784	5520
	0.9491	0.9936	0.9708	5760
	0.9325	0.9907	0.9607	2160
	0.9987	0.8815	0.9365	1680
	0.9732	0.9700	0.9716	1200
	0.9978	0.9455	0.9709	2880
	0.9695	0.8971	0.9319	3120
	0.9252	0.9792	0.9514	480
	0.8777	0.9565	0.9154	6000
	0.7654	0.9833	0.8608	720

Continued on next page

Tabela 11 – continuation

Class	Precision	Recall	F1-score	Support
	0.8759	0.6861	0.7695	720
	0.9931	0.8979	0.9431	960
	0.8127	0.9617	0.8809	1200
	0.9456	0.7972	0.8651	720
	0.9507	0.9591	0.9549	3840
	0.9490	0.8660	0.9056	1440
	0.8242	0.5958	0.6917	480
	0.8939	0.9333	0.9132	1200
	0.8870	0.9375	0.9115	720
	0.9360	0.8697	0.9017	3600
	0.8577	0.8892	0.8732	1200
	0.9815	0.9356	0.9580	2160
	0.9693	0.9854	0.9773	480
	0.9404	0.9107	0.9253	1680
	0.8730	0.9094	0.8908	960
	0.9816	0.9208	0.9502	3120
	0.8824	0.7812	0.8287	960
	0.7930	0.7583	0.7753	480
	0.8676	0.9808	0.9207	5520
	0.9548	0.5569	0.7035	720
	0.9901	0.8727	0.9277	5280
	0.8275	0.9861	0.8999	720
	0.8410	0.8812	0.8606	480
	0.9281	0.8958	0.9117	720

Continued on next page

Tabela 11 – continuation

Class	Precision	Recall	F1-score	Support
	0.8658	0.9435	0.9030	5040
	0.9731	0.9342	0.9532	1200
	0.9388	0.9678	0.9531	3600
	0.9543	0.9275	0.9407	3600
	0.9512	0.9888	0.9696	3840
Accuracy			**0.9316**	101040
Macro avg	0.9214	0.8957	0.9038	101040
Weighted avg	0.9344	0.9316	0.9309	101040

Tabela 12 – Aff-GTSRB (el=45°) - MASViT (max score)

Class	Precision	Recall	F1-score	Support
	0.9898	0.8104	0.8912	480
	0.9607	0.9557	0.9582	5760
	0.9878	0.9987	0.9932	5280
	0.9804	0.9402	0.9599	3360
	0.9925	0.9592	0.9756	5520
	0.9610	0.9964	0.9783	5760
	0.9361	0.9903	0.9624	2160
	0.9987	0.8863	0.9391	1680
	0.9734	0.9775	0.9755	1200
	0.9938	0.9535	0.9732	2880
	0.9594	0.9327	0.9459	3120
	0.9756	1.0000	0.9877	480

Continued on next page

Tabela 12 – continuation

Class	Precision	Recall	F1-score	Support
	0.9485	0.9733	0.9608	6000
	0.7756	0.9986	0.8731	720
	0.9101	0.8292	0.8677	720
	0.9978	0.9302	0.9628	960
	0.8721	1.0000	0.9317	1200
	0.9517	0.9583	0.9550	720
	0.9850	0.9734	0.9792	3840
	0.9736	0.9715	0.9725	1440
	0.9599	0.5979	0.7368	480
	0.9676	0.9700	0.9688	1200
	0.9288	0.9972	0.9618	720
	0.8809	0.9653	0.9211	3600
	0.9374	0.9367	0.9371	1200
	0.9972	0.9815	0.9893	2160
	0.9856	1.0000	0.9928	480
	0.6937	0.9720	0.8096	1680
	0.5421	0.9583	0.6925	960
	0.9698	0.8865	0.9263	3120
	0.9962	0.8198	0.8994	960
	0.9832	0.9729	0.9780	480
	0.9204	0.7837	0.8466	5520
	0.9916	0.6542	0.7883	720
	0.9930	0.9430	0.9674	5280
	0.6822	0.9778	0.8037	720

Continued on next page

Tabela 12 – continuation

Class	Precision	Recall	F1-score	Support
	0.8114	0.9771	0.8866	480
	0.9306	0.9306	0.9306	720
	0.9823	0.8810	0.9289	5040
	0.9914	0.9583	0.9746	1200
	0.9350	0.9908	0.9621	3600
	0.9648	0.9897	0.9771	3600
	0.9782	0.9951	0.9866	3840
Accuracy			**0.9449**	101040
Macro avg	0.9337	0.9343	0.9281	101040
Weighted avg	0.9519	0.9449	0.9457	101040

Tabela 13 – Aff-GTSRB (el=30°) - MASViT (fixed centroid)

Class	Precision	Recall	F1-score	Support
	0.4344	0.2208	0.2928	480
	0.7211	0.5422	0.6190	5760
	0.9123	0.8172	0.8621	5280
	0.7833	0.6714	0.7231	3360
	0.9974	0.9121	0.9529	5520
	0.7231	0.9726	0.8295	5760
	0.9176	0.9537	0.9353	2160
	0.9816	0.1268	0.2246	1680
	0.9807	0.7217	0.8315	1200
	0.9771	0.9174	0.9463	2880

Continued on next page

Tabela 13 – continuation

Class	Precision	Recall	F1-score	Support
	0.7625	0.4641	0.5770	3120
	0.7517	0.4729	0.5806	480
	0.5726	0.6198	0.5953	6000
	0.3307	0.5778	0.4206	720
	0.3853	0.2333	0.2907	720
	0.8414	0.7073	0.7685	960
	0.4800	0.7292	0.5789	1200
	0.4378	0.4056	0.4211	720
	0.5764	0.8414	0.6842	3840
	0.7643	0.7139	0.7382	1440
	0.2749	0.2417	0.2572	480
	0.3543	0.6517	0.4591	1200
	0.3906	0.7583	0.5156	720
	0.5273	0.5039	0.5153	3600
	0.3704	0.7108	0.4870	1200
	0.6978	0.7472	0.7217	2160
	0.7837	0.9812	0.8714	480
	0.7132	0.6750	0.6936	1680
	0.5459	0.3594	0.4334	960
	0.8669	0.2442	0.3811	3120
	0.6047	0.3187	0.4175	960
	0.4549	0.2208	0.2973	480
	0.5276	0.9045	0.6664	5520
	0.3985	0.2917	0.3368	720

Continued on next page

Tabela 13 – continuation

Class	Precision	Recall	F1-score	Support
	0.9297	0.3805	0.5400	5280
	0.5829	0.8597	0.6947	720
	0.4641	0.2021	0.2816	480
	0.4769	0.7444	0.5813	720
	0.4992	0.7631	0.6036	5040
	0.8751	0.6658	0.7563	1200
	0.7634	0.5969	0.6700	3600
	0.5811	0.6547	0.6157	3600
	0.9368	0.6867	0.7925	3840
Accuracy			**0.6671**	101040
Macro avg	0.6500	0.5996	0.5921	101040
Weighted avg	0.7145	0.6671	0.6597	101040

Tabela 14 – Aff-GTSRB (el=30°) - MASViT (max score)

