Combining Efficient Feature Extraction with Capsule Hierarchies for Skin Cancer Recognition

Abdulrahman Alamoudi
Bahcesehir University
abdulrahman.alamoudi@bahcesehir.edu.tr
GitHub: WajeehAlamoudi/skinlite

Abstract

*This article demonstrates the application and utilization of an advanced Capsule Network combined with a backbone lightweight model in medical image classification, providing a benchmark on the **ISIC (International Skin Imaging Collaboration)** dataset. The experiments were conducted as part of Task 3 of the ISIC 2018 Challenge dataset [3].*

*The proposed model leverages pre-trained MobileNetV2 convolutional feature extraction blocks to capture multi-level semantic features from skin images. These features are then processed by Capsule Network blocks at three clinical hierarchical levels: **Coarse** (malignant vs. benign), **Middle** (melanoma vs. non-melanoma vs. others), and **Fine** (specific lesion categories). Unlike conventional CNN-based models that treat classes as flat and mutually exclusive, the architecture—**H-CapsNet**—adopts a coarse-to-fine capsule-based design that explicitly encodes class hierarchies, enabling improved interpretability and robustness.*

In addition, this work introduces effective strategies for addressing dataset imbalance, including a custom dataset loader, class-aware transformations, and augmentation techniques. All experiments and modules were implemented using the PyTorch framework. With this foundation in place, the remainder of this article will explore the system architecture, methodology, training process, and performance evaluation in detail.

1. Introduction

Image classification is a challenging task in computer vision filed, typically when applied into real-world problems. The complexity arises not only from the technical aspects of design and train robust models but, also from intensive computation power consumption and lack of general solutions to address the issue. Applying methods to real-world data is crucial, as it shows the application in solving real problems and improving lives.

Skin cancer, one of the most prevalent and dangerous forms of cancer, underscores the importance of such advancements. According to recent statistics, over 150,000 new cases of melanoma are diagnosed annually worldwide [6], highlighting the significant impact of this disease and the urgent need for reliable diagnostic solutions.

Traditional Convolutional Neural Networks (CNNs) have achieved state-of-the-art results in medical image classification tasks, including skin lesion analysis, as demonstrated by top-performing models on the ISIC 2018 Challenge leaderboard. However, despite their success, CNNs inherently suffer from architectural limitations. They operate using scalar feature activations and rely heavily on pooling operations for spatial reduction, which leads to the loss of precise positional information. This makes it difficult for CNNs to model complex part-to-whole relationships within visual patterns—an essential requirement in medical imaging, where fine-grained details often determine the difference between classes. Moreover, CNNs typically treat all categories as flat and independent, ignoring hierarchical or taxonomic relationships that exist between medical conditions, such as the distinction between benign and malignant lesions and their respective subtypes.

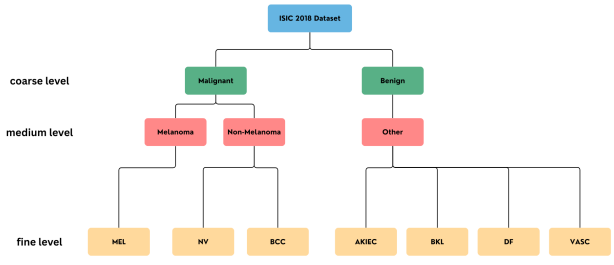

Figure 1. Diagram visualization of the hierarchical multi-level structure taxonomy used in skin lesion classification.

To overcome these challenges, I utilize a H-Caps method [7], making Hierarchical Capsule Network that integrates *MobileNetV2* as a lightweight and efficient feature extractor with a three-level capsule architecture to learn hierarchical semantic features. The presented model captures the coarse-to-fine structure of clinical taxonomy as shown in Figure 1. (e.g., Malignant → Melanoma → Subtype) and leverages dynamic margin losses across hierarchy levels. This allows the network to reason at multiple levels of abstraction, increasing both classification accuracy and interpretability.

