MELANOMA CANCER DETECTION USING CNN

A PROJECT REPORT

1. Introduction

1.1. Motivation

The reasons for detecting melanoma cancer are numerous. For melanoma, a fast-growing type of skin cancer, early detection is saving lives first. DL significantly enhances the possibility of being able to cure and treat patients. Second, conventional diagnosis methods like skin biopsies and visual checks by dermatologists are often time-consuming, subjective, and susceptible to human error. Third, in most areas, DL algorithms, particularly convolutional neural networks, have reached phenomenal performance in image classification. The algorithms are very flexible and scalable. Furthermore, with increasingly more data and inputs, DL models can always learn and update over time, thereby guaranteeing the efficacy and timeliness of the diagnostic system in the face of advances in medical knowledge and technology. Lastly, with hundreds of millions of new diagnoses every year, skin cancer most particularly melanoma is a global public health issue.

1.2. Problem Statement

Melanoma is a highly aggressive and life-threatening skin cancer that needs to be diagnosed early to be curable, but diagnostic interventions like eye examination and biopsies are time-consuming, subjective, and inaccessible to remote regions. In the spirit of enhancing diagnostic precision, reducing diagnosis time, and enhancing accessibility in resource-constrained environments, this study will develop a deep learning (DL)-based system to automate melanoma detection from dermoscopic images with the help of convolutional neural networks (CNNs).

With datasets like ISIC and HAM10000, the system will employ methods like transfer learning, data augmentation, and ensemble methods to overcome limitations like limited datasets and class imbalance and ensure clinical validity through the use of stringent validation metrics (e.g., accuracy, sensitivity, AUC-ROC).

The aim is to develop a technology that can meaningfully scale and enable clinicians to detect early melanoma and ultimately enhance patient outcomes and reduce mortality rates. While as much has been achieved with diagnosis using CNN-based melanoma, there are still some limitations, such as the need for large diverse datasets to enhance model generalizability and the possibility of algorithmic bias when the algorithms are trained on narrow groups. Moreover, implementation of these systems into clinical workflows entails the surmounting of legal and ethical challenges, such as patient data protection and AI decision-making transparency.

Future research will involve improving interpretability of the model, expanding datasets to encompass a range of skin tones and types of lesions, and working with doctors to ensure the technology is as good as, or better than, existing diagnostic techniques. With these challenges in mind, this research seeks to develop a consistent, egalitarian, and clinically validated device for early detection of melanoma to aid in global healthcare programs.

1.3. Background

Creating a Deep learning model for the detection and classification of Melanoma malignancy Detection is of utmost concern for improving public health and preventing the dissemination of this malignancy. An effective model would be able to identify rapidly and efficiently individuals who can be prone to infection. DL algorithms can be able to process large volumes of data efficiently and rapidly, thus potentially resulting in early melanoma detection and follow-up. DL-based diagnostic equipment can also be provided in different healthcare centers, even remote and poor communities, where specialized dermatological expertise may not be present.

Except for that, this study can have widespread applications, which include the creation of a new drug to treat Melanoma Skin Cancer. The patients can be treated early if they get diagnosed. By realizing the full potential of DL, researchers and doctors anticipate improving existing diagnostic methods and alleviating the suffering of melanoma skin cancer for patients and the healthcare system alike. Additionally, creating a deep learning model to identify and classify Melanoma Cancer can also save millions of lives and enhance public health results

2. Literature Survey

1. Detection of Melanoma Skin Cancer based on Image Processing and Deep Learning. This contribution is substantial to the field of medical and dermatological image analysis because it proposes a system that integrates deep learning models and image processing algorithms for melanoma lesion diagnosis. In this manner, it provides a potentially valuable means of automating detection protocols, which is crucial for timely diagnosis and treatment. The paper is deficient, though, in giving concrete details regarding the particular image processing techniques and deep learning models used in the research. This imprecision renders replication and complete comprehension of the provided process impossible. The entire description of these processes not just will

elaborate the study findings, but also will render them easier to comprehend for other researchers working in this field.

2. Computer-Based Melanoma Detection Techniques on the Basis of Deep Learning. This paper presents a comprehensive explanation of computer-assisted diagnosis (CAD) systems intended for early detection of malignant melanoma, illustrating the importance of early detection in enhancing the patient's prognosis. By examining some of the steps of computer-aided diagnosis, such as acquisition, pre-processing, segmentation, feature extraction, and classification, the paper provides a comprehensive overview of how complicated the process is. But it is lacking in some aspects, particularly in the description of some of the deep learning algorithms and their performance with respect to melanoma classification. Integration of real-world examples or case studies illustrating the effectiveness of some deep learning algorithms would be a significant contribution towards greater practicality and applicability of the suggested frameworks.

3. Melanoma skin cancer classification using convolutional neural networks. This paper specifically emphasizes the significance of early detection in minimizing mortality rates for malignant melanoma. This is a comprehensive review of Computer-Aided Diagnosis (CAD) and deep learning-based methods for melanoma detection. The paper is not very clear-cut in whether or not it makes use of real examples or case studies to illustrate the effectiveness of such techniques on real cases. A demonstration of empirical evidence or verification of targeted models would be of invaluable utility and relevance to the ideas of the paper.

4. Deep learning algorithm-based methods for diagnosis of skin cancer. The research work discussed here tries to meet the vast requirement of correct and exact diagnosis of skin lesions, especially malignant melanoma. By the application of a blend of multiple feature extraction techniques such as the ABCD rule, GLCM, and HOG, the algorithm offers an overall solution to early lesion detection, improving performance and accuracy measures. Yet, some of the limitations such as the built-in biases developed by the application of feature extraction techniques alone as the foundation of the method need to be ascertained. These would

compromise the sensitivity of the algorithm to certain types of lesions or variations, and further refinement and testing would be required.

5. The reported 83% accuracy rate cannot be claimed in the case of various datasets, and the sole use of a single classifier, SVM, without comparative use of other techniques can restrict the reliability and usability of the proposed system. There must be further research and experiments to get over such limitations.

6. Deep learning and skin cancer applications. This work examines the future prospect of Deep Learning in dermatology and posits how the utilization of AI systems can aid the detection of skin cancer. The work puts considerable questions into issue regarding inequality between AI applications within dermatology versus other branches of medicine, such as radiology, and calls for more inquiry into the usage of AI for dermatological diagnoses. However, the paper may do better in giving information regarding some of its areas, specifically on explaining DL basics. Offering a clearer definition of DL methods and algorithms would give readers a better sense of the underlying ideas, therefore extending the discourse.

7. Deep learning-based skin cancer diagnosis. While this research offers a sufficient framework for the diagnosis of skin cancer, it may offer more in-depth explanations of pre-processing techniques and feature extraction methods. Such discussions will enhance readers' knowledge of the reasons behind these procedures and how they can be applied in the context of skin cancer diagnosis. Providing such information will not only make the methodologies clearer but also advance knowledge in the field.

8. Hybrid feature fusion and deep learning algorithms for detecting melanoma skin cancer. The research greatly enhances skin cancer detection by suggesting an achievable system for automated melanoma skin cancer detection. The use of anisotropic diffusion filtering effectively removes noise from dermoscopy images and improves image quality, allowing accurate segmentation. However, the research can explore further on possible issues involving the use of the suggested approach in real-world clinical practice. In addition, a deeper analysis of false

positives and false negatives can shed light on the strengths and limitations of the model, thereby allowing for continued improvement and optimization.

9. Skin cancer detection using ensemble of deep learning and deep learning techniques. The paper proposes a novel approach that combines deep learning and deep learning techniques for skin cancer detection, focusing on melanoma. While the study presents significant improvements in accuracy compared to existing methods, it could provide more detailed insights into the rationale behind technique selection and integration. Furthermore, a comprehensive evaluation of the model's performance across diverse datasets and clinical settings would enhance its generalizability and robustness, thus contributing to its practical utility in real-world applications.

10. Deep learning approach for melanoma cancer stage identification. This research focuses on the identification of melanoma cancer stages, which are critical in treatment planning and patient outcome estimation. While it delivers hopeful results, it can also provide more explanatory accounts of CNN structure and implementation aspects to achieve more repeatability and future study. In addition, providing light on how decisions were made over techniques, such as deciding the SMTP loss function, will provide a larger contribution and give the proposed approach deeper insight.

2.1. Problem Definition

This research intends to create a advanced system to identify early melanoma skin cancer through convolutional neural networks (CNNs). Visual examination and biopsies, the traditional diagnostic methods, are time-consuming and subjective in nature and result in possible delays and mistakes.

The method is intended to automate and improve melanoma diagnosis through skin lesion image processing. Key objectives are to generate a diverse dataset, enhance diagnostic accuracy, facilitate early detection, provide scalability for widespread use, and provide assurance with rigorous testing.

3. Overview of the Work

3.1. Objectives of the Project

The main aim for choosing melanoma cancer detection system is to provide a more effective and reliable means of detecting melanoma, an aggressive skin cancer. The usual diagnosis procedures generally involve prolonged procedures such as biopsies and dermatologist consultations, which might take up to a week or even longer for results to arrive. With the help of deep learning, i.e., convolutional neural networks (CNNs) and conventional deep learning classifiers, the framework tries to make the process of diagnosis more straightforward using images of lesions. The system is designed to run three major steps: data harvesting and augmentation, model architecture, and prediction. By harvesting and augmenting an assortment of different lesion images, the model recognizes various types of skin cancer like melanoma. The CNN and classical deep learning classifiers are trained on skin lesion border, texture, and color features and can classify the different types of skin cancer.

The overall effect is to reduce the interval from suspicion to diagnosis of melanoma, as compared with what presently takes far more time. Reducing this time can make an enormous positive difference in millions of people by making earlier treatment and detection possible of melanoma, thus boosting the chances for good outcomes.

3.2. Software Requirements

System specifications are specified in the software requirements document. It should have a description of specs and requirements. It is less crucial how to do what the system must do than what the system must do. The basis for writing a software requirements specification is the software requirements.

- Python IDE: Anaconda Jupyter Notebook
- Google Chrome
- Python 3.6 or above

3.3. Hardware Requirements

Hardware requirements make system building contracts depend on them, so they must be an expression of full and consistent specs for the entire system.

They are the basis of the system design of the software engineer.
It informs you about what the system does, but not how it works.

