# Deep Learning Hyperparameter Space Optimization for Skin Lesion Classification: Model Pipeline Design

## Introduction

Deep Learning has been widely used in different fields, such as computer vision, natural language processing, and reinforcement learning. Deep Learning models have proven capable of complex problems solving, such as image classification, object detection, and speech recognition. A particular field they have been used is in medical imaging, where they try to solve different problems like classification different types of diseases, such as skin lesions, tumors, and other medical conditions. This project aims to develop a model pipeline for skin lesions classification using deep learning models. Even thought the project will concentrate on the optimization process and comparison of different architectures, part of the goals is to develop an effective classification model to tackle the challenges of current skin lesion classification. The model pipeline will address the steps necessary for the creation of an effective classification model development including; data acquisition, data preparation, exploratory data analysis, model selection and building, hyperparameter optimization, training and evaluation, and results analysis, including decision-making process, and the different techniques used throughout the creation of the model pipeline.

### Stakeholders Table

|  |  |
| --- | --- |
| Stakeholder | Role |
| Luis Enrique Ramos Soto | Data Scientist |

## Supporting Evidence and Justification

The need for advance diagnostic tools in dermatology is underscored by the rising incidence of skin cancer and the need for early detection and treatment, while maintaining the sensitivity and specificity of the diagnostic process. The incidence of misdiagnosis and overdiagnosis in skin cancer is a significant issue, with studies showing that the accuracy of dermatologists in diagnosing skin cancer is around 60-70%. The use of deep learning models for skin lesion classification has shown promising results, with studies showing that deep learning models can achieve accuracy rates of 80-90% in skin lesion classification tasks. Nonetheless, many challenges remains in the development of effective deep learning models for skin lesion classification, such as the optimization of hyperparameters space, the selection of the most suitable deep learning architecture, and the evaluation of the model's performance. This project aims to address these challenges by developing a model pipeline for skin lesion classification using deep learning models, and exploring the hyperparameters space of different deep learning architectures to enhance the model's performance.

## Proposed Solution

### Methodological Framework - Iterative Model Development CRISP-DM

The project will follow an iterative model development approach, trying to align to the CRISP-DM methodology. The iterative model development approach, where insights from following stages, like Evaluation or Hyperparameter Optimization, can present new insights that could lead to reevaluate previous stages, like Data Preparation or Model Selection. This iterative approach will allow the project to adapt to emerging insights, aligning with the goal of developing an effective classification model, that can tackle the challenges of skin lesion classification with high accuracy and clinical relevance.

By adhering to the adaptability of CRISP-DM, the project ensures a systematic yet flexible progression through the states of the model development, maintaining a rigorous and reflective research mentality to achieve the project's core objectives, while maintaining the moral and ethical considerations of the project.

### High-Level Solution Design

The project proposes the development of a deep learning-based model pipeline specifically tailored for skin lesion classification. The solution leverage state-of-the-art deep learning architectures, such as VGG16, VGG19, ResNet101V2, InceptionResNetV2, Xception, and MobileNetV2, known for their prowess in image recognition tasks, to accurately classify various type of skin lesions. This approach aims to optimize the hyperparameters space of the deep learning architectures using Optuna, a hyperparameter optimization library, to enhance the model's performance.

The model pipeline will be developed using Python, and will implement different libraries and sources, such as:

* Pandas: Data manipulation and analysis
* NumPy: Numerical computing
* Matplotlib: Data visualization
* Scikit-learn: Machine learning
* TensorFlow: Deep learning
* Optuna: Hyperparameter optimization
* HAM10000 dataset: Dermatoscopic images of common pigmented skin lesions

### High-level Pipeline Design

The project aims to research different deep learning models architectures, and implement a comprehensive model pipeline dedicated to skin lesion classification. The pipeline is designed to is designed to create a structured and systematic approach to the development of the model, each step important to the development of an efficient and effective classification model. The envisioned pipeline encapsulates the following sequential stages:

1. Data Acquisition: This stage involves the collection and sourcing of dermatological imaging data from reliables sources, such as the HAM10000 dataset, then the loading and integration of data into the main project's processing framework. This stage should ensure the data is cohesive for subsequent analysis and model training.
2. Data Preparation: Before the model training the raw data must be preprocessed. The data preprocessing stage involves the comprehensive cleaning, transformation, categorization, and normalization of numerical and categorical data. This stage is crucial to ensure the data is in a suitable format for model training.
3. Exploratory Data Analysis (EDA): EDA employs a range of statistical techniques and visual tools, such as histograms, scatter plots, and correlation matrices, to explore and identify patterns, distributions, and anomalies with in the data. EDA serves as a critical stage in hypothesizing the model structures and potential predictive features.
4. Model Selection and Building: Following the EDA insights, the model selection focus on the evaluation and selection of a suitable deep learning architecture for the skin lesion classification task. It involves a comparative analysis of different deep learning architectures. The model selections consider theoretical and empirical evidence of the architectures' effectiveness in image classification tasks. Then the model building stage involves the adaptation and implementation of the selected architecture to the skin lesion classification task and the different classification labels. It also involves techniques such as transfer learning, data augmentation, and regularization to enhance the model's performance.
5. Training and Initial Evaluation: The training stage involves the training of the model using the training dataset, and the initial evaluation of the model using the validation dataset. The training process involves the optimization of the model's weights and biases using backpropagation and gradient descent. The initial evaluation involves the calculation of different metrics, such as accuracy, precision, recall, F1-score, and ROC-AUC, to evaluate the model's performance.
6. Hyperparameter Optimization: The hyperparameter optimization stage involves the exploration of the hyperparameters space of the selected deep learning architectures using Optuna. The hyperparameters optimization aims to enhance the model's performance by finding the optimal hyperparameters values for the model. The hyperparameters optimization process involves the definition of the hyperparameters space, the objective function, and the optimization algorithm.
7. Refinement and Final Evaluation: The refinement stage involves the refinement of the model using the optimal hyperparameters values found during the hyperparameters optimization stage. The final evaluation involves the evaluation of the refined model using the test dataset. The final evaluation involves the calculation of different metrics, such as accuracy, precision, recall, F1-score, and ROC-AUC, to evaluate the model's performance.
8. Results Analysis: The results analysis stage involves the analysis of the model's performance, and the comparison of the different deep learning architectures. The results analysis aims to provide insights on the performance of the models, and recommendations for further improvements.
9. Final Model and Report: The final stage involves the creation of the final model from the refined model, and the creation of the final report. The final report will provide a detailed description of the model pipeline, and the different steps involved in the process, including decision-making process, and the different techniques used throughout the creation of the model pipeline.

## Rationale for Proposed Solution

The proposed solution aims to address the challenges of skin lesion classification using deep learning models, by developing a model pipeline that leverages state-of-the-art deep learning architectures, and optimizes the hyperparameters space of the architectures using Optuna. The proposed solution is based on the following rationale:

* **Deep Learning Efficiency**: Deep learning models, particularly convolutional neural networks (CNNs), have shown remarkable efficiency in image classification tasks, outperforming traditional machine learning models in terms of accuracy and generalization. Their ability to learn hierarchical representation makes them well-suited for complex visual task such as the skin lesion classification from dermatoscopic images.
* **Accessibility of Deep Learning Architectures**: The availability of pre-trained deep learning architectures, such as VGG16, VGG19, ResNet101V2, InceptionResNetV2, Xception, and MobileNetV2, provides a wide range of options for the development of the skin lesion classification model. These can also be deployed acros various platforms and frameworks, such as TensorFlow and Keras.
* **Data-driven Decision-Making**: Utilizing the CRISP-DM methodology, the project will adopt a data-driven approach to model development, ensuring that decisions are based on empirical evidence and insights derived from the data. This approach will enhance the robustness and reliability of the model pipeline.
* **Data Quality and Quantity**: The HAM10000 dataset provides a rich source of dermatoscopic images of common pigmented skin lesions, which is essential for the training and evaluation of the skin lesion classification model. The high-quality and diverse nature of the dataset will enable the model to learn and generalize effectively.
* **Hyperparameter Optimization**: The use of Optuna for hyperparameter optimization will enable the project to explore the hyperparameters space of the deep learning architectures efficiently, and find the optimal hyperparameters values that enhance the model's performance. This will lead to a more effective and efficient model pipeline.
* **Iterative Model Development**: The iterative model development approach, aligned with the CRISP-DM methodology, will allow the project to adapt to emerging insights and optimize the model pipeline iteratively. This approach will ensure that the model pipeline is continuously refined and improved throughout the development process.

# Detailed Solution Design

## Libraries and Tools

# Standard library imports  
import os.path  
import io  
import pickle  
import math  
  
# Third party packages used for data handling and mathematical operations  
import pandas as pd  
import numpy as np  
from sklearn.metrics import classification\_report, roc\_auc\_score, confusion\_matrix  
from sklearn.preprocessing import LabelBinarizer  
from sklearn.utils import compute\_class\_weight  
from sklearn.metrics import roc\_curve, auc  
  
# Visualization tools  
import matplotlib.pyplot as plt  
import seaborn as sns  
import plotly.io as pio  
import plotly.graph\_objects as go  
  
# Deep learning and Machine learning libraries  
import tensorflow as tf  
import tensorflow.keras.backend as K  
import tensorflow\_addons as tfa  
from tensorflow.keras.applications import VGG19  
from tensorflow.keras.applications.vgg16 import preprocess\_input as vgg16\_preprocess\_input  
from tensorflow.keras.applications.vgg19 import preprocess\_input as vgg19\_preprocess\_input  
from tensorflow.keras.applications.resnet\_v2 import preprocess\_input as resnet101v2\_preprocess\_input  
from tensorflow.keras.applications.inception\_resnet\_v2 import preprocess\_input as inception\_resnetv2\_preprocess\_input  
from tensorflow.keras.applications.xception import preprocess\_input as xception\_preprocess\_input  
from tensorflow.keras.applications.mobilenet\_v2 import preprocess\_input as mobilenet\_v2\_preprocess\_input  
from tensorflow.keras.models import Model  
from tensorflow.keras.layers import Dense, Flatten, Dropout, BatchNormalization, Activation, Input  
from tensorflow.keras.layers import Multiply, Reshape, GlobalAveragePooling2D, Conv2D, Permute  
from tensorflow.keras.optimizers import AdamW  
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau  
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
from keras.src.losses import CategoricalFocalCrossentropy  
  
# Machine learning optimization and hyperparameter tuning libraries  
from optuna.integration import TFKerasPruningCallback  
import optuna  
  
# Libraries for working with images and datasets  
from PIL import Image  
from datasets import load\_dataset, Dataset

2024-04-01 22:05:34.039441: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF\_ENABLE\_ONEDNN\_OPTS=0`.  
2024-04-01 22:05:34.064319: E external/local\_xla/xla/stream\_executor/cuda/cuda\_dnn.cc:9261] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered  
2024-04-01 22:05:34.064350: E external/local\_xla/xla/stream\_executor/cuda/cuda\_fft.cc:607] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered  
2024-04-01 22:05:34.065037: E external/local\_xla/xla/stream\_executor/cuda/cuda\_blas.cc:1515] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered  
2024-04-01 22:05:34.071538: I tensorflow/core/platform/cpu\_feature\_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.  
To enable the following instructions: AVX2 AVX\_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.  
2024-04-01 22:05:34.636122: W tensorflow/compiler/tf2tensorrt/utils/py\_utils.cc:38] TF-TRT Warning: Could not find TensorRT  
/home/lramossoto/.virtualenvs/SkinCancerClassification/lib/python3.10/site-packages/tensorflow\_addons/utils/tfa\_eol\_msg.py:23: UserWarning:   
  
TensorFlow Addons (TFA) has ended development and introduction of new features.  
TFA has entered a minimal maintenance and release mode until a planned end of life in May 2024.  
Please modify downstream libraries to take dependencies from other repositories in our TensorFlow community (e.g. Keras, Keras-CV, and Keras-NLP).   
  
For more information see: https://github.com/tensorflow/addons/issues/2807   
  
 warnings.warn(

## Data Acquisition

Hugging Face provides an easy way to load the HAM10000 dataset using the datasets library. The 'marmal88/skin\_cancer' dataset contains dermatoscopic images from the International Skin Imaging Collaboration (ISIC) 2018 challenge, which includes images of common pigmented skin lesions. The dataset is divided into training, validation, and test sets, and each sample contains the image and the corresponding label.

### Dataset HAM10000

The Human Against Machine with 10000 training images (HAM10000) was a challenge aimed at the automated diagnosis of skin lesions. The dataset contains 10015 dermatoscopic images of common pigmented skin lesions, which were collected from different populations, age groups, and skin types. The dataset includes seven different types of skin lesions: melanocytic nevi, melanoma, basal cell carcinoma, actinic keratosis, benign keratosis, dermatofibroma, and vascular lesions. Most of the images originate from the Department of Dermatology at the Medical University of Vienna, the Department of Dermatology at the University of Queensland, and the Melanoma Unit at the Hospital Clinic in Barcelona. The data is provided under the Creative Commons license (CC BY-NC-SA 4.0) by 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).

The reason for using the HAM10000 dataset from the ISIC 2018 challenge, instead of a more recent dataset, is mainly due to its availability and quality. The HAM10000 dataset is a well-known and widely used dataset in the field of skin lesion classification, and it provides a widely proven benchmark for evaluating the performance of skin lesion classification models.

# Load the HAM10000 dataset  
dataset = load\_dataset('marmal88/skin\_cancer')  
pd\_dataset = pd.DataFrame(dataset['train'].to\_pandas())  
val\_dataset = pd.DataFrame(dataset['validation'].to\_pandas())  
test\_dataset = pd.DataFrame(dataset['test'].to\_pandas())  
pd\_dataset.head()

image image\_id \  
0 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... ISIC\_0024329   
1 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... ISIC\_0024372   
2 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... ISIC\_0024418   
3 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... ISIC\_0024450   
4 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... ISIC\_0024463   
  
 lesion\_id dx dx\_type age sex localization   
0 HAM\_0002954 actinic\_keratoses histo 75.0 female lower extremity   
1 HAM\_0005389 actinic\_keratoses histo 70.0 male lower extremity   
2 HAM\_0003380 actinic\_keratoses histo 75.0 female lower extremity   
3 HAM\_0005505 actinic\_keratoses histo 50.0 male upper extremity   
4 HAM\_0004568 actinic\_keratoses histo 50.0 male upper extremity

print(type(dataset))

<class 'datasets.dataset\_dict.DatasetDict'>

## Exploratory Data Analysis (EDA)

The exploratory data analysis (EDA) is an essential step in understanding the dataset and identifying patterns, distributions, and anomalies within the data. In this section, we will perform an exploratory data analysis of the HAM10000 dataset to gain insights into the dataset's structure, distribution of labels, and characteristics of the images.

### Dataset Structure

# Display the dataset info  
dataset['train'].features.keys()

dict\_keys(['image', 'image\_id', 'lesion\_id', 'dx', 'dx\_type', 'age', 'sex', 'localization'])

### Features Description and Insights:

* image: Image (dtype: PIL.Image.Image) - The image of the skin lesion in RGB format. Probably the most important feature for the classification task. The images have been preprocessed to:
  + Remove light's reflection on the skin, hair, air bubbles, pen markings (commonly used to outline the lesion), ruler markings, and other artifacts.
  + Normalization of the color space to account for different lighting conditions or camera characteristics.
  + Focus on the region of interest (the lesion) by cropping the image to the bounding box of the lesion and removing excess of healthy skin around it.
  + Remove image metadata (e.g., EXIF data) to ensure privacy and data protection.

# Display the first image in the dataset  
print(f'Image ID: {dataset["train"]["image\_id"][0]}')  
image = dataset['train'][0]['image']  
image\_metadata = {  
 'Format': image.format,  
 'Mode': image.mode,  
 'Size': image.size  
}  
from PIL import ExifTags  
exif\_data = image.\_getexif()  
if exif\_data:  
 exif\_data = {  
 ExifTags.TAGS[k]: v  
 for k, v in exif\_data.items()  
 if k in ExifTags.TAGS  
 }  
 image\_metadata['Exif'] = exif\_data  
else:  
 image\_metadata['Exif'] = None  
   
print(image\_metadata)  
image.show()

Image ID: ISIC\_0024329  
{'Format': 'JPEG', 'Mode': 'RGB', 'Size': (600, 450), 'Exif': None}



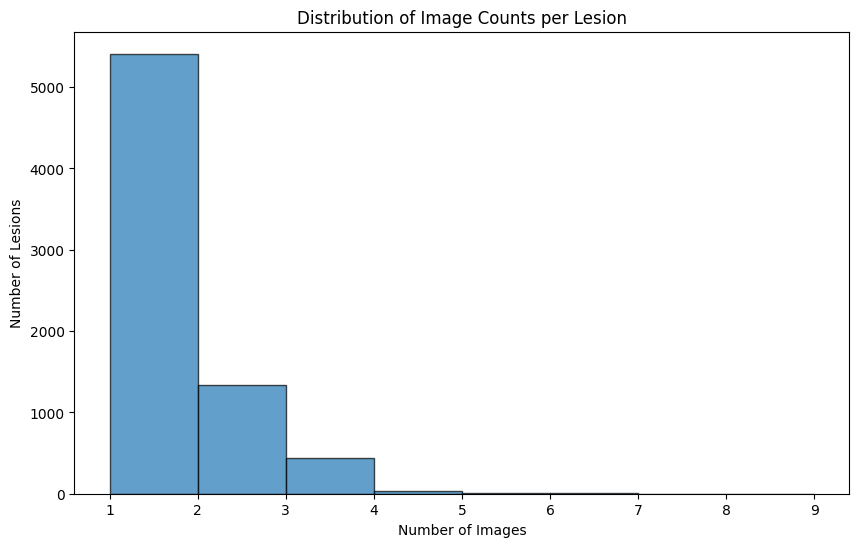
The Exif data is not available for the images, which is expected since the images have been preprocessed to remove metadata for privacy and data protection. The images are in RGB format and have been preprocessed to remove artifacts, normalize the color space, and focus on the region of interest (the lesion).

Then there are the ids of the image and the lesion, which can be used to link the images to the corresponding lesions. Some of the lesions have multiple images associated with them.