1.1. Insight over dataset

The ISIC 2018 Task 3 dataset [3] is a publicly available collection designed for automated diagnosis of skin lesions. It contains high-resolution 450 x 600 , 3 channel ".jpg" format dermoscopic images across 7 diagnostic categories, enabling multi-class classification tasks as shown in Figure 2.

The 7-classes is defined as melanoma (MEL), melanocytic nevus (NV), basal cell carcinoma (BCC), actinic keratosis (AK), benign keratosis (BKL), dermatofibroma (DF), and vascular lesion (VASC).

The dataset show massive imbalance among classes, where the "NV" take over the majority of *train set* with approximately 67% and "AKIEC", "VASC", "DF" appears as minority in training set with 6% combined.

Table 1 summarizes the total number of images and their class-wise distribution across the training, validation, and test sets.

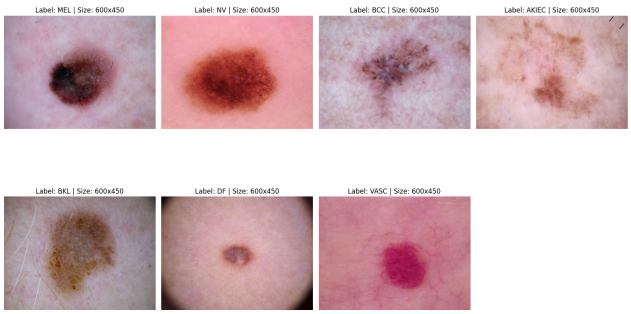

Figure 2. Example images for all classes from the ISIC 2018 dataset.

The imbalance across classes poses a significant challenge in training deep learning models, particularly for minority classes like "DF", "VASC", and "AKIEC", which have very few instances. Without proper handling, models tend to be biased toward the majority class "NV", leading to poor generalization and low recall on critical but rare conditions like melanoma.

To address this, class-balancing techniques such as weighted loss functions, oversampling of minority classes, and advanced augmentation (e.g., elastic deformations, color jittering, random cropping) are often employed. These approaches can help alleviate the skewed learning and promote a more equitable representation of each class during training.

Class	Train	Val	Test
NV	6705	123	906
MEL	1113	21	171
BKL	1099	22	216
BCC	514	15	93
AKIEC	327	8	43
VASC	142	3	35
DF	115	1	44
Total	**10015**	**193**	**1508**

Table 1. Class-wise distribution of images in the ISIC 2018 Task 3. dataset.

2. Data Pre-processing and Augmentation

To gain a consistent and generalized model, series of of preprocessing and argumentation techniques were applied to the ISIC 2018 dataset.

All images were first resized to a fixed resolution of $H \times W =$ `(IMAGE_SIZE, 1.25 × IMAGE_SIZE)` to ensure consistent aspect of ratio, followed by a centered crop to ensure a uniform square input size of `IMAGE_SIZE × IMAGE_SIZE`. This preserves important lesion details while reducing unnecessary background information. In my case the `IMAGE_SIZE = 224` since it is proper to been fed into *mobileNetV2* blocks. Then it been applied the following augmentations probabilistically during training:

- **Random Horizontal Flip**: Introduces horizontal invariance.
- **Random Vertical Flip**: Captures vertical symmetry and structural variations.
- **Random Fixed Rotation**: A custom transform that rotates the image by one of the fixed angles in $\{0°, 90°, 180°, 270°\}$ with a probability $p = 0.5$.
- **Color Jittering**: Introduced variation in brightness, contrast, and saturation within the range [0.8, 1.1].
- **Normalizing**: Normalize the image's pixels values to be among [0, 1], for simplicity and reducing the computation consuming.

These augmentations simulate natural variations in lesion orientation, color, and illumination, thereby helping the model become more robust to real-world scenarios as shown in Figure 3.

