# Deep Learning 2023 - Project I report

Bartłomiej Sobieski, Władysław Olejnik

March 2023

## 1 Introduction

Project I of the *Deep Learning* course is concerned with the application of convolutional neural networks (CNNs) to the image classification task on the *CIFAR-10*[5] dataset. In recent years, this type of neural networks has achieved state-of-the-art results in many complex tasks ranging from general computer vision problems (beginning with the famous *AlexNet* [6]) through achieving super-human performance in chess and its variants [8], proving to this day it's general applicability and high performance in comparison to e.g. classical dense neural networks. Recently, a shift in the domain of computer vision based on deep neural networks has been observed due to the introduction of a *Transformer*-based [10] architecture (initially introduced in the context of natural language processing) called *Vision Transformer* (*ViT*) [4]. This family of neural networks outperforms convolutional neural networks under conditions of very large number of parameters and data. This transition, from dense neural networks to CNNs to visual transformers, is the motivation behind the experiments that we've conducted.

## 2 Experimental setup

As the experience level of our team's members ranged from some basic knowledge about deep learning methodologies to having practical experience and research interest in this topic, our main objective was to strike a balance between implementing the newest solutions and analyzing the impact of various basic building blocks of this domain. Therefore, our experiments present a spectrum of possible solutions to the problem of multiclass classification on the *CIFAR-10* dataset and deeply analyze how their hyperparameters influence the final result. Because of this fundamentally educational approach, we kept the number of parameters of each of our models at a level that allowed us to carry out these experiments in a reasonable amount of time (around two weeks) in light of the available computational resources.

We divided our work into 4 experiments. Experiment 1 analyzes the performance of simple multi-layer perceptrons of two different sizes that operate

on flattened images and was conducted to provide a baseline for more sophisticated methods. Experiment 2 is the core of this project and aims at analyzing how a simple convolutional neural network architecture can exceed the baseline performance by not only using the convolutional layers but also paying close attention to training- and regularization-related hyperparameters and different types of augmentation techniques. Experiment 3 analyzes our implementation of a small *Vision Transformer* as a different approach to the multiclass classification problem. Experiment 4 is focused on fine-tuning well-known pre-trained architectures on the *CIFAR-10* dataset to provide a practical upper bound for the performance of our models.

Throughout the work on this project, we've used a fixed training procedure for our models. At first, we divided the training part of the *Kaggle CIFAR-10* dataset into smaller training and validation parts (in a $4:1$ ratio) to validate the performance of our models during training by calculating validation accuracy after each epoch. Each model was run for a minimum of 30 epochs after which the early stopping mechanism was applied - if a model failed to exceed its best validation performance within a window of 4 epochs, its training was terminated. We found that this window width was able to accurately capture the point at which model performance plateaued. Even if a gain in performance was still possible, it was quite evident that it would not be of the order of more than $1-2\%$ of accuracy. This approach allowed us to analyze the impact of many different hyperparameter configurations effectively and with the limited amount of computational resources at our disposal.

In terms of other components that were fixed throughout the whole process, we've used the crossentropy loss function and the *Adam*[3] optimizer for all our models. Each model was trained on 3 fixed random seeds $(0, 1, 2)$. Our implementation is based on *PyTorch*[7] and we provide the source code on our *GitHub* repository.

## 2.1 Experiment I

In this experiment, we evaluate the performance of two multi-layer perceptron architectures of different sizes on flattened images scaled to $[0, 1]$ range. More specifically, the smaller architecture $(S)$ consists of 3 hidden layers of sizes $(1500, 500, 100)$, whereas the large $(L)$ one consists of 8 hidden layers of sizes $(2048, 1024, 512, 256, 128, 64, 32, 16)$. Each layer is followed by a *ReLU* activation function and dropout (0.1) is applied after the last hidden layer. In training, we've used batches of size 128, learning rate equal to 0.001 and a small *l2* penalty (0.01). Figure 1 shows the training and validation loss curves for both architectures averaged over the random seeds with the shaded area representing the standard deviation. Clearly, the smaller architecture was able to capture some predictive power from the images, whereas the larger one didn't manage to reduce any of the losses. Interestingly, the smaller architecture seemed to keep the training and validation loss at almost the same level which suggests that it didn't overfit to the training set. It also achieved a satisfactory value of validation accuracy, which can be observed in Figure 2, of around 40%. This