* image\_id: string - The unique identifier of the image
* lesion\_id: string - The unique identifier of the lesion

count\_images = pd\_dataset.groupby('lesion\_id')['image\_id'].count()  
print(f'Total number of lesions: {len(count\_images)}')  
print(f'Total images with multiple images: {count\_images[count\_images > 1].count()}')  
print(f'Total images with single image: {count\_images[count\_images == 1].count()}')  
plt.figure(figsize=(10, 6))  
plt.hist(count\_images, bins=range(1, 10), edgecolor='black', alpha=0.7)  
plt.title('Distribution of Image Counts per Lesion')  
plt.xlabel('Number of Images')  
plt.ylabel('Number of Lesions')  
plt.show()

Total number of lesions: 7226  
Total images with multiple images: 1816  
Total images with single image: 5410

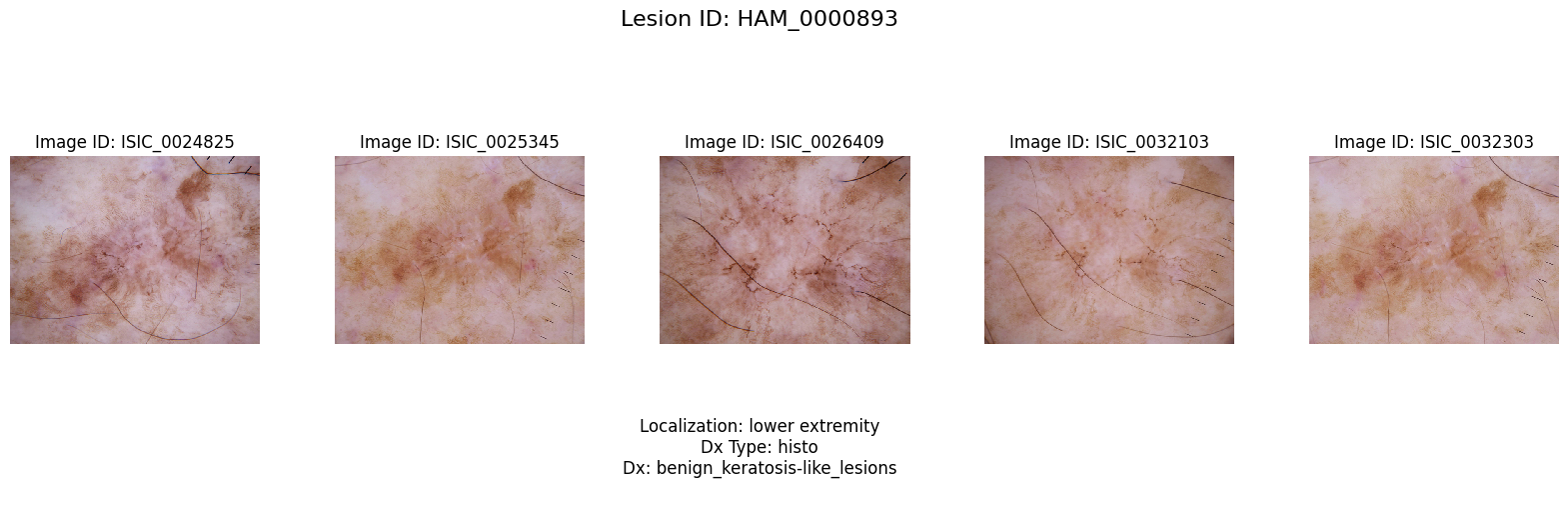


multiple\_images\_df = count\_images.value\_counts().reset\_index()  
multiple\_images\_df.columns = ['Image counts', 'Number of Images']  
multiple\_images\_df

Image counts Number of Images  
0 1 5410  
1 2 1332  
2 3 445  
3 4 30  
4 5 6  
5 6 3

The dataset contains 7226 unique lesions, with 5410 lesions having a single image and 1816 lesions having multiple images. The distribution of image counts per lesion shows that most lesions have a single image, while some lesions have multiple images. The distribution of image counts per lesion is skewed towards lesions with a single image, with a few lesions having multiple images. This distribution could affect the model's performance, as the model may have more data for lesions with multiple images compared to lesions with a single image.

lesion\_id\_five\_images = count\_images[count\_images == 5].index[0]  
lesion\_images = dataset['train'].filter(lambda x: x['lesion\_id'] == lesion\_id\_five\_images)  
lesion\_images\_list = list(  
 lesion\_images) # Since we'll be iterating over lesion\_images multiple times, convert it to list  
  
fig, axes = plt.subplots(nrows=1, ncols=5, figsize=(20, 5)) # 5 columns for 5 images  
  
# Display images along with image id as title for each subplot  
for ax, image in zip(axes, lesion\_images\_list):  
 # Display Image  
 ax.imshow(image['image'], interpolation='nearest')  
 ax.axis('off')  
 # Display image id as title  
 ax.set\_title(f"Image ID: {image['image\_id']}")  
  
# Set super title for all subplots  
plt.suptitle(f"Lesion ID: {lesion\_id\_five\_images}", fontsize=16)  
  
# Display common metadata at bottom  
metadata\_text = f"Localization: {lesion\_images\_list[0]['localization']}\n" \  
 f"Dx Type: {lesion\_images\_list[0]['dx\_type']}\n" \  
 f"Dx: {lesion\_images\_list[0]['dx']}\n"  
plt.figtext(0.5, 0.01, metadata\_text, ha="center", fontsize=12)  
  
plt.subplots\_adjust(wspace=0.3, hspace=0.3, top=0.85, bottom=0.15) # Adjust spaces and marginsolor to white  
plt.show()



# Display the unique values of the 'dx' column  
conditions = pd\_dataset['dx'].unique()  
print(f'Unique conditions: {conditions}')

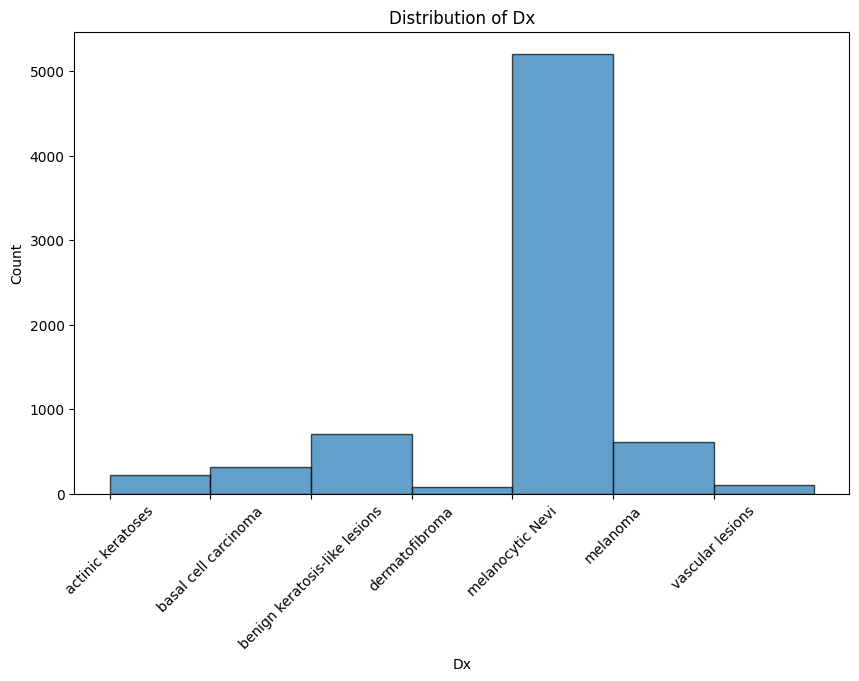
Unique conditions: ['actinic\_keratoses' 'basal\_cell\_carcinoma'  
 'benign\_keratosis-like\_lesions' 'dermatofibroma' 'melanocytic\_Nevi'  
 'melanoma' 'vascular\_lesions']

* dx: string - The diagnosis of the lesion. This will serve as the target variable for the classification task. The possible values are:
  + actinic keratoses: A common skin condition that is characterized by rough, scaly patches on the skin. It is considered a pre-cancerous condition.
  + basal cell carcinoma: A type of skin cancer that begins in the basal cells of the skin. It is the most common type of skin cancer.
  + benign keratosis: A non-cancerous skin condition that is characterized by rough, scaly patches on the skin. It is considered a pre-cancerous condition.
  + dermatofibroma: A benign skin lesion that is characterized by a hard bump under the skin. It is usually brown or red color.
  + melanocytic nevi: Commonly known as moles, these are benign skin lesions that are usually brown or black in color. They can vary in size and shape. Often considered as a risk factor for melanoma.
  + melanoma: A type of skin cancer that begins in the melanocytes, the cells that produce melanin. It is the most serious type of skin cancer.
  + vascular lesions: Skin lesions that are caused by abnormalities in the blood vessels. They can appear as red or purple spots on the skin.

# Subset data  
dx\_conditions = pd\_dataset['dx'].unique()  
  
  
# Define a function that will run before each image is loaded.  
def check\_condition(condition):  
 def \_check\_condition(example):  
 return example['dx'] == condition  
  
 return \_check\_condition

# List to store the subset images  
# subset\_images = []  
# for condition in dx\_conditions:  
# # Filter dataset for the current condition  
# condition\_dataset = dataset['train'].filter(check\_condition(condition), load\_from\_cache\_file=False)  
# subset\_images.append(condition\_dataset[0]['image'])  
#   
# # Calculate number of rows and columns for the subplot grid  
# n\_rows = 2  
# n\_cols = math.ceil(len(dx\_conditions) / n\_rows)  
#   
# fig, axes = plt.subplots(nrows=n\_rows, ncols=n\_cols, figsize=(20, 10)) # Update to (20, 10) for larger image  
# fig.patch.set\_facecolor('white') # Set figure background to white  
#   
# # Display an image for each condition  
# for ax, image, condition in zip(axes.flatten(), subset\_images, dx\_conditions):  
# ax.imshow(image, interpolation='nearest')  
# ax.axis('off')  
# ax.set\_title(f"Dx: {condition}")  
# ax.set\_facecolor('white') # Set axes background to white  
#   
# # Remove any empty subplots  
# for idx in range(len(dx\_conditions), n\_rows \* n\_cols):  
# fig.delaxes(axes.flatten()[idx])  
#   
# plt.suptitle('Sample Images for Each Condition', fontsize=16)  
# plt.subplots\_adjust(wspace=0.3, hspace=0.3)  
# plt.show()

plt.figure(figsize=(10, 6))  
plt.hist(pd\_dataset.drop\_duplicates('lesion\_id')['dx'].str.replace('\_', ' '), bins=range(8), edgecolor='black',  
 alpha=0.7)  
plt.title('Distribution of Dx')  
plt.xlabel('Dx')  
plt.ylabel('Count')  
plt.xticks(rotation=45, ha='center')  
plt.show()



The distribution of the diagnosis labels is imbalanced, with some conditions having more samples than others. This imbalance could affect the model's performance, as the model may be biased towards the majority class. Therefore, it is important to consider strategies to address the class imbalance during the model training process, such as oversampling, under sampling, or using class weights.

# Table of counts of each condition  
dx\_counts = pd\_dataset['dx'].value\_counts().reset\_index()  
dx\_counts.columns = ['Condition', 'Count']  
dx\_counts['Condition'] = dx\_counts['Condition'].str.replace('\_', ' ')  
dx\_counts['Percentage'] = (dx\_counts['Count'] / dx\_counts['Count'].sum()) \* 100  
dx\_counts

Condition Count Percentage  
0 melanocytic Nevi 6405 66.878981  
1 melanoma 1076 11.235251  
2 benign keratosis-like lesions 1048 10.942884  
3 basal cell carcinoma 487 5.085100  
4 actinic keratoses 315 3.289130  
5 vascular lesions 136 1.420069  
6 dermatofibroma 110 1.148585

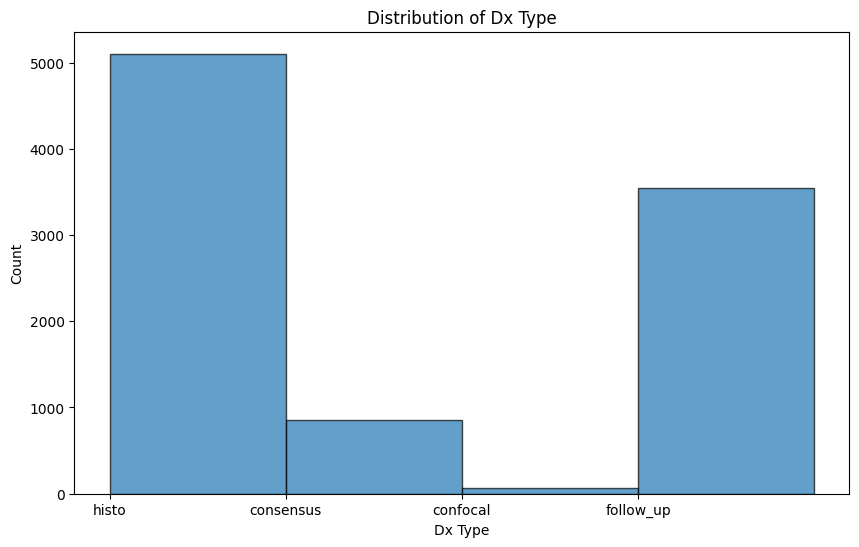
# Display the unique values of the 'dx\_type' column  
dx\_types = pd\_dataset['dx\_type'].unique()  
print(f'Unique Dx Types: {dx\_types}')

Unique Dx Types: ['histo' 'consensus' 'confocal' 'follow\_up']

* dx\_type: string - The type of the pathology diagnosis:
  + histopathology: The diagnosis was confirmed by histopathology, which involves the examination of tissue samples under a microscope.
  + follow-up: The diagnosis was confirmed by follow-up examination, which involves monitoring the lesion over time to see if it changes.
  + consensus: The diagnosis was confirmed by consensus, which involves the agreement of multiple experts on the diagnosis.
  + confocal: The diagnosis was confirmed by confocal microscopy, which involves the examination of skin lesions using a special microscope.

The 'dx\_type' column provides information on how the diagnosis was confirmed, since the main idea of the project is to predict the diagnosis of the lesion, it is likely that this information is not used as a feature for the classification task. However, it could be useful for evaluating the model's performance and understanding the dataset's characteristics.

# Display the histogram of the 'dx\_type' column  
plt.figure(figsize=(10, 6))  
plt.hist(pd\_dataset['dx\_type'], bins=range(5), edgecolor='black', alpha=0.7)  
plt.title('Distribution of Dx Type')  
plt.xlabel('Dx Type')  
plt.ylabel('Count')  
plt.show()

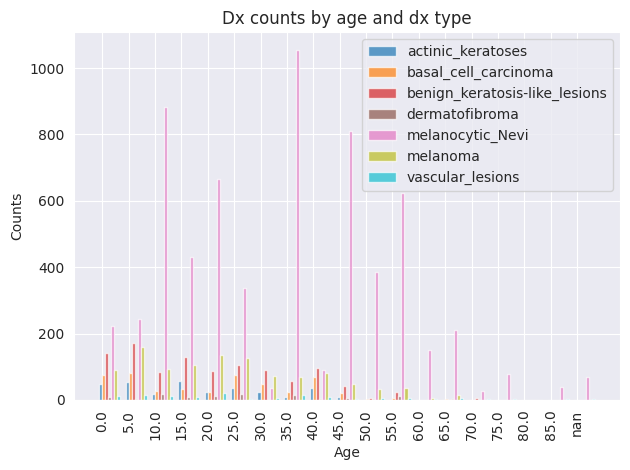


* age: int - Patient's age at the time of image acquisition, to the nearest multiple of 5 years in order to anonymize the data. This feature could be useful for predicting the diagnosis of the lesion, as certain skin conditions are more common in specific age. This could also be used to identify age-related patterns in the dataset and evaluate the model's performance across different age.

# Table of counts of each age  
age\_counts = pd\_dataset['age'].value\_counts().reset\_index()  
age\_counts.columns = ['Age', 'Count']  
age\_counts = age\_counts.sort\_values('Age')  
age\_counts = age\_counts.reset\_index(drop=True)  
age\_counts

Age Count  
0 0.0 34  
1 5.0 84  
2 10.0 40  
3 15.0 74  
4 20.0 161  
5 25.0 235  
6 30.0 440  
7 35.0 707  
8 40.0 941  
9 45.0 1247  
10 50.0 1131  
11 55.0 971  
12 60.0 774  
13 65.0 701  
14 70.0 731  
15 75.0 595  
16 80.0 384  
17 85.0 272

unique\_ages = pd\_dataset['age'].unique()  
unique\_dx = pd\_dataset['dx'].unique()  
  
new\_grouped\_data = []  
  
grouped = pd\_dataset.groupby(['dx', 'age']).size().reset\_index(name='counts')  
  
for dx in unique\_dx:  
 for age in unique\_ages:  
 if len(grouped[(grouped['dx'] == dx) & (grouped['age'] == age)]) > 0:  
 # If there's entry for this 'dx' at this age, use that count  
 new\_grouped\_data.append(  
 (dx, age, grouped[(grouped['dx'] == dx) & (grouped['age'] == age)]['counts'].values[0]))  
 else:  
 # If there's no entry for this 'dx' at this age, use 0  
 new\_grouped\_data.append((dx, age, 0))  
  
# Create a new DataFrame from our new grouped data  
grouped = pd.DataFrame(new\_grouped\_data, columns=['dx', 'age', 'counts'])  
  
dx\_types = grouped['dx'].unique()  
dx\_colors = {dx\_type: plt.cm.tab10(i / float(len(dx\_types) - 1)) for i, dx\_type in enumerate(dx\_types)}  
  
age\_unique\_values = sorted(grouped['age'].unique())  
index = np.arange(len(age\_unique\_values))  
  
bar\_width = 0.8 / len(dx\_types)  
opacity = 0.7  
  
fig, ax = plt.subplots()  
  
for i, dx\_type in enumerate(dx\_types):  
 if dx\_type == 'melanocytic\_nevi':  
 continue  
 counts = grouped[grouped['dx'] == dx\_type]['counts']  
 rects = ax.bar(index + i \* bar\_width, counts, bar\_width, alpha=opacity, color=dx\_colors[dx\_type], label=dx\_type)  
  
ax.set\_xlabel('Age')  
ax.set\_ylabel('Counts')  
ax.set\_title('Dx counts by age and dx type')  
# set labels perpendicularly  
plt.xticks(rotation=90)  
ax.set\_xticks(index + bar\_width / 2)  
ax.set\_xticklabels(age\_unique\_values)  
ax.legend()  
  
fig.tight\_layout()  
plt.show()



The age column contains 55 items with nan values. These values will need to be handled during the data preprocessing stage.

pd\_dataset['age'].isna().sum()

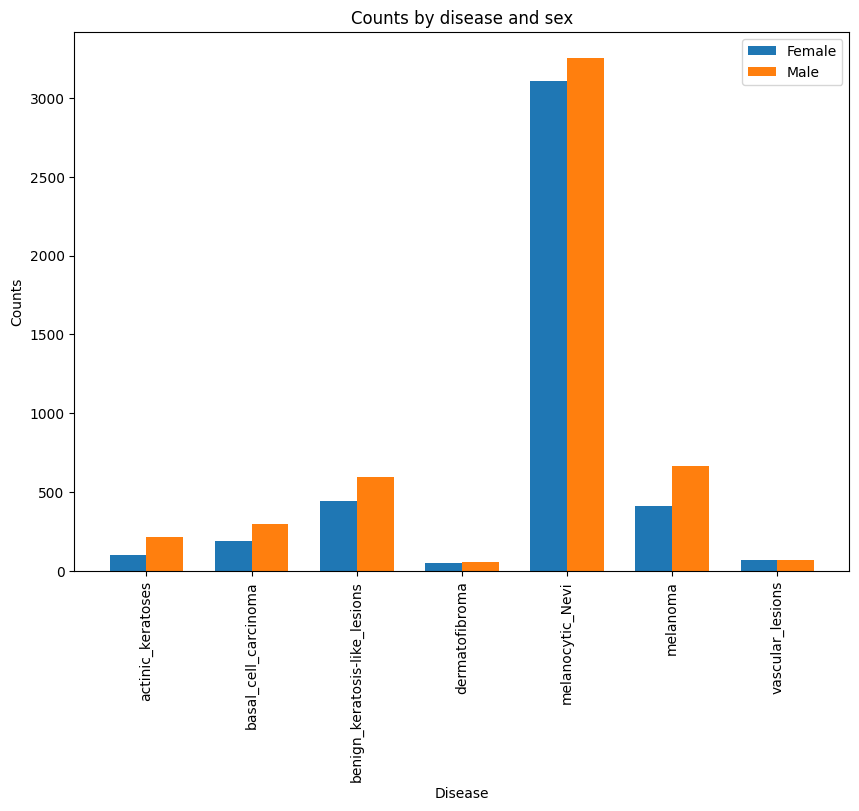
55

* sex: string - The patient's biological gender

# Display unique values  
print(f'Unique values {pd\_dataset["sex"].unique()}')

Unique values ['female' 'male' 'unknown']

grouped = pd\_dataset.groupby(['sex', 'dx']).size().reset\_index(name='counts')  
  
# Create a female and male dataframe  
df\_female = grouped[grouped['sex'] == 'female'].reset\_index()  
df\_male = grouped[grouped['sex'] == 'male'].reset\_index()  
  
# Ensure each `dx` dtype has a count for both sex  
unique\_dx = pd\_dataset['dx'].unique()  
  
for dx in unique\_dx:  
 if dx not in df\_female['dx'].unique():  
 df\_female = df\_female.append({'sex': 'female', 'dx': dx, 'counts': 0}, ignore\_index=True)  
 if dx not in df\_male['dx'].unique():  
 df\_male = df\_male.append({'sex': 'male', 'dx': dx, 'counts': 0}, ignore\_index=True)  
  
# Sort by 'dx' and reset index  
df\_female = df\_female.sort\_values('dx').reset\_index(drop=True)  
df\_male = df\_male.sort\_values('dx').reset\_index(drop=True)  
  
# Now the rest of the code remains same except we use 'df\_female' and 'df\_male' instead of 'grouped'  
bar\_width = 0.35  
index = np.arange(len(df\_female))  
  
fig, ax = plt.subplots(figsize=(10, 7))  
  
female\_bar = ax.bar(index, df\_female['counts'], bar\_width, label='Female')  
male\_bar = ax.bar(index + bar\_width, df\_male['counts'], bar\_width, label='Male')  
  
ax.set\_xlabel('Disease')  
ax.set\_ylabel('Counts')  
ax.set\_title('Counts by disease and sex')  
ax.set\_xticks(index + bar\_width / 2)  
ax.set\_xticklabels(df\_female['dx'], rotation=90)  
ax.legend()  
  
plt.show()



# Table conditions by sex group  
pivot\_table = pd\_dataset.pivot\_table(index='dx', columns='sex', aggfunc='size', fill\_value=0)  
  
print(pivot\_table)

sex female male unknown  
dx   
actinic\_keratoses 102 213 0  
basal\_cell\_carcinoma 188 299 0  
benign\_keratosis-like\_lesions 443 595 10  
dermatofibroma 50 60 0  
melanocytic\_Nevi 3104 3255 46  
melanoma 411 665 0  
vascular\_lesions 68 68 0

The distribution of the diagnosis labels seems to be skewed towards male patients. The unknown values are not removed from the dataset, since the image and lesion ids are still useful for the classification task. The unknown values will be handled during the data preprocessing stage.