On the other hand, for validation and testing, it been applied only deterministic transformations: resizing and center cropping followed be *Tensor* normalization. This ensures consistency during evaluation and avoids introducing noise that may affect performance measurement.

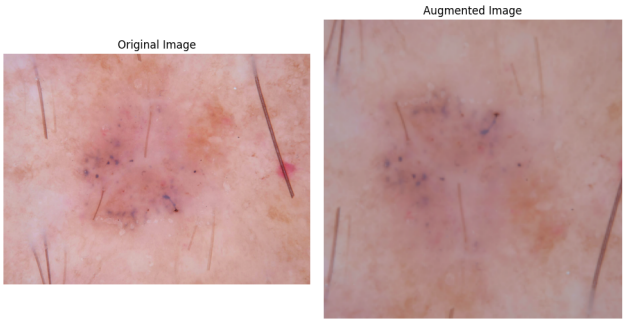

Figure 3. Visualization of the original image (left) and after applying the training augmentation pipeline (right).

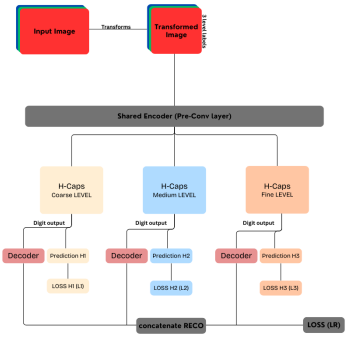

Figure 4. Full Model Architecture.

3. Hierarchical Dataset Loader Design

To support the hierarchical multi-level classification required for H-CapsNet, I implemented a custom PyTorch dataset loader named `HCAPS_ISICDataset`. This loader is built to process the ISIC 2018 Task 3 dataset and output three hierarchical label levels: coarse, medium, and fine.

3.1. Class Resampling Strategy

During training, class imbalance is handled through a configurable `CLASS_MULTIPLIER` dictionary. For each class:

- A **positive** multiplier (`>1`) duplicates the class samples.
- A **negative** multiplier (`-n`) reduces the number of samples by selecting 1 out of every `n`.
- A multiplier of `0` retains the class sample count as-is.

For example, setting `NV = -2` in `CLASS_MULTIPLIER` will reduce the number of "NV" images by half. This resampling ensures a more balanced class distribution without altering the original labels.

3.2. Hierarchical Label Mapping

The dataset returns three levels of supervision:

- **Coarse-level label (Level 1)**: A binary mapping indicating benign vs malignant.
- **Medium-level label (Level 2)**: Groups melanoma, non-melanoma, others.
- **Fine-grained label (Level 3)**: The original 7-class ground truth from the ISIC dataset.

For example, a lesion labeled as *BCC* will be:

- Coarse: `0 (Malignant)`
- Medium: `1 (Non-melanoma)`
- Fine: `2 (BCC)`

This hierarchical structure enables the network to learn from high-level to fine-grained decisions, mimicking the diagnostic reasoning of clinical dermatologists.

4. Model Architecture

Inspired by the human diagnostic process, our model adopts a three-level prediction structure aligned with the hierarchical labels described earlier: coarse (benign vs malignant), medium (disease family), and fine (specific diagnosis).

The model processes input images through a structured pipeline that mirrors hierarchical clinical diagnosis as shown in Figure 4. The pipeline includes the following stages:

- **Input Image and Transformations:** The input dermoscopic image is first preprocessed using transforms discussed above in section 2, then the transformed image will pass into *Shared Encoder*.
- **Shared Encoder:** A convolutional block consist of twice `Conv2d(filters=64, kernal_size=3, stride=1, padding=1)` → `BatchNorm2d` → `ReLu()`, allows the model to process the fed images as a features map directly and waste no passing through next architectures.
- **Hierarchical Capsule Layers:** Is the main part of the architecture, consist of three parallel *H-Caps* blocks, where the input comes from *Shared Encoder* and been fed into *H1, H2, H3* blocks.
  - **H1 - Coarse Level:** Binary prediction (Benign vs Malignant)
  - **H2 - Medium Level:** 3-class prediction (Melanoma, Non-Melanoma, Other)
  - **H3 - Fine Level:** Full 7-class classification (MEL, NV, BCC, AKIEC, BKL, DF, VASC)

each block will make it own level predication, later the total loss $\mathcal{L}_T$ will be counted as weighted summation of classification reconstruction loss across label-tree $\mathcal{L}_R$ with classification loss given as $\mathcal{L}_C$, the final loss function will be as Equation 1.