Class	Precision	Recall	F1-score	Support
	0.5866	0.4938	0.5362	480
	0.8988	0.6045	0.7229	5760
	0.8972	0.9652	0.9299	5280
	0.9106	0.7336	0.8126	3360
	0.9963	0.8710	0.9294	5520
	0.7735	0.9778	0.8637	5760
	0.8597	0.9560	0.9053	2160
	0.9962	0.1571	0.2715	1680

Continued on next page

Tabela 14 – continuation

Class	Precision	Recall	F1-score	Support
	0.9951	0.8500	0.9169	1200
	0.8781	0.9479	0.9117	2880
	0.9301	0.5885	0.7208	3120
	0.9331	0.6104	0.7380	480
	0.7783	0.7810	0.7796	6000
	0.4387	0.7958	0.5656	720
	0.6863	0.3889	0.4965	720
	0.9048	0.6833	0.7786	960
	0.6454	0.7433	0.6909	1200
	0.5860	0.6958	0.6362	720
	0.6444	0.9039	0.7524	3840
	0.9096	0.9437	0.9264	1440
	0.4218	0.2979	0.3492	480
	0.4469	0.8025	0.5741	1200
	0.3749	0.9597	0.5392	720
	0.8069	0.6711	0.7328	3600
	0.4235	0.7775	0.5483	1200
	0.7521	0.8162	0.7829	2160
	0.6316	1.0000	0.7742	480
	0.5608	0.7988	0.6590	1680
	0.4287	0.6729	0.5237	960
	0.9347	0.3397	0.4984	3120
	0.9686	0.5781	0.7241	960
	0.8199	0.6354	0.7160	480

Continued on next page

Tabela 14 – continuation

Class	Precision	Recall	F1-score	Support
	0.6718	0.8420	0.7473	5520
	0.7644	0.4778	0.5880	720
	0.9831	0.6500	0.7826	5280
	0.4761	0.7889	0.5938	720
	0.4307	0.3042	0.3565	480
	0.4482	0.8583	0.5889	720
	0.7771	0.8224	0.7991	5040
	0.8559	0.7625	0.8065	1200
	0.7813	0.7622	0.7717	3600
	0.7678	0.8944	0.8263	3600
	0.9849	0.7964	0.8806	3840
Accuracy			**0.7668**	101040
Macro avg	0.7386	0.7209	0.6988	101040
Weighted avg	0.8087	0.7668	0.7645	101040

B.1.4 Proj-GTSRB el={60°, 45°, 30°}

The tables 15, 16, 17, 18, 19 and 20 show the report of tests in aff-GTSRB datasets for elevation degrees 60°, 45° and 30°, alternating between fixed centroid and max score approaches.

Tabela 15 – Proj-GTSRB (el=60°) - MASViT (fixed centroid)

Class	Precision	Recall	F1-score	Support
	0.9658	0.7646	0.8535	480
	0.9401	0.9592	0.9496	5760
	0.9634	0.9962	0.9795	5280
	0.9795	0.9688	0.9741	3360
	0.9976	0.9627	0.9798	5520
	0.9452	0.9964	0.9701	5760
	0.9875	0.9912	0.9894	2160
	0.9993	0.8726	0.9317	1680
	0.9949	0.9817	0.9883	1200
	0.9965	0.9889	0.9927	2880
	0.9877	0.9256	0.9557	3120
	0.9481	0.9896	0.9684	480
	0.9467	0.9778	0.9620	6000
	0.8016	0.9931	0.8871	720
	0.9312	0.6764	0.7836	720
	0.9902	0.9437	0.9664	960
	0.9580	0.9683	0.9631	1200
	0.9654	0.9292	0.9469	720
	0.9691	0.9729	0.9710	3840

Continued on next page

Tabela 15 – continuation

Class	Precision	Recall	F1-score	Support
	0.9529	0.8701	0.9096	1440
	0.8848	0.7521	0.8131	480
	0.9356	0.9692	0.9521	1200
	0.9502	0.9542	0.9522	720
	0.9833	0.9325	0.9572	3600
	0.9198	0.9075	0.9136	1200
	0.9684	0.9639	0.9661	2160
	0.9125	1.0000	0.9543	480
	0.9597	0.9494	0.9545	1680
	0.8861	0.9479	0.9160	960
	0.9855	0.9571	0.9711	3120
	0.8649	0.8740	0.8694	960
	0.8727	0.7854	0.8268	480
	0.9085	0.9799	0.9428	5520
	0.9163	0.5931	0.7201	720
	0.9909	0.9316	0.9604	5280
	0.8676	0.9833	0.9219	720
	0.7809	0.9208	0.8451	480
	0.8426	0.9889	0.9099	720
	0.9114	0.9819	0.9454	5040
	0.9843	0.8875	0.9334	1200
	0.9744	0.9850	0.9797	3600
	0.9809	0.9569	0.9688	3600

Continued on next page

Tabela 15 – continuation

Class	Precision	Recall	F1-score	Support
	0.9798	0.9831	0.9814	3840
Accuracy			**0.9558**	101040
Macro avg	0.9414	0.9282	0.9320	101040
Weighted avg	0.9572	0.9558	0.9553	101040

Tabela 16 – Proj-GTSRB (el=60°) - MASViT (max score)

Class	Precision	Recall	F1-score	Support
	0.9814	0.9917	0.9865	480
	0.9816	0.9795	0.9805	5760
	0.9751	1.0000	0.9874	5280
	0.9984	0.9054	0.9496	3360
	0.9985	0.9558	0.9767	5520
	0.9559	0.9970	0.9760	5760
	0.9944	0.9903	0.9923	2160
	1.0000	0.8762	0.9340	1680
	0.9975	0.9850	0.9912	1200
	0.9907	0.9958	0.9932	2880
	0.9925	0.9301	0.9603	3120
	0.9816	1.0000	0.9907	480
	0.9862	0.9792	0.9827	6000
	0.6936	1.0000	0.8191	720
	0.9397	0.8014	0.8651	720
	0.9979	0.9677	0.9825	960

Continued on next page

Tabela 16 – continuation

Class	Precision	Recall	F1-score	Support
	0.9456	0.9992	0.9716	1200
	0.9633	0.9847	0.9739	720
	0.9892	0.9797	0.9844	3840
	0.9857	0.9604	0.9729	1440
	0.9894	0.7812	0.8731	480
	0.9800	0.9825	0.9813	1200
	0.9649	0.9931	0.9788	720
	0.9505	0.9858	0.9678	3600
	0.9095	0.9383	0.9237	1200
	0.9977	0.9912	0.9944	2160
	0.9562	1.0000	0.9776	480
	0.6569	0.9881	0.7892	1680
	0.5382	0.9823	0.6954	960
	0.9749	0.9337	0.9538	3120
	0.9887	0.8219	0.8976	960
	0.9894	0.9750	0.9822	480
	0.9451	0.7542	0.8389	5520
	0.9449	0.7625	0.8440	720
	0.9988	0.9780	0.9883	5280
	0.8312	0.9708	0.8956	720
	0.6784	0.9667	0.7973	480
	0.8197	0.9917	0.8975	720
	0.9856	0.9655	0.9754	5040
	0.9936	0.9083	0.9491	1200

