Classification of Kidney Cortex Cells(COM2028)

Andre Henriques (6644818)

Introduction of the project

This project attempts to classify images of the human kidney cortex[1, 2] in to 8 distinct categories. This dataset is part of the MedMNIST V2[1] dataset of biomedical images that are MNIST[3] like. As such the the 200 000 images(150000 for training and 50000 for testing) have been formatted as 28 by 28 pixels gray-scale images. The training images will also be split into validation sets and training sets, with a split of 30%.
As for the expected results, in the original paper[1], the dataset was tested against various models, with best result obtained, in terms of accuracy, was 70%. The objective for this project is to achieve a accuracy over 65% on the testing set. And the final model succeeds in this.

Analysis of the Problems

Since this is a machine learning problem there is a need of a library to help us code the models Tensorflow[4] was the chosen for this due to it's simplicity.

Loading Data:
The data was given as a folder with all the training images and a .csv file that contains the labels for said images. While the data in this format could be loaded using a ImageDataGerator, I decided to learn how tensorflow's dataset system works, and manually load the labels of the images. An example on tensorflow's official documentation shows a base example on how to load loading images manually[5] and that example was modified to fit the needs of the project.

Challenges:
As with most projects that involve machine learning, computing resources is one of the biggest problems as they are limited. My personal computer was used to run the models instead of Googles Colab platform. As there would be no time limitations or timeouts. This does not remove all the limitations as the GPU on my computer only has 8 Gigabytes of VRAM which limits the size of the models that can be trained.

Technical issues:
While the dataset did not have any unusable images the small size of the images and the limited color range limits the amount of information that can be extracted.
It's important to know that some the images have visual artifacts(Fig. 1) that also limit even more the amount of data that can be extracted from them.
The data set is also not balanced witch makes it harder the model to learn some of the categories of the cells.

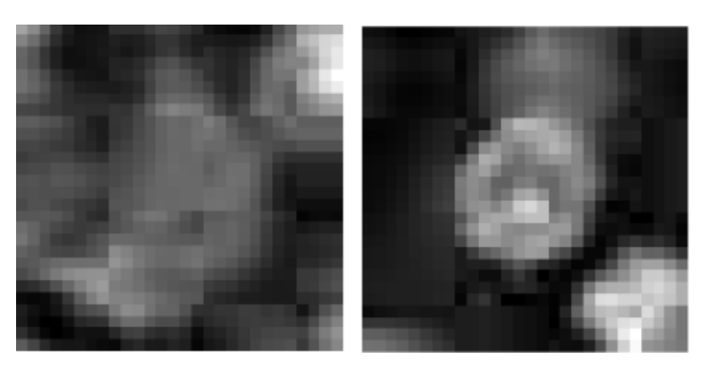
Fig. 1: Images with visual artifacts

Pre-processing and Overfitting prevention:
It is also important to add pre-processing before the model so that during training some extra images can be "generated" that the model can use, this will helps with reducing overfitting.
Therefore all models had a starting data augmentation section.
All models also had dropout layers to prevent overfitting.
Classification techniques:
There were many classification techniques attempted, the information about it's design and results will be combined into a section about that specific classification method.
The classification methods attempted were:

- Multilayer Perceptron
- Simple Convolution neural network(CNN)
- Multi-model Simple CNN
- Generative Adversarial Network(GAN)
- Block CNN

Multilayer Perceptron

While multilayer perceptrons[6] are not the optimal for the processing the images, the small size of the image makes it possible the use the multilayer perceptrons.
As expected a MLP model did not yield good results with the tested models obtaining around 50% accuracy.

