university of groningen / faculty of science and engineering / bernoulli institute

Classifying Gemstones using two Multilayer Perceptron architectures

Andra Mladen, s4749987, a.mladen@student.rug.nl
Aurel Istrate, s4718143, c.a.istrate@student.rug.nl
David Vanghelescu, s4683889, d.vanghelescu@student.rug.nl
Vlad Muscoi, S4718267, v.n.muscoi@student.rug.nl

Abstract:

Multilayer perceptrons (MLPs) are a type of artificial neural network that can learn to perform various tasks, such as image classification. In this paper, we present our work on developing and testing an MLP model that can classify images of different types of gems. We used a relatively small dataset containing 87 classes of different gems, each containing around 30 images. We designed two MLP architectures that differed in size: the first had two hidden layers with relatively few neurons while the second had three layers with many more neurons. We evaluated our models on the test set and compared each other's accuracy. We found that our MLP models achieved similar accuracy values of around 70% for a restricted class range and 40% for all 87 classes. We conclude that MLPs are a powerful and versatile tool for image classification, and that our project was a valuable learning experience for us.

Contents

1 Introduction

Usually, when thinking about doing a project, one first thinks of a topic and then gathers or finds data for the selected topic, but we took a different approach. We first searched for an interesting dataset and then came up with a project topic dependent on that dataset. We looked far and wide across the wilderness of Kaggle datasets and we managed to strike gold, or rather diamonds! The "Gemstones Images" dataset [1] contains pictures of different types of gemstones that we can use to create our model. The dataset is small enough on its own, but we decided to use an even smaller subset of it to allow us to experiment with different architectures without spending too much time retraining the model.

Equipped with a compact yet reliable data set, we ventured out to create a Neural Network model that can accurately classify images of gemstones. With the purpose of gaining a better understanding of Neural Networks and Machine Learning, our approach to tackling this project involved the following steps, which will be further elaborated on in the subsequent sections of the report:

- Selection of data set
- Development of a data processing pipeline
- Exploration of the optimal network architecture with considerations such as the number of layers, activation functions and optimisers
- Selection of an appropriate performance metric and loss function
- Division of the training data into training and validation folds in order to perform Cross-Validation
- Initial model training on a subset of the data
- Adjustment of the model based on observed results
- Training of the updated model on the entire data set
- Comprehensive discussion and reporting of the outcomes and the overall process

2 Data

In this section, we will discuss our chosen dataset, as well as the reasoning for our choice. The selected data set is "Gemstones Images" and it is available on Kaggle [1]. It was composed by gathering images from online resources dedicated to informing about, respectively selling gemstones: `www.minerals.net` and `www.rasavgems.com`.

The data set is composed of 3219 images of different gemstones. They are divided into 87 classes with an unequal number of images, each class representing a different type of gemstone. The classes include pictures of the gemstones in various forms (round, oval, square, rectangle, heart). The gemstones are photographed against different backgrounds: solid colour, gradient or textured. Some of the images also depict additional elements such as shadows or forceps holding the gem.

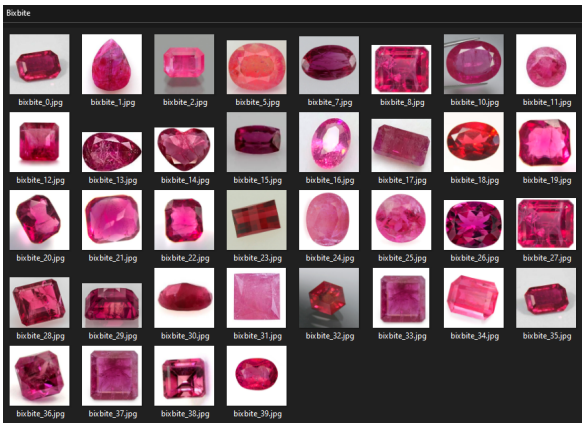


Figure 2.1: An example of a class of images.

The dataset is already split into a training and testing set, we will maintain this separation for the possibility of comparing our results with other users from Kaggle.