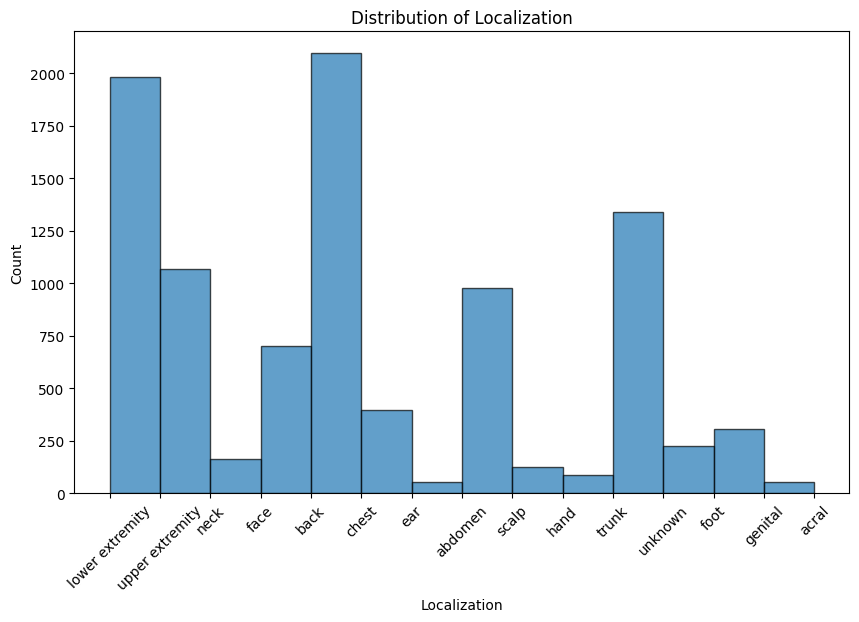
* localization: string - The localization of the lesion on the body. Which could be useful for predicting the diagnosis of the lesion, as certain skin conditions are more common in specific body locations. This could also be used to identify body location-related patterns in the dataset and evaluate the model's performance across different body locations.

# Print unique values  
print(f'Unique values {pd\_dataset["localization"].unique()}')

Unique values ['lower extremity' 'upper extremity' 'neck' 'face' 'back' 'chest' 'ear'  
 'abdomen' 'scalp' 'hand' 'trunk' 'unknown' 'foot' 'genital' 'acral']

The localization column contains 15 unique values, which represent the different body locations where the lesions are located. The localization feature could be useful for predicting the diagnosis of the lesion, as certain skin conditions are more common in specific body locations. This could also be used to identify body location-related patterns in the dataset and evaluate the model's performance across different body locations.

# Display the histogram of the 'localization' column  
plt.figure(figsize=(10, 6))  
plt.hist(pd\_dataset['localization'], bins=range(15), edgecolor='black', alpha=0.7)  
plt.title('Distribution of Localization')  
plt.xlabel('Localization')  
plt.ylabel('Count')  
plt.xticks(rotation=45, ha='center')  
plt.show()



# Table of counts of each localization  
localization\_counts = pd\_dataset['localization'].value\_counts().reset\_index()  
localization\_counts.columns = ['Localization', 'Count']  
localization\_counts

Localization Count  
0 back 2095  
1 lower extremity 1984  
2 trunk 1340  
3 upper extremity 1068  
4 abdomen 976  
5 face 703  
6 chest 395  
7 foot 307  
8 unknown 228  
9 neck 163  
10 scalp 125  
11 hand 86  
12 ear 54  
13 genital 46  
14 acral 7

### Summary of EDA Insights

pd\_dataset.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 9577 entries, 0 to 9576  
Data columns (total 8 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 image 9577 non-null object   
 1 image\_id 9577 non-null object   
 2 lesion\_id 9577 non-null object   
 3 dx 9577 non-null object   
 4 dx\_type 9577 non-null object   
 5 age 9522 non-null float64  
 6 sex 9577 non-null object   
 7 localization 9577 non-null object   
dtypes: float64(1), object(7)  
memory usage: 598.7+ KB

The exploratory data analysis (EDA) of the HAM10000 dataset has provided valuable insights into the dataset's structure, distribution of labels, and characteristics of the images. The key insights from the EDA are as follows:

* The dataset contains 10015 dermatoscopic images of common pigmented skin lesions, divided into training, validation, and test sets.
  + Some lesions have multiple images associated with them, while others have a single image; ranging from one up to seven images per lesion. The distribution of image counts per lesion is skewed towards lesions with a single image, with a few lesions having multiple images.
* The images are in RGB format and have been preprocessed to remove artifacts, normalize the color space, and focus on the region of interest (the lesion).
* The dataset contains seven different types of skin lesions: melanocytic nevi, melanoma, basal cell carcinoma, actinic keratosis, benign keratosis, dermatofibroma, and vascular lesions.
* The distribution of the diagnosis labels is imbalanced, with some conditions having more samples than others. This imbalance could affect the model's performance, and strategies to address the class imbalance will be considered during the model training process.
  + The most common condition is melanocytic nevi, followed by melanoma and basal cell carcinoma.
  + The least common conditions are vascular lesions and dermatofibroma.
* The dataset contains information about the patient's age, biological sex, and the localization of the lesion on the body. These features could be useful for predicting the diagnosis of the lesion and evaluating the model's performance across different age.
* Some of the features contain unknown values, such as nan values in the age column and unknown values in the localization and sex column. These unknown values will need to be handled during the data preprocessing stage.
* The localization feature contains 15 unique values, representing the different body locations where the lesions are located. This feature could be useful for predicting the diagnosis of the lesion and evaluating the model's performance across different body locations.

## Data Preparation

The data preparation stage involves the comprehensive cleaning, transformation, categorization, and normalization of numerical and categorical data. This stage is crucial to ensure the data is in a suitable format for model training. In this section, we will perform the data preparation steps for the HAM10000 dataset, including handling missing values, encoding categorical features, and splitting the dataset into training, validation, and test sets.

### Feature Selection

The project would combine image with the provided metadata to predict the diagnosis of the lesion. The image feature would be the primary feature for the classification task, as it contains the visual information of the skin lesion. The metadata features, such age, sex, and localization, would be used as additional features to enhance the model's performance and interpretability. The target variable for the classification task would be the 'dx' column, which contains the diagnosis of the lesion.

# Select the relevant features for the classification task  
selected\_features = ['image', 'age', 'sex', 'localization', 'dx']  
train\_ds = pd\_dataset[selected\_features]  
val\_ds = val\_dataset[selected\_features]  
test\_ds = test\_dataset[selected\_features]

Due to missing values in the 'age' column, we will fill these missing values with the median age value in both the training and validation datasets. The test dataset will be used for the final evaluation of the model and will not be used during the training process, so missing values in the test dataset will not be used for data preparation.

train\_val\_dataset = pd.concat([pd\_dataset, val\_dataset], ignore\_index=True)  
train\_val\_dataset.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 12069 entries, 0 to 12068  
Data columns (total 8 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 image 12069 non-null object   
 1 image\_id 12069 non-null object   
 2 lesion\_id 12069 non-null object   
 3 dx 12069 non-null object   
 4 dx\_type 12069 non-null object   
 5 age 12000 non-null float64  
 6 sex 12069 non-null object   
 7 localization 12069 non-null object   
dtypes: float64(1), object(7)  
memory usage: 754.4+ KB

### Handling Missing Values

# Check for missing values  
missing\_values = train\_val\_dataset.isnull().sum()  
missing\_values

image 0  
image\_id 0  
lesion\_id 0  
dx 0  
dx\_type 0  
age 69  
sex 0  
localization 0  
dtype: int64

The age column contains 55 missing values on training and 14 on validation. We will fill these missing values with the median age value in both the training and validation datasets which is 50.

train\_val\_dataset['age'].median()

50.0

# Fill missing values in the 'age' column with the median age  
train\_val\_dataset['age'] = train\_val\_dataset['age'].fillna(train\_val\_dataset['age'].median())  
# Check for missing values after filling  
missing\_values = train\_val\_dataset.isnull().sum()  
missing\_values

image 0  
image\_id 0  
lesion\_id 0  
dx 0  
dx\_type 0  
age 0  
sex 0  
localization 0  
dtype: int64

The missing values in the 'age' column have been successfully filled with the median age value. Next, we will encode the categorical features 'dx',

### Encoding Categorical Features

The categorical features are sex, localization, and dx. We will encode these categorical features using one-hot encoding to convert them into numerical format for the model training process. The 'dx' column will be the target variable for the classification task, and the other categorical features will be used as additional features for the model.

First, we will encode the 'dx' column, which contains the diagnosis of the lesion. We will use one-hot encoding to convert the 'dx' column into numerical format.

# One-hot encode the 'dx' column  
train\_ds = pd.get\_dummies(train\_ds, columns=['dx'], prefix='dx')  
val\_ds = pd.get\_dummies(val\_ds, columns=['dx'], prefix='dx')  
test\_ds = pd.get\_dummies(test\_ds, columns=['dx'], prefix='dx')  
train\_val\_dataset.head()

image image\_id \  
0 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... ISIC\_0024329   
1 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... ISIC\_0024372   
2 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... ISIC\_0024418   
3 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... ISIC\_0024450   
4 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... ISIC\_0024463   
  
 lesion\_id dx dx\_type age sex localization   
0 HAM\_0002954 actinic\_keratoses histo 75.0 female lower extremity   
1 HAM\_0005389 actinic\_keratoses histo 70.0 male lower extremity   
2 HAM\_0003380 actinic\_keratoses histo 75.0 female lower extremity   
3 HAM\_0005505 actinic\_keratoses histo 50.0 male upper extremity   
4 HAM\_0004568 actinic\_keratoses histo 50.0 male upper extremity

Next, we will encode the localization. The localization column contains the body location where the lesion is located. We will use one-hot encoding to convert the localization column into numerical format. It is important to note that the 'localization' column contains 15 unique values, which represent the different body locations where the lesions are located. However, one of the values is 'unknown'.

# Count unknown values  
train\_val\_dataset['localization'].value\_counts()

localization  
back 2653  
lower extremity 2479  
trunk 1691  
upper extremity 1354  
abdomen 1231  
face 896  
chest 475  
foot 386  
unknown 289  
neck 202  
scalp 169  
hand 110  
ear 69  
genital 58  
acral 7  
Name: count, dtype: int64

After consideration of the 'unknown' values, the project decided to maintain it as a separate category, since this This could reflect a scenario where the clinician could provide a diagnosis without knowing the exact location of the lesion. Therefore, we will encode the 'localization' column using one-hot encoding, with 'unknown' as a separate category.

# One-hot encode the 'localization' column  
train\_ds = pd.get\_dummies(train\_ds, columns=['localization'], prefix='loc')  
val\_ds = pd.get\_dummies(val\_ds, columns=['localization'], prefix='loc')  
test\_ds = pd.get\_dummies(test\_ds, columns=['localization'], prefix='loc')  
train\_ds.head()

image age sex \  
0 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... 75.0 female   
1 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... 70.0 male   
2 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... 75.0 female   
3 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... 50.0 male   
4 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... 50.0 male   
  
 dx\_actinic\_keratoses dx\_basal\_cell\_carcinoma \  
0 True False   
1 True False   
2 True False   
3 True False   
4 True False   
  
 dx\_benign\_keratosis-like\_lesions dx\_dermatofibroma dx\_melanocytic\_Nevi \  
0 False False False   
1 False False False   
2 False False False   
3 False False False   
4 False False False   
  
 dx\_melanoma dx\_vascular\_lesions ... loc\_face loc\_foot loc\_genital \  
0 False False ... False False False   
1 False False ... False False False   
2 False False ... False False False   
3 False False ... False False False   
4 False False ... False False False   
  
 loc\_hand loc\_lower extremity loc\_neck loc\_scalp loc\_trunk loc\_unknown \  
0 False True False False False False   
1 False True False False False False   
2 False True False False False False   
3 False False False False False False   
4 False False False False False False   
  
 loc\_upper extremity   
0 False   
1 False   
2 False   
3 True   
4 True   
  
[5 rows x 25 columns]

Finally, we will encode the 'sex' column, which contains the biological gender of the patient. For simplicity, we will use binary encoding to convert these categorical features into numerical format.

# Show unknown values from sex column  
train\_val\_dataset['sex'].value\_counts()

sex  
male 6494  
female 5506  
unknown 69  
Name: count, dtype: int64

The 'unknown' values will be set to 0, since removing the images with unknown values would reduce the dataset size and potentially remove valuable information. This decision is based on the assumption that the 'unknown' values are missing at random and do not introduce bias into the dataset, while the information in the image and lesion ids is still valuable for the classification task.

# Binary encode  
def binary\_encode(df, column, positive\_value):  
 df[column] = df[column].apply(lambda x: 1 if x == positive\_value else 0)  
 return df  
  
# Binary encode  
train\_ds = binary\_encode(train\_ds, 'sex', 'male')  
val\_ds = binary\_encode(train\_ds, 'sex', 'male')  
test\_ds = binary\_encode(train\_ds, 'sex', 'male')  
train\_ds.head()

image age sex \  
0 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... 75.0 0   
1 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... 70.0 0   
2 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... 75.0 0   
3 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... 50.0 0   
4 {'bytes': b'\xff\xd8\xff\xdb\x00C\x00\x01\x01\... 50.0 0   
  
 dx\_actinic\_keratoses dx\_basal\_cell\_carcinoma \  
0 True False   
1 True False   
2 True False   
3 True False   
4 True False   
  
 dx\_benign\_keratosis-like\_lesions dx\_dermatofibroma dx\_melanocytic\_Nevi \  
0 False False False   
1 False False False   
2 False False False   
3 False False False   
4 False False False   
  
 dx\_melanoma dx\_vascular\_lesions ... loc\_face loc\_foot loc\_genital \  
0 False False ... False False False   
1 False False ... False False False   
2 False False ... False False False   
3 False False ... False False False   
4 False False ... False False False   
  
 loc\_hand loc\_lower extremity loc\_neck loc\_scalp loc\_trunk loc\_unknown \  
0 False True False False False False   
1 False True False False False False   
2 False True False False False False   
3 False False False False False False   
4 False False False False False False   
  
 loc\_upper extremity   
0 False   
1 False   
2 False   
3 True   
4 True   
  
[5 rows x 25 columns]

The data preparation stage has successfully cleaned, transformed, and encoded the numerical and categorical features of the HAM10000 dataset. The missing values in the 'age' column have been filled with the median age value, and the categorical features 'dx', 'localization', and 'sex' have been encoded using one-hot encoding and binary encoding.

## Image Preprocessing

The image preprocessing stage involves the normalization, resizing, and augmentation of the images to prepare them for the model training process. In this section, we will perform the image preprocessing steps for the HAM10000 dataset, including loading the images, resizing them to a standard size, normalizing the pixel values, and applying data augmentation techniques.

### Load and Resize Images

The images in the HAM10000 dataset are in RGB format and have been preprocessed to remove artifacts, normalize the color space, and focus on the region of interest (the lesion). Typically, the images are of different sizes and resolutions, which can affect the model's performance. The goal of this step is to load the images, resize them to a standard size, and normalize the pixel values to prepare them for the model training process.

The images in the HAM10000 dataset have been successfully loaded, and the unique image sizes in the training, validation, and test datasets have been displayed. The images are of different sizes and resolutions, which can affect the model's performance. Typically, deep learning models require images of a standard size for training. However, the resize of the image is done later in the pipeline, after the data augmentation step, to avoid losing valuable information during the resizing process.

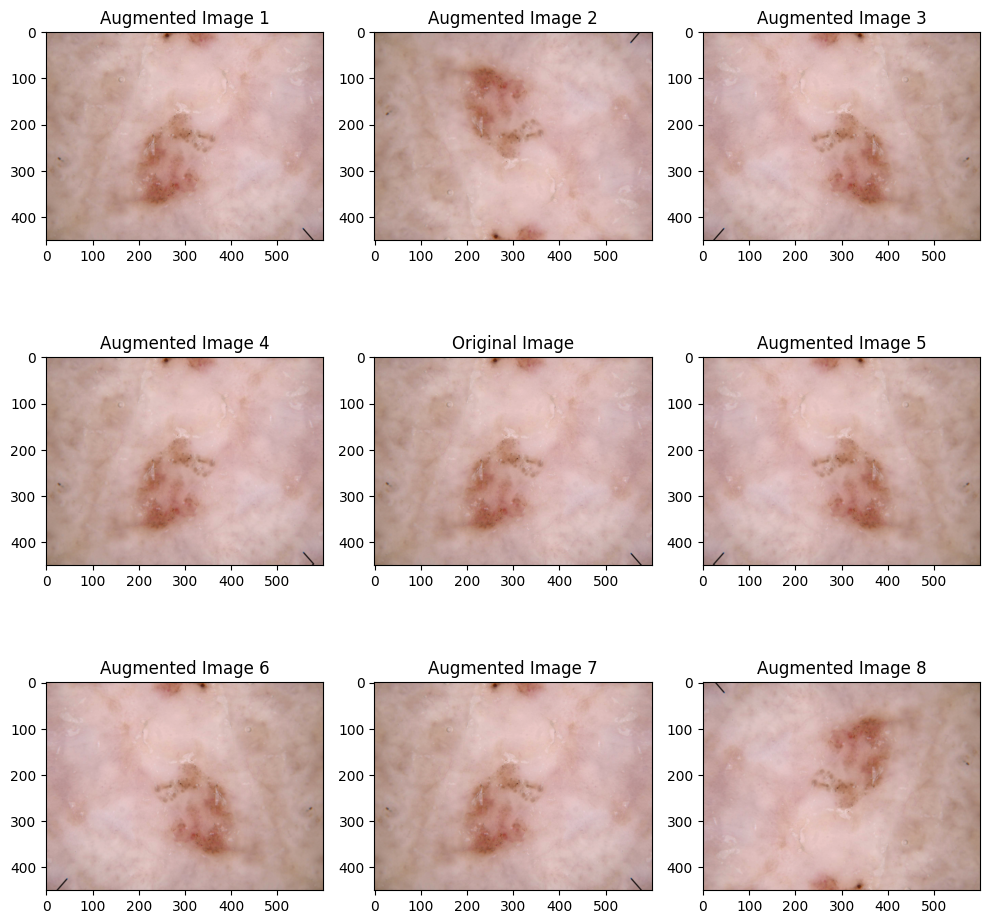
### Data Augmentation

Data augmentation is a technique used to artificially increase the size of the training dataset by applying random transformations to the images, such as rotation, flipping, zooming, and shifting. Data augmentation helps to improve the model's generalization and robustness by exposing it to a wider variety of training examples.

The data augmentation techniques are applied using tensor operations, which are more efficient and faster than applying the transformations directly to the images. For demonstration purposes, the augmentation of a sample image from the training dataset will be displayed to show the effect of the data augmentation techniques. The data augmentation techniques include rotation, width shift, height shift, and horizontal flip.