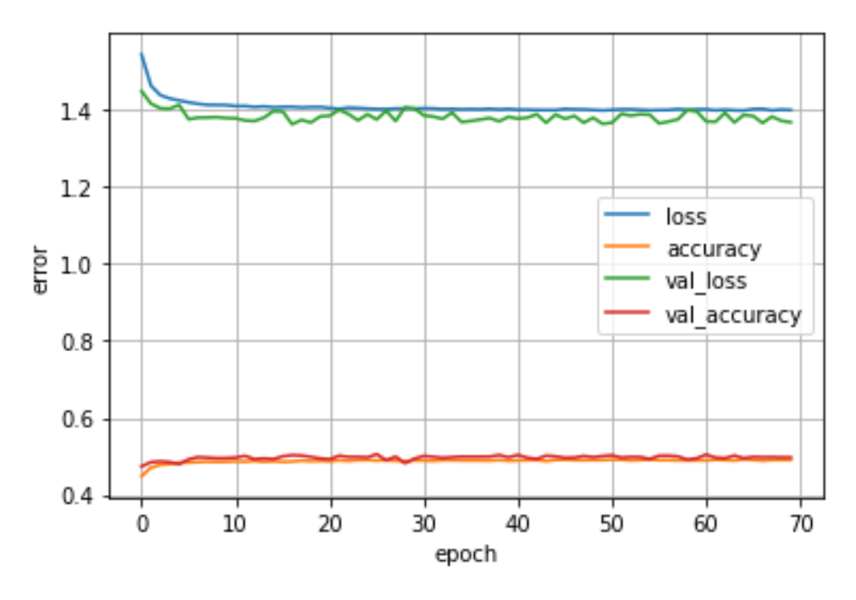

Fig. 2: Results of the MLP

Simple CNN

A simple CNN was able to provide better results with less training compared with the the multiplayer perceptron. For a CNN that had structure presented in Fig. 3 resulted in 61% accuracy, which still leaves room for improvement.

Operation Layer	Number of filters	Size of each filter	stride value	padding	size of the output image
Convolution(Conv)	64	3×3	1×1	same	$28 \times 28 \times 64$
MaxPooling(Max Pool)	1	2×2	2×2	same	$14 \times 14 \times 64$
LeakyRelu	-	-	-	-	$14 \times 14 \times 64$
Dropout	-	-	-	-	$14 \times 14 \times 64$
Conv	128	3×3	1×1	same	$14 \times 14 \times 128$
Max Pool	1	2×2	2×2	same	$7 \times 7 \times 128$
LeakyRelu	-	-	-	-	$7 \times 7 \times 128$
Dropout	-	-	-	-	$7 \times 7 \times 128$
Conv	256	3×3	1×1	same	$7 \times 7 \times 256$
LeakyRelu	-	-	-	-	$7 \times 7 \times 256$
Dropout	-	-	-	-	$7 \times 7 \times 256$
Dense layers					

