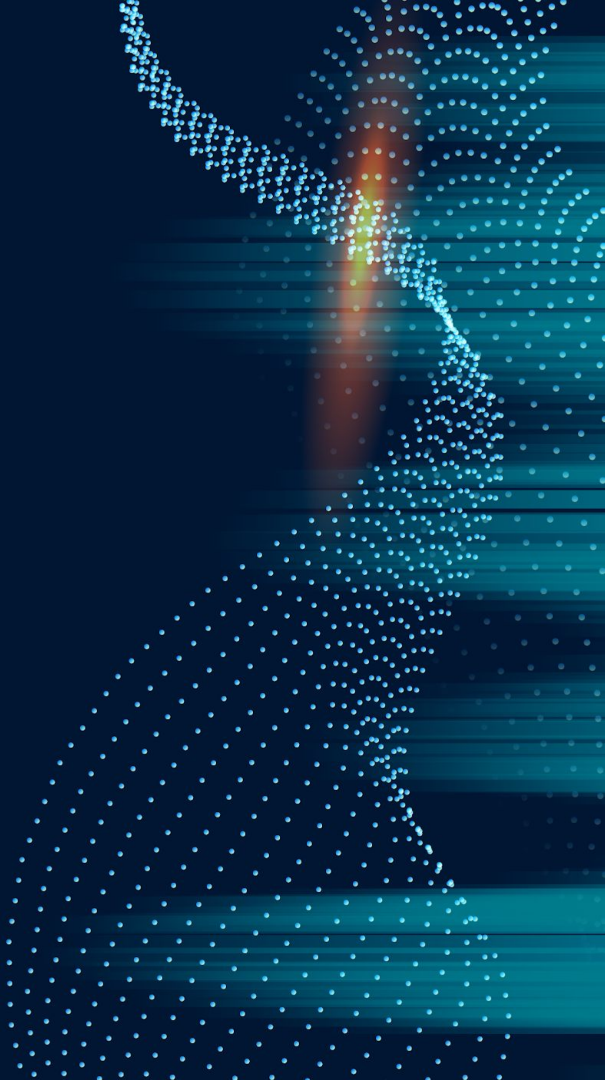
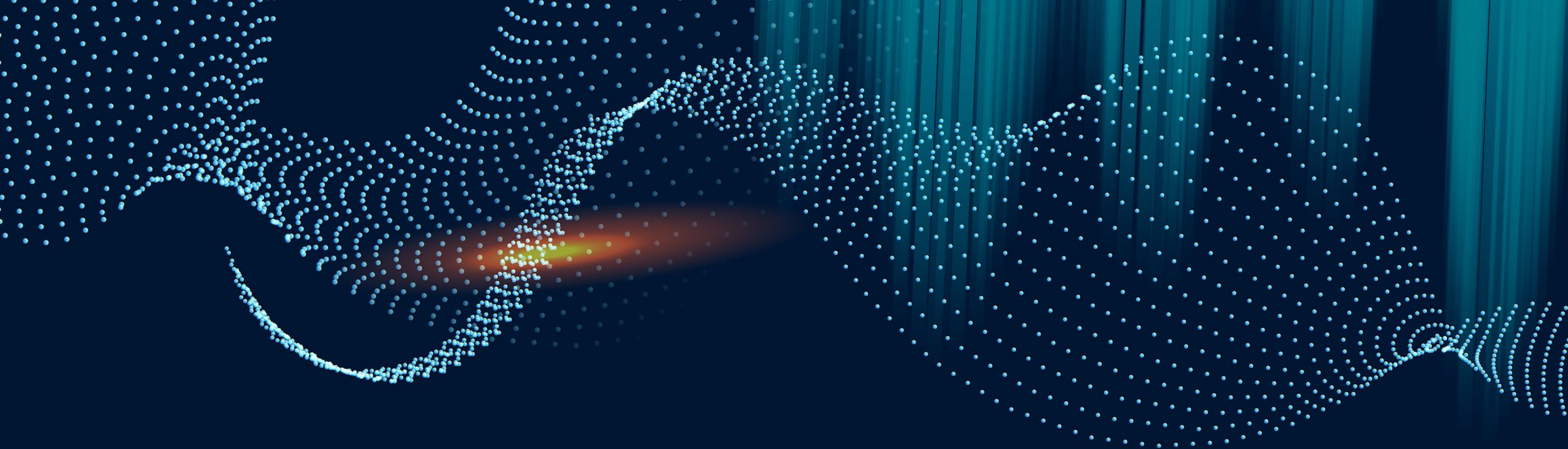
Skin Cancer Detection Using Diversified CNN Architectures

Niyat Habtom Seghid: 19967
Belsabel Teklemariam Woldemichael: 20052
Feven Belay Araya: 20027

PROFESSOR: GRIGORIEV, ANDREI

Contents





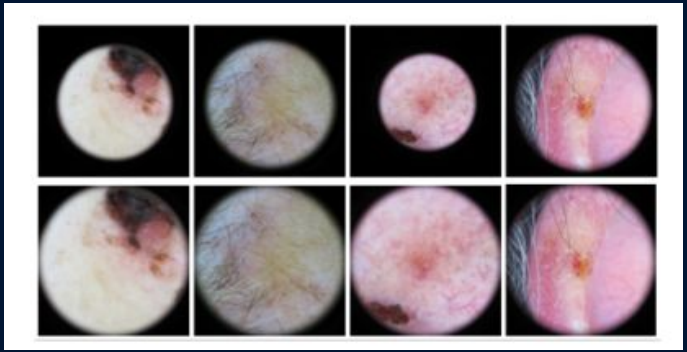
01 | Introduction

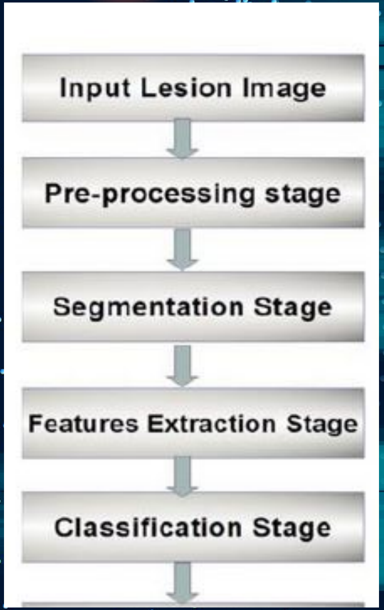
Why we chose this project?

- Skin cancer is one of the most common cancers globally, with millions of new cases annually.
- So effective early detection is critical to improving survival rates.

Objective:

- Early detection of skin cancer using deep learning to improve survival rates.
- We will apply advanced deep learning techniques to enhance early detection accuracy.





Models Used for Skin Cancer Classification

1. EfficientNet-B5

2. XceptionNet

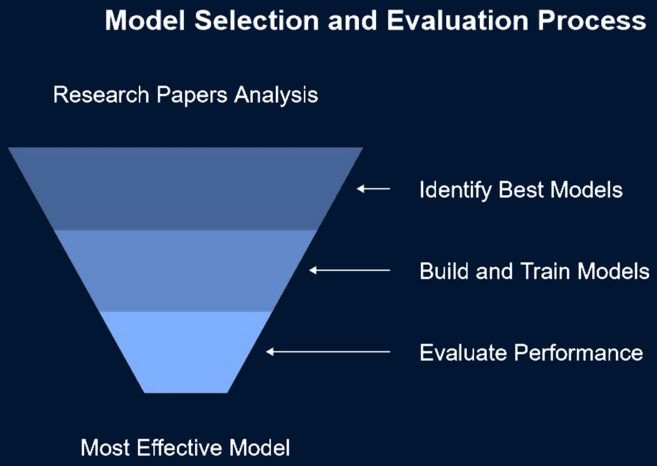
3. Dual-Path CNN

Objective

To classify skin lesions into two main categories based on their potential severity:

1. **Melanoma (Malignant)**: A serious form of skin cancer that can spread to other parts of the body, requiring early detection and treatment.
2. **Non-Melanoma (Benign)**: Less aggressive skin lesions that are generally not life-threatening and less likely to spread.

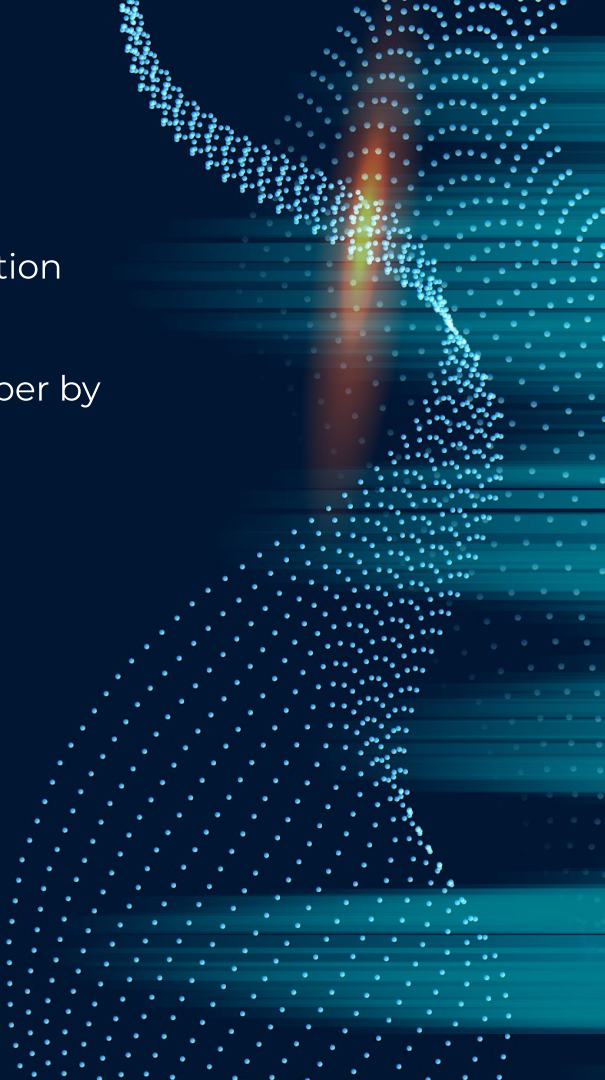
- Deliver insights on methodologies and model efficacies .
- Comparison and evaluation of three models based on performance metrics and chose the best model. (Given we find the right commuting resources)



02 | Approach

Our Approach and Expected Results:

1. We will review at least three scholarly papers on skin cancer detection using deep learning.
2. Analyze and implement the best-performing model from each paper by following their methodologies.
3. Compare the performance of the models to recommend the most effective approach.



