Deep Learning (CS F425) Term Project Report

Skin Cancer Classification

Submitted To:

Dr. Bharat Riccharya	Dept of CSIS, BITS Pilani
Dr. Pratik Narang	Dept of CSIS, BITS Pilani

Submitted By :

Sanju Suresh	(2021A7PS2537P)
Afzal Aftab	(2021A7PS2424P)
Saarthak Vijayvargiya	(2021A3PS1044P)

15 April 2025

Table of Contents

Introduction

Skin cancer is one of the most common types of cancer globally, with millions of new cases reported each year. Early detection plays a crucial role in improving survival rates and treatment outcomes. However, accurate diagnosis often requires expert dermatological knowledge, which may not always be accessible; especially in under-resourced or rural areas. This challenge presents a compelling opportunity for automation through the use of deep learning.

In this project, we explore the application of Convolutional Neural Networks (CNNs) for the automated classification of different types of skin cancer based on dermoscopic images. CNNs have proven to be highly effective in various image classification tasks due to their ability to learn spatial hierarchies of features directly from data, without the need for manual feature engineering.

The motivation behind this project stems from both a medical and technical perspective: to bridge the gap in diagnostic accessibility and to apply deep learning techniques learned throughout the course in a real-world, impactful problem domain. By developing a robust and accurate classification model, we aim to contribute to ongoing efforts in computer-aided dermatology and demonstrate how deep learning can support healthcare practitioners in making timely and informed decisions.

This project is undertaken as a partial fulfillment of the Deep Learning course requirements and serves as a practical implementation of the theoretical concepts covered in class.

Dataset Description

The dataset used for this project focuses on the classification of skin cancer lesions into seven distinct categories. The dataset used in this project is a curated collection of **dermatoscopic grayscale images** intended for the task of **multi-class classification of skin cancer**. It draws inspiration from widely adopted medical datasets such as **HAM10000**, **ISIC**, and **PH2.**

Class Distribution

The dataset encompasses **seven distinct categories** of skin lesions:

Class Name	Class Label	Chosen Label*	Description
akiec	1	0	Actinic keratoses and intraepithelial carcinoma (Bowen's disease) – a form of pre-cancerous lesion.
bcc	2	1	Basal cell carcinoma – a common type of skin cancer.
bkl	3	2	Benign keratosis-like lesions – non-cancerous skin growths.
df	4	3	Dermatofibroma – a benign fibrous nodule.
mel	5	4	Melanoma – one of the most dangerous skin cancers.
nv	6	5	Melanocytic nevi – common moles.
vasc	7	6	Vascular lesions – includes angiomas, angiokeratomas, and hemorrhages.

** We have changed the label in the project as Pytorch uses 0 indexing.*

The dataset mimics a real-world clinical scenario, where **class imbalance is prominent**, with certain conditions like **melanocytic nevi (nv)** appearing far more frequently than rare types like **dermatofibroma (df)** or **vascular lesions (vasc)**.

Challenges Inherent to the Dataset

1. **Class Imbalance**: Some classes are underrepresented, which can bias the model during training. Without mitigation techniques (like weighted loss, oversampling, or augmentation), the model may favor majority classes.

2. **Grayscale Images**: Unlike typical RGB images used in pretrained models (like MobileNet), this dataset consists of grayscale images. Thus, preprocessing had to include **channel expansion** (e.g., repeating the grayscale channel 3 times) to match expected input dimensions.

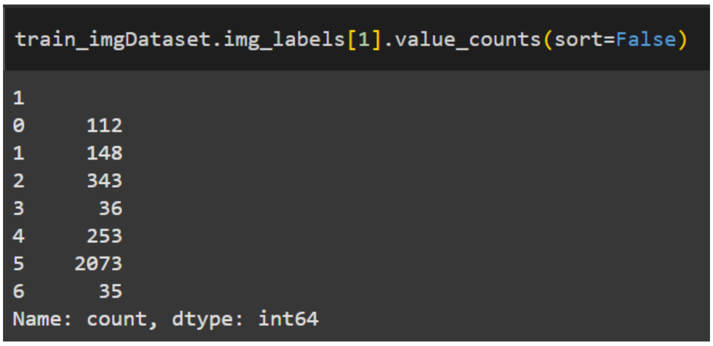
3. **Visual Similarities**: Many lesions appear similar in shape and texture, making feature extraction non-trivial, especially without color.

4. **Medical Domain Specificity**: Requires models to capture subtle textural differences that are medically significant but visually nuanced.

These categories represent a mix of malignant and benign skin conditions and are typically imbalanced in real-world datasets, a challenge also observed in this project.

Image Properties

- **Total Images**: 4000
  - **Training Set**: 3000 images
  - **Validation Set**: 1000 images
- **Image Format**: Grayscale
- **Dimensions**: 300 x 300 pixels
- **Filename Labeling**: Each image filename is prefixed with a digit (1–7) indicating its class label (e.g., 1_abc_number.png). Where 1 is its class label.



```
train_imgDataset.img_labels[1].value_counts(sort=False)
```

```
1
0     112
1     148
2     343
3      36
4     253
5    2073
6      35
Name: count, dtype: int64
```

Methodology

The overall methodology adopted for this project can be broken down into a systematic pipeline, encompassing data preprocessing, model selection, training strategies, evaluation, and fine-tuning. The project leverages transfer learning to efficiently train accurate deep learning models on a relatively small medical image dataset.

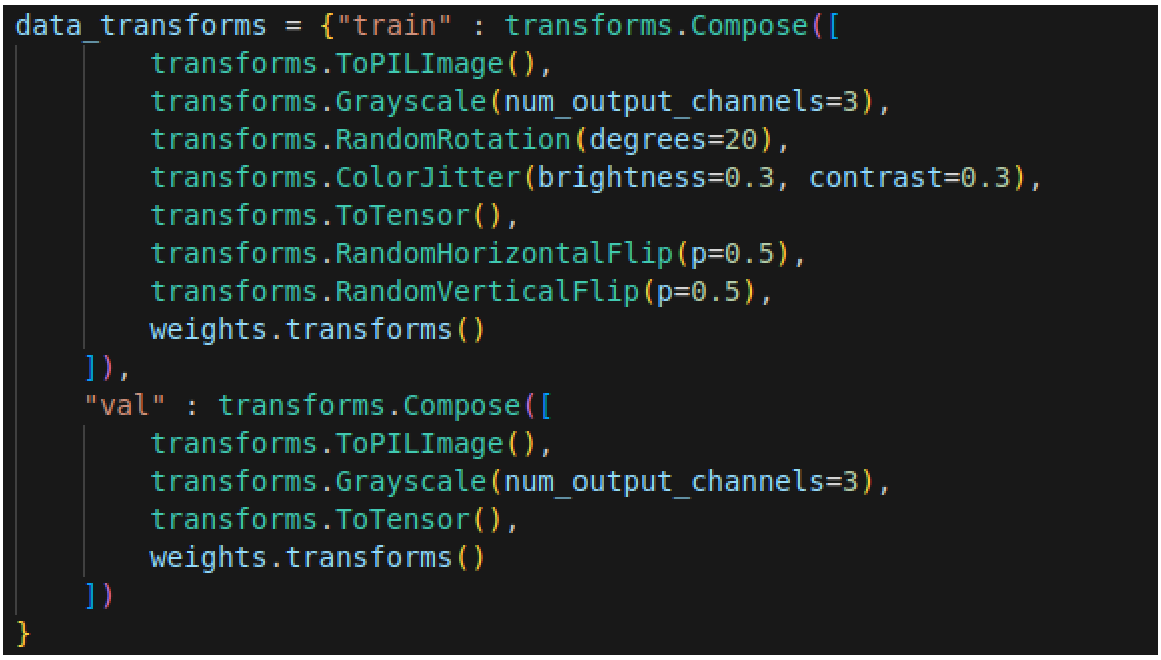
1. Data Preprocessing and Loading

- **Image Standardization**: All input images were resized to a consistent shape (based on the model architecture requirements) to ensure compatibility with the pretrained networks.

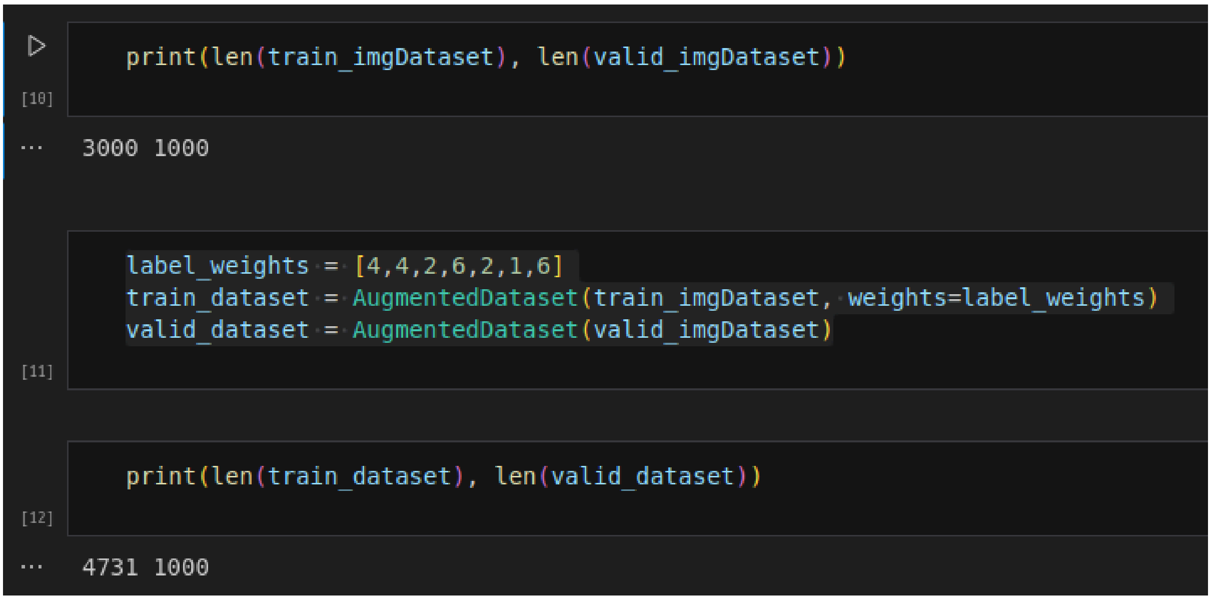
- **Normalization**: Pixel values were normalized to the range [0, 1], and optionally standardized using ImageNet mean and standard deviation, which aligns with the pretrained model expectations.