$$\mathcal{L}_T = \lambda \mathcal{L}_R + \mathcal{L}_C \tag{1}$$

4.1. H-Caps Blocks

A structured blocks contains of *feature extractor, primary capsule, digit capsule* and integrated common decoder shown in Figure 5 and Listing 1.

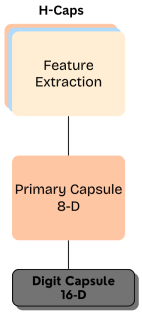

Figure 5. H-Caps Block

The feature extractor is built from selected bottleneck layers of *MobileNetV2*, allowing each hierarchical block to learn level-specific spatial features. For example, in the coarse-level H-Caps block, layers `[1:4]` of MobileNetV2 are used to extract the input feature map of shape $[B, 24, 112, 112]$. Then The feature map is passed into a primary capsule convolution layer that outputs 32 capsules, each with an 8-dimensional pose vector. The result tensor shape $[B, H_{\text{out}} \times W_{\text{out}} \times \text{num\_capsules}, 8]$ represents the flattened spatial locations and capsule count. Digit Capsule Layer is a dynamic routing module takes the output from the primary capsules and generates one capsule per class, each of dimension 16. In the coarse block, this results in $[B, 2, 16]$ for binary classification (benign vs malignant). Then shared Decoder reconstructs the input image from the capsule outputs to encourage rich internal representations. This reconstruction acts as an auxiliary task and contributes to the total loss via $\mathcal{L}_R$ in equation 1.

This modular capsule structure allows each H-Caps block to specialize in a different level of abstraction — coarse, medium, or fine — while sharing a consistent architecture.

```
class MCoarseFeatureExtractor(nn.Module):
    def __init__(self):
        super().__init__()

        mobilenet =
        models.mobilenet_v2(pretrained=False)
        self.low_level  =
        nn.Sequential(*list
        (mobilenet.features.children())[1:4])
        self.adapter =
        nn.Conv2d(64, 32, kernel_size=1)

    def forward(self, x):
        x = self.adapter(x)
        x = self.low_level(x)
        return x

class MCapsuleDecoder(nn.Module): ...

class MPrimaryCaps(nn.Module): ...

class MDigitCaps(nn.Module): ...

class H1CapsCoarse(nn.Module):
    def __init__(self, num_classes):
        super(H1CapsCoarse, self).__init__()
        self.feature_extractor =
            MCoarseFeatureExtractor()
        self.primary_caps = MPrimaryCaps(
        in_channels,
        capsule_dim,
        num_capsules,
        kernel_size,
        stride
        )
        self.digit_caps = MDigitCaps(
        input_capsules,
        input_dim,
        num_classes,
        output_dim,
        routing_iter
        )
        self.decoder = MCapsuleDecoder(
        input_dim,
        output_shape
        )

    def forward(self, x_shared):
        x = self.feature_extractor(x_shared)
        x = self.primary_caps(x)

        digit_caps_out = self.digit_caps(x)
        recon = self.decoder(digit_caps_out)

        return digit_caps_out, recon
```

Listing 1. Coarse-level Capsule Network Block (H1CapsCoarse)

