

# **MINI PROJECT**

## **Skin Disease Detection using Machine Learning**

### **OVERVIEW:**

Dermatology is the branch of bioscience that's involved with diagnosing and treatment of skin based mostly disorders. The immense spectrum of dermatologic disorders varies geographically and additionally seasonally because of temperature, humidness and alternative environmental factors. Human skin is one amongst the foremost unpredictable and tough terrains to mechanically synthesize and analyse because of its quality of unevenness, tone, presence of hair and alternative mitigating options. Though, many researches are conducted to find and model human skin victimisation (PC Vision techniques), only a few have targeted the medical paradigm of the matter. Due to lack of medical facilities available in the remote areas, patients usually ignore early symptoms which may worsen the situation as time progresses. Hence, there is a rising need for automatic skin disease detection system with high accuracy. Thus, we develop a multiclass deep learning model to differentiate between Healthy Skin Vs Skin suffering from a Disease and Classification of Skin Diseases into its main classes like MelanocyticNevi, Melanoma, Benign keratosis-like lesions, Basal cell Carcinoma, ActinicKeratoses, Vascular lesion and Dermatofibroma. We have used Deep Learning to train our model, Deep Learning is a part of Machine Learning in which unlike Machine Learning it uses large dataset and hence the number of classifiers is reduced substantially. The machine learns itself and divide the data provided into the levels of prediction and in a very short period of time gives the accurate results, thereby promoting and supporting development of Dermatology. The algorithm that we have used is Convolutional Neural Network (CNN) as it is one of the most preferred algorithm for image classification.

### **MODULE SUMMARY:**

#### **1. Data Collection**

Sources:

Images: High-quality images of skin conditions from sources like hospitals, clinics, research databases (e.g., ISIC Archive, DermNet).

Metadata: Patient demographic information (age, sex), medical history, symptoms, and other relevant clinical data.

Annotations: Expert-labelled data indicating the type of skin disease for each image.

#### **2. Preprocessing**

**Standardization:**

Format: Convert all images to a standard format (e.g., JPEG, PNG).

Size: Resize images to a uniform size (e.g., 224x224 pixels) to ensure consistency.

Normalization: Scale pixel values to a range (e.g., 0 to 1 or -1 to 1).

Segmentation: If necessary, segment images to focus on the area of interest (the lesion).  
Noise Reduction: Apply basic filtering techniques to remove noise and enhance image quality.

### **3. Feature Extraction**

Image Features:

Color Features: Mean and standard deviation of RGB values, color histograms.

Texture Features: GLCM (Gray Level Co-occurrence Matrix) features, LBP (Local Binary Patterns).

Shape Features: Contour features, area, perimeter, aspect ratio.

Deep Features: Extracted using pre-trained CNNs like VGG16, ResNet50, or custom deep learning models.

Metadata Features: Direct use of clinical features (e.g., patient age, symptom duration).

### **4. Labeling**

Annotation: Dermatologists or trained experts annotate each image with the corresponding skin disease label.

Quality Control: Verify and validate annotations to ensure high-quality, accurate labels.

### **5. Model Selection**

Algorithms:

Traditional ML: Support Vector Machines (SVM), Random Forest, Gradient Boosting.

Deep Learning: Convolutional Neural Networks (CNNs) for image data (e.g., ResNet, DenseNet), potentially combined with metadata through multi-input models.

Binary Classification: For distinguishing between healthy and diseased skin.

Multi-class Classification: For identifying specific types of skin diseases.

### **6. Model Training**

Dataset Splitting: Divide the dataset into training, validation, and test sets (e.g., 70-20-10 split).

Training Process:

Loss Functions: Use cross-entropy loss for classification tasks.

Optimization Algorithms: Use Adam, SGD, or other suitable optimizers.

Regularization: Apply dropout, L2 regularization to prevent overfitting.

Hyperparameter Tuning: Perform grid search, random search, or Bayesian optimization to fine-tune model parameters.

### **7. Model Evaluation**

Metrics:

Accuracy: Measure overall correctness.

Precision and Recall: Evaluate performance on imbalanced datasets.

F1 Score: Balance between precision and recall.

ROC-AUC: Assess the trade-off between true positive rate and false positive rate.

Validation: Use cross-validation to ensure the model generalizes well to unseen data.