- **Augmentation** :
  - ○ Random horizontal/vertical flips
  - ○ Random rotation
  - ○ Brightness/contrast jittering

These help the model generalize better by simulating different imaging conditions.



```
data_transforms = {"train" : transforms.Compose([
        transforms.ToPILImage(),
        transforms.Grayscale(num_output_channels=3),
        transforms.RandomRotation(degrees=20),
        transforms.ColorJitter(brightness=0.3, contrast=0.3),
        transforms.ToTensor(),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomVerticalFlip(p=0.5),
        weights.transforms()
    ]),
    "val" : transforms.Compose([
        transforms.ToPILImage(),
        transforms.Grayscale(num_output_channels=3),
        transforms.ToTensor(),
        weights.transforms()
    ])
}
```



```
[10]
print(len(train_imgDataset), len(valid_imgDataset))
```

```
3000 1000
```

```
[11]
label_weights = [4,4,2,6,2,1,6]
train_dataset = AugmentedDataset(train_imgDataset, weights=label_weights)
valid_dataset = AugmentedDataset(valid_imgDataset)
```

```
[12]
print(len(train_dataset), len(valid_dataset))
```

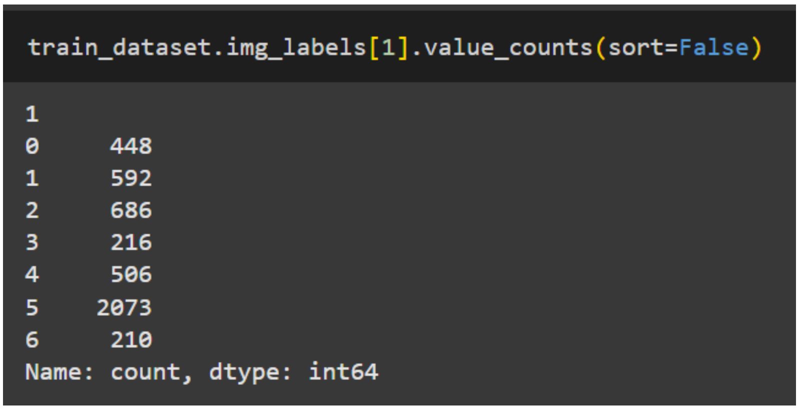
```
4731 1000
```

Handling Class Imbalance

Medical image datasets, especially those related to skin cancer, often suffer from **class imbalance** — where certain types of lesions appear far more frequently than others. To address this issue, we implemented a **class-weighted sampling strategy** to ensure that the model receives a more balanced learning signal across all classes during training.

These weights represent the **inverse frequency** of each class in the dataset. Less frequent classes were assigned **higher weights** (e.g., classes 3 and 6 have a weight of 6), while more frequent classes (e.g., class 5) had lower weights (weight = 1). This helps the model focus more on underrepresented classes.

- The AugmentedDataset class was customized to include sampling based on the provided weights.
- During training, the sampler used these weights to bias the batch composition, making it more likely to sample underrepresented classes more frequently.
- For the validation dataset, no weighting was applied since validation is intended to reflect the natural class distribution and serve as an unbiased measure of generalization.



```
train_dataset.img_labels[1].value_counts(sort=False)
```

```
1
0     448
1     592
2     686
3     216
4     506
5    2073
6     210
Name: count, dtype: int64
```

Dataset Split: The dataset was split into:

- **Training set**
- **Validation set**
- **Test set** -The test set was kept completely unseen until the final evaluation phase.

Model Architectures

MobileNet is a family of lightweight deep neural networks designed for efficient mobile and edge-device applications. It was developed by Google to provide a balance between computational efficiency and accuracy, making it ideal for tasks like image classification, object detection, and segmentation on resource-constrained devices.

Key Features of MobileNet:

1. **Depthwise Separable Convolutions:**

   - MobileNet replaces standard convolutions with **depthwise separable convolutions**, which break down convolution into two steps:

     - **Depthwise convolution**: Applies a single filter to each input channel separately.
     - **Pointwise convolution**: Uses a 1x1 convolution to combine features across channels.

   - This significantly reduces the number of parameters and computations.

2. **Width Multiplier (α):**

   - Controls the number of channels in each layer, allowing a trade-off between accuracy and efficiency.

3. **Resolution Multiplier (ρ):**

   - Reduces the input image resolution, further decreasing computational cost.

4. **Variants of MobileNet:**

   - **MobileNetV1 (2017)**: Introduced **depthwise separable convolutions** to drastically reduce the number of parameters and computations compared to traditional convolutional layers.

   - **MobileNetV2 (2018)**: Added **inverted residual blocks** and **linear bottlenecks**, enabling better memory efficiency and faster inference without significant loss in accuracy.

   - **MobileNetV3-Small (2019)**: Designed for **low-resource devices**, this variant combines **Neural Architecture Search (NAS)** optimization with **Squeeze-and-Excite (SE) blocks** to deliver decent performance with minimal computational cost. It is built for extremely lightweight applications, especially those running on **low-power microcontrollers or edge devices**. It has around **2.9 million parameters** and only **66 million FLOPs**, making it highly efficient in terms of speed and energy usage.

   - **MobileNetV3-Large (2019)**: Tailored for tasks requiring **higher accuracy**, this model leverages NAS, SE blocks, and **swish-like activations** (hard-swish) to balance performance and efficiency for more capable hardware. It has a deeper and wider architecture, approximately **5.4 million parameters**, and about **219 million FLOPs**, making it more powerful but also slightly heavier

Applications:

- Image classification (e.g., MobileNet + ImageNet)
- Object detection (e.g., SSD-MobileNet)
- Face recognition

- Pose estimation
- Embedded AI on mobile and IoT devices

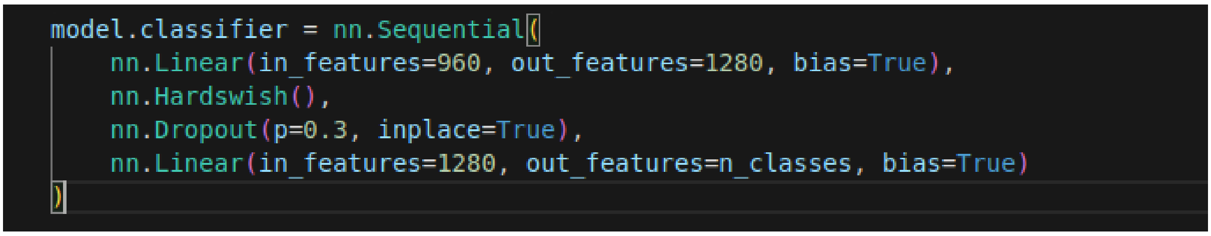
Training Strategy

Model 1 - Based on MobileNetV3(Large)

1. Model Architecture

The model used in this project is a **pre-trained MobileNetV3-Large** network, imported from the torchvision.models module. This lightweight convolutional neural network is known for its efficient design, particularly suited for mobile and edge devices.

We loaded the model with pretrained weights (MobileNet_V3_Large_Weights.DEFAULT) and then customized the final classification layers to adapt it to the target task. The original classifier of MobileNetV3-Large was replaced with a new sequential block:



```
model.classifier = nn.Sequential(
    nn.Linear(in_features=960, out_features=1280, bias=True),
    nn.Hardswish(),
    nn.Dropout(p=0.3, inplace=True),
    nn.Linear(in_features=1280, out_features=n_classes, bias=True)
)
```

- **Linear Layer 1**: Expands the feature space from 960 to 1280 dimensions.

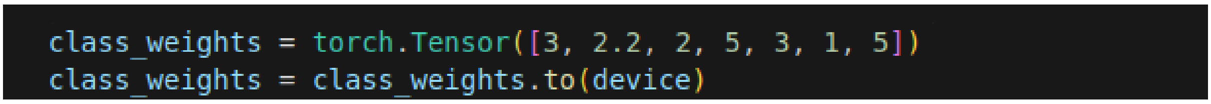
- **Hardswish Activation**: Non-linear activation function tailored for performance and accuracy trade-offs in MobileNet architectures.

- **Dropout Layer**: Regularization to prevent overfitting, with a dropout probability of 0.3.

- **Final Linear Layer**: Maps features to the number of target classes (n_classes).

2. Handling Class Imbalance

To address class imbalance, **class weights** were incorporated into the loss function. These weights penalize misclassification of under-represented classes more heavily. The class weights were defined manually:



```
class_weights = torch.Tensor([3, 2.2, 2, 5, 3, 1, 5])
class_weights = class_weights.to(device)
```

These weights were passed into a **weighted loss criterion** CrossEntropyLoss with weight=class_weights.

3. Training Loop Overview

The train_model function handles the training logic for a specified number of epochs.

Key Components

- **Inputs**:
    - model: Initialized and customized MobileNetV3 model.
    - train_loader, validation_loader: Dataloaders for training and validation sets.
    - criterion: Loss function with class weighting(**CrossEntropyLoss**). Since it is a multi class classification.
    - optimizer: Optimizer instance (**Adam**).
        - The **Adam optimizer** was chosen for its adaptive learning rate and fast convergence properties.
    - scheduler: Optional learning rate scheduler. (**StepLR**)
        - **StepLR scheduler** helps gradually reduce the learning rate at fixed intervals to fine-tune learning and improve model stability during later training stages.
    - n_epochs: Number of training epochs. (**12**)
    - Learning rate: **0.0005** Selected through empirical tuning.
        - Higher rates led to overshooting, and lower ones slowed convergence.