Continued on next page

Tabela 16 – continuation

Class	Precision	Recall	F1-score	Support
	0.9755	0.9956	0.9854	3600
	0.9898	0.9958	0.9928	3600
	0.9908	0.9867	0.9888	3840
Accuracy			**0.9567**	101040
Macro avg	0.9395	0.9518	0.9404	101040
Weighted avg	0.9647	0.9567	0.9579	101040

Tabela 17 – Proj-GTSRB (el=45°) - MASViT (fixed centroid)

Class	Precision	Recall	F1-score	Support
	0.9126	0.5437	0.6815	480
	0.8802	0.9094	0.8945	5760
	0.9543	0.9883	0.9710	5280
	0.9554	0.8985	0.9261	3360
	0.9977	0.9513	0.9739	5520
	0.9398	0.9943	0.9663	5760
	0.9878	0.9782	0.9830	2160
	0.9980	0.8887	0.9402	1680
	0.9597	0.9717	0.9656	1200
	0.9915	0.9764	0.9839	2880
	0.9728	0.8481	0.9062	3120
	0.9613	0.9313	0.9460	480
	0.8827	0.9283	0.9050	6000
	0.6861	0.9806	0.8073	720

Continued on next page

Tabela 17 – continuation

Class	Precision	Recall	F1-score	Support
	0.8373	0.6292	0.7185	720
	0.9851	0.8979	0.9395	960
	0.8999	0.9592	0.9286	1200
	0.8973	0.7889	0.8396	720
	0.9204	0.9604	0.9400	3840
	0.9344	0.8708	0.9015	1440
	0.7500	0.6125	0.6743	480
	0.8742	0.9383	0.9051	1200
	0.8978	0.9153	0.9065	720
	0.9530	0.8383	0.8920	3600
	0.8091	0.8583	0.8330	1200
	0.9488	0.9444	0.9466	2160
	0.8587	1.0000	0.9240	480
	0.9059	0.9167	0.9112	1680
	0.7998	0.8781	0.8371	960
	0.9802	0.8574	0.9147	3120
	0.7989	0.7448	0.7709	960
	0.7740	0.6208	0.6890	480
	0.8384	0.9679	0.8985	5520
	0.8613	0.5347	0.6598	720
	0.9849	0.8629	0.9198	5280
	0.8124	0.9681	0.8834	720
	0.7920	0.7854	0.7887	480
	0.7437	0.9792	0.8453	720

Continued on next page

Tabela 17 – continuation

Class	Precision	Recall	F1-score	Support
	0.8233	0.9633	0.8878	5040
	0.9796	0.8392	0.9039	1200
	0.9499	0.9542	0.9521	3600
	0.9356	0.9239	0.9297	3600
	0.9783	0.9633	0.9707	3840
Accuracy			**0.9179**	101040
Macro avg	0.8978	0.8782	0.8828	101040
Weighted avg	0.9220	0.9179	0.9174	101040

Tabela 18 – Proj-GTSRB (el=45°) - MASViT (max score)

Class	Precision	Recall	F1-score	Support
	0.9604	0.9104	0.9348	480
	0.9535	0.9535	0.9535	5760
	0.9593	1.0000	0.9792	5280
	0.9884	0.8851	0.9339	3360
	0.9981	0.9350	0.9655	5520
	0.9594	0.9962	0.9774	5760
	0.9842	0.9810	0.9826	2160
	1.0000	0.8940	0.9441	1680
	0.9991	0.9733	0.9861	1200
	0.9715	0.9927	0.9820	2880
	0.9910	0.8776	0.9308	3120
	0.9938	0.9979	0.9958	480

Continued on next page

Tabela 18 – continuation

Class	Precision	Recall	F1-score	Support
	0.9693	0.9630	0.9661	6000
	0.6492	1.0000	0.7873	720
	0.9151	0.7931	0.8497	720
	0.9920	0.9083	0.9483	960
	0.9011	0.9867	0.9419	1200
	0.9670	0.9764	0.9717	720
	0.9684	0.9724	0.9704	3840
	0.9776	0.9715	0.9746	1440
	0.9529	0.5896	0.7284	480
	0.9421	0.9758	0.9587	1200
	0.9545	0.9903	0.9721	720
	0.9309	0.9614	0.9459	3600
	0.8603	0.9183	0.8884	1200
	0.9836	0.9736	0.9786	2160
	0.8556	1.0000	0.9222	480
	0.6267	0.9762	0.7633	1680
	0.4997	0.9771	0.6613	960
	0.9806	0.8423	0.9062	3120
	0.9779	0.8313	0.8986	960
	0.9784	0.9417	0.9597	480
	0.9036	0.7467	0.8177	5520
	0.9479	0.7069	0.8099	720
	0.9972	0.9491	0.9725	5280
	0.7593	0.9333	0.8374	720

Continued on next page

Tabela 18 – continuation

Class	Precision	Recall	F1-score	Support
	0.6340	0.8625	0.7308	480
	0.7593	0.9903	0.8596	720
	0.9647	0.9381	0.9512	5040
	0.9813	0.8767	0.9261	1200
	0.9399	0.9867	0.9627	3600
	0.9681	0.9933	0.9805	3600
	0.9854	0.9656	0.9754	3840
Accuracy			**0.9376**	101040
Macro avg	0.9182	0.9278	0.9159	101040
Weighted avg	0.9478	0.9376	0.9393	101040

Tabela 19 – Proj-GTSRB (el=30°) - MASViT (fixed centroid)

Class	Precision	Recall	F1-score	Support
	0.3620	0.1667	0.2282	480
	0.6704	0.4844	0.5624	5760
	0.9051	0.8093	0.8545	5280
	0.7710	0.6324	0.6949	3360
	0.9972	0.9071	0.9500	5520
	0.7124	0.9698	0.8214	5760
	0.8937	0.9537	0.9227	2160
	0.9850	0.1173	0.2096	1680
	0.9702	0.7050	0.8166	1200
	0.9732	0.9090	0.9400	2880

Continued on next page

Tabela 19 – continuation

Class	Precision	Recall	F1-score	Support
	0.7471	0.4385	0.5526	3120
	0.8074	0.4104	0.5442	480
	0.5307	0.5832	0.5557	6000
	0.3062	0.5597	0.3959	720
	0.3753	0.2111	0.2702	720
	0.7822	0.6958	0.7365	960
	0.4761	0.6817	0.5607	1200
	0.4276	0.3611	0.3916	720
	0.5576	0.8344	0.6685	3840
	0.7249	0.6972	0.7108	1440
	0.2246	0.1979	0.2104	480
	0.3525	0.6275	0.4514	1200
	0.4009	0.7528	0.5232	720
	0.4871	0.4367	0.4605	3600
	0.3402	0.6750	0.4524	1200
	0.7148	0.6963	0.7054	2160
	0.7564	0.9833	0.8551	480
	0.7050	0.6315	0.6662	1680
	0.5000	0.3250	0.3939	960
	0.8233	0.1881	0.3063	3120
	0.5763	0.2990	0.3937	960
	0.4388	0.1792	0.2544	480
	0.4973	0.8833	0.6364	5520
	0.3298	0.2625	0.2923	720