Fig. 3: structure of the Simple CNN

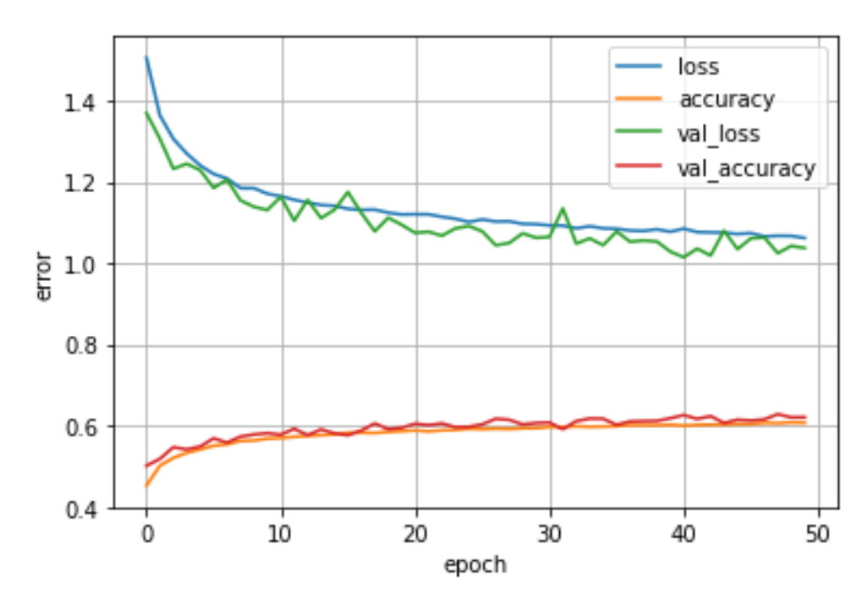

Fig. 4: Results of the Simple CNN

Multi Model CNN

The Multi Model CNN is very similar to the single model approach, but in this case each category was given a model. That model was trained to only recognize that specific cell type.
Each model was trained individually, when there is a need to predict, all models are presented with the same image, and the type is decided by measuring which model is more confident on the image being it's type.
In the end this model had similar results to the simple CNN, with overall smaller models, but an increase in the total training time, as the number of models to train also increased.

Operation Layer	Number of filters	Size of each filter	stride value	padding	size of the output image
Convolution(Conv)	64	3×3	1×1	same	$28 \times 28 \times 64$
MaxPooling(Max Pool)	1	2×2	2×2	same	$14 \times 14 \times 64$
LeakyRelu	-	-	-	-	$14 \times 14 \times 64$
Dropout	-	-	-	-	$14 \times 14 \times 64$
Conv	128	3×3	1×1	same	$14 \times 14 \times 128$
Max Pool	1	2×2	2×2	same	$7 \times 7 \times 128$
LeakyRelu	-	-	-	-	$7 \times 7 \times 128$
Dropout	-	-	-	-	$7 \times 7 \times 128$
Dense layers					

