# Assignment 3: Classification of Image Data with Multilayer Perceptrons and Convolutional Neural Networks

261250317 Léo Valette
261091403 Kayvan Dharsee

## Abstract

In this assignment, we utilized the OrganAMNIST dataset to investigate how multilayer perceptrons and convolutional neural networks performed for a multi-class classification task using image data. We further investigated how the architecture of both of these models and the data used affected the accuracy of these classifiers, along with comparing the models to each other to highlight the differences between MLPs and CNNs when working with image data. We discovered that a deeper, regularized MLP trained on normalized data allowed for accurate classification. However, using a CNN with adequate amounts of convolutional and fully connected layers offers far more accurate predictions due to positional encoding being implemented. This was most evident in our use of the pretrained model ResNet50 as a feature extractor, where we were able to achieve the highest accuracy of classification.

## Introduction

The discovery of Deep Learning revolutionized the automated analysis of medical imaging, with organ classification being a prime example of this. The aim of this assignment was to study the performance differences between multilayer perceptrons and convolutional neural networks in medical image classification using the OrganAMNIST dataset.

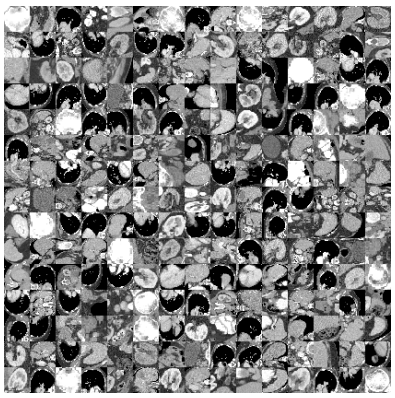


Figure 1: Example of images

Although MLPs have historically been used for image classification Lecun et al. (1998), today CNNs (and more recently transformers) are the state of the art as they enable us to better capture local relationships in images.

Our research focuses on organ classification using the OrganAMNIST dataset, which contains black-and-white images of various organs. Through numerous experiments, we explored different architectures and configurations, such as the number and size of hidden layers, different activation functions, and different regularization techniques that impacted model performance. We have also studied the impact of pre-processing, in particular normalization and image resolution, on the accuracy of our models.

Our results show that, although a properly regularized MLP can achieve reasonable performance (75.3% accuracy), CNNs consistently outperform MLPs in this task, achieving up to 87.1% accuracy. Furthermore, the use of transfer learning with a pre-trained ResNet50 He et al. (2016) model gave the best results, with 92.3% accuracy, underlining the benefits of using CNNs for medical image classification tasks.

## Dataset

The OrganAMNIST dataset consists of 58,830 grayscale images of organs such as the heart, bladder, and liver. Pairs of organs are split into different classes, such as lung-left and lung-right, resulting in 11 classes. A majority of the images were of the liver, while images of both the left and right femurs along with the heart appeared less than other organs in the dataset as shown in Figure 2. The dataset came in various image sizes, with 28 by 28 and 128 by 128 pixels being the sizes we used.

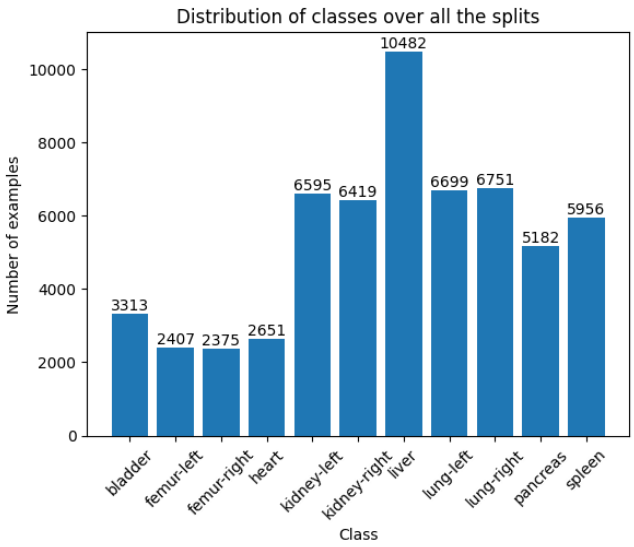



Figure 2: Class distribution

It also came split into training, testing, and validation sets, with roughly 30% of the data belonging to the test set, and the rest divided between the train and validation set. As the dataset was so large, we utilized data loaders to load batches of the data, allowing for quicker training and evaluation.