- **Outputs**:
    - accuracy_list: Validation accuracy per epoch.
      loss_list: Training loss per epoch.
    - model: Best-performing model (based on validation accuracy and training cost).
    - models_list: Model checkpoints per epoch.

Training Logic Per Epoch

1. **Set Model to Train Mode**:
   - model.train() activates dropout and batchnorm for training.
2. **Batch-wise Training Loop**:
   - For each batch, inputs and labels are moved to the target device (CPU/GPU).
   - Forward pass is executed: z = model(x)
   - Loss is computed: loss = criterion(z, y)
   - Gradients are computed and updated using backpropagation: loss.backward() → optimizer.step() → optimizer.zero_grad()
3. **Training Cost Logging**:
   - Training loss for the epoch is stored using the mean of all batch losses.
4. **Validation Evaluation**:
   - Model is set to evaluation mode: model.eval()
   - Accuracy is computed over the validation set.
   - No gradient calculation (torch.no_grad() assumed).
5. **Learning Rate Scheduler**:
   - If a scheduler is defined, it is stepped. If using ReduceLROnPlateau, the validation accuracy is passed as a metric. (Tried but didn't give satisfactory results)
6. **Model Checkpointing**:
   - If the validation accuracy improves or remains the same with a lower training cost, the model weights are saved.

4. Model Selection Strategy

Model selection is based on a **dual criterion**:

- **Primary**: Highest validation accuracy.

- **Secondary** (tie-breaker): Lowest training cost if validation accuracy is equal.

This ensures that the model generalizes well and avoids overfitting.

```
if accuracy > accuracy_best:

...

elif accuracy == accuracy_best and train_cost < train_cost_best:

  ...
```

5. Timing and Execution

Training start and end times are recorded using the datetime and time modules.This helps monitor and benchmark training efficiency and duration.

6. Summary

This training pipeline is robust and includes:

- Custom model head tailored to the classification task.
- Weighted loss function for class imbalance.
- Dynamic learning rate scheduling.
- Validation-based model checkpointing.
- Epoch-wise performance tracking (loss and accuracy).

Such design choices collectively enhance model performance, ensure better generalization, and provide transparency during the training process.

Fine-Tuning Strategy: Model 1 (MobileNetV3-Large)

After initial training, we performed **fine-tuning** on the MobileNetV3-Large model to further enhance its ability to discriminate between different skin lesion types, especially under class imbalance.

1. Loading and Unfreezing the Model

We began by loading the best-performing checkpoint from the initial training phase:

model.load_state_dict(mod_list[-1])

To enable end-to-end learning, **all layers of the model were unfrozen**:

```
for param in model.parameters():
    param.requires_grad = True
```

This allowed both low-level and high-level features to be refined based on our grayscale medical dataset.

2. Data Loaders and Training Setup

- **Train Batch Size**: 50
- **Validation Batch Size**: 25
- **Epochs**: 6

The dataset was loaded using PyTorch's DataLoader class, with shuffling enabled to ensure randomness during training.

3. Class Imbalance Handling

Given the noticeable **class imbalance**, we used a **weighted CrossEntropyLoss** with the following manually tuned class weights:

```
class_weights = torch.Tensor([3, 2.2, 2, 7, 3, 1, 5])
criterion2 = nn.CrossEntropyLoss(weight=class_weights)
```

These weights give more importance to underrepresented classes like:

- **Dermatofibroma (df)** – weight 7
- **Vascular lesions (vasc)** – weight 5

This helps the model avoid being biased toward more frequent classes like **melanocytic nevi (nv)**.

4. Optimizer and Learning Rate Scheduler

We adopted a **fine-tuning-friendly optimizer**:

- **Optimizer**: Adam
- **Learning Rate**: 1e-5 (a small value to ensure stability during weight updates). It is also lower than learning rate as we are fine tuning the model.
- **Scheduler**: StepLR with step_size=2 and gamma=0.5, reducing the learning rate gradually and it leads to a better convergence.

```
optimizer2 = torch.optim.Adam(model.parameters(), lr=0.00001)
scheduler2 = lr_scheduler.StepLR(optimizer2, step_size=2, gamma=0.5)
```

This setup helped maintain a steady learning pace, preventing overfitting and catastrophic forgetting of pretrained features.

5. Training and Evaluation

The model was trained using our custom train_model() function, and performance was tracked over epochs. At the end of training, we evaluated the model using a confusion matrix and saved the best-performing model checkpoint:

```
model.load_state_dict(mod_list2[2])
```

6. Summary of Fine-Tuning Strategy

Component	Details
Model	MobileNetV3-Large
Layers Unfrozen	All
Optimizer	Adam
Learning Rate	0.0005
Scheduler	StepLR (step=2, gamma=0.5)
Loss Function	Weighted CrossEntropyLoss
Class Weights	[3, 2.2, 2, 7, 3, 1, 5]
Epochs	6
Batch Size	50 (train), 25 (val)

4. Key Highlights

- The learning rate was intentionally kept low for fine-tuning to allow **small, precise updates** to pretrained weights.
- The use of **Adam optimizer** provides better generalization

- The inclusion of **class-weighted loss** was critical due to the dataset's **imbalanced class distribution**.

This phase allowed the MobileNetV3-Large model to refine its filters and representations, resulting in improved classification performance on complex and subtle medical patterns.This fine-tuning phase provided a more robust and balanced model, particularly improving performance on minority classes without compromising overall accuracy.

Model 2 – Based on MobileNetV3 (Small)

Training Strategy

1. Model Architecture

The model used is a pre-trained **MobileNetV3-Small**, loaded from the torchvision.models module using pretrained weights (MobileNet_V3_Small_Weights.IMAGENET1K_V1). This architecture is even more lightweight than MobileNetV3-Large and is particularly optimized for resource-constrained environments.

To leverage transfer learning, **all layers of the base model were frozen** (param.requires_grad = False) to retain learned feature representations from ImageNet and reduce training time and overfitting. Only the final classifier head is trained.

The original classifier was replaced with a custom nn.Sequential block:

- **Linear Layer 1**: Expands the feature space from 576 to 1024 dimensions.
- **Hardswish Activation**: Mobile-optimized non-linearity that balances performance and speed.
- **Dropout Layer**: Regularization with a dropout rate of 0.2 to reduce overfitting.
- **Final Linear Layer**: Maps the features to the number of target classes (n_classes = 7).

2. Handling Class Imbalance

To address class imbalance in the dataset, **class weights** were incorporated into the loss function. These weights were derived from the **inverse of class**

frequencies (using normalized value counts) to penalize under-represented classes more heavily.

Eventually, the following manually fine-tuned weights were used:

[3, 3, 2, 5, 3, 1, 6]

These weights were passed to the **CrossEntropyLoss** function via the weight argument.

3. Training Loop Overview

The train_model function (shared with MODEL 1) was reused to train this configuration.

Key Components – Inputs:

- **model**: MobileNetV3-Small with frozen base and custom classification head.
- **train_loader, validation_loader**: Batch iterators for training and validation datasets.
- **criterion**: CrossEntropyLoss with class weighting – suitable for multi-class classification.
- **optimizer**: Adam optimizer with learning rate 0.001.
  Chosen for its adaptive learning rate and ability to converge faster on deep architectures.
- **scheduler**: StepLR with step_size=3, gamma=0.5.
  Reduces learning rate by half every 3 epochs to fine-tune learning in later stages and improve stability.
- **n_epochs**: 15
- **batch_size**: 50
- **learning_rate**: 0.001
  Selected through empirical tuning. Higher rates led to overshooting, and lower ones slowed convergence.

Outputs:

- **accuracy_list**: Validation accuracy recorded per epoch.
- **loss_list**: Training loss recorded per epoch.
- **model:** Best-performing model based on validation accuracy and training cost.
- **models_list:** List of model checkpoints per epoch.

Fine-Tuning Strategy: Model 2 (MobileNetV3-Small)

For the second model, **MobileNetV3-Small**, we implemented a targeted fine-tuning strategy to adapt its lightweight architecture to the specific challenges of skin cancer classification. This model is optimized for environments with limited compute resources but still requires careful tuning for effective performance on complex medical image data.

1. Loading and Unfreezing

We began by loading the **best pretrained weights** from earlier training:

model.load_state_dict(mod_list[12]). Here we are choosing the model we obtained in last epoch which is 12th in our case

All layers of the network were unfrozen to allow full fine-tuning:

```
for param in model.parameters():
    param.requires_grad = True
```

This ensured the model could adjust both shallow and deep layers for features more relevant to grayscale medical images.

2. Data Loading

The model was trained on:

- **Batch Size (Train)**: 50
- **Batch Size (Validation)**: 25
- **Epochs**: 6
- The data was shuffled to enhance generalization and reduce bias due to ordering.

3. Class Imbalance Handling

We used **weighted loss** to address class imbalance. The class weights were chosen based on label distribution and misclassification trends:

```
class_weights = torch.Tensor([3, 3, 2, 5, 3, 1, 5])
```

This emphasized underrepresented and difficult classes such as:

- **df (Dermatofibroma)** – weight 5
- **vasc (Vascular lesions)** – weight 5

The weighted CrossEntropyLoss penalized misclassifications more heavily for minority classes, improving per-class recall.

4. Optimizer and Scheduler Configuration

- **Optimizer**: Adam, chosen for its adaptive learning behavior
- **Learning Rate**: A much lower value of lr / 16 was used (i.e., 0.000625 if lr = 0.01) to fine-tune the smaller model without causing overfitting or instability.
- **Scheduler**: StepLR with step size = 3 and gamma = 0.5 to gradually reduce learning rate:

```
optimizer2 = torch.optim.Adam(model.parameters(), lr=lr/16)

scheduler2 = lr_scheduler.StepLR(optimizer2, step_size=3, gamma=0.5)
```

This slower learning pace was especially important for MobileNetV3-Small, which has fewer parameters and is more prone to overfitting.