Continued on next page

Tabela 19 – continuation

Class	Precision	Recall	F1-score	Support
	0.8934	0.3366	0.4889	5280
	0.5352	0.8236	0.6488	720
	0.4175	0.1792	0.2507	480
	0.4554	0.7083	0.5543	720
	0.4567	0.7554	0.5693	5040
	0.8753	0.6200	0.7259	1200
	0.7418	0.5683	0.6436	3600
	0.5275	0.6206	0.5703	3600
	0.9366	0.6805	0.7882	3840
Accuracy			**0.6404**	101040
Macro avg	0.6270	0.5711	0.5635	101040
Weighted avg	0.6907	0.6404	0.6313	101040

Tabela 20 – Proj-GTSRB (el=30°) - MASViT (max score)

Class	Precision	Recall	F1-score	Support
	0.4680	0.4271	0.4466	480
	0.8780	0.5286	0.6599	5760
	0.8816	0.9559	0.9172	5280
	0.9032	0.7051	0.7919	3360
	0.9954	0.8611	0.9234	5520
	0.7613	0.9719	0.8538	5760
	0.8352	0.9569	0.8919	2160
	0.9950	0.1179	0.2108	1680

Continued on next page

Tabela 20 – continuation

Class	Precision	Recall	F1-score	Support
	0.9980	0.8375	0.9107	1200
	0.8555	0.9417	0.8965	2880
	0.9206	0.5423	0.6825	3120
	0.9223	0.5437	0.6841	480
	0.7365	0.7592	0.7476	6000
	0.3952	0.7597	0.5200	720
	0.6651	0.3972	0.4974	720
	0.8859	0.6875	0.7742	960
	0.6215	0.7333	0.6728	1200
	0.5460	0.6181	0.5798	720
	0.6337	0.8812	0.7373	3840
	0.8978	0.9215	0.9095	1440
	0.2971	0.2562	0.2752	480
	0.4172	0.7833	0.5445	1200
	0.3574	0.9361	0.5173	720
	0.7774	0.5994	0.6769	3600
	0.4234	0.7650	0.5451	1200
	0.7312	0.7694	0.7498	2160
	0.6218	1.0000	0.7668	480
	0.5596	0.7577	0.6437	1680
	0.3991	0.6323	0.4893	960
	0.9211	0.2881	0.4390	3120
	0.9625	0.5344	0.6872	960
	0.8239	0.6042	0.6971	480

Continued on next page

Tabela 20 – continuation

Class	Precision	Recall	F1-score	Support
	0.6382	0.8205	0.7179	5520
	0.7293	0.4639	0.5671	720
	0.9749	0.5960	0.7398	5280
	0.4089	0.7792	0.5363	720
	0.3971	0.2896	0.3349	480
	0.3991	0.8181	0.5364	720
	0.7132	0.8171	0.7616	5040
	0.8261	0.7442	0.7830	1200
	0.7755	0.7506	0.7628	3600
	0.7235	0.8772	0.7930	3600
	0.9827	0.7701	0.8635	3840
Accuracy			**0.7393**	101040
Macro avg	0.7129	0.6930	0.6683	101040
Weighted avg	0.7871	0.7393	0.7357	101040

APÊNDICE C – APPENDIX

C.1 CONFUSION MATRICES

We present the confusion matrices for each test dataset for MASViT experiments utilizing the fixed centroid approach, spanning across several tables. There are a total of eight tables: one for GTSRB Test, one for elevation angle 90° (Aff-GTSRB / Proj-GTSRB), three for Aff-GTSRB with elevation angles 30°, 45°, and 60°, and finally, three more for Proj-GTSRB with elevation angles 30°, 45°, and 60°. The confusion matrices were rotated and resized to accommodate their large dimensions (48×48) and fit properly on the pages and are presented in the (True $\times$ Predicted) format.

C.1.1 GTSRB Test

The table 21 describes the confusion matrix for GTSRB Test dataset using fixed centroid approach.

C.1.2 Aff-GTSRB / Proj-GTSRB (el=90°)

In elevation angle equals to 90°, there aren't not difference between affinity and projectivity. So, there are here one confusion matrix, using the fixed centroid approach described in Table 22.

C.1.3 Aff-GTSRB el={60°, 45°, 30°}

The tables 23, 24 and 25 show the confusion matrices of tests in aff-GTSRB datasets for elevation degrees 60°, 45° and 30° using the fixed centroid approach.

C.1.4 Proj-GTSRB el={60°, 45°, 30°}

The tables 26, 27 and 28 show the confusion matrices of tests in proj-GTSRB datasets for elevation degrees 60°, 45° and 30° using the fixed centroid approach.

Tabela 21 – GTSRB Test - MASViT (fixed centroid) - Confusion Matrix (True × Predicted)

