Transfer Learning for Medical Image Classification

Mahya Ehsanimehr
University of Oulu
Oulu, Finland
mahya.ehsanimehr@student.oulu.fi

GitHub Repository

1. Approach

Transfer learning uses pre-trained models from one machine learning task or dataset to improve performance and generalizability on a related task or dataset [1]. Collecting medical images is challenging due to privacy concerns, the need for expert annotations, and limited access to imaging equipment. Consequently, transfer learning is quite valuable in this field, where labeled data is rare and expensive to collect. In this project, we utilize transfer learning techniques to improve the performance of diabetic retinopathy detection.
Later on this report, we describe the methodology used to address the task of diabetic retinopathy detection. The project includes various tasks; dataset preparation, model selection, fine-tuning, image augmentation, attention mechanisms, ensemble learning and visualizations. In this project, the primary evaluation metric is Cohen's Kappa score.

1.1. Task A: Fine-Tuning Pretrained Models using DeepDRiD

The goal of Task A is to fine-tune pre-trained models on the DeepDRiD dataset to improve performance in detecting diabetic retinopathy. This involves using transfer learning to adapt models trained on ImageNet for the specific task of diabetic retinopathy detection.

1.1.1 Dataset Preparation

The DeepDRiD dataset consists of 2,000 fundus images, with each patient contributing four images (two left-eye and two right-eye). Images were resized to 256x256 pixels, and various data augmentation techniques were applied to enhance generalization and reduce overfitting, including rotation, flipping, cropping, CLAHE (Contrast Limited Adaptive Histogram Equalization).

1.1.2 Training Process

I started my work using the `template_code.py` and applied the same data transformations specified in the code. In addition, I explored other transformation techniques like Gaussian blur and CLAHE, which proved to be well-suited for my work.

The experiments included testing two approaches:

- **Single mode:** Processing one eye image at a time.
- **Dual mode:** Combining features from both eye images of a patient to provide richer context.

Based on my experiments, the dual-image model performed better than the single-image model, which makes sense because using two images from a patient allows the model to learn more comprehensive and meaningful patterns.

For fine-tuning, I adjusted hyperparameters such as learning rate, batch size, step size, gamma, dropout rate, and more. To further reduce overfitting, I added a weight decay of 0.001 to my code. I also experimented with various architectures, primarily working with ResNet18, ResNet34, and DenseNet121.

1.2. Task B: Two stage training with APTOS dataset

Task B focuses on two-stage training to improve model performance. The first stage involves fine-tuning a model on a larger dataset, which I chose APTOS 2019, and then fine-tuning this pre-trained model on the DeepDRiD dataset in the second stage. The aim is to compare this approach with directly fine-tuning on DeepDRiD, as done in task A, to see if it enhances performance.

1.2.1 Fine-tuning models on APTOS dataset

I chose the APTOS dataset for the first part of Task B. For this dataset, the single-image mode was the only feasible approach since the dataset does not include specified left and right eye images for each patient, and the photos are

provided individually without pairing information.

I applied the same settings as in the previous task for this dataset and started fine-tuning different models. Among the models tested, DenseNet121 performed the best.

Since the APTOS dataset is larger than DeepDRiD, I increased the batch size from 24 to 64 to better utilize the available data. To define the dataset, I implemented a new class called *APTOSDataset*. Although I did not use CLAHE for this task, I retained the same transformation settings as in Task A.

1.2.2 Fine-tuning APTOS model on DeepDRiD dataset

The best model from the previous task, DenseNet121, was selected as the base model for further fine-tuning in this task. The fine-tuned DenseNet121 model was loaded and optimized on the DeepDRiD dataset in single mode. Several modifications were explored during this process, including freezing the initial layers of DenseNet121 (e.g., DenseBlock1), adjusting the settings of the learning rate scheduler (e.g., gamma and step size), and reducing the dropout rate to 0.4.

The average Cohen's Kappa score for DenseNet121 in single mode was higher compared to Task A. These findings show that pre-training on the APTOS dataset followed by fine-tuning on DeepDRiD improves the model's overall performance.

1.2.3 Fine-tuning models on DeepDRiD using pre-trained weights

Since the results from the previous step were not particularly high, I decided to fine-tune additional models on the DeepDRiD dataset using pre-trained weights provided by the course TA. This approach got significantly better results, with DenseNet121 achieving the highest Cohen's Kappa score of 0.87.