* Rotation Range: ~20 degrees (pi/8 radians) - The image is rotated by a random angle within the specified range, which helps the model learn to recognize the lesion from different perspectives. To avoid anomalies, the fill mode is set to 'reflect', which reflects the pixels at the edges of the image. The fill mode as 'reflect' is used to avoid anomalies in the image after rotation, however it works with small rotations. For that reason, the rotation range is set to 20 degrees, which reflects the pixels at the edges of the image does not introduce artifacts. Rotation that result in a rhombus shape will be avoided, to prevent the model from learning incorrect patterns, especially due to the reflection of the pixels at the edges of the image.
* Horizontal and Vertical Flip: The image is flipped horizontally or vertically with a probability of 0.5. The flipping of the image introduce variation in the dataset, without changing what the image represents. This helps the model learn to recognize the lesion from different orientations.
* 90 degrees rotation: The image is rotated by 90 degrees. This is done to mitigate the limitation of the rotation range, which may not cover all possible orientations of the lesion. This rotation does not add padding to the image and since it is apply after resizing the image to a square shape, it does not introduce artifacts.
* Adaptive Resize: The image is resized to a square shape of 256x256 pixels, this size is standard and commonly use during training of deep learning model. This is done to ensure that all images have a consistent size for the model training process. The adaptive resize is used to resize the image to a square shape, without distorting the aspect ratio of the lesion. This is important to maintain the integrity of the lesion's shape and features during the resizing process.

# Initialize the ImageDataGenerator  
datagen = ImageDataGenerator(rotation\_range=math.pi / 8,   
 vertical\_flip=True,  
 horizontal\_flip=True,   
 fill\_mode='reflect')  
  
# Define the image index  
img\_dict = train\_ds['image'][0] # Replace 0 with the index of the image you want  
image\_array = np.array(Image.open(io.BytesIO(img\_dict['bytes'])))  
image\_array = image\_array.astype('float32') / 255 # normalizing pixel values  
  
# If image is grayscale convert it to 3 channels  
if len(image\_array.shape) == 2:  
 image\_array = np.stack((image\_array,) \* 3, axis=-1)  
  
# New: Expand dims to add a 'batch' dimension  
image\_array = np.expand\_dims(image\_array, axis=0)  
  
# Fit the ImageDataGenerator instance  
datagen.fit(image\_array)  
  
# Create the figure  
fig, axs = plt.subplots(3, 3, figsize=(10, 10)) # Creates a grid of 3 x 3  
  
# Put the original image at the center  
axs[1, 1].imshow(np.clip(image\_array[0], 0, 1))  
axs[1, 1].set\_title('Original Image')  
  
# Generator to create images  
generator = datagen.flow(image\_array, batch\_size=1)  
  
# Position of next image  
positions = [(i, j) for i in range(3) for j in range(3) if not (i == 1 and j == 1)]  
pos\_index = 0  
  
# Generate augmented images  
for i, new\_images in enumerate(generator):  
 if i == 8: # We already have our original image, so we only need 8 more  
 break  
 row, col = positions[pos\_index]  
 pos\_index += 1  
 axs[row, col].imshow(np.clip(new\_images[0], 0, 1))  
 axs[row, col].set\_title(f"Augmented Image {i + 1}")  
  
# Show the figure   
plt.tight\_layout()  
plt.show()



The data augmentation techniques have been successfully applied to a sample image from the training dataset, demonstrating the effect of rotation, flipping, and resizing on the image. The data augmentation techniques help to increase the diversity of the training dataset and improve the model's generalization and robustness.

### Image Preprocessing Pipeline

The image preprocessing pipeline involves loading the images, applying data augmentation techniques, resizing the images to a standard size, and normalizing the pixel values. The image preprocessing pipeline will be implemented using the TensorFlow tensor operations to ensure efficiency and scalability during the model training process. The image preprocessing pipeline will be integrated with the metadata features to create the input data for the model training process.

The data augmentation techniques will be applied to the images on the training dataset to increase the diversity of the training examples and improve the model's generalization. The validation and test datasets will not undergo data augmentation, as they are used for evaluation and inference, respectively. The augmentation will be applied during the training process, this is done to randomly generate new images during each epoch, which helps the model learn to recognize the lesion from different perspectives and orientations.

# Model Selection

The model selection stage involves choosing the appropriate deep learning architecture for the skin lesion classification task. We will evaluate the performance of these architectures on the HAM10000 dataset and select the best-performing model for the skin lesion classification task.

The selected models are as follows:

* VGG16: A deep convolutional neural network with 16 layers, known for its simplicity and effectiveness. The VGG16 model has been widely used for image classification tasks. Proposed by Simonyan and Zisserman in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" (2014).
* VGG19: A deep convolutional neural network with 19 layers, similar to VGG16 but with additional layers. The VGG19 model is known for its simplicity and effectiveness and has been used for image classification tasks. It was also, proposed by Simonyan and Zisserman in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" (2014), and it is a deeper version of the VGG16 model.
* ResNet101V2: A deep residual neural network with 101 layers, known for its ability to train very deep neural networks effectively. The ResNet101V2 model has shown good performance on skin lesion classification tasks. The ResNet101V2 model is based on the ResNet architecture proposed by He et al. in the paper "Deep Residual Learning for Image Recognition" (2015).
* InceptionResNetV2: A deep convolutional neural network that combines the Inception and ResNet architectures. The InceptionResNetV2 model is known for its efficiency and effectiveness in image classification tasks. The InceptionResNetV2 model is based on the Inception architecture proposed by Szegedy et al. in the paper "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning" (2016).
* Xception: A deep convolutional neural network that is based on the Inception architecture. The Xception model is known for its efficiency and effectiveness in image classification tasks. The Xception model is based on the Xception architecture proposed by Chollet in the paper "Xception: Deep Learning with Depthwise Separable Convolutions" (2017).
* MobileNetV2: A lightweight convolutional neural network that is optimized for mobile and embedded devices. The MobileNetV2 model is known for its efficiency and effectiveness in image classification tasks. The MobileNetV2 model is based on the MobileNet architecture proposed by Howard et al. in the paper "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications" (2017).

The model selected have been previously used as benchmark models for skin lesion classification tasks, however little to no hyperparameter optimization was done. The project will use Optuna to perform hyperparameter optimization for the selected model, to improve the model's performance and efficiency. The hyperparameter optimization will be done in an iterative manner, following the CRISP-DM methodology, to refine and optimize the model pipeline. These models have been trained on the ImageNet dataset, which is a large-scale dataset of natural images. The models have been fine-tuned on the skin lesion images in the HAM10000 dataset to adapt them to the skin lesion classification task. The models will be evaluated based on their performance metrics, such as accuracy, precision, recall, and F1 score, to select the best-performing model for the skin lesion classification task.

We will use the study done by Cassidy, B., Kendrick, C., Brodzicki, A., Jaworek-Korjakowska, J., & Yap, M. H. (2022). Analysis of the ISIC image datasets: Usage, benchmarks and recommendations. Medical Image Analysis, 75, 102305. <https://doi.org/10.1016/j.media.2021.102305> to guide the selection of the model. The study provides a comprehensive analysis of the ISIC image datasets, including the HAM10000 dataset, and evaluates the performance of various deep learning architectures on the skin lesion classification task. The study compares the performance of the models based on their architecture, number of parameters, training time, and performance metrics such as AUC. The study provides valuable insights into the performance of the models on the skin lesion classification task and helps guide the selection of the best-performing model for the project.

The study evaluates the performance of the following deep learning architectures on the skin lesion classification task using the HAM10000 dataset:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Method | Accuracy | Precision | Recall | F1 | AUC |
| EfficientNetB0 | 0.55 | 0.19 | 0.41 | 0.26 | 0.51 |
| InceptionResNetV2 | 0.40 | 0.20 | 0.67 | 0.30 | 0.49 |
| ResNet101V2 | 0.38 | 0.20 | 0.73 | 0.31 | 0.54 |
| VGG16 | 0.48 | 0.20 | 0.57 | 0.30 | 0.54 |
| VGG19 | 0.56 | 0.20 | 0.41 | 0.26 | 0.51 |
| Xception | 0.44 | 0.19 | 0.59 | 0.29 | 0.50 |

Even though MobileNetV2 was not evaluated in the study, it is included in the selection of models due to its efficiency and effectiveness in image classification tasks.

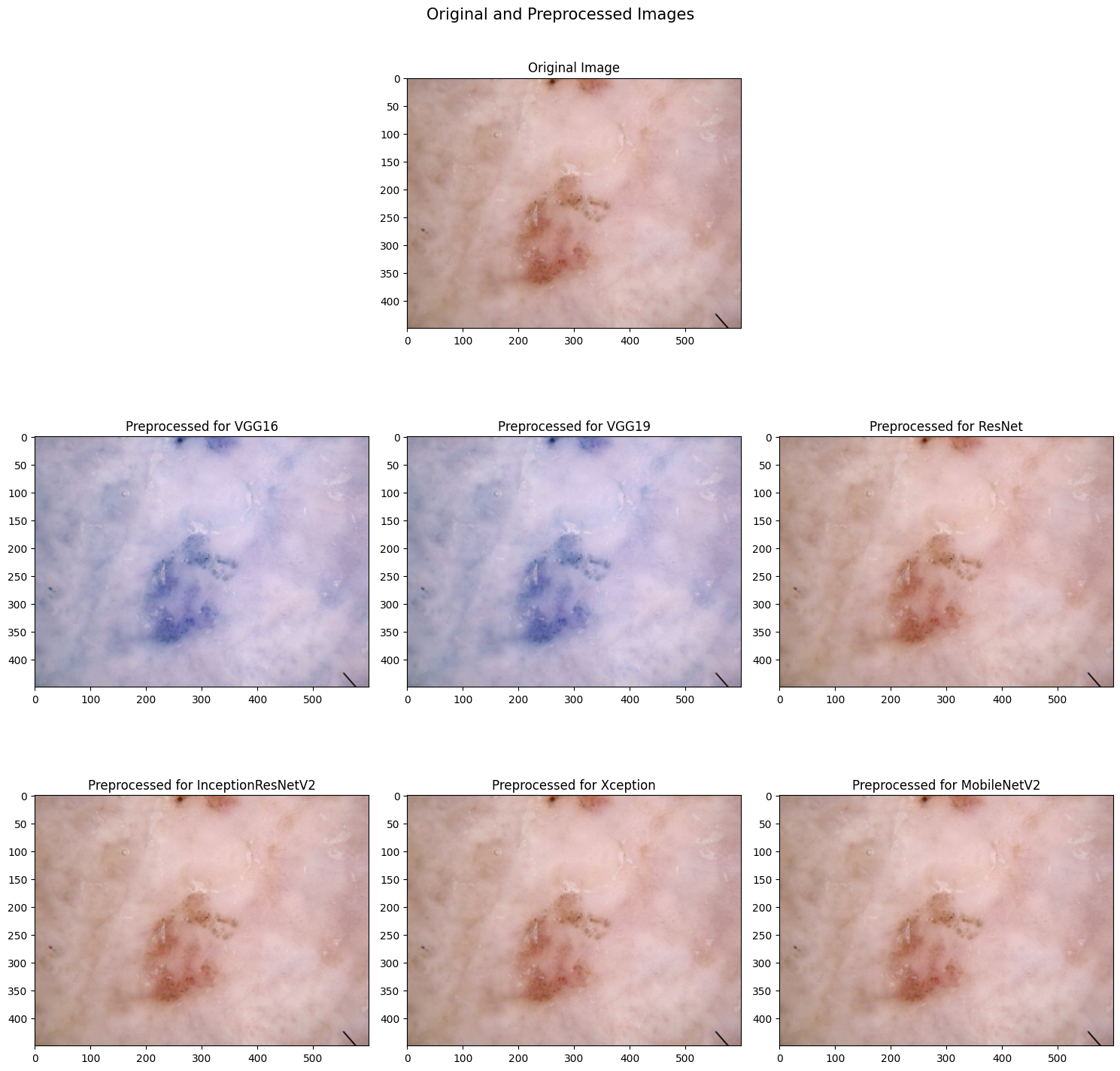
## The use of Preprocessing functions from Pretrained Models

The pipeline will not utilize the preprocessing functions from the pretrained models. The main reason being that these preprocessing function were tailored for object recognition tasks, and may not be suitable for the skin lesion classification task. The skin lesion images have unique characteristics, such as color variations, textures, and shapes, which may require specific preprocessing techniques to enhance the model's performance. Also, to avoid the loss of valuable information during the preprocessing process, the project will implement a custom preprocessing function that is tailored to the characteristics of the skin lesion images. The preprocessing function will include resizing the images to a standard size using adaptive resizing, normalizing the pixel values, and applying data augmentation techniques to increase the diversity of the training dataset.

The selected architectures have been trained on the ImageNet dataset, which has different characteristics compared to the skin lesion images in the HAM10000 dataset. Therefore, the preprocessing function will be customized to suit the skin lesion classification task and enhance the model's performance. However, here is a brief description of the preprocessing functions from the pretrained models:

* Preprossesing all preprocessing function have in common:
  + Convert the image to a float32 tensor.
  + Resize the image to the input size required by the model.
* VGG16 and VGG19: These are part of the VGG family of models, which are known for their simplicity and effectiveness. The VGG16 and VGG19 models use the same preprocessing function, which involves subtracting the mean RGB value of the ImageNet dataset from each pixel and reversing the order of the channels from RGB to BGR.
  + The main issue with these preprocessing functions is that they subtract the mean RGB value of the ImageNet dataset, which differ from the mean RGB value of the skin lesion images. This could affect the model's performance, as the color distribution of the skin lesion images is different from that of the ImageNet dataset.
* ResNet, InceptionResNetV2, Xception, and MobileNetV2: share a similar preprocessing function, which involves normalizing the pixel values to be in the range [-1, 1] and resizing the image to the input size required by the model. They focus on easier normalization steps that still allow the netweork to learn effectively.
  + These seem more suitable for the skin lesion classification task, due to their simplicity and lower risk of removing valuable information during the preprocessing process. The normalization of the pixel values to be in the range [-1, 1] helps to maintain the integrity of the image features and enhance the model's performance. Here is a demonstration of the preprocessed images for each of the selected models:

def display\_preprocessed\_images(image):  
 # Load image  
 original\_image = image  
 image\_data = np.expand\_dims(np.array(original\_image), 0)  
  
 # Define preprocess functions  
 preprocess\_functions = [  
 vgg16\_preprocess\_input,  
 vgg19\_preprocess\_input,  
 resnet101v2\_preprocess\_input,  
 inception\_resnetv2\_preprocess\_input,  
 xception\_preprocess\_input,  
 mobilenet\_v2\_preprocess\_input  
 ]  
 model\_names = ['VGG16', 'VGG19', 'ResNet', 'InceptionResNetV2', 'Xception', 'MobileNetV2']  
  
 # Create subplots grid  
 fig, axs = plt.subplots(3, 3, figsize=[15, 15])  
  
 # Display original image on top, center  
 fig.suptitle("Original and Preprocessed Images", fontsize=15)  
 axs[0, 1].imshow(original\_image)  
 axs[0, 1].title.set\_text('Original Image')  
  
 # Process image and display for each model  
 for i, preprocess\_func in enumerate(preprocess\_functions):  
 # Preprocess image  
 preprocessed = preprocess\_func(image\_data.copy())  
  
 # Scale back to range [0,255] for visualization, since processing function can change the range of pixel values  
 preprocessed = preprocessed.squeeze()  
 preprocessed -= preprocessed.min() # ensure the minimal value is 0.0  
 preprocessed /= preprocessed.max() # ensure the maximum value is 1.0  
  
 # Display preprocessed image  
 row = (i // 3) + 1  
 col = i % 3  
 axs[row, col].imshow(preprocessed)  
 axs[row, col].title.set\_text(f'Preprocessed for {model\_names[i]}')  
  
 # Hide the empty subplot  
 axs[0, 0].axis('off')  
 axs[0, 2].axis('off')  
  
 plt.tight\_layout()  
 plt.show()  
  
  
# Display the preprocessed images  
img\_dict = train\_ds['image'][0]   
image\_array = np.array(Image.open(io.BytesIO(img\_dict['bytes'])))  
display\_preprocessed\_images(image\_array)



## Data preparation Pipeline

Using the tensorflow dataset API, the data preparation, image preprocessing and augmentation steps looks as follows:

@tf.function  
def process\_training(image):  
 img = tf.image.random\_flip\_left\_right(image)  
 img = tf.image.random\_flip\_up\_down(img)  
 img = tf.cast(img, tf.float32)  
 img = adaptive\_resize(img, (256, 256))  
 img = tf.image.rot90(img, k=tf.random.uniform(shape=[], minval=0, maxval=4, dtype=tf.int32))  
 img = tfa.image.rotate(img, tf.random.uniform(shape=[], minval=-math.pi / 8.0, maxval=math.pi / 8.0),  
 fill\_mode='reflect')  
  
 return img  
  
@tf.function  
def process\_validation(image):  
 img = adaptive\_resize(image, (256, 256))  
 return img  
  
@tf.function  
def adaptive\_resize(image, target\_size=(256, 256)):  
 current\_size = tf.cast(tf.shape(image)[:2], tf.float32)  
 target\_size\_float = tf.cast(target\_size, tf.float32)  
 max\_scale\_factor = tf.reduce\_max(target\_size\_float / current\_size)  
  
 resized\_image = tf.cond(max\_scale\_factor < 1, lambda: resize\_area(image, target\_size),  
 lambda: resize\_bicubic(image, target\_size))  
  
 return resized\_image  
  
@tf.function  
def resize\_area(image, target\_size=(256, 256)):  
 return tf.image.resize(image, target\_size, method=tf.image.ResizeMethod.AREA)  
  
@tf.function  
def resize\_bicubic(image, target\_size=(256, 256)):  
 return tf.image.resize(image, target\_size, method=tf.image.ResizeMethod.BICUBIC)  
  
def create\_tf\_datasets(dataset, batch\_size=32):  
 # Compute age scaling parameters  
 ages = np.concatenate([dataset['train']['age'], dataset['validation']['age']])  
 ages = [age for age in ages if age is not None]  
 age\_mean, age\_std = np.mean(ages), np.std(ages)  
  
 # Add initialization for seen lesion\_ids  
 # seen\_lesion\_ids = {}  
 #   
 # def filter\_duplicate(sample):  
 # if sample['lesion\_id'] in seen\_lesion\_ids:  
 # return False  
 # seen\_lesion\_ids[sample['lesion\_id']] = True  
 # return True  
 #   
 # train\_ds = dataset['train'].filter(filter\_duplicate)  
 # val\_ds = dataset['validation'].filter(filter\_duplicate)  
  
 # Define pre-tensor transformation function  
 def pre\_tensor\_transform(sample):  
 sample['age'] = age\_mean if sample['age'] is None else sample['age']  
 sample['sex'] = int(sample['sex'] == 'male')  
 return sample  
  
 # Apply pre-tensor transformations  
 train\_ds = dataset['train'].map(pre\_tensor\_transform)  
 val\_ds = dataset['validation'].map(pre\_tensor\_transform)  
  
 def filter\_None\_Nan\_samples(example):  
 for value in example.values():  
 if value is None:  
 return False  
 elif isinstance(value, float) and math.isnan(value):  
 return False  
 return True  
  
 train\_ds = train\_ds.filter(filter\_None\_Nan\_samples)  
 val\_ds = val\_ds.filter(filter\_None\_Nan\_samples)  
  
 # Convert to TensorFlow datasets  
 train\_tf\_ds = train\_ds.to\_tf\_dataset(columns=['image', 'sex', 'age', 'localization', 'dx'], label\_cols=['dx'], shuffle=True, batch\_size=batch\_size)  
 val\_tf\_ds = val\_ds.to\_tf\_dataset(columns=['image', 'sex', 'age', 'localization', 'dx'], label\_cols=['dx'], shuffle=False, batch\_size=batch\_size)  
  
 # # Preprocessing functions  
 unique\_localizations = len(pd\_dataset["localization"].unique())  
 unique\_classes = len(pd\_dataset["dx"].unique())  
  
 # Define lookup tables for 'localization' and 'dx'  
 localization\_lookup\_table = tf.lookup.StaticHashTable(  
 initializer=tf.lookup.KeyValueTensorInitializer(  
 keys=tf.constant(pd\_dataset["localization"].unique()),  
 values=tf.constant(list(range(unique\_localizations)), dtype=tf.int64),  
 ),  
 default\_value=-1 # You can adjust this as needed  
 )  
  
 dx\_lookup\_table = tf.lookup.StaticHashTable(  
 initializer=tf.lookup.KeyValueTensorInitializer(  
 keys=tf.constant(pd\_dataset["dx"].unique()),  
 values=tf.constant(list(range(unique\_classes)), dtype=tf.int64),  
 ),  
 default\_value=-1 # You can adjust this as needed  
 )  
  
