CompSci Project 2
Convolutional Neural Networks for Classification of Cells

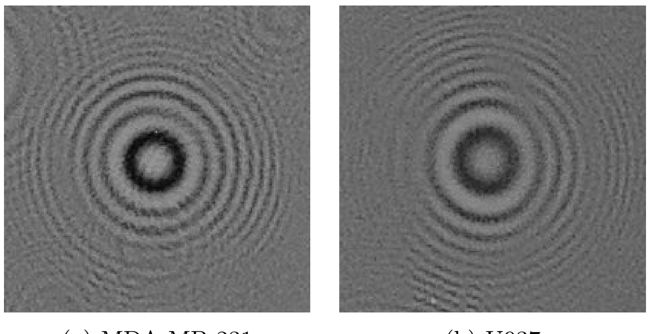
Ali Aslan Demir[1], Silja Borring Låstad[1], and Franziska Schöb[1]
[1] *The Njord Centre, Department of Physics, University of Oslo, P.O. Box 1048, 0316 Oslo, Norway*

This project uses the computational path with Convolutional Neural Networks (CNNs) to classify two types of biological cell images taken from holographic microscopy. To replace the computationally expensive reconstruction step with the power of CNNs, we used discriminative features of holographic images without pre-processing and data-augmentation. Our method obtained high accuracy in differentiating cancer cells (MDA-MB-231) from normal cells (U937), making it highly relevant for pre-cancer diagnosis. We optimized our model by exploring various CNN models, loss-activation functions, batch/epoch size, kfold split ratio, optimizer model, and learning rate to achieve the best possible classification accuracy.

I. INTRODUCTION

Holographic microscopy is a valuable computational quantitative microscopy tool widely used for microscale visualization. Its low-cost and portable optical setup make it particularly attractive for resource-limited facilities. During the hologram formation process, the interaction of light from biological cells and background light creates a diffraction pattern. This pattern represents the object located at camera plane further from its focus point, requiring computational reconstruction to find the focus [1]. However, this reconstruction process is computationally expensive and involves optimization techniques like twin image artifact elimination[2]. Recently, researchers have started to use deep learning for hologram reconstruction process [3], but it still remains a labor-intensive and error-prone process. Alternatively, scientists are started to use raw holograms for classification task to obtain high accuracy results [4].

In the project, we focused on two different types of cell holograms: MDA-MB-231 cells representing cancer (FIG. 1a) and U937 cells representing normal cells (FIG. 1b). We worked with 728 raw hologram images for each class, without any preprocessing steps, and cropped them to a size of 200x200 pixels. For the training, we used Google Colab and obtained our results with Tesla T4 and NVIDIA A100 GPUs. The dataset is directly taken from Kaggle.



(a) MDA-MB-231 (b) U937

FIG. 1: Hologram Images of Cells

II. METHODS

For the training of the simple model (our code), a sequential model consisting of three convolutional layers activated with the rectified linear unit (ReLU) function. Max pooling used to downsample the feature maps. The resulting feature maps were then flattened and fed into a dense layer with 32 neurons, followed by a 0.4 rate of dropout layer. Finally, a softmax activation function with two classes was used to classify the images.

To compare our code with the pre-trained weights and obtain better control over training metrics, we employed an another training model consisting of kFold implementation. In each fold, weights are resetted with random shuffling to remove bias. For the training, we started with VGG 16 model [5], used Resnet 50-152 models [6], and finally Xception model [7]. The dataset is initially split into a training set and a test set, with 20% of the data reserved for testing and not used in the training process. With train-validation split ratio of 0.75 balanced distribution of data achieved during training. Overall, the train-validation-test ratio becomes 60:20:20, with 60% of the data used for training, 20% for validation, and the remaining 20% for final testing. The main strategy of the model is given in FIG. 2.