In this step, I also experimented with tuning hyperparameters and freezing different layers of the models, but these fine-tunings did not lead to notable improvements. Additionally, I explored other pre-trained weights, such as ResNet18 and ResNet34, but ultimately decided to proceed with DenseNet121.

1.3. Task C: Incorporating Attention Mechanisms

The objective of this task is to implement and evaluate the impact of different attention mechanisms on model performance. I implemented spatial, channel, and self-attention mechanisms, using the fine-tuned DenseNet121 model from Task B as the base model. Each attention mechanism was tested multiple times in single mode, with self-attention getting the best average performance.

1.3.1 Types of Attention Mechanisms Implemented:

An attention mechanism is an Encoder-Decoder kind of neural network architecture that allows the model to focus on specific sections of the input while executing a task. It dynamically assigns weights to different elements in the input, indicating their relative importance or relevance [2].

- **Spatial Attention:** Spatial Attention uses a three-step process. First, the input tensor is pooled across channels to create two channels representing max and average pooling. Then these are passed through a convolution layer, followed by batch normalization, and optionally a ReLU activation (which I used in my code). The output is then passed through a sigmoid activation to generate a spatial attention mask. [3] The spatial attention module can be seen in fig 1

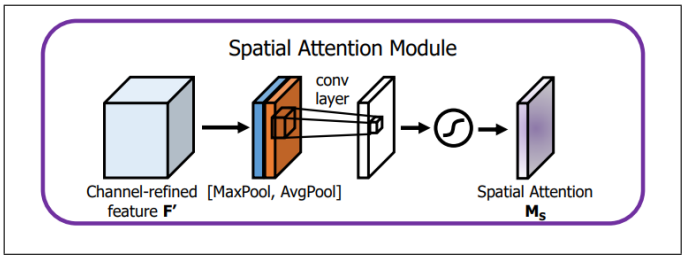

Figure 1. Spatial attention diagram [3]

- **Channel Attention:** Channel attention mechanisms focus on boosting the importance of certain feature channels while suppressing others within the data. [4]

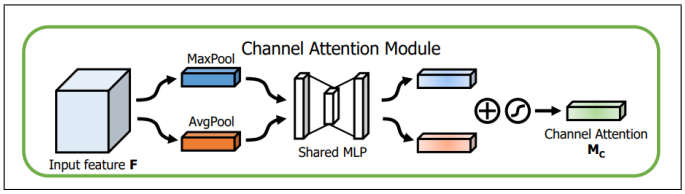

Figure 2. Channel attention diagram [3]

- **Self-attention:** Self-attention transforms the input into query, key, and value vectors. It calculates a weighted sum of the values based on the similarity between the query and key, allowing the model to focus on relevant information.

The average Cohen's Kappa score did not show significant improvement with the addition of attention mechanisms. This could possibly be due to implementation issues or the complexity added by attention mechanisms,

which may not align well with the small dataset size. Since our model is already vulnerable to overfitting due to small dataset, the use of attention mechanisms may have increased the complexity of training without providing benefits.

1.4. Task D: Ensemble Learning and Preprocessing

The goal of Task D is to explore and evaluate ensemble learning techniques and additional preprocessing methods to enhance model performance. Ensemble learning combines the predictions of multiple models to achieve better accuracy and robustness, while preprocessing techniques aim to improve the quality of the input data.

1.4.1 Ensemble Learning

To implement ensemble learning, I explored various methods, including hard voting, soft voting, weighted average, stacking, boosting, and bagging. Each technique combines model predictions differently to improve overall performance.

- **Stacking:** Stacking combines the predictions of multiple models using a higher-level model, called a meta-model, to make the final prediction.
- **Soft Voting**: In soft voting, for each class, it sums the predicted probabilities and predicts the class with the highest sum [5].
- **Hard Voting**: In this method, the class that receives the majority of votes is selected as the final prediction [5].
- **Weighted Average**: In this method, each model receives a weight. Models that perform better are given higher weights, contributing more to the final prediction, while weaker models are given lower weights [6].
- **Bagging**: In bagging, multiple models are trained on different random subsets of the training data with replacement. The average or majority vote of all the distinct models' predictions is used to determine the final prediction. [7]
- **Boosting**: In boosting, models are trained one after another, with each model focusing on correcting the errors made by the previous one. [7]

I used three models pre-trained on DeepDRiD with weights from Task B: DenseNet121, ResNet34, and ResNet18. I then applied the above ensemble methods to combine the predictions of these models.