Because of the relatively large number of classes, we decided to choose and work with a subset of the dataset which will be used while we get familiar with designing our first Artificial Neural Network (ANN). This way, we can save time while training and validating models between changes in their architecture. We made a Python script that analyzed our data set by counting the total number of images for both training and testing folders of each class and sorting them in ascending order of total training images. We then chose 12 gemstone

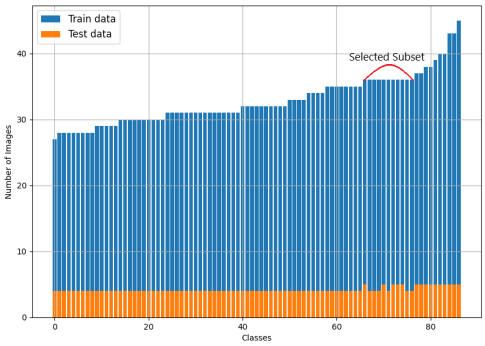


Figure 2.2: A plot depicting the distribution of images from our dataset.

classes that had equal amounts of training images. The image below shows the distribution of data in our data set and the 12 classes that we used while building our model.

The images themselves are in .jpeg file format and of different resolutions, thus needing preprocessing before being fed into the model. The preprocessing will be done when the data will be loaded into the model. More details about this procedure will be listed in a following section

3 Methods and Experiments

In this section, we will present in detail each step of our project. For our implementations, we decided to use the *PyTorch* library, since there is an abundance of documentation and resources detailing its inner workings as well as proper use cases. For all of our experiments, we used *Python* 3.11.4, with *PyTorch* 2.0.1 for *CUDA* 11.8. We used the platform *GitHub* for synchronising our work across multiple devices (`https://github.com/caistrate/NN_Project`).

3.1 *FirstModel*

Our first attempt at creating a Neural Network, appropriately named *FirstModel*, was designed to be as simple as possible, acting as a proof of concept. Thus, it was designed as a Multilayer perceptron (MLP) with three fully-connected layers. The input and output layers were purposefully made with a variable number of neurons, to allow testing on a variable number of classes and a variable image size. The number of neurons in the described model was $l * l * 3$ for the input layer, where l is the used image size, 400 for the first hidden layer, 200 for the second hidden layer, and n for the output layer, where n is the number of classes the model was trained on. Due to the relatively small size of the model, all training runs were done using GPU-accelerated computations, on a NVIDIA GTX 1650 GPU.

3.1.1 Activation Function

For the hidden layers, the initial attempt implemented the hyperbolic tangent activation function ($tanh$). This produced undesirable results, potentially because of the vanishing gradient problem.

Subsequently, our attempt with the rectified linear unit ($ReLu$) activation function showed significant improvement in our results. It is formally defined as:

$$ReLu(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & otherwise \end{cases} \tag{3.1}$$

The motivation behind the $ReLu$ implementation stems from this function's ability to capture intricate image features. The nonlinearity of this activation function allows complex relationship learning throughout all the hidden layers. The function has a large derivative (a steep slope), meaning that the output is very sensitive to input changes, which can make the network highly perceptive to fine details and ultimately improve its classification performance.

$ReLu$ also presents an advantage compared to other activation functions such as the $Sigmoid$ function or the $tanh$ function because not only is it computationally simple, requiring only a comparison,it also helps overcome the vanishing gradient problem which causes the network to learn at a slow rate. When the input takes a wide range of values, the $Sigmoid$ and $tanh$ activation functions can cause the gradients to diminish as they backpropagate resulting in very small weight updates in the early layers which minimises the impact of these neurons in the network.

We did not implement an activation function in the last layer of our gemstone classifier neural network. Because the $ReLu$ function is unbounded and outputs 0 for negative values, it does not produce values fit for a probability distribution as is. In order to obtain a probability distribution across the various classes corresponding to potential types of gemstones, a different activation function such as $softmax$ can be implemented in the last layer. However, we decided to use no activation function for the last layer, considering instead the class with the highest output value as the class predicted by the model.