Fig. 5: Structure of each of the models used by the multimodel approach

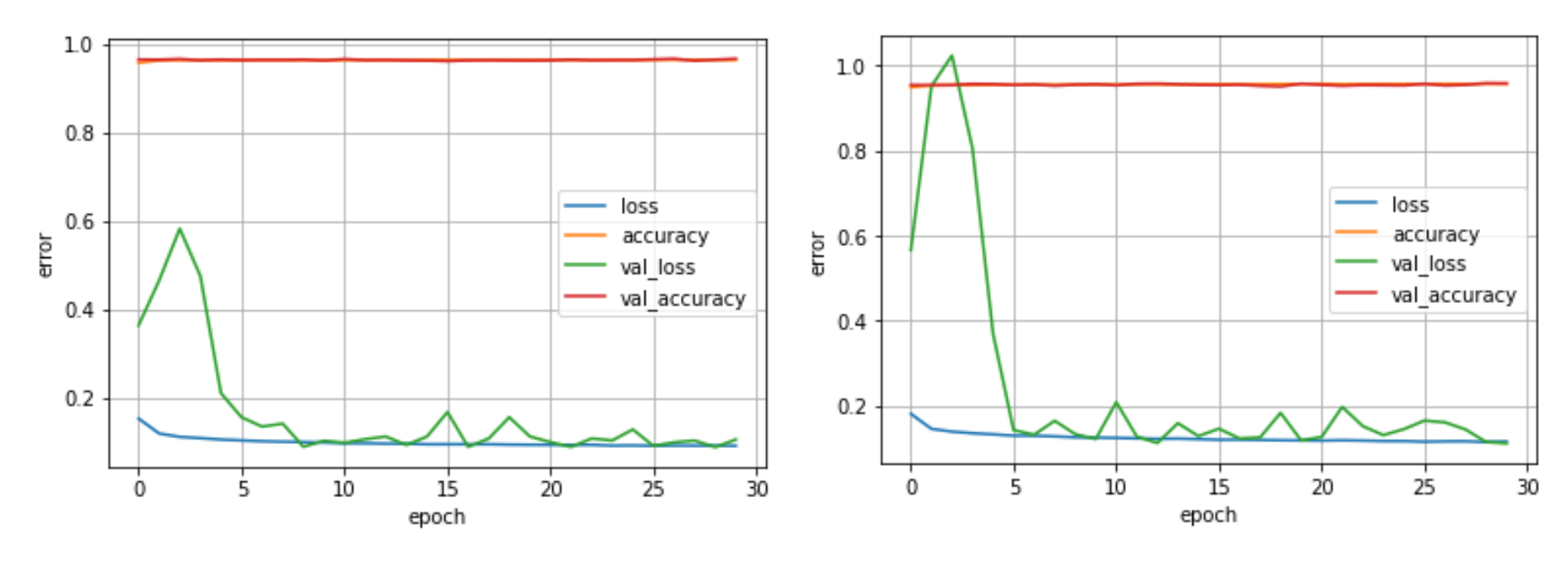

Fig. 6: Results of some of the models

GAN

GAN[7] are created by having 2 models a discriminator and generator. The generator tries to create new images based on the dataset and the discriminator tries to tell the difference between the generated images and the real ones. The loss of the discriminator is based on how good is the generator at recognizing the differences between the generated image and the real one, and so it's the generators. In the best case, both the generator and the discriminator get incredibly effective at generating/discriminating images.
The generator and the discriminator were modified so that the they would take into account type of the cells that the original image had.
Unfortunately the generator model did not converge and the images that where generated, where not good enough for this method to be effective.
The overall accuracy ended up being around 30%.
After the last model was created I discovered that the average pool function is better for this images and which means that generator model could have converged if they were used instead of the Max pooling layers that were used.

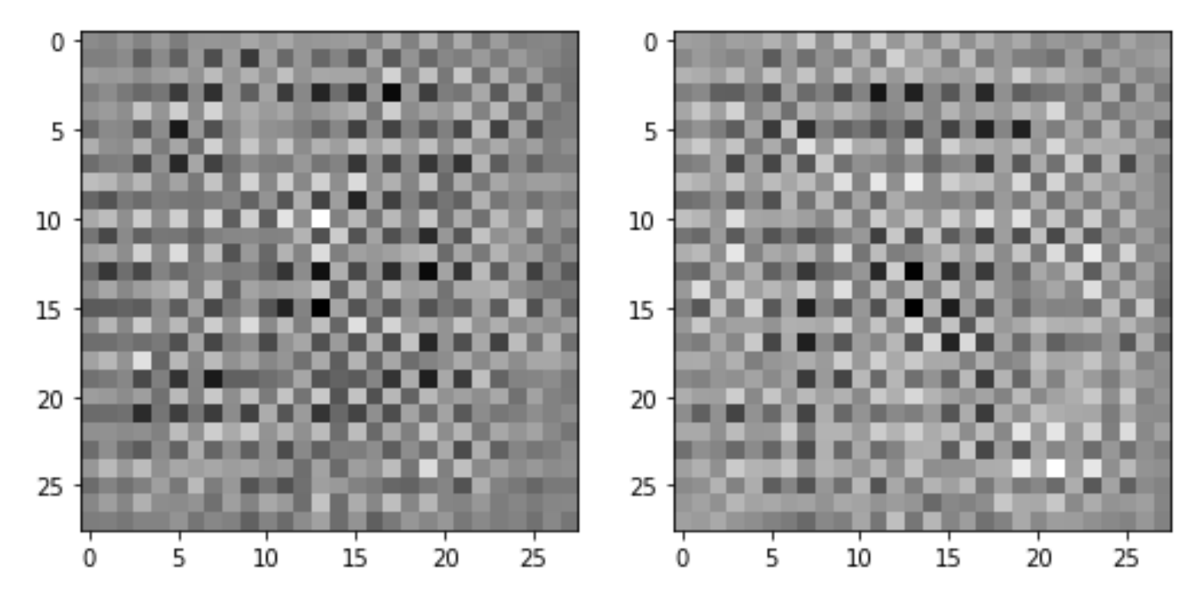
Fig. 7: Failed generated images

Block CNN

The Block CNN is formed a group of blocks(defined in Fig. 8), the blocks contain a group of convolution layers separated with ReLU activation functions. Each block ends with a batch normalization, a pooling, a LeakyReLU, and a Dropout layers.
This design was chosen do to many tests, where the value accuracy and tested for improvements, and the bests techniques where selected.
The AdamW optimizer was chosen as it helps with reducing overfitting[8]. Average pooling was found to be more suited for this kind of images as they don't have hard edges which benefit more from Max pooling.
As a final optimization this model was not loaded the images as 28 by 28 images but 32 by 32 upscaled images, as it provided more data to the model witch tends improve performance. After the model was build the model was run but with gamma corrected images[9] which also improved performance.