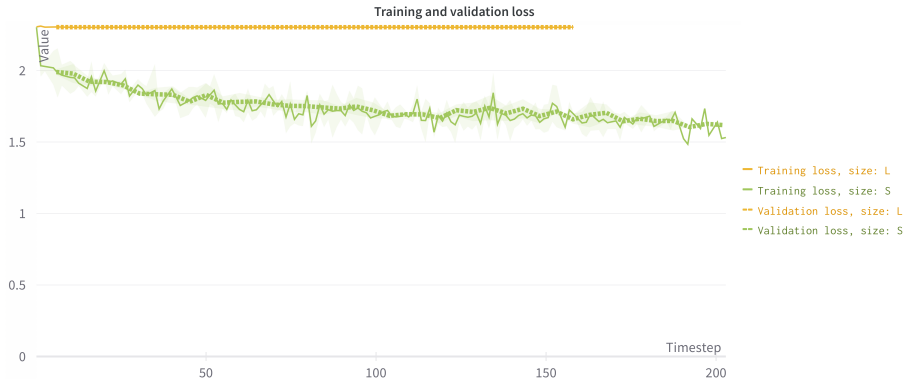


Figure 1: Training and validation loss depending on network size in experiment one

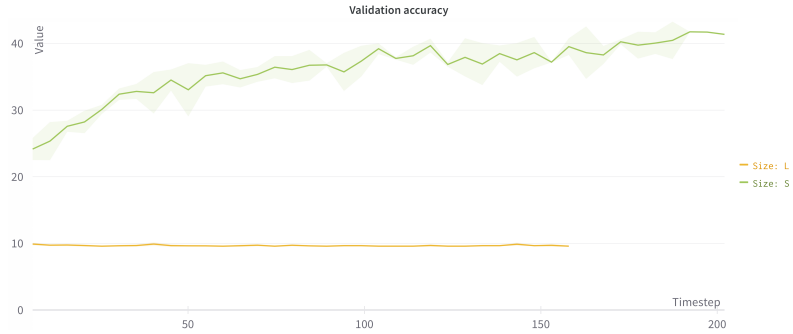


Figure 2: Validation accuracy depending on network size in experiment one

score surely poses a small surprise as this network type doesn't inherently use any spatial information from a 2-dimensional image. On the other hand, both Figures show that the larger architecture has not made any progress throughout the training process. This is certainly due to the excessive number of parameters that the optimization process tries to fit.

## 2.2 Experiment II

In this experiment, our aim was to evaluate the performance of a typical convolutional neural network. Our architecture consisted of 3 convolutional layers with increasing numbers of channels $(32, 64, 128)$ followed by a max pooling layer and 2-dimensional dropout. Next, the obtained features were flattened and fed to a sequence of linear layers of sizes $(512, 40, 10)$ to obtain logits for each class. Each layer was followed by a $ReLU$ activation function.