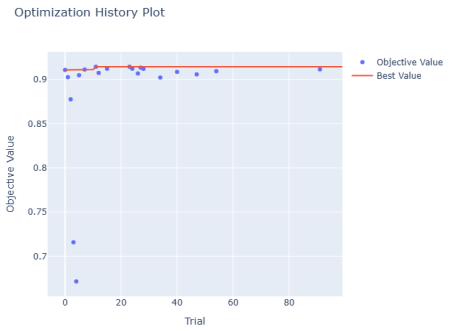
## Results

| Category | Model Name | Test Acc. | Test Loss | Epochs | Time (s) | Time / epoch ($s^{-1}$) | Trainable Params |
|---|---|---|---|---|---|---|---|
| **MLP** | No hidden | 59.7% | 2.010 | 7 | 59 | 8.4 | $8.6 \times 10^3$ |
| | 1 hidden | 70.9% | 1.884 | 15 | 124 | **8.2** | $2.0 \times 10^5$ |
| | 2 hidden (Base model) | 71.6% | 1.838 | 8 | 66 | 8.3 | $2.7 \times 10^5$ |
| | Leaky-ReLU | 71.9% | 1.852 | 13 | 110 | 8.5 | $2.7 \times 10^5$ |
| | Tanh | 70.6% | 1.858 | 21 | 176 | 8.4 | $2.7 \times 10^5$ |
| | L1 & L2 reg | **75.3%** | **1.828** | 15 | 127 | 8.5 | $2.7 \times 10^5$ |
| | No normalization | 65.4% | 1.973 | 10 | 84 | 8.4 | $2.7 \times 10^5$ |
| | 128px MLP | 74.7% | 1.837 | 8 | 96 | 12 | $4.3 \times 10^6$ |
| | Big model (Bonus) | 74.4% | 1.882 | 17 | 195 | 11.5 | $1.7 \times 10^7$ |
| **CNN** | 28px CNN | 85.6% | 1.684 | 17 | 186 | **10.9** | $3.3 \times 10^5$ |
| | 128px CNN | **87.1%** | **1.671** | 28 | 1170 | 41.8 | $\mathbf{1.1 \times 10^8}$ |
| **Pre-trained** | Transfer Learning (ResNet) | **92.3%** | **1.625** | 37 | 1437 | **38.8** | $4.6 \times 10^5$ |

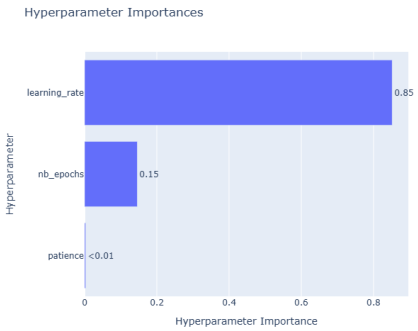
Table 1: Performance comparison of MLP, CNN, and Pretrained Models on Test set

Before beginning our experiments, we completed hyper-parameter tuning for the learning rate, number of epochs, and patience of our MLP. Patience allowed us to regularize our model using early stopping, and load the most accurate model for evaluation instead. We were then able to find the optimal values for these parameters, along with the importance of each. We utilized Optuna for the tuning, as it uses a Bayesian optimization algorithm which is both more accurate and more efficient than traditional grid search. In non-rigorous terms, "Optuna uses a history record of trials to determine which hyperparameter values to try next. Using this data, it estimates a promising area and tries values in that area" *Optuna: An Automatic Hyperparameter Optimization Framework — Open Data Science Conference* (n.d.).

All of our training times, including the ones for the MLPs, were done on a T4 GPU. For the MLP, we only used basic matrix computation functions from pytorch, not any of the autograd features. We've separated the code into 3 notebooks, and all the results of the various hyperparameter searches are available in those 3 different notebooks, along with additional visualizations of the searches.




(a) Optimization history




(b) Hyperparameters importance

| Metric | Value |
|---|---|
| Learning rate | 0.0225 |
| Nb epochs | 78 |
| Patience | 4 |

(c) Best values found

Figure 3: Optimization for the first basics models (without regularization)

In the first experiment, we analyzed how the amount of hidden layers in our MLP affected its performance. We saw a leap in performance from no hidden layers to one hidden layer, with accuracy increasing from 59.7% to 70.9%. This is expected as an MLP with no hidden layers can only create linear decision boundaries, and thus cannot classify non-linear data well. Adding a hidden layer with a non-linear activation function such as ReLU allows our model to create non-linear decision boundaries, letting it classify non-linear data with better accuracy. Adding a second hidden layer had a positive yet negligible effect on performance, although deepening the network is supposed to make the model more powerful. This is due to the model becoming too expressive and overfitting to the training data, as seen by the increase in train accuracy not being followed by an increase in validation accuracy in the graph below. Training curves for the first 2 models and all other MLPs are in this figure.

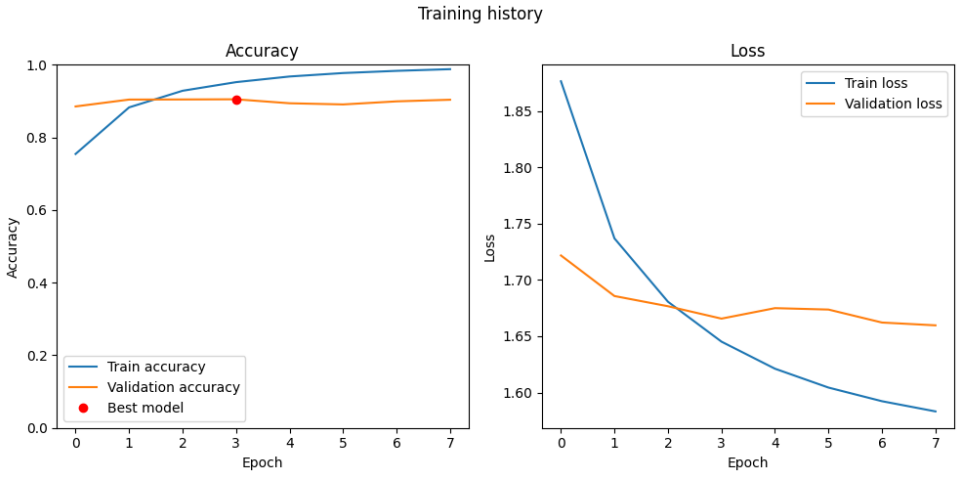



Figure 4: Training accuracy and loss on the train and validations splits for the 2 hidden layer model

In the second experiment, we analyzed how various activation functions affected the performance of our MLP. With two hidden layers, we see minimal differences in accuracy from using tanh, ReLU, and Leaky ReLU as our activation function of choice. Tanh had the worst performance with 70.6% accuracy, then ReLU with 71.6%, and Leaky ReLU had the best performance with 71.9% accuracy. Tanh likely performed on par with the other two due to our data being normalized, reducing the chance of vanishing gradients caused from passing values ranging far from 0. The similarity in ReLU and Leaky ReLU indicates that we do not run into many zero-gradient problems with ReLU, as that is what Leaky ReLU is designed to improve on. This is likely due to our initialization ensuring that we have active units at the beginning of optimization.

