Table of Contents

Introduction

The objective of our project is to leverage Python programming and apply scientific concepts to demonstrate problem-solving skills. Additionally, we were assigned the task of developing a deep learning-based medical application. Specifically, our group focused on creating a deep learning model for skin cancer image classification. During the initial phase, we divided our responsibilities among team members. Each of us delved into different aspects: one explored deep learning techniques and convolutional neural networks, another researched relevant dataset, and a third member investigated existing source code from GitHub. With this knowledge, we began experimenting with the existing codebase, making modifications to enhance the model's accuracy and contribute meaningfully to the project.

Libraries

The libraries used for the deep learning models are, EfficientNetB5, ReduceLROnPlateau, BatchNormalization, AdamW, train_test_split, classification_report, Albumentations (Data Augmentation), keras tuner import HyperModelkeras_tuner and keras_tuner.tuners import RandomSearch.

Source code and datasets

The source code was acquired from a GitHub repository associated with a Kaggle competition. The decision to select this particular codebase was driven by its alignment with the chosen topic—skin cancer classification. Subsequently, we embarked on a process of modification to enhance accuracy and expand the available datasets. Our dataset collection involved multiple sources, including Kaggle and publicly available datasets on Amazon Web Services (AWS). Specifically, we focused on melanoma, nevus, and seborrheic keratosis images. To ensure balanced representation, we meticulously selected the dataset, striving for equal numbers of images across these different skin cancer types to avoid the model from overfitting.

Enhancements and Modifications

Original Source Code	Aspects	Our Project
▪ Import glob ▪ Import InceptionV3 ▪ Earlystopping, ModelCheckPoint ▪ Dense, Dropout, GlobalAveragePooling2D ▪ Sequential ▪ Tqdm ▪ Load_dataset	**Libraries**	• Addition in importing numpy, os, cv2, EfficientNetB5, ReduceLROnPlateau, BatchNormalization, AdamW, train_test_split, classification_report • Albumentations (Data Augmentation) • keras_tuner import HyperModel • keras_tuner.tuners import RandomSearch •
None	**Data Augmentation**	• Defining the Augmentation pipeline • Convert to tensor with the correct shape • Generate augmented images
Loading entire dataset into memory	**Loading Datasets**	• Load paths and targets, reducing initial memory usage • Efficient data loading and better memory
• The original code uses InceptionV3 • transfer_model = InceptionV3(include_top=False, weights="imagenet")	**Feature Extraction**	Switched to EfficientNetB5 for potentially better feature extraction. Implemented gradual unfreezing of layers for fine-tuning.
No Hyperparameter tuning	**Hyperparameter Tuning**	• Added hyperparameter tuning using Keras Tuner • Automated search for the best hyperparameters, potentially leading to better model performance.

checkpointer = ModelCheckpoint(filepath="saved_models_weights_checkpointer/weights.best.model.hdf5", verbose=1, save_best_only=True) early_stopping = EarlyStopping(monitor="val_loss", patience=5, verbose=1, mode="min")	**Callbacks and Training Configuration**	• Added **ReduceLROnPlateau** to reduce the learning rate when the validation loss plateaus, which can help in fine-tuning. • Added TensorBoard for better visualization and monitoring of the training process.
Does not handle class imbalance	**Handling Class Imbalance**	• Computed class weights to handle class imbalance. • Handling class imbalance improves model performance, especially in datasets with skewed class distributions.
CNN_model = Sequential() CNN_model.add(GlobalAveragePooling2D(input_shape=train_data.shape[1:])) CNN_model.add(Dropout(0.2)) CNN_model.add(Dense(1024, activation="relu")) CNN_model.add(Dropout(0.2)) CNN_model.add(Dense(3, activation="softmax"))	**Model Architecture**	• Added more layers and dropout rates, and, used BatchNormalization for better regularization and performance. • Automated selection of hyperparameters for the dense layers and dropout rates using Keras Tuner
CNN_model.fit(train_data, train_targets, validation_data=(valid_data, valid_targets), epochs=60, batch_size=200, callbacks=[checkpointer, early_stopping], verbose=1,)	**Training and Evaluation**	• Split the training data for validation, added class weights, and used the best hyperparameters for training. • Enhanced the callbacks and reduced the batch size for better learning dynamics.
No evaluation	**Evaluation**	• Added evaluation of the model on the validation data and

		prints the classification report. • Added evaluation step to provide detailed performance metrics using classification_report from sklearn.
No web application for cancer skin detection	**Web Application**	Create a web (html) that run under flask in Python Image can be uploaded onto the web and results will be generated

How to run the program

Firstly, the training and testing code of the model is in train_model.py and test_model.py. To verify, run the get_result.py. Run the training of the model then followed by the testing of the model. The cancer_detection.py file can be run to detect the cancer image. The way to run this file is to include the path of the image. Example to run the code is by using this syntax – (python cancer_detection.py -i "C:/Users/naash/Documents/programming/skin-cancer-classification-main/cancer images/test_melanoma.jpg").

The program will be able to detect one of the three different skin diseases (melanoma, nevus, seborrheic keratosis). The skin diseases images are placed in the test, train and valid file. In addition, the web browser can be used to detect the cancer image as well. To do this, run the app.py file then copy paste this link http://127.0.0.1:5000/ on a web browser. The index.html serve as the main landing page for the web application, while result.html contains code that display the results of the image classification. Lastly, The Data Augumentation.ipynb shows how data augmentation works, and some sample images generated.

Project limitations

Dataset size and quality

When developing machine learning models, the size and quality of the dataset play critical roles. Firstly, inadequate dataset size. When the dataset is not large enough it can hinder model performance. Insufficient data may lead to overfitting, where the model learns noise or specific examples rather than generalizable patterns. Secondly, the quality of the dataset matters significantly. High-resolution images with accurate annotations are essential for robust training.

Conversely, low-resolution images or poorly annotated data can introduce noise and negatively impact model accuracy. Therefore, meticulous curation and augmentation of datasets are crucial to achieving reliable results in skin cancer classification.

Data Augmentation

When applying data augmentation methods, it is crucial to avoid excessive data augmentation as it may lead to overfitting or poor model generalization. Additionally, not all augmentation techniques are appropriate for every dataset. The augmentation may distort the data in ways that are unhelpful or misleading, thus careful consideration is needed when selecting these techniques.

Model Architecture

EfficientNetB5 demands substantial data and computational power for effective training. However, if the dataset is not large or diverse enough, overfitting becomes a concern.

Class Imbalance

Class imbalance can lead to model bias, favouring the majority class during training. Addressing this imbalance through techniques like oversampling or weighted loss functions is essential. Secondly, while pre-trained models like EfficientNetB5 offer a robust starting point due to their prior training on ImageNet, we must recognize that the features learned may not perfectly align with the nuances required for skin cancer detection.

Time Consuming

As the model is improved, it takes more time to train the model. Thus, debugging the code would be time consuming and the whole process will consume a lot of time.

Conclusion

In conclusion, the project was successful as we significantly enhanced the original source code and introduced several key improvements. We transitioned from InceptionV3 to EfficientNetB5 for superior feature extraction, integrated hyperparameter tuning, and implemented techniques to address class imbalance. Additionally, we added new callbacks and training configurations to optimize learning dynamics and performance monitoring. The development of a web application using Flask further enabled user interaction by allowing image uploads for classification. These modifications not only improved the model's accuracy but also deepened our understanding of the complexities of skin cancer detection. This project lays a solid foundation for future advancements, contributing to both the fields of medicine and computer science.