The first part of the experiment focused on analyzing the impact of different

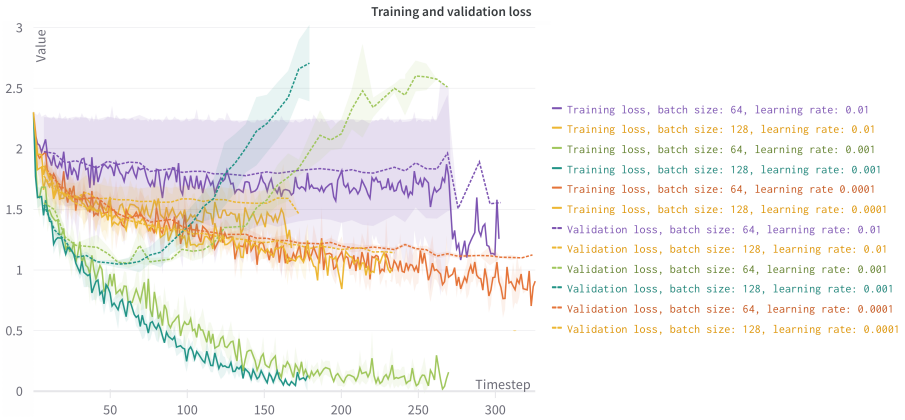


Figure 3: Training and validation loss depending on batch size and learning rate in experiment two

values of hyperparameters related to the training process: batch size (64, 128) and learning rate (0.01, 0.001 and 0.0001). Choosing appropriate values of those should lead to a more stable learning process with greater potential for generalization. Indeed, by looking at Figure 3 we can clearly observe that their impact on the training process is very high. A small batch size (64) and high learning rate (0.01) made it almost impossible to decrease both the training and validation loss, and made the learning process very unstable (high standard deviation). Keeping the same value of the learning rate and increasing the batch size to 128 allowed the optimization process to gradually decrease both losses while keeping them close together (thus not overfitting). Decreasing the learning rate for both batch sizes lead to a substantial progress in reducing the training loss. However the network quickly started to overfit, as evidenced by the $U$-shaped validation loss curves. Decreasing the learning rate further made the learning process stable and with almost no overfitting to the training set. Looking at validation accuracy curves in Figure 4 once again proves that a small batch size and high learning rate wasn't an effective combination. The loss curves are also well reflected in this Figure, where the overfitting configuration made great progress at the beginning but quickly plateaued and the more stable one was gradually improving throughout the whole training process to finally achieve accuracy of around 63%.

The second part of the experiment focused on analyzing the impact of different values of hyperparameters related to regularization: *l2* penalty (0, 0.01) and dropout (0, 0.5). We've evaluated all configurations from the previous part with these new hyperparameters, but only present the results for batch size 128 and learning rate 0.001 for clarity, as these were the best in general. Looking at Figure 5, we see that, as indicated before, applying no regularization leads to decreasing the losses quickly but also to severe overfitting. This can be mitigated

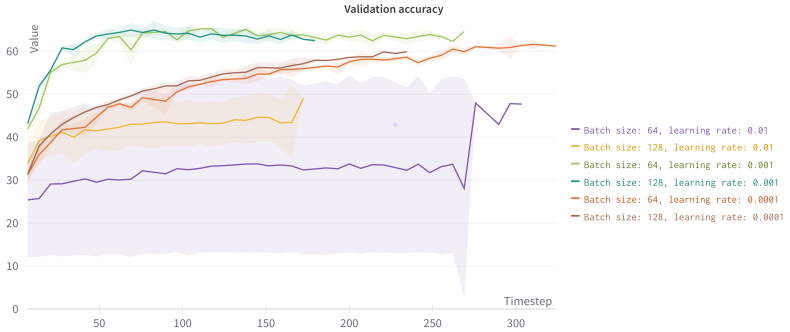


Figure 4: Validation accuracy depending on batch size and learning rate in experiment two