- Processor : Intel i5 and above
- RAM :4 GB (minimum)
- Internal memory:40 GB (minimum)

4. System Design

4.1. Algorithms

Sequential Model: The Sequential model in Keras is a basic and widely used form of neural network architecture. It enables you to construct a linear stack of layers in a neural network where each layer consumes exactly one input tensor and produces exactly one output tensor. This linear stack of layers is the basic building block of most deep learning models. With the Sequential model, you can add layers one by one, defining the configuration of each layer. The layers are added in sequential order, with the output of each layer being used as input to the subsequent layer. This ease of use makes it easy to develop and test different neural network topologies, especially for deep learning beginners.

Convolutional Layers (Conv2D): Convolutional Layers (Conv2D) in Convolutional Neural Networks (CNNs) extract data from input images through convolution processes. They shift small kernels across the image, collecting different patterns like edges and textures. Piling multiple Conv2D layers, the network learns progressively complex features hierarchically. Activation functions like ReLU inject non-linearity, improving the network's capability to approximate deep correlations in the data. This method enables CNNs to learn and perceive images efficiently for tasks like categorisation and detection.

Max Pooling Layers (MaxPooling2D): Max Pooling Layers (MaxPooling2D) are fundamental blocks of Convolutional Neural Networks (CNNs), which primarily serve the purpose of down sampling the feature maps produced by the convolutional layers. These layers have a critical role in lowering the spatial dimension of the feature maps without the loss of essential information. MaxPooling2D does this down sampling by partitioning the input feature map into small rectangular blocks and taking the maximum value within each block to highlight it in the output. By choosing the maximum value, MaxPooling2D keeps the most significant features, thus maintaining the most significant features of the input data

while minimizing computational complexity. This down sampling achieves spatial invariance, making the network less sensitive to input variations, e.g., shifts and distortions.

Flatten Layer: Flatten layer is a simple building block in Convolutional Neural Networks (CNNs), with the responsibility of flattening the multi-dimensional feature maps generated by the convolutional layers into one dimensional vector. This is needed to shift from the spatial hierarchies learned by the convolutional layers to the fully connected layers that take one-dimensional input. By flattening the feature maps into a vector, the Flatten layer essentially unwraps the spatial layout of the feature maps without altering the information held within them. This permits the subsequent fully connected layers to acquire global patterns and correlations over the entire image to perform classification, regression, or other tasks requiring a fixed-length input vector.

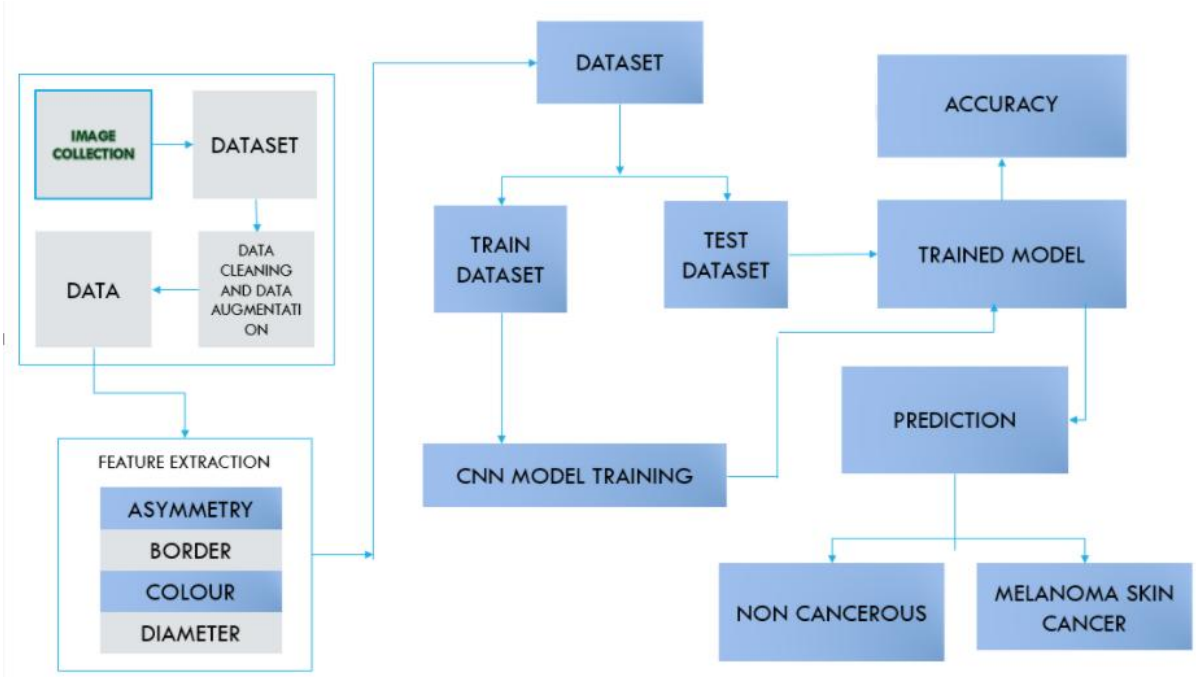
Dense Layers (Dense): Dense Layers, also referred to as fully linked layers, are a simple building block of neural networks, especially Convolutional Neural Networks (CNNs).
In Dense layers, every neuron is connected to each neuron in the previous layer, giving a fully connected network. In the case of CNNs, Dense layers are mostly used following the convolutional and pooling layers to perform high-level reasoning and decision-making. They take input either from the flattened feature maps or the output from the previous layer and perform a linear operation on the input, followed by applying an activation function.

Activation Functions (Activation): Activation functions introduce non-linearity into the network, and this allows it to learn complex patterns. Some common activation functions are ReLU, sigmoid, and softmax.

Dropout Layers (Dropout): Dropout Layers, used in the Dropout layer, are a regularization technique employed to reduce overfitting in neural networks.
Overfitting occurs when the model learns to memorize the training data instead of generalizing well to unseen data. Dropout serves to decrease this issue by randomly "dropping out" (i.e., setting to zero) a portion of neurons in the network while training.

Batch Normalization Layers (Batch Normalization): These layers normalize the activations of the preceding layer, making the training process more stable and quickening convergence.

4.2. Block Diagrams



5. Implementation

5.1. Description of Modules

- Data Collection
- Preprocessing and Image Data Generation
- Model Building and Training
- Testing and Evaluation

Data Collection: Data collection is an important step that entails procuring and collecting databases with crucial details on skin lesions, including melanoma and benign lesions. After finding relevant datasets, efforts are made to obtain the data. This involves obtaining consent from patients for using their medical images and corresponding data for research purposes. In certain situations, data can also be obtained prospectively from clinical trials or imaging sessions. High-quality images of skin lesions are required for melanoma detection. Dermatoscopy images, which provide detailed images of skin lesions, are used routinely. These images can be obtained with specialist dermatoscopes or imaging devices in clinical environments.

Preprocessing and Image Data Generation: Here, the data set is loaded onto a data frame in the form of image paths and corresponding labels. The test data is divided into two: Validation data and Testing data. Validation data is utilized to confirm whether the models have been trained or not. The Image Features are produced with the help of the Keras-Image Data Generator function. The generated data is also pre-processed prior to using it to train the deep learning models. All three data splits; training data, validation data and test data are used to get the features and generate data to use on the pretrained deep models.

Model Building and Training: For this project here we have used several models to check which one is the most appropriate and by checking we have gained some of the models that are appropriate for the detection with high accuracy and improved results compared to the other models.

Models used:

- Model – InceptionV3
- Optimizers - Adam, Adamax
- Layers - Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout, Batch Normalization

Testing and Evaluation: Here, the saved model is loaded, compiled and tested with the generated test data against different evaluation measures. The Models are tested against different measures such as the precision, recall, f1-score and support utilizing the categorical crossen-trophy module. The performance of the models in each measure and plotted to compare the models

5.2. Source Code

#Load Required Libraries

```
import numpy as np # Handles arrays
import os # Works with files & folders
import zipfile # EXTRACT ZIP FILE
import cv2 # Reads, processes, and resizes images.
import matplotlib.pyplot as plt
import seaborn as sns # Makes heatmaps (confusion matrix visualization)
```

```
from collections import Counter # Counts occurrences of classes (e.g., melanoma vs. non-melanoma).
from tensorflow.keras.preprocessing.image import ImageDataGenerator # Augments images (rotation, flipping, etc.) to improve training.
from tensorflow.keras.models import Sequential, load_model # Builds and loads deep learning models.
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout # Layers for CNN model.
from tensorflow.keras.applications import InceptionV3 # Pre-trained deep learning model (transfer learning).
from tensorflow.keras.optimizers import Adam # Optimizer that improves model learning.
from sklearn.metrics import classification_report, confusion_matrix, precision_score, recall_score, f1_score
```

 # Verify Extraction & List Images

```
import os

# Define correct paths as strings
train_folder = "ISIC_2019_Training_Extracted/ISIC_2019_Training_Input"
test_folder = "ISIC_2019_Test_Extracted/ISIC_2019_Test_Input"

# Check if folders exist before listing images
if os.path.exists(train_folder):
    train_images = os.listdir(train_folder)
    print(f"✓ Total training images: {len(train_images)}")
else:
    print(f"❌ Training folder not found: {train_folder}")

if os.path.exists(test_folder):
    test_images = os.listdir(test_folder)
    print(f"✓ Total test images: {len(test_images)}")
else:
    print(f"❌ Test folder not found: {test_folder}")
```

 # Load CSV Files & Display Dataset Information

```
import pandas as pd  # Import pandas

# Load the metadata CSV file
csv_path = "ISIC_2019_Training_GroundTruth (1).csv"
df = pd.read_csv(csv_path)

# Display first 5 rows
print("First 5 rows of CSV file:")
print(df.head())

# Check number of unique classes
print("\nUnique classes in dataset:")
print(df.columns[1:])  # Labels start from column 1

# Show column names
print("\nColumn names in the dataset:")
print(df.columns)

# Check total number of images
print(f"\nTotal images available: {df.shape[0]}")
#  Visualize Sample Images
def show_random_images(dataframe, img_folder, num_images=5, title="Random Images"):
    fig, axes = plt.subplots(1, num_images, figsize=(15, 5))
    fig.suptitle(title, fontsize=16)

    for i in range(num_images):
        img_name = np.random.choice(dataframe['image'])  # RandoDLy select an image
        img_path = os.path.join(img_folder, img_name + ".jpg")  # Construct full path

        img = cv2.imread(img_path)  # Read image

        if img is None:  # ✅ Check if image was loaded successfully
            print(f"Error: Could not read image at {img_path}")
```

```
            continue  # Skip to next image

        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB

        axes[i].imshow(img)  # Display image
        axes[i].axis("off")  # Hide axes
        axes[i].set_title(f"Image: {img_name}")  # Set title

    plt.show()  # Show images

# Paths for training & test images
train_folder = "ISIC_2019_Training_Extracted/ISIC_2019_Training_Input"
test_folder = "ISIC_2019_Test_Extracted/ISIC_2019_Test_Input"

# Visualize Training Images
show_random_images(df, train_folder, num_images=5, title="Random Training Images")

# Visualize Test Images (Assuming test images do not have labels)
test_images = [f.split('.')[0] for f in os.listdir(test_folder) if f.endswith('.jpg')]
test_df = pd.DataFrame({'image': test_images})

# Visualize Test Images
show_random_images(test_df, test_folder, num_images=5, title="Random Test Images")
```