 # Define image and metadata preprocessing function  
 def preprocess\_image\_and\_metadata(features, labels, training):  
 # Image preprocessing  
  
 image = features['image']  
  
 processed\_images = tf.cond(  
 tf.equal(training, True),  
 lambda: process\_training(image),  
 lambda: process\_validation(image)  
 )  
  
 # Localization one-hot encoding using lookup table  
 localization\_indices = localization\_lookup\_table.lookup(features['localization'])  
 localization\_one\_hot = tf.one\_hot(localization\_indices, depth=unique\_localizations)  
  
 # Dx one-hot encoding using lookup table  
 dx\_indices = dx\_lookup\_table.lookup(labels)  
 dx\_one\_hot = tf.one\_hot(dx\_indices, depth=unique\_classes)  
  
 # Sex and age processing  
 sex = tf.expand\_dims(tf.cast(features['sex'], tf.float32), -1)  
 age = tf.expand\_dims(tf.cast(features['age'], tf.float32), -1)  
  
 # Concatenate metadata features  
 metadata\_input = tf.concat([localization\_one\_hot, sex, age], axis=-1)  
  
 processed\_features = {  
 'image\_input': processed\_images,  
 'metadata\_input': metadata\_input  
 }  
 return processed\_features, dx\_one\_hot  
  
 # Wrap the preprocessing function to include the training flag  
 def wrap\_preprocess\_image\_and\_metadata(training):  
 return lambda features, labels: preprocess\_image\_and\_metadata(features, labels, training)  
  
 # Apply image preprocessing  
 train\_tf\_ds = train\_tf\_ds.map(wrap\_preprocess\_image\_and\_metadata(training=True), num\_parallel\_calls=tf.data.AUTOTUNE).prefetch(tf.data.AUTOTUNE)  
 val\_tf\_ds = val\_tf\_ds.map(wrap\_preprocess\_image\_and\_metadata(training=False), num\_parallel\_calls=tf.data.AUTOTUNE).cache().prefetch(tf.data.AUTOTUNE)  
  
 return train\_tf\_ds, val\_tf\_ds  
  
# Create TensorFlow datasets  
train\_tf\_ds, val\_tf\_ds = create\_tf\_datasets(dataset, batch\_size=32)

/home/lramossoto/.virtualenvs/SkinCancerClassification/lib/python3.10/site-packages/datasets/arrow\_dataset.py:401: FutureWarning:  
  
The output of `to\_tf\_dataset` will change when a passing single element list for `labels` or `columns` in the next datasets version. To return a tuple structure rather than dict, pass a single string.  
Old behaviour: columns=['a'], labels=['labels'] -> (tf.Tensor, tf.Tensor)   
 : columns='a', labels='labels' -> (tf.Tensor, tf.Tensor)   
New behaviour: columns=['a'],labels=['labels'] -> ({'a': tf.Tensor}, {'labels': tf.Tensor})   
 : columns='a', labels='labels' -> (tf.Tensor, tf.Tensor)

# Model Building

The initial model training stage involves training the selected deep learning architecture on the HAM10000 dataset using the image and metadata features. In this section, we will train the selected model on the training dataset, validate it on the validation dataset, and evaluate its performance on the test dataset. The model will be trained using the image preprocessing pipeline and the metadata features to predict the diagnosis of the lesion.

For simplicity, the initial model training will use the default hyperparameters of the selected deep learning architecture. The model will be trained for a fixed number of epochs, and the training process will be monitored using the training and validation loss and accuracy metrics. The performance of the model will be evaluated on the test dataset to assess its generalization and robustness.

The model training process will involve the following steps:

1. Define the input features: The input features for the model will include the image and metadata features. The image features will be processed using the image preprocessing pipeline, and the metadata features will be used as additional features to enhance the model's performance.
2. Define the model architecture: The model architecture will be based on the selected deep learning architecture, such as VGG16, VGG19, ResNet101V2, InceptionResNetV2, Xception, or MobileNetV2. The model will include the image and metadata input layers, convolutional layers, pooling layers, and fully connected layers.
3. Compile the model: The model will be compiled with the AdamW optimizer, categorical crossentropy loss function, and accuracy metric. The AdamW optimizer is a variant of the Adam optimizer that decouples weight decay regularization from the optimization steps. The categorical crossentropy loss function is commonly used for multi-class classification tasks, and the accuracy metric measures the proportion of correctly classified samples.
4. Train the model: The model will be trained on the training dataset and validated on the validation dataset. The model will be trained for a fixed number of epochs, and the training process will be monitored using the training and validation loss and accuracy metrics.
5. Evaluate the model: The performance of the model will be evaluated on the test dataset using the accuracy, precision, recall, and F1 score metrics. The confusion matrix will provide detailed insights into the model's performance across different classes and help identify areas for improvement.
6. Save the model: The trained model will be saved for future use and deployment in the skin lesion classification task.

metadata\_input = Input(shape=len(localization\_counts)+2, name='metadata\_input')  
image\_input = Input(shape=(256, 256, 3), name='image\_input')  
  
base\_model = VGG19(weights='imagenet', include\_top=False, input\_shape=(256, 256, 3))  
x = base\_model(image\_input)  
  
# Freeze the base model  
base\_model.trainable = False  
  
#Unfreeze last 5 layers  
for layer in base\_model.layers[-5:]:  
 layer.trainable = True  
   
# Attention mechanism CBAM  
def cbam\_block(input\_tensor, ratio=16, kernel\_size=7):  
 channel = GlobalAveragePooling2D()(input\_tensor)  
 channel = Dense(channel.shape[-1] // ratio, kernel\_initializer='he\_normal', use\_bias=True)(channel)  
 channel = BatchNormalization()(channel)  
 channel = Activation(tf.keras.activations.gelu)(channel)  
 channel = Dense(input\_tensor.shape[-1], kernel\_initializer='he\_normal', use\_bias=True)(channel)  
 channel = Activation('sigmoid')(channel)  
  
 if K.image\_data\_format() == 'channels\_first':  
 channel = Reshape((input\_tensor.shape[1], 1, 1))(channel)  
 else: # 'channels\_last'  
 channel = Reshape((1, 1, input\_tensor.shape[-1]))(channel)  
  
 channel\_attention = Multiply()([input\_tensor, channel])  
  
 spatial = Conv2D(1, kernel\_size, padding='same', use\_bias=True)(channel\_attention)  
 spatial = Activation('sigmoid')(spatial)  
 output = Multiply()([channel\_attention, spatial])  
  
 return output  
  
x = cbam\_block(x)  
  
# Add a global average pooling layer  
x = Flatten()(x)  
x = tf.keras.layers.concatenate([x, metadata\_input])  
  
# Add a fully connected layer  
x = Dense(512)(x)  
x = BatchNormalization()(x)  
x = Activation('swish')(x)  
  
# Add a dropout layer  
x = Dropout(0.5)(x)  
  
# Add a final dense layer  
outputs = Dense(len(dx\_types), activation='softmax')(x)  
  
# Create the model  
model = Model(inputs=[image\_input, metadata\_input], outputs=outputs)

## Activation Function

The activation function is a key component of deep learning models that introduces non-linearity into the model and enables the model to learn complex patterns and relationships in the data. In this section, we will define the activation function for the model based on the Swish and GELU activation functions. The Swish and GELU activation functions have been shown to improve the performance of deep learning models on various tasks, including image classification.

The activation functions considered are:

* ReLU: Rectified Linear Unit activation function
* Swish: Self-Gated activation function
* GELU: Gaussian Error Linear Unit activation function
* Sigmoid: Sigmoid activation function
* Softmax: Softmax activation function

Each activation function has unique properties and characteristics that can affect the model's performance and convergence during training. The activation function will be used in the hidden layers of the model to introduce non-linearity and enable the model to learn complex patterns in the data.

### ReLU Activation Function

The ReLU activation function is a popular activation function that introduces non-linearity into the model by setting negative values to zero. The ReLU activation function is defined as follows:

The ReLU activation function is commonly used in deep learning models due to its simplicity and effectiveness in training deep neural networks. The ReLU activation function helps to address the vanishing gradient problem and improve the convergence of the model during training. The ReLU activation function has been widely used in various deep learning architectures, including VGG, ResNet, and Inception.

### Swish Activation Function

The Swish activation function is a self-gated activation function that has been shown to improve the performance of deep learning models. The Swish activation function is defined as follows:

The Swish activation function introduces a non-linearity into the model by multiplying the input with the sigmoid function of the input. The Swish activation function has been shown to improve the convergence and performance of deep learning models on various tasks, including image classification. The Swish activation function was proposed by Ramachandran et al. in the paper "Searching for Activation Functions" (2017).

### GELU Activation Function

The gelu activation function is a variant of the Swish activation function that uses the Gaussian Error Linear Unit (GELU) function instead of the sigmoid function. The GELU function is defined as follows:

The GELU activation function similarly to Swish, uses a non-linearity applying the Gaussian Error Linear Unit function to the input. The GELU activation function has been shown to improve the convergence and performance of deep learning models on various tasks, including image classification. The GELU activation function was proposed by Hendrycks and Gimpel in the paper "Gaussian Error Linear Units (GELUs)" (2016).

### Sigmoid Activation Function

The sigmoid activation function is a non-linear activation function that squashes the input values to the range [0, 1]. The sigmoid activation function is defined as follows:

The sigmoid activation function is commonly used in binary classification tasks to predict the probability of a sample belonging to a particular class. The sigmoid activation function introduces non-linearity into the model and helps the model learn complex patterns in the data. The sigmoid activation function is useful for tasks where the output is binary or probabilistic. It was proposed by David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams in the paper "Learning representations by back-propagating errors" (1986).

### Softmax Activation Function

The softmax activation function is a generalization of the sigmoid activation function that squashes the input values to the range [0, 1] and normalizes them to sum to one. The softmax activation function is defined as follows:

Where:

* : Number of classes
* : Input value for class
* : Euler's number
* : Index of the classes
* : Index of the classes

The softmax activation function is commonly used in multi-class classification tasks to predict the probability distribution of the classes. The softmax activation function introduces non-linearity into the model and helps the model learn complex patterns in the data. The softmax activation function is useful for tasks where the output is multi-class or probabilistic. It was proposed by Geoffrey E. Hinton in the paper "Neural Networks for Machine Learning" (2012).

## Optimizer AdamW

The AdamW optimizer is a variant of the Adam optimizer that decouples weight decay regularization from the optimization steps. The AdamW optimizer has been shown to improve the generalization performance of deep learning models by decoupling weight decay from the optimization steps. The AdamW optimizer is commonly used for image classification tasks and has been shown to improve the performance of deep learning models on various datasets.

optimizer = AdamW(weight\_decay=0.004, learning\_rate=0.001)

## Loss Function

The categorical\_crossentropy is given by the following equation:

Where:

* : Number of classes
* : True probability distribution of the classes
* : Predicted probability distribution of the classes
* : Index of the classes

The loss function for the skin lesion classification task is categorical crossentropy, which is commonly used for multi-class classification tasks. The categorical crossentropy loss function measures the difference between the predicted probability distribution and the true probability distribution of the classes. The model aims to minimize the categorical crossentropy loss function during training to improve its performance on the test dataset.

loss = 'categorical\_crossentropy'

## Metrics

The model will be evaluated based on the accuracy metric, which measures the proportion of correctly classified samples. The accuracy metric is commonly used for classification tasks to evaluate the model's performance on the test dataset. The accuracy metric will be used to assess the model's generalization and robustness on the unseen test dataset.

### Accuracy

Accuracy formula is given by the following equation:

Where:

* TP: True Positives
* TN: True Negatives
* FP: False Positives
* FN: False Negatives
* TP + TN: Total Correct Predictions
* FP + FN: Total Incorrect Predictions
* TP + TN + FP + FN: Total Predictions

The accuracy metric measures the proportion of correctly classified samples. It is calculated as the number of correct predictions divided by the total number of predictions. The accuracy metric is commonly used for classification tasks to evaluate the model's performance on the test dataset.

### Recall

Recall formula is given by the following equation:

Where:

* TP + FN: Total Actual Positives

The recall metric measures the proportion of true positive samples that were correctly identified by the model. It is calculated as the number of true positive samples divided by the sum of true positive and false negative samples. The recall metric is useful for evaluating the model's ability to correctly identify positive samples.

### Precision

Precision formula is given by the following equation:

Where:

* TP + FP: Total Predicted Positives

The precision metric measures the proportion of true positive samples among the samples predicted as positive by the model. It is calculated as the number of true positive samples divided by the sum of true positive and false positive samples. The precision metric is useful for evaluating the model's ability to avoid false positive predictions.

### F1 Score

F1 score formula is given by the following equation:

Where:

* Precision + Recall: Total Predicted Positives
* Precision + Recall: Total Actual Positives
* 2: Harmonic Mean

The F1 score metric is the harmonic mean of precision and recall. It provides a balance between precision and recall, taking into account both false positive and false negative predictions. The F1 score metric is useful for evaluating the model's overall performance on the test dataset.

### Confusion Matrix

The confusion matrix is a table that summarizes the model's performance on the test dataset. It shows the number of true positive, true negative, false positive, and false negative predictions. The confusion matrix is useful for evaluating the model's performance across different classes and identifying areas for improvement.

These metrics will be used to evaluate the model's performance on the test dataset and assess its generalization and robustness. The confusion matrix will provide detailed insights into the model's performance across different classes and help identify areas for improvement.

## Model Compilation

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

## Callbacks

Callbacks are used to monitor the training process, prevent overfitting, and improve the model's generalization. In this section, we will define the early stopping and learning rate reduction callbacks to prevent overfitting and improve the model's performance. The early stopping callback will monitor the validation loss and stop the training process if the loss does not improve after a certain number of epochs. The learning rate reduction callback will reduce the learning rate if the validation loss does not improve after a certain number of epochs.

### Early Stopping

The early stopping callback will monitor the validation loss and stop the training process if the loss does not improve after a certain number of epochs. This helps prevent overfitting and improves the model's generalization.

### ReduceLROnPlateau

Similarly, the ReduceLROnPlateau callback will reduce the learning rate if the validation loss does not improve after a certain number of epochs. This helps improve the model's performance and convergence during training.

early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)  
reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.2, patience=3, verbose=1)

callbacks = [early\_stopping, reduce\_lr]

def save\_model(model, history, filename):  
 with open(filename, 'wb') as file:  
 pickle.dump((model, history), file)  
  
  
def load\_model\_pkl(filename):  
 with open(filename, 'rb') as file:  
 model, history = pickle.load(file)  
 return model, history

# Train the model  
  
file\_path = 'skin\_lesion\_classification\_model.pkl'  
file\_path\_keras = 'skin\_lesion\_classification\_model.keras'  
  
# To avoid running the training process multiple times, we will check if the model has already been trained and saved  
if os.path.isfile(file\_path):  
 from tensorflow.keras.models import load\_model  
 model = load\_model(file\_path\_keras, safe\_mode=False)  
 \_, history = load\_model\_pkl(file\_path)  
else:   
 history = model.fit(train\_tf\_ds, validation\_data=val\_tf\_ds, epochs=50, callbacks=callbacks)  
 model.save(file\_path\_keras)

Epoch 1/50

2024-04-01 20:44:53.465690: I external/local\_xla/xla/stream\_executor/cuda/cuda\_dnn.cc:454] Loaded cuDNN version 8907  
2024-04-01 20:44:53.686684: I external/local\_tsl/tsl/platform/default/subprocess.cc:304] Start cannot spawn child process: No such file or directory  
2024-04-01 20:44:56.723326: I external/local\_tsl/tsl/platform/default/subprocess.cc:304] Start cannot spawn child process: No such file or directory  
2024-04-01 20:44:57.502762: I external/local\_xla/xla/service/service.cc:168] XLA service 0x560249eea980 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:  
2024-04-01 20:44:57.502814: I external/local\_xla/xla/service/service.cc:176] StreamExecutor device (0): NVIDIA GeForce RTX 4080, Compute Capability 8.9  
2024-04-01 20:44:57.527130: I tensorflow/compiler/mlir/tensorflow/utils/dump\_mlir\_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR\_CRASH\_REPRODUCER\_DIRECTORY` to enable.  
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR  
I0000 00:00:1712018697.609386 2468060 device\_compiler.h:186] Compiled cluster using XLA! This line is logged at most once for the lifetime of the process.

300/300 [==============================] - 108s 337ms/step - loss: 1.1937 - accuracy: 0.6485 - val\_loss: 1.0308 - val\_accuracy: 0.6850 - lr: 0.0010  
Epoch 2/50  
300/300 [==============================] - 73s 244ms/step - loss: 0.9739 - accuracy: 0.6739 - val\_loss: 0.9030 - val\_accuracy: 0.6958 - lr: 0.0010  
Epoch 3/50  
300/300 [==============================] - 71s 234ms/step - loss: 0.9220 - accuracy: 0.6807 - val\_loss: 0.8408 - val\_accuracy: 0.7127 - lr: 0.0010  
Epoch 4/50  
300/300 [==============================] - 74s 244ms/step - loss: 0.8851 - accuracy: 0.6925 - val\_loss: 0.8075 - val\_accuracy: 0.7159 - lr: 0.0010  
Epoch 5/50  
300/300 [==============================] - 72s 239ms/step - loss: 0.8448 - accuracy: 0.6997 - val\_loss: 0.7840 - val\_accuracy: 0.7199 - lr: 0.0010  
Epoch 6/50  
300/300 [==============================] - 71s 237ms/step - loss: 0.8126 - accuracy: 0.7054 - val\_loss: 0.7598 - val\_accuracy: 0.7271 - lr: 0.0010  
Epoch 7/50  
300/300 [==============================] - 70s 231ms/step - loss: 0.7811 - accuracy: 0.7198 - val\_loss: 0.7568 - val\_accuracy: 0.7331 - lr: 0.0010  
Epoch 8/50  
300/300 [==============================] - 70s 232ms/step - loss: 0.7577 - accuracy: 0.7236 - val\_loss: 0.7242 - val\_accuracy: 0.7368 - lr: 0.0010  
Epoch 9/50  
300/300 [==============================] - 72s 238ms/step - loss: 0.7506 - accuracy: 0.7244 - val\_loss: 0.7173 - val\_accuracy: 0.7428 - lr: 0.0010  
Epoch 10/50  
300/300 [==============================] - 72s 240ms/step - loss: 0.7253 - accuracy: 0.7345 - val\_loss: 0.6852 - val\_accuracy: 0.7500 - lr: 0.0010  
Epoch 11/50  
300/300 [==============================] - 72s 240ms/step - loss: 0.7040 - accuracy: 0.7392 - val\_loss: 0.6752 - val\_accuracy: 0.7580 - lr: 0.0010  
Epoch 12/50  
300/300 [==============================] - 73s 241ms/step - loss: 0.6979 - accuracy: 0.7448 - val\_loss: 0.6789 - val\_accuracy: 0.7596 - lr: 0.0010  
Epoch 13/50  
300/300 [==============================] - 72s 238ms/step - loss: 0.6900 - accuracy: 0.7452 - val\_loss: 0.6474 - val\_accuracy: 0.7648 - lr: 0.0010  
Epoch 14/50  
300/300 [==============================] - 70s 231ms/step - loss: 0.6574 - accuracy: 0.7607 - val\_loss: 0.6184 - val\_accuracy: 0.7749 - lr: 0.0010  
Epoch 15/50  
300/300 [==============================] - 70s 233ms/step - loss: 0.6474 - accuracy: 0.7607 - val\_loss: 0.6339 - val\_accuracy: 0.7681 - lr: 0.0010  
Epoch 16/50  
300/300 [==============================] - 69s 230ms/step - loss: 0.6407 - accuracy: 0.7631 - val\_loss: 0.6157 - val\_accuracy: 0.7781 - lr: 0.0010  
Epoch 17/50  
300/300 [==============================] - 70s 231ms/step - loss: 0.6128 - accuracy: 0.7761 - val\_loss: 0.5953 - val\_accuracy: 0.7849 - lr: 0.0010  
Epoch 18/50  
300/300 [==============================] - 70s 232ms/step - loss: 0.6135 - accuracy: 0.7723 - val\_loss: 0.5678 - val\_accuracy: 0.7893 - lr: 0.0010  
Epoch 19/50  
300/300 [==============================] - 70s 234ms/step - loss: 0.6013 - accuracy: 0.7794 - val\_loss: 0.5554 - val\_accuracy: 0.7953 - lr: 0.0010  
Epoch 20/50  
300/300 [==============================] - 71s 235ms/step - loss: 0.5905 - accuracy: 0.7854 - val\_loss: 0.5423 - val\_accuracy: 0.7965 - lr: 0.0010  
Epoch 21/50  
300/300 [==============================] - 71s 235ms/step - loss: 0.5773 - accuracy: 0.7855 - val\_loss: 0.5444 - val\_accuracy: 0.8002 - lr: 0.0010  
Epoch 22/50  
300/300 [==============================] - 70s 233ms/step - loss: 0.5724 - accuracy: 0.7883 - val\_loss: 0.5561 - val\_accuracy: 0.7986 - lr: 0.0010  
Epoch 23/50  
300/300 [==============================] - 72s 238ms/step - loss: 0.5653 - accuracy: 0.7948 - val\_loss: 0.4976 - val\_accuracy: 0.8138 - lr: 0.0010  
Epoch 24/50  
300/300 [==============================] - 71s 235ms/step - loss: 0.5431 - accuracy: 0.7992 - val\_loss: 0.4981 - val\_accuracy: 0.8158 - lr: 0.0010  
Epoch 25/50  
300/300 [==============================] - 70s 233ms/step - loss: 0.5437 - accuracy: 0.7951 - val\_loss: 0.4771 - val\_accuracy: 0.8218 - lr: 0.0010  
Epoch 26/50  
300/300 [==============================] - 71s 235ms/step - loss: 0.5260 - accuracy: 0.8101 - val\_loss: 0.4681 - val\_accuracy: 0.8266 - lr: 0.0010  
Epoch 27/50  
300/300 [==============================] - 71s 234ms/step - loss: 0.5240 - accuracy: 0.8066 - val\_loss: 0.4587 - val\_accuracy: 0.8291 - lr: 0.0010  
Epoch 28/50  
300/300 [==============================] - 72s 239ms/step - loss: 0.5034 - accuracy: 0.8162 - val\_loss: 0.4715 - val\_accuracy: 0.8283 - lr: 0.0010  
Epoch 29/50  
300/300 [==============================] - 71s 237ms/step - loss: 0.5028 - accuracy: 0.8127 - val\_loss: 0.4468 - val\_accuracy: 0.8371 - lr: 0.0010  
Epoch 30/50  
300/300 [==============================] - 72s 238ms/step - loss: 0.4787 - accuracy: 0.8206 - val\_loss: 0.4567 - val\_accuracy: 0.8355 - lr: 0.0010  
Epoch 31/50  
300/300 [==============================] - 72s 239ms/step - loss: 0.4989 - accuracy: 0.8171 - val\_loss: 0.4684 - val\_accuracy: 0.8230 - lr: 0.0010  
Epoch 32/50  
 1/300 [..............................] - ETA: 2:33 - loss: 0.7194 - accuracy: 0.6875