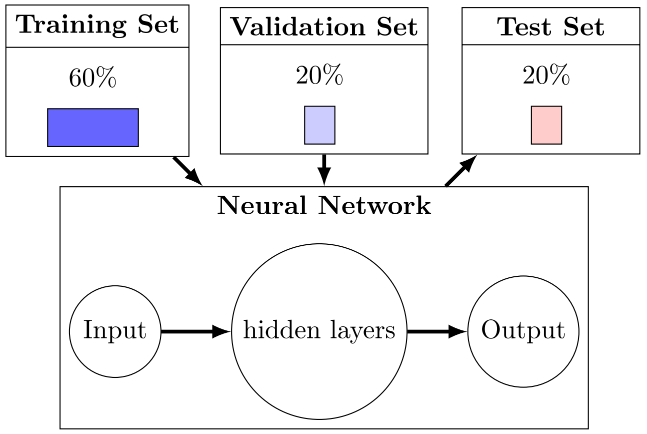


FIG. 2: K-Fold Neural Network Training

Unless otherwise stated, all the training metrics are chosen from the base model (that gives the best test accuracy); model: VGG16, loss function: Categorical-Crossentropy, activation function: softmax, batch size: 16, number of epochs: 50, k-fold split ratio: 4, optimizer: Nadam, and learning rate of 0.001. To preserve the uniformity and eliminate bias, random shuffling is performed in each step. Stepwise plan:

1. Data acquisition from Kaggle and assign into pandas dataframe
2. Train-test split to define test data that is not going to be used in the training
3. Data scaling without augmentation
4. Assign model metrics
5. Obtain pre-trained weights with base model to change final layers for 2 classes
6. Batch normalization to normalize previous layers, dropout to remove possible overfitting, flattened and feeding into dense layers with relu activation followed by dropout
7. Final classification layer with softmax activation function for 2 classes to build up the model
8. kFold training, assign accuracies into history, reset weights between folds, get mean accuracies
9. Plot training accuracies
10. Evaluate test dataset and get test accuracy
11. Obtain confusion matrix
12. Get accuracy, precision, recall and F1-score
13. Assign training metrics into txt file
14. Download training metrics and confusion matrix

III. RESULTS AND DISCUSSION

To assess the performance on unseen test data, images are feed forward into the trained model using a batch size of 1. The mean accuracy values are recorded for comparison. In order to compare the predicted results with the ground truth labels, the confusion matrix of each training is saved as a figure. In Table I, the training metrics are presented for different training types. Each four rows are highlighted with gray colors, representing the optimizer, model, loss function, batch size, activation function, learning rate, epoch size, and k-fold split respectively. The table includes test accuracies, training times, and GPU names for each training types for comparison.

Training Type	Test Acc	Time	GPU
SGD	0.9589	1689.81	Tesla T4
Adam	0.9521	1631.77	Tesla T4
Adagrad	0.9521	1006.62	Tesla T4
Nadam	0.9726	1013.12	Tesla T4
VGG16	0.9726	975.40	Tesla T4
ResNet50	0.8356	900.18	Tesla T4
Resnet152V2	0.9418	1789.57	Tesla T4
Xception	0.9247	1013.12	Tesla T4
Mean Squared Error	0.8459	1018.69	Tesla T4
BinaryCrossEntropy	0.9692	1031.38	Tesla T4
CaterogicalCrossEn.	0.9726	1013.12	Tesla T4
Hinge	0.8527	1017.39	Tesla T4
4	0.9521	1664.31	Tesla T4
8	0.9658	1494.86	Tesla T4
16	0.9726	1013.12	Tesla T4
32	0.9623	998.10	Tesla T4
ReLU	0.5068	492.84	NVIDIA A100
Sigmoid	0.9315	492.48	NVIDIA A100
Softmax	0.9726	1013.12	Tesla T4
Tanh	0.5068	494.34	NVIDIA A100
0.1	0.4932	717.65	NVIDIA A100
0.01	0.9212	1080.792	Tesla T4
0.001	0.9726	1013.12	Tesla T4
0.0001	0.9555	488.85	NVIDIA A100
20	0.9623	423.61	NVIDIA A100
50	0.9726	1013.12	Tesla T4
100	0.9658	997.32	NVIDIA A100
200	0.9486	1966.73	NVIDIA A100
2	0.9041	487.54	Tesla T4
4	0.9726	1013.12	Tesla T4
6	0.9623	1539.24	Tesla T4
8	0.9658	2095.01	Tesla T4

TABLE I: Comparison of training metrics

All the training results are stored in an Excel file for further analysis and graphs. Figures and codes are given in the Github repository.

The training performance of our code, SimpleNNTrain, resulted in a decent test accuracy of 0.914. The validation accuracy per epoch during training is given in FIG. 3. However, for hyperparameter optimization, we choose to use pre-trained weights from available models in the kFoldFinal Jupyter notebook.

