Skin Cancer Classification using Mobilenet Architecture

Deep Learning (CS F425) – 2nd Semester 2024–25

By

Nishant Pradhan (2023A7PS0030P)
Shantnu Sarda (2023A7PS0017P)
Arjun Ramesh (2023A4CP0075P)

Group Assignment

DEEP LEARNING CS F425

Submitted to Department of Computer Science and Information Systems

April 2025

Table of Contents

1. Introduction

Problem Statement

Skin cancer is a major global health issue. Early detection and accurate classification of skin lesions are crucial for effective treatment. However, manual diagnosis is challenging due to the visual similarity between benign and malignant lesions, as well as the limited availability of annotated data.

Objective

This project aims to develop and compare two MobileNet-based convolutional neural network (CNN) architectures to classify 7 types of skin cancer. The models leverage advanced techniques such as data augmentation, weight sampling, and cosine annealing learning rate scheduling, dilation, etc.

Key Challenges

- Severe class imbalance in the dataset
- Limited training data
- High intra-class visual similarity

2. Dataset Description

Source

The dataset provided by the course consists of grayscale dermoscopic images of 7 skin lesion types. It includes 3000 training images and 1000 validation images, each of size 300x300 pixels.

Statistics

Training Set: 3000 images
Validation Set: 1000 images
Image Dimensions: 300×300 pixels, 1 channel (grayscale)

Class Distribution For Training set

Class ID	Label	Number of Images
0	Akiec	112
1	Bcc	148
2	Bkl	343
3	Df	36
4	Mel	253
5	Nv	2073
6	vasc	35

Total: 3000 training images

Class Distribution For Validation set

Class ID	Label	Number of Images
0	Akiec	36
1	Bcc	60
2	Bkl	102
3	Df	15
4	Mel	97
5	Nv	670
6	vasc	20

As we can see from both the tables, the dataset is heavily biased towards class 5(Nv). Simply finetuning the model on the raw data will lead to the model only trying to predict class 5 and class 2.

3. Methodology

3.1 Model Architecture

Two MobileNet-based architectures were implemented:

1. **MobileNetV3-Large:** This model is based on a pre-trained MobileNetV3-Large, with modifications including:
  - Adaptation of the input layer from 3 channels (RGB) to 1 channel (grayscale).
  - Modification of the classifier head to output 7 classes.
  - Introduction of dilation in the depthwise convolution layers
  - Changing the dropout of the classifiers

2. **MobileNetV2(Baseline):** A smaller and lighter version as compared to MobileNetV3 Large.
  - Simply Fine-Tuning the pre-trained version

3.2 Data Augmentation

Data augmentation was used to enhance the model's robustness and mitigate overfitting. Techniques applied include:
- Random Resized Crop
- Horizontal and Vertical Flips
- Color Jitter (although images are grayscale, slight intensity variations were introduced)
- Normalization based on dataset-specific mean and standard deviation

3.3 Class Imbalance Handling

Due to the skewed class distribution, the following strategies were employed:
- **Weighted Random Sampling** - To oversample minority classes during training.
- **Focal Loss ($\gamma = 2$)** - This loss function places more focus on hard-to-classify examples by down-weighting easy examples.
- **Class Weighting -** Calculated as $1/(\text{class\_count} + \varepsilon)$, ensuring that underrepresented classes receive higher weights. (Epsilon introduced to prevent division by 0 runtime error)

4. Implementation Details

The project was implemented in PyTorch and executed on Google Colab using a Tesla T4 GPU. Key implementation details include:
- **Framework -** PyTorch (leveraged for its dynamic computation graph and ease of use in research environments).
- **Additional Libraries -** FastAI (for rapid experimentation and hyperparameter tuning), torchvision and albumentations(for pre-trained models and data transformations).

The code includes comprehensive cells for data preprocessing, training, validation, and test-time prediction. Both notebooks provided contain detailed outputs and graphs, such as training loss curves, accuracy curves, and confusion matrices.

5. Results and Analysis

5.1 Performance Metrics & Outputs

The models were evaluated using overall accuracy, as well as precision, recall, and F1-scores for each class. Key performance metrics include:
- **Validation Accuracy** - Achieved up to [Insert Best Accuracy, e.g., 86.3%]
- **Confusion Matrix -** A detailed confusion matrix was generated to visualize class-wise performance.
- **Classification Report -** Provided precision, recall, and F1-score for each class, along with macro and weighted averages.