The model architecture allows multi-class learning to be performed if a different loss function is implemented.

3.1.2 Data Preprocessing

Our manual check of the data revealed that the images are consistent enough to allow the preprocessing step to be performed fully automatically. The preprocessing we applied to the input data was a simple resize to a square form factor of a fixed size of 300, followed by a centered crop of 250. After processing, the images are converted to tensors, which are interpretable by the model. These tensors are used in the training process in batches of 50.

3.1.3 Loss Function

For our loss function, we chose the mean Cross Entropy Loss function defined as

$$L(\hat{y}, y) = \frac{1}{N}\sum_{n=1}^{N} -log\frac{exp(\hat{y}_{n,y_n})}{\sum_{c=1}^{C} exp(\hat{y}_{n,c})} \quad (3.2)$$

where N is the number of observations and C is the number of classes, $\hat{y}$ is the output of the model, meaning a vector of N logits of size C, and y is the vector of the true labels for the N observations. A logit is a vector describing the output of a classification model, assigning a value representing the predicted likelihood for each class. In our case, we deem the index of the highest value in a logit to be the class predicted by the model.

For each logit $\hat{y}_i$, the proportion of exponential of the value representing the true class to the sum of all exponents of the values in the logit. Since the negative logarithm of these calculations is taken, proportions close to the value of 1 will yield small losses, while ones close to 0 will result in large losses. This behaviour ensures that incorrect predictions are penalized more than correct ones, thus encouraging the model to learn features that discriminate between different classes of images [2]. Cross entropy loss also has some desirable properties, such as being convex and differentiable, which make it easier to optimize using gradient-based methods.

3.1.4 Update Step

For the update step of our model, we decided to use the *Adam* optimiser. This method is based on the iterative gradient descent method, but differs in the fact that every parameter has a different learning rate that can change during the training process. It is a popular algorithm for gradient-based optimization of neural networks as it combines the advantages of two other optimisers: *AdaGrad*, which adapts the learning rate to each parameter, and *RMSProp*, which uses a moving average of squared gradients to avoid large fluctuations. The Adam optimizer also introduces a bias correction term to account for the initial low values of the moving averages [3].

Our understanding of the inner workings of the *Adam* optimiser is not complete; however, we are aware that the basic idea is the same as with gradient descent, with separate learning rates for each of the parameters that are modified based on metrics calculated by the algorithm. Our choice was determined by many of the resources we consulted suggesting the use of this particular optimiser [4, 5], including our one-on-one meeting for the semester project.

This choice also presented us with a hyperparameter, namely the learning rate, which will be discussed further down.

3.1.5 Regularization

In order to avoid overfitting, we implemented the L2 regularization technique, also referred to as weight decay. By adding to the loss function a penalty term that accounts for the complexity of the model and fosters simpler weight configurations, the model shows better performance on unseen data. The penalty term is calculated by summing up the square values of all feature weights:

$$L2_{penalty} = \lambda \cdot \|w\|^2 \quad (3.3)$$

The choice of L2 is motivated by the method's simplicity and ease of implementation, its stability provided by the increased penalty for large weights which leads to a more even distribution of importance across features, and the adjustability of the λ parameter which enables control over the level of regularization.

3.1.6 Performance Metric

Depending on the choice of function and the regularisation method utilised, the loss can be a bad approximation for the performance of a model. The variability in its actual value also renders it a bad metric for comparing the performance of separate models. Because of this, separate performance metrics are used when discussing the performance of models. In our case, we decided upon the accuracy performance metric. Accuracy represents the proportion of correctly classified inputs over the total number of inputs in the dataset. Accuracy has the great advantage of being easy to understand as well as easy to compute, but it is only usable for classification tasks and only on balanced datasets, as it is prone to bias. However, since our dataset is mostly balanced, this did not represent a concern.

