Optimizing Vision Transformer Performance with Customizable Parameters

E. Ibrahimovic
The Second Gymnasium, Sarajevo, Bosnia and Hercegovina
emira1.ibrahimovic@gmail.com

***Abstract* – This paper experimentally examined the effects of changing the size of image patches and the number of transformer layers on the training time and accuracy of a vision transformer used for image classification. The transformer architecture was first introduced in 2017 as a new way of processing natural language and has since found applications in computer vision as well. In this experiment, we trained and tested fourteen versions of a vision transformer on the CIFAR-100 dataset using graphical processing units provided by Google Colaboratory. The results showed that increasing the number of transformer layers and decreasing the patch size both increased test accuracy and training time. However, learning curves generated by the models showed overfitting for very small patch sizes. Overall, changing patch size had a greater impact on accuracy than changing the number of transformer layers. The results also suggested that transformers are more resource-intensive to train than other models. We suppose that including a classification token could lead to shorter training times, but another experiment is needed to examine its influence on accuracy.**

Keywords* – *vision transformers; computer vision; nerual networks;

I. Introduction

Teaching machines to see and process the world around them similarly to a human is an increasingly popular task as technology is developing. Computer vision solutions are automating processes in manufacturing, surveillance, optical character recognition and robotics by reducing the amount of human error and decreasing operational costs [1], [2].

Convolutional neural networks are currently the most commonly used architecture in computer vision solutions. Even though CNNs reach accuracies as high as 91.7% on certain data sets [3], on others accuracy is limited by the amount of data and processing power available [2]. Use cases such as vehicle detection, human pose estimation and medical imaging require higher accuracies than those currently achieved [2].

New machine learning paradigms which cope with this issue are emerging and being developed at an accelerated pace [4]. The most recent development has been the introduction of the vision transformer. The transformer architecture was first introduced in [5] as a new way of processing natural language and has since become the default in the field [6].

This paper explores the effect of changing the number of self-attention layers and size of image patches on the test accuracy and training time of vision transformer models. The relationship was explored experimentally by implementing a ViT on the platform Google Colaboratory [7]. Expected results of the experiment were that test accuracy would increase as image patch size decreased. The accuracy would ideally increase as the number of self-attention layers increased. The training time was expected to increase as the number of layers increased and decrease as image patch size decreased.

II. Theoretical Background

A. Neural Networks

A neural network is a computing system which produces a set of outputs from a set of inputs by performing a complex series of computations. For classification purposes, its output is the likelihood that the input belongs to any of a number of pre-determined classes. [8]

A neural network is made up of units called nodes, which are arranged in layers. A network consists of one input layer, multiple hidden layers and one output layer. In the simplest case, layers are ordered in a row, with each layer except for the input connected to the layer before it. Each node except for those in the input layer is connected to all the nodes in the previous layer, along with one additional node named the bias unit. The connection between every pair of nodes is assigned a value called the weight. This is illustrated in Fig. 1. [9]

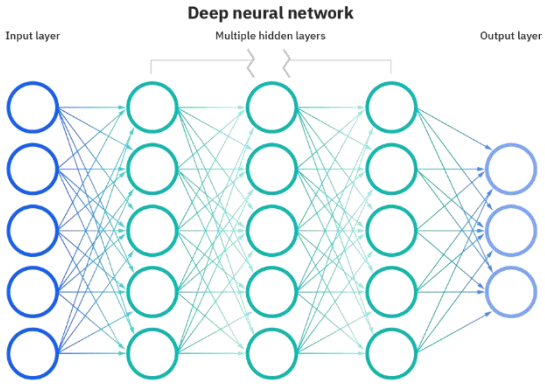

Figure 1. Each node has a weighted connection to all nodes from the previous layer [9]

The input layer contains as many nodes as there are inputs. Each input node holds a floating-point value, called the activation, corresponding to the input it represents.