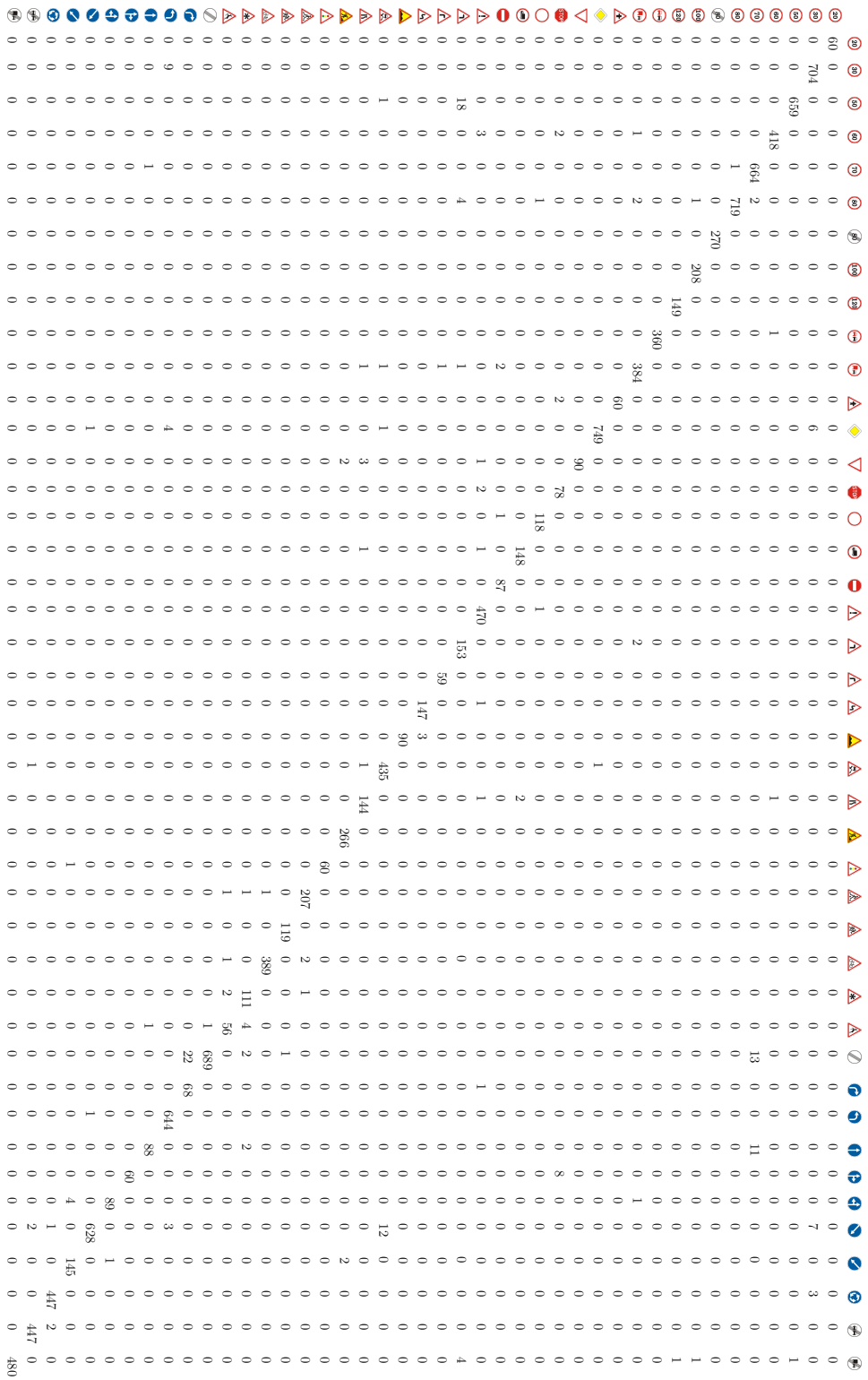


Tabela 22 – Aff-GTSRB / Proj-GTSRB (el=90°) - MASViT (fixed centroid) - Confusion Matrix (True × Predicted)

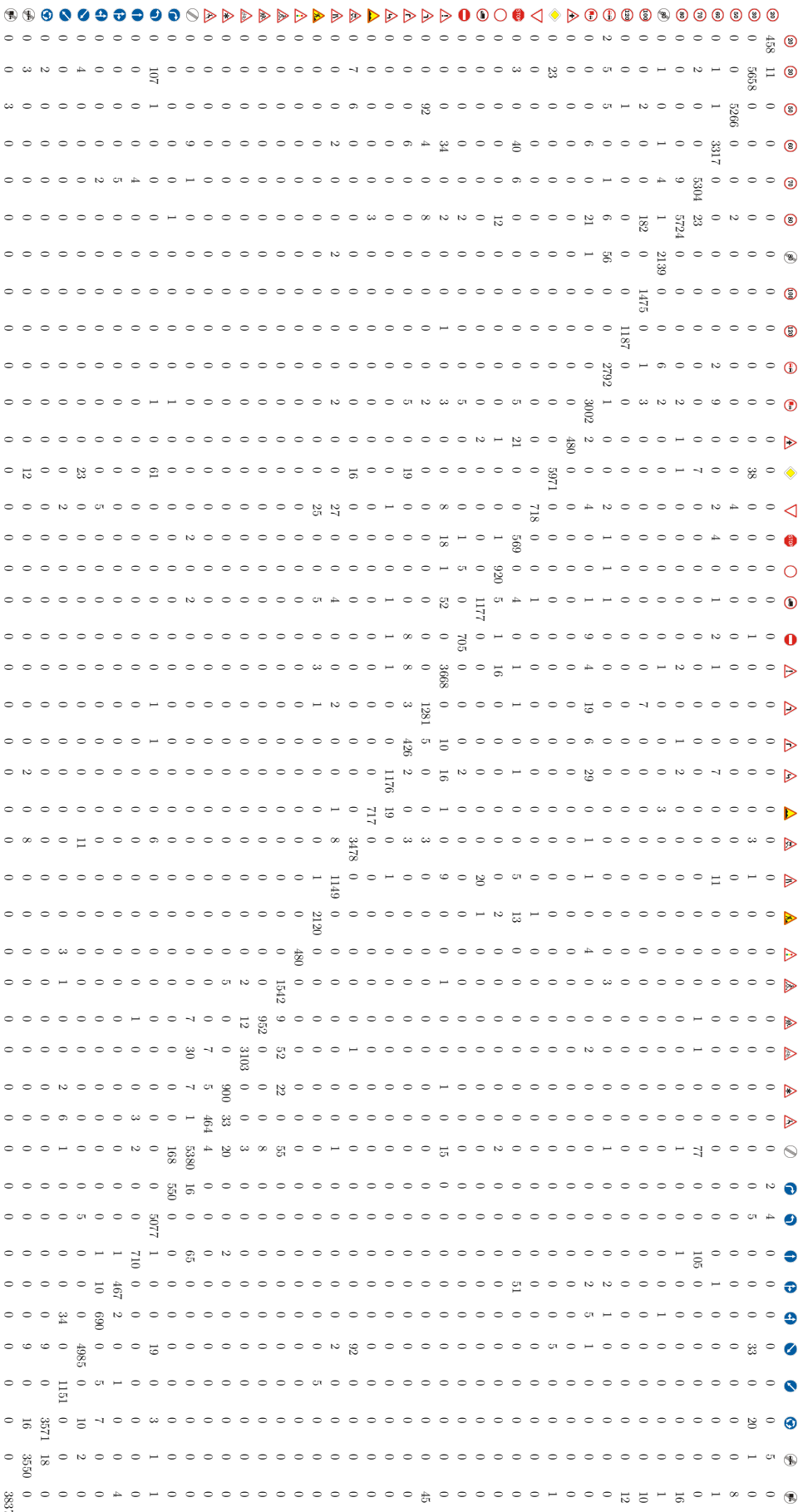


Tabela 23 – Aff-GTSRB (el=60°) - MASViT (fixed centroid) - Confusion Matrix (True × Predicted)