1. An Interpretable Deep Learning Approach for Skin Cancer Categorization

XceptionNet

Data set

Total Images: **10,015**

- The dataset contains **RGB dermatoscopic images** of skin lesions.
- Images are curated from two sources:
  - **HAM10000 ("HAM") collection**: A larger dataset.
  - **Vienna Dermatology Dataset**: Another public resource.

Computes 10% of the dataset to reduce computation (subset size).

- **Model:** XceptionNet
- Tools: TensorFlow, Keras, OpenCV.
- XceptionNet stands for "Extreme Inception" and was developed by Google.
- XceptionNet is a type of **Convolutional Neural Network (CNN)** designed for image classification. It's an improvement over traditional CNNs because it uses **depth-wise separable convolutions** to reduce computational costs while preserving accuracy.

Data Preprocessing

Resizing:

- All images were resized to 224x224 pixels to match the input requirements of the XceptionNet model.

Normalization:

- Pixel values were scaled to the range [0, 1] by dividing by 255.0. This helps in speeding up convergence during training.

Data Augmentation:

- Augmentation techniques were applied to artificially increase the size and diversity of the dataset:
    - Random rotations (up to 20 degrees).
    - Horizontal flipping.
    - Zooming and cropping.
    - Shifting the image horizontally and vertically.

Model Architecture

The base architecture used was **XceptionNet**, a convolutional neural network designed for efficient image classification.

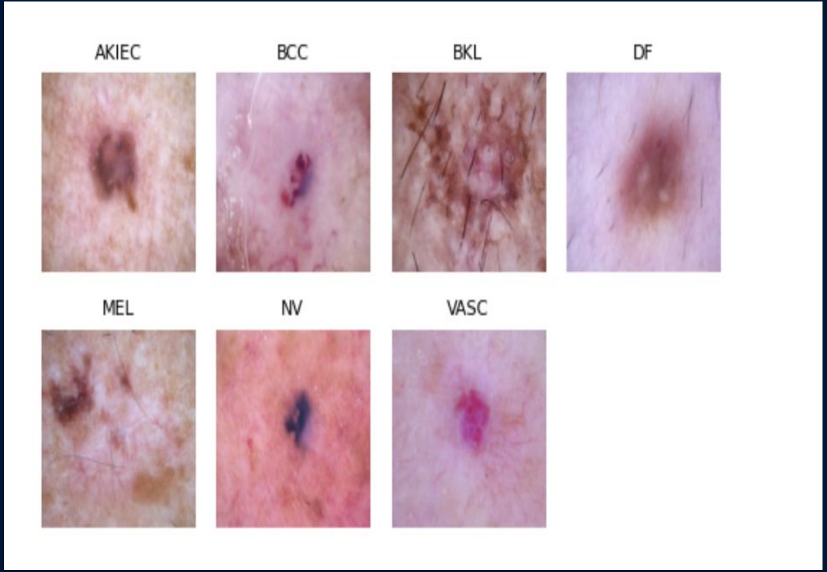
Key Features of XceptionNet:

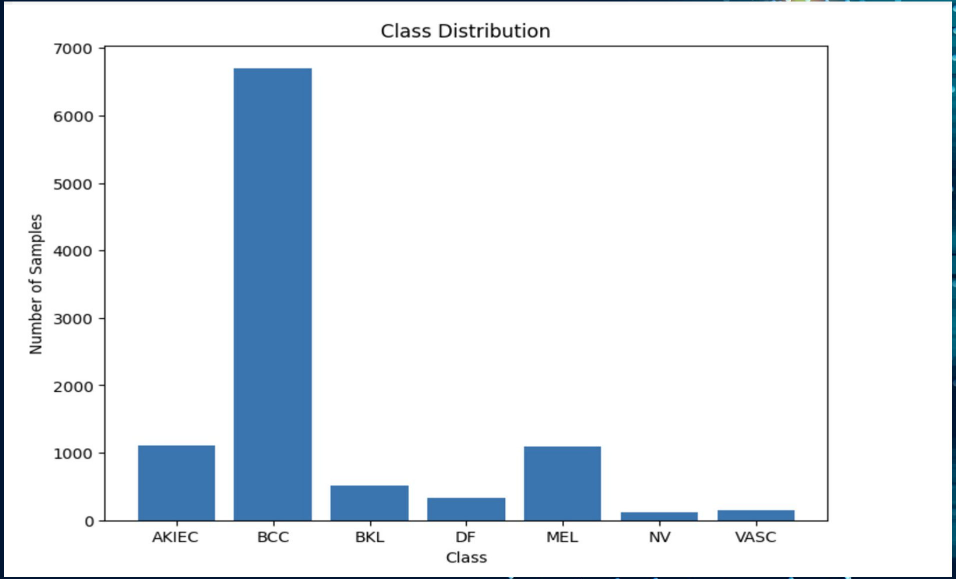
- Uses depth-wise separable convolutions, which split the standard convolution operation into two steps:
    1. Depth-wise convolution (per-channel spatial feature extraction).
    2. Point-wise convolution (cross-channel feature aggregation).
- Reduces computational overhead while maintaining high representational power.

Custom Layers:

- A fully connected layer with 128 neurons (ReLU activation).
- Dropout (50%) to prevent overfitting.
- A final dense layer with 7 neurons (softmax activation) for multiclass classification.

Distribution of classes





Training Procedure

Transfer Learning:

- Pretrained on ImageNet.
- Initial layers were frozen to leverage pretrained features.

Fine-Tuning:

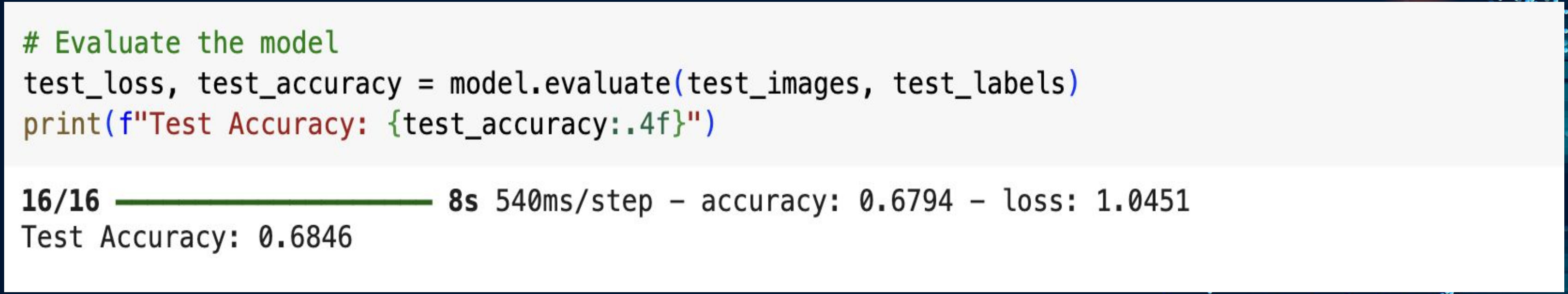
- The last 10 layers of the base XceptionNet were unfrozen for additional training to adapt to the skin lesion dataset.

Parameters:

- Optimizer: Adam
- Loss Function: Categorical Cross Entropy
- Batch Size: 16
- Epochs: 30
- Validation Set: 20% of training data.

Evaluation

- **Test Accuracy**: Achieved 68.4 % test accuracy.



```
# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
print(f"Test Accuracy: {test_accuracy:.4f}")
```

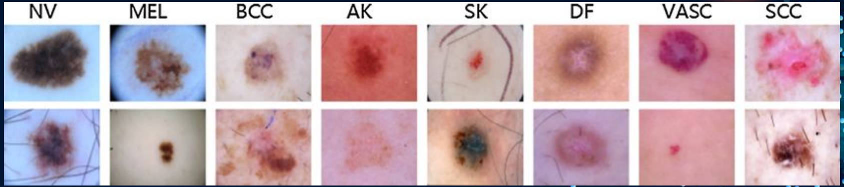
```
16/16 ━━━━━━━━━━━━━━━━━━━━ 8s 540ms/step - accuracy: 0.6794 - loss: 1.0451
Test Accuracy: 0.6846
```

2. Analysis of Skin Lesion Images with Deep Learning

EfficientNet-B5

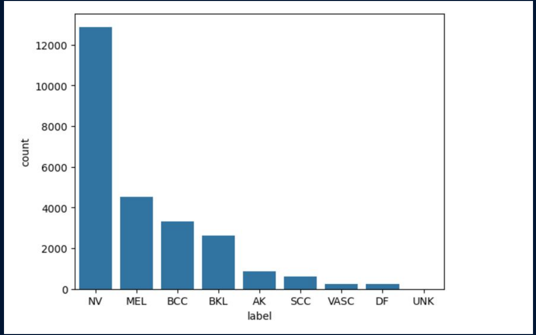
Dataset

- **Datasets:**
  - Primary Dataset: ISIC-2019 Challenge dataset (25,331 images)- 9 diagnostic categories
  - Additional: BCN 20000, HAM10000, MSK, and others totaling 32,748 images.
- **Models (CNN Architectures):**
  - EfficientNet-B5.



Classification Class categories:

1. Melanoma
2. Melanocytic nevus
3. Basal cell carcinoma
4. Actinic keratosis
5. Benign keratosis (solar lentigo / seborrheic keratosis / lichen planus-like keratosis)
6. Dermatofibroma
7. Vascular lesion
8. Squamous cell carcinoma
9. None of the others



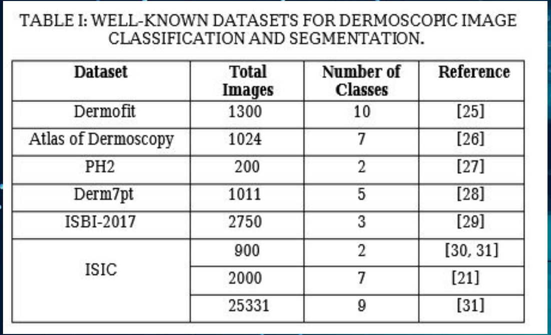


TABLE I: WELL-KNOWN DATASETS FOR DERMOSCOPIC IMAGE CLASSIFICATION AND SEGMENTATION.

