Deep Learning Mini-Project: Breast Cancer Classification using CNNs and Cloud Deployment

Student: Ayachi Hazem

Date: April 17, 2025

Abstract

This report details a mini-project focused on applying deep learning techniques, specifically Convolutional Neural Networks (CNNs), to the problem of breast cancer classification using the BreaKHis histology image dataset. The project followed a structured approach involving the creation of a baseline CNN, performance enhancement using data augmentation, and leveraging transfer learning with a pre-trained DenseNet121 model incorporating feature fusion. Each model was trained and evaluated based on metrics including accuracy, precision, recall, and F1-score. Finally, the best-performing model was packaged into a web API using Flask, containerized using Docker, and deployed to Google Cloud Platform (GCP) via Cloud Run, making the classification service accessible via a public URL. The report outlines the methodology, choices made at each stage, evaluation results comparing the different approaches, and the successful deployment process.

1. Introduction

The objective of this mini-project was to design, evaluate, and deploy a Convolutional Neural Network (CNN) architecture for a medical image classification task, as per the course requirements. The chosen application is the classification of breast cancer histology images as either benign or malignant, utilizing the publicly available BreaKHis dataset. Early and accurate diagnosis of breast cancer is crucial for effective treatment, and computer-aided diagnosis systems using deep learning show significant promise in assisting pathologists.

This project involved several key phases:

1. Establishing a performance baseline with a custom-built CNN.
2. Investigating the impact of data augmentation techniques on the baseline model.
3. Implementing and evaluating a transfer learning approach using a pre-trained DenseNet121 model.
4. Deploying the final model as a web service on Google Cloud Platform (GCP).

This report details the dataset used, preprocessing steps, the architectures implemented, the results obtained, the deployment procedure, and discusses the challenges faced and choices made throughout the project.

2. Dataset and Preprocessing

- **Dataset:** The BreaKHis v1 dataset was used for this project. It contains microscopic biopsy images of breast tumor tissue, captured at different magnification factors. The images are categorized into benign and malignant classes, with further subtypes available (though this project primarily focused on the binary classification task for the core comparison steps). The dataset comprises approximately ~7900 PNG images.

- **Data Loading:** Images were loaded using Python libraries (glob). Paths were parsed to determine the binary label (0 for Benign, 1 for Malignant).
- **Preprocessing:** A consistent preprocessing pipeline was applied to all images:
    - Images were decoded into 3-channel (RGB) format.
    - Images were resized to a uniform input dimension of (128, 128) pixels using tf.image.resize.
    - Pixel values were normalized to the range [0, 1] by dividing by 255.0.
- **Data Splitting:** The dataset was split into training (60%), validation (20%), and testing (20%) sets. The split was performed using sklearn.model_selection.train_test_split with stratification enabled (stratify=labels) to maintain the original class distribution in each subset. The final sizes were: Training: 4745 images, Validation: 1582 images, Test: 1582 images.
- **Input Pipeline:** tf.data.Dataset was used to create efficient input pipelines for training, validation, and testing, including shuffling, batching (batch size of 64), and prefetching. Labels were one-hot encoded for use with categorical cross-entropy loss.

3. Methodology

The project systematically explored three different CNN approaches as required.

3.1. Baseline CNN Architecture

- **Objective:** To establish a baseline performance using a relatively simple, custom-built CNN architecture.
- **Architecture:** A sequential Keras model was constructed consisting of:
    - Input Layer: Expecting shape (128, 128, 3).
    - Convolutional Blocks: Three blocks of Conv2D (with 32, 64, and 128 filters respectively, kernel size 3x3, ReLU activation) followed by MaxPooling2D (2x2 pool size).
    - Flatten Layer: To convert the 2D feature maps into a 1D vector.
    - Dense Layers: A Dense layer with 128 units (ReLU activation), followed by a Dropout layer (rate 0.5) for regularization.
    - Output Layer: A Dense layer with 2 units (one for each class) and softmax activation for probability output.
- **Training:** The model was compiled using the Adam optimizer (learning rate 0.001), categorical_crossentropy loss function, and accuracy as the evaluation metric. Training was performed on the **non-augmented** training dataset for 20 epochs, using EarlyStopping (patience=10) and ReduceLROnPlateau (patience=5) callbacks to prevent overfitting and adjust the learning rate.