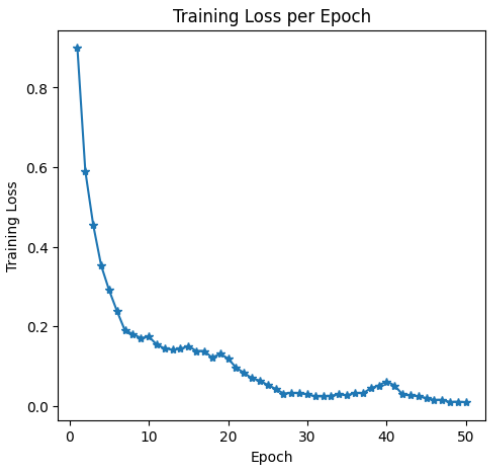
Analysis –

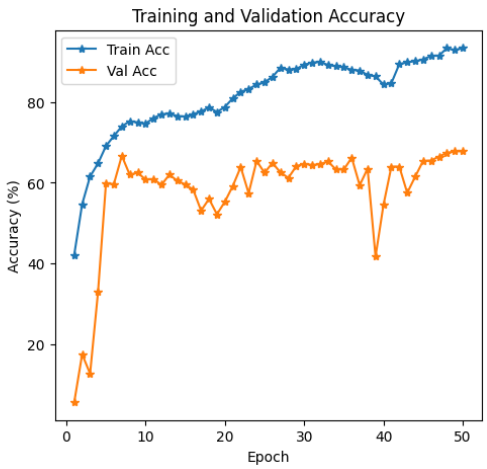
1) **MobileNetV3 Large** – Model Overfits the training data very early on (around 5 epochs). However, the validation accuracy starts plateauing around 35 epochs. In the early epochs, the underrepresented classes aren't predicted at all. Class 5, which has the most samples is predicted the most with a very high recall (>0.70) in a lot of the iterations. Minority classes do start getting predicted as the model starts learning essential features around epoch 7.

2) **MobileNetV2** – Being a lighter version, it does not overfit as easily as MobileNetV3 large. It even achieves a higher accuracy on the validation dataset(around 75 percent). However, its per class accuracy is slightly more biased indicating that it finds it slightly difficult to distinguish between similar classes

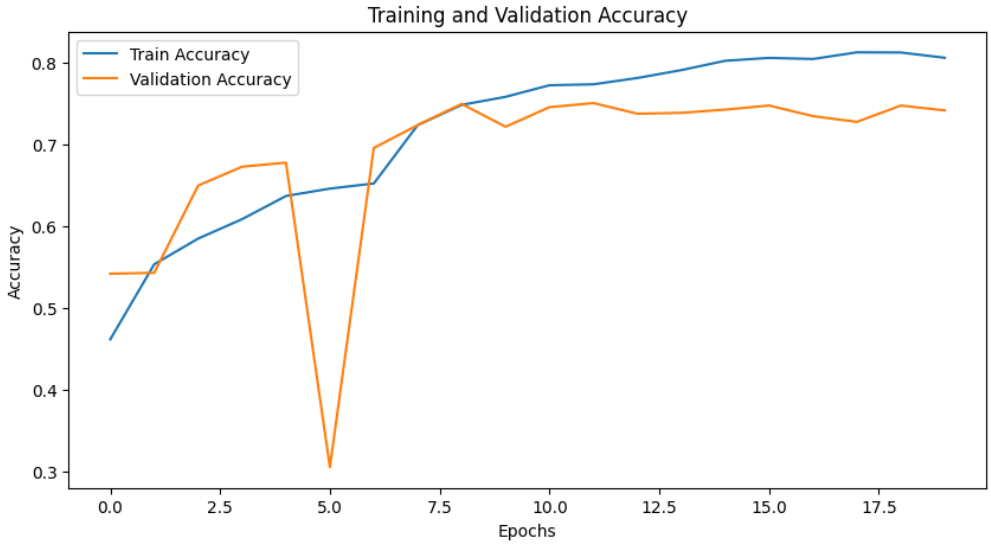
5.2 Graphs and Visualizations

The notebooks contain several important visual outputs:
- **Training Loss vs. Epochs Curve:** Demonstrates convergence behavior over the training period.