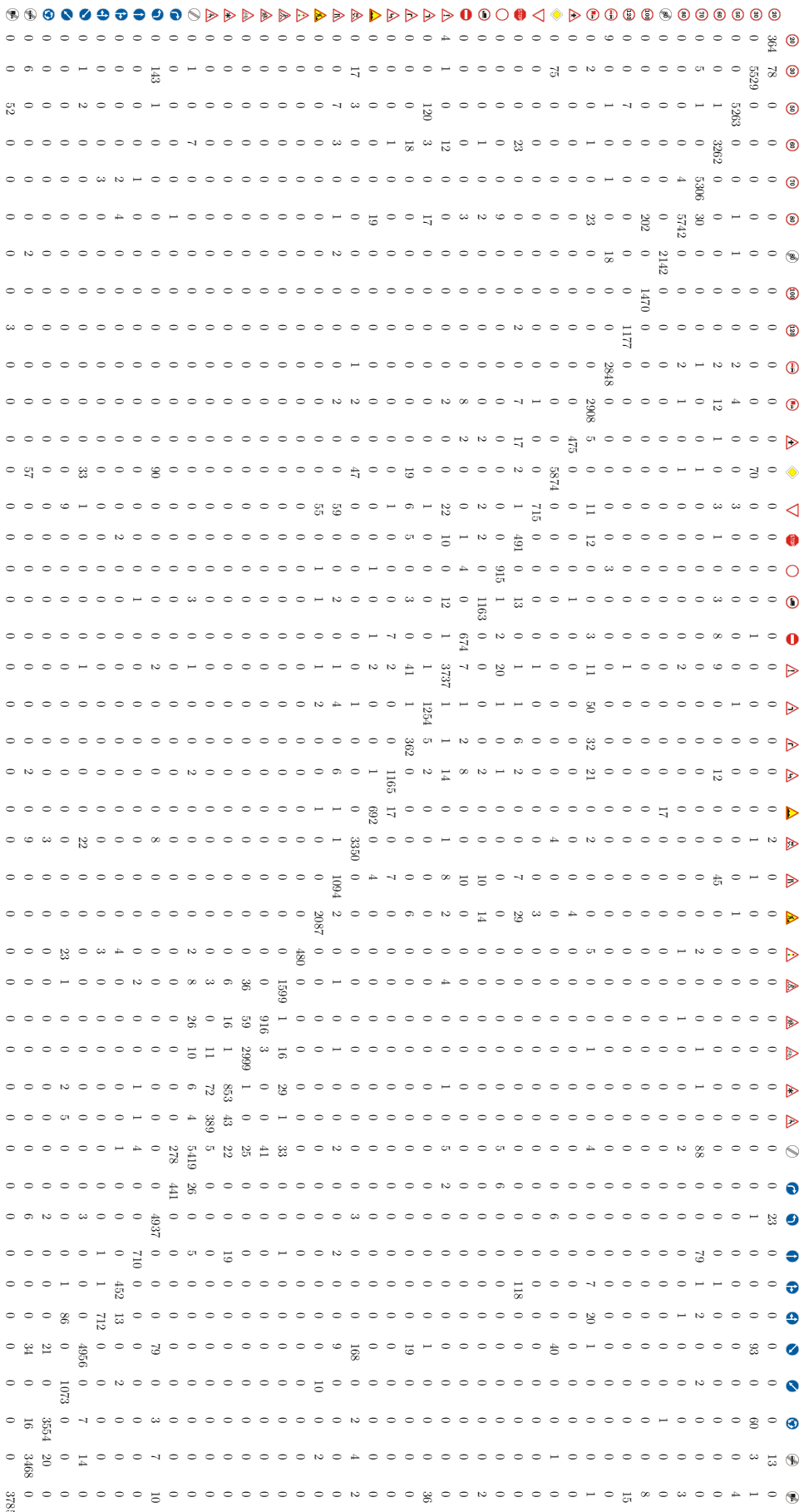


Tabela 24 – Aff-GTSRB (el=45°) - MASViT (fixed centroid) - Confusion Matrix (True × Predicted)

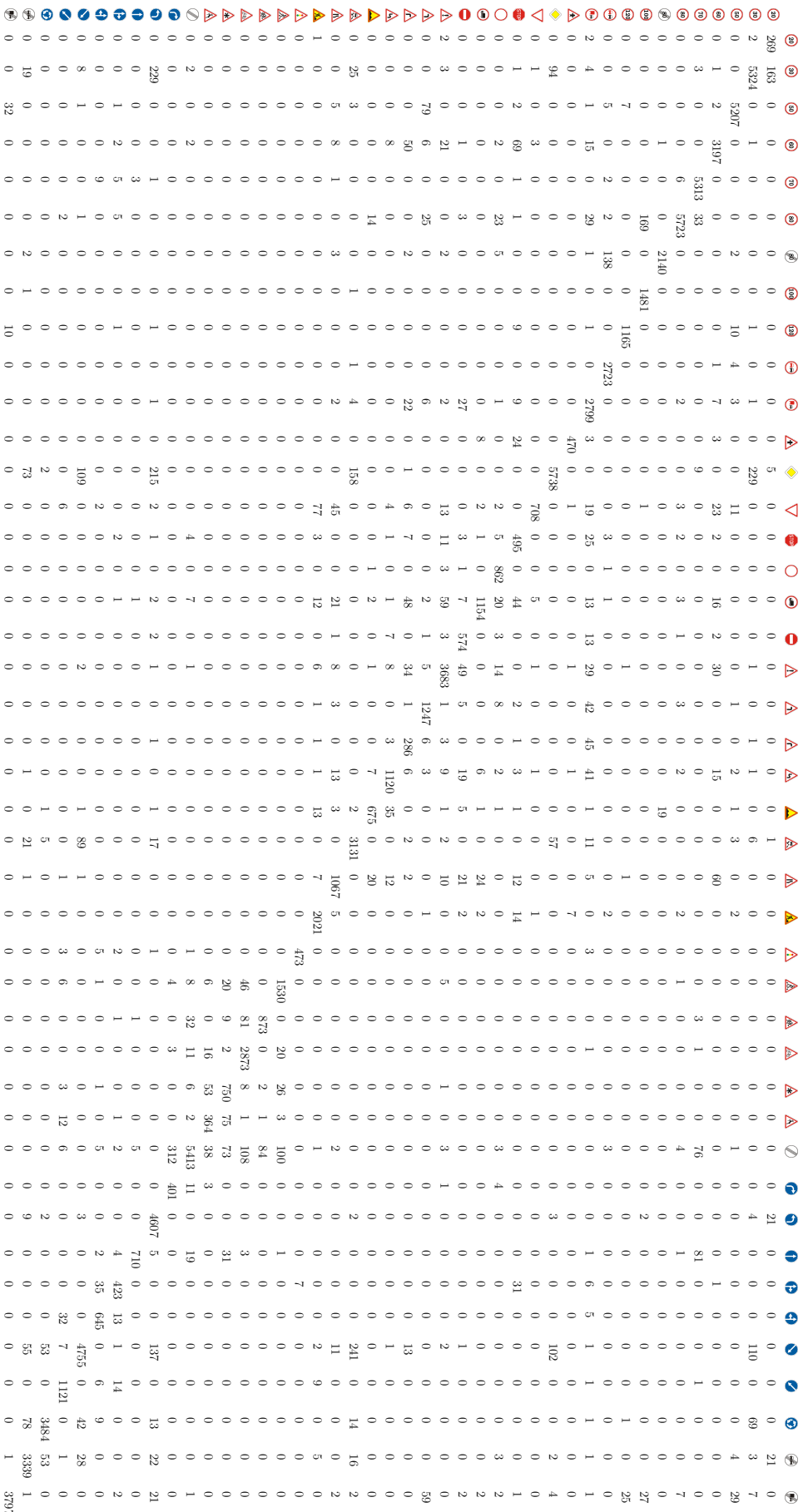


Tabela 25 – Aff-GTSRB (el=30°) - MASViT (fixed centroid) - Confusion Matrix (True × Predicted)

