Deep Learning - Convolutional neural networks

Michal Gromadzki

March 2024

1 Description

Image classification is a fundamental task in computer vision where an algorithm analyzes an image and assigns it to one or more predefined categories. The goal is to automatically identify the objects, scenes, or patterns present in the image. This process involves extracting relevant features from the image, such as shapes, textures, and colours, and using them to make predictions. Deep learning techniques, particularly convolutional neural networks[8], have revolutionized image classification by achieving remarkable accuracy across various domains.

This project aims to explore the performance of three deep learning models — Convolutional Neural Network (CNN), MobileNet[6], and Vision Transformer (ViT)[4]—on the CINIC-10 dataset[2]. Moreover, different hyperparameters combinations, along with various data augmentation techniques, are to be tested. CINIC-10 is a widely used benchmark dataset in computer vision tasks, consisting of 270,000 32x32 color images across 10 classes.

The CNN architecture serves as a traditional baseline model for comparison, known for its effectiveness in image classification tasks. MobileNet, on the other hand, represents a lightweight convolutional neural network designed specifically for mobile applications, optimized for efficiency without compromising much on accuracy. Lastly, Vision Transformer (ViT) represents a paradigm shift in computer vision with its transformer-based architecture, which has demonstrated state-of-the-art performance in various tasks by leveraging self-attention mechanisms[11].

2 Instruction

In this project, three main Python[10] files were implemented: cnn.py, mobilenet.py, and vit.py, each containing the code for training and evaluating Convolutional Neural Network (CNN), MobileNet, and Vision Transformer (ViT) models, respectively. These models were trained and tested on the CINIC-10 dataset. All trained models are saved to a folder named models/. All helper functions such as data-loading, data augmentation and preprocessing were implemented in utils.py. Additionally, the results of the experiments, including metrics such as accuracy, loss and used regularization techniques are recorded in a file named results.csv. Finally, a comprehensive analysis of the performance of each model was conducted using Jupyter Notebook, with the analysis presented in a file named analysis.ipynb. All neural network related functionality was implemented in Tensorflow[1].

To reproduce the results, it is necessary to execute the three main Python scripts: cnn.py, mobilenet.py, and vit.py. Additionally, it's important to note that a fixed seed was used to ensure reproducibility across experiments. Lastly, it's worth mentioning that running these scripts may take some time due to the training and evaluation processes involved.

3 Theoretical introduction

3.1 CNNs

Image classification is a fundamental task in the field of computer vision, with numerous applications ranging from medical diagnosis to autonomous driving. It involves the categorization of images into predefined classes or labels

based on their visual content. Over the past few decades, significant advancements in machine learning, particularly in the realm of neural networks, have revolutionized image classification techniques.

One of the most powerful approaches to image classification is through the utilization of convolutional neural networks. CNNs are inspired by the biological visual cortex and are designed to automatically and adaptively learn spatial hierarchies of features from input images. They have demonstrated exceptional performance in various image recognition tasks, surpassing traditional machine learning algorithms and even human-level accuracy in certain domains. At the core of CNNs are convolutional layers, which apply filters to input images to extract local patterns and features. The convolution operation involves sliding a filter (kernel) over the input image and computing the element-wise multiplication between the filter weights and the corresponding pixel values in the receptive field. The resulting products are summed to produce a single value, which represents the activation at a specific spatial location. This process is repeated across the entire input image, generating a feature map that captures relevant patterns and structure. Figure 2 presents a simple example of a convolution.

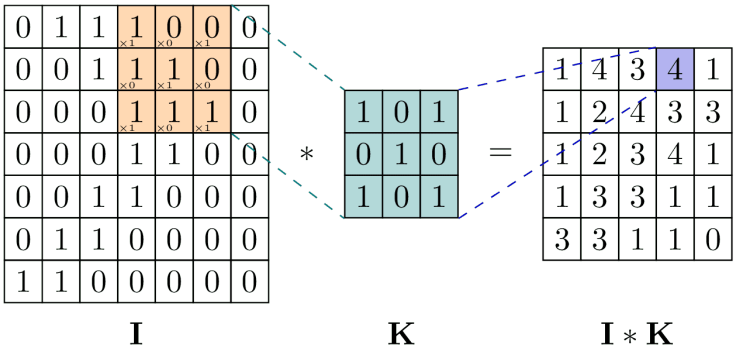

Figure 1: Example of a convolution. Source: https://tikz.net/conv2d/

These filters are learned during the training process, enabling the network to automatically discover relevant features such as edges, textures, and shapes. Pooling layers then reduce the spatial dimensions of the feature maps generated by convolutional layers, while retaining the most important information. Following the convolutional and pooling layers, fully connected layers are employed to perform classification based on the extracted features. These layers combine the learned features from earlier layers to make predictions about the input image's class labels. Figure 2 presents a sample architecture of CNN used for image classification.