The neural network performs calculations layer by layer. Each node in layer *i* takes the activation of a node from layer *i-1* and multiplies it with the weight of the connection between them. These products, along with the weight of the connection with the bias unit, are summed to get some value *S*. This value is passed to an activation function and its output becomes the current node's activation. After activations of nodes in layer *i* are all determined, the process is repeated for the following layer, stopping once the values in the output layer are computed [8].

The activations of the output nodes represent the output of the network. For classification purposes, each node in the output layer corresponds to one of the classes in the classification problem. The activation of a node in the output layer represents the neural network's certainty that the input belongs to the respective class. The class with the highest certainty is considered the network's answer to the classification problem [8].

The weights in a neural network are initially unknown and therefore assigned randomly. They are adjusted through training: a process in which a network processes a series of inputs with known outputs [8].

The type of neural network described here is a type of feed-forward neural network (FFNN) known as a Multi-layer Perceptron (MLP). It is also the basis on which all NNs work [10], [11], [9].

Convolutional neural networks have been the standard model for image classification and have shown significant success on standard computer vision benchmarks, even surpassing human-level performance on some [12]. However, there are still open problems which they face such as: difficulty to justify their successes theoretically, difficulty recognizing image manipulations designed to trick them, and difficulty giving multiple labels to images and describing the contents of an image [12].

The vision transformer (ViT), a neural network architecture inspired by the golden standard for natural language processing could prove successful in solving some of these issues. Apart from achieving higher accuracies on several public datasets than the best CNNs [13], the ViT also had a shorter training time. They have shown promise in various recognition tasks, generative modeling, multi-modal tasks, video processing, and 3D analysis [14].

B. Vision Transformers

The transformer is a neural network architecture whose job is to learn the context created by a set of inputs [15]. These inputs, called tokens, are all parts of a larger whole, e.g., words in a sentence or patches of an image. The transformer is able to use the relationships between these tokens to assign a new meaning to each of them, one which conveys the context in which each token is situated [6]. A transformer is composed of a number of identical layers called encoder blocks.

The initial input to the transformer is given to the first encoder in the stack. Each subsequent encoder receives as inputs the outputs of the encoder below it. The output of the final encoder is the output that is returned by the transformer [5]. An encoder block consists of 2 layers: a layer of self-attention, whose output becomes the input to the second layer – a feed-forward neural network. Self-attention is a transformer's primary mechanism of action.

Information about each token is embedded in a vector, where each value in the vector describes a particular aspect of the token. In the case of the vision transformer, each token embedding is a representation of a patch of an image. There are a few steps to turning an image into a sequence of token embeddings.

The input image comes in the form of a tensor: a H×W matrix, where each cell in the matrix contains C values, one for each of the primary colors, also called channels.

The image is broken down into N patches of size P×P ($N = \frac{HW}{P^2}$). The patches are then flattened into (CP^2)-dimensional vectors and made into rows of the matrix $\boldsymbol{x_p}$ of dimensions $N \times (CP^2)$. Next, these N flattened patches are transformed via a linear projection E into N D-dimensional vectors. This linear projection is learned during the training process. The results of these transformations are called patch embeddings [13].

The final piece of information that is included in the token embedding for a particular patch is its position in the image. This is done by adding a D-dimensional position embedding to the patch embedding. The position needs only reflect the patch's 1D position in the $\boldsymbol{x_p}$ matrix. The exact position embeddings are also one of the parameters that are learned through training [13].

The number of the transformer's outputs is the same as its number of inputs. Each output represents the contextualized embedding corresponding to one of the input tokens. Every output embedding will therefore contain information about the entire image, but the focus of its "knowledge" is on the specific image patch it corresponds to. It is then somewhat, but not entirely, suitable to classify the image based on one specific token [6].

For this reason, a special classification token is passed as input along with the image patch tokens. This token initially contains no information about the image, but as it is passed through the encoder stack along with other tokens, it gets imbued with context that is not localised to any specific part of the image [6]. It is the value of the output corresponding to this token that is used to classify the image at the end. The initial value of the classification token's embedding is a parameter learned through training [13].