5. Loss Function Design

The total loss for H-CapsNet is composed of two main components: a hierarchical classification loss $\mathcal{L}_C$ and a reconstruction loss $\mathcal{L}_R$, balanced using a hyperparameter λ. The total loss is defined as:

$$\mathcal{L}_T = \lambda \mathcal{L}_R + \sum_{j=1}^{N} \gamma_j \mathcal{L}_{M_j} \tag{2}$$

Where:

- $\mathcal{L}_R$ is the pixel-wise reconstruction loss (mean squared error).
- $\mathcal{L}_{M_j}$ is the margin loss for level j of the hierarchy (coarse, medium, fine).
- γ_j is a level-specific dynamic weight for loss balancing.
- λ is a fixed scalar weight controlling reconstruction influence.

5.1. Margin Loss (per level)

For each output capsule set (e.g., digit1 for coarse), margin loss is calculated as:

$$\mathcal{L}_M = T_i \cdot \max(0, m^+ - \|v_i\|)^2 + \eta \cdot (1 - T_i) \cdot \max(0, \|v_i\| - m^-)^2$$

Where:

- v_i is the output vector of capsule i.
- T_i is a one-hot encoded label for class i.
- m^+, m^- are positive and negative margins (e.g., 0.9 and 0.1).
- η controls down-weighting of negative terms (default: 0.5).

Optionally, per-class weights can be applied to reduce the impact of class imbalance by weighting the contribution of each sample according to its label.

5.2. Dynamic Weighting γ

To adaptively emphasize harder classes or levels, we define γ weights dynamically using both class frequency and prediction accuracy:

$$\gamma_j = (1 - \lambda) \cdot \frac{(1 - \text{acc}_j) \cdot \rho_j}{\sum_{k=1}^{N} (1 - \text{acc}_k) \cdot \rho_k}$$

Where:

- ρ_j is the class frequency proportion for level j.
- acc_j is the current accuracy at level j.

The final loss combines the hierarchical margin loss (hinge loss) with the reconstruction loss. This combination encourages the model to learn discriminative features at each hierarchy level, while also enforcing the quality of internal representations through reconstruction.

6. Optimizer and Learning Rate Scheduling

I used the AdamW optimizer for all H-CapsNet experiments due to its robustness in handling sparse gradients and regularization.

To improve generalization and training stability, I implemented a custom learning rate scheduler defined by the following rule:

$$\eta = \hat{\eta} \cdot \beta^{\max(0, E - \kappa)} \tag{3}$$

Where:

- $\hat{\eta}$ is the initial learning rate
- β is the decay rate ($\beta \in (0, 1)$)
- κ is the warm-up threshold (number of epochs with constant LR)
- E is the current epoch

The learning rate remains constant at $\hat{\eta}$ for the first κ epochs and then decays exponentially after epoch $E > \kappa$. This allows the model to first converge with a stable rate before gradually reducing the step size for fine-tuning.

In our experiments, we used the following hyperparameters:

- Initial learning rate: $\hat{\eta} = 0.001$
- Decay rate: $\beta = 0.95$
- Warm-up threshold: $\kappa = 10$

As during the training loop it check the current number of the epoch then exponentially decaying applied to learning rate given above in Equation 3.

7. Training Configuration

The entire training pipeline was implemented in a modular and well-structured format. A dedicated configuration file named `setting.py` was created to centralize all hyperparameters and control parameters such as learning rate, batch size, epoch count, decay factors, model architecture options, and classes multiplayer and more. "visit the github repository".

7.1. Training Environment

Training was conducted on Google Colab using an NVIDIA A100 GPU. The project directory was cloned from a working Git-based environment to ensure reproducibility. With a batch size of 32, each training epoch required approximately 2 minutes and 30 seconds to complete. The dataset was already split into training, validation, and test sets using predefined ISIC 2018 metadata.

To calculate the number of iterations per epoch:

$$\text{iterations} = \left\lfloor \frac{\text{total training samples}}{\text{batch size}} \right\rfloor$$

For example, with 10,015 training images and a batch size of 32, each epoch consists of approximately 313 steps.