 # Data Preprocessing & Augmentation

```
# Define image size
IMG_SIZE = (224, 224) #  Target image size for the model ( for deep learning models their will be fixed size)

# Create a function to preprocess images
def load_and_preprocess_image(img_path): # loads an image from a file and applies preprocessing to prepare it for model prediction.
    img = cv2.imread(img_path) # loads an image in BGR format (default for OpenCV)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, IMG_SIZE)
```

```
    img = img / 255.0  # Normalize Pixel Values (Scale Between 0 and 1) Normalizing between 0 and 1 improves model training by stabilizing learning.
    return img
```

 # TRY

```
import pandas as pd # for handling structured data (CSV files, tables, datasets, etc.).
import tensorflow as tf # TensorFlow is an open-source deep learning framework used for building, training, and deploying deep learning models.
from sklearn.model_selection import train_test_split
# Ensure image filenames have .jpg extension
df["image"] = df["image"].astype(str) + ".jpg"

# Check if images exist in the directory
missing_files = [img for img in df["image"] if not os.path.exists(os.path.join(train_extract_path, img))]
print(f"\nMissing images: {len(missing_files)}")

# Remove missing files from dataframe
df = df[~df["image"].isin(missing_files)]
print(f"Updated dataset size: {df.shape}")
```

 # Apply Data Augmentation

```
import shutil # used for copying, moving, renaming, and deleting files & directories.
from tensorflow.keras.utils import to_categorical # Convert Labels to One-Hot Encoding
from tqdm import tqdm # makes loops visually trackable by displaying a progress bar. Useful for iterating over large datasets efficiently.
# Define paths for training images, test images, and labels
train_img_dir = "ISIC_2019_Training_Extracted/ISIC_2019_Training_Input"
test_img_dir = "ISIC_2019_Test_Extracted/ISIC_2019_Test_Input"
train_csv_path = "ISIC_2019_Training_GroundTruth (1).csv"

# Ensure directories exist
assert os.path.exists(train_img_dir), f"Training image directory not found: {train_img_dir}"
assert os.path.exists(test_img_dir), f"Test image directory not found: {test_img_dir}"
assert os.path.exists(train_csv_path), f"Training CSV file not found: {train_csv_path}"
# Load CSV containing ground truth labels
```

```
df = pd.read_csv(train_csv_path)

# Display first few rows
print("Training Data Labels Preview:")
display(df.head())

# Check for missing values
print("Missing Values in Labels:")
print(df.isnull().sum())

# Get all class columns (disease types)
label_columns = df.columns[1:]  # Skipping the first column (image ID)

# Ensure only one label per image
df['num_labels'] = df[label_columns].sum(axis=1)
assert df['num_labels'].max() == 1, "Some images have multiple labels!"

# Convert multi-label to single-label (find the column with 1)
df['label'] = df[label_columns].idxmax(axis=1)

# Drop unnecessary columns
df = df[['image', 'label']]
# Load the CSV file
df = pd.read_csv("ISIC_2019_Training_GroundTruth (1).csv")

# Print initial preview
print("Original Data Labels Preview:")
print(df.head())

# Ensure column names are correct
print("\nColumn Names in CSV:")
print(df.columns.tolist())

# Get all class columns (excluding 'image')
```

```
label_columns = df.columns[1:]  # Exclude 'image' column

# Debug: Check if label_columns contains class names
print("\nClass Columns Detected:", label_columns)

# Convert one-hot encoding to single-label classification
df['label'] = df[label_columns].idxmax(axis=1)  # Get the class with max value

# Drop the original one-hot encoded columns
df = df[['image', 'label']]

# Print final output
print("\nUpdated Data Labels:")
print(df.head())
# Convert label column to string (required for ImageDataGenerator with
class_mode='binary')
df['label'] = df['label'].astype(str)

# Display class distribution
print("Class Distribution:")
print(df['label'].value_counts())

# Plot class distribution
plt.figure(figsize=(8, 4))
sns.countplot(y=df['label'])
plt.title("Class Distribution")
plt.show()
# Split data into training and validation sets (80% train, 20% validation)
train_df, val_df = train_test_split(df, test_size=0.2, stratify=df['label'], random_state=42)

# Reset index
train_df = train_df.reset_index(drop=True)
val_df = val_df.reset_index(drop=True)
```

```
print("Training Samples:", len(train_df))
print("Validation Samples:", len(val_df))
print(train_df["label"].unique())  # Should output ['0', '1']
print(val_df["label"].unique())    # Should output ['0', '1']
train_df['label'] = train_df['label'].astype(str)  # Ensure labels are strings
val_df['label'] = val_df['label'].astype(str)
# Print first few image names from dataframe
print(train_df["image"].head())
print(val_df["image"].head())
train_df["image"] = train_df["image"] + ".jpg"
val_df["image"] = val_df["image"] + ".jpg"
import os

train_img_dir = "ISIC_2019_Training_Extracted/ISIC_2019_Training_Input"

# List first 10 images in the directory
print("Files in image directory:", os.listdir(train_img_dir)[:10])
print(train_df["image"].head())  # Should contain only filenames (e.g., "ISIC_0000001.jpg")
# Define image size and batch size
IMG_SIZE = (224, 224)
BATCH_SIZE = 64

# Define Data Augmentation for training set
train_datagen = ImageDataGenerator(  # ImageDataGenerator is used to preprocess and
augment images before feeding them into a deep learning model. This helps improve model
generalization and prevent overfitting by generating slightly modified versions of the same
image.
    rescale=1./255, # Normalize pixel values (0-255 → 0-1)
    rotation_range=30, # RandoDLy rotate images up to 30 degrees
    width_shift_range=0.2, # Shift image horizontally by up to 20% of width
    height_shift_range=0.2, # Shift image horizontally by up to 20% of height
    shear_range=0.2, # Apply shear transformation  # stretching it diagonally look object
slantely
    zoom_range=0.2, #  RandoDLy zoom in on images
```

```
    horizontal_flip=True, # Flip images horizontally
    fill_mode="nearest" # Fill missing pixels after transformation
)

# No augmentation for validation set, only rescaling
val_datagen = ImageDataGenerator(rescale=1./255) # Only normalization
# Path where images are stored
train_img_dir = "ISIC_2019_Training_Extracted/ISIC_2019_Training_Input"

# Flow images from directory for training
train_generator = train_datagen.flow_from_dataframe(
    dataframe=train_df, # DataFrame containing image names & labels
    directory=train_img_dir, # Folder where images are stored
    x_col="image",  # Image filenames
    y_col="label",  # Binary class labels
    target_size=IMG_SIZE, # Resize images to required size (224x224)
    batch_size=BATCH_SIZE,  # Number of images per batch
    class_mode="categorical",  # Binary classification # Multi-class classification
    shuffle=True # RandoDLy shuffle images for training
)

# Flow images from directory for validation
val_generator = val_datagen.flow_from_dataframe(
    dataframe=val_df, # Validation DataFrame
    directory=train_img_dir, # Folder containing images
    x_col="image",
    y_col="label",
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False # No shuffling for validation (ensures consistency)
)

# Verify class indices
```

```
print("Class indices:", train_generator.class_indices)
```

 # MODEL TRY

```
from tensorflow.keras.callbacks import EarlyStopping
def build_cnn_model(input_shape=(224, 224, 3), num_classes=8): # The model expects 224×224 images with 3 color channels (RGB). num_classes=8 → The dataset has 8 different skin disease categories (e.g., MEL, NV, BCC, etc.).
   model = Sequential([ # Creates a model where layers are added one after another.
      Conv2D(32, (3,3), activation='relu', input_shape=input_shape), # Uses 32 filters (small patterns to detect edges, textures). Each filter is 3×3 in size (small region scanning). ReLU activation → Removes negative values (improves learning).
      MaxPooling2D(2,2), # Reduces image size by half (takes the most important features).
      BatchNormalization(), # Normalizes activations (faster training, avoids overfitting).

      Conv2D(64, (3,3), activation='relu'),
      MaxPooling2D(2,2),
      BatchNormalization(), #  Stabilizes training.

      Conv2D(128, (3,3), activation='relu'),
      MaxPooling2D(2,2), # Further reduces dimensions.
      BatchNormalization(), # keeps values normalized for stability.

      Flatten(), # Converts 2D feature maps into 1D vector for classification.
      Dense(128, activation='relu'), # Fully connected layer to learn patterns
      Dropout(0.5), # Drops 50% of neurons to prevent overfitting.
      Dense(num_classes, activation='softmax')  # inal layer outputs 8 values (1 per class). Softmax activation assigns probabilities to each class (total = 100%).

   ])

   model.compile(optimizer=Adam(learning_rate=0.0001),  # Adaptive learning optimizer (efficient gradient updates). Small learning rate (0.0001) ensures smooth learning.
                 loss='categorical_crossentropy', # Used because the dataset has multiple classes (8 classes). Computes error between predicted probabilities & actual labels.
                 metrics=['accuracy']) # Evaluates model performance based on accuracy.
```

```
    return model

# Create CNN model
cnn_model = build_cnn_model()
cnn_model.summary()
def build_inception_model(input_shape=(224, 224, 3), num_classes=8): # The model expects 224×224 images with 3 color channels (RGB). num_classes=8 → The dataset has 8 different skin disease categories (e.g., MEL, NV, BCC, etc.).
    base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=input_shape) # Loads InceptionV3 with pre-trained weights from ImageNet.
# include_top=False → Removes the fully connected layers of InceptionV3 (since we will add our own classifier).
# input_shape=input_shape ensures the model takes in 224×224 images.

    for layer in base_model.layers:
        layer.trainable = False  # Freeze base model layers # Freezes all layers in InceptionV3 to prevent modification during training.
# This helps retain the pre-trained features from ImageNet while we train only the new classifier.

    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.5)(x)
    output = Dense(num_classes, activation='softmax')(x)  # Multi-class classification # GlobalAveragePooling2D(): Replaces the removed InceptionV3 fully connected layers. It reduces feature maps into a single vector per image.
 # Dense(128, activation='relu'): Adds a fully connected (FC) layer with 128 neurons and ReLU activation to learn new patterns.
# Dropout(0.5): RandoDLy drops 50% of neurons to prevent overfitting.
# Dense(num_classes, activation='softmax'): Final layer for multi-class classification (8 classes)

    model = Model(inputs=base_model.input, outputs=output)
```

```
    model.compile(optimizer=Adam(learning_rate=0.0001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
# Adam(learning_rate=0.0001) → Optimizes the model using the Adam optimizer (efficient and widely used).
# categorical_crossentropy → Loss function used for multi-class classification.
# metrics=['accuracy'] → Model evaluates performance based on accuracy.

    return model

# Create InceptionV3 model
inception_model = build_inception_model()
inception_model.summary()
# Define early stopping to prevent overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train CNN
history_cnn = cnn_model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=7,
    callbacks=[early_stop]
)

# Train InceptionV3
history_inception = inception_model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=7,
    callbacks=[early_stop]
)
# Create a label map for your classes (adjust as needed)
label_map = {'MEL': 0, 'NV': 1, 'BCC': 2, 'AK': 3, 'BKL': 4, 'DF': 5, 'VASC': 6, 'SCC': 7}
```

```
# Map the labels in train_df
train_df['label'] = train_df['label'].map(label_map)

# Check the mapping
print(train_df['label'].value_counts())
from sklearn.metrics import accuracy_score, precision_score, f1_score
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Ensure generator is reset
val_generator.reset()

# Get true labels
y_true = val_generator.classes

# Predict using CNN
val_generator.reset()
y_pred_cnn = np.argmax(cnn_model.predict(val_generator, steps=len(val_generator)),
axis=1)

# Predict using InceptionV3
val_generator.reset()
y_pred_incep = np.argmax(inception_model.predict(val_generator,
steps=len(val_generator)), axis=1)
```