3.1.7 Hyperparameter Tuning

The hyperparameters for which appropriate values had to be found were the learning rate and the regularization coefficient. To find the combination of hyperparameter values that maximise the performance of the model, we used a simple grid search. For the learning rate, we decided upon the search space $\{1 \times 10^{-3}, 1 \times 10^{-4}, 5 \times 10^{-5}, 1 \times 10^{-5}, 1 \times 10^{-6}\}$ and for the weight decay parameter $\lambda \in \{1 \times 10^{-1}, 1 \times 10^{-2}, 5 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}\}$, resulting in 25 possible combinations. The values were chosen due to the tendency of the tuned parameters to show changes in the performance of the model only for changes in the order of magnitude of the value.

To select the best pair of hyperparameters, each combination was tested in a cross-validation scheme. For this, the training set was split into a 40% validation set and a 60% training set. Each model

was trained and evaluated for 30 epochs. The selected set of parameters was chosen based on the cross-validation model that had the highest mean accuracy for the last 10 epochs of the training run. The final model was then trained for 100 epochs, on the original training set.

3.2 *SecondModel*

The first model was limited in size and capabilities and as such, we created a new one based on it, creatively called *SecondModel*. The only changes come in the form of adding an additional hidden layer and increasing the number of neurons across all of the layers. The input layer stayed the same, the size of the first hidden layer was increased from 400 to 2000 neurons, of the second from 200 to 1000 neurons and a third hidden layer was added in this model with 500 neurons. Finally, the output layer stayed the same. We expected that an increase in size would, in turn, increase the performance of the model.

4 Results

4.1 *FirstModel*

We ran the model for both 12 classes which represents a smaller part of the data set and the full data set comprising 87 classes. For the subset, our model managed a training accuracy of 100% and a testing accuracy of 70%, with a training duration of around 25 minutes (including cross-validation). The resulting graphs can be observed in Figure 4.1.

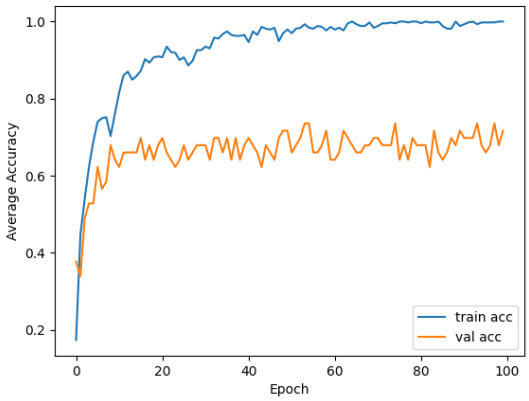
Furthermore, for the entire data set the model scored 97% in training and 44% during the validation period, with a training duration of approximately 2 hours and 15 minutes. The resulting graphs can be seen in Figure 4.2

However, when looking at the raw data set, we can see that some classes (types of gemstones) are similar which can confuse the model. That can be seen when running for the full data set which only gives an accuracy of 44%. An example of such similarity can be found in Figure 5.1. This happens because we use a small network and as such, its capabilities are limited.

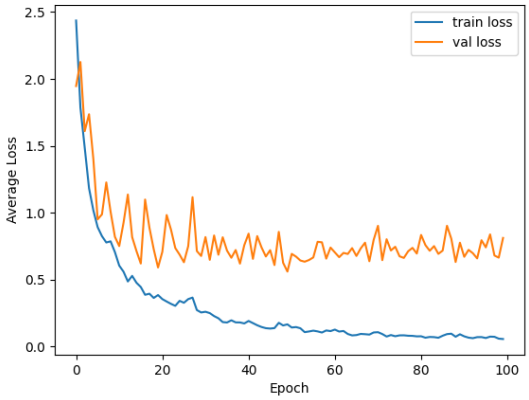
4.2 *SecondModel*

After running this model for the 12 class subset we got a training accuracy of 100% and a testing accuracy of 72%, with a training duration of 3 hours and 15 minutes. We can observe an approximate one to two percent increase in performance from the previous model.