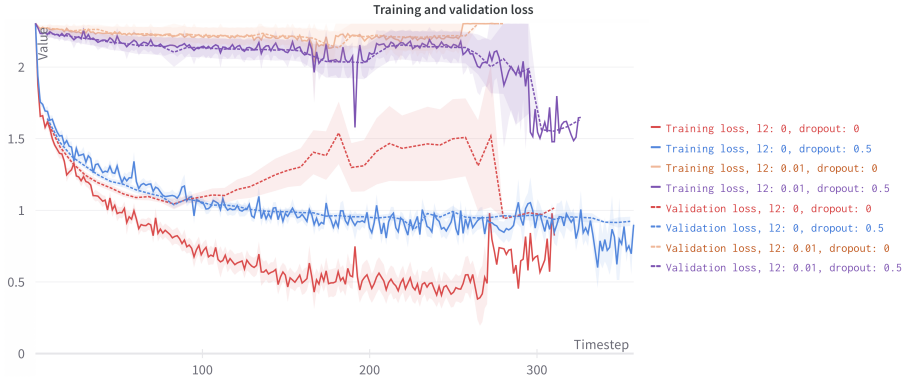


Figure 5: Training and validation loss depending on l2 penalty and dropout in experiment two

by applying dropout which helps with both problems. However, applying any non-zero *l2* penalty in our case lead to a substantial drop in performance and inability to decrease the loss almost at all. This can be also observed by looking at validation accuracy curves in Figure 6. They show that using *l2* penalty is, in this case, very ineffective. Using dropout alone resulted in improving the final validation accuracy to around 67%.

The last part of the experiment focused on analyzing the impact of applying different augmentation techniques. To this point, we've presented the results with no augmentation applied. To further improve performance, we decided to additionally evaluate each hyperparameter configuration with basic augmentation techniques (random horizontal flip, color jitter and random rotation by an angle up to 10 degrees) and one advanced augmentation technique (random erasing). Therefore, we either applied no augmentation (0), only the basic ones (3) or basic techniques combined with the advanced one (4). Once again, we

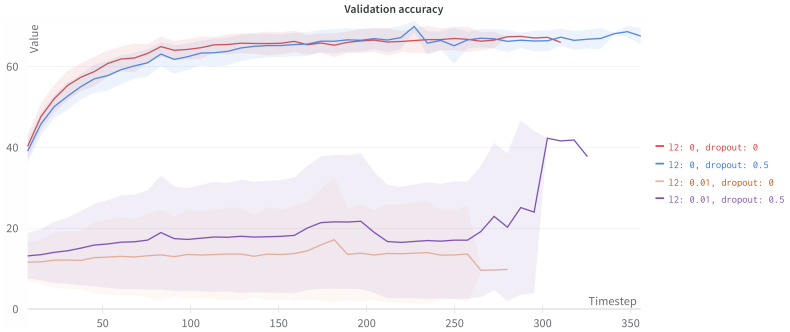


Figure 6: Validation accuracy depending on l2 penalty and dropout in experiment two

only show the best performing configuration for clarity (batch size 128, learning rate 0.001, dropout 0.5, no *l2* penalty) as the conclusions for the other ones were very similar. Figure 7 shows that the more augmentation techniques we've used, the harder the problem has become for the network to solve, i.e. the training and validation losses were decreasing but to higher values when augmentation was applied. This is expected as augmenting the data should force the network to learn in a different, more general way by dealing with more diverse data. Figure 8 shows that, in our case, augmenting the data to some degree is beneficial, but harms the final validation accuracy when used too extensively. More specifically, basic techniques lead to improving the validation accuracy to around 71% but adding random erasing decreases the final performance to around 65%. This might be due to the fact that the *CIFAR-10* dataset consists of images of very low resolution ($32 \times 32$) and applying very complicated augmentation might have too much of an impact on the actual information contained within the image (even the original images are sometimes very hard to classify by a human).

## 2.3 Experiment III

In this experiment, we evaluate the performance of our implementation of *Vision Transformer* from [4]. Our architecture consists of 6 transformer encoder layers (i.e. the sequence *Multi-head attention -¿ Feed forward* with residual connections and layer normalization in between repeated 6 times) with the internal embedding dimension equal to 30 and patches of size $4 \times 4$. Due to computational constraints, we focus on training related hyperparameters only. Figure 9 shows that their impact on the training and validation loss curves is marginal - in every case the network doesn't seem to overfit and gradually decreases the losses in a stable manner. Figure 10 proves this phenomenon by showing validation accuracy curves that don't differ in any significant way - each configuration finishes its training process with around 51% accuracy.