2024-04-01 21:22:25.801810: W tensorflow/core/kernels/data/prefetch\_autotuner.cc:52] Prefetch autotuner tried to allocate 25168896 bytes after encountering the first element of size 25168896 bytes.This already causes the autotune ram budget to be exceeded. To stay within the ram budget, either increase the ram budget or reduce element size

299/300 [============================>.] - ETA: 0s - loss: 0.4943 - accuracy: 0.8154  
Epoch 32: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.  
300/300 [==============================] - 70s 234ms/step - loss: 0.4944 - accuracy: 0.8155 - val\_loss: 0.4585 - val\_accuracy: 0.8319 - lr: 0.0010  
Epoch 33/50  
300/300 [==============================] - 72s 239ms/step - loss: 0.4664 - accuracy: 0.8307 - val\_loss: 0.4117 - val\_accuracy: 0.8495 - lr: 2.0000e-04  
Epoch 34/50  
300/300 [==============================] - 71s 237ms/step - loss: 0.4523 - accuracy: 0.8350 - val\_loss: 0.3995 - val\_accuracy: 0.8507 - lr: 2.0000e-04  
Epoch 35/50  
 1/300 [..............................] - ETA: 2:32 - loss: 0.6295 - accuracy: 0.7812

2024-04-01 21:25:59.675329: W tensorflow/core/kernels/data/prefetch\_autotuner.cc:52] Prefetch autotuner tried to allocate 25168896 bytes after encountering the first element of size 25168896 bytes.This already causes the autotune ram budget to be exceeded. To stay within the ram budget, either increase the ram budget or reduce element size

300/300 [==============================] - 72s 239ms/step - loss: 0.4324 - accuracy: 0.8378 - val\_loss: 0.4003 - val\_accuracy: 0.8507 - lr: 2.0000e-04  
Epoch 36/50  
 1/300 [..............................] - ETA: 2:32 - loss: 0.2755 - accuracy: 0.8438

2024-04-01 21:27:11.813103: W tensorflow/core/kernels/data/prefetch\_autotuner.cc:52] Prefetch autotuner tried to allocate 25168896 bytes after encountering the first element of size 25168896 bytes.This already causes the autotune ram budget to be exceeded. To stay within the ram budget, either increase the ram budget or reduce element size

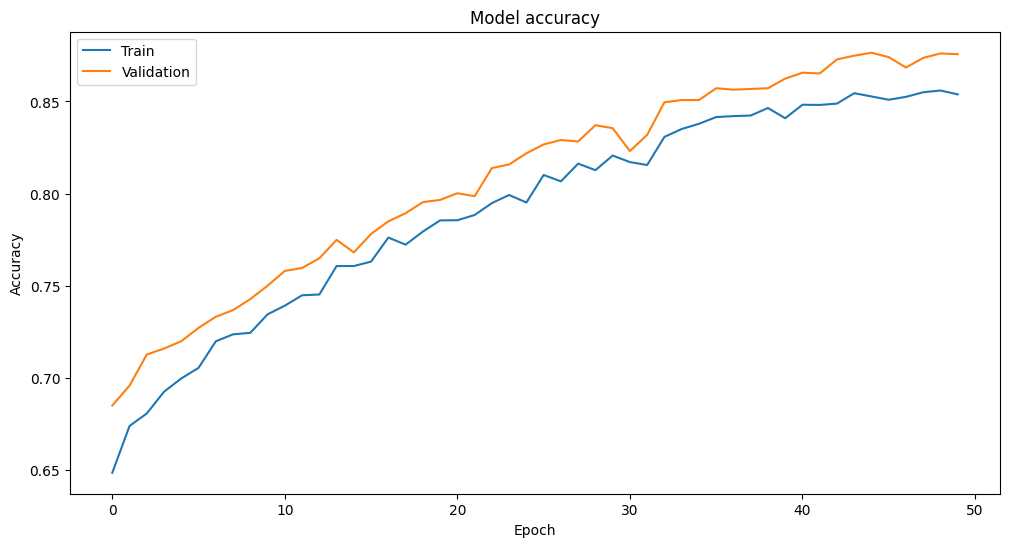
300/300 [==============================] - 73s 243ms/step - loss: 0.4308 - accuracy: 0.8415 - val\_loss: 0.3921 - val\_accuracy: 0.8571 - lr: 2.0000e-04  
Epoch 37/50  
300/300 [==============================] - 73s 242ms/step - loss: 0.4260 - accuracy: 0.8420 - val\_loss: 0.3869 - val\_accuracy: 0.8563 - lr: 2.0000e-04  
Epoch 38/50  
300/300 [==============================] - 71s 236ms/step - loss: 0.4255 - accuracy: 0.8423 - val\_loss: 0.3820 - val\_accuracy: 0.8567 - lr: 2.0000e-04  
Epoch 39/50  
300/300 [==============================] - 70s 233ms/step - loss: 0.4174 - accuracy: 0.8464 - val\_loss: 0.3783 - val\_accuracy: 0.8571 - lr: 2.0000e-04  
Epoch 40/50  
300/300 [==============================] - 71s 236ms/step - loss: 0.4288 - accuracy: 0.8409 - val\_loss: 0.3708 - val\_accuracy: 0.8624 - lr: 2.0000e-04  
Epoch 41/50  
300/300 [==============================] - 70s 232ms/step - loss: 0.4091 - accuracy: 0.8482 - val\_loss: 0.3673 - val\_accuracy: 0.8656 - lr: 2.0000e-04  
Epoch 42/50  
300/300 [==============================] - 72s 238ms/step - loss: 0.4131 - accuracy: 0.8481 - val\_loss: 0.3641 - val\_accuracy: 0.8652 - lr: 2.0000e-04  
Epoch 43/50  
300/300 [==============================] - 72s 239ms/step - loss: 0.4087 - accuracy: 0.8488 - val\_loss: 0.3588 - val\_accuracy: 0.8728 - lr: 2.0000e-04  
Epoch 44/50  
300/300 [==============================] - 71s 237ms/step - loss: 0.4023 - accuracy: 0.8544 - val\_loss: 0.3536 - val\_accuracy: 0.8748 - lr: 2.0000e-04  
Epoch 45/50  
300/300 [==============================] - 74s 244ms/step - loss: 0.3996 - accuracy: 0.8527 - val\_loss: 0.3541 - val\_accuracy: 0.8764 - lr: 2.0000e-04  
Epoch 46/50  
300/300 [==============================] - 71s 237ms/step - loss: 0.4032 - accuracy: 0.8509 - val\_loss: 0.3544 - val\_accuracy: 0.8740 - lr: 2.0000e-04  
Epoch 47/50  
299/300 [============================>.] - ETA: 0s - loss: 0.3962 - accuracy: 0.8526  
Epoch 47: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.  
300/300 [==============================] - 71s 237ms/step - loss: 0.3968 - accuracy: 0.8525 - val\_loss: 0.3549 - val\_accuracy: 0.8684 - lr: 2.0000e-04  
Epoch 48/50  
300/300 [==============================] - 71s 237ms/step - loss: 0.3849 - accuracy: 0.8550 - val\_loss: 0.3525 - val\_accuracy: 0.8736 - lr: 4.0000e-05  
Epoch 49/50  
300/300 [==============================] - 72s 239ms/step - loss: 0.3913 - accuracy: 0.8559 - val\_loss: 0.3500 - val\_accuracy: 0.8760 - lr: 4.0000e-05  
Epoch 50/50  
300/300 [==============================] - 73s 243ms/step - loss: 0.3886 - accuracy: 0.8538 - val\_loss: 0.3488 - val\_accuracy: 0.8756 - lr: 4.0000e-05

The model training process has been successfully completed, and the model has been trained on the training dataset and validated on the validation dataset. The training process was monitored using the training and validation loss and accuracy metrics.

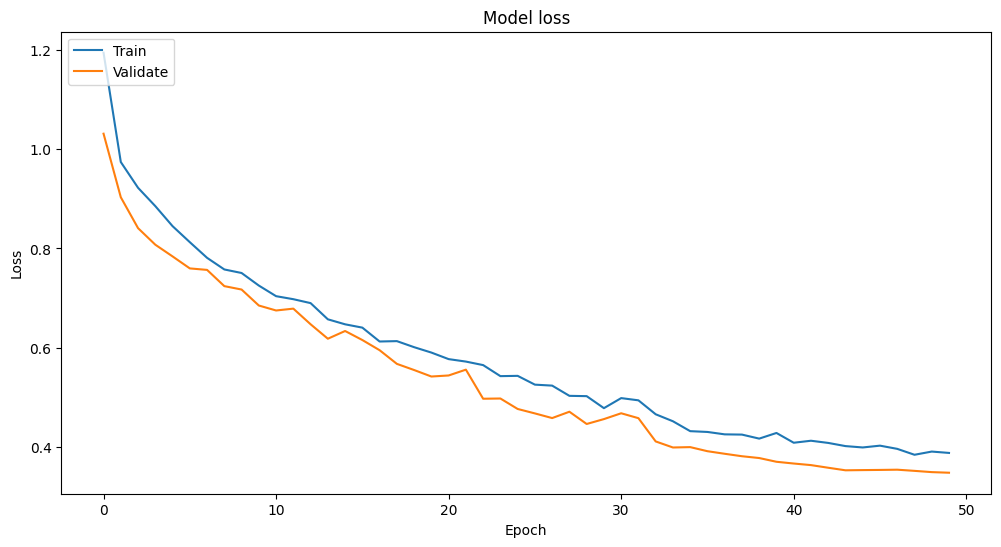
The early stopping and learning rate reduction callbacks were used to prevent overfitting and improve the model's generalization. The model was trained for 50 epochs, the early stopping was not triggered, and the learning rate was reduced by a factor of 0.5 after 3 epochs without improvement in the validation loss.

The training process has been completed

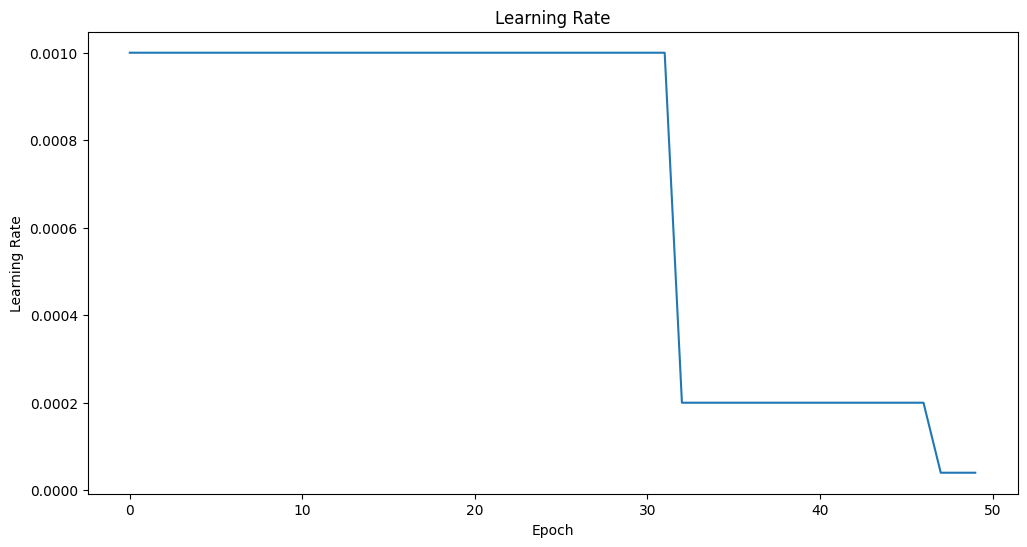
# Plot training & validation accuracy values   
plt.figure(figsize=(12, 6))  
plt.plot(history.history['accuracy'])  
plt.plot(history.history['val\_accuracy'])  
plt.title('Model accuracy')  
plt.ylabel('Accuracy')  
plt.xlabel('Epoch')  
plt.legend(['Train', 'Validation'], loc='upper left')  
plt.show()



# Plot training & validation loss values  
plt.figure(figsize=(12, 6))  
plt.plot(history.history['loss'])  
plt.plot(history.history['val\_loss'])  
plt.title('Model loss')  
plt.ylabel('Loss')  
plt.xlabel('Epoch')  
plt.legend(['Train', 'Validate'], loc='upper left')  
plt.show()



plt.figure(figsize=(12, 6))  
plt.plot(history.history['lr'])  
plt.title('Learning Rate')  
plt.ylabel('Learning Rate')  
plt.xlabel('Epoch')  
plt.show()



# Model Evaluation

def create\_test\_df(dataset, batch\_size=32):  
 # Compute age scaling parameters  
 ages = np.concatenate([dataset['train']['age'], dataset['validation']['age']])  
 ages = [age for age in ages if age is not None]  
 age\_mean, age\_std = np.mean(ages), np.std(ages)  
  
 # Define pre-tensor transformation function  
 def pre\_tensor\_transform(sample):  
 sample['age'] = age\_mean if sample['age'] is None else sample['age']  
 sample['sex'] = int(sample['sex'] == 'male')  
 return sample  
  
 # Apply pre-tensor transformations  
 test\_ds = dataset['test'].map(pre\_tensor\_transform)  
  
 def filter\_None\_Nan\_samples(example):  
 for value in example.values():  
 if value is None:  
 return False  
 elif isinstance(value, float) and math.isnan(value):  
 return False  
 return True  
  
 test\_ds = test\_ds.filter(filter\_None\_Nan\_samples)  
  
 # Convert to TensorFlow datasets  
 test\_tf\_ds = test\_ds.to\_tf\_dataset(columns=['image', 'sex', 'age', 'localization', 'dx'], label\_cols=['dx'], shuffle=True, batch\_size=batch\_size)  
  
 # # Preprocessing functions  
 unique\_localizations = len(pd\_dataset["localization"].unique())  
 unique\_classes = len(pd\_dataset["dx"].unique())  
  
 # Define lookup tables for 'localization' and 'dx'  
 localization\_lookup\_table = tf.lookup.StaticHashTable(  
 initializer=tf.lookup.KeyValueTensorInitializer(  
 keys=tf.constant(pd\_dataset["localization"].unique()),  
 values=tf.constant(list(range(unique\_localizations)), dtype=tf.int64),  
 ),  
 default\_value=-1  
 )  
  
 dx\_lookup\_table = tf.lookup.StaticHashTable(  
 initializer=tf.lookup.KeyValueTensorInitializer(  
 keys=tf.constant(pd\_dataset["dx"].unique()),  
 values=tf.constant(list(range(unique\_classes)), dtype=tf.int64),  
 ),  
 default\_value=-1  
 )  
  
 # Define image and metadata preprocessing function  
 def preprocess\_image\_and\_metadata(features, labels, training):  
 # Image preprocessing  
  
 image = features['image']  
  
 processed\_images = tf.cond(  
 tf.equal(training, True),  
 lambda: process\_training(image),  
 lambda: process\_validation(image)  
 )  
  
 # Localization one-hot encoding using lookup table  
 localization\_indices = localization\_lookup\_table.lookup(features['localization'])  
 localization\_one\_hot = tf.one\_hot(localization\_indices, depth=unique\_localizations)  
  
 # Dx one-hot encoding using lookup table  
 dx\_indices = dx\_lookup\_table.lookup(labels)  
 dx\_one\_hot = tf.one\_hot(dx\_indices, depth=unique\_classes)  
  
 # Sex and age processing  
 sex = tf.expand\_dims(tf.cast(features['sex'], tf.float32), -1)  
 age = tf.expand\_dims(tf.cast(features['age'], tf.float32), -1)  
  
 # Concatenate metadata features  
 metadata\_input = tf.concat([localization\_one\_hot, sex, age], axis=-1)  
  
 processed\_features = {  
 'image\_input': processed\_images,  
 'metadata\_input': metadata\_input  
 }  
 return processed\_features, dx\_one\_hot  
  
 # Wrap the preprocessing function to include the training flag  
 def wrap\_preprocess\_image\_and\_metadata(training):  
 return lambda features, labels: preprocess\_image\_and\_metadata(features, labels, training)  
  
 # Apply image preprocessing  
 test\_tf\_ds = test\_tf\_ds.map(wrap\_preprocess\_image\_and\_metadata(training=True), num\_parallel\_calls=tf.data.AUTOTUNE).prefetch(tf.data.AUTOTUNE)  
  
 return test\_tf\_ds  
  
# Create TensorFlow datasets  
test\_tf\_ds = create\_test\_df(dataset, batch\_size=32)

/home/lramossoto/.virtualenvs/SkinCancerClassification/lib/python3.10/site-packages/datasets/arrow\_dataset.py:401: FutureWarning:  
  
The output of `to\_tf\_dataset` will change when a passing single element list for `labels` or `columns` in the next datasets version. To return a tuple structure rather than dict, pass a single string.  
Old behaviour: columns=['a'], labels=['labels'] -> (tf.Tensor, tf.Tensor)   
 : columns='a', labels='labels' -> (tf.Tensor, tf.Tensor)   
New behaviour: columns=['a'],labels=['labels'] -> ({'a': tf.Tensor}, {'labels': tf.Tensor})   
 : columns='a', labels='labels' -> (tf.Tensor, tf.Tensor)

test\_image\_inputs = [] # to store image inputs  
test\_metadata\_inputs = [] # to store metadata inputs  
test\_labels = [] # to store true labels  
  
for features, label in test\_tf\_ds.unbatch().as\_numpy\_iterator():  
 test\_image\_inputs.append(features['image\_input'])  
 test\_metadata\_inputs.append(features['metadata\_input'])  
 test\_labels.append(np.argmax(label))  
  
test\_image\_inputs = np.array(test\_image\_inputs)  
test\_metadata\_inputs = np.array(test\_metadata\_inputs)  
test\_labels = np.array(test\_labels)  
  