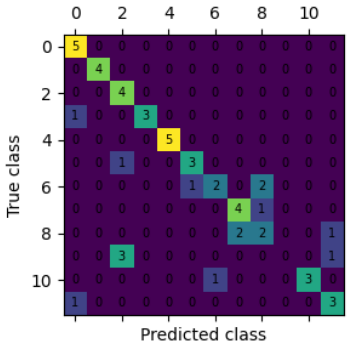
Furthermore, after running for the full data set, we got a training accuracy of 99.3% and a validation accuracy of 50%, with a training duration of 5 hours and 1 minute, scoring slightly better than the previous model.



(a) Accuracy of the FirstModel trained on the 12 classes subset.

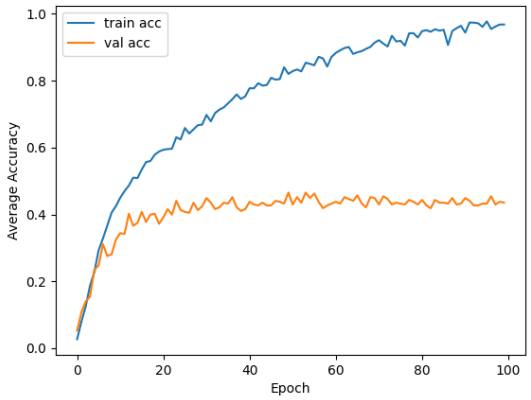


(b) Loss of the FirstModel trained on the 12 classes subset.

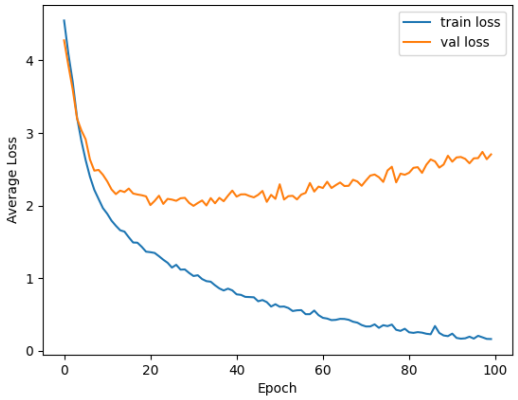


(c) Confusion Matrix of the FirstModel trained on the 12 classes subset.

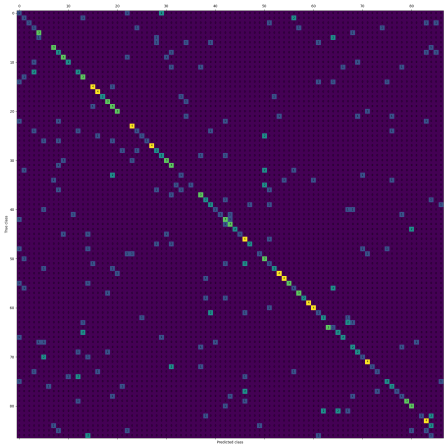
Figure 4.1: Results of the FirstModel trained on the 12 classes subset



(a) Accuracy of the FirstModel trained on the full data set.

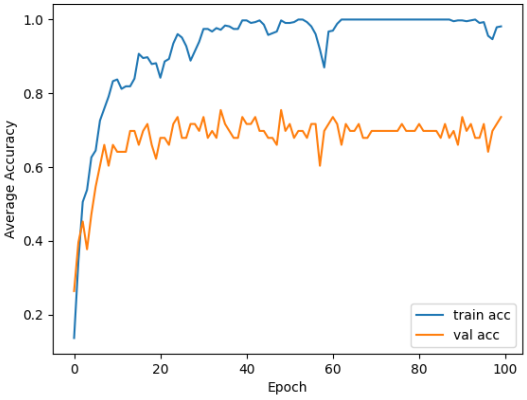


(b) Loss of the FirstModel trained on the full data set.