Dataset	Total Images	Number of Classes	Reference
Dermofit	1300	10	[25]
Atlas of Dermoscopy	1024	7	[26]
PH2	200	2	[27]
Derm7pt	1011	5	[28]
ISBI-2017	2750	3	[29]
ISIC	900	2	[30, 31]
	2000	7	[21]
	25331	9	[31]

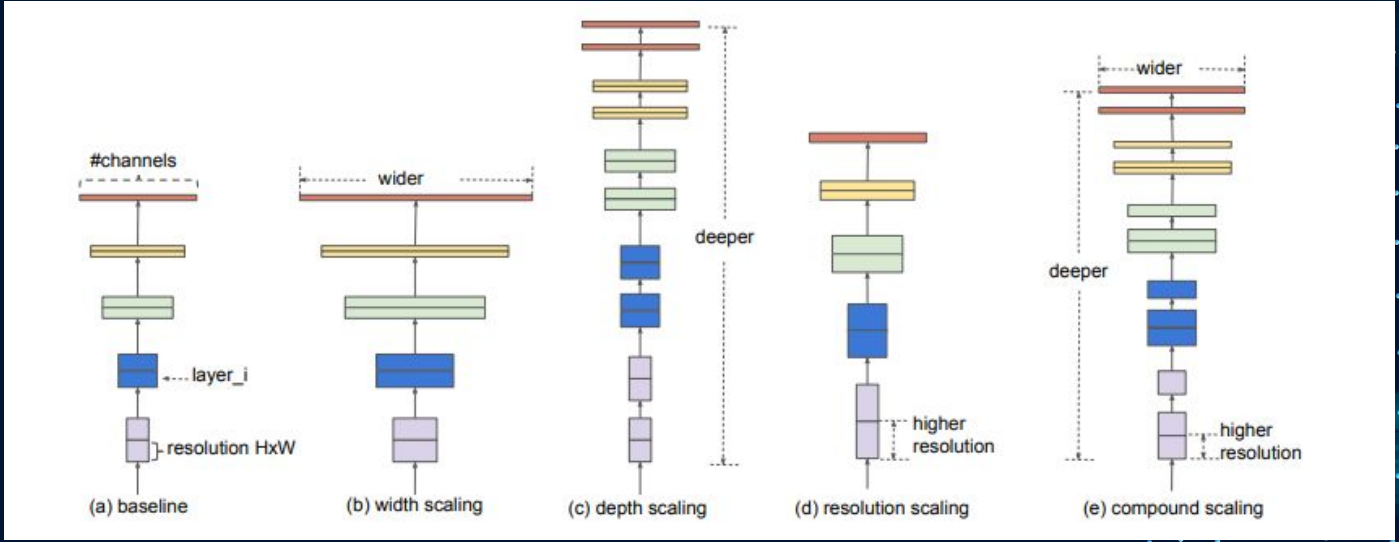
EfficientNet-B5

Model Scaling.

(a) is a baseline network example

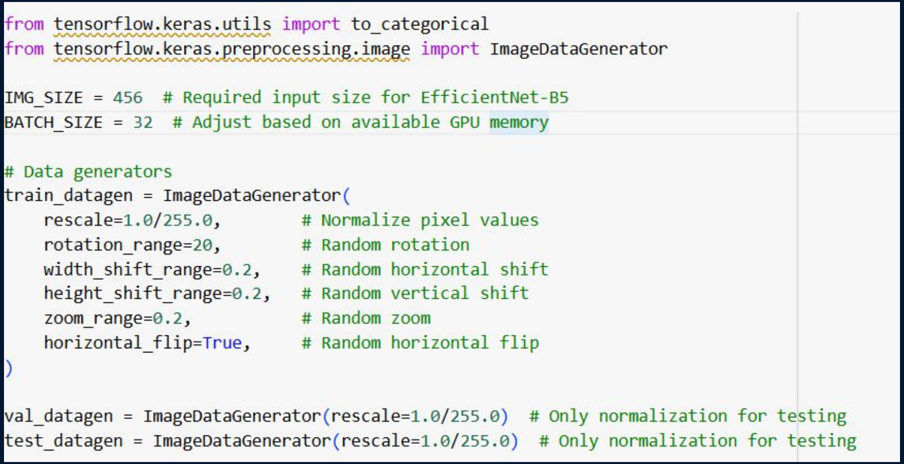
(b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution.

(e) is the architecture we used which is compound scaling method that uniformly scales all three dimensions with a fixed ratio

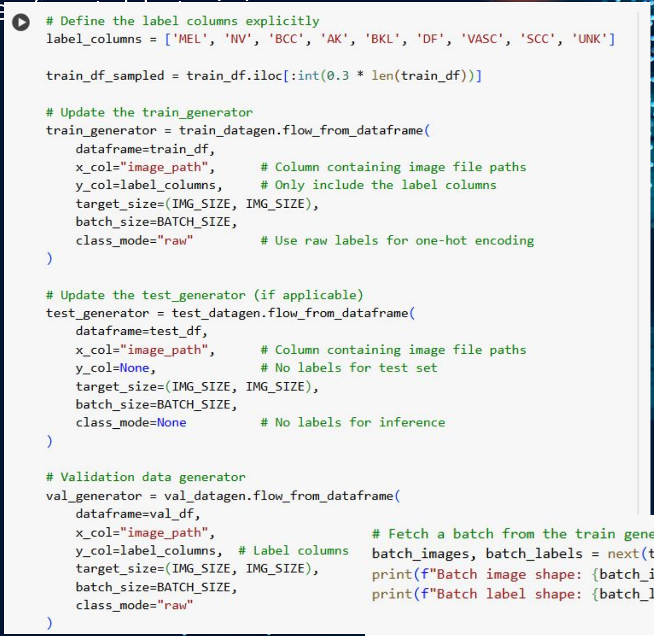


Preprocessing (Normalization and Image Augmentations)

- **Image size required for EfficientNet-B5 Architecture => 456**
- **Augmentation:** Prevents overfitting by simulating real-world image variations. (artificially increases the dataset's diversity)
- **Normalization (rescale=1./255):** Ensures consistent pixel value range

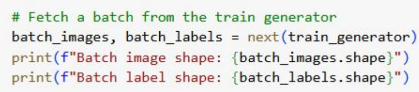


```python
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator

IMG_SIZE = 456  # Required input size for EfficientNet-B5
BATCH_SIZE = 32  # Adjust based on available GPU memory

# Data generators
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,          # Normalize pixel values
    rotation_range=20,          # Random rotation
    width_shift_range=0.2,      # Random horizontal shift
    height_shift_range=0.2,     # Random vertical shift
    zoom_range=0.2,             # Random zoom
    horizontal_flip=True,       # Random horizontal flip
)

val_datagen = ImageDataGenerator(rescale=1.0/255.0)  # Only normalization for testing
test_datagen = ImageDataGenerator(rescale=1.0/255.0)  # Only normalization for testing
```

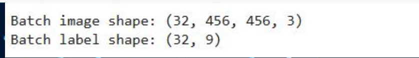


```python
# Define the label columns explicitly
label_columns = ['MEL', 'NV', 'BCC', 'AK', 'BKL', 'DF', 'VASC', 'SCC', 'UNK']

train_df_sampled = train_df.iloc[:int(0.3 * len(train_df))]

# Update the train_generator
train_generator = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    x_col="image_path",       # Column containing image file paths
    y_col=label_columns,      # Only include the label columns
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode="raw"          # Use raw labels for one-hot encoding
)

# Update the test_generator (if applicable)
test_generator = test_datagen.flow_from_dataframe(
    dataframe=test_df,
    x_col="image_path",       # Column containing image file paths
    y_col=None,               # No labels for test set
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode=None           # No labels for inference
)

# Validation data generator
val_generator = val_datagen.flow_from_dataframe(
    dataframe=val_df,
    x_col="image_path",
    y_col=label_columns,  # Label columns
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode="raw"
)
```

Data Generators

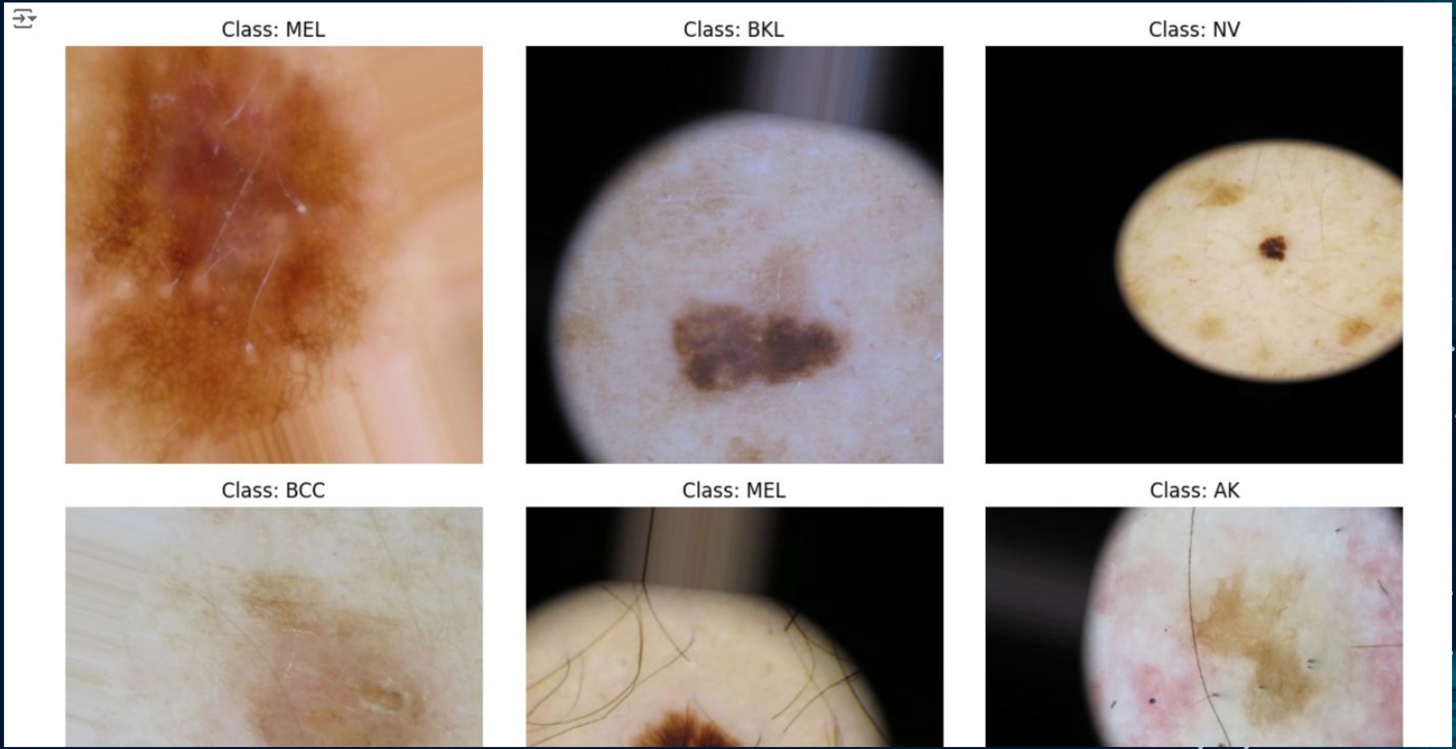


```python
# Fetch a batch from the train generator
batch_images, batch_labels = next(train_generator)
print(f"Batch image shape: {batch_images.shape}")
print(f"Batch label shape: {batch_labels.shape}")
```



