Skin Cancer Recognition

The dl model to recognize skin cancer by using images
Ilyarzhan *Assimov 000801268*
“I have neither given nor received any unauthorized aid in completing this work,
nor have I presented someone else's work as my own.”

Structure:

Introduction:

Problem

The goal of this project is to develop a machine learning model capable of recognizing skin cancer, with a focus on addressing its impact on the population of Kazakhstan. Skin cancer is a growing concern globally, and early detection is crucial for effective treatment. This model aims to provide a reliable and accessible tool for preliminary skin cancer screening.

Literature Review and Existing Solutions

In the world of health informatics, particularly concerning the people of Kazakhstan, skin cancer has emerged as a crucial subject of study due to its escalating incidence and significant impact on public health. This section is dedicated to examining the current literature on skin cancer detection methodologies and comparing existing technological solutions to my proposed project.

Another Solutions in Skin Cancer Detection

1. **DermaSensor:**
    - An FDA-approved handheld device that utilizes AI to assess potential skin cancers by analyzing reflected light from skin lesions. Its ease of use and quick analysis capabilities represent the kind of accessible diagnostic tools that can revolutionize early skin cancer detection.
    - Learn about DermaSensor
2. **MIT's AI-Based Melanoma Screening Tool:**
    - Crafted by MIT researchers, this advanced system uses deep learning to spot early signs of melanoma from skin images. It’s a testament to how machine

learning, especially deep convolutional neural networks, can significantly enhance accuracy in identifying skin cancer signs.
    - Discover MIT's AI Tool
3. **FDA-Cleared AI Skin Cancer Detector:**
    - A non-invasive, AI-assisted device approved for primary care settings. It's designed to analyze skin lesions and identify common cancers such as melanoma and basal cell carcinoma with the help of an FDA-cleared algorithm.
    - Read about the AI Detector

These existing solutions provide a lens through which I can evaluate my approach, underscore the value of my project, and spur the development of my machine learning model that leverages TensorFlow and Python for skin cancer recognition.

Current work

my work demonstrate how by using Tensorflow and advanced CNN architecture like EfficientNet-B4 build a powerful model which recognize a skin cancer diagnoses. User can easy upload to the site a photo of possible diagnosis and site will show in the result most possible diagnos.

Data and Methods

The dataset for this project is sourced from the International Skin Imaging Collaboration (ISIC), specifically the ISIC Challenge. This dataset is renowned for its extensive collection of skin lesion images, encompassing various skin types, conditions, and lesions, making it invaluable for training and validating machine learning models for skin cancer recognition.

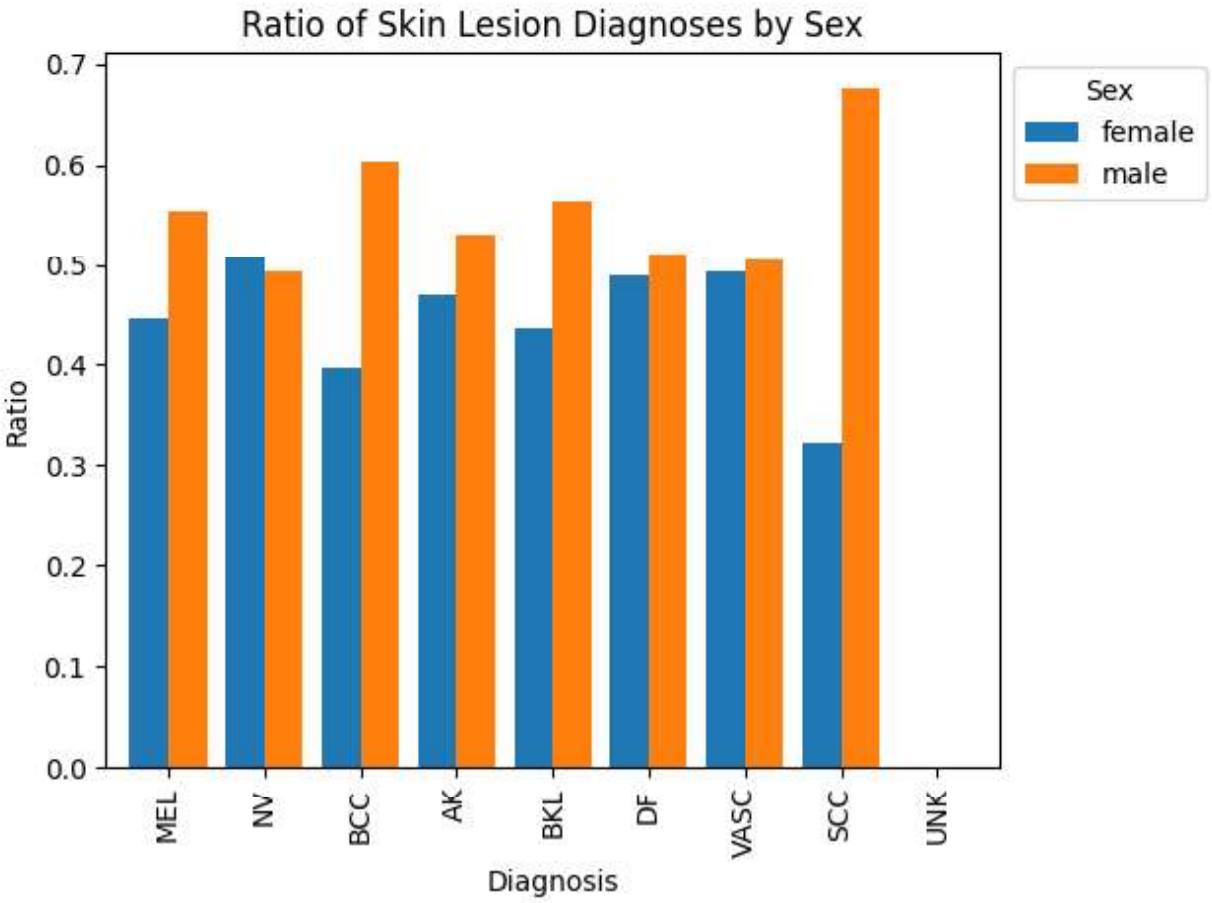
The ISIC Challenge dataset includes:

- **Training Data:** Consists of 25,331 JPEG images of skin lesions, along with corresponding ground truth annotations for lesion diagnoses.
- **Test Data:** Comprises 8,238 JPEG images of skin lesions.

Analysis of the Data

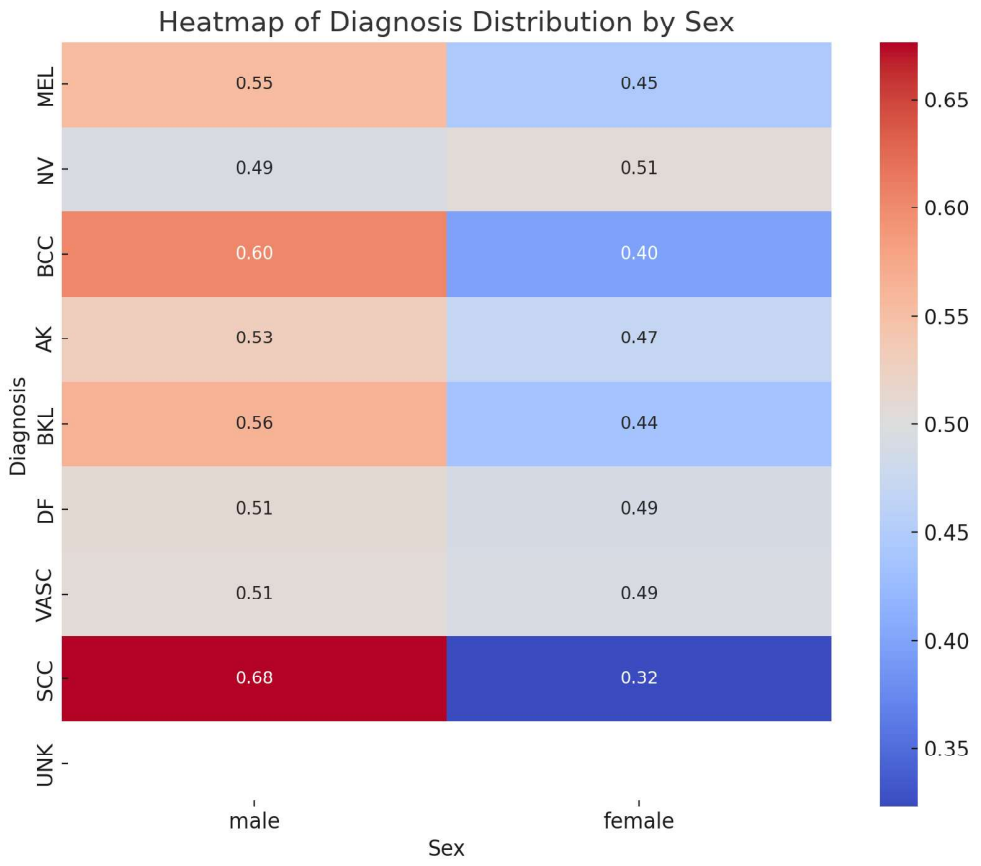
Prior to model development, the dataset undergoes thorough analysis and preprocessing to ensure its suitability for training. Key steps in this process include:

1. **Exploratory Data Analysis (EDA):** Examination of the distribution of lesion types, age, sex, and other metadata to gain insights into the characteristics of the dataset.