7.2. Training Design and Tracking

The training loop was designed to simultaneously perform:
- Training with forward and backward propagation
- Validation after each epoch using integrated loss computation
- Saving model state after improvement in macro F1-score
- Updating a performance history dictionary for later analysis

The training process outputs three files:
- `model.pt`: the entire model checkpoint
- `model.pth`: state dictionary for PyTorch loading
- `history.pkl`: serialized training history including loss, accuracy, and F1-score per epoch

7.3. Early Stopping and Overfitting Control

A patience-based early stopping mechanism was implemented to avoid overfitting. If the validation macro F1-score did not improve for a predefined number of consecutive epochs (patience threshold), training was halted early. This approach ensured computational efficiency and better generalization.

This setup allowed for fast iteration, robust experiment tracking, and simplified visualization and analysis post-training.

8. Model Wrap-Up

To train the proposed H-CapsNet model, a complete pipeline was implemented, covering data loading, forward propagation, loss computation, performance evaluation, dynamic learning rate scheduling, early stopping, and model checkpointing.

8.1. Pipeline Overview

The training process begins with loading the dataset using the `HCAPS_ISICDataset` class for both the training and validation sets. Data is loaded in batches using PyTorch's `DataLoader`, with a batch size of 32. The training dataset is augmented, while the validation set is preprocessed deterministically.

A hierarchical capsule network is initialized and trained using the Adam optimizer, and hyperparameters are loaded from a centralized configuration file `setting.py`. The model is trained for a maximum of N epochs (e.g., 50), with exponential learning rate decay applied after a warm-up threshold $\kappa = 10$.

8.2. Training and Validation Flow

Each epoch includes the following steps:
- Compute updated learning rate based on epoch and apply it to the optimizer.
- Train the model on the training set and calculate the hierarchical classification loss $\mathcal{L}_C$ and reconstruction loss $\mathcal{L}_R$.
- Evaluate on the validation set to compute accuracy at all three levels and macro F1-score at the fine level.
- Save the model weights and architecture if the macro F1-score improves.
- Apply early stopping if no improvement is observed over a fixed number of epochs (patience).
- Log metrics into a dictionary for training history.

All metrics such as loss, accuracy, F1-score, and dynamic γ_j weights are stored per epoch in a `history` dictionary. This dictionary is saved as a `.pkl` file at each epoch and used for post-training analysis and visualization.

8.3. Output Artifacts

Upon completion (or early stopping), the following files are produced:
- **`model.pt`**: the full PyTorch model for direct reuse.
- **`model.pth`**: the model state dictionary for fine-tuning or loading into new instances.
- **`history.pkl`**: serialized training history including training/validation loss, accuracy, and macro F1-score.

Each file is named automatically using the model type and a unique timestamp to ensure reproducibility and prevent overwriting.

This structured training pipeline ensures reproducibility, modularity, and ease of experimentation.

9. Evaluation Metrics

To assess the performance of the H-CapsNet model, I evaluate its predictions at three hierarchical levels: coarse, medium, and fine. The following metrics are computed for each level:

9.1. Accuracy

Accuracy is computed as the ratio of correctly predicted samples to the total number of samples. It gives a general sense of model correctness, but it may be misleading in the presence of class imbalance.

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(y_i = \hat{y}_i)$$

Where y_i is the ground truth and $\hat{y}_i$ is the predicted label.

9.2. Macro-Averaged F1-Score

To better capture performance across imbalanced classes (particularly in fine-grained classification), I use the macro F1-score, which averages the F1-scores across all classes equally—regardless of how many samples each class has.

$$\text{Macro F1} = \frac{1}{C}\sum_{c=1}^{C} \frac{2 \cdot \text{Precision}_c \cdot \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c}$$

This metric ensures that rare classes (e.g., *DF*, *VASC*) have a meaningful impact on the overall score and are not overshadowed by dominant classes (e.g., *NV*).