Operation Layer	Number of filters	Size of each filter	stride value	padding	size of the output image
Conv ReLu	number of filters -	Size of each filter -	1×1 -	same -	$\text{Input}_x \times \text{Input}_y \times$ Number of filters
(block size times)					
Batch Normalization Pooling Function LeakyRelu Dropout	- 1 - -	- 2×2 - -	- 2×2 or 1×1 - -	- valid or same - -	$\text{Input}_x \times \text{Input}_y \times$ Number of filters or $\frac{\text{Input}_x}{2} \times \frac{\text{Input}_y}{2} \times$ Number of filters

Fig. 8: Structure of each block

Then this sets of this blocks are used to create the full network.

Operation Layer	Size of The block	Number of filters	Size of each filter	padding	Pooling function	size of the output image
Block	1	64	1	same	Average	$32 \times 32 \times 64$
Block	2	128	3	valid	Average	$16 \times 16 \times 128$
Block	2	250	3	valid	Average	$8 \times 8 \times 250$
Block	2	500	3	valid	Average	$4 \times 4 \times 500$
Block	1	1000	3	same	Average	$4 \times 4 \times 1000$
Global Average						

Fig. 9: Structure of the network

The first block is used as a way to increase the number of channels of the input image without the increase the big computational overhead due to the 1 by 1 filter, the same way it's used by ResNet[10].

Global average Pooling proved to be better than using dense layers, which is supported by other studies dealing with the classification other biomedical issues[11, 12]. And and this case the model with global average pooling (Fig. 13) outperformed the one without global average pooling (Fig. 10).

A deeper(same structure but bigger block sizes) model, then the one shown in Fig. 9 without gamma correction was able to get an accuracy score of 66.8%, while this smaller model was able to 67.23%.