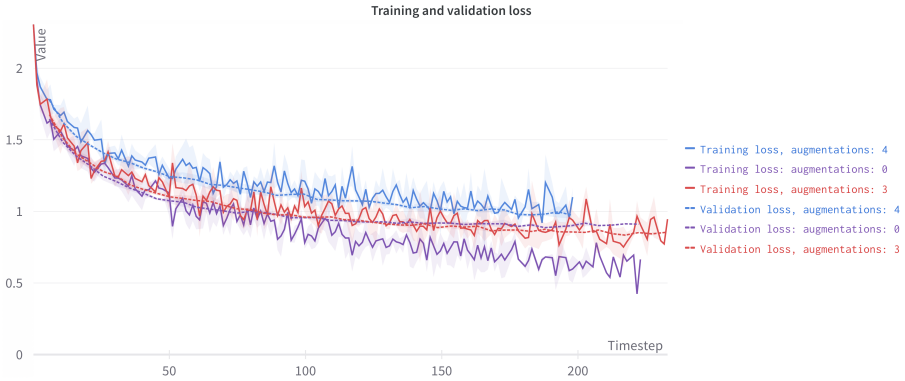


Figure 7: Training and validation loss depending on augmentations in experiment two

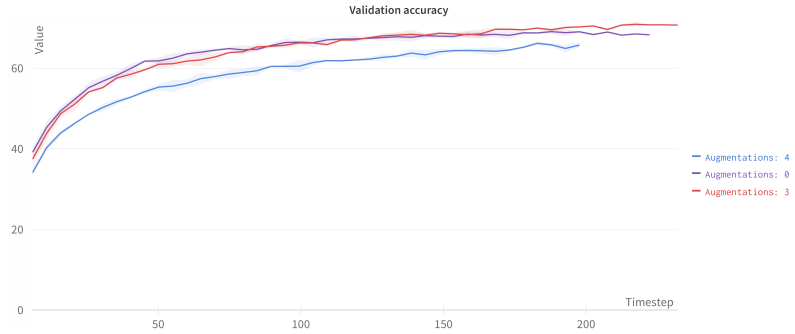


Figure 8: Validation accuracy depending on augmentations in experiment two

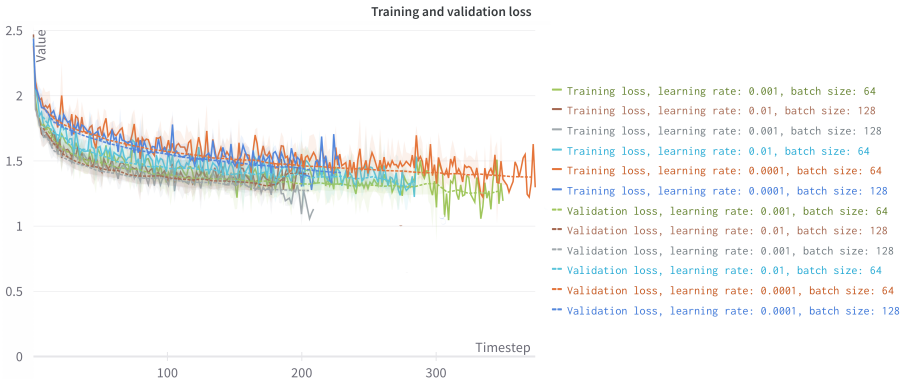


Figure 9: Training and validation loss depending on batch size and learning rate in experiment three