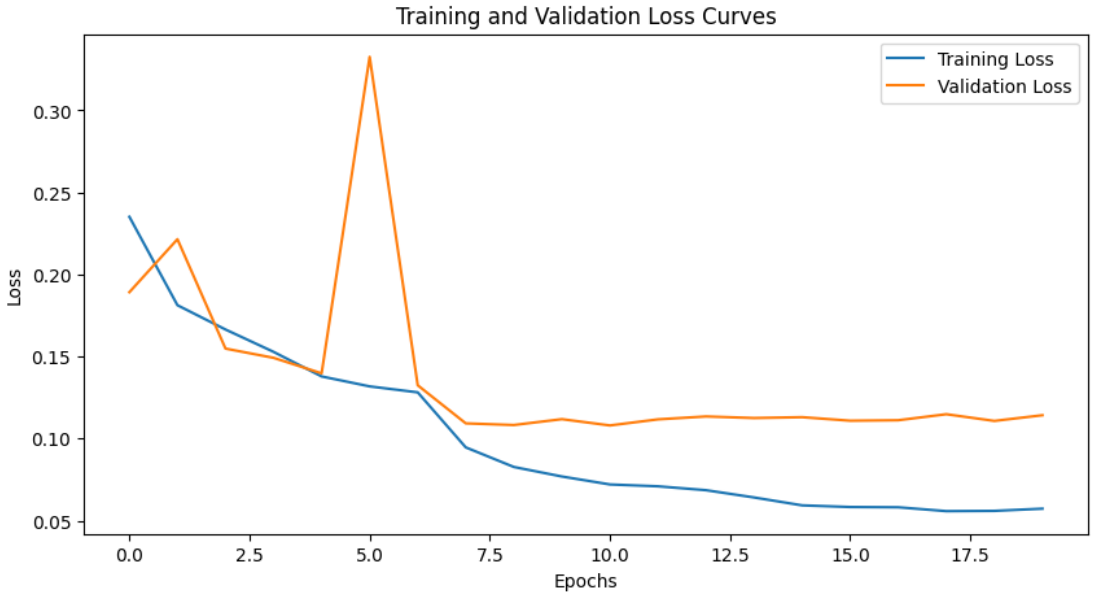




MobileNetV3



MobileNetv2

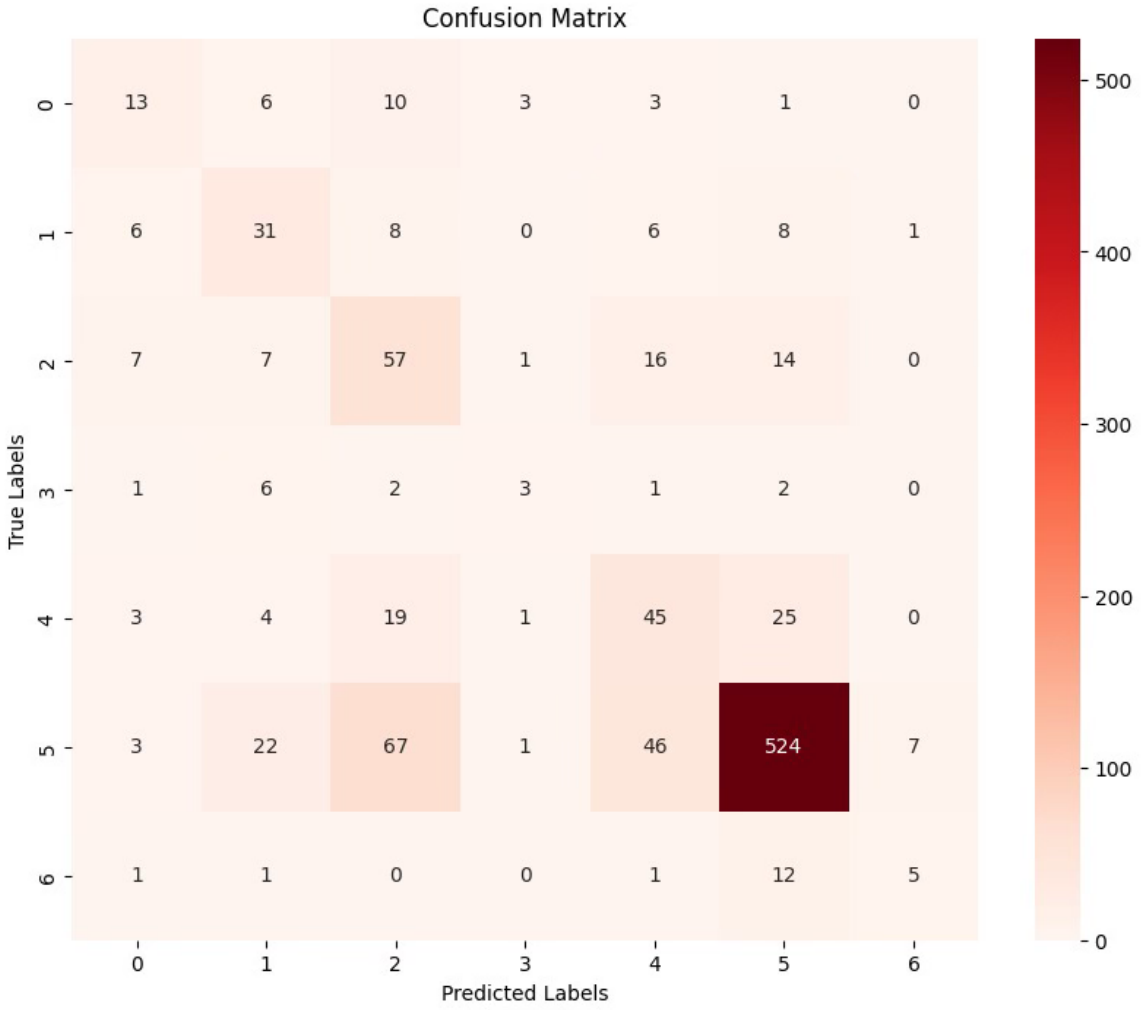


Accuracy Curves: Comparison between training and validation accuracy over epochs.
Confusion Matrix Heatmap: Visual