9.3. Precision and Recall

For further analysis, I optionally compute:

- **Precision:** Measures the proportion of correctly predicted positive samples.
- **Recall:** Measures the proportion of actual positive samples correctly predicted.

9.4. Confusion Matrix

A confusion matrix is used to visualize how well the model distinguishes between individual classes at the fine level. It reveals which classes are most often confused and is especially useful for error analysis in medical datasets where mis-classification may have critical consequences.

9.5. Level-wise Evaluation

I apply the above metrics to all three prediction levels:

- **Level 1 (Coarse):** Benign vs Malignant
- **Level 2 (Medium):** Melanoma, Carcinoma, Other
- **Level 3 (Fine):** 7-class skin lesion classification

During training, the macro F1-score at the fine level is used to monitor validation performance and determine model check-pointing and early stopping.

10. Results and Discussion

In this section, I present and analyze the final performance of the proposed H-CapsNet model on the ISIC 2018 test set. Evaluation was conducted at the fine-grained level (7 classes), and metrics include accuracy, macro F1, and detailed per-class statistics.

10.1. Fine-Level Results

The model achieved an overall accuracy of 81.1% and a macro-averaged F1-score of 72.3%. These results confirm the model's strong ability to generalize, even across highly imbalanced lesion classes.

Metric	Value
Accuracy	81.10%
Macro F1-score	72.26%
Micro F1-score	81.10%
Weighted F1-score	81.27%

Table 2. Test performance of H-CapsNet on ISIC 2018 fine-level classification.

10.2. Per-Class Analysis

Table 3 summarizes the precision, recall, and F1-score per class. The model performed best on dominant classes such as *NV* (F1: 0.89) and *VASC* (F1: 0.81), while performance was lower on minority and harder-to-distinguish classes such as *AKIEC* (F1: 0.62) and *DF* (F1: 0.66).

Class	Precision	Recall	F1-score
MEL	0.59	0.68	0.63
NV	0.89	0.89	0.89
BCC	0.72	0.68	0.70
AKIEC	0.53	0.74	0.62
BKL	0.77	0.72	0.74
DF	0.78	0.57	0.66
VASC	0.90	0.74	0.81

Table 3. Per-class performance of H-CapsNet on the test set.

Confusion Matrix:

	MEL	NV	BCC	AKIEC	BKL	DF	VASC
MEL	116	37	2	2	11	1	2
NV	57	805	11	6	25	1	1
BCC	8	7	63	9	5	1	0
AKIEC	0	6	1	32	3	1	0
BKL	15	29	7	7	156	2	0
DF	0	11	2	3	3	25	0
VASC	1	5	1	1	0	1	26

Figure 6. Confusion matrix for fine-level predictions on the test set.

10.3. Confusion Matrix and Misclassifications

The confusion matrix (Figure 6) reveals common misclassification patterns. The model frequently confused:

- *MEL* and *NV* — due to visual similarity in melanocytic structures
- *BCC* and *AKIEC* — both malignant and texturally complex
- *DF* and *NV* — limited DF samples impacted recall

10.4. Level-wise Accuracy

Table 4 shows the model's classification accuracy across the three hierarchical levels on the ISIC 2018 test set. As expected, the model performs best at the medium level (disease grouping), while accuracy drops slightly at the coarse level due to its binary structure being influenced by class imbalance (e.g., benign NV vs malignant MEL/BCC/AKIEC).

Interestingly, Level 2 shows higher accuracy than Level 1. This may be attributed to the model learning better inter-class boundaries at the group level (e.g., melanoma vs non-melanoma vs other), while Level 1 suffers from an

Hierarchy Level	Accuracy
Level 1 (Benign vs Malignant)	77.59%
Level 2 (Disease Grouping)	84.88%
Level 3 (Fine 7-Class)	81.10%

Table 4. Test accuracy of H-CapsNet at each hierarchy level.

imbalanced distribution of benign and malignant cases, particularly due to the dominance of the *NV* class in the benign group.