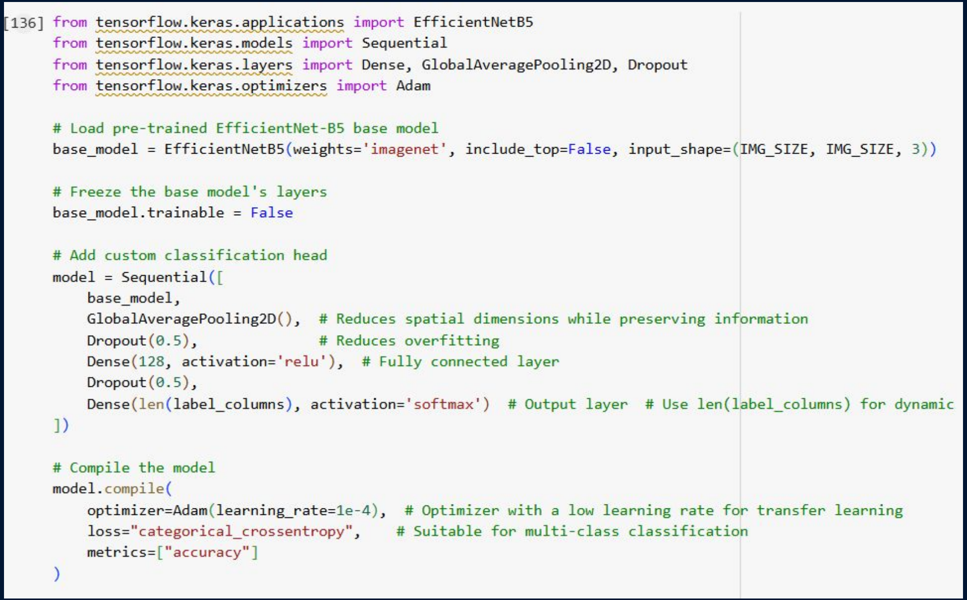
```
Batch image shape: (32, 456, 456, 3)
Batch label shape: (32, 9)
```

Sample Data after Preprocessing



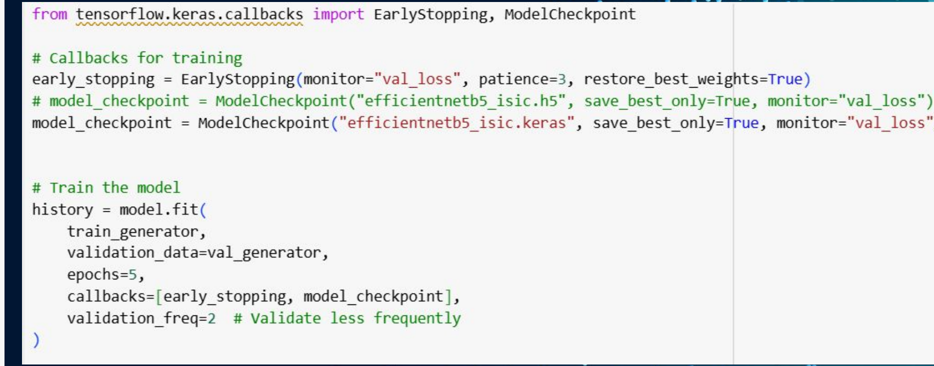
Building and Training the Model Using EfficientNet-B5

- **This is a Pre-trained CNN on ImageNet:** it transfers features learned from millions of images.
- **include_top=False:** Removes the final classification layer to add custom layers.
- Freezing layers prevents overwriting pre-trained weights. Or to reduce the number of trainable parameters



```
[136] from tensorflow.keras.applications import EfficientNetB5
      from tensorflow.keras.models import Sequential
      from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
      from tensorflow.keras.optimizers import Adam

      # Load pre-trained EfficientNet-B5 base model
      base_model = EfficientNetB5(weights='imagenet', include_top=False, input_shape=(IMG_SIZE, IMG_SIZE, 3))

      # Freeze the base model's layers
      base_model.trainable = False

      # Add custom classification head
      model = Sequential([
          base_model,
          GlobalAveragePooling2D(),  # Reduces spatial dimensions while preserving information
          Dropout(0.5),              # Reduces overfitting
          Dense(128, activation='relu'),  # Fully connected layer
          Dropout(0.5),
          Dense(len(label_columns), activation='softmax')  # Output layer  # Use len(label_columns) for dynamic
      ])

      # Compile the model
      model.compile(
          optimizer=Adam(learning_rate=1e-4),  # Optimizer with a low learning rate for transfer learning
          loss="categorical_crossentropy",    # Suitable for multi-class classification
          metrics=["accuracy"]
      )
```

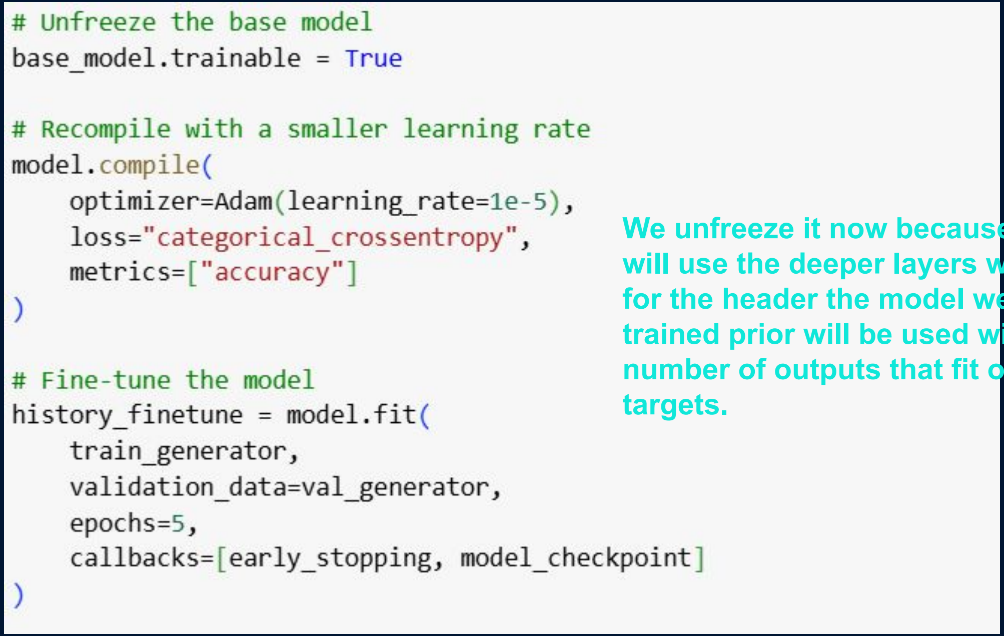
Model training with early stopping



```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

# Callbacks for training
early_stopping = EarlyStopping(monitor="val_loss", patience=3, restore_best_weights=True)
# model_checkpoint = ModelCheckpoint("efficientnetb5_isic.h5", save_best_only=True, monitor="val_loss")
model_checkpoint = ModelCheckpoint("efficientnetb5_isic.keras", save_best_only=True, monitor="val_loss"


# Train the model
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=5,
    callbacks=[early_stopping, model_checkpoint],
    validation_freq=2  # Validate less frequently
)
```

Finetuning the Base Model we are using

- **Unfreeze the base model to fine-tune deeper layers**



```
# Unfreeze the base model
base_model.trainable = True

# Recompile with a smaller learning rate
model.compile(
    optimizer=Adam(learning_rate=1e-5),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

# Fine-tune the model
history_finetune = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=5,
    callbacks=[early_stopping, model_checkpoint]
)
```

We unfreeze it now because we will use the deeper layers whereas for the header the model we trained prior will be used with number of outputs that fit our targets.

Results:

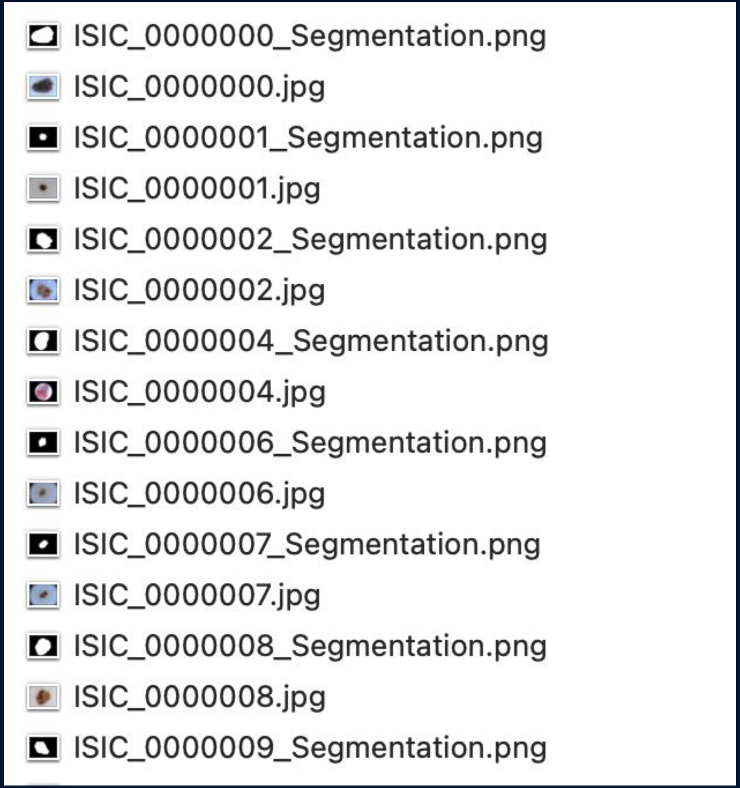
Test Accuracy:0.54

3. Methods to Classify Skin Lesions using Dermoscopic images

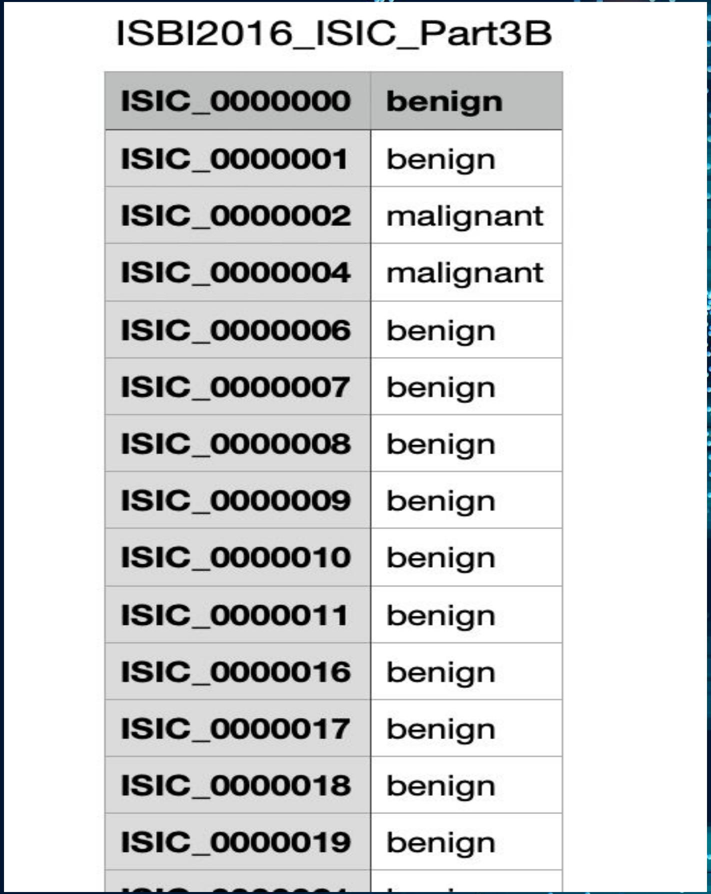
Dual- Path CNN Model

Dataset

- **Dataset:** ISBI2016 ISIC Part 3B (Skin Lesion Classification)
- **Content:**
  - The dataset consists of **skin images** and corresponding **segmentation masks**.
  - **Labels:**
    - **0:** Benign lesions (non-cancerous).
    - **1:** Malignant lesions (cancerous,).
- **Data Overview:**
  - Images: High-resolution skin images of lesions.
  - Masks: Segmentation masks to delineate the lesion from the rest of the skin.
  - Image Size: 512x512 pixels.
  - Data size: 10,000 Images

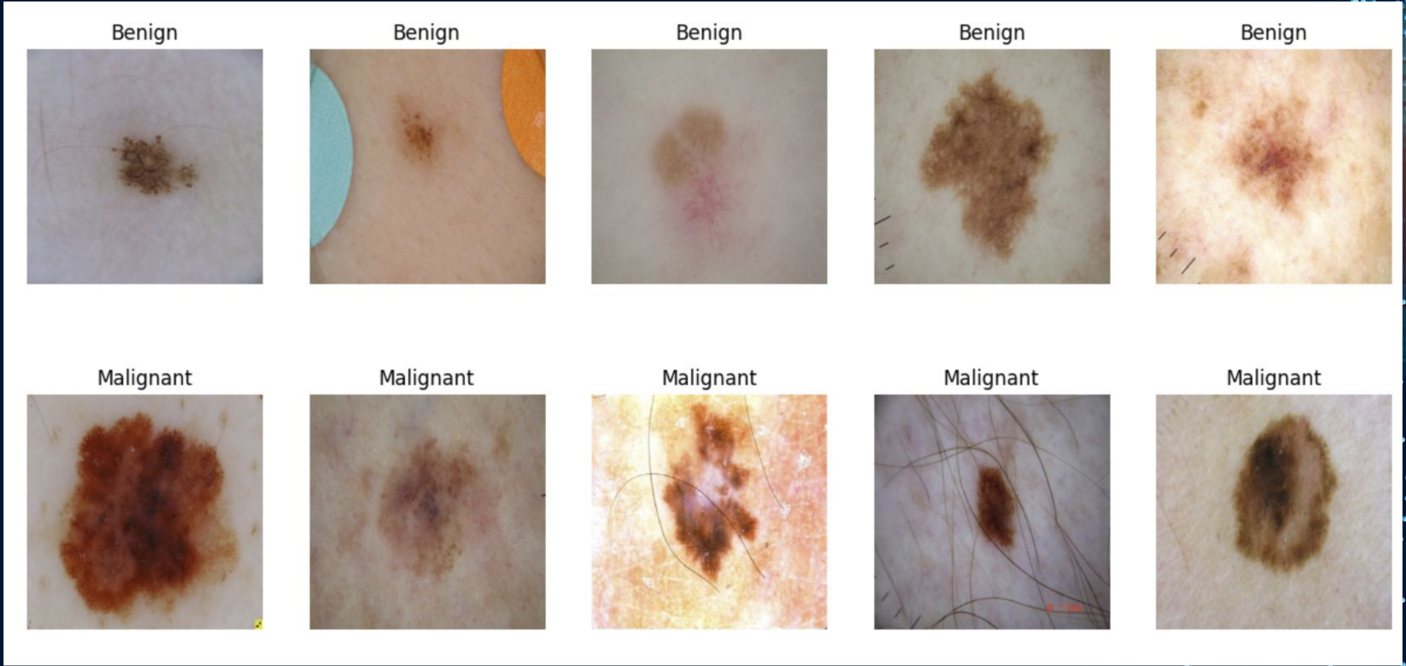


ISIC_0000000_Segmentation.png
ISIC_0000000.jpg
ISIC_0000001_Segmentation.png
ISIC_0000001.jpg
ISIC_0000002_Segmentation.png
ISIC_0000002.jpg
ISIC_0000004_Segmentation.png
ISIC_0000004.jpg
ISIC_0000006_Segmentation.png
ISIC_0000006.jpg
ISIC_0000007_Segmentation.png
ISIC_0000007.jpg
ISIC_0000008_Segmentation.png
ISIC_0000008.jpg
ISIC_0000009_Segmentation.png



ISBI2016_ISIC_Part3B

ISIC_0000000	benign
ISIC_0000001	benign
ISIC_0000002	malignant
ISIC_0000004	malignant
ISIC_0000006	benign
ISIC_0000007	benign
ISIC_0000008	benign
ISIC_0000009	benign
ISIC_0000010	benign
ISIC_0000011	benign
ISIC_0000016	benign
ISIC_0000017	benign
ISIC_0000018	benign
ISIC_0000019	benign


Benign
Benign
Benign
Benign
Benign
Malignant
Malignant
Malignant
Malignant
Malignant

Data Preprocessing

1. Image Loading and Resizing:

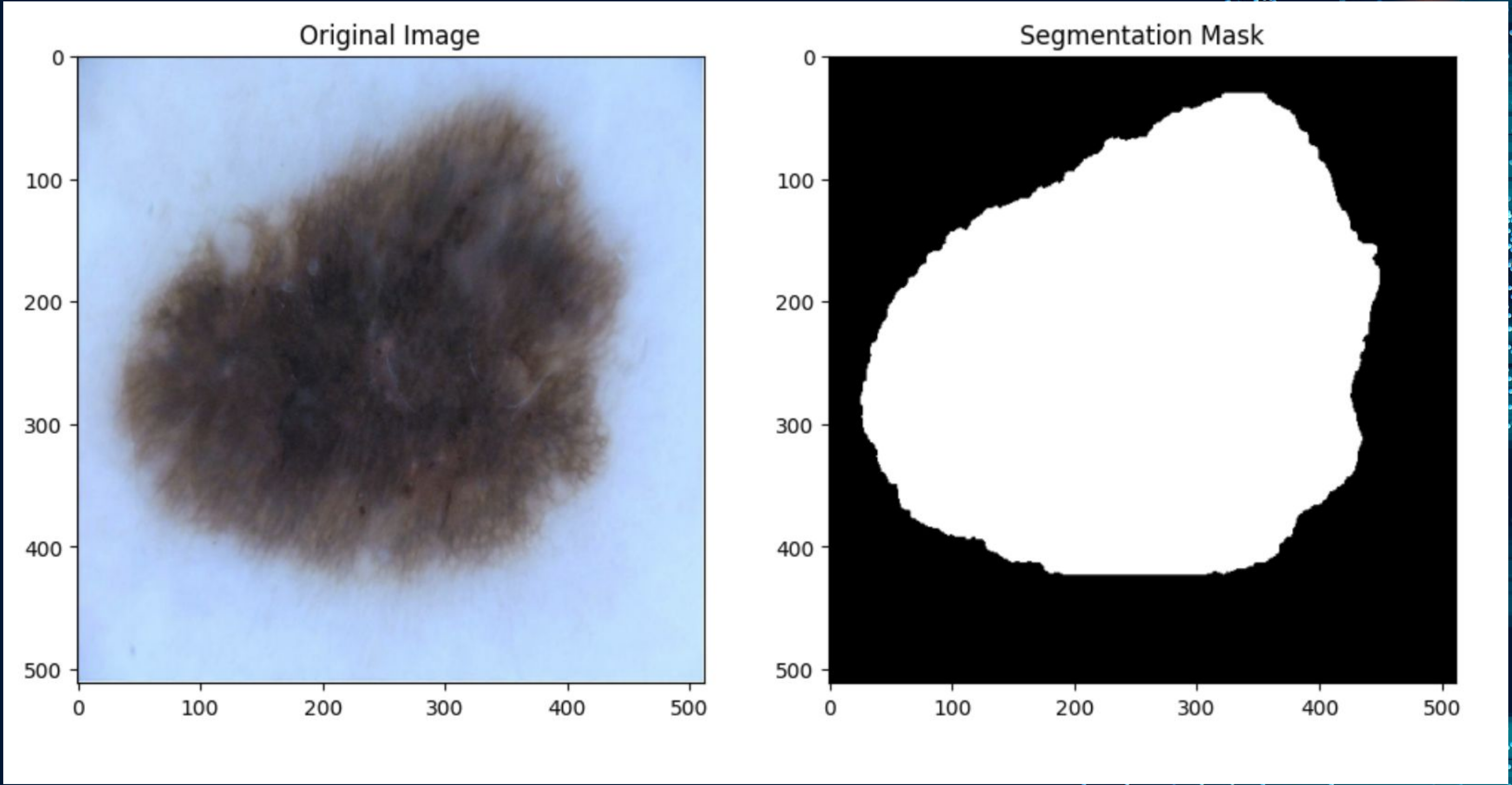
- Images are loaded and resized to **512x512** for consistent input size.