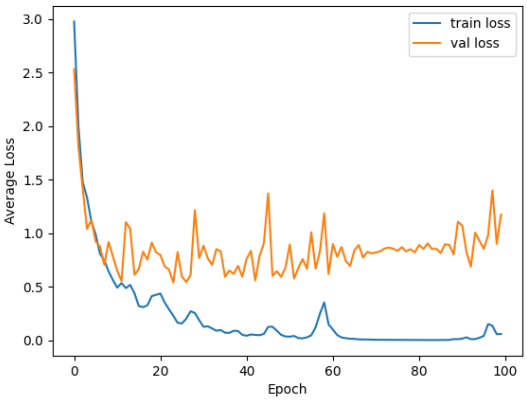


(c) Confusion Matrix of the FirstModel trained on the full data set.

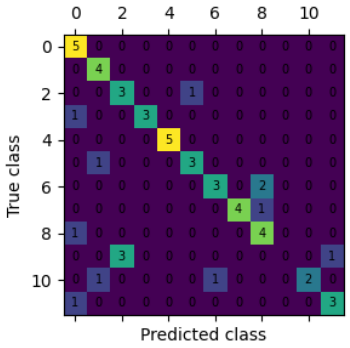
Figure 4.2: Results of the FirstModel trained on the full data set



(a) Accuracy of the SecondModel trained on the 12 class subset.

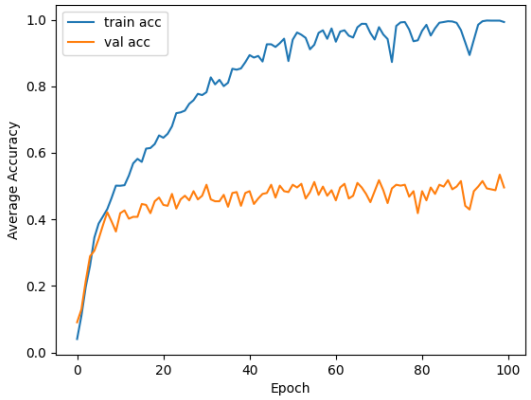


(b) Loss of the SecondModel trained on the 12 class subset.

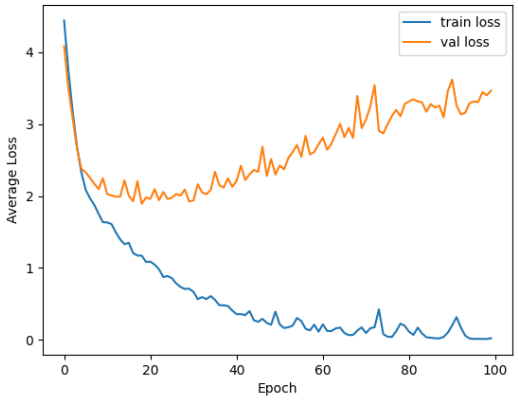


(c) Confusion Matrix of the SecondModel trained on the 12 class subset.

Figure 4.3: Results of the SecondModel trained on the 12 class subset



(a) Accuracy of the SecondModel trained on the full data set.



(b) Loss of the SecondModel trained on the full data set.

Figure 4.4: Results of the SecondModel trained on the full data set

5 Discussion

5.1 *Experience*

One of the main challenges that we faced in this project was the lack of prior knowledge and experience with building any Neural Network architectures. We had to start from scratch and learn the fundamental concepts and techniques of creating a Neural Network model that can perform a specific task. However, we did not let this obstacle discourage us. Instead, we adopted a gradual and collaborative approach, where we divided the work into manageable steps and helped each other along the way. We also consulted various online resources and tutorials to guide us through the process of designing, training, testing and evaluating our model. Most notably, we used the tutorial published by Khan [5] on their machine learning blog. As a result, we were able to build a functional Neural Network model that achieved a reasonable level of accuracy on our chosen dataset. However, we are also aware that our model is far from perfect and that there are many aspects that can be enhanced or modified to improve its performance. For example, we could experiment with different types of layers, activation functions, optimizers, loss functions, hyperparameters, regularization techniques, data augmentation methods and so on. We could also try to apply our model to different datasets or tasks and see how it generalizes to new situations. But, despite these limitations and challenges, we are proud of what we have accomplished in this project and we have gained valuable insights and skills that will help us in our future endeavors. We have learned how to approach a new problem with a Neural Network model and how to overcome some of the difficulties and uncertainties that may arise along the way.