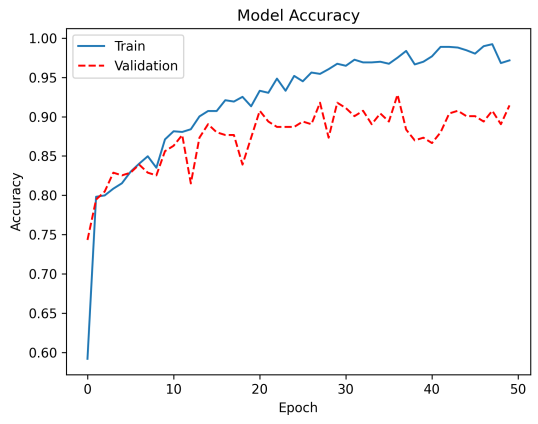


FIG. 3: Training performance of our NN model

To evaluate the training performance using different optimizers, we used different optimizer models. The validation accuracies for each epoch are displayed in FIG. 4. After comparing the results, we determined that the best optimizer model performance was achieved with Nadam. Nadam demonstrated the highest validation accuracy among the optimizer models tested.

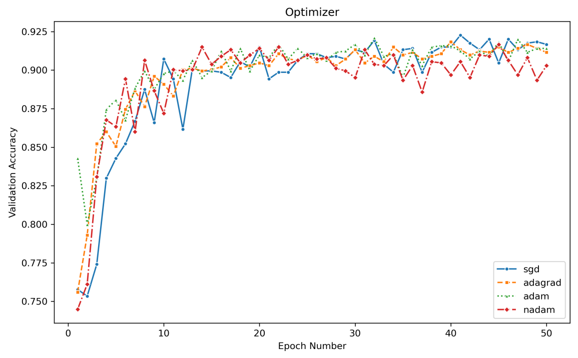


FIG. 4: Model performance on different optimizers

We tested 4 CNN models and evaluated their training performance by recording the validation accuracies for each epoch, as shown in FIG.. 5. Among the models, VGG16 achieved the best performance with the highest validation accuracy.

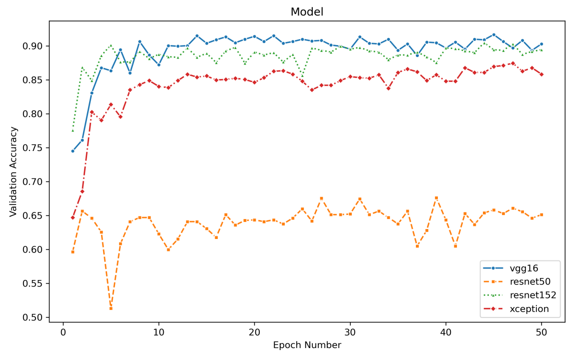


FIG. 5: Model performance on different CNNs

We tested four different loss functions and assessed their impact on the training performance. FIG. 6 shows the validation accuracies for each epoch. Categorical-CrossEntropy resulted in the best model performance, achieving the highest validation accuracy.

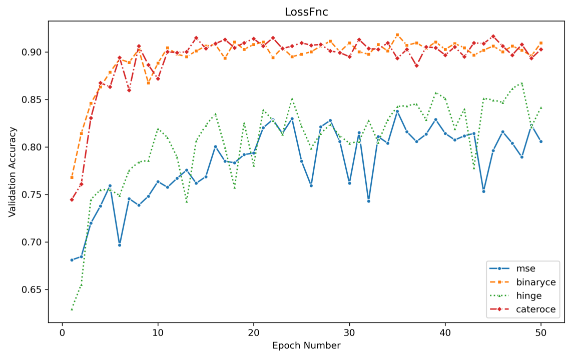


FIG. 6: Model performance on different Loss Functions

We tested different batch sizes (4, 8, 16, and 32) and evaluated their impact on the training performance. FIG. 7 displays the validation accuracies for each epochs. The best model performance was obtained with a batch size of 16 with the highest validation accuracy.