In our third experiment, we improved the accuracy of our two hidden layers, ReLU activation MLP by adding L1 and

L2 regularization with independent lambda values. Combining this with proper hyper-parameter tuning allowed for the best balance of both regularization methods for our model. It is important that we tune the hyper-parameters we tuned in the first run again, as the regularized model has different optimal values. We analyzed the importance of the regularization parameters, and found that our L1 lambda value had more importance than our L2 value. We saw our test accuracy increase from 71.6% to 75.3%. This result is expected, as our model was previously overfitting. Regularization ensures our model does not overfit, allowing for better performance on the test set.

In our fourth experiment, we trained our base MLP with unnormalized data, and observed a dropoff in performance with our model having 65.4% accuracy on the test set. Passing unnormalized data to an MLP can negatively impact the performance in numerous ways. For example, when we pass unnormalized data to our MLP, the noise of the data can negatively affect training, causing our model to not learn the important features in the data as clearly, as it learns the noise of the unnormalized data.

In the fifth experiment, we utilized the larger 128 by 128 pixel images, and observed the performance of our base MLP on this data. As seen in Table 1, the training time per epochs for the model increases from 8.5 seconds to 12 seconds, our accuracy lowers to 74.7%, and the number of trainable parameters increases more than tenfold. The deterioration in accuracy was likely due to the fact that the larger image results in a massive input vector size, and the MLP may not be expressive enough to utilize all the features effectively. When analyzing the training loss of our model on the large data, we noticed that after a certain amount of epochs, we ended up with nan values for our loss. We suspected this was due to our model not being powerful enough for the larger data, and confirmed our suspicions by building a wider and deeper MLP for the large dataset which did not experience the same issues. We believe the degressive size of the hidden layers in our larger MLP allowed for the model to handle the large amount of input features more effectively than the base model (see here).

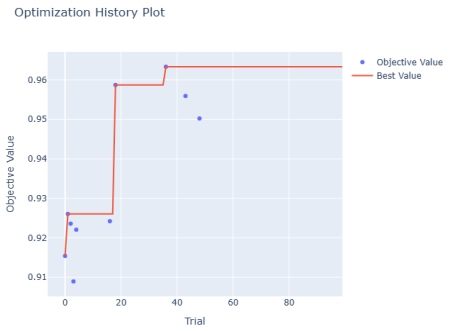
For the MLP, we found that increasing the amount of hidden layers improved performance at first, but led to overfitting. We were then able to regularize our model with L1 and L2 regularization to maximize performance. Data normalization also served a key role in producing accurate classifications, however the choice of activation function did not have a great effect on performance. A study of confusion matrices provides a better understanding of the limitations of the various models.

In our sixth experiment, we switched over to CNN architecture, building a CNN with two convolutional layers, one fully connected hidden layer, and one fully connected output layer. We used ReLU activations on all layers excluding the output layer where we utilized softmax. After tuning the hyper-parameters and convolutional parameters, we observed an increase in performance with 85.6% accuracy on the test set. The reason our first CNN was able to outperform our best MLP to such a great extent is due to the convolutional layers, which enable positional encoding. As we are working with image data, being able to encode positional information serves as a big advantage that the MLP cannot use.

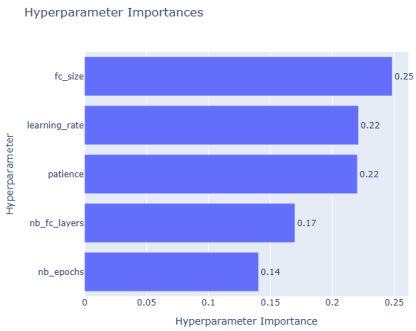
In the seventh experiment, we adjusted our CNN architecture to accommodate for the higher resolution dataset. Our optuna trials revealed that pooling had low importance and was not helpful, however we found that adding two extra fully connected hidden layers along with 286 units per layer gave the best results. We believe that pooling did not help, as we kept the number of channels constant through the convolutional layers. In classical CNNs (see LeNet-5 Lecun et al. (1998)), we tend to increase the amount of channels through the convolutional layers while adding pooling. We see that the time per epoch increased to 41.8 seconds with this larger model and the amount of epochs increased to 28. Unlike the MLP, the test accuracy actually increased to 87.1% for the larger CNN. This is likely due to our addition of layers and the ability for the convolutional layers to effectively extract information even from the higher resolution images. As the images are larger, we greatly increase the number of trainable parameters, reaching $1.1 \times 10^8$.

In our final experiment, we loaded the pre-trained model ResNet-50, froze the convolutional layers, removed the fully connected layers, and added our own to train the model for our classification task. This process is called Transfer Learning. We then tuned the hyper-parameters and searched for the optimal number and size of hidden layers to allow our model to reach optimal performance. The pre-trained model outperformed both our best CNN and best MLP, with a test accuracy of 92.3%. In terms of time per epoch, our model took longer to train than the best MLP, but shorter to train than the best CNN with 38.8 seconds per epoch. The reason our model was able to outperform the best CNN in time per epoch is due to the fact that we do not have the convolutional parameters as trainable parameters, reducing the total amount of trainable parameters to $4.6 \times 10^5$.