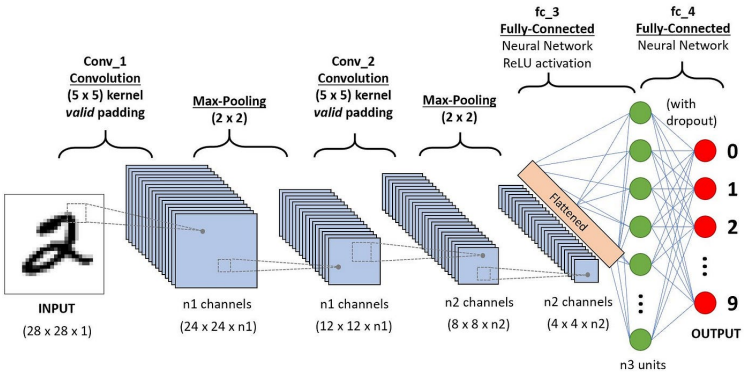

Figure 2: Sample architecture of CNN used for image classification. Source: https://saturncloud.io/

One of the key advantages of convolutional layers is their ability to achieve spatial invariance. Spatial invariance means that the network can identify the same object, feature, or pattern regardless of where it appears in the image. This feature is one of the key reasons why transfer learning is such a prominent technique.

3.2 Transfer learning

Transfer learning[5] in image classification is a powerful technique that leverages pre-trained models to solve similar tasks with minimal data. At its core, transfer learning involves taking knowledge gained from solving one problem and applying it to a different but related problem. In the context of image classification, this means utilizing a neural network model that has been trained on a large dataset, such as ImageNet, and fine-tuning it for a specific classification task with a smaller dataset. By fine-tuning a pre-trained model on a new dataset, the model adapts its learned representations to better suit the specific characteristics of the new task. During this fine-tuning process, the model adjusts its parameters to focus on the features that are most relevant for the new classification task, while still retaining the knowledge gained from the original training data.

3.3 ViTs

Vision Transformers (ViTs) are a cutting-edge approach in the realm of computer vision, revolutionizing the way machines understand and process visual data. Unlike traditional convolutional neural networks which have long dominated this field, ViTs apply the transformer architecture, originally devised for natural language processing tasks[11], to images.

At the heart of ViTs lies the transformer model, renowned for its success in processing sequential data by capturing long-range dependencies. ViTs leverage this capability by splitting an input image into patches, treating each patch as a token, and feeding them through a transformer network. This enables ViTs to process images as sequences of tokens, allowing for global context understanding without relying on spatial hierarchies present in CNNs. Figure 3 presents the ViT architecture.

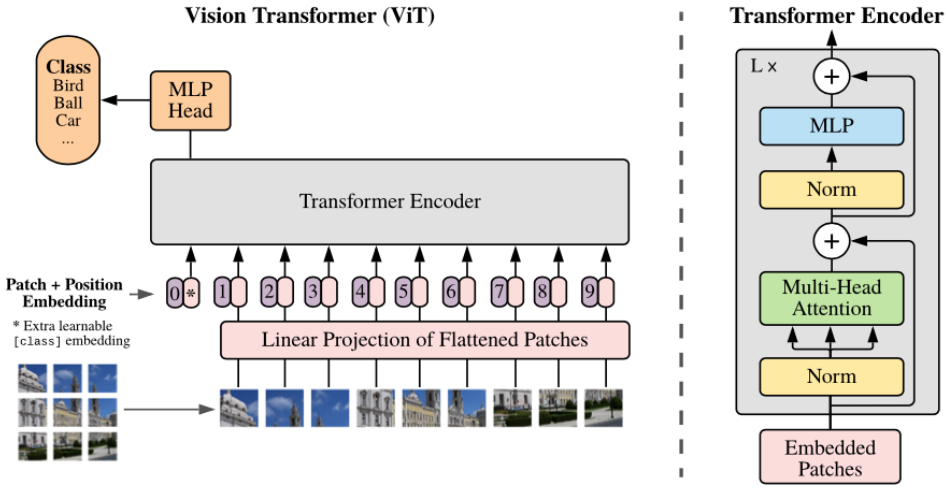

Figure 3: ViT architecture. Source: https://github.com/google-research/vision_transformer

One of the key advantages of ViTs is their scalability and generalization. Unlike CNNs, which require intricate architectural modifications to accommodate different input sizes, ViTs can seamlessly handle images of varying resolutions through their patch-based approach. This makes ViTs more flexible and efficient, particularly when dealing with large-scale datasets and diverse image sizes. Moreover, ViTs have demonstrated impressive performance across

various computer vision tasks, including image classification, object detection, and segmentation. Despite their relative novelty, ViTs have already achieved competitive results, often surpassing traditional CNN-based methods on benchmark datasets.

4 Conducted experiments

The experiments were split into 3 main sections based on the trained model.

4.1 Models

Three distinct models were analyzed: CNN serves as the baseline, MobileNetV3Small represents the middle ground between number of parameters and performance, and ViT-B32, a Vision Transformer, is considered overkill in terms of complexity and parameter count.

4.1.1 CNN

The first model is a convolutional neural network. It consists of four convolutional layers with increasing depth, followed by max-pooling layers to downsample feature maps. After flattening the output, two dense layers are employed as preclassifier and classifier. Final layer consists of 10 neurons, one for each class. The model utilizes ReLU activation functions in convolutional and dense layers, enhancing non-linearity, and employs padding to maintain input size. The model has 653 386 parameters.