Metric function

```
def get_metrics(y_true, y_pred):
    acc = accuracy_score(y_true, y_pred) * 100
    prec = precision_score(y_true, y_pred, average='macro', zero_division=0) * 100
    f1 = f1_score(y_true, y_pred, average='macro', zero_division=0) * 100
    return round(acc, 2), round(prec, 2), round(f1, 2)
```

```
# Compute metrics
cnn_metrics = get_metrics(y_true, y_pred_cnn)
inception_metrics = get_metrics(y_true, y_pred_incep)

# Display table
df = pd.DataFrame({
    'Metric': ['Accuracy', 'Precision', 'F1-Score'],
    'CNN (%)': cnn_metrics,
    'InceptionV3 (%)': inception_metrics
})

print("📊 Model Evaluation (in %):")
print(df)

# Bar graph
df.set_index('Metric').plot(kind='bar', colormap='viridis', figsize=(8, 5))
plt.title('Model Comparison: CNN vs InceptionV3 (%)')
plt.ylabel('Percentage')
plt.ylim(0, 100)
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.tight_layout()
plt.show()

# Best model based on average
avg_cnn = np.mean(cnn_metrics)
avg_incep = np.mean(inception_metrics)

if avg_cnn > avg_incep:
    print(f"\n✅ Best Model: CNN (Average: {avg_cnn:.2f}%)")
else:
    print(f"\n✅ Best Model: InceptionV3 (Average: {avg_incep:.2f}%)")
```

 # Preprocess Test Image

```
import numpy as np
from tensorflow.keras.preprocessing import image

# Function to preprocess a test image
def preprocess_image(img_path):
    img = image.load_img(img_path, target_size=(224, 224))  # Resize
    img_array = image.img_to_array(img) / 255.0  # Normalize (0-1)
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    return img_array
```

 # Predict Using CNN & InceptionV3

```
# Path to the test image (update with actual test image)
test_img_path = "ISIC_2019_Test_Extracted/ISIC_2019_Test_Input/ISIC_0034332.jpg"

# Preprocess the image
img_array = preprocess_image(test_img_path)

# Predict using CNN & InceptionV3
cnn_pred = cnn_model.predict(img_array)
inception_pred = inception_model.predict(img_array)

# Print raw probabilities for debugging
print("\n 🔹 Raw Prediction Probabilities:")
print(f"CNN Prediction Probabilities: {cnn_pred}")
print(f"InceptionV3 Prediction Probabilities: {inception_pred}")

# Convert probabilities to class labels
cnn_class = np.argmax(cnn_pred, axis=1)[0]
inception_class = np.argmax(inception_pred, axis=1)[0]

print("\n 🔹 Model Predictions:")
print(f"CNN Model Prediction (Class Index): {cnn_class}")
print(f"InceptionV3 Model Prediction (Class Index): {inception_class}")
```

```
# Check if the final prediction is MEL (Melanoma)
if final_label == "MEL":
    print("\n⚠️ Melanoma Detected! ⚠️")
else:
    print("\n✅ No Melanoma Detected.")
```

5.3. Test cases

Step 4: Visualize Sample Images

```
[309]:
def show_random_images(dataframe, img_folder, num_images=5, title="Random Images"):
    fig, axes = plt.subplots(1, num_images, figsize=(15, 5))
    fig.suptitle(title, fontsize=16)

    for i in range(num_images):
        img_name = np.random.choice(dataframe['image'])  # Randomly select an image
        img_path = os.path.join(img_folder, img_name + ".jpg")  # Construct full path

        img = cv2.imread(img_path)  # Read image

        if img is None:  # ✅ Check if image was loaded successfully
            print(f"Error: Could not read image at {img_path}")
            continue  # Skip to next image

        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB

        axes[i].imshow(img)  # Display image
        axes[i].axis("off")  # Hide axes
        axes[i].set_title(f"Image: {img_name}")  # Set title

    plt.show()  # Show images

# Paths for training & test images
train_folder = "ISIC_2019_Training_Extracted/ISIC_2019_Training_Input"
test_folder = "ISIC_2019_Test_Extracted/ISIC_2019_Test_Input"

# Visualize Training Images
show_random_images(df, train_folder, num_images=5, title="Random Training Images")

# Visualize Test Images (Assuming test images do not have labels)
test_images = [f.split('.')[0] for f in os.listdir(test_folder) if f.endswith('.jpg')]
test_df = pd.DataFrame({'image': test_images})

# Visualize Test Images
show_random_images(test_df, test_folder, num_images=5, title="Random Test Images")
```

6. Output and Performance Analysis

6.1. Execution snapshots

jupyter MELANOMA ISIC Last Checkpoint: 11 days ago

File Edit View Run Kernel Settings Help — Not Trusted

Markdown — JupyterLab — Python 3 (ipykernel)

Step 1: Load Required Libraries

[1]:
```
import numpy as np # Handles arrays
import os # Works with files & folders
import zipfile # EXTRACT ZIP FILE
import cv2 # Reads, processes, and resizes images.
import matplotlib.pyplot as plt
import seaborn as sns # Makes heatmaps (confusion matrix visualization)
from collections import Counter # Counts occurrences of classes (e.g., melanoma vs. non-melanoma).
from tensorflow.keras.preprocessing.image import ImageDataGenerator # Augments images (rotation, flipping, etc.) to improve training.
from tensorflow.keras.models import Sequential, load_model # Builds and loads deep learning models.
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout # Layers for CNN model.
from tensorflow.keras.applications import InceptionV3 # Pre-trained deep learning model (transfer learning).
from tensorflow.keras.optimizers import Adam # Optimizer that improves model learning.
from sklearn.metrics import classification_report, confusion_matrix, precision_score, recall_score, f1_score
```

[2]:
```
!pip install opencv-python
```

```
Requirement already satisfied: opencv-python in c:\users\dell\appdata\local\programs\python\python312\lib\site-packages (4.11.0.86)
Requirement already satisfied: numpy>=1.21.2 in c:\users\dell\appdata\local\programs\python\python312\lib\site-packages (from opencv-python) (2.0.2)

[notice] A new release of pip is available: 24.2 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Step 2: Verify Extraction & List Images

[307]:
```
import os

# Define correct paths as strings
train_folder = "ISIC_2019_Training_Extracted/ISIC_2019_Training_Input"
test_folder = "ISIC_2019_Test_Extracted/ISIC_2019_Test_Input"

# Check if folders exist before listing images
if os.path.exists(train_folder):
    train_images = os.listdir(train_folder)
    print(f"✓ Total training images: {len(train_images)}")
else:
    print(f"❌ Training folder not found: {train_folder}")

if os.path.exists(test_folder):
    test_images = os.listdir(test_folder)
    print(f"✓ Total test images: {len(test_images)}")
else:
    print(f"❌ Test folder not found: {test_folder}")
```

```
✓ Total training images: 25333
✓ Total test images: 8241
```

Step 3: Load CSV Files & Display Dataset Information

[308]:
```
import pandas as pd  # Import pandas

# Load the metadata CSV file
csv_path = "ISIC_2019_Training_GroundTruth (1).csv"
df = pd.read_csv(csv_path)

# Display first 5 rows
print("First 5 rows of CSV file:")
print(df.head())

# Check number of unique classes
print("\nUnique classes in dataset:")
print(df.columns[1:])  # Labels start from column 1

# Show column names
print("\nColumn names in the dataset:")
print(df.columns)

# Check total number of images
print(f"\nTotal images available: {df.shape[0]}")
```

```
First 5 rows of CSV file:
         image  MEL   NV  BCC   AK  BKL   DF  VASC  SCC  UNK
0  ISIC_0000000  0.0  1.0  0.0  0.0  0.0  0.0   0.0  0.0  0.0
1  ISIC_0000001  0.0  1.0  0.0  0.0  0.0  0.0   0.0  0.0  0.0
2  ISIC_0000002  1.0  0.0  0.0  0.0  0.0  0.0   0.0  0.0  0.0
3  ISIC_0000003  0.0  1.0  0.0  0.0  0.0  0.0   0.0  0.0  0.0
4  ISIC_0000004  1.0  0.0  0.0  0.0  0.0  0.0   0.0  0.0  0.0

Unique classes in dataset:
Index(['MEL', 'NV', 'BCC', 'AK', 'BKL', 'DF', 'VASC', 'SCC', 'UNK'], dtype='object')