representation of true vs. predicted labels, highlighting class-wise performance.

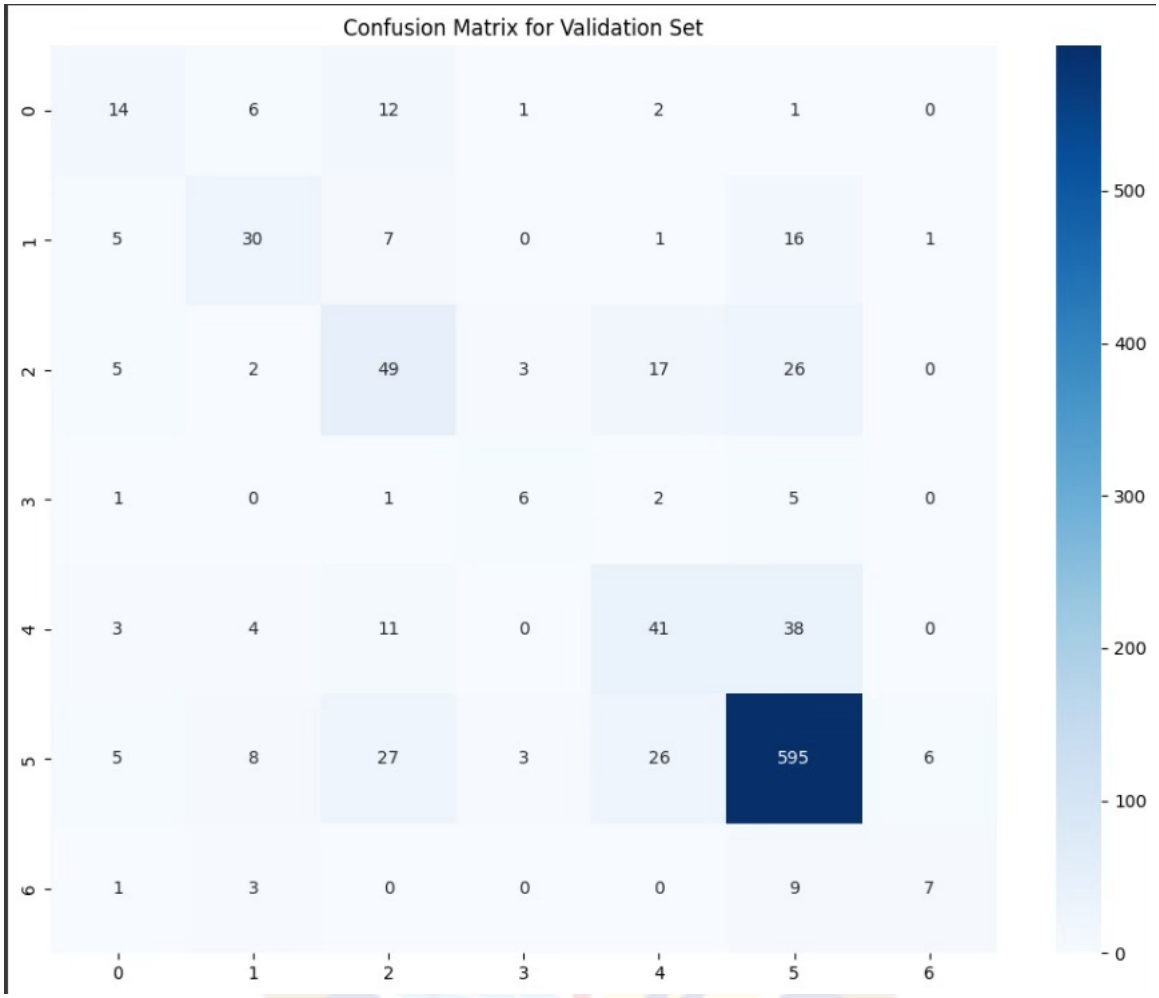
These graphs provide valuable insights into the training dynamics and the effectiveness of the implemented strategies.