As seen in the confusion matrices, the pre-trained model performed the best, followed by the larger CNN, as they




(a) Optimization history



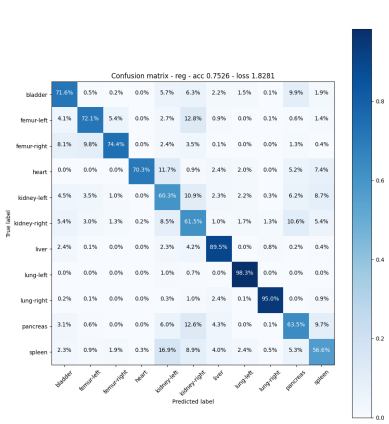

(b) Hyperparameters importance

| Metric | Value |
|---|---|
| Learning rate | $8.9 \times 10^{-5}$ |
| Nb epochs | 70 |
| Patience | 8 |
| Nb FC layers | 1 |
| FC layer size | 222 |

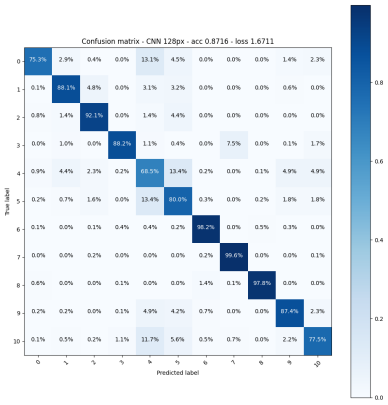
(c) Best values found

Figure 5: Optimization for Transfer Learning model

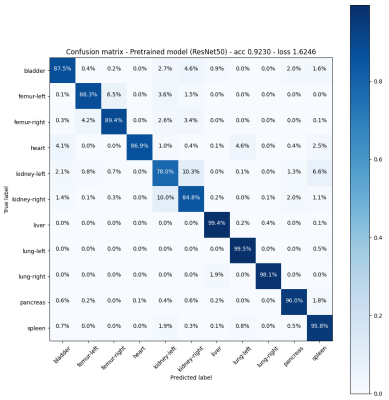
maximize the values on the diagonal to a greater extent. A key example of this performance hierarchy is the spleen class, where we only classify 56.6% of the spleen images correctly for the best MLP, 77.5% correctly for the best CNN, and 95.8% correctly for the Transfer Learning model. This means that the latest model was able to extract relevant features for this class, which the MLP was unable to do.




(a) MLP with regularization




(b) CNN 128px




(c) Transfer Learning model (ResNet-50)

Figure 6: Confusion matrices for the best MLP, CNN and the Transfer Learning model.

## Discussion and Conclusion

There are various ways to adjust the architecture of an MLP to allow it to classify organ image data effectively. It will however always be limited by the fact it cannot encode positional information of the data. The use of convolutional layers in CNNs allow us to reach performances we simply cannot reach with the MLP. This improvement is also scalable as shown in the improvement in performance from the pre-trained model, which contains a more complex architecture than our best CNN. Utilizing the pre-trained model also allowed for faster training, as we only need to learn the fully connected layer parameters. As for future improvements, we could experiment with other pre-trained models, and find which model performs best for our organ classification task. We could also perform further experimentation on our own CNNs, such as increasing the number of channels throough the convolutional layers and experimenting with batch normalization between layers.

## Statement of Contributions

Leo and Kayvan combined their code and reports to create a final report and models, however our third group member did not contribute to this assignment.

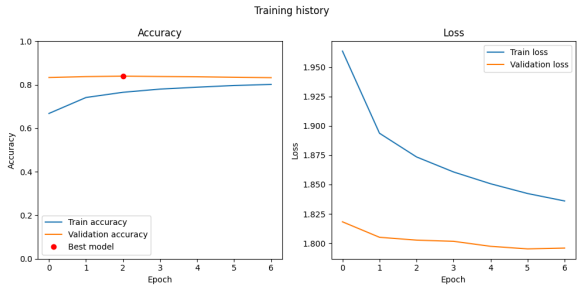
# References

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *2016 ieee conference on computer vision and pattern recognition (cvpr)* (p. 770-778). Retrieved from https://arxiv.org/abs/1512.03385 doi: 10.1109/CVPR.2016.90

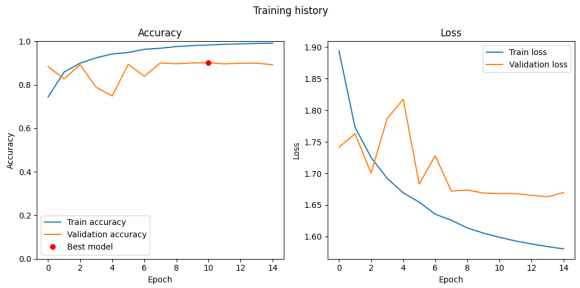
Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324. Retrieved from https://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf doi: 10.1109/5.726791

*Optuna: An automatic hyperparameter optimization framework — open data science conference.* (n.d.). https://odsc.com/blog/optuna-an-automatic-hyperparameter-optimization-framework/#:~:text=Optuna%20uses%20a%20history%20record,based%20on%20the%20new%20result. ((Accessed on 11/18/2024))