# Get predictions  
predictions = model.predict([test\_image\_inputs, test\_metadata\_inputs])  
predicted\_classes = np.argmax(predictions, axis=1)

41/41 [==============================] - 3s 64ms/step

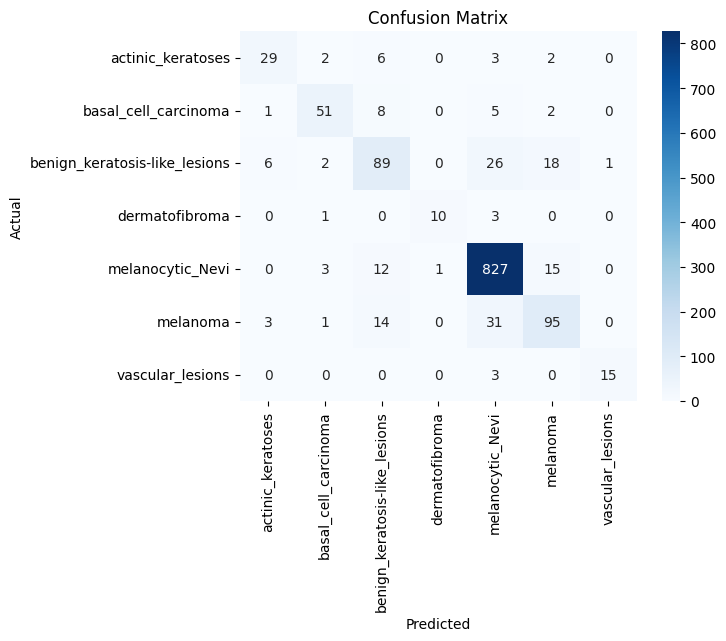
print(classification\_report(test\_labels, predicted\_classes, target\_names=dx\_types))

precision recall f1-score support  
  
 actinic\_keratoses 0.74 0.69 0.72 42  
 basal\_cell\_carcinoma 0.85 0.76 0.80 67  
benign\_keratosis-like\_lesions 0.69 0.63 0.66 142  
 dermatofibroma 0.91 0.71 0.80 14  
 melanocytic\_Nevi 0.92 0.96 0.94 858  
 melanoma 0.72 0.66 0.69 144  
 vascular\_lesions 0.94 0.83 0.88 18  
  
 accuracy 0.87 1285  
 macro avg 0.82 0.75 0.78 1285  
 weighted avg 0.86 0.87 0.87 1285

def multiclass\_roc\_auc\_score(y\_test, y\_pred, average="macro"):  
 label\_binarizer = LabelBinarizer()  
 label\_binarizer.fit(y\_test)  
  
 true\_labels = label\_binarizer.transform(y\_test)  
 pred\_labels = label\_binarizer.transform(y\_pred)  
  
 return roc\_auc\_score(true\_labels, pred\_labels, average=average)  
  
  
print(multiclass\_roc\_auc\_score(test\_labels, predicted\_classes))

0.8749576411817841

conf\_mat = confusion\_matrix(test\_labels, predicted\_classes)  
sns.heatmap(conf\_mat, annot=True, fmt='d', cmap='Blues', xticklabels=dx\_types, yticklabels=dx\_types)  
plt.ylabel('Actual')  
plt.xlabel('Predicted')  
plt.title('Confusion Matrix')  
plt.show()



# Hyperparameter Optimization

The hyperparameter optimization stage involves tuning the hyperparameters of the selected deep learning architecture to improve the model's performance and efficiency. In this section, we will use the Optuna library to perform hyperparameter optimization for the selected model on the HAM10000 dataset. The hyperparameters that will be optimized include:

* Batch Size: The number of samples in each batch for training steps.
* Number of Dense Layers: These are additional dense layers added to the model, each with a specified number of units, batch normalization, and dropout rate.
  + Dense Units: The number of units in each dense layer.
  + Batch Normalization: Whether to include batch normalization in each dense layer.
  + Dropout Rate: The dropout rate in each dense layer.
* Loss Function: The loss function used for training the model.
  + Categorical Crossentropy: The categorical crossentropy loss function for multi-class classification tasks.
    - Weighted Categorical Crossentropy: The weighted categorical crossentropy loss function for imbalanced datasets.
    - Non-Weighted Categorical Focal Loss: The categorical focal loss function for multi-class classification tasks.
    - Proposed by Lin et al. in the paper "Focal Loss for Dense Object Detection" (2017).
    - Its formula is given by the following equation:
  + Categorical Focal Loss: The categorical focal loss function for imbalanced datasets.
    - Alpha: is a hyperparameter that controls the balance between the loss of well-classified examples and poorly classified examples. The alpha parameter is used to assign different weights to the positive and negative classes, based on their importance.
    - Gamma: is a hyperparameter that controls the rate at which the loss decreases as the predicted probability of the true class increases. The gamma parameter is used to focus the model's attention on hard-to-classify examples, by increasing the loss for well-classified examples.
    - The focal loss function was proposed by Lin et al. in the paper "Focal Loss for Dense Object Detection" (2017).
    - Its formula is given by the following equation:
* Weight Decay: The weight decay parameter for the AdamW optimizer. Weight decay is a form of L2 regularization that penalizes large weights in the model, to prevent overfitting and improve the model's generalization.
* Learning Rate: The learning rate for the AdamW optimizer. Similarly, the learning rate is a hyperparameter that controls the rate at which the model learns during training. The learning rate is an important hyperparameter that can significantly impact the model's performance and convergence.
* Base Model Architecture: The base model architecture for feature extraction. The base model architecture will be used to extract features from the images, which will be combined with the metadata features for the final classification.
  + VGG16: The VGG16 architecture is a deep convolutional neural network that has been widely used for image classification tasks. The VGG16 architecture consists of multiple convolutional and pooling layers, followed by fully connected layers for classification.
  + VGG19: The VGG19 architecture is an extension of the VGG16 architecture, with additional convolutional and pooling layers for feature extraction. The VGG19 architecture has been shown to improve the model's performance on image classification tasks.
  + ResNet101V2: The ResNet101V2 architecture is a deep residual neural network that uses skip connections to improve the flow of information through the network. The ResNet101V2 architecture has been shown to improve the model's performance on image classification tasks.
  + InceptionResNetV2: The InceptionResNetV2 architecture is a hybrid model that combines the Inception and ResNet architectures. The InceptionResNetV2 architecture has been shown to improve the model's performance on image classification tasks.
  + Xception: The Xception architecture is a deep convolutional neural network that uses depthwise separable convolutions to improve the model's performance. The Xception architecture has been shown to improve the model's performance on image classification tasks.
  + MobileNetV2: The MobileNetV2 architecture is a lightweight convolutional neural network that uses depthwise separable convolutions to reduce the model's complexity. The MobileNetV2 architecture has been shown to improve the model's performance on image classification tasks.
* Attention Mechanism: The attention mechanism to include in the model. This attention mechanism were introduced with the main goal of improving the model's performance, and reduce the augmentation impact on the model's performance. The attention mechanisms that will be considered include:
  + Squeeze-and-Excitation Block: It employs a recursive filter selection strategy to enhance the network's expressivity, while not adding significant computational overhead. Basically recognize common patterns from separate areas, and combine these patterns together as a singular feature. The SE block recalibrates channel-wise feature responses by explicitly modeling interdependencies between channels. SE Block was proposed by Hu et al. in the paper "Squeeze-and-Excitation Networks" (2018).
  + CBAM Block: The Convolutional Block Attention Module (CBAM) is a lightweight attention mechanism that adaptively recalibrates channel-wise and spatial-wise feature responses by explicitly modeling interdependencies between channels and spatial locations. In other words, it considers the data in block sections, and give a level of importance for each, providing more attention to more important areas. CBAM Block was proposed by Woo et al. in the paper "CBAM: Convolutional Block Attention Module" (2018).

def squeeze\_excite\_block(input, ratio=16):  
 init = input  
 channel\_axis = 1 if K.image\_data\_format() == "channels\_first" else -1  
 filters = init.shape[channel\_axis]  
 se\_shape = (1, 1, filters)  
  
 se = GlobalAveragePooling2D()(init)  
 se = Reshape(se\_shape)(se)  
 se = Dense(filters // ratio, kernel\_initializer='he\_normal', use\_bias=False)(se)  
 se = BatchNormalization()(se)  
 se = Activation(tf.keras.activations.gelu)(se)  
 se = Dense(filters, kernel\_initializer='he\_normal', use\_bias=False)(se)  
 se = Activation('sigmoid')(se)  
  
 if K.image\_data\_format() == 'channels\_first':  
 se = Permute((3, 1, 2))(se)  
  
 x = Multiply()([init, se])  
 return x

def cbam\_block(input\_tensor, ratio=16, kernel\_size=7):  
 channel = GlobalAveragePooling2D()(input\_tensor)  
 channel = Dense(channel.shape[-1] // ratio, kernel\_initializer='he\_normal', use\_bias=True)(channel)  
 channel = BatchNormalization()(channel)  
 channel = Activation(tf.keras.activations.gelu)(channel)  
 channel = Dense(input\_tensor.shape[-1], kernel\_initializer='he\_normal', use\_bias=True)(channel)  
 channel = Activation('sigmoid')(channel)  
  
 if K.image\_data\_format() == 'channels\_first':  
 channel = Reshape((input\_tensor.shape[1], 1, 1))(channel)  
 else: # 'channels\_last'  
 channel = Reshape((1, 1, input\_tensor.shape[-1]))(channel)  
  
 channel\_attention = Multiply()([input\_tensor, channel])  
  
 spatial = Conv2D(1, kernel\_size, padding='same', use\_bias=True)(channel\_attention)  
 spatial = Activation('sigmoid')(spatial)  
 output = Multiply()([channel\_attention, spatial])  
  
 return output

* Use AMSGrad: Whether to use the AMSGrad variant of the Adam optimizer. The AMSGrad variant of the Adam optimizer has been shown to improve the convergence and generalization of deep learning models. However, in some scenarios, the AMSGrad variant may not provide significant improvements over the standard Adam optimizer, or may even lead to slower convergence.
* Pre-trained Weights: Whether to use pre-trained weights for the base model. The pre-trained weights are initialized with weights learned from a large dataset, such as ImageNet, and can improve the model's performance and convergence.
* Number of Layers to Unfreeze: The number of layers to unfreeze for fine-tuning. The number of layers to unfreeze can impact the model's performance and convergence during fine-tuning.

Optuna is a hyperparameter optimization framework that uses the Tree-structured Parzen Estimator (TPE) algorithm to search the hyperparameter space efficiently. The hyperparameter optimization process will be conducted in an iterative manner, following the CRISP-DM methodology, to refine and optimize the model pipeline. Based on the results of the hyperparameter optimization process, the best hyperparameters will be selected to train the final model on the HAM10000 dataset. The final model will be evaluated on the test dataset to assess its performance and generalization.

The hyperparameter optimization process will involve defining the objective function, which evaluates the model's performance based on the hyperparameters selected by Optuna. The objective function will train the model with the selected hyperparameters, evaluate its performance on the validation dataset, and return the validation loss as the metric to be minimized by Optuna. The hyperparameters will be selected based on their impact on the validation loss, with the goal of improving the model's performance and efficiency.

image\_size = (256, 256)  
unique\_classes = len(pd\_dataset['dx'].unique())  
  
  
def objective(trial):  
 batch\_size = trial.suggest\_int('batch', 32, 64)  
 num\_dense\_layers = trial.suggest\_int('num\_dense\_layers', 1, 3)  
 dense\_units = [trial.suggest\_int(f'dense\_units\_{i}', 32, 1024) for i in range(num\_dense\_layers)]  
 batch\_normalization\_layers = [trial.suggest\_categorical(f'batch\_normalization\_{i}', [True, False]) for i in  
 range(num\_dense\_layers)]  
 dropout\_rate\_layers = [trial.suggest\_float(f'dropout\_rate\_{i}', 0.3, 0.7) for i in range(num\_dense\_layers)]  
 loss\_type = trial.suggest\_categorical('loss\_type', ['focal\_loss', 'categorical\_crossentropy'])  
 alpha\_phase1 = trial.suggest\_float('alpha\_phase1', 0.2, 0.8) if loss\_type == 'focal\_loss' else None  
 gamma\_phase1 = trial.suggest\_float('gamma\_phase1', 1.0, 5.0) if loss\_type == 'focal\_loss' else None  
 use\_class\_weights = trial.suggest\_categorical('use\_class\_weights', [True, False])  
 weight\_decay\_phase1 = trial.suggest\_float('weight\_decay\_phase1', 1e-6, 1e-2, log=True)  
 learning\_rate\_phase1 = trial.suggest\_float('learning\_rate\_phase1', 1e-5, 1e-3, log=True)  
 learning\_rate\_phase2 = trial.suggest\_float('learning\_rate\_phase2', 1e-6, 1e-3, log=True)  
 alpha\_phase2 = trial.suggest\_float('alpha\_phase2', 0.2, 0.8) if loss\_type == 'focal\_loss' else None  
 gamma\_phase2 = trial.suggest\_float('gamma\_phase2', 1.0, 5.0) if loss\_type == 'focal\_loss' else None  
 weight\_decay\_phase2 = trial.suggest\_float('weight\_decay\_phase2', 1e-6, 1e-2, log=True)  
 base\_model\_architecture = trial.suggest\_categorical('base\_model\_architecture', ['VGG16', 'VGG19', 'ResNet101V2',  
 'InceptionResNetV2',  
 'Xception', 'MobileNetV2'])  
 attention\_mechanism = trial.suggest\_categorical('attention\_mechanism', ['SENet', 'CBAM', 'None'])  
 use\_amsgrad = trial.suggest\_categorical('use\_amsgrad', [True, False])  
 pre\_trained\_weights = trial.suggest\_categorical('pre\_trained\_weights', [True, False])  
  
  
 weight = 'imagenet' if pre\_trained\_weights else None  
 # Define the base model  
 if base\_model\_architecture == 'VGG16':  
 base\_model = tf.keras.applications.VGG16(include\_top=False, weights=weight, input\_shape=image\_size + (3,))  
 elif base\_model\_architecture == 'VGG19':  
 base\_model = tf.keras.applications.VGG19(include\_top=False, weights=weight, input\_shape=image\_size + (3,))  
 elif base\_model\_architecture == 'ResNet101V2':  
 base\_model = tf.keras.applications.ResNet101V2(include\_top=False, weights=weight, input\_shape=image\_size + (3,))  
 elif base\_model\_architecture == 'InceptionResNetV2':  
 base\_model = tf.keras.applications.InceptionResNetV2(include\_top=False, weights=weight, input\_shape=image\_size + (3,))  
 elif base\_model\_architecture == 'Xception':  
 base\_model = tf.keras.applications.Xception(include\_top=False, weights=weight, input\_shape=image\_size + (3,))  
 elif base\_model\_architecture == 'MobileNetV2':  
 base\_model = tf.keras.applications.MobileNetV2(include\_top=False, weights=weight, input\_shape=image\_size + (3,))  
 else:  
 raise ValueError(f"Invalid base model architecture: {base\_model\_architecture}")  
  
 # unfreeze layers for phase 2 (1 to total layers)  
 if pre\_trained\_weights:  
 num\_layers\_unfreeze = trial.suggest\_int('num\_layers\_unfreeze', 1, len(base\_model.layers))  
 for layer in base\_model.layers[-num\_layers\_unfreeze:]:  
 layer.trainable = False  
 else:  
 num\_layers\_unfreeze = len(base\_model.layers)  
 for layer in base\_model.layers:  
 layer.trainable = True  
  
 attention\_func = None  
 if attention\_mechanism == 'SENet':  
 attention\_func = squeeze\_excite\_block  
 elif attention\_mechanism == 'CBAM':  
 attention\_func = cbam\_block  
  
 # Model definition  
 metadata\_input\_shape = (len(dataset['train'][0]['localization']) + 2,)  
 metadata\_input = tf.keras.layers.Input(shape=metadata\_input\_shape, name='metadata\_input')  
 image\_input = tf.keras.layers.Input(shape=(256, 256, 3), name='image\_input')  
  
 # Apply the base model  
 x = base\_model(image\_input)  
  
 if pre\_trained\_weights:  
 for layer in base\_model.layers:  
 layer.trainable = False  
 else:  
 for layer in base\_model.layers:  
 layer.trainable = True  
  
 # Apply the attention mechanism if any  
 if attention\_func:  
 x = attention\_func(x)  
  
 x = Flatten()(x)  
 x = tf.keras.layers.concatenate([x, metadata\_input])  
  
 for i in range(num\_dense\_layers):  
 x = Dense(dense\_units[i])(x)  
 if batch\_normalization\_layers[i]:  
 x = BatchNormalization()(x)  
 x = Activation('swish')(x)  
 x = Dropout(dropout\_rate\_layers[i])(x)  
 predictions = Dense(len(pd\_dataset['dx'].unique()), activation='softmax')(x)  
  
 model = Model(inputs=[image\_input, metadata\_input], outputs=predictions)  
  
 train\_ds, val\_ds = create\_tf\_datasets(dataset, image\_size=(256, 256), batch\_size=batch\_size)  
  
 # Phase 1  
 opt = AdamW(weight\_decay=weight\_decay\_phase1, learning\_rate=learning\_rate\_phase1, amsgrad=use\_amsgrad, clipnorm=1.0)  
 if loss\_type == 'focal\_loss':  
 loss\_fn = CategoricalFocalCrossentropy(alpha=alpha\_phase1, gamma=gamma\_phase1)  
 class\_weights = None  
 else:  
 loss\_fn = 'categorical\_crossentropy'  
 if use\_class\_weights:  
 train\_labels = [item['dx'] for item in dataset['train']]  
 weights = compute\_class\_weight(class\_weight='balanced', classes=np.unique(train\_labels), y=train\_labels)  
 class\_weights = dict(enumerate(weights))  
 else:  
 class\_weights = None  
  
 model.compile(optimizer=opt, loss=loss\_fn, metrics=['accuracy'])  
  
 model.fit(train\_ds  
 , validation\_data=val\_ds  
 , epochs=3  
 , class\_weight=class\_weights  
 , callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True),  
 tf.keras.callbacks.ReduceLROnPlateau(monitor='val\_loss', factor=0.1, patience=3, cooldown=2)  
 ]  
 , use\_multiprocessing=True  
 )  
  
 # Phase 2  
 opt = AdamW(weight\_decay=weight\_decay\_phase2, learning\_rate=learning\_rate\_phase2, amsgrad=use\_amsgrad, clipnorm=1.0)  
 if loss\_type == 'focal\_loss':  
 loss\_fn = CategoricalFocalCrossentropy(alpha=alpha\_phase2, gamma=gamma\_phase2)  
 class\_weights = None  
 else:  
 loss\_fn = 'categorical\_crossentropy'  
 if use\_class\_weights:  
 weights = compute\_class\_weight(class\_weight='balanced', classes=np.unique(unique\_classes), y=train\_labels)  
 class\_weights = dict(enumerate(weights))  
 else:  
 class\_weights = None  
 model.compile(optimizer=opt, loss=loss\_fn, metrics=['accuracy'])  
  
 for layer in base\_model.layers[-num\_layers\_unfreeze:]:  
 layer.trainable = True  
   
 class BestValueTracker(tf.keras.callbacks.Callback):  
 def \_\_init\_\_(self):  
 super(BestValueTracker, self).\_\_init\_\_()  
 self.best\_val\_accuracy = 0  
 self.best\_epoch = 0  
   
 def on\_epoch\_end(self, epoch, logs=None):  
 current\_val\_accuracy = logs.get("val\_accuracy")  
 if current\_val\_accuracy > self.best\_val\_accuracy:  
 self.best\_val\_accuracy = current\_val\_accuracy  
 self.best\_epoch = epoch  
 best\_value\_tracker = BestValueTracker()  
  
 try:  
 model.fit(train\_ds  
 , validation\_data=val\_ds  
 , epochs=100  
 , class\_weight=class\_weights  
 , callbacks=[  
 TFKerasPruningCallback(trial, 'val\_accuracy'),  
 tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True,  
 start\_from\_epoch=15),  
 tf.keras.callbacks.ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=3, cooldown=2),  
 best\_value\_tracker  
 ]  
 , use\_multiprocessing=True  
 )  
 except optuna.exceptions.TrialPruned as e:  
 best\_accuracy = best\_value\_tracker.best\_val\_accuracy  
 best\_epoch = best\_value\_tracker.best\_epoch  
 print(f"Best validation accuracy before pruning: {best\_accuracy} at epoch {best\_epoch}")  
 if best\_accuracy < 0.70:  
 # Prune if best accuracy is less than 60%  
 raise e  
 else:  
 return best\_accuracy  
 # Evaluate  
 accuracy = model.evaluate(val\_ds, return\_dict=True)['accuracy']  
 print(f"accuracy: {accuracy}")  
  
 return accuracy

The optuna study will be created and the optimization process will be started. The optimization process will be conducted in an iterative manner, following the CRISP-DM methodology, to refine and optimize the model pipeline. The hyperparameters will be selected based on their impact on the validation loss, with the goal of improving the model's performance and efficiency.