4.1.2 MobileNetV3Small

The second model utilizes the MobileNetV3Small architecture, a compact and efficient convolutional neural network designed for tasks like image classification. It is pretrained on the ImageNet dataset. This way, the model can utilize the knowledge learned from ImageNet dataset to help better learn the dependencies in CINIC-10 dataset - transfer learning. Note that the MobileNetV3Small model is pretrained on 224x224 images, thus the first layer of the model resizes the images from 32x32 to 224x224. Finally global average pooling and dense layers are implemented as the classification head. The model has 1 089 402 parameters. Figure 4 presents architecture of the described model.

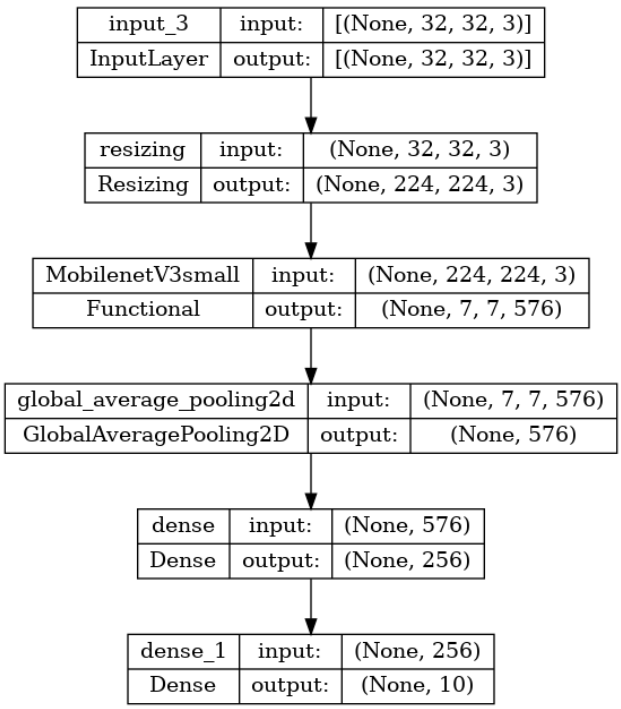

Figure 4: MobileNetV3Small model architecture

4.1.3 ViT-B32

The last model is a Vision Transformer (ViT) with a base model architecture of ViT-B32. The base model is initialized with pretrained weights. Note that this model is alo pretrained on 224x224 images, so the first layer of the model resizes the images from 32x32 to 224x224. The model architecture is extended with two additional dense layers serving the purpose of the classification head. The model has 87 455 232 parameters. Figure 5 presents architecture of the described model.

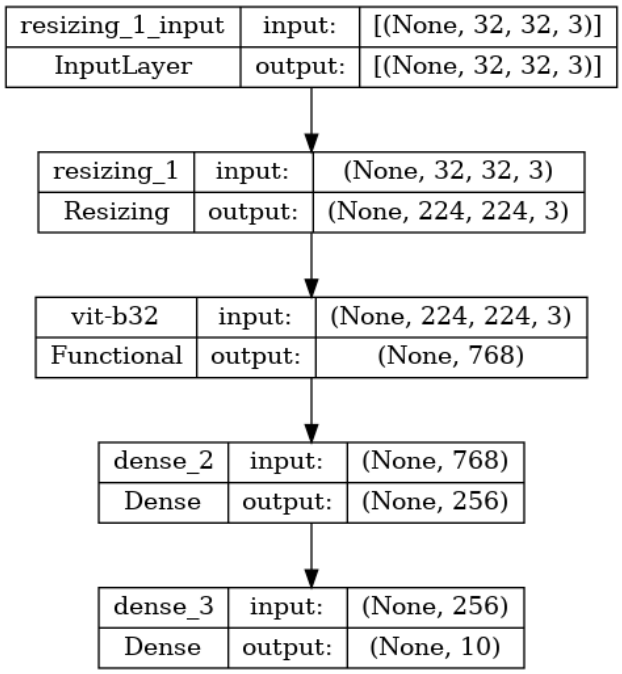

Figure 5: ViT-B32 model architecture

4.2 Data Augmentation

Two data augmentation techniques were used to facilitate the training process.

4.2.1 Type 1

The pipeline consists of three distinct augmentation techniques: random horizontal flipping, random rotation, and random brightness adjustment. The "RandomFlip" layer introduces random horizontal flips to the input images, effectively mirroring them along the vertical axis. Subsequently, the "RandomRotation" layer applies random rotation transformations to the images within a specified range, with a maximum angle of 0.3 radians. Lastly, the "RandomBrightness" layer randomly adjusts the brightness of the images, with a maximum factor of 0.2. By combining these augmentation techniques, the pipeline dynamically generates diverse variations of the input data during training.

4.2.2 Type 2

The Mixup data augmentation technique[12] is a method used to enhance training data for neural networks. It operates by blending pairs of input images and their corresponding labels. Beta distribution is sampled with a 0.2 alpha parameter, generating mixing coefficients. These coefficients determine the ratio of blending between original images and their randomly shuffled counterparts. The images are mixed based on the calculated coefficients, while the labels undergo similar mixing to ensure consistency between input and output pairs. This technique helps to regularize the model and encourage smoother decision boundaries, ultimately improving its generalization performance.