2. Normalization:

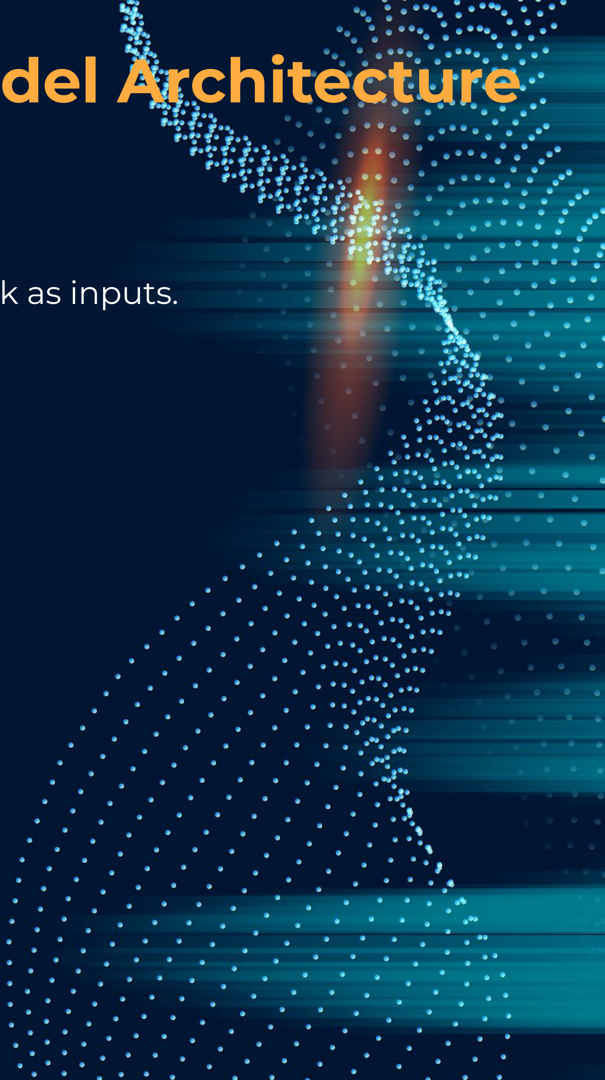
- Pixel values are normalized by dividing by **255** to scale the pixel values between **0 and 1**.
- This helps in faster convergence and better training.

3. Segmentation Masks:

- Masks are also resized and normalized to match the image size and pixel intensity.


Original Image
Segmentation Mask
0
100
200
300
400
500

Convolutional Neural Network (CNN) Model Architecture



Model Overview:

- **Dual Input Model**: Uses both the original image and segmentation mask as inputs.

Convolutional Layers:

- **Conv2D layers** for feature extraction (e.g., detecting edges, textures).

Pooling Layers:

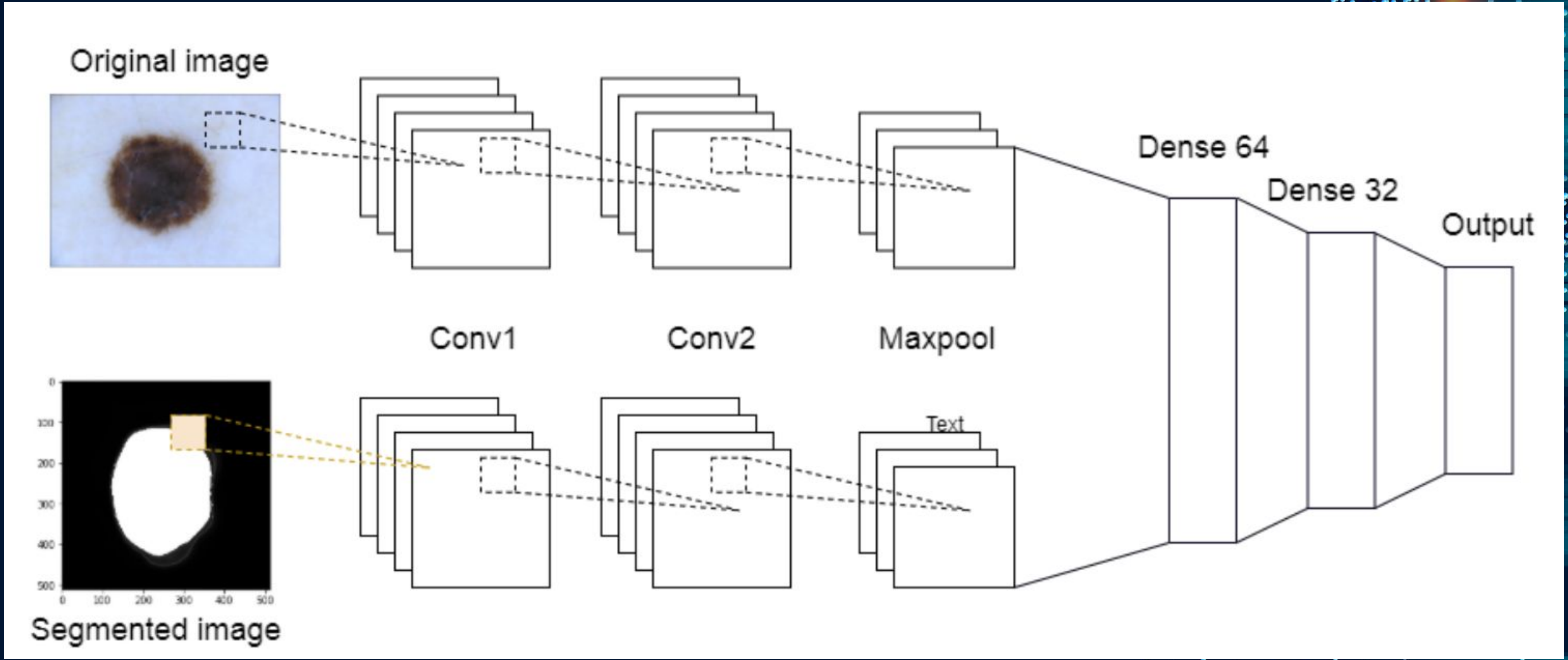
- **MaxPooling** to reduce spatial dimensions and retain key features.

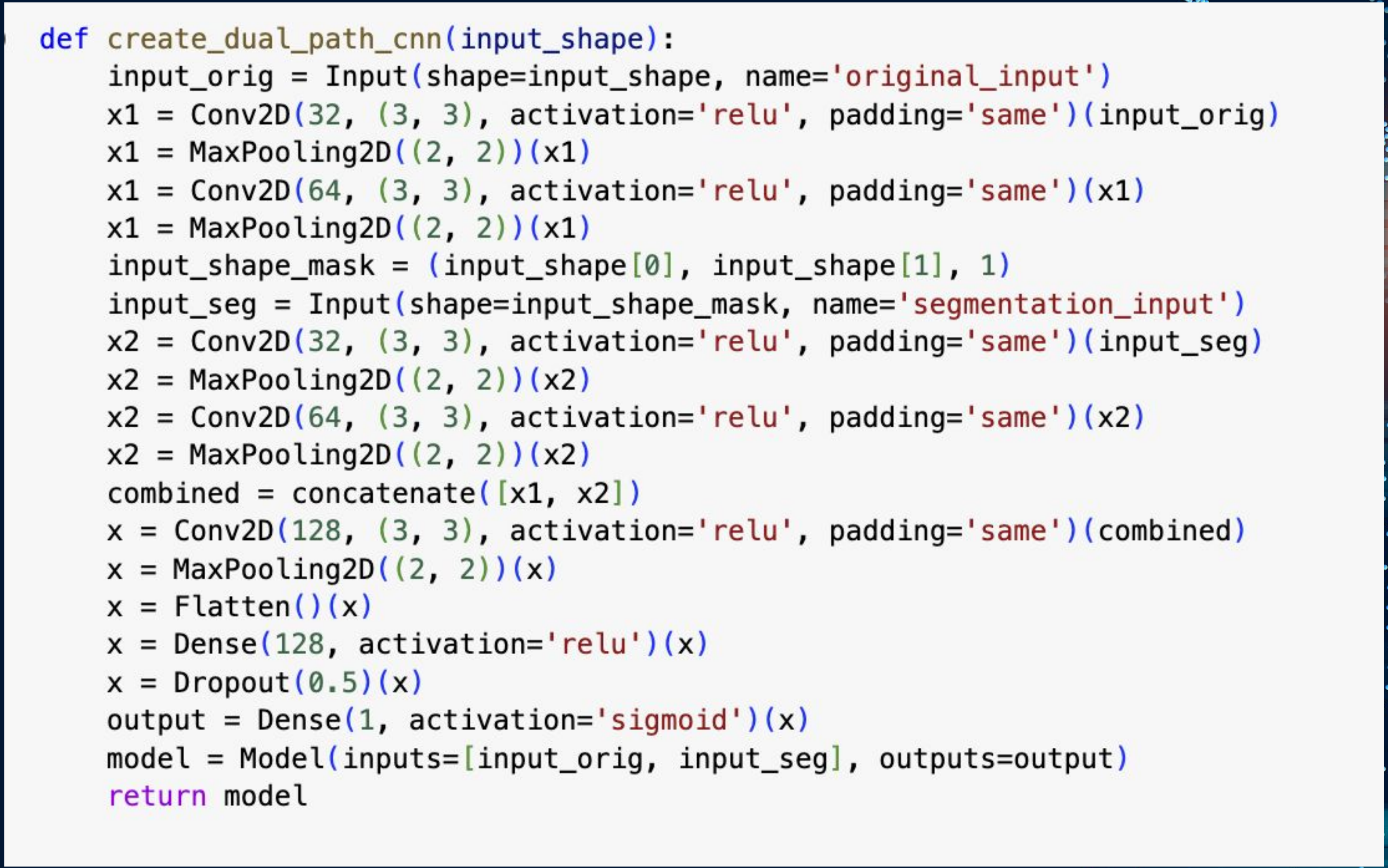
Fully Connected Layers:

- Dense layers for decision-making based on extracted features.