5. Training and Evaluation

The model was trained using a custom train_model() function and evaluated after each epoch. The best-performing checkpoint from the fine-tuning phase was reloaded:

```
model.load_state_dict(mod_list2[3])
```

Confusion matrices and performance metrics were analyzed post-training to assess class-wise performance.

6. Summary of Fine-Tuning Strategy

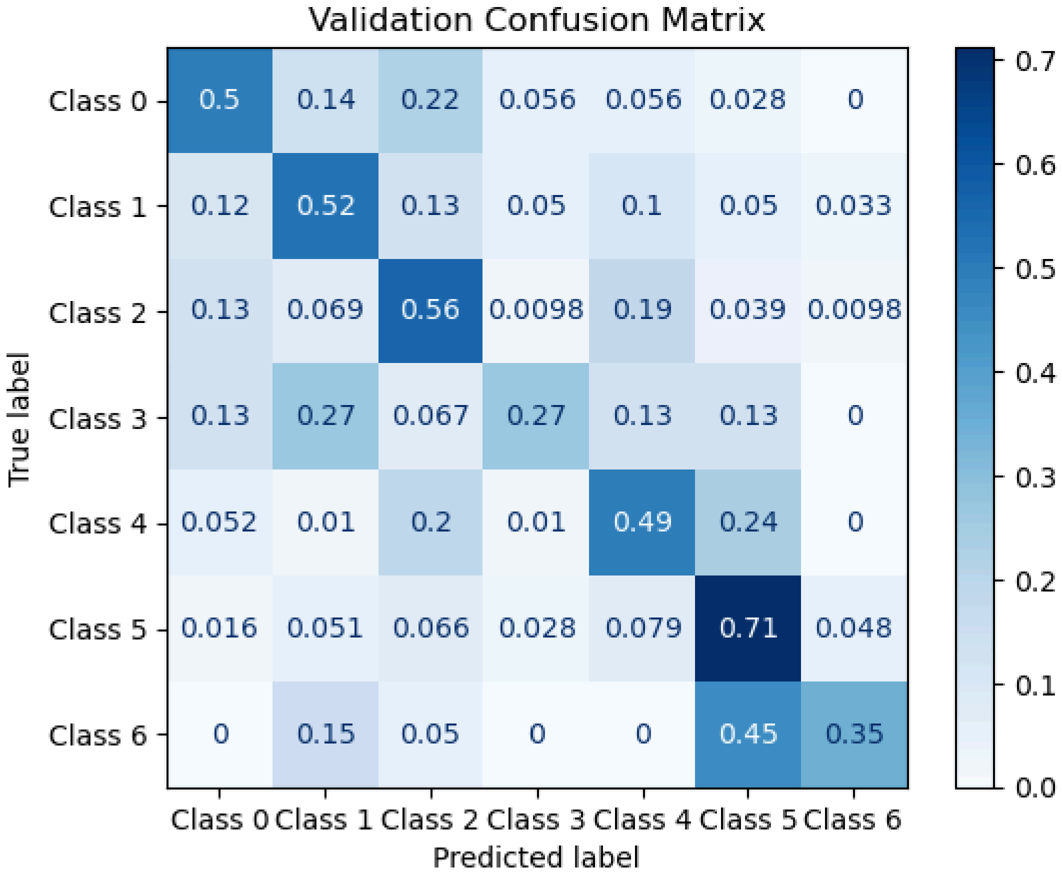
Component	Details
Model	MobileNetV3-Small
Layers Unfrozen	All
Optimizer	Adam
Learning Rate	lr / 16
Scheduler	StepLR (step=3, gamma=0.5)
Loss Function	Weighted CrossEntropyLoss
Class Weights	[3, 3, 2, 5, 3, 1, 5]
Epochs	6
Batch Size	50 (train), 25 (val)

Although MobileNetV3-Small is more lightweight than its Large counterpart, the fine-tuning strategy enabled it to achieve competitive performance while maintaining computational efficiency.

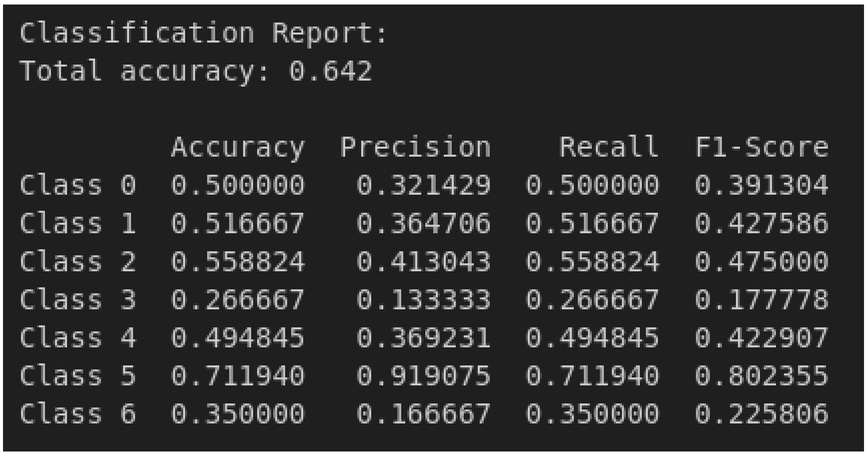
Results

Model-1 [MobileNet-V3 Large]

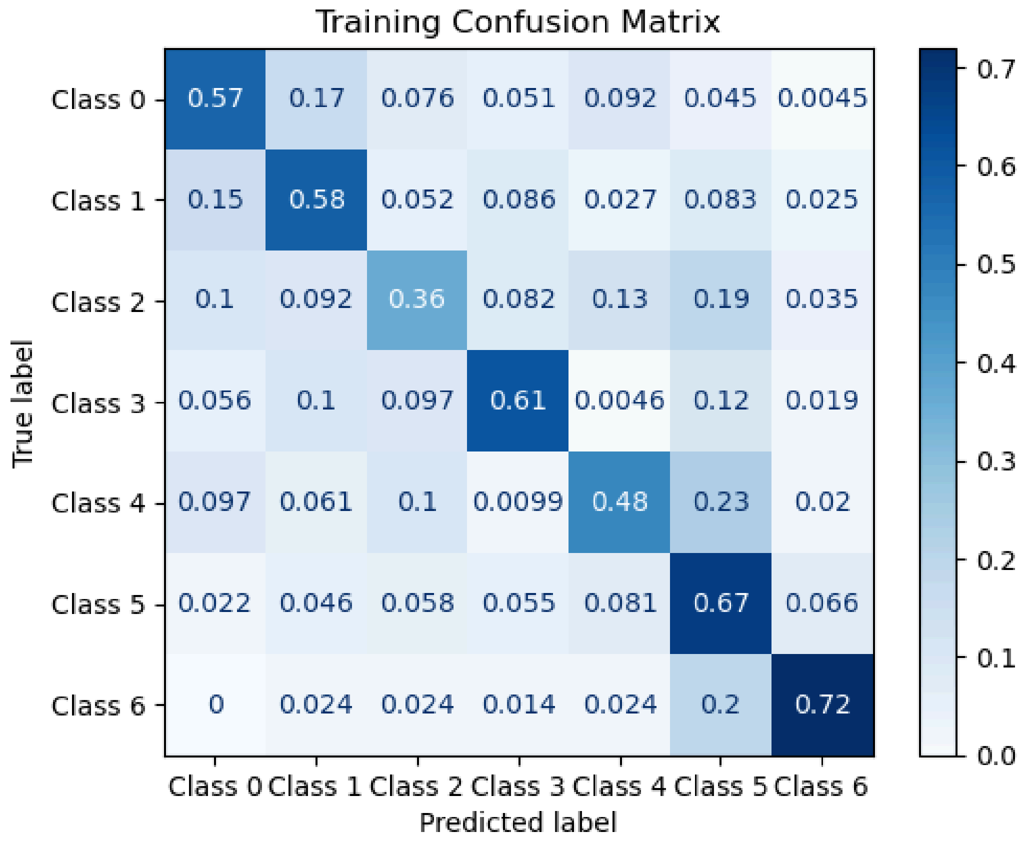
Train:

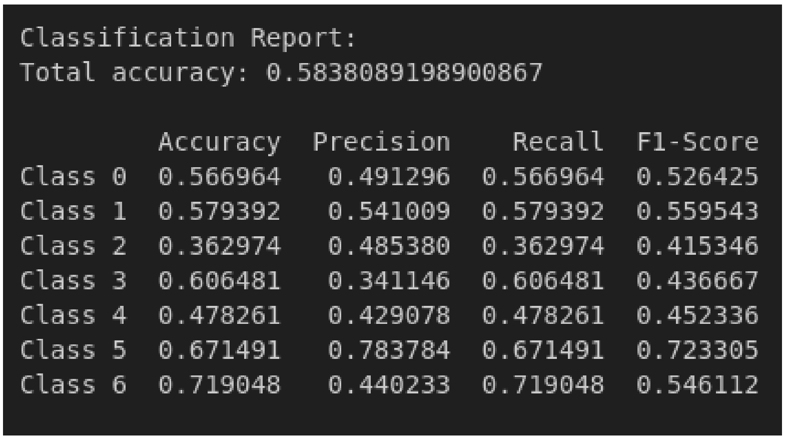


	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
Class 0	0.5	0.14	0.22	0.056	0.056	0.028	0
Class 1	0.12	0.52	0.13	0.05	0.1	0.05	0.033
Class 2	0.13	0.069	0.56	0.0098	0.19	0.039	0.0098
Class 3	0.13	0.27	0.067	0.27	0.13	0.13	0
Class 4	0.052	0.01	0.2	0.01	0.49	0.24	0
Class 5	0.016	0.051	0.066	0.028	0.079	0.71	0.048
Class 6	0	0.15	0.05	0	0	0.45	0.35