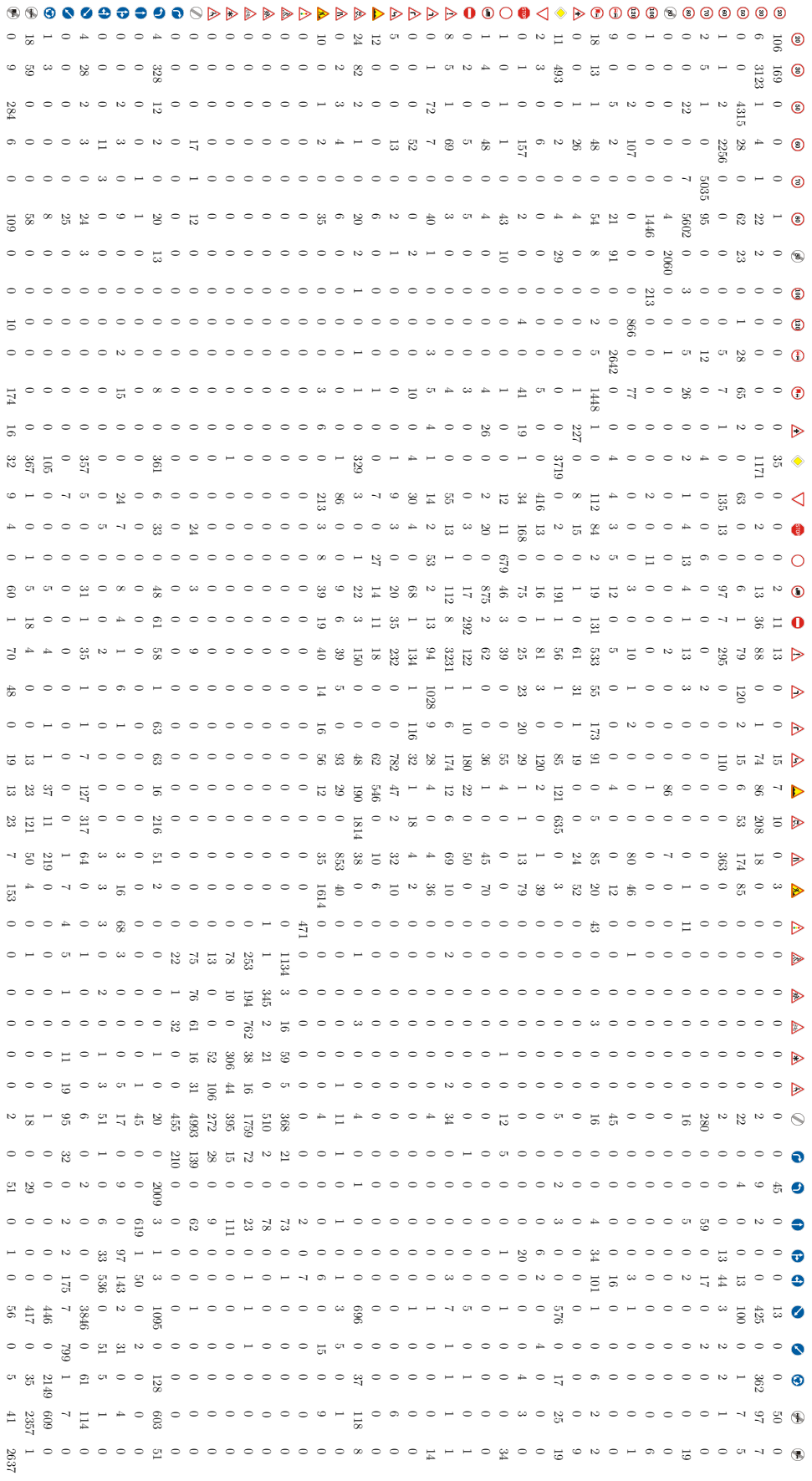


Tabela 26 – Proj-GTSRB (el=60°) - MASViT (fixed centroid) - Confusion Matrix (True × Predicted)

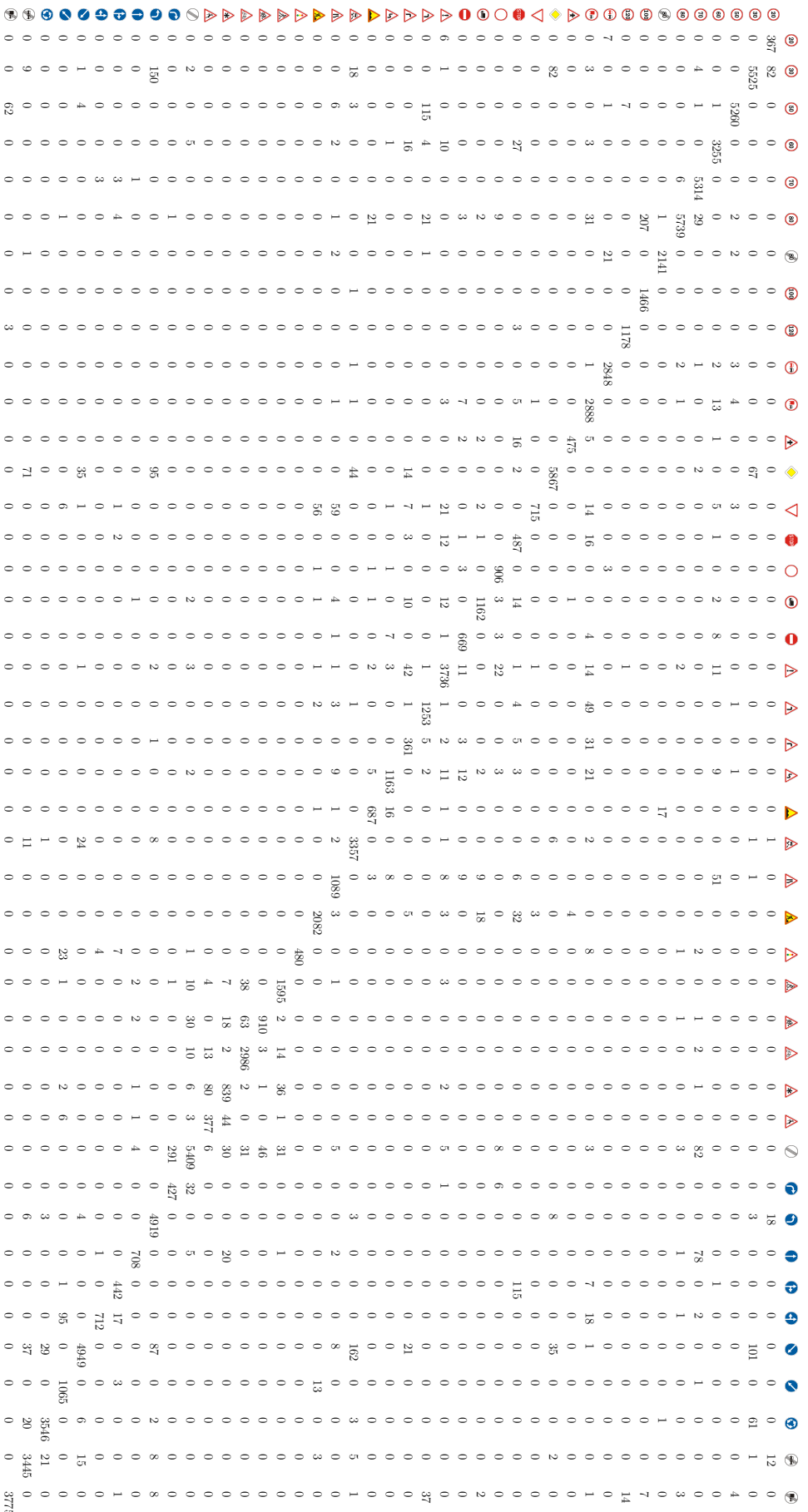


Tabela 27 – Proj-GTSRB (el=45°) - MASViT (fixed centroid) - Confusion Matrix (True × Predicted)