Column names in the dataset:
Index(['image', 'MEL', 'NV', 'BCC', 'AK', 'BKL', 'DF', 'VASC', 'SCC', 'UNK'], dtype='object')

Total images available: 25331
```

jupyter MELANOMA ISIC Last Checkpoint: 11 days ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

Step 4: Visualize Sample Images

[309]:

```
def show_random_images(dataframe, img_folder, num_images=5, title="Random Images"):
    fig, axes = plt.subplots(1, num_images, figsize=(15, 5))
    fig.suptitle(title, fontsize=16)

    for i in range(num_images):
        img_name = np.random.choice(dataframe['image'])  # Randomly select an image
        img_path = os.path.join(img_folder, img_name + ".jpg")  # Construct full path

        img = cv2.imread(img_path)  # Read image

        if img is None:  # ✅ Check if image was loaded successfully
            print(f"Error: Could not read image at {img_path}")
            continue  # Skip to next image

        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB

        axes[i].imshow(img)  # Display image
        axes[i].axis("off")  # Hide axes
        axes[i].set_title(f"Image: {img_name}")  # Set title

    plt.show()  # Show images

# Paths for training & test images
train_folder = "ISIC_2019_Training_Extracted/ISIC_2019_Training_Input"
test_folder = "ISIC_2019_Test_Extracted/ISIC_2019_Test_Input"

# Visualize Training Images
show_random_images(df, train_folder, num_images=5, title="Random Training Images")

# Visualize Test Images (Assuming test images do not have labels)
test_images = [f.split('.')[0] for f in os.listdir(test_folder) if f.endswith('.jpg')]
test_df = pd.DataFrame({'image': test_images})

# Visualize Test Images
show_random_images(test_df, test_folder, num_images=5, title="Random Test Images")
```

Random Training Images

Random Test Images

jupyter MELANOMA ISIC Last Checkpoint: 11 days ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

Step 5: Data Preprocessing & Augmentation

[310]:
```
# Define image size
IMG_SIZE = (224, 224) #  Target image size for the model ( for deep learning models their will be fixed size)

# Create a function to preprocess images
def load_and_preprocess_image(img_path): # Loads an image from a file and applies preprocessing to prepare it for model prediction.
    img = cv2.imread(img_path) # Loads an image in BGR format (default for OpenCV)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, IMG_SIZE)
    img = img / 255.0  # Normalize Pixel Values (Scale Between 0 and 1) Normalizing between 0 and 1 improves model training by stabilizing learning.
    return img
```

TRY

[311]:
```
import pandas as pd # for handling structured data (CSV files, tables, datasets, etc.).
import tensorflow as tf # TensorFlow is an open-source deep learning framework used for building, training, and deploying machine learning models.
from sklearn.model_selection import train_test_split
```

[ ]:

[135]:
```
import os
import pandas as pd

# ✅ 1. Set the correct image folder path
train_extract_path = "ISIC_2019_Training_Extracted/ISIC_2019_Training_Input"

# ✅ 2. Check a few file names from folder
image_files = [f for f in os.listdir(train_extract_path) if f.endswith('.jpg')]
print(f"Total image files found in folder: {len(image_files)}")
print("Sample filenames from folder:", image_files[:5])

# ✅ 3. Check your DataFrame
print("\nFirst few image names in your DataFrame BEFORE adding .jpg:")
print(df['image'].head())

# ✅ 4. Add .jpg extension
df["image"] = df["image"].astype(str) + ".jpg"

print("\nAfter adding .jpg:")
print(df["image"].head())

# ✅ 5. Check which image names are NOT in the folder
existing_images = set(image_files)
df["exists"] = df["image"].apply(lambda x: x in existing_images)

# ✅ 6. Show how many exist and how many don't
print("\nImage existence check:")
print(df["exists"].value_counts())
```

```
# ✅ 7. Filter DataFrame to keep only existing images
df = df[df["exists"] == True].drop(columns=["exists"])
print(f"\n✅ Updated DataFrame size: {df.shape}")
```

```
Total image files found in folder: 25331
Sample filenames from folder: ['ISIC_0000000.jpg', 'ISIC_0000001.jpg', 'ISIC_0000002.jpg', 'ISIC_0000003.jpg', 'ISIC_0000004.jpg']

First few image names in your DataFrame BEFORE adding .jpg:
0    ISIC_0000000
1    ISIC_0000001
2    ISIC_0000002
3    ISIC_0000003
4    ISIC_0000004
Name: image, dtype: object

After adding .jpg:
0    ISIC_0000000.jpg
1    ISIC_0000001.jpg
2    ISIC_0000002.jpg
3    ISIC_0000003.jpg
4    ISIC_0000004.jpg
Name: image, dtype: object

Image existence check:
exists
True    25331
Name: count, dtype: int64

✅ Updated DataFrame size: (25331, 10)
```

jupyter MELANOMA ISIC Last Checkpoint: 11 days ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

Apply Data Augmentation

[314]:
```
import shutil # used for copying, moving, renaming, and deleting files & directories.
from tensorflow.keras.utils import to_categorical # Convert Labels to One-Hot Encoding
from tqdm import tqdm # makes loops visually trackable by displaying a progress bar. Useful for iterating over large datasets efficiently.
```

[315]:
```
# Define paths for training images, test images, and labels
train_img_dir = "ISIC_2019_Training_Extracted/ISIC_2019_Training_Input"
test_img_dir = "ISIC_2019_Test_Extracted/ISIC_2019_Test_Input"
train_csv_path = "ISIC_2019_Training_GroundTruth (1).csv"

# Ensure directories exist
assert os.path.exists(train_img_dir), f"Training image directory not found: {train_img_dir}"
assert os.path.exists(test_img_dir), f"Test image directory not found: {test_img_dir}"
assert os.path.exists(train_csv_path), f"Training CSV file not found: {train_csv_path}"
```

```
# Check for missing values
print("Missing Values in Labels:")
print(df.isnull().sum())

# Get all class columns (disease types)
label_columns = df.columns[1:]  # Skipping the first column (image ID)

# Ensure only one label per image
df['num_labels'] = df[label_columns].sum(axis=1)
assert df['num_labels'].max() == 1, "Some images have multiple labels!"

# Convert multi-label to single-label (find the column with 1)
df['label'] = df[label_columns].idxmax(axis=1)

# Drop unnecessary columns
df = df[['image', 'label']]
```

```
Missing Values in Labels:
image    0
MEL      0
NV       0
BCC      0
AK       0
BKL      0
DF       0
VASC     0
SCC      0
UNK      0
dtype: int64
```

[317]:

```
# Load the CSV file
df = pd.read_csv("ISIC_2019_Training_GroundTruth (1).csv")

# Print initial preview
print("Original Data Labels Preview:")
print(df.head())

# Ensure column names are correct
print("\nColumn Names in CSV:")
print(df.columns.tolist())

# Get all class columns (excluding 'image')
label_columns = df.columns[1:]  # Exclude 'image' column

# Debug: Check if label_columns contains class names
print("\nClass Columns Detected:", label_columns)

# Convert one-hot encoding to single-label classification
df['label'] = df[label_columns].idxmax(axis=1)  # Get the class with max value

# Drop the original one-hot encoded columns
df = df[['image', 'label']]

# Print final output
print("\nUpdated Data Labels:")
print(df.head())
```

```
Original Data Labels Preview:
          image  MEL   NV  BCC   AK  BKL   DF  VASC  SCC  UNK
0  ISIC_0000000  0.0  1.0  0.0  0.0  0.0  0.0   0.0  0.0  0.0
1  ISIC_0000001  0.0  1.0  0.0  0.0  0.0  0.0   0.0  0.0  0.0
2  ISIC_0000002  1.0  0.0  0.0  0.0  0.0  0.0   0.0  0.0  0.0
3  ISIC_0000003  0.0  1.0  0.0  0.0  0.0  0.0   0.0  0.0  0.0
4  ISIC_0000004  1.0  0.0  0.0  0.0  0.0  0.0   0.0  0.0  0.0

Column Names in CSV:
['image', 'MEL', 'NV', 'BCC', 'AK', 'BKL', 'DF', 'VASC', 'SCC', 'UNK']

Class Columns Detected: Index(['MEL', 'NV', 'BCC', 'AK', 'BKL', 'DF', 'VASC', 'SCC', 'UNK'], dtype='object')

Updated Data Labels:
          image label
0  ISIC_0000000    NV
1  ISIC_0000001    NV
2  ISIC_0000002   MEL
3  ISIC_0000003    NV
4  ISIC_0000004   MEL
```

jupyter MELANOMA ISIC Last Checkpoint: 11 days ago

File Edit View Run Kernel Settings Help

Not Trusted

Code

JupyterLab Python 3 (ipykernel)

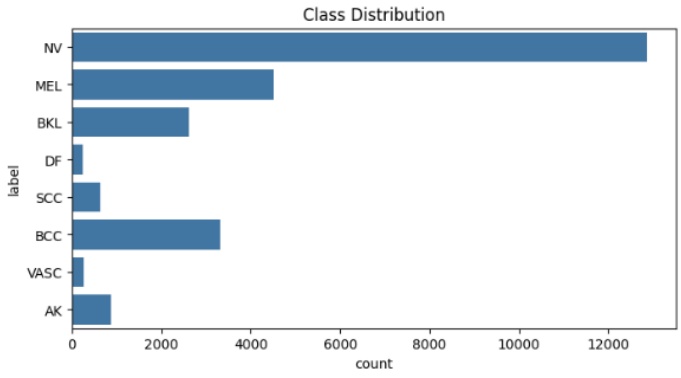
[319]:

```
# Convert label column to string (required for ImageDataGenerator with class_mode='binary')
df['label'] = df['label'].astype(str)

# Display class distribution
print("Class Distribution:")
print(df['label'].value_counts())

# Plot class distribution
plt.figure(figsize=(8, 4))
sns.countplot(y=df['label'])
plt.title("Class Distribution")
plt.show()
```

```
Class Distribution:
label
NV      12875
MEL      4522
BCC      3323
BKL      2624
AK        867
SCC       628
VASC      253
DF        239
Name: count, dtype: int64
```



jupyter MELANOMA ISIC Last Checkpoint: 11 days ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

[320]:
```
# Split data into training and validation sets (80% train, 20% validation)
train_df, val_df = train_test_split(df, test_size=0.2, stratify=df['label'], random_state=42)

# Reset index
train_df = train_df.reset_index(drop=True)
val_df = val_df.reset_index(drop=True)

print("Training Samples:", len(train_df))
print("Validation Samples:", len(val_df))
```