## 8. Deployment

Integration:

Healthcare Systems: Ensure compatibility with Electronic Health Records (EHR) systems.

API Development: Create APIs for easy integration into existing applications.

Real-time Inference: Optimize the model for fast, real-time predictions.

User Interface: Design an intuitive interface for dermatologists and healthcare providers to use the model effectively.

## 9. Post-Deployment Monitoring

Continuous Learning: Regularly update the model with new data to improve accuracy and performance.

Monitoring: Implement monitoring tools to track model performance and detect issues.

## CODE SEGMENT:

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import classification_report, confusion_matrix

# Define directories
train_dir = 'data/train'
val_dir = 'data/val'
test_dir = 'data/test'

# Data augmentation and normalization for training
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

```

# Only rescaling for validation and testing
val_test_datagen = ImageDataGenerator(rescale=1./255)

# Flow images from directories
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

val_generator = val_test_datagen.flow_from_directory(
    val_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

test_generator = val_test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(len(train_generator.class_indices), activation='softmax')
])

model.compile(optimizer=Adam(learning_rate=1e-4),
              loss='categorical_crossentropy', metrics=['accuracy'])

```

```
model.summary()

# Define callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_model.h5', save_best_only=True)

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    validation_data=val_generator,
    validation_steps=val_generator.samples // val_generator.batch_size,
    epochs=50,
    callbacks=[early_stopping, model_checkpoint]
)

# Load the best model
model.load_weights('best_model.h5')

# Evaluate on test data
test_loss, test_acc = model.evaluate(test_generator, steps=test_generator.samples //
test_generator.batch_size)
print(f'Test accuracy: {test_acc:.2f}')

# Predict on test data
y_pred = model.predict(test_generator, steps=test_generator.samples //
test_generator.batch_size)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = test_generator.classes

# Classification report
print(classification_report(y_true, y_pred_classes,
target_names=test_generator.class_indices.keys()))

# Confusion matrix
cm = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(10, 8))
plt.imshow(cm, cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

## INPUT / OUTPUT

```
[ ] epochs = 2
    batch_size = 10
    history = model.fit_generator(
        datagen.flow(x_train,y_train, batch_size=batch_size),class_weight=class_weights,
        steps_per_epoch=x_train.shape[0] // batch_size,
        epochs=epochs,
        validation_data=(x_validate,y_validate),
        validation_steps=x_validate.shape[0] // batch_size
        ,callbacks=[learning_rate_reduction]
    )

Epoch 1/2
720/721 [=====] - ETA: 0s - loss: 0.1701 - acc: 0.9400Epoch 1/2
802/721 [=====] - 9s 11ms/sample - loss: 0.1401 - acc: 0.9375
721/721 [=====] - 136s 466ms/step - loss: 0.1705 - acc: 0.9400 - val_loss: 0.1514 - val_acc: 0.9375
Epoch 2/2
720/721 [=====] - ETA: 0s - loss: 0.1670 - acc: 0.9415Epoch 1/2
802/721 [=====] - 9s 11ms/sample - loss: 0.1420 - acc: 0.9371
721/721 [=====] - 136s 466ms/step - loss: 0.1678 - acc: 0.9412 - val_loss: 0.1536 - val_acc: 0.9371

[ ] loss, accuracy = model.evaluate(x_test, y_test, verbose=1)
    loss_v, accuracy_v = model.evaluate(x_validate, y_validate, verbose=1)
    print("Validation: accuracy = %f ; loss_v = %f" % (accuracy_v, loss_v))
    print("Test: accuracy = %f ; loss = %f" % (accuracy, loss))

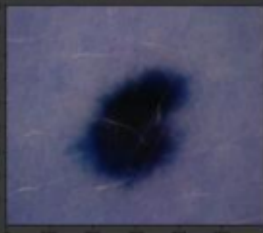
2803/2803 [=====] - 20s 18ms/sample - loss: 0.1082 - acc: 0.9328
802/802 [=====] - 8s 10ms/sample - loss: 0.1047 - acc: 0.9371
Validation: accuracy = 0.937121 ; loss_v = 0.104717
Test: accuracy = 0.932815 ; loss = 0.108198
```