5.2 *Activation and Optimization functions*

For example, the activation function and the optimizer function were adopted from a tutorial. We believe that there might be other functions that would be better suited for this task, but we restrained ourselves from venturing too much into the unknown in order to be able to make sure we actually understand what we are doing in the end. With that being said, we kept the functions from the tutorials we followed also because they worked rather well from our beginning tests. We then studied these functions in order to get a better grasp of what they actually do.

5.3 *Limited data*

Another challenge that we faced in our project was the quality and quantity of our dataset. We had to deal with a large number of classes, each with a relatively small number of images. This could have affected the performance of our model, as it might not have learned enough features from each class. To overcome this issue, we could have applied some data augmentation techniques, such as cropping, flipping, rotating, or adding noise to the images. However, we did not have enough time to implement and test these methods, as we were unfamiliar with methods of implementing them properly.

5.4 *Simple Architecture*

One of the limitations of our approach is the choice of architecture for our model. We used a Multi Layer Perceptron (MLP), which is a simple and widely used type of artificial neural network, but it is not very effective for image processing tasks. A Convolution Neural Network (CNN), on the other hand, is a specialized kind of neural network that can extract features from images and perform better on tasks such as classification, segmentation, and detection. However, we decided to start with an MLP because we wanted to build a baseline model and then gradually increase the difficulty and complexity of our project. Unfortunately, due to time and complexity constraints, we were not able to implement a CNN or compare its performance with our MLP model. This is a major drawback for our project and it limits the accuracy and generalization of our results.

5.5 *Data similarity*

Previously we mentioned that we selected a subset of our whole dataset containing 12 classes. We noticed that two of these classes are very similar to each other, that is that the gems look similar. When inspecting a confusion matrix generated after training and validating a model, we can observe that one of these classes overshadows the other. Figure 5.1 shows the four images from the test set for the two similar classes.

5.6 *Early stopping*

The model runs for the full amount of epochs without accounting for overfitting, thus the model has a higher chance of ending up overfitted. We were aware that overfitting could be a problem, but in our limited testing, overfitting did not appear to impact the accuracy of the model. In Figures 4.2b, 4.3b, and 4.4b it can be observed that the validation loss starts to increase after 20 epochs, a clear sign of overfitting. However, the training accuracy does not drop, as it either remains stable or grows.

5.7 *Computational Power*

In this section, we revisit our previous discussion on the two models we developed for our task. Both models follow the same pipeline, but they differ in their network structure and complexity. The first model is a simpler and lighter one, which enables us to train and modify it quickly, while the second model is a more elaborate and heavy one, which requires more computational resources and time to run and evaluate. We conducted a comparative analysis of the two models in the results section, where we found that increasing the model