11. Conclusion and Future Work

In this work, I developed a customized Hierarchical Capsule Network (H-CapsNet) architecture for skin lesion classification using the ISIC 2018 dataset. The contribution lies in combining a lightweight convolutional feature extractor with a hierarchical capsule structure designed to classify lesions at three levels of abstraction: coarse (benign vs malignant), medium (disease group), and fine (7 specific classes).

To address the challenge of extreme class imbalance in the dataset, I introduced a tailored oversampling strategy using a configurable per-class multiplier mechanism. This custom loader dynamically expands or reduces sample frequency for each class, ensuring more balanced training without manual duplication. Additionally, I implemented a dynamic loss weighting system that adjusts supervision at each level based on real-time accuracy, helping the model focus on underperforming hierarchies.

The final architecture integrates multi-level capsule outputs, shared decoders, and reconstruction loss in a unified pipeline, achieving competitive performance while maintaining interpretability. The model reached 81.1% accuracy and 72.3% macro F1 at the fine level, with strong results at medium and coarse levels as well. These contributions demonstrate how lightweight design, hierarchical capsules, and custom training strategies can be effectively combined for real-world medical image classification.

Future Work

Several improvements can be explored in future work:

- Fine-tuning the hyperparameters, including which convolutional layers are used for feature extraction at each hierarchy level.
- Experimenting with attention-based dynamic routing or capsule dropout mechanisms to improve capsule activation.
- Extending the evaluation to larger and more diverse datasets such as ISIC 2019 and ISIC 2020 to validate scalability.
- Assessing model robustness and generalization under domain shifts, including testing on real-world clinical data.

Acknowledgments

This work draws inspiration from HI-MViT, a lightweight model for explainable skin disease classification. As suggested by Ding et al. [5], "the use of capsule networks could enhance the preservation of spatial relationships in learned features, offering promising improvements for image classification tasks."

The hierarchical capsule network (H-CapsNet) proposed by Noor and Robles-Kelly [7] served as the architectural foundation for this research. Their methodology demonstrates the natural ability of capsule networks to model hierarchical relationships, which is leveraged here for multi-level feature extraction and classification.

I also thank the ISIC team [3] for providing a publicly available, clinically annotated dermoscopic image dataset. In addition, I acknowledge the creators of the open-source Capsule Network reference code [2], as well as the PyTorch documentation [1] and modular tutorials [4], which facilitated rapid experimentation and prototyping in this project.

References

[1] Pytorch documentation. `https://pytorch.org/docs/stable/index.html`, 2024. Accessed: 2025-04-26.

[2] Nikhil Barhate. Capsule networks in pytorch. `https://github.com/nikhilbarhate99/capsule-net-pytorch`, 2020. Accessed: 2025-04-26.

[3] Noel Codella, Veronica Rotemberg, Philipp Tschandl, M. Emre Celebi, Stephen Dusza, David Gutman, Brian Helba, Aadi Kalloo, Konstantinos Liopyris, Michael Marchetti, et al. Skin lesion analysis toward melanoma detection 2018: A challenge hosted by the international skin imaging collaboration (isic). *arXiv preprint arXiv:1902.03368*, 2018.

[4] PyTorch Community. Design patterns: Modular class structure and routing in pytorch. `https://pytorch.org/tutorials/`, 2023. Accessed: 2025-04-26.

[5] Y. Ding, Z. Yi, M. Li, et al. Hi-mvit: A lightweight model for explainable skin disease classification based on modified mobilevit. *DIGITAL HEALTH*, 9, 2023.

[6] World Cancer Research Fund International. Melanoma skin cancer statistics. 2020.

[7] Khondaker Tasrif Noor and Antonio Robles-Kelly. H-capsnet: A capsule network for hierarchical image classification. *Pattern Recognition*, 147:110135, 2024.