```
Training Samples: 20264
Validation Samples: 5067
```

[321]:
```
print(train_df["label"].unique())  # Should output ['0', '1']
print(val_df["label"].unique())    # Should output ['0', '1']
```

```
['NV' 'MEL' 'BCC' 'SCC' 'AK' 'BKL' 'DF' 'VASC']
['BKL' 'SCC' 'MEL' 'BCC' 'AK' 'NV' 'DF' 'VASC']
```

[322]:
```
train_df['label'] = train_df['label'].astype(str)  # Ensure labels are strings
val_df['label'] = val_df['label'].astype(str)
```

[330]:
```
# Print first few image names from dataframe
print(train_df["image"].head())
print(val_df["image"].head())
```

```
0    ISIC_0033283.jpg
1    ISIC_0055317.jpg
2    ISIC_0056356.jpg
3    ISIC_0070561.jpg
4    ISIC_0066198.jpg
Name: image, dtype: object
0    ISIC_0028977.jpg
1    ISIC_0031738.jpg
2    ISIC_0057361.jpg
3    ISIC_0058026.jpg
4    ISIC_0025376.jpg
Name: image, dtype: object
```

[324]:
```
train_df["image"] = train_df["image"] + ".jpg"
val_df["image"] = val_df["image"] + ".jpg"
```

jupyter MELANOMA ISIC Last Checkpoint: 11 days ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

[325]:
```
import os

train_img_dir = "ISIC_2019_Training_Extracted/ISIC_2019_Training_Input"

# List first 10 images in the directory
print("Files in image directory:", os.listdir(train_img_dir)[:10])
```

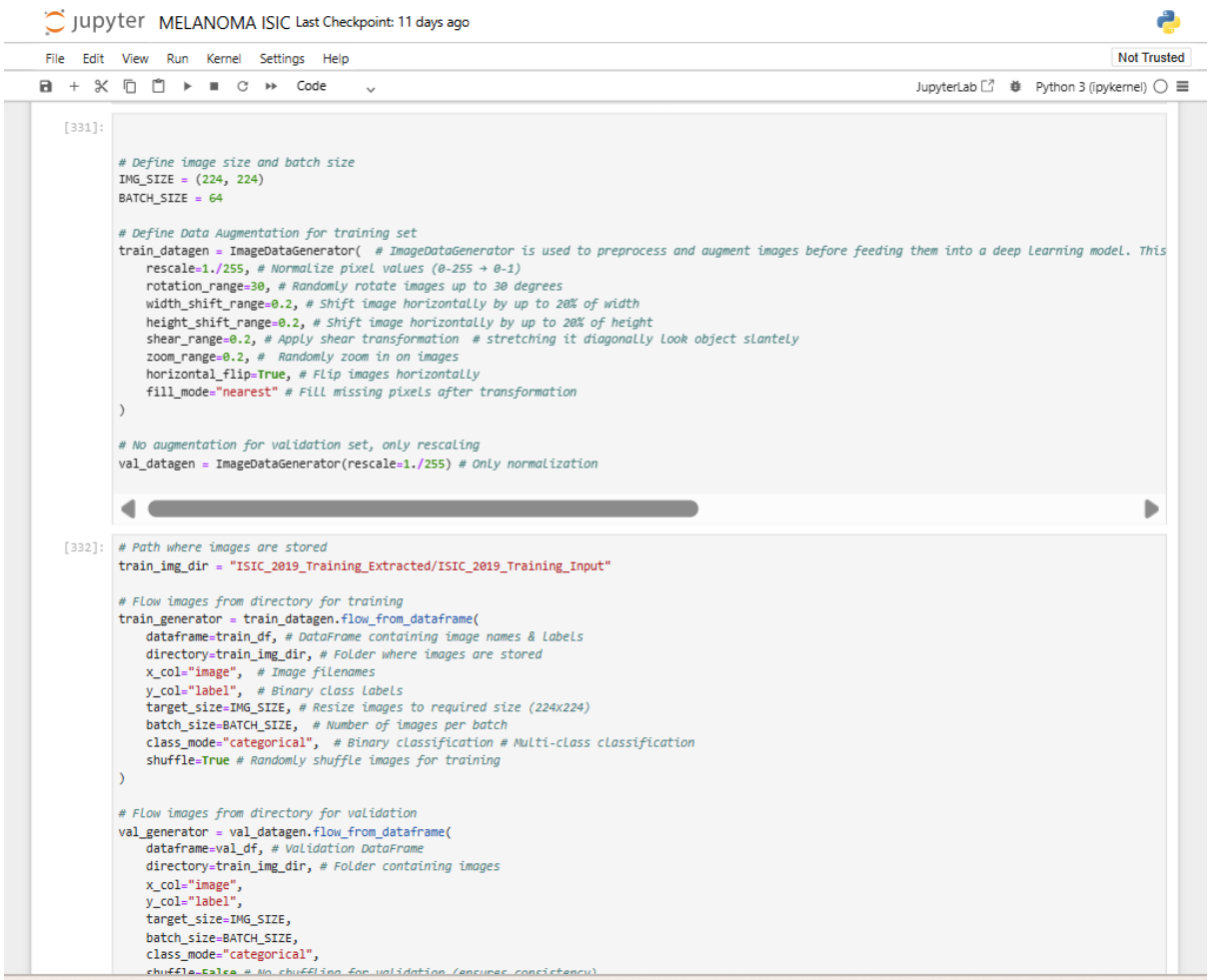
```
Files in image directory: ['ATTRIBUTION.txt', 'ISIC_0000000.jpg', 'ISIC_0000001.jpg', 'ISIC_0000002.jpg', 'ISIC_0000003.jpg', 'ISIC_0000004.jpg', 'ISIC
_0000006.jpg', 'ISIC_0000007.jpg', 'ISIC_0000008.jpg', 'ISIC_0000009.jpg']
```

[326]:
```
...
```

[326]:
```
Ellipsis
```

[327]:
```
print(train_df["image"].head())  # Should contain only filenames (e.g., "ISIC_0000001.jpg")
```

```
0    ISIC_0033283.jpg
1    ISIC_0055317.jpg
2    ISIC_0056356.jpg
3    ISIC_0070561.jpg
4    ISIC_0066198.jpg
Name: image, dtype: object
```




```
    x_col="image",
    y_col="label",
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False # No shuffling for validation (ensures consistency)
)

# Verify class indices
print("Class indices:", train_generator.class_indices)
```

```
Found 20264 validated image filenames belonging to 8 classes.
Found 5067 validated image filenames belonging to 8 classes.
Class indices: {'AK': 0, 'BCC': 1, 'BKL': 2, 'DF': 3, 'MEL': 4, 'NV': 5, 'SCC': 6, 'VASC': 7}
```

MODEL TRY

[149]:
```
from tensorflow.keras.callbacks import EarlyStopping
```

[150]:
```
from tensorflow.keras.layers import BatchNormalization
def build_cnn_model(input_shape=(224, 224, 3), num_classes=8): # The model expects 224×224 images with 3 color channels (RGB). num_classes=8 → The data
    model = Sequential([ # Creates a model where layers are added one after another.
        Conv2D(32, (3,3), activation='relu', input_shape=input_shape), # Uses 32 filters (small patterns to detect edges, textures). Each filter is 3×3
        MaxPooling2D(2,2), # Reduces image size by half (takes the most important features).
        BatchNormalization(), # Normalizes activations (faster training, avoids overfitting).

        Conv2D(64, (3,3), activation='relu'),
        MaxPooling2D(2,2),
        BatchNormalization(), #  Stabilizes training.

        Conv2D(128, (3,3), activation='relu'),
        MaxPooling2D(2,2), # Further reduces dimensions.
        BatchNormalization(), # keeps values normalized for stability.

        Flatten(), # Converts 2D feature maps into 1D vector for classification.
        Dense(128, activation='relu'), # Fully connected layer to learn patterns
        Dropout(0.5), # Drops 50% of neurons to prevent overfitting.
        Dense(num_classes, activation='softmax')  # inal layer outputs 8 values (1 per class). Softmax activation assigns probabilities to each class (

    ])

    model.compile(optimizer=Adam(learning_rate=0.0001),  # Adaptive learning optimizer (efficient gradient updates). Small learning rate (0.0001) ensur
                  loss='categorical_crossentropy', # Used because the dataset has multiple classes (8 classes). Computes error between predicted probab
                  metrics=['accuracy']) # Evaluates model performance based on accuracy.
    return model

# Create CNN model
cnn_model = build_cnn_model()
cnn_model.summary()
```

```
C:\Users\Dell\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `i
nput_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instea
d.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_403 (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d_43 (MaxPooling2D)	(None, 111, 111, 32)	0
batch_normalization_391 (BatchNormalization)	(None, 111, 111, 32)	128
conv2d_404 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_44 (MaxPooling2D)	(None, 54, 54, 64)	0
batch_normalization_392 (BatchNormalization)	(None, 54, 54, 64)	256
conv2d_405 (Conv2D)	(None, 52, 52, 128)	73,856
max_pooling2d_45 (MaxPooling2D)	(None, 26, 26, 128)	0
batch_normalization_393 (BatchNormalization)	(None, 26, 26, 128)	512
flatten_9 (Flatten)	(None, 86528)	0
dense_26 (Dense)	(None, 128)	11,075,712
dropout_13 (Dropout)	(None, 128)	0
dense_27 (Dense)	(None, 8)	1,032

Total params: 11,170,888 (42.61 MB)

Trainable params: 11,170,440 (42.61 MB)

Non-trainable params: 448 (1.75 KB)

[151]:
```
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.models import Model

def build_inception_model(input_shape=(224, 224, 3), num_classes=8): # The model expects 224×224 images with 3 color channels (RGB). num_classes=8 → Th
    base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=input_shape) # Loads InceptionV3 with pre-trained weights from ImageNet
# include_top=False → Removes the fully connected layers of InceptionV3 (since we will add our own classifier).
# input_shape=input_shape ensures the model takes in 224×224 images.


    for layer in base_model.layers:
        layer.trainable = False  # Freeze base model layers # Freezes all layers in InceptionV3 to prevent modification during training.
# This helps retain the pre-trained features from ImageNet while we train only the new classifier.

    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.5)(x)
    output = Dense(num_classes, activation='softmax')(x)  # Multi-class classification # GlobalAveragePooling2D(): Replaces the removed InceptionV3 ful
 # Dense(128, activation='relu'): Adds a fully connected (FC) layer with 128 neurons and ReLU activation to learn new patterns.
# Dropout(0.5): Randomly drops 50% of neurons to prevent overfitting.
# Dense(num_classes, activation='softmax'): Final layer for multi-class classification (8 classes)

    model = Model(inputs=base_model.input, outputs=output)

    model.compile(optimizer=Adam(learning_rate=0.0001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
# Adam(learning_rate=0.0001) → Optimizes the model using the Adam optimizer (efficient and widely used).
# categorical_crossentropy → Loss function used for multi-class classification.
# metrics=['accuracy'] → Model evaluates performance based on accuracy.

    return model

# Create InceptionV3 model
inception_model = build_inception_model()
inception_model.summary()
```