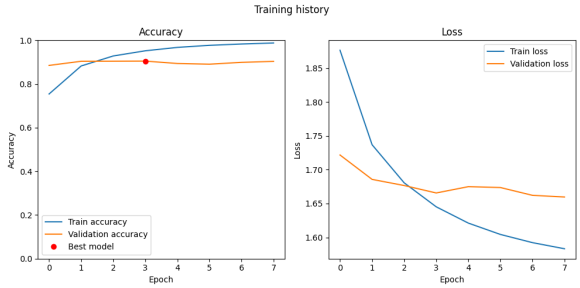
# Appendix



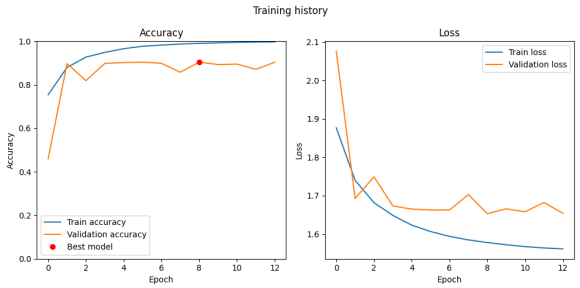

(a) No hidden



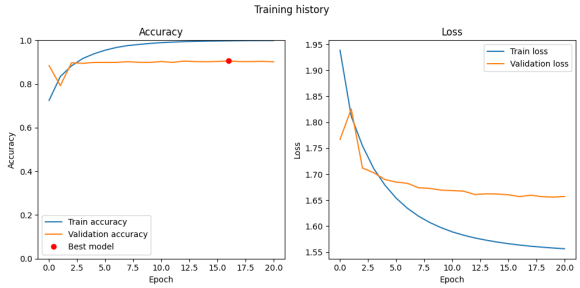

(b) 1 hidden



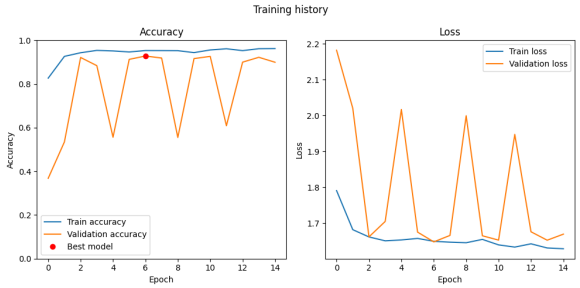

(c) 2 hidden



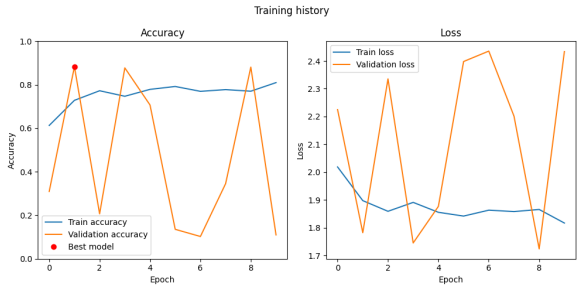

(d) Leaky-ReLU



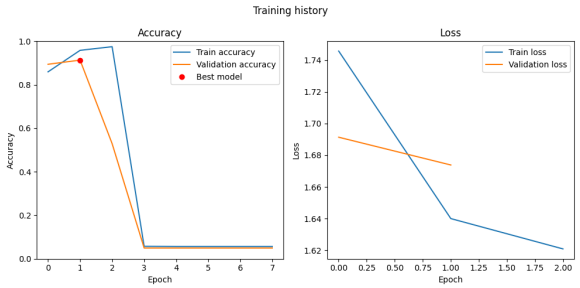

(e) Tanh




(f) L1 & L2 regularization




(g) Unnormalized images




(h) MLP 128px

Figure 7: Training curves for all 8 MLP. Each plot shows the training and validation accuracy/loss over epochs.

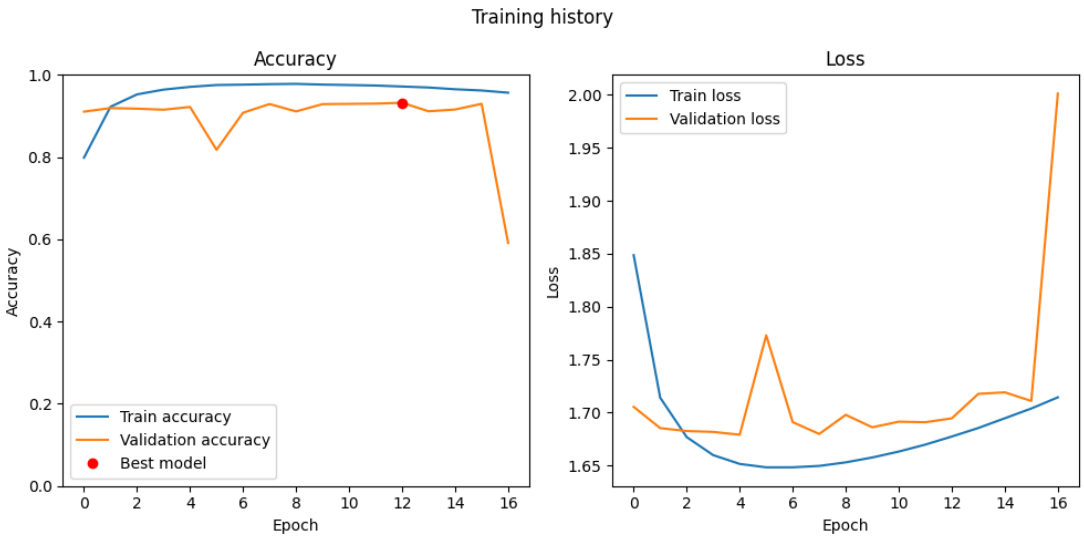
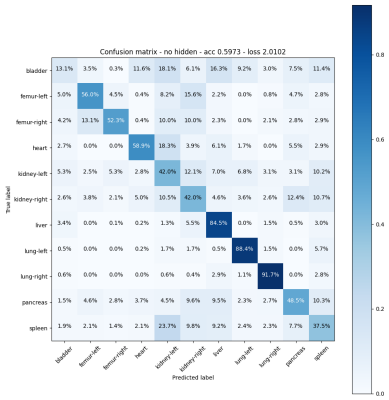
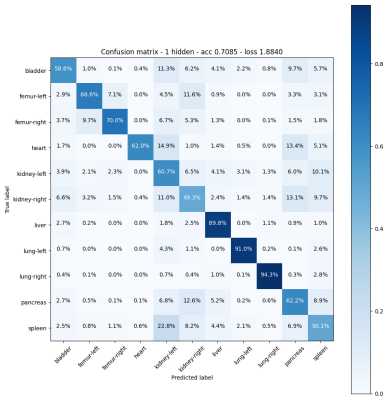