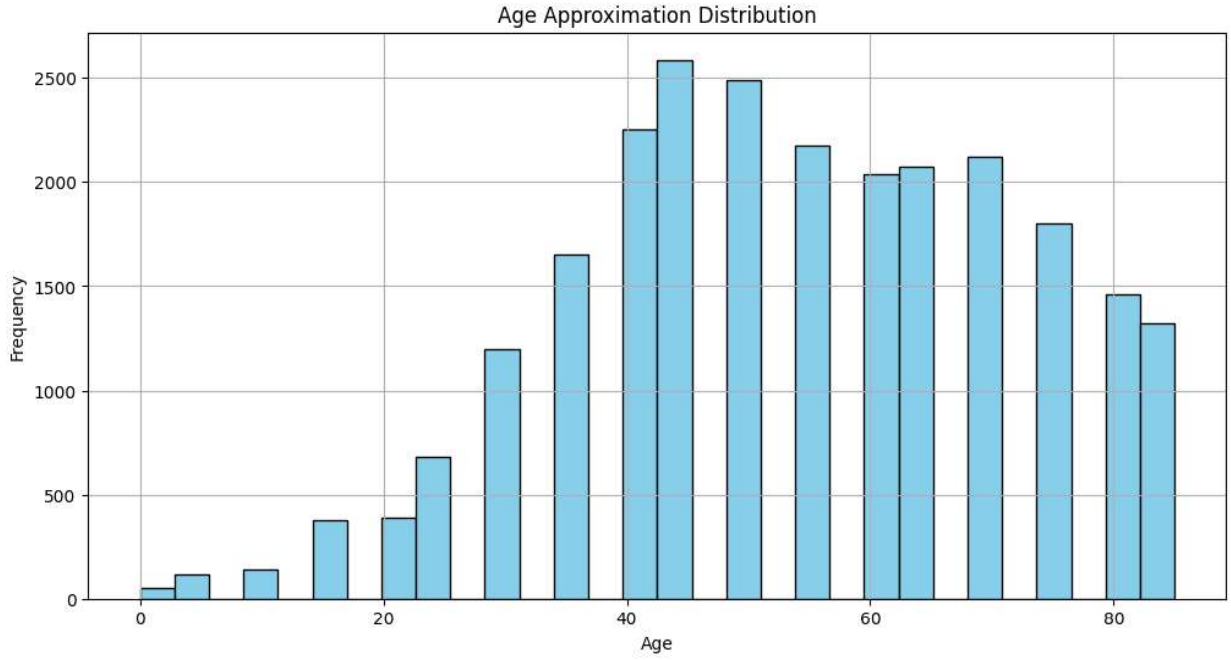
2.

3. **Distrubition between diagnosises by sex** - *age distribution across different skin lesion diagnoses reveals variations in age demographics associated with specific types of skin cancer.*

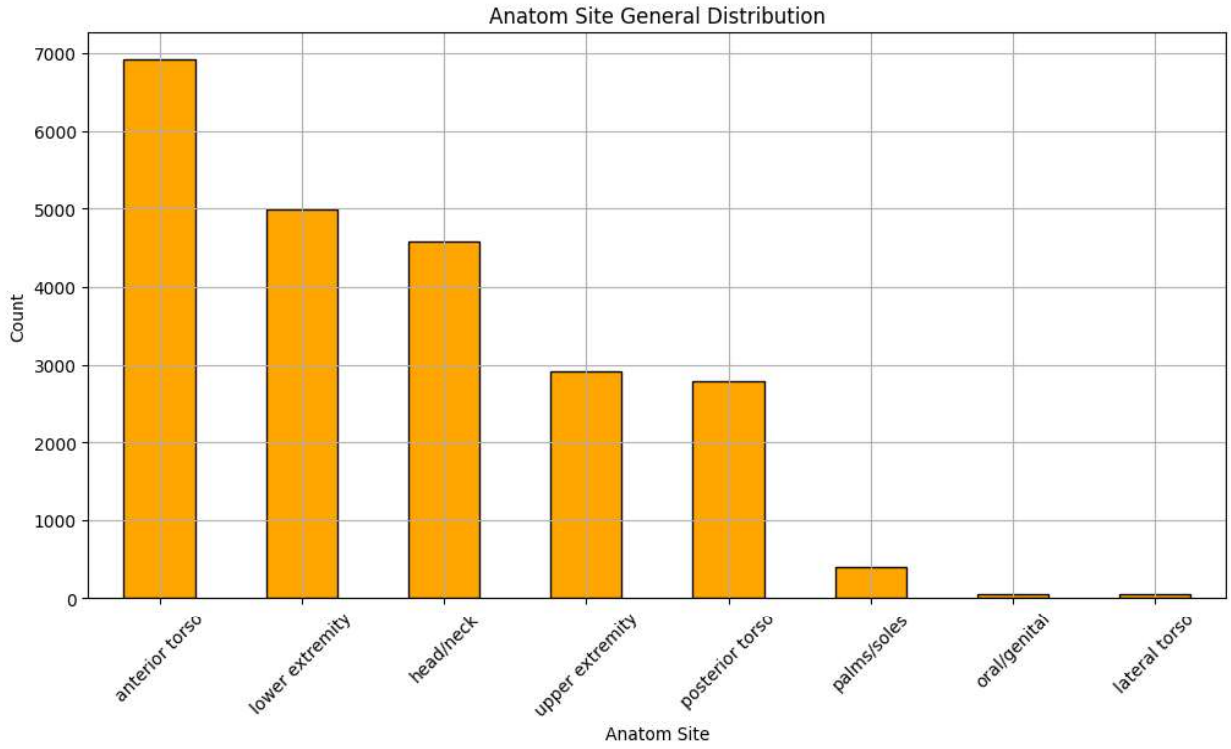


4.