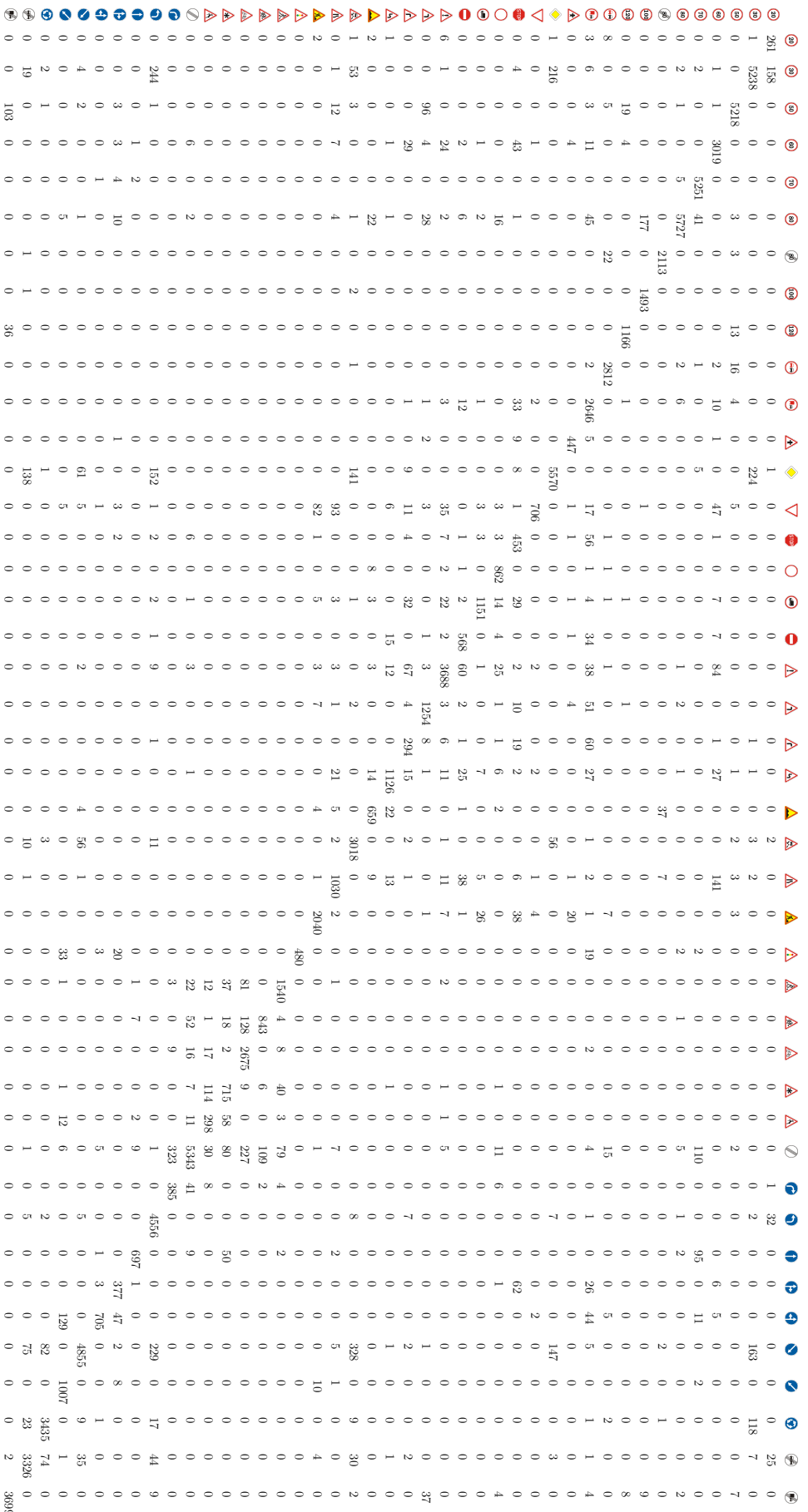


Tabela 28 – Proj-GTSRB (el=30°) - MASViT (fixed centroid) - Confusion Matrix (True × Predicted)

APÊNDICE D – APPENDIX

D.1 COMPUTING INFRASTRUCTURE FOR EXPERIMENTS

In our experiments, we used a machine running Ubuntu/Linux equipped with an AMD Ryzen 9 3900X 12-Core Processor, 64 GB of RAM, and a NVIDIA GeForce RTX 3070 Ti with 8 GB of GPU memory. The model and experiments were implemented in Python 3.8.10 using the PyTorch framework, which leverages the NVIDIA GPU via the CUDA library.

For training and validation, we used a batch size of 16 samples, resulting in a maximum GPU memory usage of 1.4 GB. Each training epoch took an average of 138.57 seconds, totaling approximately 11.5 hours over 300 epochs, using the original GTSRB train and test samples for training and validation, respectively.

For all tests with the fixed centroid approach, we used a batch size of 768 samples, which occupied 7.8 GB of GPU memory. In the test experiments, for each aff-GTSRB and proj-GTSRB dataset using the fixed centroid approach, the time spent was around 24.28 seconds, totaling 2 minutes and 50 seconds. For the GTSRB test, the time spent was 4.93 seconds.

For the max score approach tests, we used a single sample at a time, which occupied 0.7 GB of GPU memory. We limited the number of centroids for polar transformations by choosing local pixels around the center. We used a 7×7 grid to generate 49 polar images per Euclidean image instead of 48×48. Even so, the average time spent was 5 hours and 25 minutes per dataset, totaling in 37 hours and 55 minutes for all aff-GTSRB and proj-GTSRB datasets. For the GTSRB test, the time spent was 1 hour.

The Table 29 shows the times spent in each experiment category, the number of used datasets, batch sizes, and the occupied GPU memory. The total computation time was approximately 50 hours and 28 minutes. The maximum GPU memory used was 7.8 GB in Tests with fixed centroid, but the values smaller of memory can to be used in this Tests, using smaller batch size values, it will is not to affect the performance reproducible of this tests experiments. There are here only the amount of compute of latest relevant experiments to replicate the same results, the compute resources used in entire research was bigger than the reported.

Table 30 presents the number of parameters and the average inference times for MAS-

Tabela 29 – MASViT Experiment Times

Experiments	**# Datasets**	**Batch Size**	**Occupied GPU Mem. (GB)**	**Execution Time (hh:mm:ss)**
Training & Validation	2	16	1.4	11:30:00
Tests (fixed centroid)	8	768	7.8	00:02:55
Tests (max score)	8	1	0.7	38:55:00
Total				**50:27:55**

ViT and the concurrent models proposed by (ARCOS-GARCÍA; ALVAREZ-GARCIA; SORIA-MORILLO, 2018) and (CHEN et al., 2024). Inference times were measured by computing the elapsed time required for a single forward pass of the model using one data sample (or batch), since the models perform computations in parallel, the inference time remains the same whether processing a single sample or an entire batch. MASViT exhibits the highest number of model parameters and the lowest average inference time processing.

Tabela 30 – Number of parameters and average inference time of models
(*) (ARCOS-GARCÍA; ALVAREZ-GARCIA; SORIA-MORILLO, 2018)
(**)(CHEN et al., 2024)

Models	**# Parameters**	**Avg. Inference Time**
MASViT	16,781,867	≈ 3 ms
CNN + 3 Spatial Transformers *	14,678,561	≈ 9 ms
Global Routing CapsNet **	608,656	≈ 26 ms