1.4.2 Preprocessing the Dataset

In this part of the project, I explored additional preprocessing techniques to improve model performance. The preprocessing methods include CLAHE, circular cropping, Ben Graham, and Gaussian blur. These techniques were applied to a dual-image model trained on ResNet18 from Task A. A random seed of 29 was set to ensure consistent results and enable a fair comparison of the preprocessing methods.

During the experiments, I noticed that removing ColorJitter from the transformations improved the model's performance. As a result, I excluded ColorJitter for this task's experiments. However, due to time limitations, I was unable to re-run previous experiments without ColorJitter to validate its impact.

1.5. Task E: Visualizations and Explainable AI

The goal of Task E is to create visualizations and incorporate explainable AI techniques to better understand the model's behavior and predictions. To visualize the loss and accuracy during training and validation, I used Matplotlib library in Python and trained a simple model over 20 epochs.

1.5.1 GradCAM:

For Explainable AI, as said in the project instructions, I used GradCAM. Grad-CAM (Gradient-weighted Class Activation Mapping) is a method used to make CNNs more interpretable by visually explaining their decisions. It shows the regions in an input image that contribute most to the model's predictions. This is useful in domains like medical imaging [8].

2. Experiment and Discussion

2.1. Datasets Details

2.1.1 DeepDRID Dataset [9]

In this project, we used the DeepDRiD dataset, which contains 2,000 images from patients with diverse backgrounds. Each patient has a total of four images: two for the left eye and two for the right eye. DeepDRID consists of 5 classes, as shown in Table 1. Since the dataset is relatively small, training a model without overfitting presents a significant challenge.

description	label
No apparent retinopathy	0
Mild – NPDR	1
Moderate – NPDR	2
Severe – NPDR	3
PDR	4

Table 1. DeepDRID classes

2.1.2 APTOS Dataset [10]

The APTOS 2019 Blindness Detection dataset contains 3,662 fundus images collected from rural areas of India. The images were taken under varying conditions and labeled by trained doctors. The dataset is categorized into five classes the same as DeepDRID dataset: no diabetic retinopathy, mild, moderate, severe, and proliferative diabetic retinopathy.
This dataset is very similar to the DeepDRiD dataset in terms of concepts and images. It is divided into training, testing, and validation sets. The features include *id_code*, which identifies each sample, and *diagnosis*, a label ranging from 0 to 4 that indicates the level of diabetic retinopathy.

2.2. Task A

I run each model a couple of times for 20 epochs. The best Cohen's Kappa score was the result of resnet18 model in dual mode. The learning rate was set to 1e-4, weight decay 0.001, batch size 24, and dropout rate of 0.5. The attempts and details are shown in tables 2 for single models and 3 for dual models.

As shown in tables, DenseNet121 in single mode and ResNet18 in dual mode achieved the highest Cohen's Kappa scores. However, the accuracy of these models remained relatively low, ranging between 0.65 and 0.7, which shows potential signs of overfitting.

Model	Average Test	Best Test	Best Val
ResNet18	0.7793	0.7966	0.8365
ResNet34	0.7968	0.7991	0.8293
Densenet121	0.7881	**0.8015**	0.8509

Table 2. Task A: experiments details (single mode)

Model	Average Test	Best Test	Best Val
ResNet18	0.8139	**0.8346**	0.8735
ResNet34	0.7675	0.8100	0.8540
Densenet121	0.7979	0.8174	0.7942

Table 3. Task A: experiments details (dual mode)

2.3. Task B

2.3.1 Fine-tuning models on APTOS dataset

I implemented a new class called *APTOSDataset* for defining the dataset. To fine-tune the models on APTOS, I implemented several strategies. I used an enhanced model architecture by incorporating batch normalization and global pooling. Additionally, I experimented with various pretrained ImageNet architectures, including MobileNet and VGG16, to explore their effectiveness for the task. However, using the same settings as previous tasks helped me fine-tune an acceptable model on APTOS. The results of experiments are shown in table 4. Densenet121 achieved the highest Cohen's Kappa score on validation set.

Model	Best Val
ResNet34	0.8925
Densenet121	**0.9070**

Table 4. Task B: Results on fine-tuning models on APTOS dataset

2.3.2 Fine-tuning APTOS model on DeepDRiD dataset

In this section, I used the DenseNet121 model from the previous task, which was pre-trained on the APTOS dataset, and fine-tuned it further on the DeepDRiD dataset. The results are presented in table 5. As shown, the average Cohen's Kappa score for the single DenseNet121 model improved compared to Task A, where the score was 0.7881. This demonstrates that pretraining on APTOS followed by fine-tuning on DeepDRiD leads to better performance.