4.3 Hyper-parameters related to training process and regularization

Two different combinations of batch size and learning rates were tested:

- Learning rate - 0.001, batch size - 256
- Learning rate - 0.0001, batch size - 32

For the regularization of the training process, different combinations of L2 penalty, Dropout[9] and gradient clipping were used. The tested combinations were:

- Dropout
- L2
- L2 penalty + Dropout
- Gradient clipping + Dropout

4.4 Training process

The training process depends on the selected model. Convolutional neural networks were trained with the Adam optimizer[7], the training stopped when there was no increase in the best validation accuracy for 4 consecutive epochs. Training lasted a maximum of 100 epochs. All models have been trained 3 times to ensure better credibility of the results.

The MobileNet and ViT models underwent a two-step training process. Initially, the 'pretraining' phase involved training the models for a maximum of 20 or 30 epochs (depending on the model) or until there was no improvement in the best validation accuracy for 3 consecutive epochs. During this phase, only the classification head was trained while the base models remained frozen. This approach ensured that during the 'fine-tuning' phase gradients wouldn't explode. Training utilized the Adam optimizer with a fixed learning rate of either 0.001 or 0.0001. Models achieving the highest accuracy on the validation set were saved. All models have been pretrained once, due to long training times.

After the pretraining phase, the 'fine-tuning' phase began. During this phase, the previously trained models were loaded, and approximately the last 20% to 30% of the layers in the base models, along with the classification heads, were further trained. The learning rate was decreased by a factor of 10, and the RMSprop optimizer[3] was used to ensure a more stable training process. Training continued for an additional 20 epochs, with the best-performing models on the validation set being saved. All models have been fine-tuned once, due to long training times.

4.5 Table of all trained models

The table 1 includes all trained models.

Model	Batch size	Learning Rate	Optimizer	Regularization	DA
CNN	256	0.001	Adam	No	No
CNN	32	0.0001	Adam	No	No
CNN	256	0.001	Adam	Dropout	No
CNN	256	0.001	Adam	L2	No
CNN	256	0.001	Adam	L2 + Dropout	No
CNN	256	0.001	Adam	Gradient Clipping + Dropout	No
CNN	256	0.001	Adam	No	Type 1
CNN	32	0.0001	Adam	No	Type 1
CNN	256	0.001	Adam	Dropout	Type 1
CNN	256	0.001	Adam	L2	Type 1
CNN	256	0.001	Adam	L2 + Dropout	Type 1
CNN	256	0.001	Adam	Gradient Clipping + Dropout	Type 1
CNN	256	0.001	Adam	No	Type 2
CNN	32	0.0001	Adam	No	Type 2
CNN	256	0.001	Adam	Dropout	Type 2
CNN	256	0.001	Adam	L2	Type 2
CNN	256	0.001	Adam	L2 + Dropout	Type 2
CNN	256	0.001	Adam	Gradient Clipping + Dropout	Type 2
MobileNet_pretrained	256	0.0001	Adam	No	No
MobileNet_finetuned	256	0.0001	Adam + RMSprop	No	No
MobileNet_pretrained	32	0.00001	Adam	No	No
MobileNet_finetuned	32	0.00001	Adam + RMSprop	No	No
MobileNet_pretrained	256	0.0001	Adam	Gradient Clipping + Dropout	No
MobileNet_finetuned	256	0.0001	Adam + RMSprop	Gradient Clipping + Dropout	No
MobileNet_pretrained	256	0.0001	Adam	No	Type 1
MobileNet_finetuned	256	0.0001	Adam + RMSprop	No	Type 1
MobileNet_pretrained	32	0.00001	Adam	No	Type 1
MobileNet_finetuned	32	0.00001	Adam + RMSprop	No	Type 1
MobileNet_pretrained	256	0.0001	Adam	Gradient Clipping + Dropout	Type 1
MobileNet_finetuned	256	0.0001	Adam + RMSprop	Gradient Clipping + Dropout	Type 1
MobileNet_pretrained	256	0.0001	Adam	No	Type 2
MobileNet_finetuned	256	0.0001	Adam + RMSprop	No	Type 2
MobileNet_pretrained	32	0.00001	Adam	No	Type 2
MobileNet_finetuned	32	0.00001	Adam + RMSprop	No	Type 2
MobileNet_pretrained	256	0.0001	Adam	Gradient Clipping + Dropout	Type 2
MobileNet_finetuned	256	0.0001	Adam + RMSprop	Gradient Clipping + Dropout	Type 2
ViT_pretrained	256	0.0001	Adam	Dropout	No
ViT_finetuned	256	0.0001	Adam + RMSprop	Gradient Clipping + Dropout	No
ViT_pretrained	256	0.0001	Adam	Dropout	Type 1
ViT_finetuned	256	0.0001	Adam + RMSprop	Gradient Clipping + Dropout	Type 1
ViT_pretrained	256	0.0001	Adam	Dropout	Type 2
ViT_finetuned	256	0.0001	Adam + RMSprop	Gradient Clipping + Dropout	Type 2

Table 1: Table of all trained models

5 Results

The results were split into 3 sections, based on the model.

5.1 CNN

All models have 653 386 parameters. The training stopped when there was no increase in the best validation accuracy for 4 consecutive epochs.