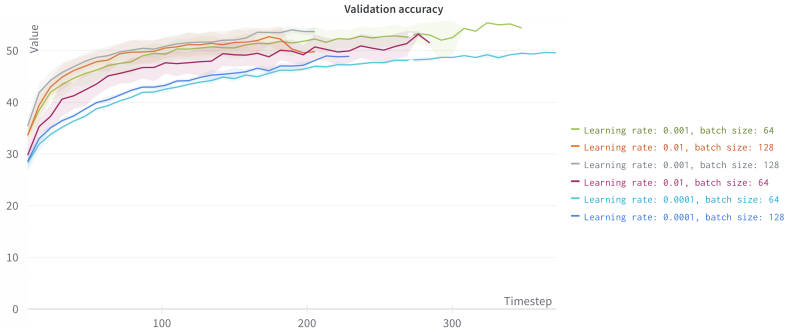


Figure 10: Validation accuracy depending on batch size and learning rate in experiment three

## 2.4 Experiment IV

In this experiment, we focus on fine-tuning well-known pre-trained architectures to our problem of multiclass classification on the *CIFAR-10* dataset. More specifically, we've used the following models: *MobileNet v3 S*[2](2.5$M$ parameters), *EfficientNet v2 S*[9](21.5M parameters) and *ResNet18*[1](11.7$M$ parameters) - all pretrained on *ImageNet* dataset with 1000 classes. We fine-tune them by replacing their last classification layer with randomly initialized linear layer consisting of 10 neurons (since we have 10 classes in our problem). Figure 11 shows training and validation loss curves for each model averaged over 3 seeds. These curves should gradually approach lower values as before - however, we observe a very surprising behaviour of *ResNet18* on the validation set. This network seems to struggle greatly with stabilizing its validation loss while its training loss seems to behave as expected. We suspected that this was due to an error in our code, but a deeper inspection reassured us that this was not the case. In addition, the stable behaviour of the training loss indicates that the learning process was conducted in an appropriate manner. Fortunately, Figure 12 shows that this behaviour wasn't reflected on the general progress in validation accuracy. There, we observe that *ResNet18* achieved around 77% accuracy on the validation set. *EfficientNet v2 S* finished with around 73% accuracy which is very close to our best result from experiment 2. Surprisingly, the best validation accuracy of around 87% was noted for *MobileNet v3 S* - the smallest of these 3 networks. We suspect that this is due to a very low resolution of the images from the *CIFAR-10* dataset. The bigger the network, the more sophisticated the learned features become and at some point they might be orthogonal to what should be extracted from images of very low resolution. Also, we suspect that the fact that *EfficientNet v2 S* performs the worst might be also due to a different input size used by this network. For each model, we use their original preprocessing procedure and, contrary to two other models that resize their input to $256 \times 256$, *EfficientNet v2 S* resizes its input to $384 \times 384$. This interpolation might lead to confusing the network as it results in non-realistic

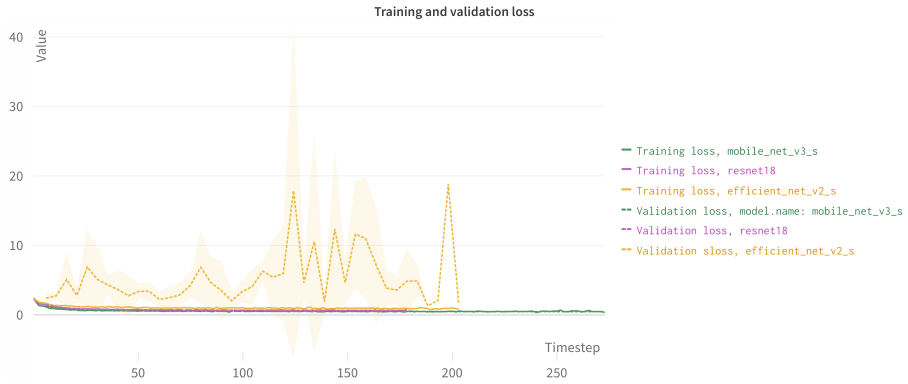


Figure 11: Training and validation loss depending on architecture in experiment four