Model	Average Test	Best Test	Best Val
Densenet121	0.8050	0.8374	0.8551

Table 5. Task B: fine-tuning Densenet121 on DeepDRiD

2.3.3 Fine-tuning models on DeepDRiD using pre-trained weights

For further experiments, I used pre-trained weights and fine-tuned them on the DeepDRiD dataset. The results are summarized in Table 6. Among the models, DenseNet121 in

single mode achieved a Cohen's Kappa score of 0.8738, which is a significant improvement.

Model	Average Test	Best Test
Densenet121	0.8688	**0.8738**
Resnet34	0.8543	0.8558
Resnet18	0.8403	0.8462

Table 6. Task B: fine-tuning on DeepDRiD using pre-trained weights

2.4. Task C

The overall result for attention mechanisms is shown in table 7. I used fine-tuned Densenet121 from task b as my base model. I run each of the attentions a couple of times in single mode and self-attention performed better on an average basis. I used kernel 7 and 11 for Spatial attention. I also used this website [3] as a reference to implement attentions. Among these attention mechanisms, self attention outperformed with a slightly better score than the rest.

Attention	Average Test	Best Test
Spatial (kernel=7)	0.8222	0.8376
Spatial (kernel=11)	0.8229	0.8517
Channel	0.8227	0.8631
Self-attention	**0.8370**	0.8511

Table 7. Task C: Different Attention Mechanisms

2.5. Task D

In this task, I used the models that have been trained on DeepDRiD using pre-trained weights on task b 6, to build ensemble models and compare their performances.

2.5.1 Implementing Ensemble Learning

To implemenet ensebmle learning, I used many technique including hard voting, soft voting, weighted average, stacking, boosting and bagging. I set 40 as the random seed to get consistent results and fair comparisons 8. Boosting and bagging required retraining the models. I conducted these experiments, and the results are presented in table 9 after a couple of trainings.

Model	Best Test
Soft Voting	0.8644
Hard Voting	0.8627
Stacking	0.8649
Weighted Avg	**0.8712**

Table 8. Task D: Ensemble Methods Results

Model	Best Test	Test (avg)
Boosting	0.8151	0.8345
Bagging	0.8228	0.8351

Table 9. Task D: Ensemble Methods Resultss

2.5.2 Different Preprocessing Techniques

In this task, I applied several preprocessing techniques to a dual mode model trained on Resnet18 from task A. I also used 29 as random seed to get consistent results.

I also noticed that removing ColorJitter from transformations enhances the performance of model. The overall results of pre-processing is shown in table 10.

CLAHE combined with circular cropping achieved the best Cohen's Kappa score. I believe circular cropping is effective here, because this project focuses on retinal images, which are circular in shape. By applying circular cropping, the model can focus more on the important regions of the image, reducing distractions from unimportant areas.

Model	Test Score
no pre-processing	0.8284
CLAHE + Circular cropping	**0.8501**
CLAHE + Circular cropping + Ben Graham	0.7840
CLAHE + Circular cropping + Guassian Blur	0.8374

Table 10. Task D: Preprocessing Techniques

Figure 3 illustrates the output of images after applying CLAHE and circular cropping.

2.5.3 Task E

The loss and accuracy for both training and validation for a model on 20 epochs, can be seen in fig 4. As indicated, the model suffers from overfitting which is mostly due to small dataset.

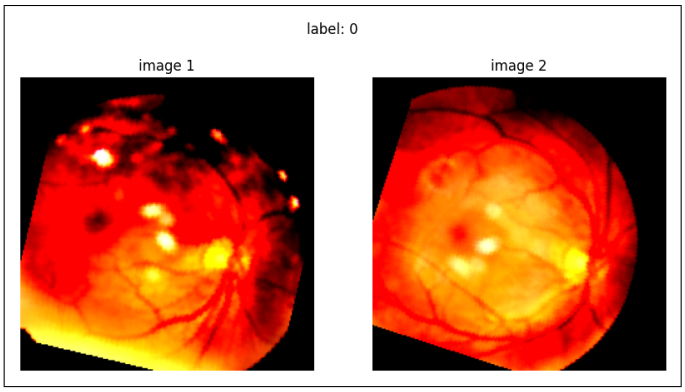

Figure 3. Example of caption.