Model: "functional_46"

Layer (type)	Output Shape	Param #	Connected to
input_layer_14 (InputLayer)	(None, 224, 224, 3)	0	-
conv2d_406 (Conv2D)	(None, 111, 111, 32)	864	input_layer_14[0][0]
batch_normalization_394 (BatchNormalization)	(None, 111, 111, 32)	96	conv2d_406[0][0]
activation_376 (Activation)	(None, 111, 111, 32)	0	batch_normalization_394[0…
conv2d_407 (Conv2D)	(None, 109, 109, 32)	9,216	activation_376[0][0]
batch_normalization_395 (BatchNormalization)	(None, 109, 109, 32)	96	conv2d_407[0][0]
activation_377 (Activation)	(None, 109, 109, 32)	0	batch_normalization_395[0…
conv2d_408 (Conv2D)	(None, 109, 109, 64)	18,432	activation_377[0][0]
batch_normalization_396 (BatchNormalization)	(None, 109, 109, 64)	192	conv2d_408[0][0]
activation_378 (Activation)	(None, 109, 109, 64)	0	batch_normalization_396[0…
max_pooling2d_46 (MaxPooling2D)	(None, 54, 54, 64)	0	activation_378[0][0]
conv2d_409 (Conv2D)	(None, 54, 54, 80)	5,120	max_pooling2d_46[0][0]
batch_normalization_397 (BatchNormalization)	(None, 54, 54, 80)	240	conv2d_409[0][0]
activation_379 (Activation)	(None, 54, 54, 80)	0	batch_normalization_397[0…
conv2d_410 (Conv2D)	(None, 52, 52, 192)	138,240	activation_379[0][0]
batch_normalization_398 (BatchNormalization)	(None, 52, 52, 192)	576	conv2d_410[0][0]
activation_380 (Activation)	(None, 52, 52, 192)	0	batch_normalization_398[0…
max_pooling2d_47 (MaxPooling2D)	(None, 25, 25, 192)	0	activation_380[0][0]
conv2d_414 (Conv2D)	(None, 25, 25, 64)	12,288	max_pooling2d_47[0][0]
batch_normalization_402 (BatchNormalization)	(None, 25, 25, 64)	192	conv2d_414[0][0]
activation_384 (Activation)	(None, 25, 25, 64)	0	batch_normalization_402[0…

activation_387 (Activation)	(None, 25, 25, 32)	0	batch_normalization_405[0…
mixed0 (Concatenate)	(None, 25, 25, 256)	0	activation_381[0][0], activation_383[0][0], activation_386[0][0], activation_387[0][0]
conv2d_421 (Conv2D)	(None, 25, 25, 64)	16,384	mixed0[0][0]
batch_normalization_409 (BatchNormalization)	(None, 25, 25, 64)	192	conv2d_421[0][0]
activation_391 (Activation)	(None, 25, 25, 64)	0	batch_normalization_409[0…
conv2d_419 (Conv2D)	(None, 25, 25, 48)	12,288	mixed0[0][0]
conv2d_422 (Conv2D)	(None, 25, 25, 96)	55,296	activation_391[0][0]
batch_normalization_407 (BatchNormalization)	(None, 25, 25, 48)	144	conv2d_419[0][0]
batch_normalization_410 (BatchNormalization)	(None, 25, 25, 96)	288	conv2d_422[0][0]
activation_389 (Activation)	(None, 25, 25, 48)	0	batch_normalization_407[0…
activation_392 (Activation)	(None, 25, 25, 96)	0	batch_normalization_410[0…
average_pooling2d_37 (AveragePooling2D)	(None, 25, 25, 256)	0	mixed0[0][0]
conv2d_418 (Conv2D)	(None, 25, 25, 64)	16,384	mixed0[0][0]
conv2d_420 (Conv2D)	(None, 25, 25, 64)	76,800	activation_389[0][0]
conv2d_423 (Conv2D)	(None, 25, 25, 96)	82,944	activation_392[0][0]
conv2d_424 (Conv2D)	(None, 25, 25, 64)	16,384	average_pooling2d_37[0][0]
batch_normalization_406 (BatchNormalization)	(None, 25, 25, 64)	192	conv2d_418[0][0]
batch_normalization_408 (BatchNormalization)	(None, 25, 25, 64)	192	conv2d_420[0][0]
batch_normalization_411 (BatchNormalization)	(None, 25, 25, 96)	288	conv2d_423[0][0]
batch_normalization_412 (BatchNormalization)	(None, 25, 25, 64)	192	conv2d_424[0][0]
activation_388 (Activation)	(None, 25, 25, 64)	0	batch_normalization_406[0…
activation_390 (Activation)	(None, 25, 25, 64)	0	batch_normalization_408[0…
activation_393 (Activation)	(None, 25, 25, 96)	0	batch_normalization_411[0…
conv2d_429 (Conv2D)	(None, 25, 25, 96)	55,296	activation_398[0][0]
batch_normalization_414 (BatchNormalization)	(None, 25, 25, 48)	144	conv2d_426[0][0]
batch_normalization_417 (BatchNormalization)	(None, 25, 25, 96)	288	conv2d_429[0][0]
activation_396 (Activation)	(None, 25, 25, 48)	0	batch_normalization_414[0…
activation_399 (Activation)	(None, 25, 25, 96)	0	batch_normalization_417[0…
average_pooling2d_38 (AveragePooling2D)	(None, 25, 25, 288)	0	mixed1[0][0]
conv2d_425 (Conv2D)	(None, 25, 25, 64)	18,432	mixed1[0][0]
conv2d_427 (Conv2D)	(None, 25, 25, 64)	76,800	activation_396[0][0]
conv2d_430 (Conv2D)	(None, 25, 25, 96)	82,944	activation_399[0][0]
conv2d_431 (Conv2D)	(None, 25, 25, 64)	18,432	average_pooling2d_38[0][0]
batch_normalization_413 (BatchNormalization)	(None, 25, 25, 64)	192	conv2d_425[0][0]
batch_normalization_415 (BatchNormalization)	(None, 25, 25, 64)	192	conv2d_427[0][0]
batch_normalization_418 (BatchNormalization)	(None, 25, 25, 96)	288	conv2d_430[0][0]
batch_normalization_419 (BatchNormalization)	(None, 25, 25, 64)	192	conv2d_431[0][0]
activation_395 (Activation)	(None, 25, 25, 64)	0	batch_normalization_413[0…
activation_397 (Activation)	(None, 25, 25, 64)	0	batch_normalization_415[0…
activation_400 (Activation)	(None, 25, 25, 96)	0	batch_normalization_418[0…
activation_401 (Activation)	(None, 25, 25, 64)	0	batch_normalization_419[0…
mixed2 (Concatenate)	(None, 25, 25, 288)	0	activation_395[0][0], activation_397[0][0], activation_400[0][0], activation_401[0][0]
conv2d_433 (Conv2D)	(None, 25, 25, 64)	18,432	mixed2[0][0]
batch_normalization_421 (BatchNormalization)	(None, 25, 25, 64)	192	conv2d_433[0][0]
activation_403 (Activation)	(None, 25, 25, 64)	0	batch_normalization_421[0…
conv2d_434 (Conv2D)	(None, 25, 25, 96)	55,296	activation_403[0][0]

batch_normalization_485 (BatchNormalization)	(None, 5, 5, 384)	1,152	conv2d_497[0][0]
batch_normalization_486 (BatchNormalization)	(None, 5, 5, 384)	1,152	conv2d_498[0][0]
conv2d_499 (Conv2D)	(None, 5, 5, 192)	393,216	average_pooling2d_44[0][0]
batch_normalization_479 (BatchNormalization)	(None, 5, 5, 320)	960	conv2d_491[0][0]
activation_463 (Activation)	(None, 5, 5, 384)	0	batch_normalization_481[0…
activation_464 (Activation)	(None, 5, 5, 384)	0	batch_normalization_482[0…
activation_467 (Activation)	(None, 5, 5, 384)	0	batch_normalization_485[0…
activation_468 (Activation)	(None, 5, 5, 384)	0	batch_normalization_486[0…
batch_normalization_487 (BatchNormalization)	(None, 5, 5, 192)	576	conv2d_499[0][0]
activation_461 (Activation)	(None, 5, 5, 320)	0	batch_normalization_479[0…
mixed9_1 (Concatenate)	(None, 5, 5, 768)	0	activation_463[0][0], activation_464[0][0]
concatenate_9 (Concatenate)	(None, 5, 5, 768)	0	activation_467[0][0], activation_468[0][0]
activation_469 (Activation)	(None, 5, 5, 192)	0	batch_normalization_487[0…
mixed10 (Concatenate)	(None, 5, 5, 2048)	0	activation_461[0][0], mixed9_1[0][0], concatenate_9[0][0], activation_469[0][0]
global_average_pooling2d_4 (GlobalAveragePooling2D)	(None, 2048)	0	mixed10[0][0]
dense_28 (Dense)	(None, 128)	262,272	global_average_pooling2d_…
dropout_14 (Dropout)	(None, 128)	0	dense_28[0][0]
dense_29 (Dense)	(None, 8)	1,032	dropout_14[0][0]

Total params: 22,066,088 (84.18 MB)

Trainable params: 263,304 (1.00 MB)

Non-trainable params: 21,802,784 (83.17 MB)

[77]:
```
# Define early stopping to prevent overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train CNN
history_cnn = cnn_model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=20,
    callbacks=[early_stop]
)

# Train InceptionV3
history_inception = inception_model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=20,
    callbacks=[early_stop]
)
```

[152]:
```
from tensorflow.keras.models import load_model
cnn_model = load_model("cnn_model (1).h5")
inception_model = load_model("inception_model (1).h5")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
```