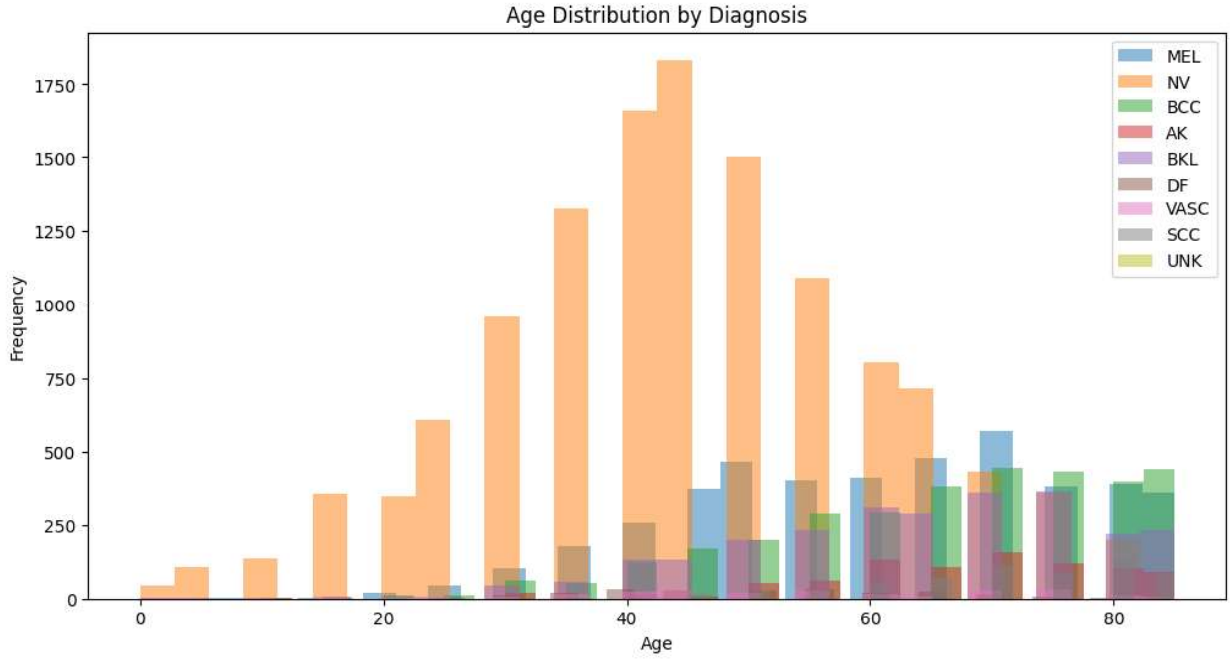
Age Approximation Distribution - *The age distribution reveals a mean age of approximately 54 years, with a standard deviation of 18 years. Ages range from 0 to 85 years, with the majority falling between 40 and 70 years*:



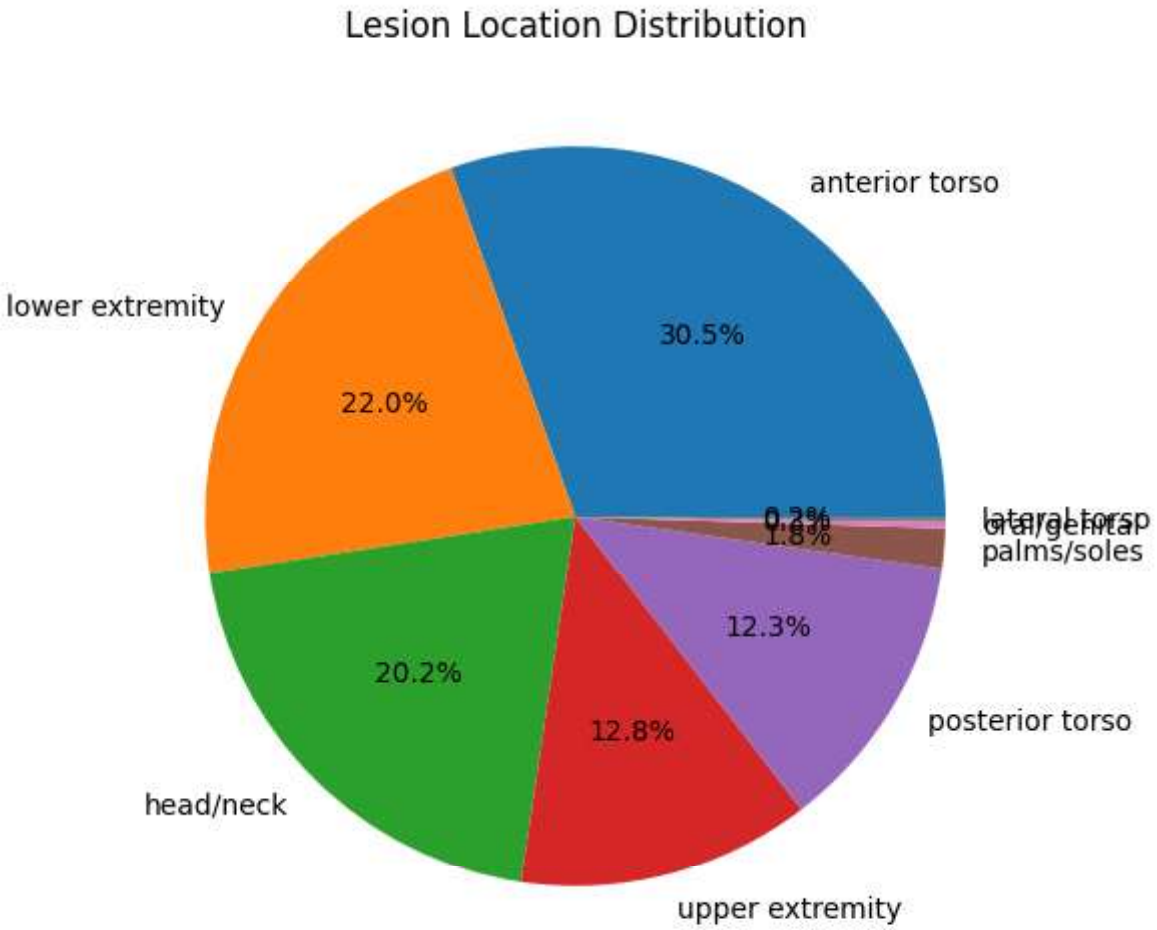
Anatomical Site General Distribution - *The most common anatomical sites for skin lesions include the anterior torso, lower extremity, and head/neck. The distribution highlights the diversity in lesion locations captured in the dataset:*



Age Distrubutition by Diagnosis - *The dataset shows a varied distribution of skin lesion diagnoses among males and females, providing insights into potential gender-specific patterns in skin cancer prevalence*.