Output Layer:

- **Sigmoid activation** for binary classification (Benign vs Malignant).


Original image
Conv1
Conv2
Maxpool
Dense 64
Dense 32
Output
Text
Segmented image
0
100
200
300
400
500
0
100
200
300
400
500



```python
def create_dual_path_cnn(input_shape):
    input_orig = Input(shape=input_shape, name='original_input')
    x1 = Conv2D(32, (3, 3), activation='relu', padding='same')(input_orig)
    x1 = MaxPooling2D((2, 2))(x1)
    x1 = Conv2D(64, (3, 3), activation='relu', padding='same')(x1)
    x1 = MaxPooling2D((2, 2))(x1)
    input_shape_mask = (input_shape[0], input_shape[1], 1)
    input_seg = Input(shape=input_shape_mask, name='segmentation_input')
    x2 = Conv2D(32, (3, 3), activation='relu', padding='same')(input_seg)
    x2 = MaxPooling2D((2, 2))(x2)
    x2 = Conv2D(64, (3, 3), activation='relu', padding='same')(x2)
    x2 = MaxPooling2D((2, 2))(x2)
    combined = concatenate([x1, x2])
    x = Conv2D(128, (3, 3), activation='relu', padding='same')(combined)
    x = MaxPooling2D((2, 2))(x)
    x = Flatten()(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.5)(x)
    output = Dense(1, activation='sigmoid')(x)
    model = Model(inputs=[input_orig, input_seg], outputs=output)
    return model
```

Model Training and Cross-Validation

K-Fold Cross-Validation:

- **3-fold cross-validation** to evaluate model robustness and avoid overfitting.

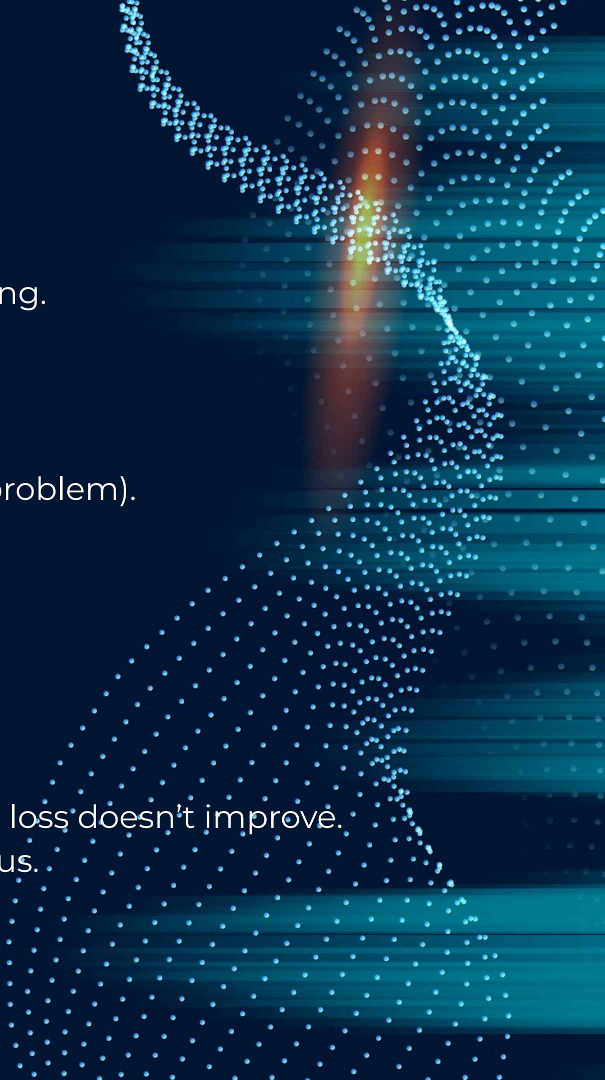
Training Details:

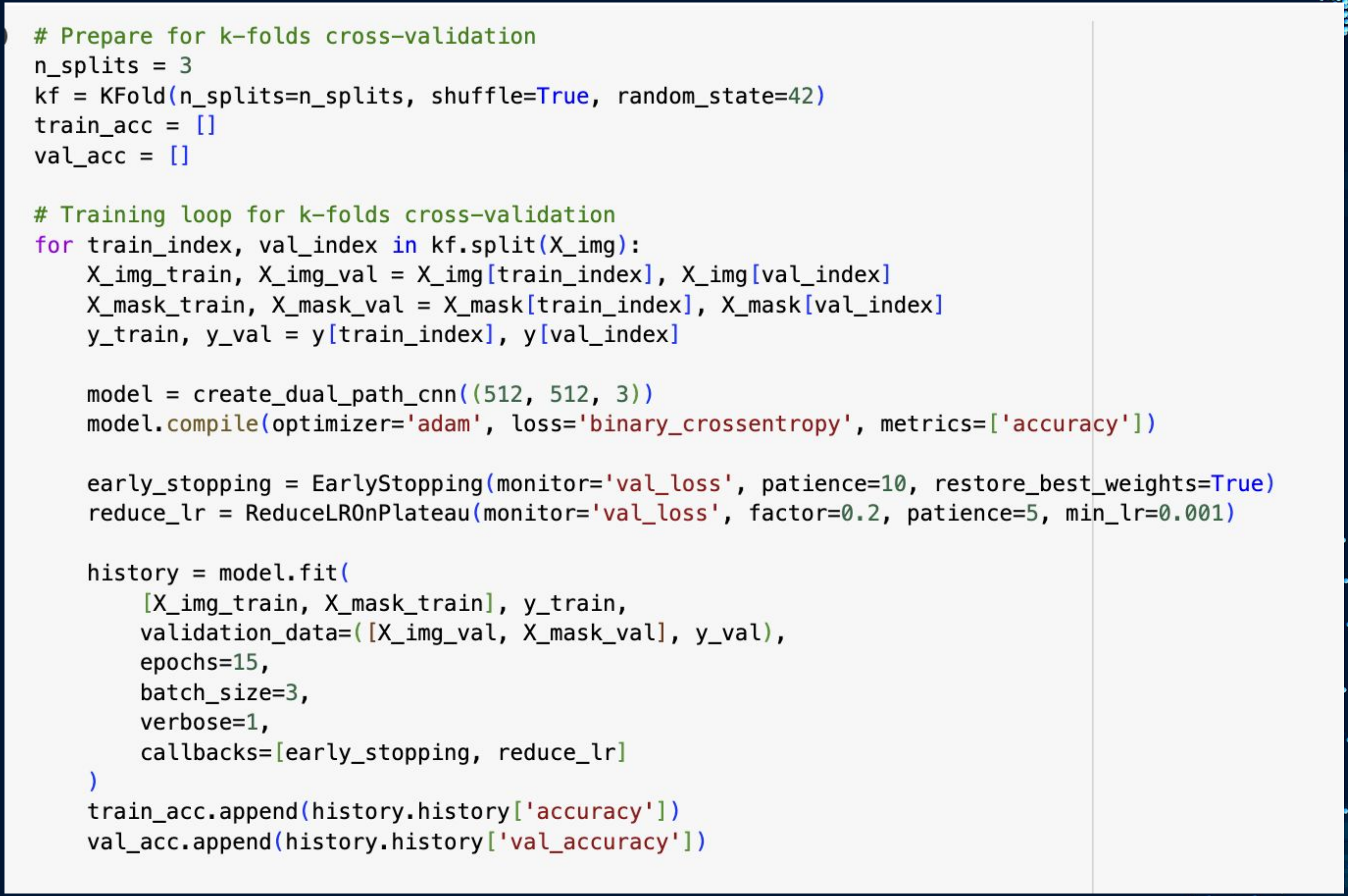
- **Optimizer:** Adam optimizer.
- **Loss Function:** Binary Crossentropy (since this is a binary classification problem).
- **Metrics:** Accuracy.

Epochs: 15
Batch Size: 3

Callbacks:

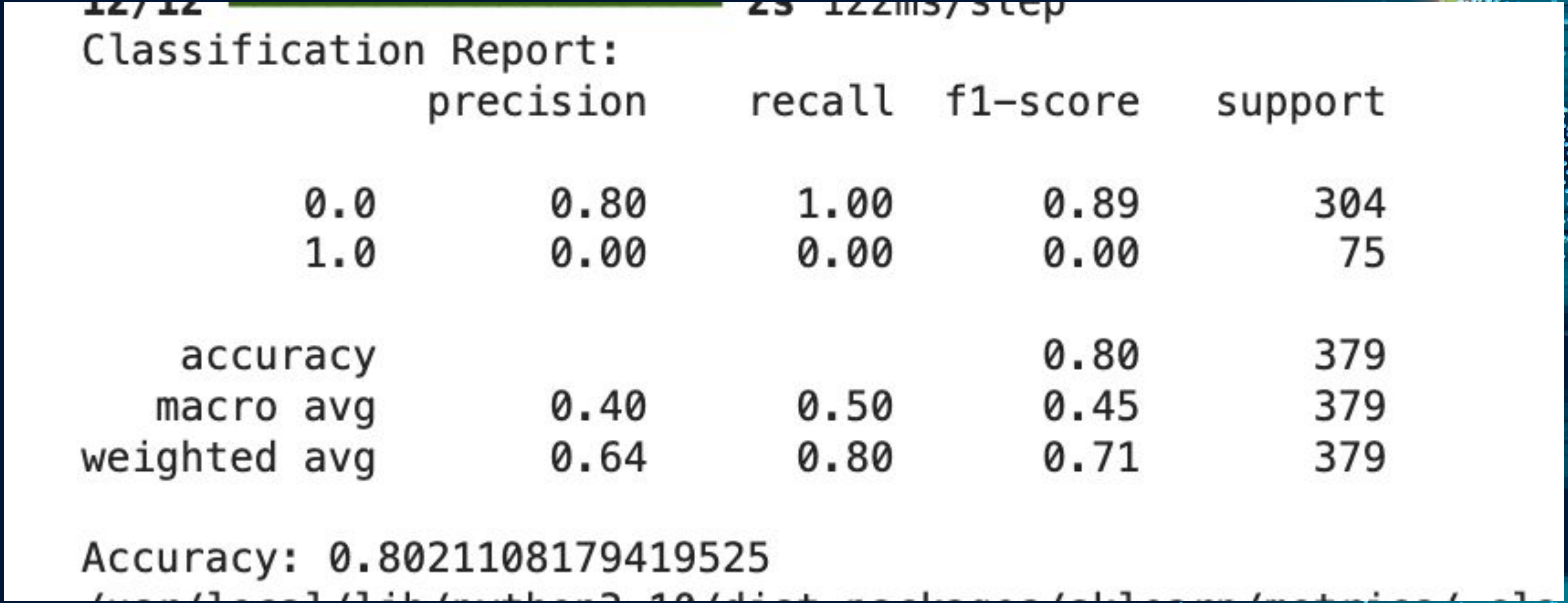
- **EarlyStopping** to prevent overfitting by halting training when validation loss doesn't improve.
- **ReduceLROnPlateau** to adjust learning rate when validation loss plateaus.