3.2. Data Augmentation

- **Objective:** To improve the generalization and performance of the baseline model by artificially expanding the training dataset.
- **Techniques:** A custom augmentation function (augment_image_tf) was implemented using tensorflow.image operations. For each input image, the following augmentations were applied:
    - 90-degree rotation followed by a horizontal flip.
    - 180-degree rotation followed by a horizontal flip. The original image was also kept, effectively tripling the number of samples generated from the original training set during data loading.

- **Rationale:** Augmentation helps expose the model to variations in orientation and mirroring, making it more robust to such changes in unseen images and mitigating overfitting on the limited original dataset.
- **Application:** The augmentation function was integrated into the `create_tf_dataset` function and applied only to the training data (`augment=True`).
- **Retraining:** The *exact same baseline CNN architecture* from section 3.1 was re-compiled and retrained using the **augmented** training dataset, keeping other training parameters (optimizer, loss, epochs, callbacks) consistent for fair comparison.

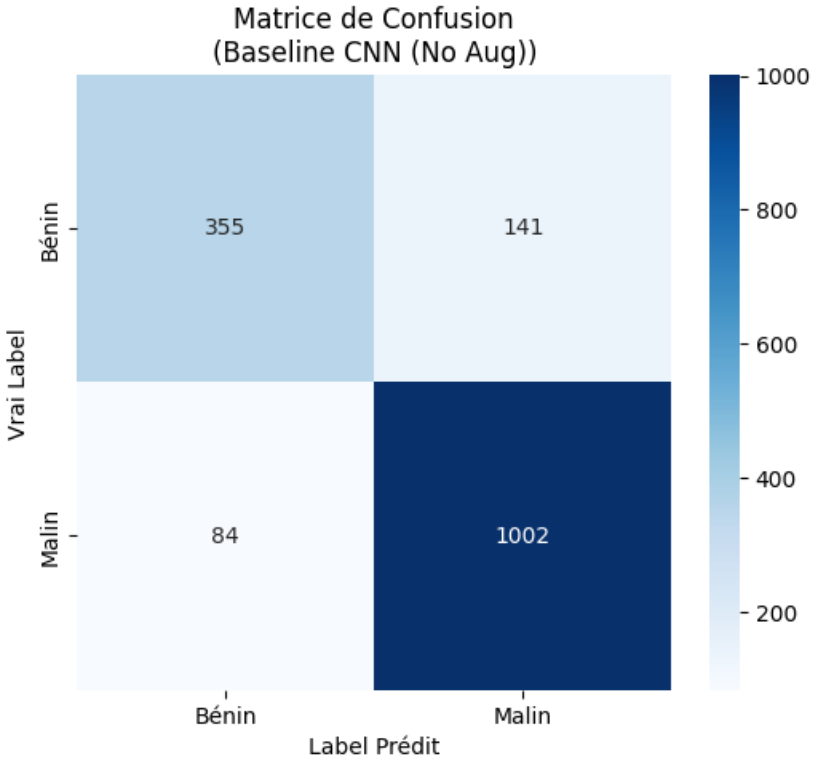
3.3. Transfer Learning

- **Objective:** To leverage knowledge learned from a large-scale dataset (ImageNet) by using a pre-trained model.
- **Chosen Model:** DenseNet121, pre-trained on ImageNet, was selected via `tf.keras.applications.DenseNet121`. DenseNet architectures are known for efficient feature propagation and gradient flow.
- **Implementation Strategy:**