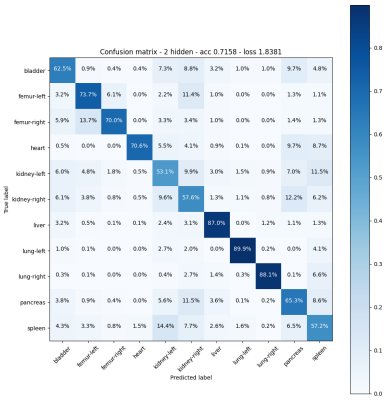

Figure 8: Training accuracy and loss on the train and validations splits for the big MLP model (bonus)



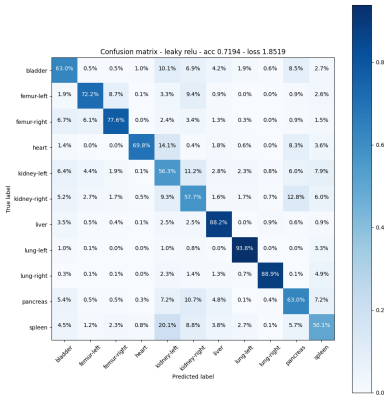

(a) No hidden



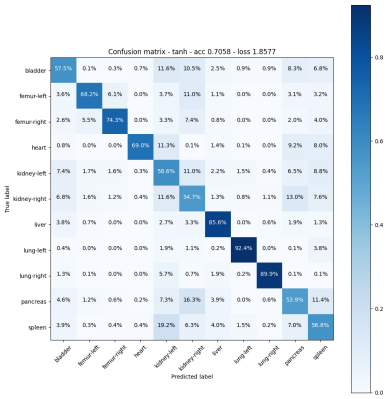

(b) 1 hidden



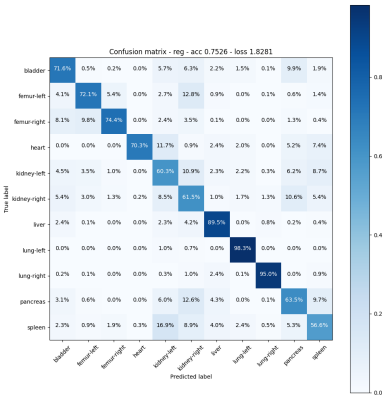

(c) 2 hidden



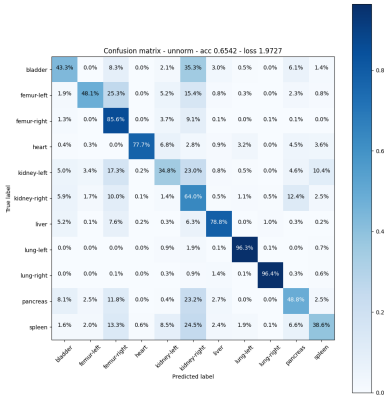

(d) Leaky-ReLU



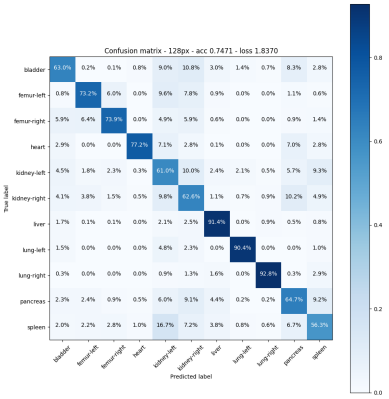

(e) Tanh




(f) L1 & L2 regularization

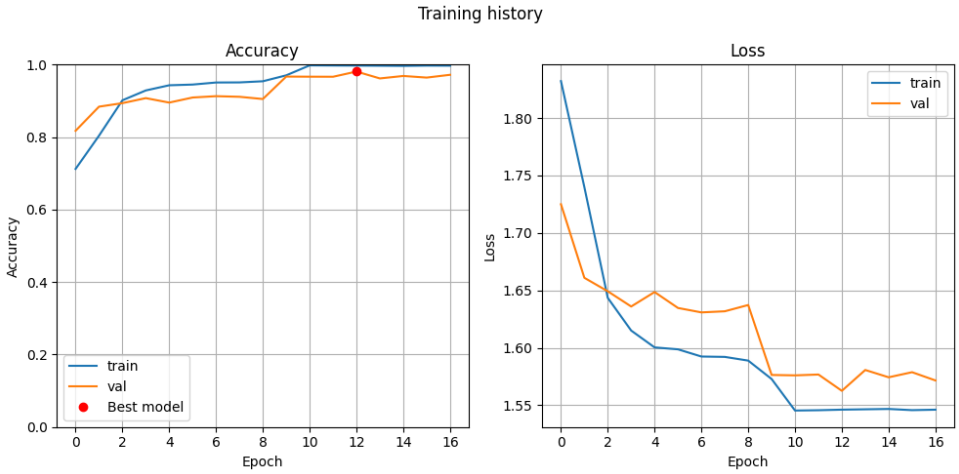



(g) Unnormalized images

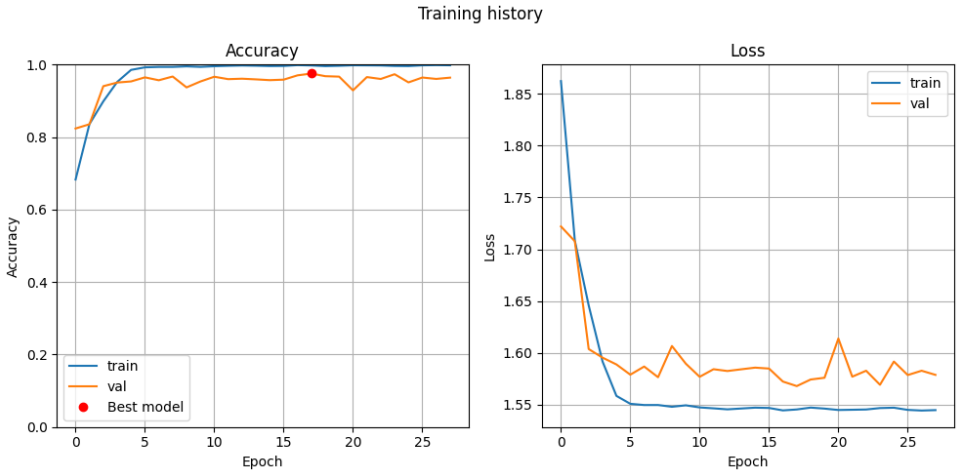



(h) MLP 128px

Figure 9: Confusion matrices for all 8 MLP.




(a) 28px images CNN




(b) 128px images CNN

Figure 10: Training curves for 2 CNNs models. Each plot shows the training and validation accuracy/loss over epochs.