The transformer's input can be summarized with the formula:

$$\boldsymbol{z_0} = \left[\boldsymbol{x_{class}}; \boldsymbol{x_p^1 E}; \boldsymbol{x_p^2 E}; \ldots; \boldsymbol{x_p^N E}\right] + \boldsymbol{E_{pos}} \qquad (1)$$

where: each row represents a token embedding passed as input to the transformer, $\boldsymbol{x_{class}}$ represents the initial embedding for the classification token, $\boldsymbol{x_p^i}$ represents the *i*-

th row of the matrix $\boldsymbol{x_p}$ of flattened image patches, $\boldsymbol{E} \in \mathbb{R}^{(P^2C)\times D}$ is the linear projection which transforms a flattened image patch into a D-dimensional patch embedding, $\boldsymbol{E_{pos}} \in \mathbb{R}^{(N+1)\times D}$ is the matrix of positional embeddings, where row i represents the positional embedding of token i.

Self-attention is the mechanism that allows the transformer to gain a further understanding of a token by examining the tokens which surround it. The token being contextualized in the current step is called the query token and the surrounding tokens are called key tokens.

Given a query token and a sequence of key tokens, self-attention computes a weighted average of values assigned to the key tokens. Each weight is computed during run-time and is dependent on the relationship between the query token and a corresponding key token. If the relationship is in some respect strong, the weight associated with that key will be greater, increasing the influence of the value attached to the key on the final output [6].

The process is implemented using 3 2D matrices: the key, query, and value matrices, all of which are learned during training. Each of the token embeddings received as input is multiplied by the key matrix and the query token embedding is multiplied by the query matrix. Every one of the transformed keys is then multiplied via cross-product with the transformed query to obtain an attention score, which serves as a measure of similarity between a key and the query. Then, each of the initial key token embeddings is multiplied by the values matrix and by its attention score. These final embeddings are added together to reach the final result [6].

Vision transformers use multi-headed self-attention. The sequence of operations described above is performed by every attention head, only with different parameters in the key, query, and value matrices. This way, one output embedding is calculated by each attention head. These embeddings are passed as input to a feed-forward neural network to combine them into a single embedding [6].

A vision transformer takes an image as input and segments it into N equal-sized patches. These patches are transformed via a linear projection into patch embeddings. The embeddings, along with one classification embedding, pass through a number of encoder blocks composed of multi-headed self-attention and feed-forward neural networks. The last encoder in the stack outputs N+1 contextualized embeddings corresponding to the N+1 input tokens.

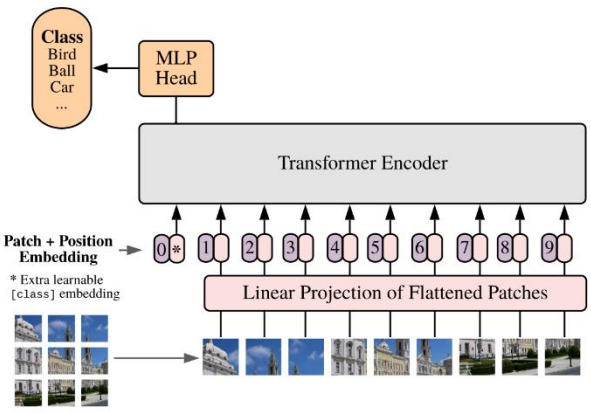

Figure 2. Overview of vision transformer architecture [13]

The embedding corresponding to the classification token is given as input to a multi-layer perceptron whose number of outputs is the same as the number of classes in the classification problem [13]. The output activations will be the desired probability distribution for the input image over all classes in the dataset. A complete overview is given in Fig. 2.

III. EXPERIMENTAL METHODOLOGY

An experiment was conducted to obtain primary data which will be used to answer the research question. 14 versions of a vision transformer were trained and tested on the platform Google Colaboratory [7]. All code was run on graphical processing units provided by Colab. The ViTs were programmed in Python 3.0 using TensorFlow's Keras, adapting code by [16].

