**Software Requirement Specification (SRS)**

**For Skin Disease Recognition Model using Machine Learning**

**Version:** 1.0  
**Tool Used:** Visual Studio Code  
**Dataset Source:** Kaggle

**1. Introduction**

**1.1 Purpose**

The purpose of this project is to develop a **Skin Disease Recognition Model** using **machine learning** to detect and classify different types of skin diseases from images. The system uses **image datasets from Kaggle**, trains a **deep learning model**, and predicts the type of disease based on the uploaded skin image.  
The model aims to help dermatologists, medical professionals, and patients by providing quick, accurate, and automated detection.

**1.2 Scope**

* The system takes **skin lesion images** as input.
* Uses a **CNN-based deep learning model** for classification.
* Dataset will be collected from **Kaggle**.
* The model will be trained, validated, and tested using **Visual Studio Code**.
* Future deployment can be done via a **web app** or **desktop interface**.
* Example diseases:
  + Melanoma
  + Psoriasis
  + Eczema
  + Acne
  + Ringworm
  + Normal skin

**Benefits:**

* Assists dermatologists in faster decision-making.
* Provides early detection and better treatment options.
* Saves time and reduces manual analysis.

**1.3 Definitions, Acronyms, and Abbreviations**

| **Term** | **Description** |
| --- | --- |
| CNN | Convolutional Neural Network |
| ML | Machine Learning |
| DL | Deep Learning |
| Kaggle | Platform for datasets and competitions |
| Accuracy | Percentage of correctly classified images |

**1.4 References**

* Kaggle Dataset: *Provide the dataset URL*
* IEEE 830 SRS Documentation Standard
* TensorFlow & Keras Official Documentation
* Visual Studio Code Official Docs

**1.5 Overview**

The SRS outlines:

* Functional and non-functional requirements
* Dataset details
* Model architecture
* Training and testing methodology
* Hardware/software requirements

**2. Overall Description**

**2.1 Product Perspective**

* The model will be trained using **Python** and **TensorFlow/Keras** in **Visual Studio Code**.
* Dataset from **Kaggle** will be preprocessed, augmented, and split into **training**, **validation**, and **testing** sets.
* Final trained model will classify images into disease categories.

**System Flow:**

1. Input skin image →
2. Preprocessing →
3. CNN model →
4. Feature extraction →
5. Prediction →
6. Output (Disease name + confidence score)

**2.2 Product Functions**

* **Image Upload**: Accepts images from the dataset or user input.
* **Preprocessing**: Resizes, normalizes, and augments images.
* **Training**: Uses CNN to learn disease patterns.
* **Testing**: Evaluates the model on unseen data.
* **Prediction**: Returns the most likely disease class.
* **Accuracy Visualization**: Displays loss and accuracy graphs.

**2.3 User Classes and Characteristics**

| **User Type** | **Description** | **Technical Knowledge** |
| --- | --- | --- |
| Researcher | Uses the model for academic purposes | Medium |
| Dermatologist | Uses the model for diagnosis support | Low |
| Patient | Uses for personal analysis | Very Low |

**2.4 Operating Environment**

* **Platform:** Visual Studio Code
* **Language:** Python
* **Frameworks:** TensorFlow, Keras, OpenCV
* **OS:** Windows 10/11 or Linux
* **Dataset Source:** Kaggle
* **Training Device:** Laptop/PC with GPU preferred

**2.5 Design and Implementation Constraints**

* Requires a high-quality dataset.
* High computation power required for CNN training.
* Requires proper preprocessing to avoid overfitting.

**2.6 Assumptions and Dependencies**

* Dataset from Kaggle is publicly available.
* Python libraries are compatible.
* Sufficient storage and memory available.

**3. Specific Requirements**

**3.1 Functional Requirements**

| **ID** | **Requirement** | **Priority** |
| --- | --- | --- |
| FR-1 | Upload skin image | High |
| FR-2 | Preprocess images (resize, normalize) | High |
| FR-3 | Train CNN model | High |
| FR-4 | Predict skin disease | High |
| FR-5 | Display accuracy and loss graphs | Medium |
| FR-6 | Export trained model | Medium |

**3.2 Non-Functional Requirements**

| **ID** | **Requirement** | **Type** |
| --- | --- | --- |
| NFR-1 | Accuracy ≥ 90% | Performance |
| NFR-2 | Training time ≤ 3 hours | Performance |
| NFR-3 | User-friendly interface | Usability |
| NFR-4 | Should work offline after model training | Reliability |

**3.3 Dataset Details**

* **Source:** Kaggle
* **Size:** Depends on dataset (e.g., ~5,000 images)
* **Format:** JPG/PNG
* **Classes:** At least 5–7 disease categories
* **Split Ratio:**
  + Training: 70%
  + Validation: 15%
  + Testing: 15%