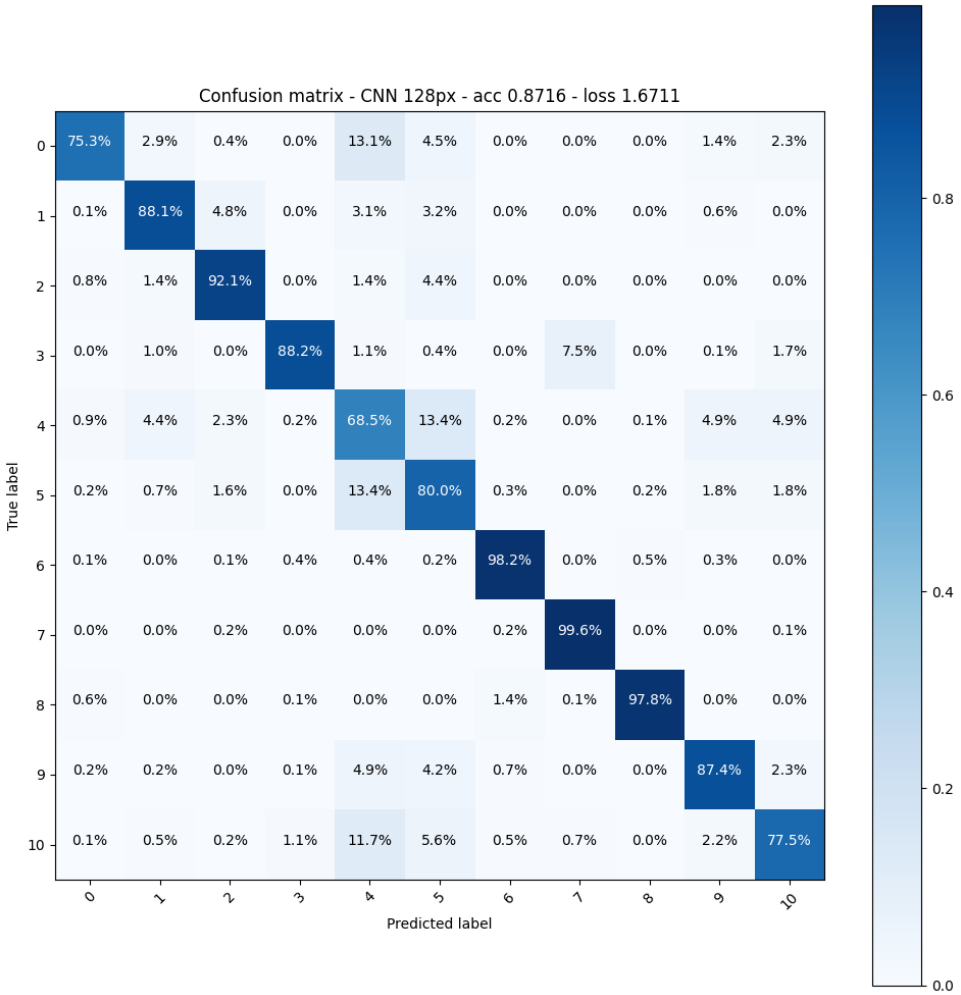



Figure 11: Confusion matrix for 128px images CNN

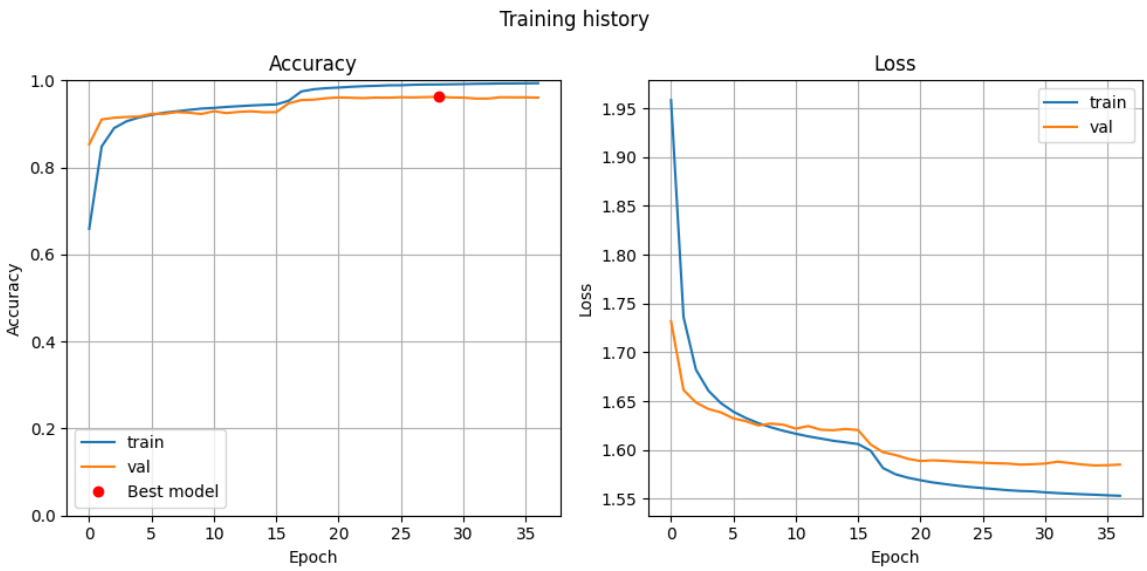



Figure 12: Training curves for Transfer Learning model (ResNet50)

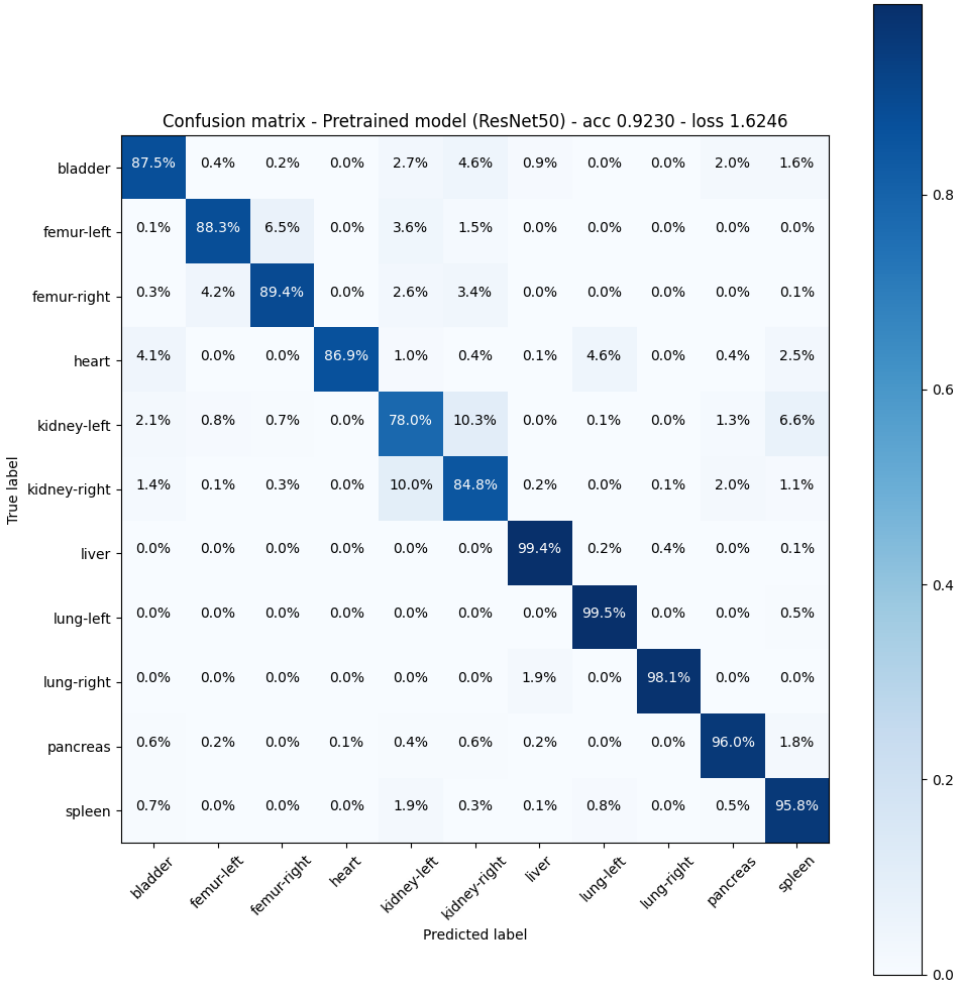
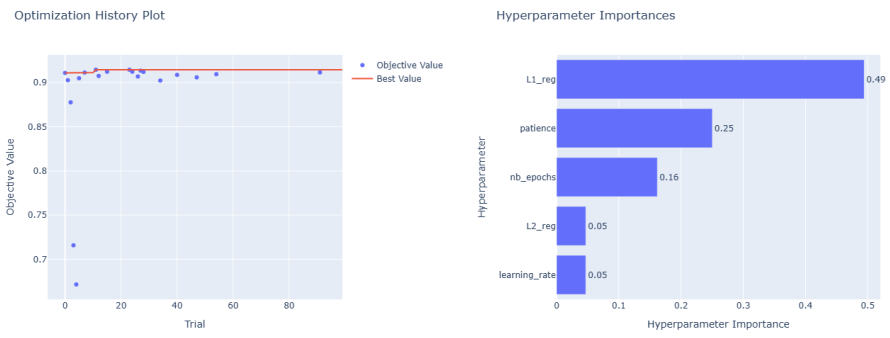



Figure 13: Confusion matrix for Transfer Learning model (ResNet50)




(a) Optimization history

(b) Hyperparameters importance

| Metric | Value |
|---|---|
| L1 reg | 0.000183 |
| L2 reg | 0.0015 |
| Learning rate | 0.0892 |
| Nb epochs | 90 |
| Patience | 8 |

(c) Best values found

Figure 14: Optimization with regularization