A. *Variables*

Independent variables in models are:

- Number of transformer layers – The values chosen for the experiment were 2, 4, 8, 16. The values are in geometric progression so that changes in the dependent variables are clearly noticeable.
- Image patch size – The values chosen for the experiment were 2px, 4px, 8px, and 16px. Images in the data set are 32px × 32px.

The two independent variables were manipulated within the code. The only exceptions to the range of values chosen were pairs (P, L) ∈ {(2,8), (2,16)} because the training of these models exceeded usage limits set by Colab.

We examined the following dependent variables:

- Training time – When training a TensorFlow model, an overview of the training process is output to the terminal. Among the values in this summary is the time taken to train the model. This is the value taken for this dependent variable.
- Accuracy – Among the values in the overview of model training is the test accuracy achieved after training. This is the value taken for this dependent variable. Accuracy is the quotient of the number of images whose classes were correctly identified and the total number of images in the test set. Accuracy was chosen as an evaluation metric to be able to compare results to [3], [13].

B. *Dataset*

The dataset used for the training and testing of the models is CIFAR-100 which contains 60 000 images of dimensions 32× 32 pixels [17]. Each image is assigned to a class which describes the contents of the image. There are 100 classes with 600 images per class, examples of which can be seen in Figure 3 below.

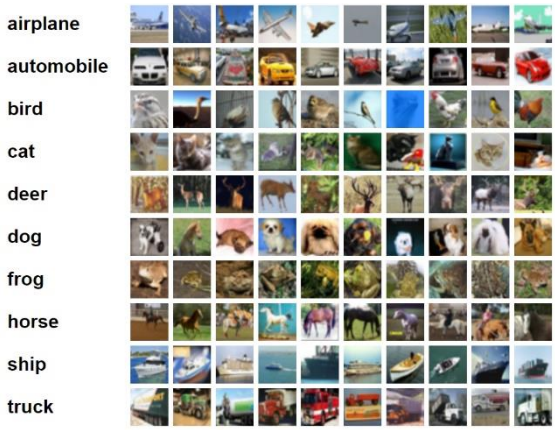

Figure 3. 10 randomly selected images from 10 classes in CIFAR-100

During each run, the dataset was split into 3 groups called the training, validation, and test sets. For each run, the test set consisted of the same set of 10 000 randomly selected images such that there are 100 images in each of the 100 classes. The validation set always consisted of 5 000 images and the test set of the remaining 45 000, though the exact images were randomly selected at the start of each run.

C. *Model Layers*

Each of the 14 models consisted of 3 parts: the input image processing layer, multi-headed self-attention, and the classification head. In the input processing layer, $\frac{32^2}{P^2}$ image patches of dimensions P×P are created and linearly projected into $\frac{32^2}{P^2}$ patch embeddings of dimension 32. The parameters of the linear projection are learned during training. This is implemented with Keras layer Conv2D [18].

Next, come L encoder blocks are implemented according to [13]. The first layer in the block is Keras Layer Normalization [19], which normalizes the activations of the layer that came before it. Doing this helps reduce training time [20]. This becomes the input to Keras Multi-Head Attention layer with 4 attention heads [21]. The outputs of this layer are added to the inputs to the block and passed to another normalization layer. Then, an MLP with one 64-node layer and one 32-node layer is attached. Finally, the outputs of the MLP are added to the outputs of the multi-head attention layer and the inputs to the block.

The classification head first normalizes the outputs of the final MHA block, then flattens the set of embedding vectors into a single vector. This is passed to a dense layer with 2048 nodes and then to another with 1024 nodes. A dense layer with 100 nodes corresponding to the 100 classes in CIFAR-100 gives the final output.

The models were trained in 200 epochs with batch size 100.