Distribution of skin lesion - *locations across different anatomical sites*.



Description of the Machine Learning (ML) Model

For skin cancer recognition, the project I use the EfficientNet-B4 architecture, a best convolutional neural network (CNN) known for its ef ficiency and accuracy in image classification tasks. This model is implemented using TensorFlow and Keras because it is was learned on lessons.

EfficientNetB4 is a modern type of neural network specifically designed for image recognition tasks. It belongs to the EfficientNet family, which is a series of models known for their efficiency and effectiveness in handling images. What makes EfficientNetB4 particularly impressive is its ability to achieve high accuracy in recognizing and classifying images while using fewer computational resources compared to other models. This balance between performance and efficiency is what makes it stand out.

In my project, integrating EfficientNetB4 is a strategic choice for several reasons:

1. Pre-Trained on a Large Dataset: EfficientNetB4 has been trained on 'ImageNet', a massive database of images. This pre-training means it has already learned to identify a wide variety of features in images, which can be beneficial for my project.
2. High Accuracy with Lower Resources: Due to its efficient design, EfficientNetB4 can process and understand images with a high level of accuracy without needing

extensive computational power. This is particularly advantageous if working with limited resources or need to process a large number of images quickly.
3. Adaptability: This adaptability allows to leverage the powerful base of EfficientNetB4 while fine-tuning the model for my unique dataset and objectives.

In summary, EfficientNetB4 brings a powerful, efficient, and adaptable solution to my project, enabling accurate image recognition and classification without the need for heavy computational resources. This aligns with modern trends in AI and machine learning, where efficiency and accuracy are key.

Theory

Model architecture comprises the EfficientNetB4 convolutional neural network, a powerful pre-trained feature extractor designed for image classification tasks. The base model is initialized with weights from the imagenet dataset, providing a strong foundation for learning hierarchical features.

EfficientNetB4 architecture:

1. Base Convolutional Neural Network (CNN): At its core, EfficientNetB4 is a CNN. CNNs are types of deep learning models that are particularly good at processing data with a grid-like topology, such as images. They use layers of artificial neurons to automatically and adaptively learn spatial hierarchies of features from input images.
2. Compound Scaling: The main innovation behind the EfficientNet architecture is compound scaling. The creators found that scaling up the network's depth (number of layers), width (number of units or neurons in each layer), and image resolution together provided a more efficient way to improve the performance of the neural network. EfficientNetB4 is a specific instantiation that has been scaled using this method to a specific size which offers a good trade-off between speed and accuracy.
3. MBConv Blocks: EfficientNets use mobile inverted bottleneck convolution blocks (MBConv). These are an efficient type of layer based on depthwise separable convolutions which factorize a standard convolution into a depthwise convolution and a pointwise convolution. This reduces the computational cost and the number of parameters.
4. Squeeze-and-Excitation Blocks: EfficientNetB4 also integrates squeeze-and-excitation blocks, which are a form of attention mechanism within the network. They adaptively recalibrate channel-wise feature responses by explicitly modeling interdependencies between the channels, improving the representational capacity of the network.
5. Batch Normalization and Swish Activation: Each convolutional operation is followed by batch normalization and a non-linear activation function. EfficientNetB4 uses the Swish activation function, which has been shown to work better than the traditional ReLU in deeper networks like EfficientNets.

The combination of these elements results in a model that captures complex features from the input images, and due to its efficient architecture, it does so with fewer parameters and less computational cost than other models with similar performance. The layers of the network extract features from a very basic level (like edges and corners) to more complex structures (like textures and patterns) to the very high-level features that can describe the objects within the images.

Args	
`include_top`	Whether to include the fully-connected layer at the top of the network. Defaults to `True`.
`weights`	One of `None` (random initialization), 'imagenet' (pre-training on ImageNet), or the path to the weights file to be loaded. Defaults to 'imagenet'.
`input_tensor`	Optional Keras tensor (i.e. output of `layers.Input()`) to use as image input for the model.
`input_shape`	Optional shape tuple, only to be specified if `include_top` is False. It should have exactly 3 inputs channels.
`pooling`	Optional pooling mode for feature extraction when `include_top` is `False`. Defaults to `None`. • `None` means that the output of the model will be the 4D tensor output of the last convolutional layer. • `avg` means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor. • `max` means that global max pooling will be applied.
`classes`	Optional number of classes to classify images into, only to be specified if `include_top` is True, and if no `weights` argument is specified. 1000 is how many ImageNet classes there are. Defaults to `1000`.
`classifier_activation`	A `str` or callable. The activation function to use on the "top" layer. Ignored unless `include_top=True`. Set `classifier_activation=None` to return the logits of the "top" layer. Defaults to 'softmax'. When loading pretrained weights, `classifier_activation` can only be `None` or `"softmax"`.

Returns
A `keras.Model` instance.