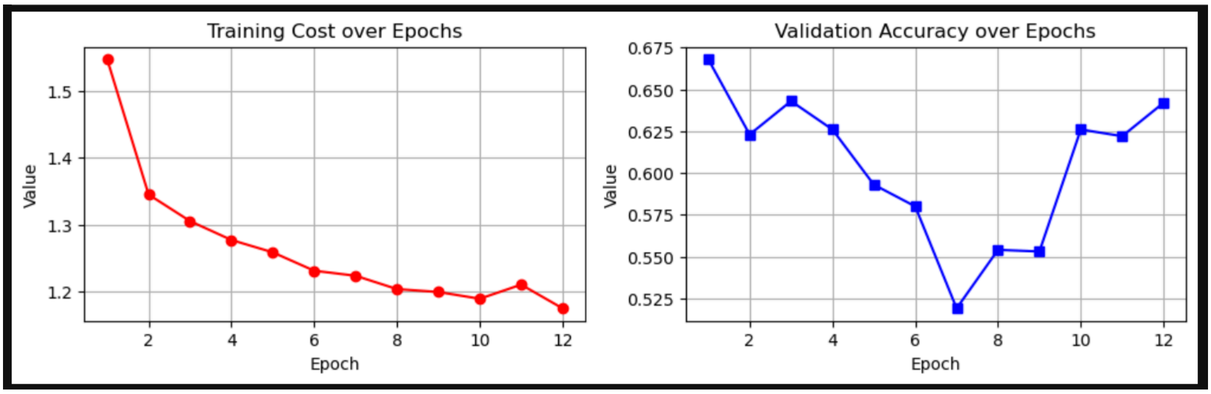
```
Classification Report:
Total accuracy: 0.642

         Accuracy  Precision    Recall  F1-Score
Class 0  0.500000   0.321429  0.500000  0.391304
Class 1  0.516667   0.364706  0.516667  0.427586
Class 2  0.558824   0.413043  0.558824  0.475000
Class 3  0.266667   0.133333  0.266667  0.177778
Class 4  0.494845   0.369231  0.494845  0.422907
Class 5  0.711940   0.919075  0.711940  0.802355
Class 6  0.350000   0.166667  0.350000  0.225806
```



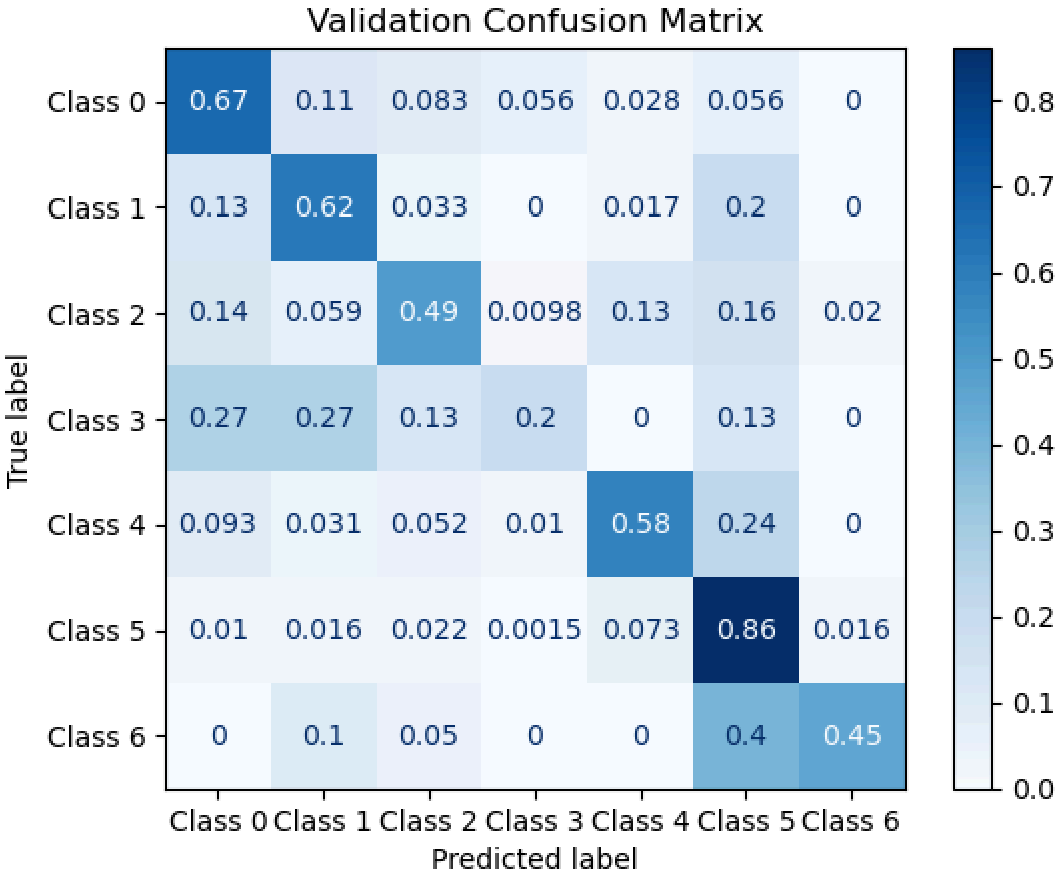


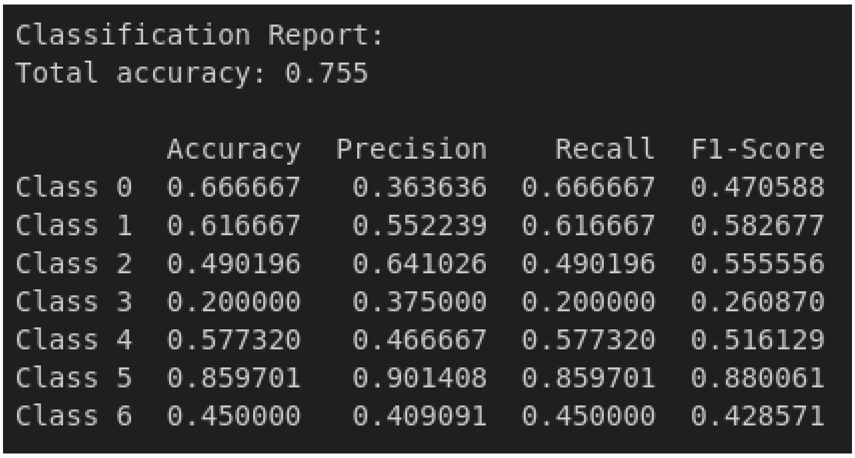
```
Classification Report:
Total accuracy: 0.5838089198900867

         Accuracy  Precision    Recall  F1-Score
Class 0  0.566964   0.491296  0.566964  0.526425
Class 1  0.579392   0.541009  0.579392  0.559543
Class 2  0.362974   0.485380  0.362974  0.415346
Class 3  0.606481   0.341146  0.606481  0.436667
Class 4  0.478261   0.429078  0.478261  0.452336
Class 5  0.671491   0.783784  0.671491  0.723305
Class 6  0.719048   0.440233  0.719048  0.546112
```

Plots:

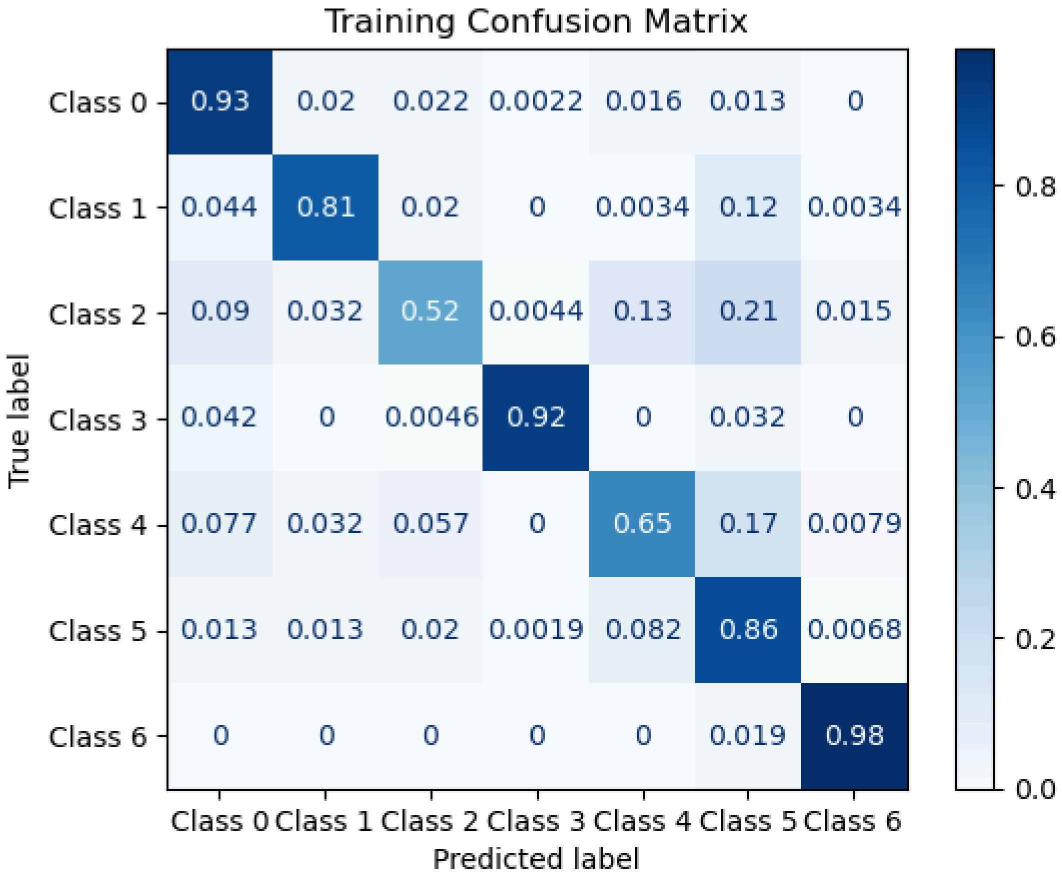


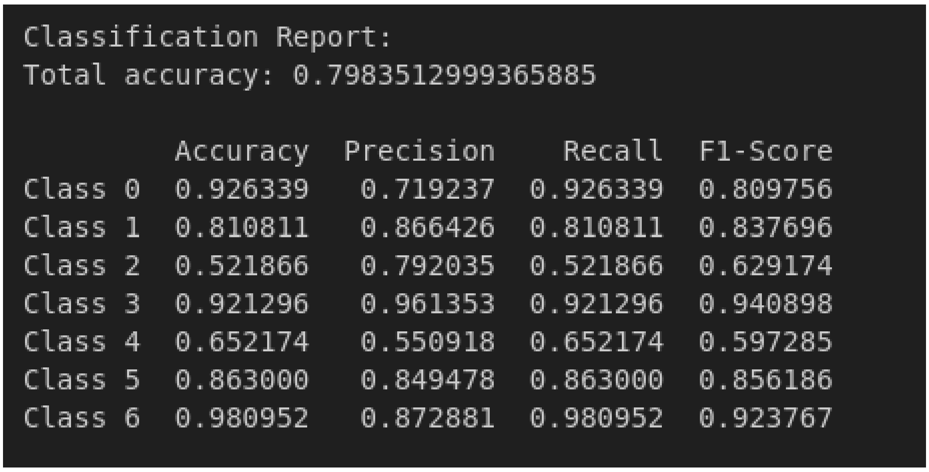
Fine Tuning





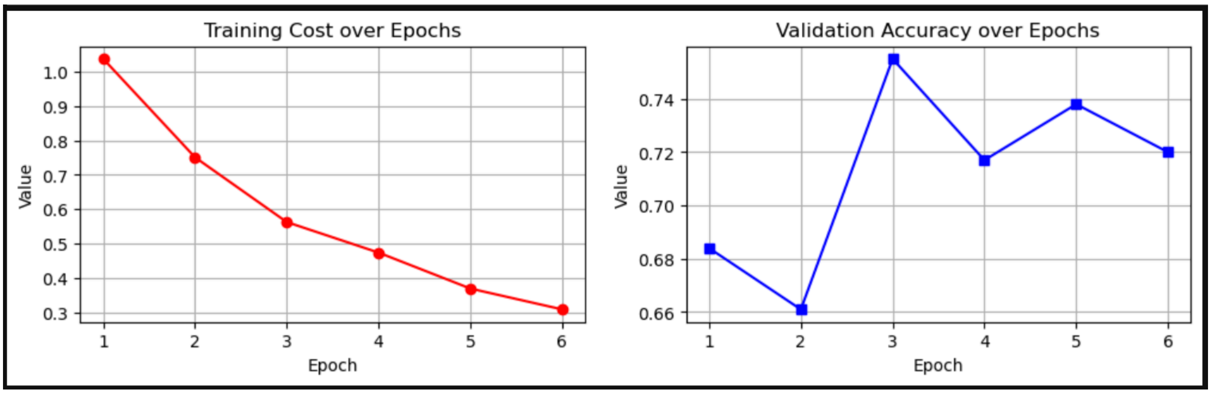
```
Classification Report:
Total accuracy: 0.755

         Accuracy  Precision    Recall  F1-Score
Class 0  0.666667   0.363636  0.666667  0.470588
Class 1  0.616667   0.552239  0.616667  0.582677
Class 2  0.490196   0.641026  0.490196  0.555556
Class 3  0.200000   0.375000  0.200000  0.260870
Class 4  0.577320   0.466667  0.577320  0.516129
Class 5  0.859701   0.901408  0.859701  0.880061
Class 6  0.450000   0.409091  0.450000  0.428571
```





```
Classification Report:
Total accuracy: 0.7983512999365885

         Accuracy  Precision    Recall  F1-Score
Class 0  0.926339   0.719237  0.926339  0.809756
Class 1  0.810811   0.866426  0.810811  0.837696
Class 2  0.521866   0.792035  0.521866  0.629174
Class 3  0.921296   0.961353  0.921296  0.940898
Class 4  0.652174   0.550918  0.652174  0.597285
Class 5  0.863000   0.849478  0.863000  0.856186
Class 6  0.980952   0.872881  0.980952  0.923767
```

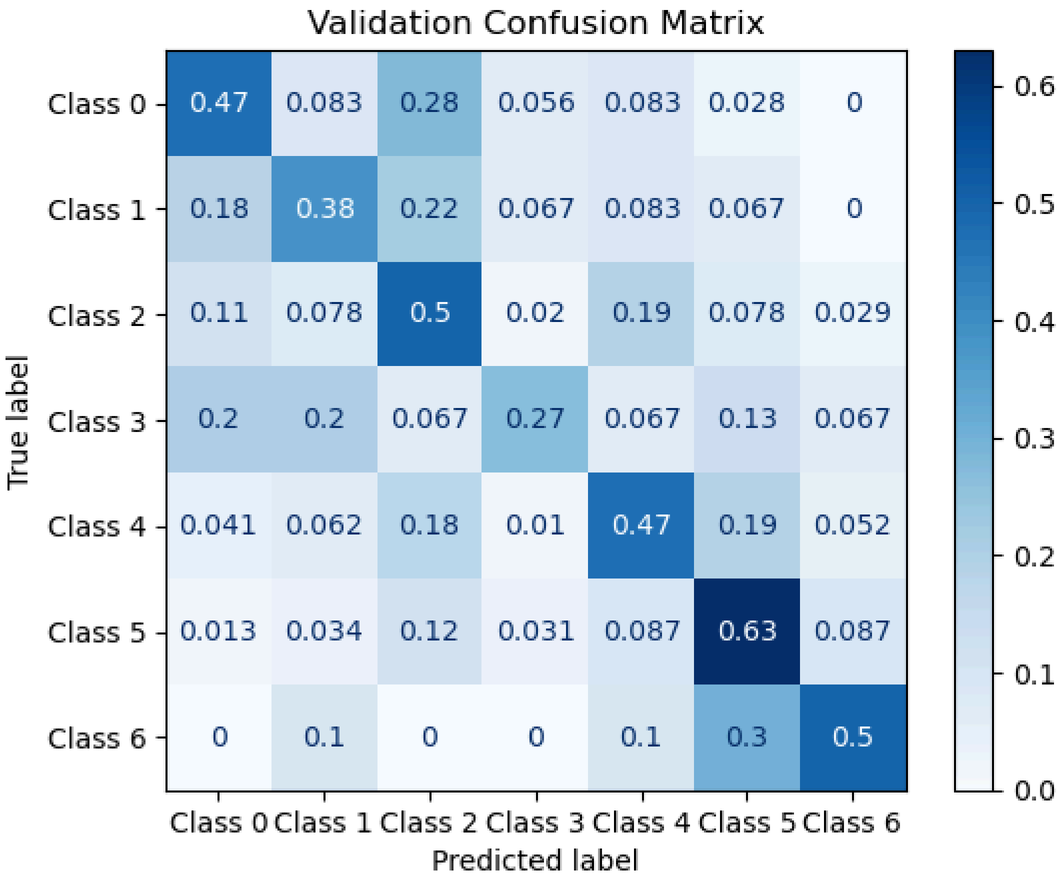
Plots:

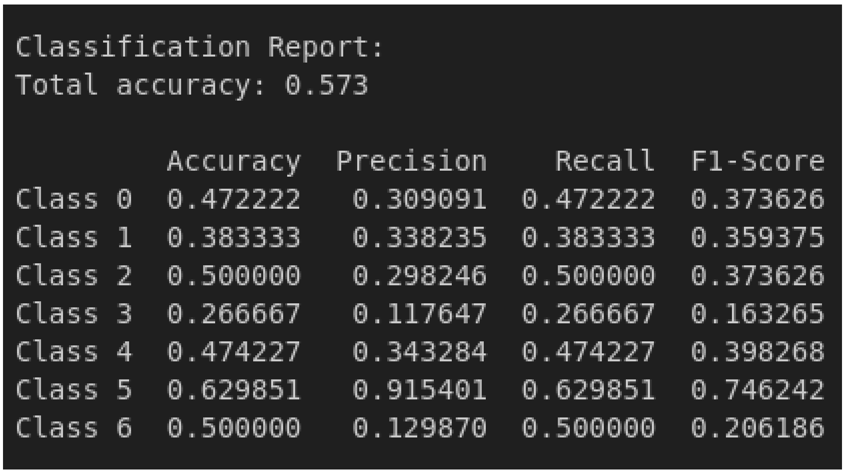


For MobileNet-V3 Large as we can see that the model was having high training loss and validation inaccuracy but after finetuning it has decreased significantly.