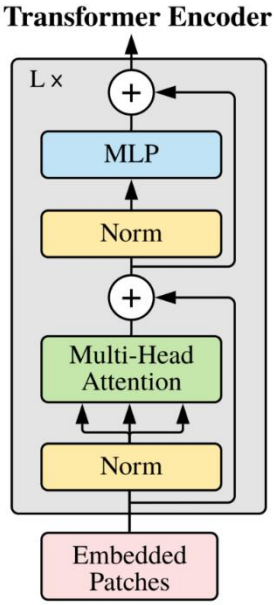

Figure 4. Multi-Headed Attention Layer Block

IV. RESULTS

A. *Model Training Time*

Training models with more transformer layers resulted in longer training times, as is shown in TABLE 1. Decreasing patch size, and therefore increasing the number of patches led to longer training times, with the exception of the transition from (P=16, L=4) to (P=8, L=4). These results have a common reason. Both increasing the number of patches and the number of self-attention layers resulted in a larger number of trainable parameters in the model (TABLE 2).

After processing each batch of input images, the transformer adjusts a large proportion (determined by the dropout rate) of parameters. Each calculation resulting from inputting an image takes one operation per parameter, and each adjustment of a parameter requires a machine to do at least one operation. Considering the fact that the number of operations a machine has to do is directly proportional to the time taken to complete them all, it follows that increasing the number of parameters would necessarily lead to an increase in training time. However, the decrease in training time seen in the transition from (P = 16, L = 4) to (P = 8, L = 4) where the increase in the parameter number is still present suggests that the relationship between parameter number and training time is not so straightforward.

TABLE 1. TRAINING TIME IN SECONDS [22]

Size of patch	Number of self-attention layers			
	2	4	8	16
16	802	1143	1171	3121
8	803	1059	1930	3303
4	1121	1814	3219	6011
2	5931	10 895	×	×

TABLE 2. NUMBER OF TRAINABLE PARAMETERS [22]

Size of patch	Number of self-attention layers			
	2	4	8	16
16	2 531 780	2 574 020	2 6585 00	2 827 460
8	3 299 780	3 342 020	3 426 500	3 595 460
4	6 440 900	6 483 140	6 567 620	6 736 580
2	19 022 660	19 064 900	×	×

Decreasing patch size from P=8 to P=4 with L=16 leads to an increase in parameter number of 3 141 120 and a near doubling of training time. At the same time, increasing the number of layers from L=8 to L=16 with P=4 leads to an increase in parameter number of just 168 960 but a near doubling in training time. This suggests that some parameters either require more operations to be adjusted or more operations are needed to compute outputs in a specific layer. The increase in parameter number resulting from a decrease in patch size is almost entirely due to the third to last layer in a model: a dense layer converting all transformer outputs into outputs. Adding another transformer layer increases parameter number by 21 120, regardless of the initial shape of the input. This result complied with a study performed by [23].

Examining the structure of these two layers gives us a possible explanation as to why one is more computationally intense than the other. During self-attention, each of the N outputs need to consider all N outputs from the previous layer, all of which vectors, and perform a series of matrix multiplications to get a result. This means that calculating all N outputs result in an order of magnitude of N^2 operations, assuming that matrix multiplication takes an order of magnitude of 1 operations. Given that the models are working with 4 attention heads each and with L layers, the final number of operations for this stack of layers is an order of magnitude of $N^2 \times L$. On the other hand, the third final dense layer in the model flattens these N vectors into a single vector of dimension $32 \times N$ which becomes its input, and returns 2048 outputs. Each of these 2048 outputs require the processing of all inputs, which is done by a single matrix multiplication, resulting in a number of operations in an order of magnitude of N. This means that as N and L increase, the time taken for the transformer encoder to process the image grows at least a whole order of magnitude faster than the time it takes for the MLP to process it.

This does not give an explanation for the decrease in training time from (P = 16, L = 4) to (P = 8, L = 4) The best answer which can be offered is either randomization steps during training, or the Google Colab platform assigning different types of GPUs to these two runs.