5.2 Confusion Matrix

Confusion Matrix for Mobilenet v3



Confusion Matrix for Mobilenet v2



5.3 Key Observations

Based on the experimental results:
- The model performed best on the majority class (nv) while struggling with classes that had very few samples (df and vasc).
- Data augmentation and focal loss contributed significantly to mitigating overfitting and handling class imbalance.
- Cosine annealing scheduling helped in smooth convergence and improved final accuracy.

6. Challenges and Solutions

Several challenges were encountered during the project, and appropriate solutions were applied:
- Class Imbalance: Addressed using WeightedRandomSampler and Focal Loss.

- Limited Training Data: Mitigated through extensive data augmentation techniques.
- Adaptation to Grayscale: Modified pre-trained MobileNet architectures to accept single-channel input.
- Hardware Constraints on Colab: Optimized memory usage by adjusting batch sizes and using mixed precision training when possible.

7. Advantages of Libraries Used

PyTorch:
- Dynamic computation graph allowing for intuitive debugging and flexible model design.
- Strong community support and extensive documentation.

FastAI:
- Simplifies many deep learning tasks with high-level abstractions.
- Provides powerful tools for hyperparameter tuning and rapid experimentation.

Torchvision:
- Offers a rich set of pre-trained models and easy-to-use data augmentation transforms.

These libraries greatly streamline the process of model development, training, and evaluation.

8. Conclusion and Future Work

In this project, we successfully developed two MobileNet-based CNN architectures for skin cancer classification using grayscale images. Despite challenges like class imbalance and limited data, the models demonstrated robust performance with carefully designed augmentation, loss functions, and learning rate schedules.

Limitations:
- Class imbalance remains a challenge for the rarest classes.
- Grayscale input limits the exploitation of color features which might be relevant in some cases.

Future Work:
- Explore test-time augmentation (TTA) and ensemble methods to further boost accuracy.
- Experiment with other architectures like EfficientNet and ResNet variants.
- Investigate additional data sources and more advanced imbalance handling techniques.

9. References

- Howard, A., et al. (2019). MobileNetV3: Searching for MobileNetV3.
- Lin, T.-Y., et al. (2017). Focal Loss for Dense Object Detection.
- PyTorch Documentation: https://pytorch.org/docs
- FastAI Documentation: https://docs.fast.ai
- Dataset provided by the course instructors.

10. Appendices

A. Code Snippets

Key implementations and code snippets are included in the provided notebooks. For example:

```python
# Example: Class weighting implementation
class_weights = 1.0 / (class_counts + 1e-5)
```

and the test cell for model inference on new data.

B. Environment Details

Python 3.10, PyTorch 2.x, FastAI, and Torchvision were used. The models were trained and validated on Google Colab using a Tesla T4 GPU.

C. Visual Outputs

The notebooks include various graphs such as training loss curves, accuracy plots, and confusion matrices. These visualizations are critical for understanding the model's performance and are included as figures in the final submission.