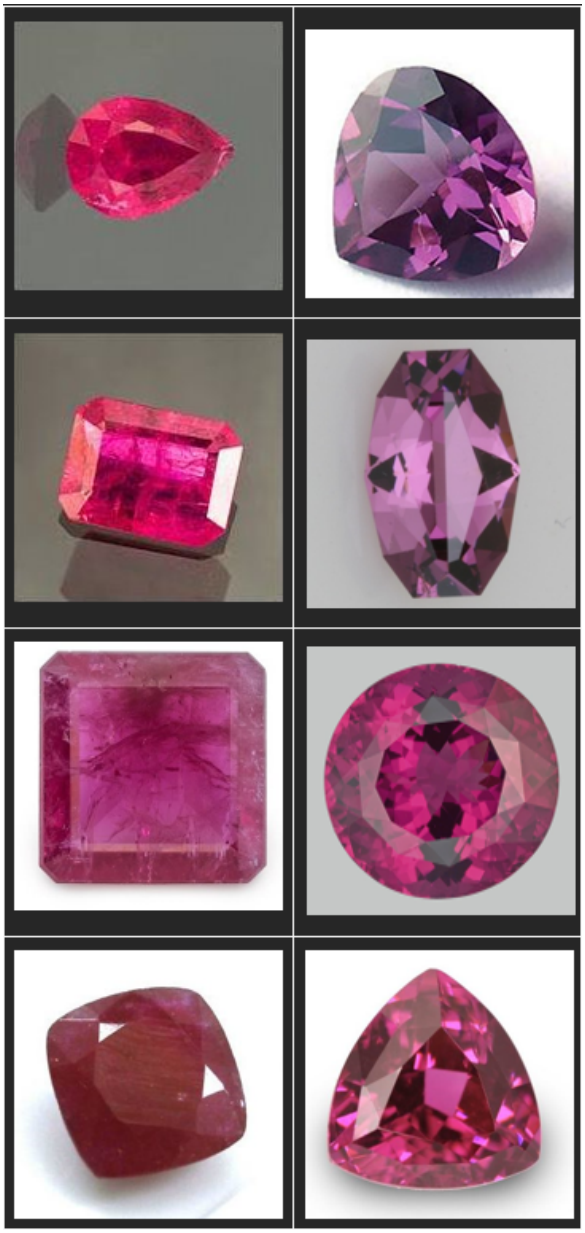


Figure 5.1: Two different classes of gems that look similar. Left-Bixbite, Right-Rhodolite

complexity does not necessarily lead to better performance. In fact, the accuracy of the model does not improve significantly as we add more layers and neurons per layer. This finding suggests that the main limitation is not the model size, but the architecture design. Therefore, we conclude that we need to explore alternative architectures that can better capture the patterns and relationships in the data. The paper written by Lippmann [6] confirms our claim: a two-hidden-layer MLP is sufficient for generating classification regions with arbitrary shapes.

5.8 *Multi Dimensional MLPs and future reaserch*

One of the recent developments in the field of deep learning is the use of multiple multilayer perceptrons (MLPs) to create powerful and efficient models. In a paper by Tian et al. [7], the authors propose a novel and interesting architecture that consists of several MLPs that are connected by attention mechanisms. This architecture is capable of obtaining far better accuracy with a smaller quantity of data than conventional convolutional neural networks (CNNs) or transformers. The authors demonstrate the effectiveness of their approach on various computer vision tasks, such as image classification, object detection, and semantic segmentation. The paper presents a significant contribution to the literature and opens up new possibilities for further research. Since it is a relatively new concept, there are many open questions and challenges that need to be addressed, such as how to optimize the number and size of the MLPs, how to design the attention modules, and how to generalize the architecture to other domains and modalities.

6 Bibliography

[1] D. Chemkaeva. Gemstones images, Mar 2020. URL `https://www.kaggle.com/datasets/lsind18/gemstones-images`.

[2] A. Mao, M. Mohri, and Y. Zhong. Cross-Entropy Loss Functions: Theoretical Analysis and Applications. In *Proceedings of the 40 th International Conference on Machine Learning*, 2023. doi:https://doi.org/10.48550/arXiv.2304.07288.

[3] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014. doi:https://doi.org/10.48550/arXiv.1412.6980.

[4] J. Brownlee. Gentle introduction to the adam optimization algorithm for deep learning, Jan 2021. URL `https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/`.

[5] M.A.I. Khan. Building an Image Classifier with a Single-Layer Neural Network in PyTorch, 4 2023. URL `https://machinelearningmastery.com/building-an-image-classifier-with-a-single-layer-neural-network-in-pytorch/`.

[6] R. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4(2): 4–22, 1987. doi:10.1109/MASSP.1987.1165576.

[7] L. Tian, B. Chongyang, and W. Chaojie. MDMLP: Image Classification from Scratch on Small Datasets with MLP. 2022. doi:https://doi.org/10.48550/arXiv.2205.14477.

A Appendix

In the writing of the present article, Large Language Models (LLMs) have been used. Their uses were mostly relegated to:

- Spellchecking our writing
- Rewriting certain paragraphs / sections in a more academically appropriate way
- Explaining the basic details of certain concepts and providing good resources for understanding them at an appropriate level.