**3.4 Model Architecture**

* **Input Layer:** 224x224x3 (RGB image)
* **Convolution Layers:** Multiple layers with ReLU activation
* **Pooling Layers:** MaxPooling for dimensionality reduction
* **Dropout Layer:** To prevent overfitting
* **Fully Connected Layers:** Dense layers for classification
* **Output Layer:** Softmax activation for multi-class classification

**3.5 Hardware Requirements**

| **Component** | **Minimum** | **Recommended** |
| --- | --- | --- |
| Processor | Intel i3 | Intel i5 / Ryzen 5 |
| RAM | 4 GB | 8+ GB |
| Storage | 20 GB | 50 GB SSD |
| GPU | Optional | NVIDIA GTX 1650 or higher |

**3.6 Software Requirements**

| **Software** | **Version** |
| --- | --- |
| Visual Studio Code | Latest |
| Python | 3.10+ |
| TensorFlow | 2.15+ |
| Keras | Latest |
| OpenCV | Latest |
| NumPy, Pandas, Matplotlib | Latest |

**4. Testing and Validation**

**4.1 Testing Strategy**

* **Unit Testing:** Test each function like image preprocessing, model training.
* **Integration Testing:** Ensure the pipeline works from input to prediction.
* **Performance Testing:** Check accuracy and loss.
* **User Acceptance Testing:** Validate with real images.

**4.2 Evaluation Metrics**

* Accuracy
* Precision
* Recall
* F1 Score
* Confusion Matrix

**5. Future Enhancements**

* Web-based application deployment.
* Integration with mobile apps.
* Use of advanced architectures like ResNet or EfficientNet.
* Integration with a real-time camera for live predictions.

**6. Appendix**

* Kaggle dataset links
* Research papers used
* Model training graphs
* Sample output screenshots

## ****Diagrams to Include in SRS****

### ****1. Context-Level Diagram (Level 0 DFD)****

**Purpose:** Shows the overall system at a high level, how external entities interact with it.

**Description:**

* The **user** uploads a skin image.
* The **system** processes the image using the trained ML model.
* The **system** outputs the disease prediction.
* Optionally, connect with the **dataset source** (Kaggle) and **prediction report generation**.

**Elements:**

* **Entities:** User, Dataset (Kaggle).
* **Process:** Skin Disease Recognition System.
* **Data Flow:** Image input → Model → Result output.

**Tool Suggestions:** Draw.io, Lucidchart, Microsoft Visio.

### ****2. Level 1 Data Flow Diagram (DFD)****

**Purpose:** Breaks down the internal data flow of the system.

**Description:**  
Shows how data moves between different components:

1. **User uploads image** →
2. **Image Preprocessing Module** (resize, normalization, augmentation) →
3. **CNN Model Training Module** →
4. **Prediction Module** →
5. **Output Result to User**

**Key Processes to Include:**

* Dataset Import
* Preprocessing
* Model Training
* Model Testing
* Prediction

### ****3. Use Case Diagram****

**Purpose:** Shows the interaction between **actors** and **system functionalities**.

**Actors:**

* **Admin/Researcher** – manages dataset and training.
* **User (Patient/Doctor)** – uploads images and gets results.

**Use Cases:**

* Upload Skin Image
* Preprocess Image
* Train CNN Model
* Test Model
* Predict Disease
* View Accuracy & Reports

**Tools:** Draw.io, StarUML, Lucidchart.

### ****4. System Architecture Diagram****

**Purpose:** Explains the technical design of the project, including **data sources, model, backend, and output**.

**Structure:**

* **Input Layer:** User uploads skin image
* **Preprocessing Layer:** OpenCV + NumPy
* **Model Layer:** CNN (TensorFlow/Keras)
* **Training & Testing:** Using Kaggle dataset
* **Output Layer:** Disease prediction + Confidence score

You can make a **3-tier architecture diagram**:

* **Presentation Layer** → User Interface (VS Code testing, later web/app UI)
* **Logic Layer** → ML Model + Preprocessing
* **Data Layer** → Kaggle Dataset

### ****5. Flowchart of the Model Workflow****

**Purpose:** Shows step-by-step workflow of your machine learning pipeline.

**Steps:**

1. Start
2. Load Dataset from Kaggle
3. Preprocess Images (resize, normalize, augment)
4. Split Data (Train, Validation, Test)
5. Build CNN Model
6. Train Model
7. Evaluate Model (Accuracy, Loss)
8. Predict Disease on Test Images
9. Display Result
10. End