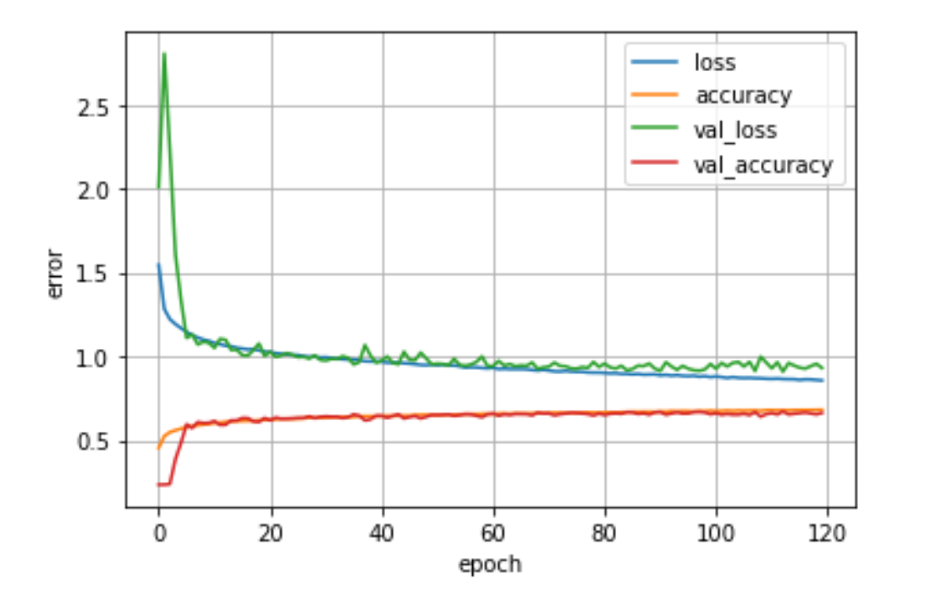

Fig. 10: Results with larger model and without global average; gamma correction

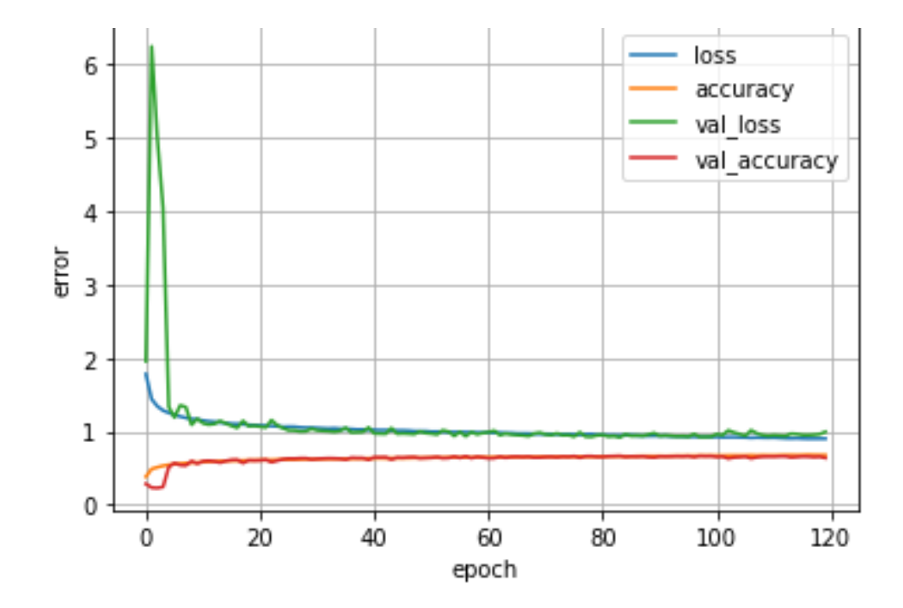

Fig. 11: Results with larger model and without: gamma correction

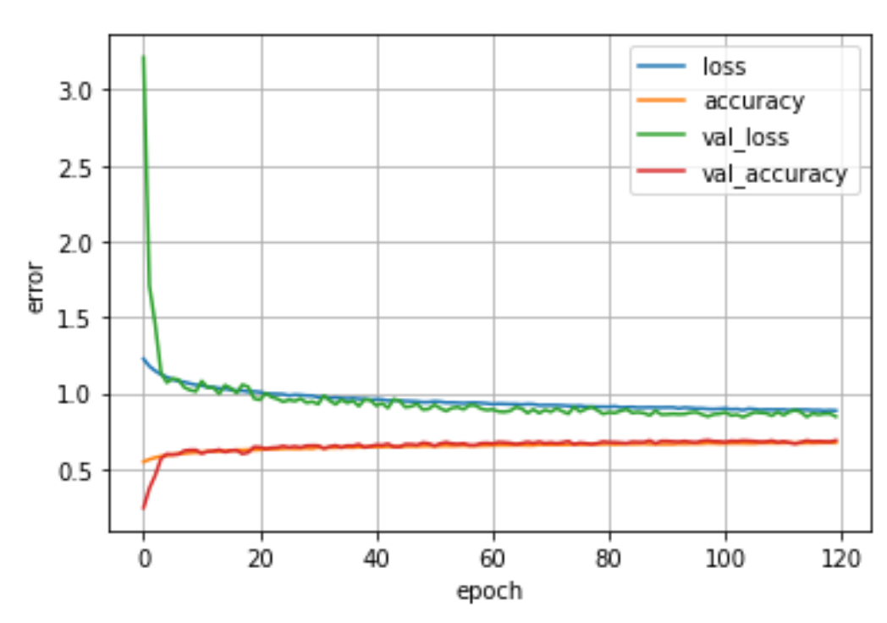

Fig. 12: Final results

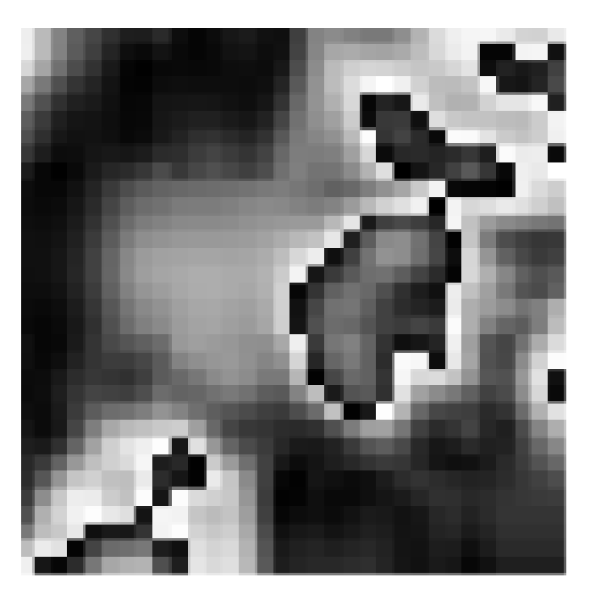
Fig. 13: Gamma corrected cell

Results

As it was mention before the model that resulted in better accuracy scores was the Block CNN model.
But this does not mean that it is the best model. I expect if the block CNN was applied with a multi model approach it would out perform the main base block CNN model.
Machine learning could also be used to train the hyper parameters of the model, as it was done with the original paper[1], with a tool like AutoKeras[13].
Since we only can submit one notebook, but more models were created the rest of the models can be found in this GitHub repository: `https://github.com/andr3h3nriqu3s11/COM2028-other-models`

References

[1] Jiancheng Yang et al. "MedMNIST v2: A Large-Scale Lightweight Benchmark for 2D and 3D Biomedical Image Classification". In: *CoRR* abs/2110.14795 (2021). arXiv: 2110.14795. URL: https://arxiv.org/abs/2110.14795.
[2] Andre Woloshuk et al. "In Situ Classification of Cell Types in Human Kidney Tissue Using 3D Nuclear Staining". In: *Cytometry Part A* 99.7 (2021), pp. 707–721. DOI: https://doi.org/10.1002/cyto.a.24274. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cyto.a.24274. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cyto.a.24274.
[3] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.
[4] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.