Batch size	**Learning Rate**	**Optimizer**	**Regularization**	**DA**	**Loss $\pm$ std**	**Accuracy $\pm$ std**
256	0.001	Adam	No	No	1.098 ± 0.034	0.615 ± 0.006
32	0.0001	Adam	No	No	1.096 ± 0.017	0.624 ± 0.002
256	0.001	Adam	Dropout	No	$\mathbf{1.046 \pm 0.017}$	$\mathbf{0.641 \pm 0.001}$
256	0.001	Adam	L2	No	1.474 ± 0.062	0.548 ± 0.029
256	0.001	Adam	L2 + Dropout	No	1.423 ± 0.025	0.575 ± 0.011
256	0.001	Adam	Gradient Clipping + Dropout	No	1.033 ± 0.010	0.640 ± 0.003
256	0.001	Adam	No	Type 1	1.502 ± 0.06	0.480 ± 0.009
32	0.0001	Adam	No	Type 1	1.558 ± 0.028	0.476 ± 0.007
256	0.001	Adam	Dropout	Type 1	1.550 ± 0.116	0.494 ± 0.005
256	0.001	Adam	L2	Type 1	1.689 ± 0.019	0.448 ± 0.008
256	0.001	Adam	L2 + Dropout	Type 1	1.768 ± 0.030	0.414 ± 0.015
256	0.001	Adam	Gradient Clipping + Dropout	Type 1	1.491 ± 0.065	0.507 ± 0.003
256	0.001	Adam	No	Type 2	1.154 ± 0.025	0.602 ± 0.003
32	0.0001	Adam	No	Type 2	1.181 ± 0.039	0.592 ± 0.009
256	0.001	Adam	Dropout	Type 2	1.120 ± 0.009	0.613 ± 0.006
256	0.001	Adam	L2	Type 2	1.494 ± 0.047	0.523 ± 0.016
256	0.001	Adam	L2 + Dropout	Type 2	1.492 ± 0.023	0.523 ± 0.014
256	0.001	Adam	Gradient Clipping + Dropout	Type 2	1.108 ± 0.014	0.614 ± 0.003

Table 2: Metrics - CNN

5.2 MobileNetV3Small

All models have 1 089 402 parameters. The models with Adam optimizer represent the pretrained models, while the models with Adam + RMSprop optimizes represent fine-tuned models. Each stage of the training lasted for 30 epochs.

Batch size	Learning Rate	Optimizer	Regularization	DA	Loss	Accuracy
256	0.0001	Adam	No	No	0.571	0.797
256	0.0001	Adam + RMSprop	No	No	**0.460**	0.840
32	0.00001	Adam	No	No	0.571	0.798
32	0.00001	Adam + RMSprop	No	No	0.474	**0.842**
256	0.0001	Adam	Gradient Clipping + Dropout	No	0.574	0.797
256	0.0001	Adam + RMSprop	Gradient Clipping + Dropout	No	0.463	0.839
256	0.0001	Adam	No	Type 1	0.739	0.743
256	0.0001	Adam + RMSprop	No	Type 1	0.618	0.784
32	0.00001	Adam	No	Type 1	0.735	0.745
32	0.00001	Adam + RMSprop	No	Type 1	0.623	0.787
256	0.0001	Adam	Gradient Clipping + Dropout	Type 1	0.738	0.741
256	0.0001	Adam + RMSprop	Gradient Clipping + Dropout	Type 1	0.612	0.785
256	0.0001	Adam	No	Type 2	0.616	0.791
256	0.0001	Adam + RMSprop	No	Type 2	0.498	0.833
32	0.00001	Adam	No	Type 2	0.608	0.795
32	0.00001	Adam + RMSprop	No	Type 2	0.502	0.833
256	0.0001	Adam	Gradient Clipping + Dropout	Type 2	0.617	0.791
256	0.0001	Adam + RMSprop	Gradient Clipping + Dropout	Typp 2	0.500	0.833

Table 3: Metrics - MobileNetV3Small

5.3 ViT-B32

All models have $87\,455\,232$ parameters. The models with Adam optimizer represent the pretrained models, while the models with Adam + RMSprop optimizes represent fine-tuned models. Each stage of the training lasted for 20 epochs.

Batch size	Learning Rate	Optimizer	Regularization	DA	Loss	Accuracy
256	0.0001	Adam	Dropout	No	0.329	0.885
256	0.0001	Adam + RMSprop	Gradient Clipping + Dropout	No	**0.276**	**0.906**
256	0.0001	Adam	Dropout	Type 1	0.401	0.861
256	0.0001	Adam + RMSprop	Gradient Clipping + Dropout	Type 1	0.342	0.883
256	0.0001	Adam	Dropout	Type 2	0.358	0.883
256	0.0001	Adam + RMSprop	Gradient Clipping + Dropout	Type 2	0.288	0.902

Table 4: Metrics - ViT-B32

6 Analysis

CNNs perform the worst out of all tested models. ViT and MobilNet perform significantly better, showcasing the power of transfer learning. Figures 6 and 7 depict loss and accuracy of the models on the test dataset.