    - **Fine-tuning:** The pre-trained weights of DenseNet121 were used, and the base model's layers were made trainable (`base_model.trainable = True`) to allow adjustment to the specific features of the medical histology images.
    - **Feature Fusion:** Instead of just using the final output of DenseNet121, features were extracted from three intermediate convolutional blocks (`conv3_block12_concat`, `conv4_block24_concat`, `conv5_block16_concat`). Each intermediate output was passed through a Global Average Pooling (GAP) layer, a Dense layer (64 units, ReLU), and Batch Normalization. These processed features from different depths were then concatenated.
    - **Classification Head:** The fused features were passed through a final block (Dense(16, ReLU), Batch Normalization, Dropout(0.45)) before the output layer (Dense(2, softmax)).
- **Rationale:** Fine-tuning allows the model to adapt better to the target domain than pure feature extraction. Feature fusion from multiple layers allows the model to combine low-level and high-level features, potentially improving classification accuracy.
- **Training:** The transfer learning model was trained on the **augmented** dataset. A lower learning rate (e.g., 1e-5) was used for the Adam optimizer, which is common practice during fine-tuning. `EarlyStopping` and `ReduceLROnPlateau` callbacks were also used.

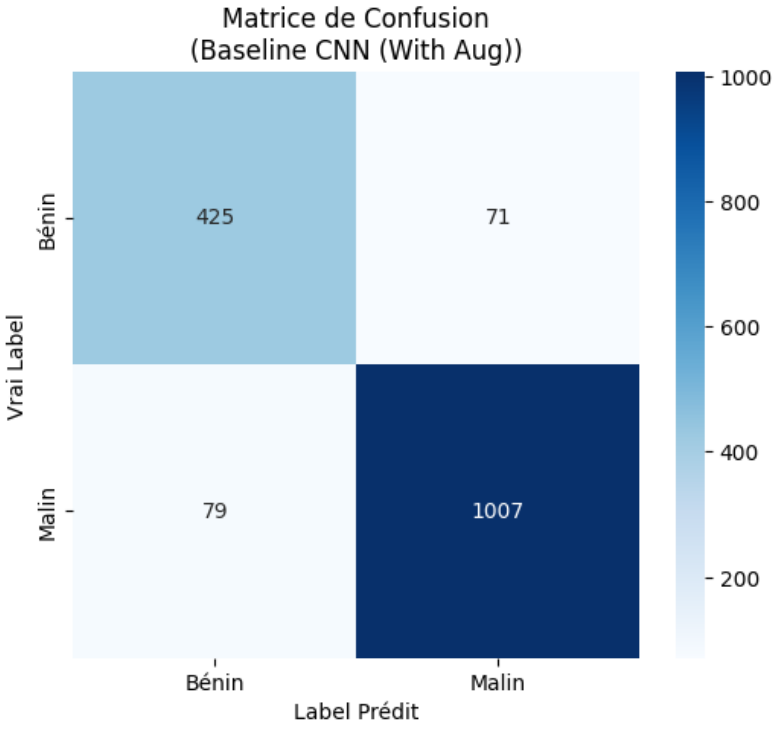
4. Evaluation and Results

Models were evaluated on the unseen test set using accuracy, loss, precision, recall, F1-score (binary average), and confusion matrices. Training/validation curves were plotted to monitor learning progress.