[153]:
```
# Create a label map for your classes (adjust as needed)
label_map = {'MEL': 0, 'NV': 1, 'BCC': 2, 'AK': 3, 'BKL': 4, 'DF': 5, 'VASC': 6, 'SCC': 7}

# Map the labels in train_df
train_df['label'] = train_df['label'].map(label_map)

# Check the mapping
print(train_df['label'].value_counts())
```

```
label
1    10300
0     3618
2     2658
4     2099
3      694
7      502
6      202
5      191
Name: count, dtype: int64
```

[154]:
```
from sklearn.metrics import accuracy_score, precision_score, f1_score
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Ensure generator is reset
val_generator.reset()

# Get true labels
y_true = val_generator.classes

# Predict using CNN
val_generator.reset()
y_pred_cnn = np.argmax(cnn_model.predict(val_generator, steps=len(val_generator)), axis=1)

# Predict using InceptionV3
val_generator.reset()
y_pred_incep = np.argmax(inception_model.predict(val_generator, steps=len(val_generator)), axis=1)
```

```
C:\Users\Dell\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
80/80 ━━━━━━━━━━━━━━━━━━━━ 150s 2s/step
80/80 ━━━━━━━━━━━━━━━━━━━━ 260s 3s/step
```

[125]:
```
# Metric function
def get_metrics(y_true, y_pred):
    acc = accuracy_score(y_true, y_pred) * 100
    prec = precision_score(y_true, y_pred, average='macro', zero_division=0) * 100
    f1 = f1_score(y_true, y_pred, average='macro', zero_division=0) * 100
    return round(acc, 2), round(prec, 2), round(f1, 2)

# Compute metrics
cnn_metrics = get_metrics(y_true, y_pred_cnn)
inception_metrics = get_metrics(y_true, y_pred_incep)

# Display table
df = pd.DataFrame({
    'Metric': ['Accuracy', 'Precision', 'F1-Score'],
    'CNN (%)': cnn_metrics,
    'InceptionV3 (%)': inception_metrics
})

print("📊 Model Evaluation (in %):")
print(df)

# Bar graph
df.set_index('Metric').plot(kind='bar', colormap='viridis', figsize=(8, 5))
plt.title('Model Comparison: CNN vs InceptionV3 (%)')
plt.ylabel('Percentage')
plt.ylim(0, 100)
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

```
# Best model based on average
avg_cnn = np.mean(cnn_metrics)
avg_incep = np.mean(inception_metrics)

if avg_cnn > avg_incep:
    print(f"\n✅ Best Model: CNN (Average: {avg_cnn:.2f}%)")
else:
    print(f"\n✅ Best Model: InceptionV3 (Average: {avg_incep:.2f}%)")
```

Preprocess Test Image

[337]:
```
import numpy as np
from tensorflow.keras.preprocessing import image

# Function to preprocess a test image
def preprocess_image(img_path):
    img = image.load_img(img_path, target_size=(224, 224))  # Resize
    img_array = image.img_to_array(img) / 255.0  # Normalize (0-1)
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    return img_array
```

Predict Using CNN & InceptionV3

[32]:
```
# Path to the test image (update with actual test image)
test_img_path = "ISIC_2019_Test_Extracted/ISIC_2019_Test_Input/ISIC_0034332.jpg"

# Preprocess the image
img_array = preprocess_image(test_img_path)

# Predict using CNN & InceptionV3
cnn_pred = cnn_model.predict(img_array)
inception_pred = inception_model.predict(img_array)

# Print raw probabilities for debugging
print("\n🔹 Raw Prediction Probabilities:")
print(f"CNN Prediction Probabilities: {cnn_pred}")
print(f"InceptionV3 Prediction Probabilities: {inception_pred}")

# Convert probabilities to class labels
cnn_class = np.argmax(cnn_pred, axis=1)[0]
inception_class = np.argmax(inception_pred, axis=1)[0]

print("\n🔹 Model Predictions:")
print(f"CNN Model Prediction (Class Index): {cnn_class}")
print(f"InceptionV3 Model Prediction (Class Index): {inception_class}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 192ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step

🔹 Raw Prediction Probabilities:
CNN Prediction Probabilities: [[1.0825823e-03 6.2582147e-01 1.4140760e-02 3.2524395e-04 5.7990267e-04
  3.5461587e-01 1.0061577e-03 2.4279957e-03]]
InceptionV3 Prediction Probabilities: [[0.05231854 0.17789435 0.11159055 0.06041492 0.1005499  0.40278146
  0.0618153  0.03263494]]

🔹 Model Predictions:
CNN Model Prediction (Class Index): 1
InceptionV3 Model Prediction (Class Index): 5
```

[127]:
```
# Check if the final prediction is MEL (Melanoma)
if final_label == "MEL":
    print("\n⚠️ Melanoma Detected! ⚠️")
else:
    print("\n✅ No Melanoma Detected.")
```

```
✅ No Melanoma Detected.
```

[ ]:

[ ]:

[ ]:

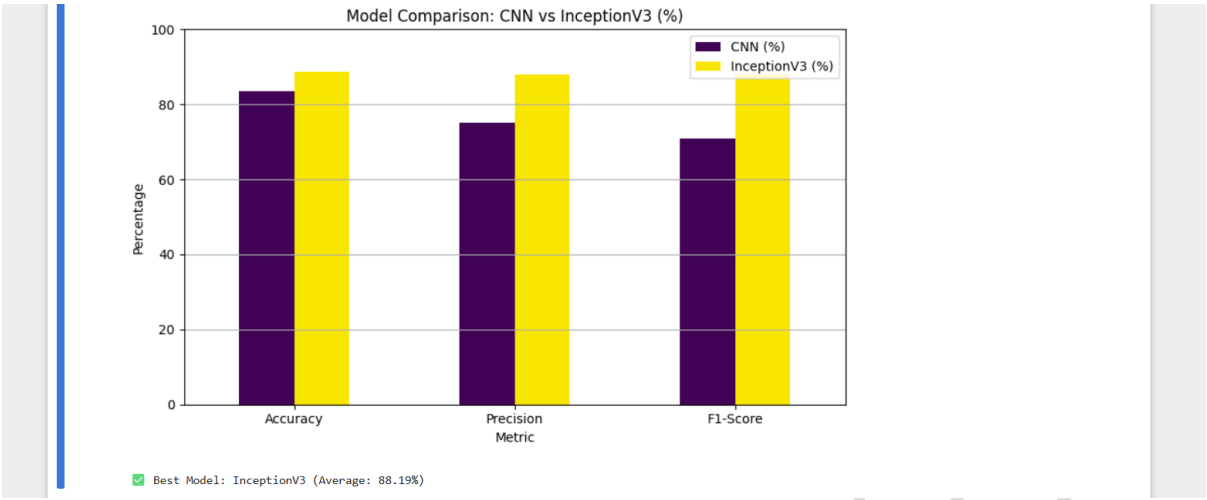
6.2. Output – in terms of performance metrics

```
# Best model based on average
avg_cnn = np.mean(cnn_metrics)
avg_incep = np.mean(inception_metrics)

if avg_cnn > avg_incep:
    print(f"\n✅ Best Model: CNN (Average: {avg_cnn:.2f}%)")
else:
    print(f"\n✅ Best Model: InceptionV3 (Average: {avg_incep:.2f}%)")
```

```
📊 Model Evaluation (in %):
      Metric  CNN (%)  InceptionV3 (%)
0   Accuracy    83.61            88.67
1  Precision    75.01            87.96
2   F1-Score    70.98            87.95
```

6.3. Performance comparison with existing works



7. Conclusion and Future Directions

Deep learning (DL) for melanoma skin cancer detection has yielded revolutionary potential, predominantly in the detection and treatment in the early stages. DL models and convolutional neural networks (CNNs), offer a suitable way to streamline and enhance the detection process through the evaluation of skin lesion images. This is particularly necessary to overcome the limitations of traditional diagnostic methods, which involve subjectivity and cumbersome procedures. DL systems have been poised to deliver fast, real-time diagnosis, facilitate telemedicine consultations, and equip patients with the power of self-examination behavior. Moreover, DL model ability to learn continuously enabled them to grow and adapt over time and have sustained accuracy and reliability. However, deployment of DL in healthcare is needed with due attention to ensuring the ethical issues of data confidentiality, algorithmic transparency, and minimizing bias. Resolution of these issues is important to build trust in DL-driven diagnostic devices and provide high-quality care equally. Interdisciplinary collaboration between clinicians, DL experts, and researchers can be the key to the advancement of melanoma diagnosis and employing technology to improve patient results in the most effective way.

8. References

[1] Vijayalakshmi, M. M. (2019). Melanoma skin cancer detection using image processing and deep learning. International Journal of Trend in Scientific Research and Development (IJTSRD), 3(4), 780-784.

[2] Vocaturo, E., Perna, D., & Zumpano, E. (2019, November). Deep learning techniques for automated melanoma detection. In 2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) (pp. 2310-2317). IEEE.

[3] Refianti, R., Mutiara, A. B., & Priyandini, R. P. (2019). Classification of melanoma skin cancer using convolutional neural network. International Journal of Advanced Computer Science and Applications, 10(3).

[4] Vidya, M., & Karki, M. V. (2020, July). Skin cancer detection using deep learning techniques. In 2020 IEEE international conference on electronics, computing and communication technologies (CONECCT) (pp. 1-5). IEEE.

[5] Thaajwer, M. A., & Ishanka, U. P. (2020, December). Melanoma skin cancer detection using image processing and deep learning techniques. In 2020 2nd International Conference on Advancements in Computing (ICAC) (Vol. 1, pp. 363-368). IEEE.

[6] Das, K., Cockerell, C. J., Patil, A., Pietkiewicz, P., Giulini, M., Grabbe, S., & Goldust, M. (2021). Deep learning and its application in skin cancer. International Journal of Environmental Research and Public Health, 18(24), 13409.

[7] Murugan, A., Nair, S. A. H., Preethi, A. A. P., & Kumar, K. S. (2021). Diagnosis of skin cancer using deep learning techniques. Microprocessors and Microsystems, 81, 103727.

[8] Rahman, M. M., Nasir, M. K., Nur, A., Khan, S. I., Band, S., Dehzangi, I., ... & Rokny, H. A. (2022). Hybrid feature fusion and deep learning approaches for melanoma skin cancer detection.

[9] Tembhurne, J. V., Hebbar, N., Patil, H. Y., & Diwan, T. (2023). Skin cancer detection using ensemble of deep learning and deep learning techniques. Multimedia Tools and Applications, 1-24.

[10] Patil, R., & Bellary, S. (2022). Deep learning approach in melanoma cancer stage detection. Journal of King Saud University-Computer and Information Sciences, 34(6), 3285-3293.