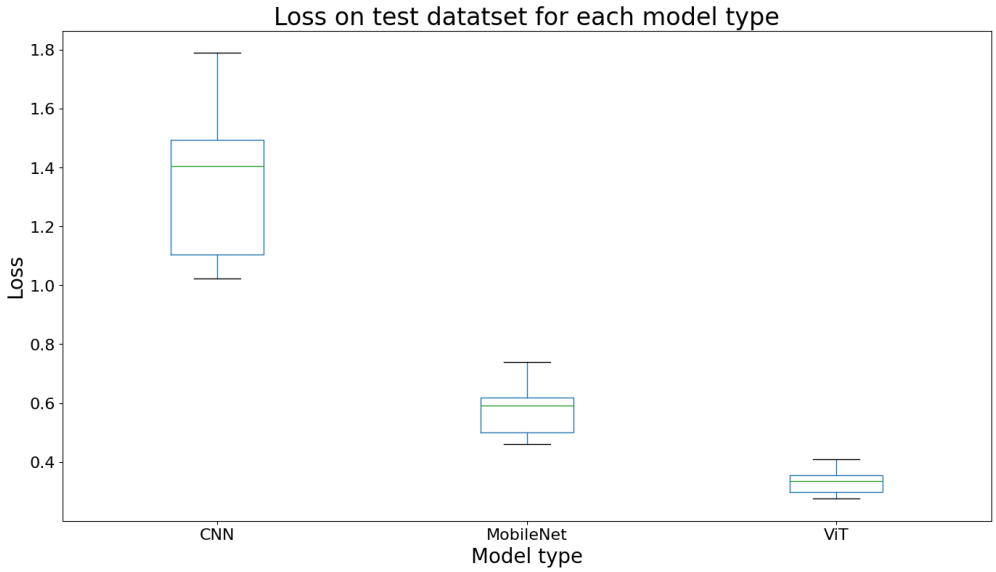

Figure 6: Loss on test dataset

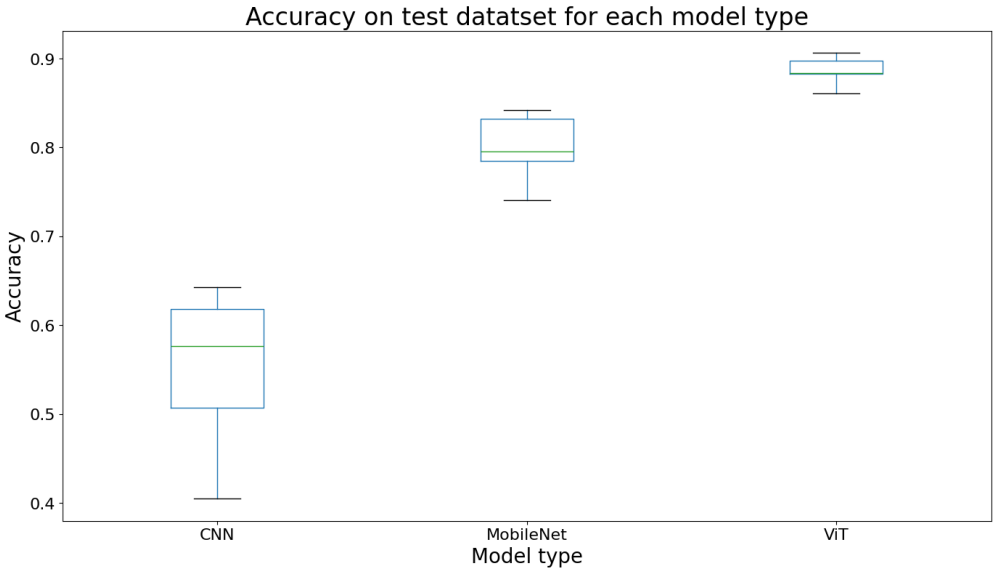

Figure 7: Accuracy on test dataset

ViT achieves the highest accuracy of 90.62%. Even though MobileNet has about 80 times less parameters, it only loses about 6% in accuracy, showing it is truly the middle ground in performance and efficiency. CNN performs worse than the previously mentioned models, achieving the highest accuracy of 64.1%. Next part of analysis investigates the

influence of used hyper-parameters, regularization techniques and data augmentation. Figure 8 presents performance of the models, grouped by used learning rates and batch sizes.