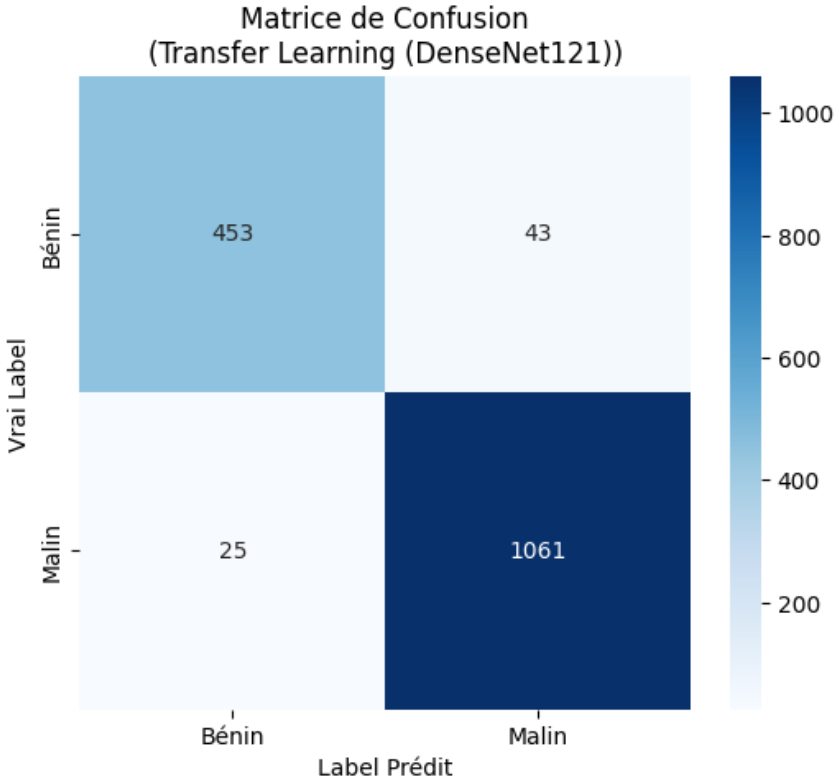
- **4.1. Baseline CNN Results (No Augmentation):**
    - The baseline model achieved an accuracy of **[0.8578 %]** and a loss of **[0.3527]** on the test set.
    - Precision: **[0.8766]**, Recall: **[0.9227]**, F1-Score: **[0.8991]**.
    - *Briefly describe performance, e.g., "The model showed some learning capability but performance was limited..."*



- **4.2. Impact of Data Augmentation:**
  - Training the same baseline architecture with data augmentation resulted in an improved test accuracy of **[0.9052 %]** and a loss of **[0.2412]**.
  - Precision: **[0.9341]**, Recall: **[0.9273]**, F1-Score: **[0.9307]**.
  - Comparison: Accuracy improved by **[0.0474] %** compared to the non-augmented baseline.
  - *Discussion: "Data augmentation significantly boosted performance, likely by improving the model's robustness and reducing overfitting..."*



- **4.3. Transfer Learning Results (DenseNet121 + Augmentation):**
    - The fine-tuned DenseNet121 model with feature fusion achieved the highest performance, with a test accuracy of **[0.9570%]** and a loss of **[0.4597]**.
    - Precision: **[0.9611]**, Recall: **[0.9770]**, F1-Score: **[0.9689]**.
    - Comparison: This represents an improvement of **[0.0518] %** over the augmented baseline CNN.
    - *Discussion: "Transfer learning provided a substantial performance gain, demonstrating the effectiveness of leveraging pre-trained weights even across different domains. The fine-tuning approach and feature fusion likely contributed to this success."*



- (Optional Summary Table):

Model	Test Accuracy	Test Loss	F1-Score
Baseline CNN (No Aug)	**0.8578**	**0.3527**	**0.8991**
Baseline CNN (With Aug)	**0.9052**	**0.2412**	**0.9307**
Transfer Learning (Aug)	**0.9570**	**0.4597**	**0.9689**

5. Cloud Deployment

The best-performing model (Transfer Learning DenseNet121) was deployed to Google Cloud Platform (GCP) to make it accessible as an API.