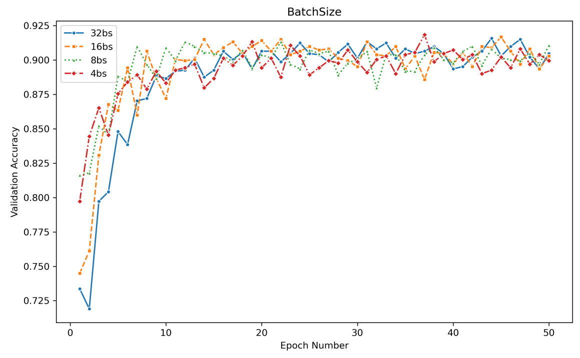


FIG. 7: Model performance on Batch Size

In order to assess the training performance on different activation functions, we utilized ReLU, Sigmoid, Softmax, and Tanh activation functions. The validation accuracies for each epoch are presented in FIG. 8. The ReLU activation function is commonly used in hidden layers, while the softmax function is typically applied to the final output layer for classification purposes. The Tanh activation function is often used in recurrent neural networks [8]. However, due to our implementation of the final layers, the model failed to classify properly using ReLU and Tanh activation functions in the final output layer. The best performance was achieved with the Softmax activation function, which yielded the highest accuracy among the activation functions tested.

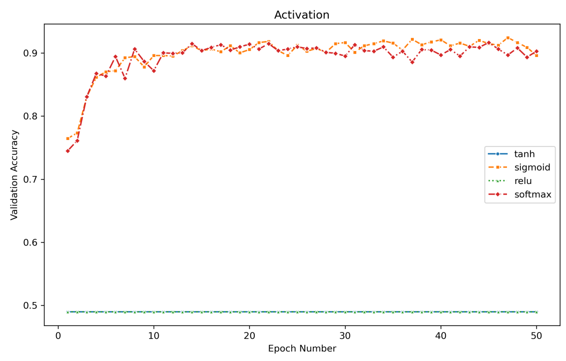


FIG. 8: Model performance on Activation Function

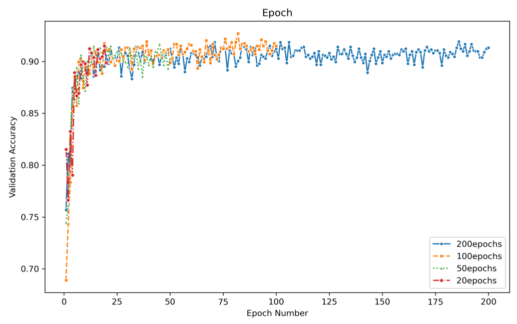


FIG. 10: Model performance on Epoch Numbers

To assess the role of learning rate on the training, we used 0,1, 0.01, 0.001 and 0.0001 to get validation accuracies for each epochs that are given in FIG. 9. Learning rate of 0.1 gives lower validation accuracies in terms of generalization and the main reason might be related with the batch size[9] and the implementation of final layers. The best model performance is obtained from the default learning rate value of Nadam which is 0.001.

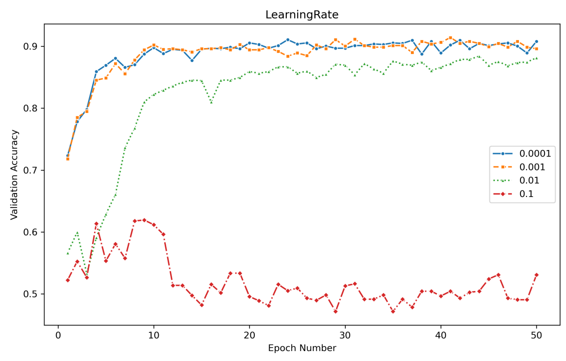


FIG. 9: Model performance on Learning Rate

We examined the effect of varying the number of epochs (20, 50, 100, and 200) on the training performance. FIG. 10 shows the validation accuracies for each epoch. After 50 epochs, the validation accuracies is not increased further, indicating that increasing the number of epochs did not significantly improve the model's performance.

We examined the effect of k-fold split ratios (2, 4, 6, and 8) on the training performance. FIG. 11 displays the validation accuracies for each epoch. The best model performance was obtained with a split ratio of 4.