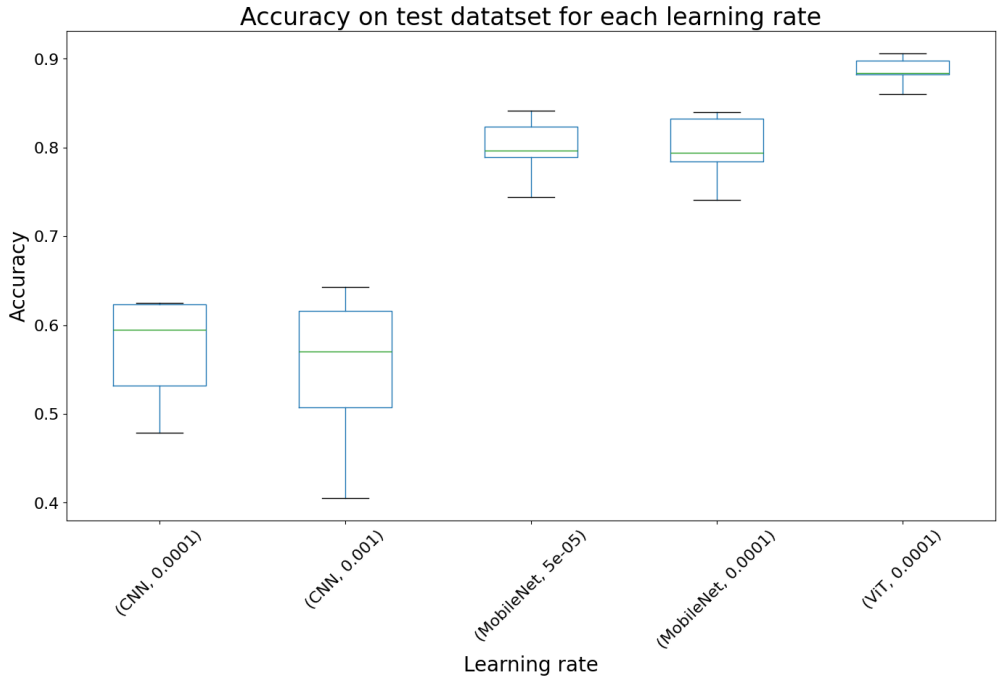

Figure 8: Accuracy on test dataset, grouped by learning rate

It is clear that changing the learning rates and batch sizes didn't have big impact on the models' performance. However, it is worth noting that the training of the models with lower batch size took significantly longer. Figure presents 9 performance of the models grouped by used regularization technique.

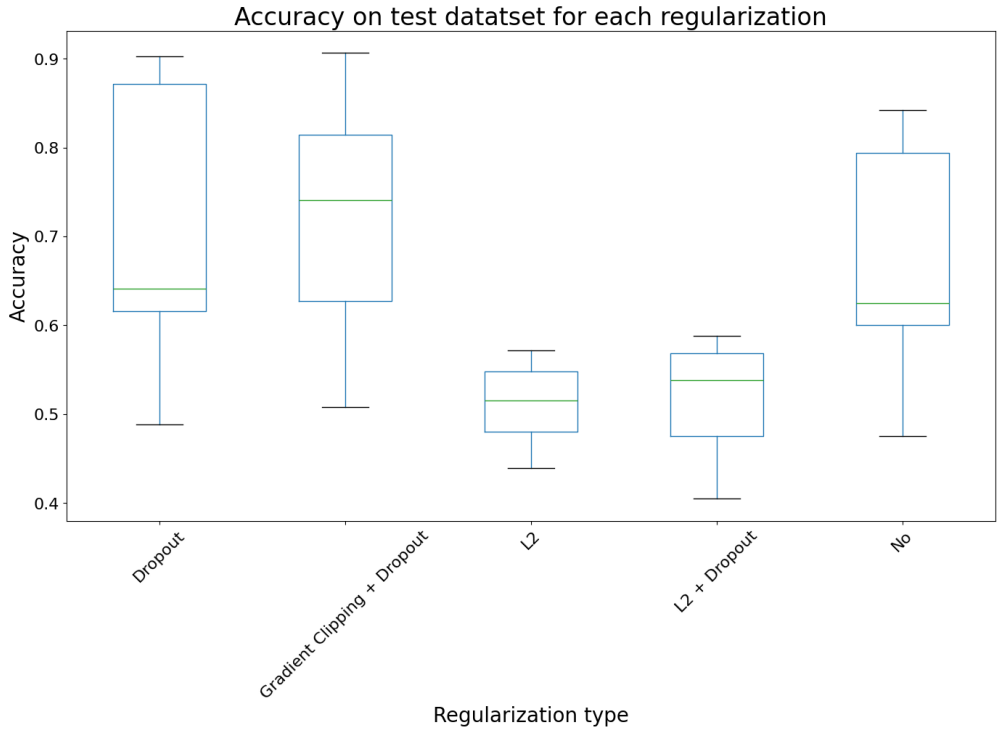

Figure 9: Accuracy on test dataset, grouped by regularization

The best performing regularization techniques are Dropout and Gradient Clipping + Dropout. Not using any regularization technique seems to be the second best option. Moreover, using L2 penalty often decreases model's performance. It may be due to additional constraints put on the parameters space. While L2 penalty can potentially reduce overfitting, it can also constrain the model's flexibility too much. Figure 10 depicts performance of the models grouped by the type of data augmentation.