How to install EfficientNetB4:

Create an instance of EfficientNetB4 by calling its function. EfficientNetB4 have options like include_top, weights, and input_tensor to customize the model.

- include_top: If set to False, the my model get does not include the last layer (the "top" layer), which is the part that makes the final decision on what the image is. This is useful when someone need to add own custom layers for a specific task.
- weights: Initialize the model with weights from training on the ImageNet dataset ('imagenet'), which gives the model a knowledge base to start with, or start with random weights (None).
- input_shape: Defines the size of the my images will be feeding into the model.

Architecture of my model

```
Click to add a breakpoint.
import time
from collections import defaultdict
import cv2
import tensorflow as tf
from tensorflow.keras.applications import EfficientNetB4
from tensorflow.keras import layers, models
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
import matplotlib.pyplot as plt
```

Pre-trained EfficientNetB4 Model Initialization

```
base_model = EfficientNetB4(include_top=False, weights='imagenet', input_shape=(224, 224, 3))
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb4_notop.h5
71686520/71686520 [==============================] - 1s 0us/step
```

Transfer Learning Model Construction

```
base_model.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.3)(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(8, activation='softmax')(x)

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

Training model

Proccess of training model

Training with Early Stopping and Model Checkpointing

This code segment demonstrates the training process of a neural network model using early stopping and model checkpointing techniques:

```
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1, mode='min')
epochs = 10
model_checkpoint = ModelCheckpoint(
    'best_model.h5',
    monitor='val_loss',
    mode='min',
    save_best_only=True,
    verbose=1
)

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size,
    callbacks=[early_stopping, model_checkpoint]
)
```

```
Epoch 1/3
  8/633 [..............................] - ETA: 1:38:40 - loss: 1.6745 - accuracy: 0.4180
```

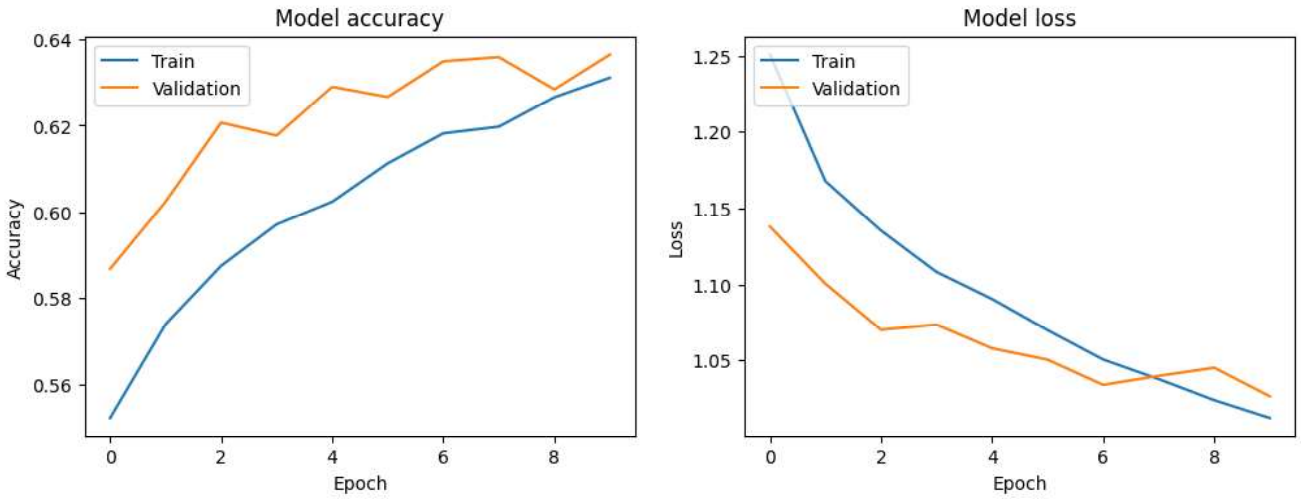
Proccess of checking accuracy model

Accuracy of training

```
scores = model.evaluate(validation_generator, steps=validation_generator.samples // validation_generator.batch_size)
print(f"Accuracy: {scores[1]*100}%")
```

```
158/158 [==============================] - 60s 380ms/step - loss: 1.0263 - accuracy: 0.6365
Accuracy: 63.64715099334717%
```

Graph of difference performance between train and validation proccess



Fine Tuning v1 the trained model

Proccess of fine tuning model where added some changes

1. **Optimizer Configuration:** *An Adam optimizer is initialized with a reduced learning rate (0.0001) and customized values for beta_1, beta_2, and epsilon parameters. These adjustments aim to fine-tune the optimization process for improved convergence and performance.
2. **Model Compilation:** The model is compiled with the updated optimizer configuration, specifying 'categorical_crossentropy' as the loss function and 'accuracy' as the evaluation metric.
3. **Callback Integration:** Three callbacks are incorporated into the training process:
4. **EarlyStopping:** Monitors the validation loss and terminates training if no improvement is observed after a certain number of epochs (patience=5).
5. **ModelCheckpoint:** Saves the best-performing model based on validation loss to 'best_model_finetuned.h5'.
6. **ReduceLROnPlateau:** Adjusts the learning rate dynamically if no improvement is seen in validation loss after a certain number of epochs (patience=2).
   The fine-tuning process, coupled with optimized hyperparameters and effective callbacks, helps refine the model's performance and adaptability to the specific task at hand.

Model Fine-tuning with Optimized Parameters