```
+ Code + Text
WARNING:tensorflow:from /tensorflow-1.15.2/python3.6/tensorflow_core/python/ops/resource_variable_ops.py:1698: calling ResourceVariable.__init__ (from tensorflow
Instructions for updating:
If using Keras pass *constraint arguments to layers.
WARNING:tensorflow:large dropout rate: 0.55 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate instead of keep_prob. Please ensure that this is intended.
Model: "sequential"

Layer (type)                 Output Shape                 Param #
-----
conv2d (Conv2D)              (None, 75, 100, 32)         896
-----
conv2d_1 (Conv2D)            (None, 75, 100, 32)         8240
-----
max_pooling2d (MaxPooling2D) (None, 37, 50, 32)          0
-----
dropout (Dropout)            (None, 37, 50, 32)          0
-----
conv2d_2 (Conv2D)            (None, 37, 50, 64)         18496
-----
conv2d_3 (Conv2D)            (None, 37, 50, 64)         36928
-----
max_pooling2d_1 (MaxPooling2 (None, 18, 25, 64)          0
-----
dropout_1 (Dropout)          (None, 18, 25, 64)          0
-----
conv2d_4 (Conv2D)            (None, 18, 25, 128)        73856
-----
max_pooling2d_2 (MaxPooling2 (None, 9, 12, 128)          0
-----
dropout_2 (Dropout)          (None, 9, 12, 128)          0
-----
flatten (Flatten)            (None, 1152)                0
-----
dense (Dense)                (None, 512)                 707840
-----
dropout_3 (Dropout)          (None, 512)                 0
-----
dense_1 (Dense)              (None, 7)                   3591
-----
Total params: 7,221,415
Trainable params: 7,221,415
Non-trainable params: 0
```

```
[ ] import cv2
test_image = cv2.imread('image3.jpg')

[ ] import matplotlib.pyplot as plt
plt.imshow(test_image)
plt.show()

<matplotlib.image.AxesImage at 0x2f3ffcc32828>


[ ] test_image = np.expand_dims(test_image, axis=0)
t1 = np.resize(test_image, (75,100,3))
t2 = np.expand_dims(t1, axis=0)

[ ] print(model.predict(t2))
print(model.predict_classes(t2))