**Tools:** Lucidchart, Creately, PowerPoint, or Canva.

### ****6. Sequence Diagram****

**Purpose:** Shows the **interaction** between components in a **time-ordered manner**.

**Actors:**

* User
* System (Model)
* Dataset

**Example Flow:**

1. User → Uploads image
2. System → Preprocess image
3. System → Sends image to trained model
4. Model → Returns prediction result
5. System → Displays prediction to user

### ****7. Class Diagram (Optional – If Using OOP)****

**Purpose:** Useful if you're using **object-oriented programming** in Python.

**Classes to Include:**

* **DatasetHandler** → Loads and preprocesses images
* **CNNModel** → Builds and trains the CNN model
* **Predictor** → Predicts disease
* **Visualizer** → Displays graphs

### ****8. ER Diagram (If Database Is Used)****

If later you deploy your model into a **web app** or **attendance-type system** with a database, you should include an **Entity-Relationship Diagram**.

**Entities:**

* User
* Dataset
* Predictions
* Reports

## ****A. ML-Related Diagrams to Include**** ✅ (For SRS & Report)

### ****1. CNN Model Architecture Diagram**** (Highly Recommended)

**Purpose:** Shows the structure of your deep learning model.

**Components to include:**

* **Input Layer:** 224x224x3 RGB skin image
* **Convolutional Layers:** Feature extraction
* **Pooling Layers:** Dimensionality reduction
* **Dropout Layers:** Prevent overfitting
* **Fully Connected Dense Layers:** Classification
* **Output Layer:** Softmax activation → predicts disease class

**Why include it:**  
It visually explains how your ML model processes images step by step.

**Tools to use:**

* TensorFlow’s **plot\_model()** function
* Draw.io / Canva (manual diagram)

### ****2. End-to-End ML Workflow Diagram**** (Must Have)

**Purpose:** Shows the **pipeline** of your project from dataset to prediction.

**Steps:**

1. Load dataset from Kaggle
2. Image preprocessing (resize, normalize, augment)
3. Train-test split
4. Build CNN model
5. Train model
6. Evaluate performance
7. Predict disease
8. Display result

**Why include it:**  
Helps reviewers understand the entire workflow at a glance.

### ****3. Data Preprocessing Flowchart**** (Recommended)

**Purpose:** Shows how raw Kaggle images are prepared before model training.

**Steps:**

* Read dataset → Resize images → Normalize pixel values → Data augmentation → Save processed data → Train model

**Why include it:**  
Demonstrates that you handled the dataset properly, which is important for academic grading.

### ****4. Confusion Matrix Diagram**** (Highly Recommended)

**Purpose:** Shows how well your model classifies images into correct skin disease categories.

**Example:**

|  | **Predicted: Healthy** | **Predicted: Acne** | **Predicted: Eczema** |
| --- | --- | --- | --- |
| **Actual: Healthy** | 95 | 2 | 3 |
| **Actual: Acne** | 1 | 92 | 7 |
| **Actual: Eczema** | 2 | 6 | 92 |

**Why include it:**

* Helps visualize classification performance.
* Essential for academic ML projects.

### ****5. Dataset Class Distribution Graph**** (Optional but Good)

**Purpose:** Shows how many images you have per disease category.

**Example:**

* Healthy: 1000 images
* Melanoma: 800 images
* Psoriasis: 750 images
* Acne: 900 images
* Eczema: 850 images

**Why include it:**  
Shows dataset balance. Reviewers love this graph.

## ****B. Performance Graphs to Include**** 📊 (For Model Evaluation)

### ****1. Training & Validation Accuracy Graph**** (Must Have)

**Purpose:** Shows how your model learns over epochs.

**X-axis:** Epochs  
**Y-axis:** Accuracy (%)

* Training accuracy → Blue line
* Validation accuracy → Orange line

**Why include it:**  
Proves your model isn’t overfitting or underfitting.

### ****2. Training & Validation Loss Graph**** (Must Have)

**Purpose:** Shows model’s learning efficiency.

**X-axis:** Epochs  
**Y-axis:** Loss value

* Training loss → Green line
* Validation loss → Red line

**Why include it:**  
Shows if the CNN is optimizing properly.

### ****3. ROC Curve & AUC Score**** (Optional, but Recommended)

**Purpose:** Measures classification performance for each class.

* Higher AUC (>0.9) = Better performance.
* Useful if you have more than two classes.

### ****4. Sample Prediction Visualization**** (Optional, but Great for Report)

**Purpose:** Show some **real test images** along with:

* True label
* Predicted label
* Confidence score

**Example:**

| **Image** | **True Label** | **Predicted Label** | **Confidence** |
| --- | --- | --- | --- |
| 🖼️ Image1 | Psoriasis | Psoriasis | 97% |
| 🖼️ Image2 | Acne | Eczema | 70% |
| 🖼️ Image3 | Healthy | Healthy | 95% |