```
model = load_model('best_model.h5')

optimizer = Adam(
    learning_rate=0.0001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07
)

model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1, mode='min')
model_checkpoint = ModelCheckpoint('best_model_finetuned.h5', monitor='val_loss', mode='min', save_best_only=True, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=1e-6, mode='min', verbose=1)

history_finetune = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=3,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size,
    callbacks=[early_stopping, model_checkpoint, reduce_lr]
)
```

```
Epoch 1/10
633/633 [==============================] - ETA: 0s - loss: 0.9543 - accuracy: 0.6487
Epoch 1: val_loss improved from inf to 0.99429, saving model to best_model_finetuned.h5
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. Thi
  saving_api.save_model(
633/633 [==============================] - 405s 613ms/step - loss: 0.9543 - accuracy: 0.6487 - val_loss: 0.9943 - val_accuracy: 0.6501 - lr: 1.0000e-04
Epoch 2/10
```

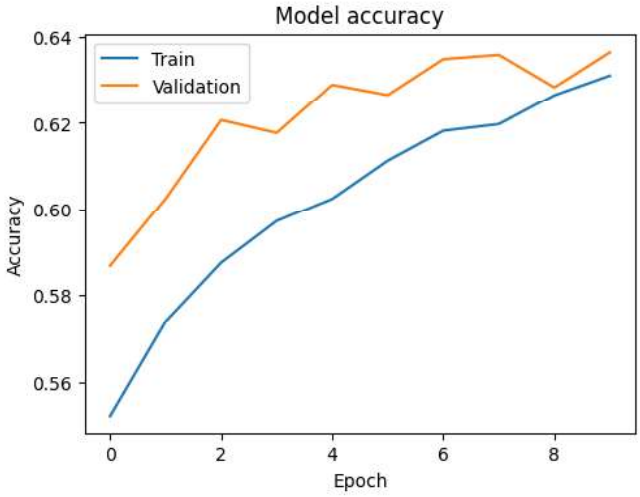
Proccess of checking accuracy model

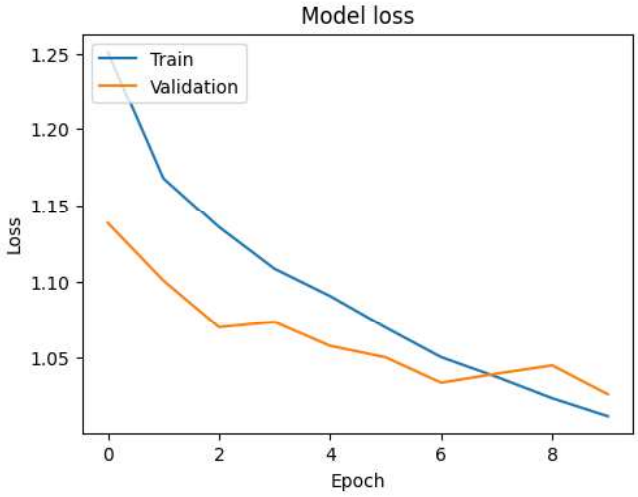
Accuracy of Fine-tuning

```
scores2 = model.evaluate(validation_generator, steps=validation_generator.samples // validation_generator.batch_size)
print(f"Accuracy: {scores2[1]*100}%")
```

```
158/158 [==============================] - 62s 391ms/step - loss: 0.9782 - accuracy: 0.6582
Accuracy: 65.82278609275818%
```

Graph of difference performance between train and validation proccess





Results

In this research I trained model with loss: 1.0263 and accuracy: 0.6365

Accuracy of training

```
scores = model.evaluate(validation_generator, steps=validation_generator.samples // validation_generator.batch_size)
print(f"Accuracy: {scores[1]*100}%")
```