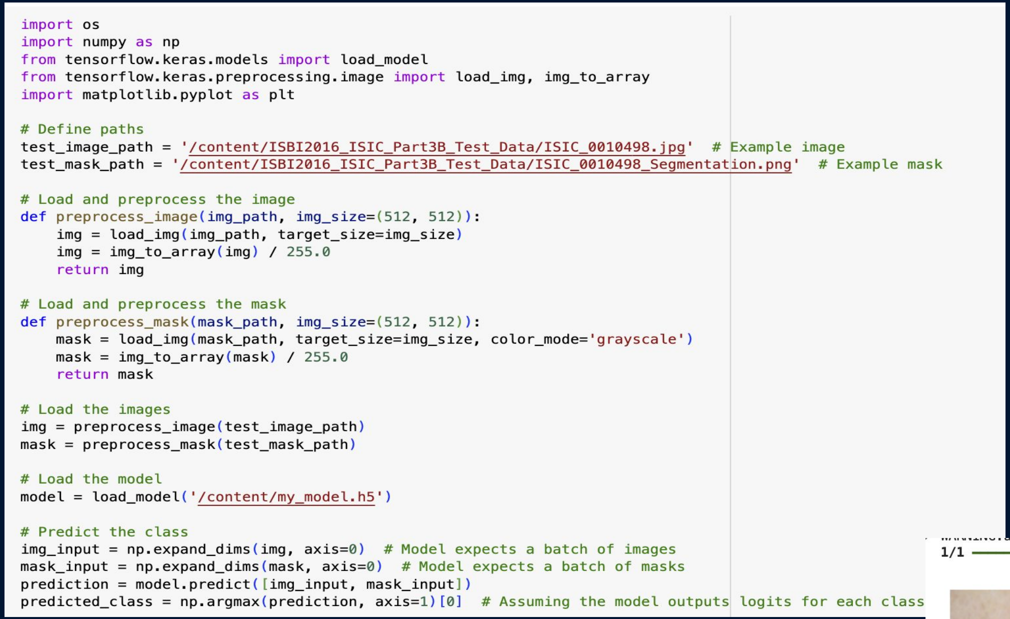
```
# Prepare for k-folds cross-validation
n_splits = 3
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
train_acc = []
val_acc = []

# Training loop for k-folds cross-validation
for train_index, val_index in kf.split(X_img):
    X_img_train, X_img_val = X_img[train_index], X_img[val_index]
    X_mask_train, X_mask_val = X_mask[train_index], X_mask[val_index]
    y_train, y_val = y[train_index], y[val_index]

    model = create_dual_path_cnn((512, 512, 3))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.001)

    history = model.fit(
        [X_img_train, X_mask_train], y_train,
        validation_data=([X_img_val, X_mask_val], y_val),
        epochs=15,
        batch_size=3,
        verbose=1,
        callbacks=[early_stopping, reduce_lr]
    )
    train_acc.append(history.history['accuracy'])
    val_acc.append(history.history['val_accuracy'])
```

Performance Metrics

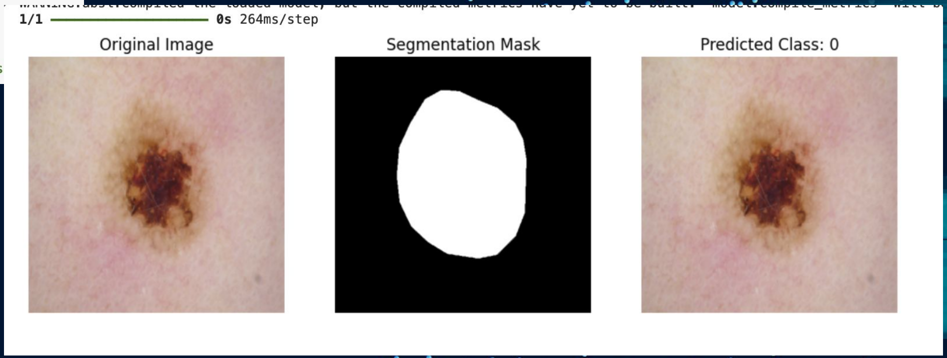


```
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 122ms/step
Classification Report:
              precision    recall  f1-score   support

         0.0       0.80      1.00      0.89       304
         1.0       0.00      0.00      0.00        75

    accuracy                           0.80       379
   macro avg       0.40      0.50      0.45       379
weighted avg       0.64      0.80      0.71       379

Accuracy: 0.8021108179419525
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_cla
```

Prediction



```python
import os
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import matplotlib.pyplot as plt

# Define paths
test_image_path = '/content/ISBI2016_ISIC_Part3B_Test_Data/ISIC_0010498.jpg'  # Example image
test_mask_path = '/content/ISBI2016_ISIC_Part3B_Test_Data/ISIC_0010498_Segmentation.png'  # Example mask

# Load and preprocess the image
def preprocess_image(img_path, img_size=(512, 512)):
    img = load_img(img_path, target_size=img_size)
    img = img_to_array(img) / 255.0
    return img

# Load and preprocess the mask
def preprocess_mask(mask_path, img_size=(512, 512)):
    mask = load_img(mask_path, target_size=img_size, color_mode='grayscale')
    mask = img_to_array(mask) / 255.0
    return mask

# Load the images
img = preprocess_image(test_image_path)
mask = preprocess_mask(test_mask_path)

# Load the model
model = load_model('/content/my_model.h5')

# Predict the class
img_input = np.expand_dims(img, axis=0)  # Model expects a batch of images
mask_input = np.expand_dims(mask, axis=0)  # Model expects a batch of masks
prediction = model.predict([img_input, mask_input])
predicted_class = np.argmax(prediction, axis=1)[0]  # Assuming the model outputs logits for each class
```



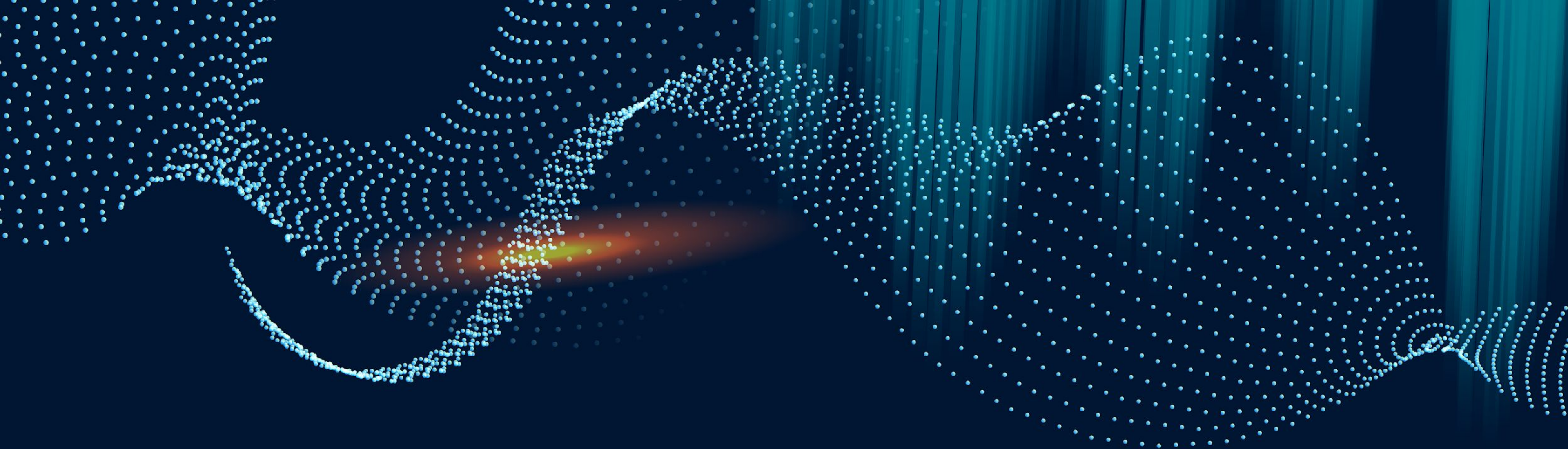
```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will b
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 264ms/step
```

Conclusion and Future Work

Testing on Larger Datasets:

- Extend the model's evaluation to larger and more diverse datasets to improve generalization across different populations and skin tones.

- The Dual Path CNN performed high accuracy than the rest two models which is 80%..
- Accomplished to Classify the images as Benign and Malignant.



Thank you!

References

- **D. S. Charan, H. Nadipineni, S. Sahayam, and U. Jayaraman**, “Method to classify skin lesions using dermoscopic images,” *Department of Computer Science, Indian Institute of Information Technology, Design and Manufacturing*, Chennai, Tamil Nadu, India, 2022.
- F. Mahmood, D. Desai, and J. Zhou, “An interpretable deep learning approach for skin cancer categorization,” *IEEE Transactions on Medical Imaging*, vol. 42, pp. 1123–1135, 2023.
- J. Kawahara, S. Daneshvar, G. Argenziano, and G. Hamarneh, “Seven-point checklist and skin lesion classification using multitask multimodal neural nets,” *IEEE Transactions on Biomedical Engineering*, vol. 69, pp. 348–358, 2018.

05 Conclusion

- The goal is to develop a deep learning-based framework for the early detection of skin cancer, focusing on improving diagnostic accuracy and survival rates.
- Build, Evaluate, Compare, and Select the best performing deep learning model to achieve this.