B. Model accuracy

Decreasing patch size always led to an increase in test accuracy, as is shown in TABLE 3. Increasing the number of self-attention layers generally led to an increase in accuracy, with the exception of the transition from (P = 16, L = 4) to (P = 16, L = 8). However, none of the accuracies obtained with these models comes close to accuracies achieved by convolutional neural networks of comparable size [3]. Searching for the reasons for this leads us to examine the learning curves produced by the models.

Learning curves [22] for models with the same patch size have nearly the same shapes, whereas learning curves for models with the same number of transformer layers differ greatly. This tells us that patch size could have a greater impact on accuracy than the number of layers. The test accuracy generally displays this pattern, despite great variation in the ranges of values obtained for one fixed variable, as shown in

TABLE 4 and TABLE 5 below. We can see that range expressed as the percentage of the mean when the number of transformer layers is kept constant is nearly always greater than the same value when patch size is kept constant. This suggests that changing patch size has a greater percentage impact on accuracy than changing the number of transformer layers does.

Observing the learning curves for P=2, P=4, and P=8, we determine they are consistent with signs of overfitting. For example, on the accuracy graph for P=4 ∧ L=8 we see that training accuracy decreases logarithmically as epochs go on, reaching a final accuracy of around 80%. The validation accuracy increases significantly within the first 25 epochs, then slows down its increase until it plateaus after the 100th epoch at around 40% accuracy. Overfitting can be solved by having a dataset with more training images, which could suggest why the ViT trained by [13] achieved such high accuracies: their models were pre-trained on datasets with 15 and 300 million images.

We can see that with increasing patch size the training and validation accuracy curves began to converge, as did the training and validation loss curves. This is consistent with signs of underfitting. Even though the training and validation accuracy curves for P=16 converge, they are both worryingly low at around 28% during the entire run. Increasing patch size while keeping the projection dimension constant led to the network trying to encode more data into the same projection vector, losing useful information in the process.

TABLE 3.TEST ACCURACY (%) [22]

Size of patch	Number of self-attention layers			
	2	4	8	16
16	28.77	29.31	28.8	29.48
8	32.76	35.12	36.17	36.94
4	35.20	39.43	41.81	44.19
2	35.82	38.88	×	×

TABLE 4. RANGE OF ACCURACIES FOR CONSTANT PATCH SIZE [22]

Size of patch	Range of accuracies	Mean accuracy	Range as % of the mean
16	0.71	29.09	2.44
8	4.18	35.25	11.86
4	8.99	40.16	22.39

TABLE 5. RANGE OF ACCURACIES FOR CONSTANT NUMBER OF TRANSFORMER LAYERS [22]

Number of layers	Range of accuracies	Mean accuracy	Range as % of the mean
2	7.05	33.14	21.27
4	10.11	35.69	28.33

V. CONCLUSION

This paper experimentally investigated the effects of changing the size of image patches and number of transformer layers on the training time and test accuracy of a vision transformer.

The results confirmed that training time increases with decreasing patch size and increasing number of transformer layers. Both of these changes correspond to an increase in the number of trainable parameters in the model, which leads to increasing the number of operations a machine has to perform, ultimately resulting in more training time. The results also showed that adding transformer layers has a greater impact on increasing training time than decreasing patch size. In particular, the change in training time per change in parameter number was generally significantly greater for parameters added by transformer layers. This suggests that transformers are inherently more resource intensive to train, confirming results by [23].

The experiment results have shown that models increasingly exhibited signs of underfitting as patch size increased, as well as that they overfit for small patch sizes. Increasing the number of transformer layers was shown to always increase the test accuracy of the models. However, this trend does not promise to continue indefinitely, since it could lead to overfitting and a corresponding fall in accuracy. Changing patch size has a greater impact on accuracy than changing the number of transformer layers.

There are a number of limitations to the chosen methodology. Firstly, GPU type was not controlled for. Google Colab is likely to have assigned different GPUs to different model runs, which could have influenced training time data, and perhaps led to the inconsistency previously observed. Secondly, because of usage limitations for free accounts on Google Colab, we were unable to run experiments for two additional values of P and L, resulting in a limited overview of the effects of the independent variables on training time and accuracy.