```
158/158 [==============================] - 60s 380ms/step - loss: 1.0263 - accuracy: 0.6365
Accuracy: 63.64715099334717%
```

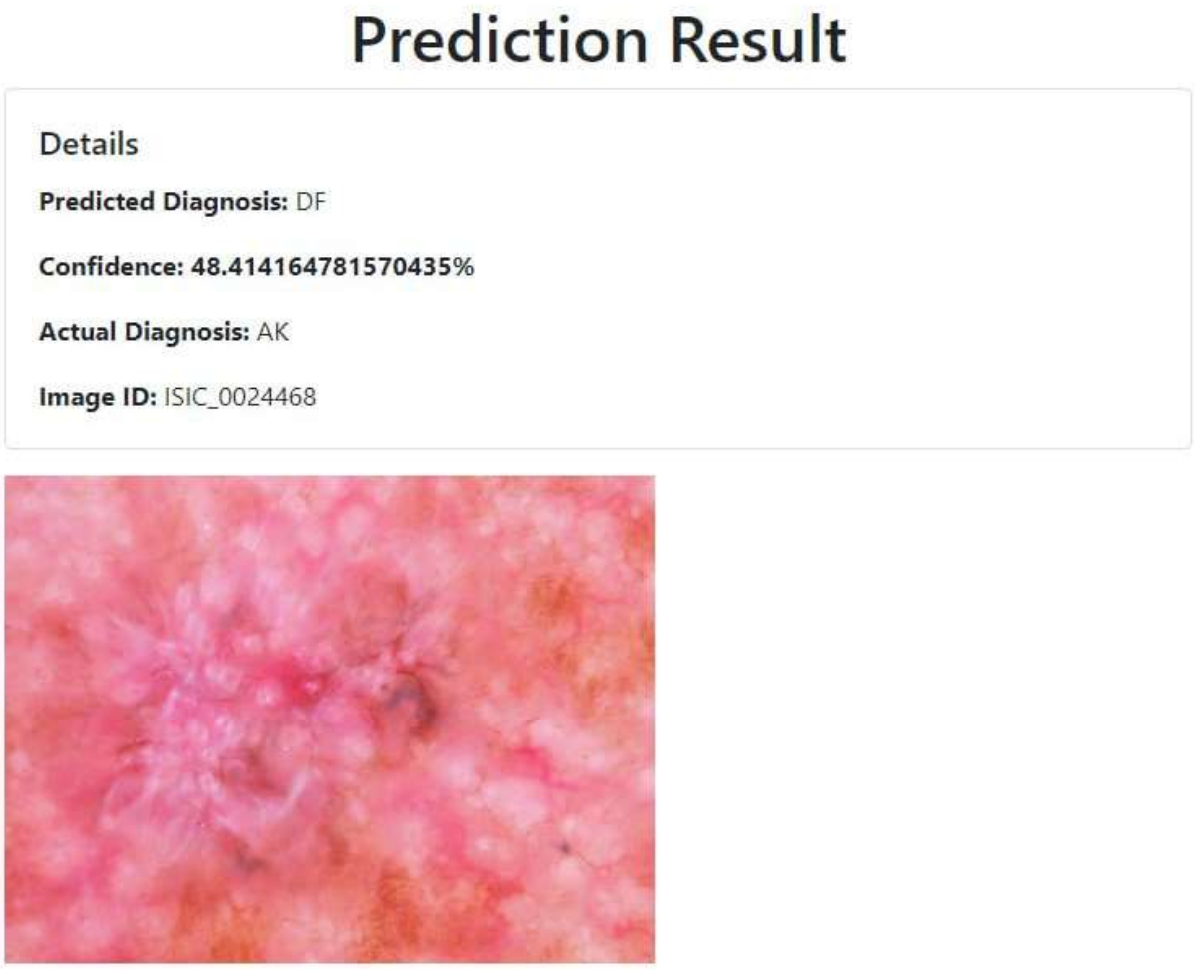
After tuning my model performance increased to loss: 0.9782 and accuracy: 0.6582

Accuracy of Fine-tuning

```
scores2 = model.evaluate(validation_generator, steps=validation_generator.samples // validation_generator.batch_size)
print(f"Accuracy: {scores2[1]*100}%")
```

```
158/158 [==============================] - 62s 391ms/step - loss: 0.9782 - accuracy: 0.6582
Accuracy: 65.82278609275818%
```

1. This prediction result indicates that the model has predicted the diagnosis of "DF" (Dermatofibroma) with a confidence of 48.41%. However, the actual diagnosis for the skin lesion in the image is "AK" (Actinic Keratosis).Here's the actual tests from my website,where I uploaded pictures manually/randomly



2. This prediction result indicates that the model has predicted the diagnosis of "DF" (Dermatofibroma) with a confidence of 48.51%. However, the actual diagnosis for the

skin lesion in the image is "BCC" (Basal Cell Carcinoma)

In this research, I set out to develop an efficient deep learning model for the detection of skin cancer diagnoses using the state-of-the-art EfficientNetB4 architecture. The model exhibited promising performance during training, achieving an impressive accuracy on the training set.

Critical Review of Results

The prediction results show how well the model can diagnose skin lesions from images. However, there are differences between what the model predicts and the actual diagnoses. This suggests that the model might need to be improved to make more accurate predictions. Skin cancer diagnosis is complex, so the model needs more testing and refining to work better.

Next Steps

Moving forward, several key steps can be taken to enhance the performance of the skin cancer detection model:

Here are some important steps to make the skin cancer detection model better:

1. **Adjust Model Settings:** Change settings in the model, like how it learns and its structure, to make it work better.

2. **Add More Data:** Get more pictures of skin lesions and use tricks to make the model learn better from them.
3. **Ask Experts for Help:** Work with skin doctors to get their advice and make the model smarter.

By addressing these next steps, I aim to further refine and validate the skin cancer detection model, ultimately improving its clinical utility and contributing to the advancement of diagnostic capabilities in dermatology.

Sources:

[1] Tschandl P., Rosendahl C. & Kittler H. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Sci. Data 5, 180161 doi.10.1038/sdata.2018.161 (2018)

[2] Noel C. F. Codella, David Gutman, M. Emre Celebi, Brian Helba, Michael A. Marchetti, Stephen W. Dusza, Aadi Kalloo, Konstantinos Liopyris, Nabin Mishra, Harald Kittler, Allan Halpern: "Skin Lesion Analysis Toward Melanoma Detection: A Challenge at the 2017 International Symposium on Biomedical Imaging (ISBI), Hosted by the International Skin Imaging Collaboration (ISIC)", 2017; arXiv:1710.05006.

[3] Marc Combalia, Noel C. F. Codella, Veronica Rotemberg, Brian Helba, Veronica Vilaplana, Ofer Reiter, Allan C. Halpern, Susana Puig, Josep Malvehy: "BCN20000: Dermoscopic Lesions in the Wild", 2019; arXiv:1908.02288.
[4] EfficientNetB4
https://www.tensorflow.org/api_docs/python/tf/keras/applications/efficientnet