- **5.1. API Development:**

    - A web application was developed using the **Flask** framework in Python (main.py).
    - It loads the saved Keras model (transfer_learning_model.keras).
    - It exposes two endpoints:
        - / (GET): A health check endpoint returning service status.
        - /predict (POST): Accepts an image file upload (multipart/form-data), performs the necessary preprocessing (resizing, normalization) using Pillow and TensorFlow, runs model.predict(), and returns the predicted label (e.g., "Benign" or "Malignant") and confidence score in JSON format.
- **5.2. Containerization (Docker):**
    - The application, along with its dependencies (TensorFlow, Flask, Gunicorn, etc., listed in requirements.txt) and the model file, was packaged into a **Docker** container.
    - A Dockerfile was created, starting from a python:3.9-slim base image, installing dependencies via pip, copying the application code and model, exposing port 8080, and setting the entry point to run the Flask app using the **Gunicorn** WSGI server (CMD ["gunicorn", ...]) for production readiness.
- **5.3. Deployment to Cloud Run:**
    - **Google Cloud Run** was chosen as the deployment platform for its serverless nature, automatic scaling (including to zero), and direct support for container deployment.
    - The built Docker image was pushed to **Google Artifact Registry**.
    - The service was deployed using the gcloud run deploy command, specifying the image path from Artifact Registry, the region (europe-west1), target port (8080), desired CPU/Memory resources (1 CPU, 2Gi RAM), and allowing unauthenticated access for testing purposes during the project.
- **5.4. Testing:**
    - The deployed service URL (https://breast-cancer-54106035782.europe-west1.run.app) was tested.
    - The health check endpoint (/) was verified using a web browser.
    - The prediction endpoint (/predict) was tested by sending POST requests containing image files using tools like curl and Postman, confirming successful prediction responses in JSON format.

```
C:\Users\ayach>curl -X POST -F "file=@C:\Users\ayach\tf_env\BreaKHis_v1\histology_slides\breast\malignant\SOB\ductal_carcinoma\SOB_M_DC_14-2980\40X\SOB_M_DC
-14-2980-40-015.png" https://breast-cancer-54106035782.europe-west1.run.app/predict
{"confidence":0.8833152055740356,"predicted_label":"Malin","raw_prediction":[0.11668472737073898,0.8833152055740356]}
```

6. Discussion and Challenges

- **Key Findings:** Transfer learning with fine-tuning significantly outperformed the custom baseline CNN, even when the baseline was enhanced with data augmentation. Data augmentation provided a clear benefit over training without it. The deployed Cloud Run service successfully served predictions based on the trained model.
- **Choices Made:** DenseNet121 was chosen for its strong performance on ImageNet and its parameter efficiency. The specific data augmentations (rotations, flips) were selected as standard techniques for image data. Flask was chosen for its simplicity in creating the API. Cloud Run was selected over GKE or AI Platform for its balance of ease of use and container support, fitting

the scope of a mini-project. Gunicorn was used as the production WSGI server.

- **Challenges Encountered:**
  - Initial deployment attempts failed due to the `gunicorn` executable not being found within the container; this was resolved by ensuring `gunicorn` was correctly listed in `requirements.txt` and installed during the Docker build.
  - Understanding the interaction between `tf.data.Dataset`, `.repeat()`, and the `steps_per_epoch` argument in `model.fit()` required careful attention to avoid infinite loops or premature training interruption.
  - Selecting appropriate CPU/Memory resources for the Cloud Run service needed consideration, as TensorFlow models can be memory-intensive.
- **Future Work:** Explore other pre-trained architectures (e.g., ResNet, EfficientNet). Implement more sophisticated data augmentation techniques. Experiment with different fine-tuning strategies. Deploy using Kubernetes (GKE) for more complex scaling scenarios. Implement proper authentication for the API endpoint. Further optimize the model for inference speed.

7. Conclusion

This mini-project successfully demonstrated the process of building, evaluating, and comparing different CNN approaches for breast cancer histology image classification. The effectiveness of data augmentation and transfer learning was clearly shown. Furthermore, the project culminated in the successful deployment of the trained model as a containerized web service on Google Cloud Run, fulfilling all the requirements outlined in the project description. The process provided valuable hands-on experience in deep learning model development, evaluation, and cloud deployment practices.

8. References

- BreaKHis Dataset: https://www.kaggle.com/datasets/ambarish/breakhis?select=BreaKHis_v1
- TensorFlow: https://www.tensorflow.org/
- Keras: https://keras.io/
- Flask: https://flask.palletsprojects.com/
- Docker: https://www.docker.com/