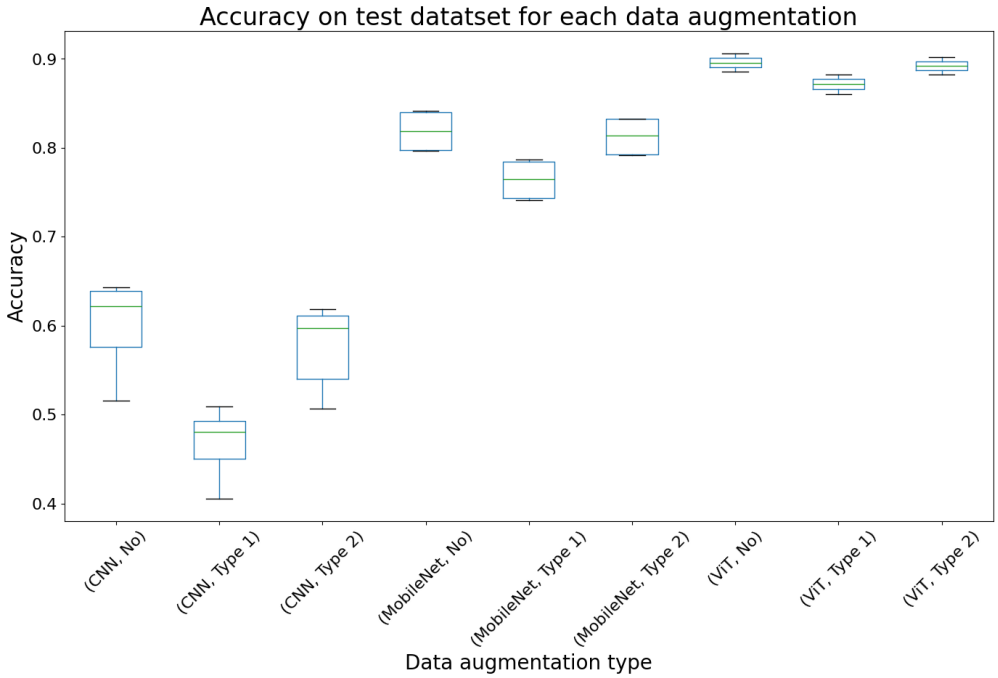

Figure 10: Accuracy on test dataset, grouped by data augmentation

Surprisingly, all models performed the best without using any data augmentation. In all cases, using type 1 data augmentation reduced models' performance. Using data augmentation of type 2, mixup, also reduced the models' performance, however the difference was smaller than in the first type. This effect may be caused by relatively small size of the images. The images of shape 32x32 are often blurry and cannot carry a lot of information and therefore, augmenting them might introduce additional noise or distortions that hinder rather than aid in the learning process. Lastly, Figure 11 presents the difference in performance of the pretrained and fine-tuned models, alongside the CNNs.

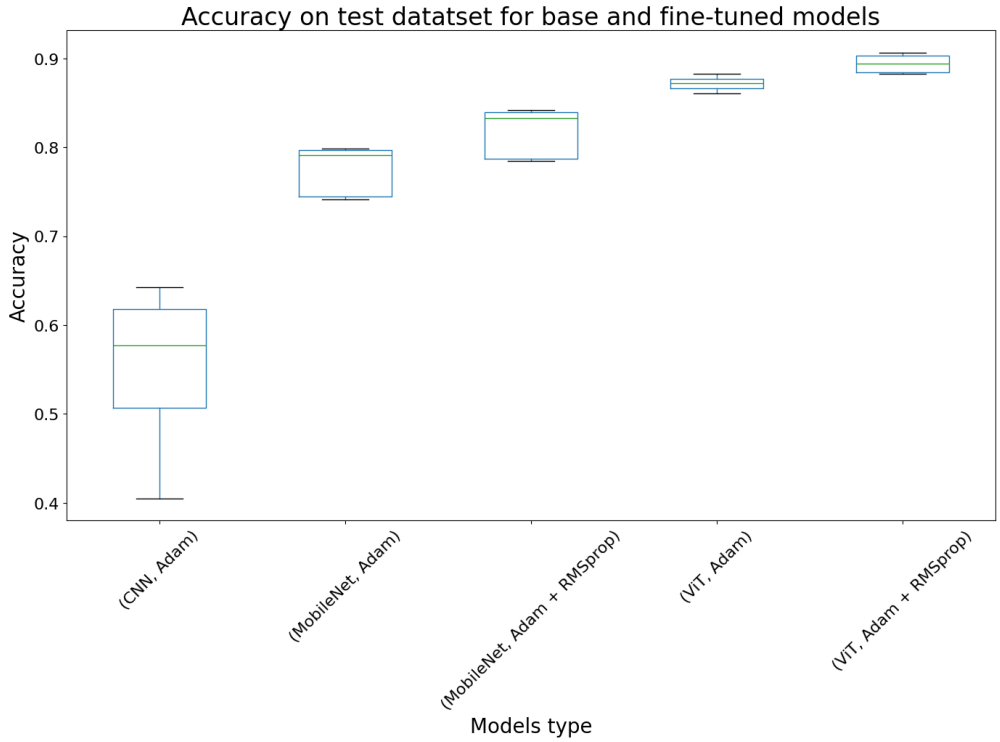

Figure 11: Accuracy on test dataset of pretrained and fine-tuned models

As expected the fine-tuned models perform significantly better than just pretrained models.

7 Conclusion

The best accuracy of 90.62% was achieved by a ViT, without data augmentation. Tested pairs of learning rates and batch sizes didn't have a significant effect on the models' performance. However, lower batch sizes increased the training time. The best regularization techniques were Dropout and Gradient Clipping + Dropout. Surprisingly not using any data augmentation turned out to be the best approach for this problem.

In conclusion, results confirmed that the CNN will serve as a baseline results. MobileNet provided a good middle-ground between performance and efficiency. Lastly, ViT outperformed all other models, at the cost of much more computation.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Luke Nicholas Darlow, Elliot J. Crowley, Antreas Antoniou, and Amos J. Storkey. CINIC-10 is not imagenet or CIFAR-10. *CoRR*, abs/1810.03505, 2018.

[3] Yann N. Dauphin, Harm de Vries, Junyoung Chung, and Yoshua Bengio. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. *CoRR*, abs/1502.04390, 2015.

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[5] A. Hosna, E. Merry, J. Gyalmo, et al. Transfer learning: a friendly introduction. *Journal of Big Data*, 9(1):102, 2022.

[6] 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. *CoRR*, abs/1905.02244, 2019.

[7] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015.

[8] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[10] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[12] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *CoRR*, abs/1710.09412, 2017.