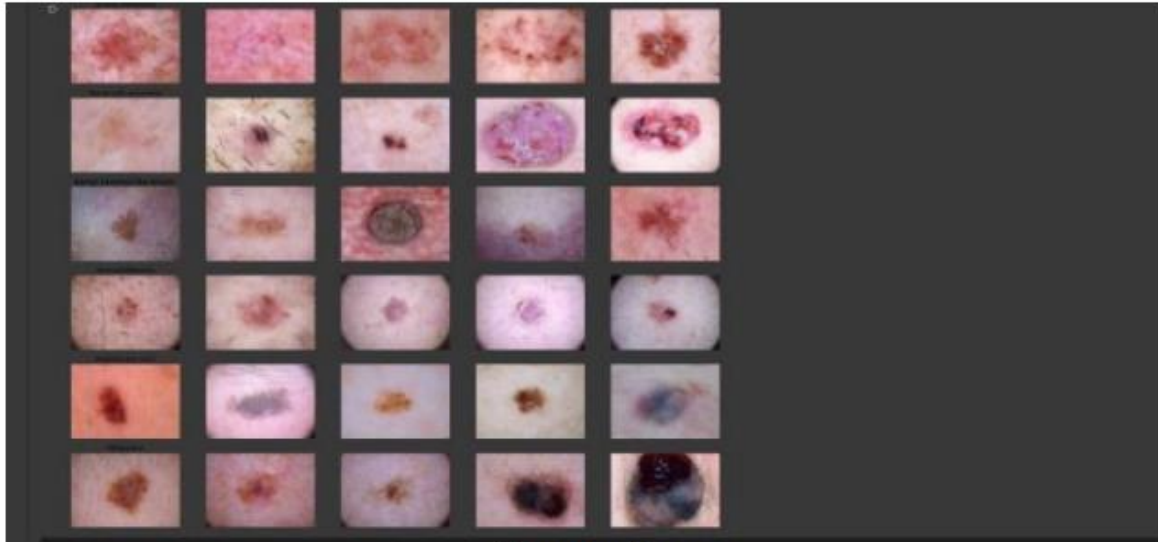
[[0. 0. 0. 0. 1. 0. 0.]]
[4]
```

Testing for an Image with Detection

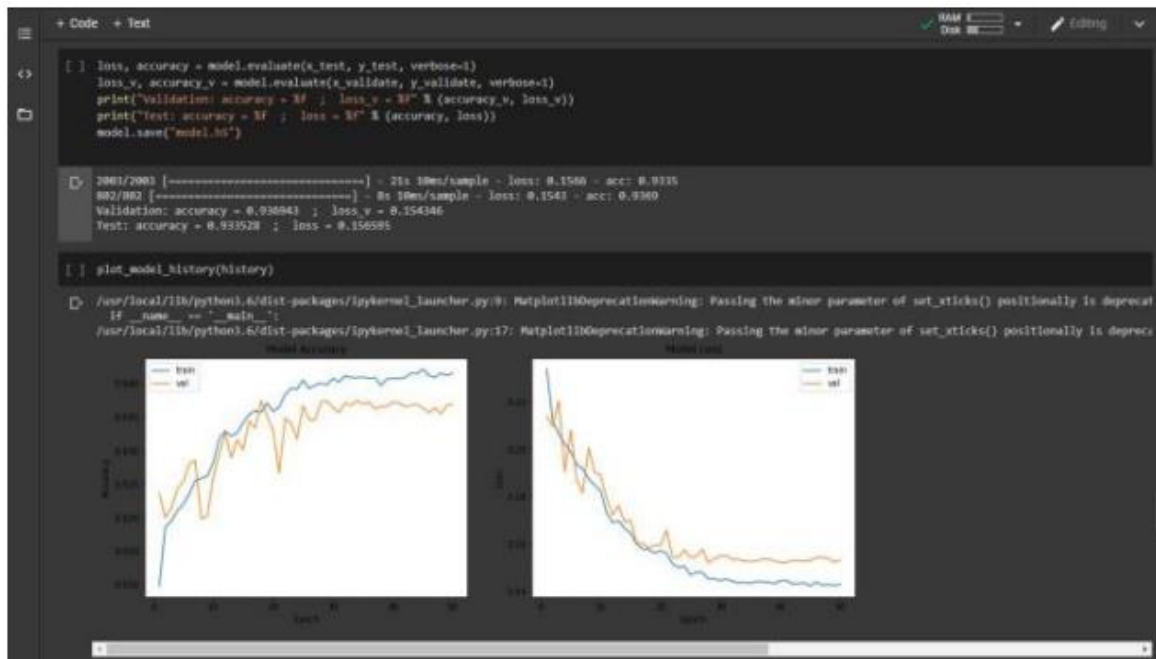
```
[ ] tile_df = pd.read_csv(os.path.join(base_skin_dir, 'HAM10000_metadata.csv'))
tile_df['path'] = tile_df['image_id'].map(image_id_path_dict.get)
tile_df['cell_type'] = tile_df['dx'].map(lesion_type_dict.get)
tile_df['cell_type_idx'] = pd.Categorical(tile_df['cell_type']).codes
tile_df.sample(3)
```

	lesion_id	image_id	dx	dx_type	age	sex	localization	path	cell_type	cell_type_idx
6019	HAM_0000422	ISIC_0027664	nv	follow_up	40.0	male	trunk	/content/HAM10000_images_part_1/ISIC_0027664.jpg	Melanocytic nevi	4
9667	HAM_0005018	ISIC_0028008	nv	consensus	55.0	male	upper extremity	/content/HAM10000_images_part_1/ISIC_0028008.jpg	Melanocytic nevi	4
7009	HAM_0006145	ISIC_0031713	nv	histo	60.0	female	back	/content/HAM10000_images_part_2/ISIC_0031713.jpg	Melanocytic nevi	4

Model Summary



Streamed Images



## CONCLUSION:

Skin Diseases are ranked fourth most common cause of human illness, but many still do not consult doctors. We presented a robust and automated method for the diagnosis of dermatological diseases. Treatments for skin are more effective and less disfiguring when found early. We should point out that it is to replace doctors because no machine can yet replace the human input on analysis and intuition. Researches in European Society of Medical Oncology have shown for the first time that form of AI or ML is better than experienced dermatologists. In this a brief description of the system and the implementation methodology is presented.