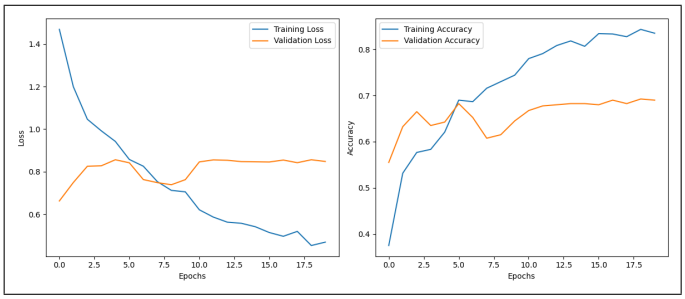

Figure 4. Task E: Loss and Accuracy Visualization

An example of GradCAM output shown in figure 5, highlights the regions of the retina image that the model found most important for its prediction. Warmer colors (reds and yellows) indicate areas of high relevance, likely corresponding to key features or abnormalities used by the model for classification.

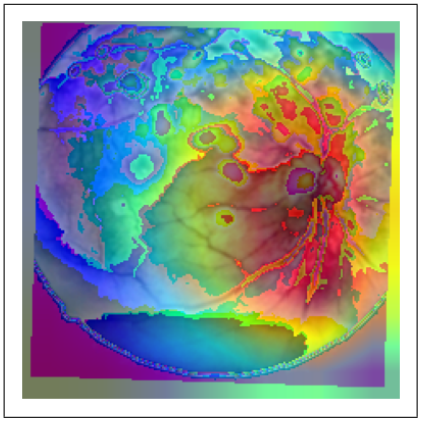

Figure 5. Example of caption.

3. Conclusion

This project applies transfer learning to improve diabetic retinopathy detection using the DeepDRiD and APTOS datasets. We noticed that fine-tuning pre-trained models, such as DenseNet121, along with a two-stage training process, improved performance significantly. Attention mechanisms were explored but unfortunately offered limited improvements due to the small dataset. Ensemble methods, like weighted averaging, and preprocessing techniques, such as CLAHE with circular cropping, further enhanced results. Grad-CAM was used for explainability, highlighting key regions in images that influenced predictions. The project shows the potentials of transfer learning and preprocessing while addressing challenges like overfitting. Ultimately, in medical imaging projects where we are facing limited datasets and ethical issues, transfer learning techniques prove to be highly effective and valuable.

References

[1] IBM. Transfer learning: Accelerating ai development with pre-trained models. https://www.ibm.com/think/topics/transfer-learning, 2023. Accessed: 2024-12-31. 1

[2] GeeksforGeeks. Ml — attention mechanism. https://www.geeksforgeeks.org/ml-attention-mechanism/, 2023. Accessed: 2024-12-31. 2

[3] DigitalOcean Community. Attention mechanisms in computer vision — cbam. https://www.digitalocean.com/community/tutorials/attention-mechanisms-in-computer-vision-cbam, 2023. Accessed: 2024-12-31. 2, 5

[4] BeeiLab. Channel attention mechanisms in deep learning for geospatial tasks. https://medium.com/@beeilab.yt/channel-attention-mechanisms-in-deep-learning-for-geospatial-tasks-9ecd2da42ddc, 2023. Accessed: 2024-12-31. 2

[5] GeeksforGeeks. Voting in machine learning. https://www.geeksforgeeks.org/voting-in-machine-learning/, 2023. Accessed: 2024-12-31. 3

[6] Jason Brownlee. Weighted average ensemble with python. https://machinelearningmastery.com/weighted-average-ensemble-with-python/, 2023. Accessed: 2024-12-31. 3

[7] GeeksforGeeks. Ensemble methods and wisdom of the crowd. https://www.geeksforgeeks.org/ensemble-methods-and-wisdom-of-the-crowd/, 2023. Accessed: 2024-12-31. 3

[8] DataScientest. What is the grad-cam method? https://datascientest.com/en/what-is-the-grad-cam-method, 2023. Accessed: 2024-12-31. 3

[9] DeepDRiD Challenge. DeepDRiD Dataset. https://www.sciencedirect.com/science/article/

pii/S2666389922001040, 2022. Accessed: 2024-12-26. 3

[10] Aravind Eye Hospital. APTOS 2019 Blindness Detection Dataset. https://www.kaggle.com/datasets/mariaherrerot/aptos2019, 2019. Accessed: 2024-12-26. 4