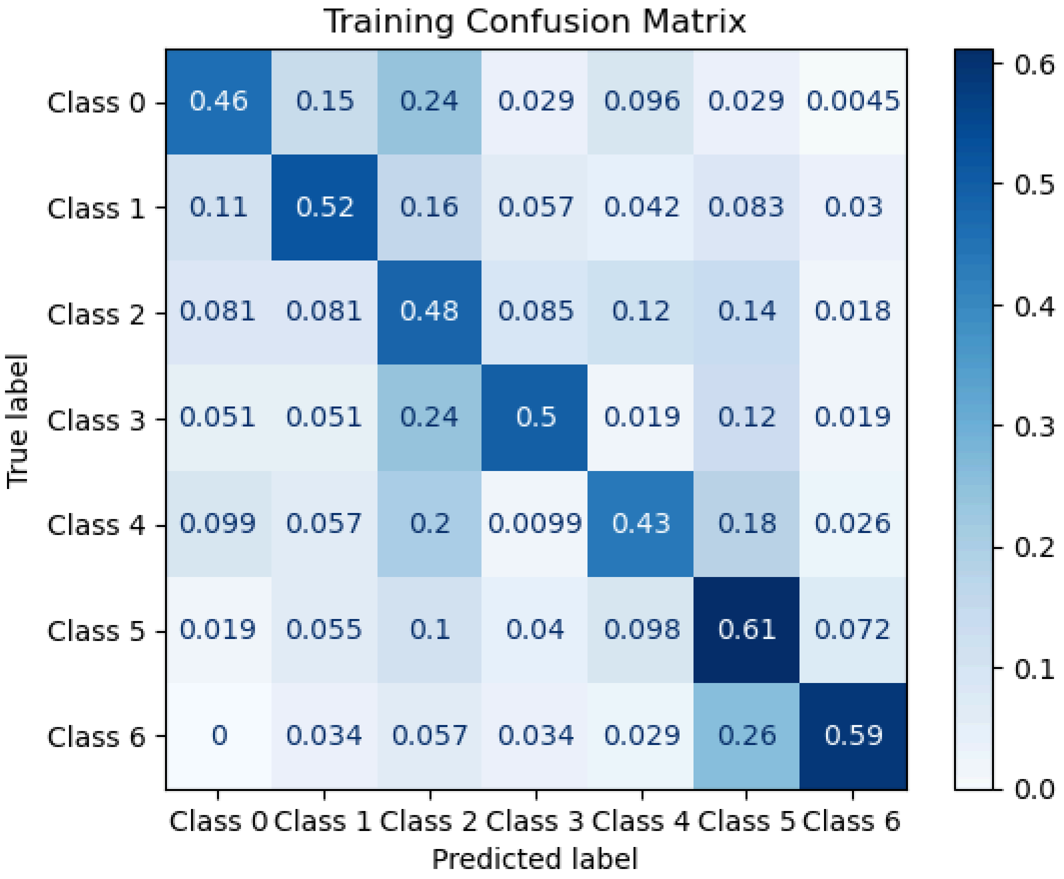
Model-2 [MobileNet-V3 Small]

Training:

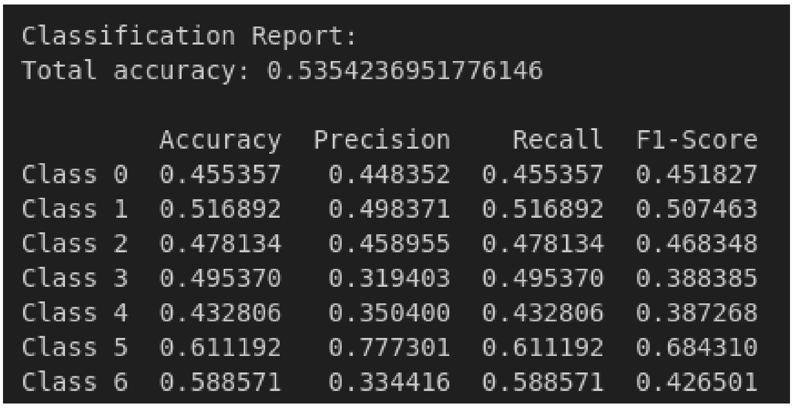




```
Classification Report:
Total accuracy: 0.573

         Accuracy  Precision    Recall  F1-Score
Class 0  0.472222   0.309091  0.472222  0.373626
Class 1  0.383333   0.338235  0.383333  0.359375
Class 2  0.500000   0.298246  0.500000  0.373626
Class 3  0.266667   0.117647  0.266667  0.163265
Class 4  0.474227   0.343284  0.474227  0.398268
Class 5  0.629851   0.915401  0.629851  0.746242
Class 6  0.500000   0.129870  0.500000  0.206186
```

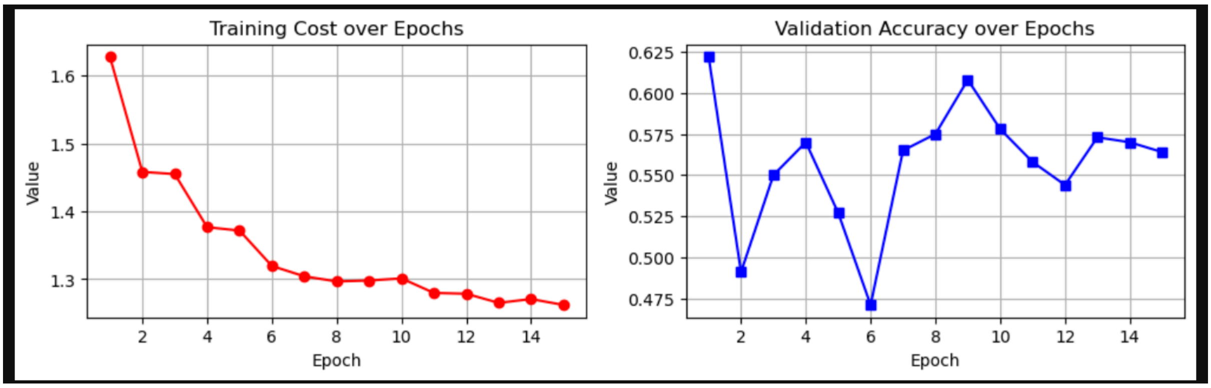


True label \ Predicted label	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
Class 0	0.46	0.15	0.24	0.029	0.096	0.029	0.0045
Class 1	0.11	0.52	0.16	0.057	0.042	0.083	0.03
Class 2	0.081	0.081	0.48	0.085	0.12	0.14	0.018
Class 3	0.051	0.051	0.24	0.5	0.019	0.12	0.019
Class 4	0.099	0.057	0.2	0.0099	0.43	0.18	0.026
Class 5	0.019	0.055	0.1	0.04	0.098	0.61	0.072
Class 6	0	0.034	0.057	0.034	0.029	0.26	0.59

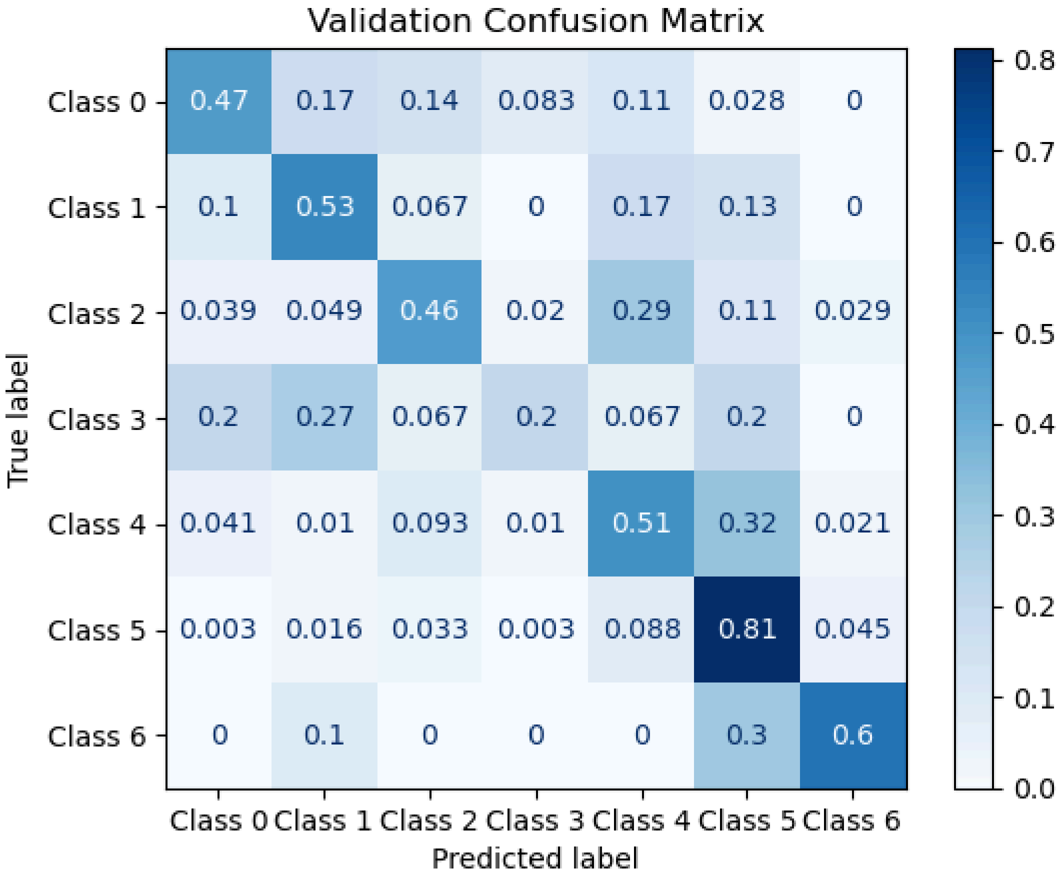


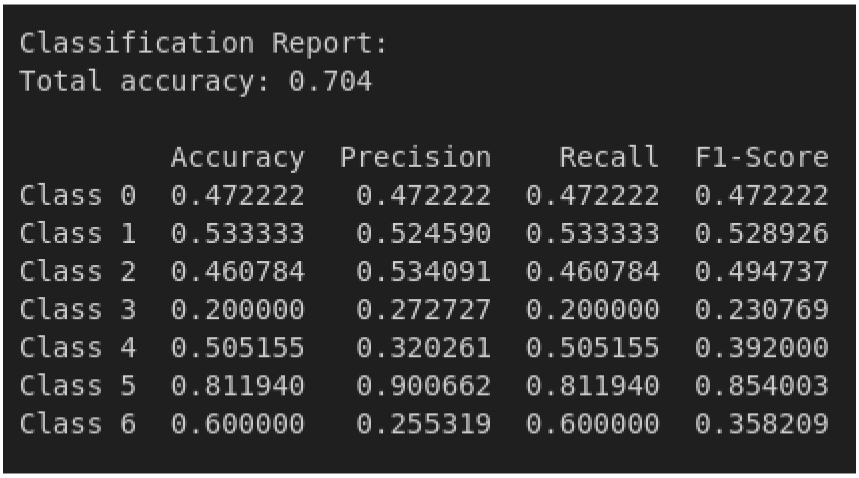
```
Classification Report:
Total accuracy: 0.5354236951776146

         Accuracy  Precision    Recall  F1-Score
Class 0  0.455357   0.448352  0.455357  0.451827
Class 1  0.516892   0.498371  0.516892  0.507463
Class 2  0.478134   0.458955  0.478134  0.468348
Class 3  0.495370   0.319403  0.495370  0.388385
Class 4  0.432806   0.350400  0.432806  0.387268
Class 5  0.611192   0.777301  0.611192  0.684310
Class 6  0.588571   0.334416  0.588571  0.426501
```