Though it is likely that including a classification token could lead to shorter training times, it is a question what its influence would be on the accuracy. Another experiment could be conducted to determine the exact relationship.

REFERENCES

[1] "Computer Vision Use Cases in Manufacturing | Altamira Softworks." https://www.altamira.ai/blog/computer-vision-in-manufacturing/ (accessed Nov. 06, 2022).

[2] A. A. Khan, A. A. Laghari, and S. A. Awan, "Machine Learning in Computer Vision: A Review," EAI Endorsed Transactions on Scalable Information Systems, vol. 8, no. 32, pp. e4–e4, Apr. 2021, doi: 10.4108/EAI.21-4-2021.169418.

[3] "CIFAR-100 Benchmark (Image Classification) | Papers With Code." https://paperswithcode.com/sota/image-classification-on-cifar-100 (accessed Nov. 06, 2022).

[4] R. Pugliese, S. Regondi, and R. Marini, "Machine learning-based approach: global trends, research directions, and regulatory standpoints," Data Science and Management, vol. 4, pp. 19–29, Dec. 2021, doi: 10.1016/J.DSM.2021.12.002.

[5] A. Vaswani et al., "Attention Is All You Need." 2017. doi: 10.48550/arXiv.1706.03762.

[6] I. Turc, "Transfer learning and Trasformer models (ML Tech Talks)," YouTube, Jun. 22, 2021. https://www.youtube.com/watch?v=LE3NfEULV6k

[7] "Google Colaboratory," Google. https://colab.research.google.com/

[8] "Machine Learning - Linear Regression with Multiple Variables | Coursera." https://www.coursera.org/learn/machine-learning-course/home/week/4 (accessed Nov. 26, 2022).

[9] "What are Neural Networks? | IBM." https://www.ibm.com/cloud/learn/neural-networks (accessed Nov. 26, 2022).

[10] ["Feed Forward Neural Network Definition | DeepAI." https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network (accessed Nov. 27, 2022).

[11] "Multilayer Perceptron Definition | DeepAI." https://deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron (accessed Nov. 26, 2022).

[12] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," Neural Comput, vol. 29, no. 9, pp. 2352–2449, Sep. 2017, doi: 10.1162/NECO_A_00990.

[13] A. Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." 2021. doi: 10.48550/ARXIV.2010.11929.

[14] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in Vision: A Survey," ACM Computing Surveys (CSUR), vol. 54, no. 10s, pp. 1–41, Sep. 2022, doi: 10.1145/3505244.

[15] Y. Kilcher, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (Paper Explained)," YouTube, Oct. 04, 2020. https://www.youtube.com/watch?v=TrdevFK_am4

[16] K. Salama, "Image Classification with Vision Transformer," Keras.io, Jan. 18, 2021. https://keras.io/examples/vision/image_classification_with_vision_transformer/

[17] A. Krizhevsky, G. Hinton, and et al., "Learning Multiple Layers of Features from Tiny Images," Toronto, ON, Canada, 2009.

[18] "Conv2D layer." https://keras.io/api/layers/convolution_layers/convolution2d/ (accessed Nov. 25, 2022).

[19] "LayerNormalization layer." https://keras.io/api/layers/normalization_layers/layer_normalization/ (accessed Nov. 25, 2022).

[20] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," Jul. 2016, doi: 10.48550/arxiv.1607.06450.

[21] "MultiHeadAttention layer." https://keras.io/api/layers/attention_layers/multi_head_attention/ (accessed Nov. 25, 2022).

[22] E. Ibrahimović, "Investigating the Effects of Customizable Parameters of Vision Transformers on Their Performance," Extended Essay, International Baccalaureate Organization, 2022.

[23] L. Liu, X. Liu, J. Gao, W. Chen, and J. Han, "Understanding the Difficulty of Training Transformers," EMNLP 2020 - 2020 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference, pp. 5747–5763, Apr. 2020, doi: 10.48550/arxiv.2004.08249.