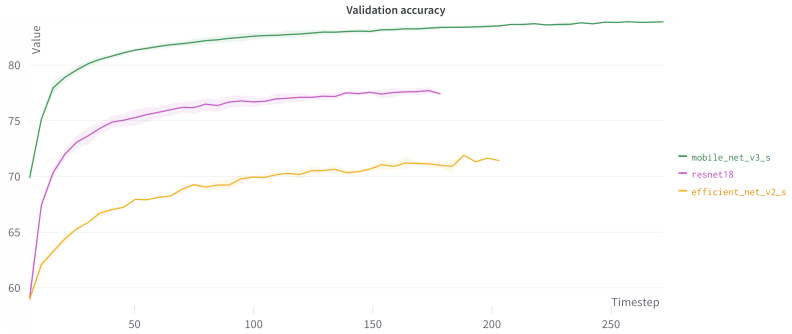


Figure 12: Validation accuracy depending on architecture in experiment four

images - it seems unfeasible to perform such interpolation and preserve the same kind of information as in the original image.

# 3 Results

We evaluated the best models from each experiment on the kaggle challenge test set, which contained 300,000 images. To do this, we used an additional script that loaded the saved models and performed the classification. We submitted the results to the platform. The accuracy we obtained was slightly different from the validation stage.

- Experiment 1 - 42.42%
- Experiment 2 - 72.52%
- Experiment 3 - 54.08%

- Experiment 4 - 83.16%

The results suggest that the choice of neural network architecture is critical in achieving high accuracy on image classification tasks, and different architectures have different strengths and weaknesses.

The MLP architecture achieved the lowest accuracy of 42%, which is not surprising as MLPs are not well-suited for image classification tasks since they do not inherently use any spatial information from a 2-dimensional image.

The CNN architecture achieved a higher accuracy of 72.52%, which is a significant improvement over the MLP architecture. CNNs are well-suited for image classification tasks as they can learn local spatial features using convolutional layers and pooling layers. The VIT architecture achieved an accuracy of 54.08%, which is lower than the CNN architecture but higher than the MLP architecture. VITs are a recent advancement in computer vision and have shown promising results in image classification tasks, but they may require more training data and computational resources than CNNs.

Finally, the Mobile Net architecture achieved the highest accuracy of 83.16%, which is a significant improvement over all the other architectures. Mobile Net is a lightweight and efficient architecture that is designed for mobile devices and has been shown to perform well on image classification tasks while requiring fewer computational resources.

# 4 Conclusions and future work

In this work, we trained models with a small number of parameters due to the educational approach, computational and time constraints. We explored the effect of neural network architecture on image classification tasks and found that MLP architectures are not effective for this type of task, while CNN and Mobile Net architectures can achieve high accuracy with less computation. We also applied regularization techniques such as dropout and L2 regularization to avoid overfitting and improve the generalization ability of neural networks. Furthermore, we observed that using different seeds can have a significant impact on the performance of the model with the same configuration. We also tested different batch sizes, image transformations, and learning rates.

The results of the experiments reveal the crucial role of hyperparameter tuning in finding optimal architectures. Some hyperparameters have a large impact on the performance, while others require a combination of adjustments to make a difference. Moreover, the effectiveness of some hyperparameters depends on the values of others, creating complex interactions. Therefore, it is essential to have a good intuition and understanding of how different hyperparameters affect different architectures, especially when facing computational limitations.

Future work could focus on extending our approach to larger scale and more advanced architectures. In addition, a deeper inspection of training small *ViT* models could be carried out and a more detailed analysis of a surprising behaviour of *ResNet18* from experiment 4 would be desirable.

# References


[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[2] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[4] Alexander Kolesnikov, Alexey Dosovitskiy, Dirk Weissenborn, Georg Heigold, Jakob Uszkoreit, Lucas Beyer, Matthias Minderer, Mostafa Dehghani, Neil Houlsby, Sylvain Gelly, Thomas Unterthiner, and Xiaohua Zhai. An image is worth 16x16 words: Transformers for image recognition at scale. 2021.

[5] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[8] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

[9] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training, 2021.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.