[5] *Load and preprocess images: Tensorflow Core*. URL: https://www.tensorflow.org/tutorials/load_data/images.
[6] Christoph von der Malsburg. "Frank Rosenblatt: Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms". In: *Brain Theory* (Jan. 1986), pp. 245–248. DOI: 10.1007/978-3-642-70911-1_20.
[7] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. DOI: 10.48550/ARXIV.1406.2661. URL: https://arxiv.org/abs/1406.2661.
[8] Ilya Loshchilov and Frank Hutter. "Fixing Weight Decay Regularization in Adam". In: *CoRR* abs/1711.05101 (2017). arXiv: 1711.05101. URL: http://arxiv.org/abs/1711.05101.
[9] Sonali Dash and Manas Ranjan Senapati. "Enhancing detection of retinal blood vessels by combined approach of DWT, Tyler Coye and Gamma correction". In: *Biomedical Signal Processing and Control* 57 (2020), p. 101740. ISSN: 1746-8094. DOI: https://doi.org/10.1016/j.bspc.2019.101740. URL: https://www.sciencedirect.com/science/article/pii/S1746809419303210.
[10] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.
[11] R. Lokesh Kumar et al. "Multi-class brain tumor classification using residual network and global average pooling". In: *Multimedia Tools and Applications* 80.9 (Apr. 2021), pp. 13429–13438. ISSN: 1573-7721. DOI: 10.1007/s11042-020-10335-4. URL: https://doi.org/10.1007/s11042-020-10335-4.
[12] Sara Hosseinzadeh Kassani et al. "Breast Cancer Diagnosis with Transfer Learning and Global Pooling". In: *2019 International Conference on Information and Communication Technology Convergence (ICTC)*. 2019, pp. 519–524. DOI: 10.1109/ICTC46691.2019.8939878.
[13] Haifeng Jin, Qingquan Song, and Xia Hu. "Efficient Neural Architecture Search with Network Morphism". In: *CoRR* abs/1806.10282 (2018). arXiv: 1806.10282. URL: http://arxiv.org/abs/1806.10282.