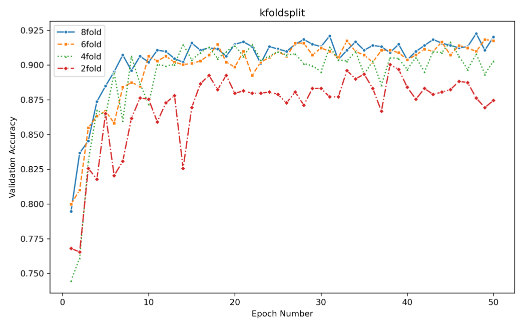


FIG. 11: Model performance on kFold Split Ratio

After evaluating all the training metrics, the base model is selected as with the highest test accuracy. The confusion matrix for this best model is shown in FIG. 12. The overall test accuracy achieved by the model is 0.9726, which can be considered as a good performance.

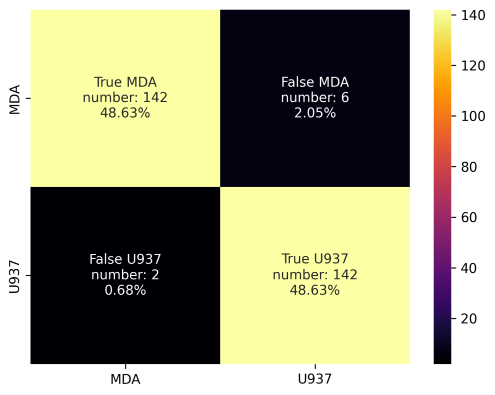


FIG. 12: Confusion matrix of cell classification

IV. CONCLUSION

Classifying cell holograms is a crucial task in prediagnosing cancer cell types. Deep learning-based approaches have replaced traditional reconstruction steps, demonstrating their effectiveness in reconstruction and classification tasks. In this project, we utilize the VGG16 architecture as the base model with specific hyperparameters, including the Nadam optimizer, CategoricalCrossEntropy loss function, Softmax activation function, a batch size of 16, learning rate of 0.001, 50 epochs, and a k-fold split ratio of 4. The VGG16 model achieves the highest test accuracy of 0.9726, showing the power of CNNs in cell hologram classification. These results almost reach up to recent scientific findings and validate our research approach [10].

For the purpose of reproducibility, we have specifically designed our code to record training hyperparameters and accuracies into a text file. The training metrics, including all necessary parameters, are provided as input before compiling the model. Our code and training process are optimized for Google Colab. By organizing the data into pandas dataframes, the training steps become more efficient and faster to execute. As the raw holograms already contain distant features, we did not apply any data augmentation techniques. During the k-fold training step, we record the accuracies for each fold, and the weights from the previous fold are reset. Finally, we calculate the mean values to ensure scalability and uniformity of the results.

In conclusion, our project demonstrates that CNN models can outperform traditional computational models and achieve superior classification results beyond human capability. The next step of our research is to explore how neural network models can be used for direct object detection tasks without the need for cropping or preprocessing steps. This would enable more efficient and accurate analysis of raw images, pushing the boundaries of computer vision in the field of biological cell classification.

BIBLIOGRAPHY

[1] O. Mudanyali, D. Tseng, C. Oh, S. O. Isikman, I. Sencan, W. Bishara, C. Oztoprak, S. Seo, B. Khademhosseini, and A. Ozcan, Lab Chip **10**, 1417 (2010).

[2] T. Latychevskaia and H.-W. Fink, Phys. Rev. Lett. **98**, 233901 (2007).

[3] T. Zeng, Y. Zhu, and E. Y. Lam, Opt. Express **29**, 40572 (2021).

[4] S.-J. Kim, C. Wang, B. Zhao, H. Im, J. Min, H. J. Choi, J. Tadros, N. R. Choi, C. M. Castro, R. Weissleder, and et al., Scientific Reports **8** (2018), 10.1038/s41598-018-35274-x.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," (2015), arXiv:1409.1556 [cs.CV].

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," (2015), arXiv:1512.03385 [cs.CV].

[7] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," (2017), arXiv:1610.02357 [cs.CV].

[8] T. Szandała, *Bio-inspired Neurocomputing*, edited by A. K. Bhoi, P. K. Mallick, C.-M. Liu, and V. E. Balas (Springer Singapore, 2021).

[9] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," (2018), arXiv:1706.02677 [cs.CV].

[10] D. Chen, Z. Wang, K. Chen, Q. Zeng, L. Wang, X. Xu, J. Liang, and X. Chen, Quantitative Imaging in Medicine and Surgery **11**, 4137–4148 (2021).