Plots:

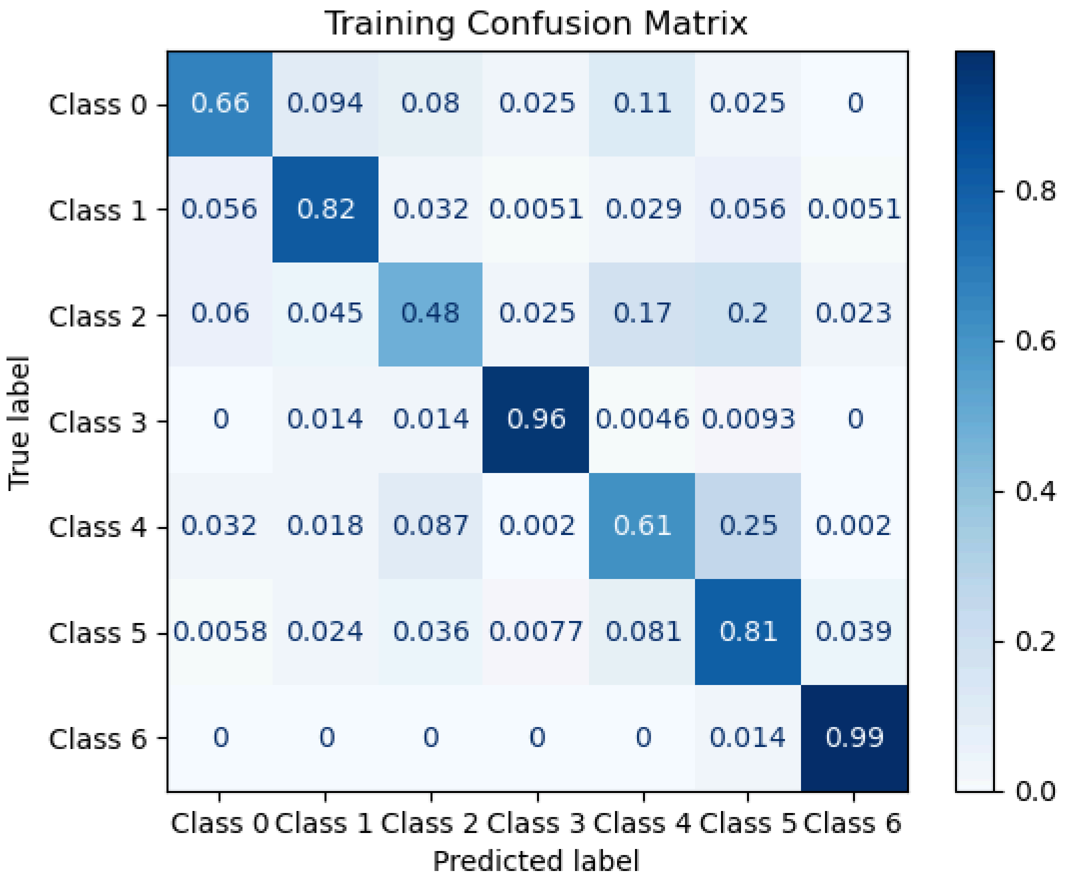


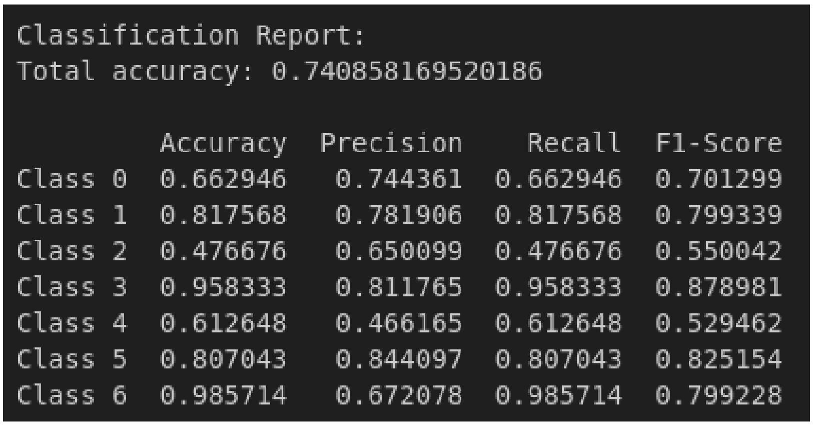
Fine-tuning





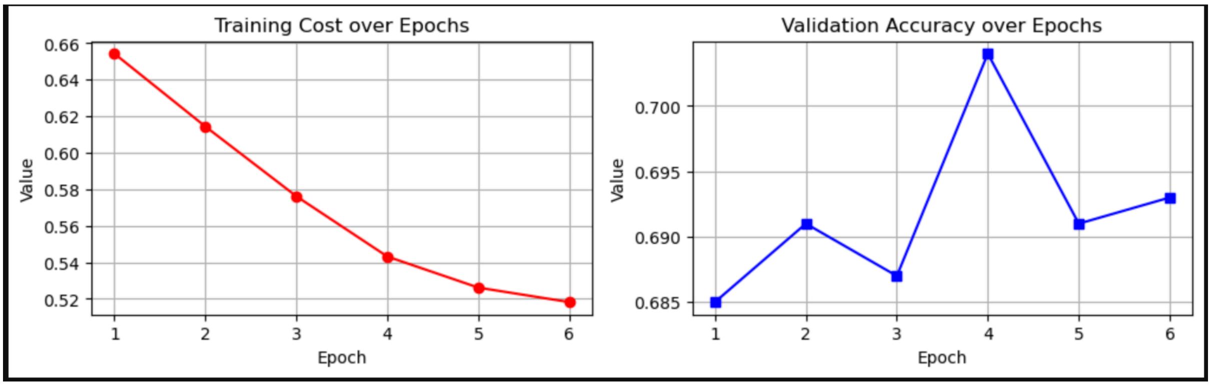
```
Classification Report:
Total accuracy: 0.704

         Accuracy  Precision    Recall  F1-Score
Class 0  0.472222   0.472222  0.472222  0.472222
Class 1  0.533333   0.524590  0.533333  0.528926
Class 2  0.460784   0.534091  0.460784  0.494737
Class 3  0.200000   0.272727  0.200000  0.230769
Class 4  0.505155   0.320261  0.505155  0.392000
Class 5  0.811940   0.900662  0.811940  0.854003
Class 6  0.600000   0.255319  0.600000  0.358209
```





```
Classification Report:
Total accuracy: 0.740858169520186

         Accuracy  Precision    Recall  F1-Score
Class 0  0.662946   0.744361  0.662946  0.701299
Class 1  0.817568   0.781906  0.817568  0.799339
Class 2  0.476676   0.650099  0.476676  0.550042
Class 3  0.958333   0.811765  0.958333  0.878981
Class 4  0.612648   0.466165  0.612648  0.529462
Class 5  0.807043   0.844097  0.807043  0.825154
Class 6  0.985714   0.672078  0.985714  0.799228
```

Plots:



For MobileNet-V3 Small as we can see that the model was having high training loss and lower validation accuracy as compared to large model as it has a lightweight architecture which resulted in lesser ability to detect complex features.

Conclusion

This project successfully explored the application of deep learning models, particularly the MobileNetV3 family, for the classification of skin cancer lesions using grayscale dermoscopic images. By leveraging pretrained architectures through transfer learning and fine-tuning, we were able to overcome limitations associated with small medical datasets and achieve meaningful results in a complex, real-world problem domain.

Two distinct models were developed and analyzed:

- **MobileNetV3-Large**, which offered superior classification performance due to its deeper and wider architecture.
- **MobileNetV3-Small**, which, while computationally more efficient, still delivered competitive results after careful fine-tuning.

Both models were trained using robust pipelines that accounted for:

- **Class imbalance** through weighted loss functions,
- **Data augmentation** to increase generalizability,
- **Custom model heads** tailored to the 7-class classification task, and
- **Learning rate schedulers** to enhance convergence stability.

Among the two models tested, **MobileNetV3-Large achieved higher validation accuracy** compared to its smaller counterpart. Its deeper and wider architecture allowed it to learn more complex and discriminative features, which proved beneficial for handling the subtle differences between various skin lesion classes. This makes it especially suitable for tasks where classification accuracy is critical, even if it comes at a slightly higher computational cost.

One of the most important observations was the significant improvement in both models after fine-tuning, especially for underrepresented classes such as dermatofibroma and vascular lesions. Fine-tuning not only helped reduce training loss and improve validation accuracy but also balanced performance across all lesion types, as reflected in the confusion matrices and class-wise accuracy metrics.

Ultimately, this project demonstrated the potential of MobileNet architectures for efficient and accurate skin cancer detection.