Optuna support the use of database to store the optimization results, this can be done by setting the storage parameter to the database URL. The database URL can be a local file or a remote database. The database will store the optimization results, including the hyperparameters and the evaluation metrics, for further analysis and comparison.

The optuna study was created in an optimized environment, and the optimization process was started. The hyperparameters were selected based on their impact on the validation loss, with the goal of improving the model's performance and efficiency. The optimization process was conducted in an iterative manner, following the CRISP-DM methodology, to refine and optimize the model pipeline. The hyperparameters were selected based on their impact on the validation loss, with the goal of improving the model's performance and efficiency.

import os  
from pathlib import Path  
from dotenv import load\_dotenv  
  
# Load environment variables from .env file  
load\_dotenv()  
  
# Get the project root directory from the environment variable  
project\_root = Path(os.getenv('PROJECT\_ROOT'))  
  
study\_name = 'skin\_lesion\_classification\_with\_HAM10000\_dataset'  
storage\_name = f"sqlite:///../experiments/{study\_name}.db"  
print(storage\_name)

sqlite:///../experiments/skin\_lesion\_classification\_with\_HAM10000\_dataset.db

# Load the study from the database  
study = optuna.load\_study(study\_name=study\_name, storage=storage\_name)  
  
# Print the study summary  
print(f"Study direction: {study.direction}")  
  
# Print the best trial  
best\_trial = study.best\_trial  
print(f"Best trial: {best\_trial.number}\nParams: {best\_trial.params}")  
  
# Print all trials as a DataFrame  
df = study.trials\_dataframe()  
print(df.head())

Study direction: 2  
Best trial: 49  
Params: {'batch': 46, 'num\_dense\_layers': 2, 'dense\_units\_0': 667, 'dense\_units\_1': 515, 'batch\_normalization\_0': False, 'batch\_normalization\_1': True, 'dropout\_rate\_0': 0.362800923727492, 'dropout\_rate\_1': 0.35888302707521497, 'loss\_type': 'focal\_loss', 'alpha\_phase1': 0.48834840832599113, 'gamma\_phase1': 4.208586719042138, 'use\_class\_weights': True, 'weight\_decay\_phase1': 7.133686465531547e-05, 'learning\_rate\_phase1': 0.00041421607571416667, 'learning\_rate\_phase2': 4.427308850392865e-05, 'alpha\_phase2': 0.6047625441951349, 'gamma\_phase2': 1.019029980284372, 'weight\_decay\_phase2': 0.0012188886659969672, 'base\_model\_architecture': 'VGG19', 'attention\_mechanism': 'CBAM', 'use\_amsgrad': False, 'pre\_trained\_weights': True, 'num\_layers\_unfreeze': 5}  
 number value datetime\_start datetime\_complete \  
0 0 0.670546 2024-03-25 08:44:20.221133 2024-03-25 10:32:21.194798   
1 1 0.668539 2024-03-25 10:32:31.974155 2024-03-25 12:22:15.737592   
2 2 0.721910 2024-03-25 12:22:26.600509 2024-03-25 14:16:06.822478   
3 3 0.787319 2024-03-25 14:16:15.636035 2024-03-25 16:39:10.139323   
4 4 NaN 2024-03-25 16:39:19.735557 2024-03-25 16:49:12.730368   
  
 duration params\_alpha\_phase1 params\_alpha\_phase2 \  
0 0 days 01:48:00.973665 0.384711 0.535273   
1 0 days 01:49:43.763437 0.328690 0.483442   
2 0 days 01:53:40.221969 0.638033 0.285963   
3 0 days 02:22:54.503288 0.506607 0.360034   
4 0 days 00:09:52.994811 NaN NaN   
  
 params\_attention\_mechanism params\_base\_model\_architecture params\_batch \  
0 None InceptionResNetV2 53   
1 None VGG16 47   
2 CBAM VGG16 26   
3 None VGG19 50   
4 SENet VGG16 56   
  
 ... params\_loss\_type params\_num\_dense\_layers \  
0 ... focal\_loss 3   
1 ... focal\_loss 3   
2 ... focal\_loss 3   
3 ... focal\_loss 2   
4 ... categorical\_crossentropy 3   
  
 params\_num\_layers\_unfreeze params\_pre\_trained\_weights params\_use\_amsgrad \  
0 635.0 True True   
1 NaN False False   
2 11.0 True True   
3 16.0 True False   
4 NaN False True   
  
 params\_use\_class\_weights params\_weight\_decay\_phase1 \  
0 False 0.000029   
1 True 0.000559   
2 True 0.000038   
3 False 0.005147   
4 True 0.005779   
  
 params\_weight\_decay\_phase2 system\_attrs\_fixed\_params state   
0 0.000076 NaN COMPLETE   
1 0.000010 NaN COMPLETE   
2 0.000108 NaN COMPLETE   
3 0.000002 NaN COMPLETE   
4 0.002519 NaN FAIL   
  
[5 rows x 33 columns]

# Plot optimization history  
opt\_hist = optuna.visualization.plot\_optimization\_history(study)  
opt\_hist.update\_layout(autosize=False, width=1200, height=800) # You can adjust the width and height as needed  
pio.show(opt\_hist)

{"layout":{"height":800,"autosize":false,"yaxis":{"title":{"text":"Objective Value"}},"xaxis":{"title":{"text":"Trial"}},"width":1200,"title":{"text":"Optimization History Plot"},"template":{"layout":{"annotationdefaults":{"arrowwidth":1,"arrowhead":0,"arrowcolor":"#f2f5fa"},"yaxis":{"zerolinecolor":"#283442","zerolinewidth":2,"gridcolor":"#283442","title":{"standoff":15},"linecolor":"#506784","ticks":"","automargin":true},"sliderdefaults":{"borderwidth":1,"bgcolor":"#C8D4E3","tickwidth":0,"bordercolor":"rgb(17,17,17)"},"hovermode":"closest","hoverlabel":{"align":"left"},"polar":{"bgcolor":"rgb(17,17,17)","angularaxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""},"radialaxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""}},"xaxis":{"zerolinecolor":"#283442","zerolinewidth":2,"gridcolor":"#283442","title":{"standoff":15},"linecolor":"#506784","ticks":"","automargin":true},"coloraxis":{"colorbar":{"outlinewidth":0,"ticks":""}},"paper\_bgcolor":"rgb(17,17,17)","ternary":{"bgcolor":"rgb(17,17,17)","caxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""},"baxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""},"aaxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""}},"shapedefaults":{"line":{"color":"#f2f5fa"}},"autotypenumbers":"strict","updatemenudefaults":{"borderwidth":0,"bgcolor":"#506784"},"colorway":["#636efa","#EF553B","#00cc96","#ab63fa","#FFA15A","#19d3f3","#FF6692","#B6E880","#FF97FF","#FECB52"],"geo":{"subunitcolor":"#506784","bgcolor":"rgb(17,17,17)","showlakes":true,"lakecolor":"rgb(17,17,17)","showland":true,"landcolor":"rgb(17,17,17)"},"colorscale":{"sequentialminus":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"diverging":[[0,"#8e0152"],[0.1,"#c51b7d"],[0.2,"#de77ae"],[0.3,"#f1b6da"],[0.4,"#fde0ef"],[0.5,"#f7f7f7"],[0.6,"#e6f5d0"],[0.7,"#b8e186"],[0.8,"#7fbc41"],[0.9,"#4d9221"],[1,"#276419"]],"sequential":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]]},"plot\_bgcolor":"rgb(17,17,17)","title":{"x":5.0e-2},"font":{"color":"#f2f5fa"},"scene":{"yaxis":{"backgroundcolor":"rgb(17,17,17)","zerolinecolor":"#C8D4E3","gridcolor":"#506784","showbackground":true,"linecolor":"#506784","ticks":"","gridwidth":2},"xaxis":{"backgroundcolor":"rgb(17,17,17)","zerolinecolor":"#C8D4E3","gridcolor":"#506784","showbackground":true,"linecolor":"#506784","ticks":"","gridwidth":2},"zaxis":{"backgroundcolor":"rgb(17,17,17)","zerolinecolor":"#C8D4E3","gridcolor":"#506784","showbackground":true,"linecolor":"#506784","ticks":"","gridwidth":2}},"mapbox":{"style":"dark"}},"data":{"mesh3d":[{"colorbar":{"outlinewidth":0,"ticks":""},"type":"mesh3d"}],"scattercarpet":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scattercarpet"}],"heatmapgl":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"heatmapgl"}],"parcoords":[{"line":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"parcoords"}],"histogram2dcontour":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"histogram2dcontour"}],"contour":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"contour"}],"histogram2d":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"histogram2d"}],"barpolar":[{"marker":{"pattern":{"size":10,"solidity":0.2,"fillmode":"overlay"},"line":{"color":"rgb(17,17,17)","width":0.5}},"type":"barpolar"}],"carpet":[{"baxis":{"minorgridcolor":"#506784","endlinecolor":"#A2B1C6","gridcolor":"#506784","startlinecolor":"#A2B1C6","linecolor":"#506784"},"aaxis":{"minorgridcolor":"#506784","endlinecolor":"#A2B1C6","gridcolor":"#506784","startlinecolor":"#A2B1C6","linecolor":"#506784"},"type":"carpet"}],"heatmap":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"heatmap"}],"pie":[{"type":"pie","automargin":true}],"choropleth":[{"colorbar":{"outlinewidth":0,"ticks":""},"type":"choropleth"}],"contourcarpet":[{"colorbar":{"outlinewidth":0,"ticks":""},"type":"contourcarpet"}],"scattermapbox":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scattermapbox"}],"scattergl":[{"marker":{"line":{"color":"#283442"}},"type":"scattergl"}],"scatterpolargl":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scatterpolargl"}],"surface":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"surface"}],"histogram":[{"marker":{"pattern":{"size":10,"solidity":0.2,"fillmode":"overlay"}},"type":"histogram"}],"scattergeo":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scattergeo"}],"scatter":[{"marker":{"line":{"color":"#283442"}},"type":"scatter"}],"table":[{"cells":{"line":{"color":"rgb(17,17,17)"},"fill":{"color":"#506784"}},"header":{"line":{"color":"rgb(17,17,17)"},"fill":{"color":"#2a3f5f"}},"type":"table"}],"scatterpolar":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scatterpolar"}],"scatterternary":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scatterternary"}],"bar":[{"error\_y":{"color":"#f2f5fa"},"error\_x":{"color":"#f2f5fa"},"marker":{"pattern":{"size":10,"solidity":0.2,"fillmode":"overlay"},"line":{"color":"rgb(17,17,17)","width":0.5}},"type":"bar"}],"scatter3d":[{"line":{"colorbar":{"outlinewidth":0,"ticks":""}},"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scatter3d"}]}}},"config":{"plotlyServerURL":"https://plot.ly"},"data":[{"mode":"markers","name":"Objective Value","x":[0,1,2,3,7,10,11,13,14,16,17,18,19,20,21,23,25,26,27,29,30,31,32,34,35,36,37,38,39,40,41,42,43,44,46,48,49,50,51,52,55,57,59,67,68,69,71,72,73,74,155,156,157,158,159,160,161,162,164,165,166,167,168,169,170,171,173,174,178,179,180,181,182,183,184,185,186,187,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208],"type":"scatter","y":[0.670545756816864,0.6685393452644348,0.7219101190567017,0.7873194217681885,0.8330658078193665,0.7106741666793823,0.7118780016899109,0.9081059098243713,0.6685393452644348,0.11235955357551575,0.728330671787262,0.9265649914741516,0.8350722193717957,0.944622814655304,0.9406099319458008,0.790930986404419,0.9478330612182617,0.942215085029602,0.9329855442047119,0.9197431802749634,0.94261634349823,0.925361156463623,0.9060994982719421,0.7126805782318115,0.7760834693908691,0.7776886224746704,0.7066613435745239,0.9249598979949951,0.7684590816497803,0.9261637330055237,0.9069020748138428,0.9514446258544922,0.6685393452644348,0.9498394727706909,0.8804173469543457,0.6685393452644348,0.9558587670326233,0.8619582653045654,0.9486356377601624,0.9418138265609741,0.7772873044013977,0.7897271513938904,0.7315409183502197,0.7692616581916809,0.7303370833396912,0.7965489625930786,0.7564205527305603,0.7086677551269531,0.6215890645980835,0.6685393452644348,0.6793739795684814,0.8479133248329163,0.8483145833015442,0.846308171749115,0.812600314617157,0.7997592091560364,0.853130042552948,0.8258426785469055,0.790930986404419,0.8471107482910156,0.8382825255393982,0.7905296683311462,0.7969502210617065,0.8142054677009583,0.841091513633728,0.8238362669944763,0.6685393452644348,0.9289727210998535,0.6685393452644348,0.9498394727706909,0.7776886224746704,0.8037720918655396,0.9462279081344604,0.7692616581916809,0.8198234438896179,0.9474318027496338,0.477126806974411,0.8374799489974976,0.8218298554420471,0.6653290390968323,0.745184600353241,0.9361958503723145,0.7813001871109009,0.8523274660110474,0.9229534268379211,0.6601123809814453,0.5100321173667908,0.94261634349823,0.6685393452644348,0.7893258333206177,0.5943017601966858,0.7853130102157593,0.7857142686843872,0.9341893792152405,0.7845104336738586,0.7796950340270996]},{"mode":"lines","name":"Best Value","x":[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208,209],"type":"scatter","y":[0.670545756816864,0.670545756816864,0.7219101190567017,0.7873194217681885,0.7873194217681885,0.7873194217681885,0.7873194217681885,0.8330658078193665,0.8330658078193665,0.8330658078193665,0.8330658078193665,0.8330658078193665,0.8330658078193665,0.9081059098243713,0.9081059098243713,0.9081059098243713,0.9081059098243713,0.9081059098243713,0.9265649914741516,0.9265649914741516,0.944622814655304,0.944622814655304,0.944622814655304,0.944622814655304,0.944622814655304,0.9478330612182617,0.9478330612182617,0.9478330612182617,0.9478330612182617,0.9478330612182617,0.9478330612182617,0.9478330612182617,0.9478330612182617,0.9478330612182617,0.9478330612182617,0.9478330612182617,0.9478330612182617,0.9478330612182617,0.9478330612182617,0.9478330612182617,0.9478330612182617,0.9478330612182617,0.9514446258544922,0.9514446258544922,0.9514446258544922,0.9514446258544922,0.9514446258544922,0.9514446258544922,0.9514446258544922,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233,0.9558587670326233]},{"showlegend":false,"mode":"markers","name":"Infeasible Trial","marker":{"color":"#cccccc"},"x":[],"type":"scatter","y":[]}]}

# Plot parameter importance  
param\_imp = optuna.visualization.plot\_param\_importances(study)  
param\_imp.update\_layout(autosize=False, width=1200, height=800) # Adjust the width and height  
pio.show(param\_imp)

{"layout":{"height":800,"autosize":false,"yaxis":{"title":{"text":"Hyperparameter"}},"xaxis":{"title":{"text":"Hyperparameter Importance"}},"width":1200,"title":{"text":"Hyperparameter Importances"},"template":{"layout":{"annotationdefaults":{"arrowwidth":1,"arrowhead":0,"arrowcolor":"#f2f5fa"},"yaxis":{"zerolinecolor":"#283442","zerolinewidth":2,"gridcolor":"#283442","title":{"standoff":15},"linecolor":"#506784","ticks":"","automargin":true},"sliderdefaults":{"borderwidth":1,"bgcolor":"#C8D4E3","tickwidth":0,"bordercolor":"rgb(17,17,17)"},"hovermode":"closest","hoverlabel":{"align":"left"},"polar":{"bgcolor":"rgb(17,17,17)","angularaxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""},"radialaxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""}},"xaxis":{"zerolinecolor":"#283442","zerolinewidth":2,"gridcolor":"#283442","title":{"standoff":15},"linecolor":"#506784","ticks":"","automargin":true},"coloraxis":{"colorbar":{"outlinewidth":0,"ticks":""}},"paper\_bgcolor":"rgb(17,17,17)","ternary":{"bgcolor":"rgb(17,17,17)","caxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""},"baxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""},"aaxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""}},"shapedefaults":{"line":{"color":"#f2f5fa"}},"autotypenumbers":"strict","updatemenudefaults":{"borderwidth":0,"bgcolor":"#506784"},"colorway":["#636efa","#EF553B","#00cc96","#ab63fa","#FFA15A","#19d3f3","#FF6692","#B6E880","#FF97FF","#FECB52"],"geo":{"subunitcolor":"#506784","bgcolor":"rgb(17,17,17)","showlakes":true,"lakecolor":"rgb(17,17,17)","showland":true,"landcolor":"rgb(17,17,17)"},"colorscale":{"sequentialminus":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"diverging":[[0,"#8e0152"],[0.1,"#c51b7d"],[0.2,"#de77ae"],[0.3,"#f1b6da"],[0.4,"#fde0ef"],[0.5,"#f7f7f7"],[0.6,"#e6f5d0"],[0.7,"#b8e186"],[0.8,"#7fbc41"],[0.9,"#4d9221"],[1,"#276419"]],"sequential":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]]},"plot\_bgcolor":"rgb(17,17,17)","title":{"x":5.0e-2},"font":{"color":"#f2f5fa"},"scene":{"yaxis":{"backgroundcolor":"rgb(17,17,17)","zerolinecolor":"#C8D4E3","gridcolor":"#506784","showbackground":true,"linecolor":"#506784","ticks":"","gridwidth":2},"xaxis":{"backgroundcolor":"rgb(17,17,17)","zerolinecolor":"#C8D4E3","gridcolor":"#506784","showbackground":true,"linecolor":"#506784","ticks":"","gridwidth":2},"zaxis":{"backgroundcolor":"rgb(17,17,17)","zerolinecolor":"#C8D4E3","gridcolor":"#506784","showbackground":true,"linecolor":"#506784","ticks":"","gridwidth":2}},"mapbox":{"style":"dark"}},"data":{"mesh3d":[{"colorbar":{"outlinewidth":0,"ticks":""},"type":"mesh3d"}],"scattercarpet":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scattercarpet"}],"heatmapgl":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"heatmapgl"}],"parcoords":[{"line":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"parcoords"}],"histogram2dcontour":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"histogram2dcontour"}],"contour":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"contour"}],"histogram2d":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"histogram2d"}],"barpolar":[{"marker":{"pattern":{"size":10,"solidity":0.2,"fillmode":"overlay"},"line":{"color":"rgb(17,17,17)","width":0.5}},"type":"barpolar"}],"carpet":[{"baxis":{"minorgridcolor":"#506784","endlinecolor":"#A2B1C6","gridcolor":"#506784","startlinecolor":"#A2B1C6","linecolor":"#506784"},"aaxis":{"minorgridcolor":"#506784","endlinecolor":"#A2B1C6","gridcolor":"#506784","startlinecolor":"#A2B1C6","linecolor":"#506784"},"type":"carpet"}],"heatmap":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"heatmap"}],"pie":[{"type":"pie","automargin":true}],"choropleth":[{"colorbar":{"outlinewidth":0,"ticks":""},"type":"choropleth"}],"contourcarpet":[{"colorbar":{"outlinewidth":0,"ticks":""},"type":"contourcarpet"}],"scattermapbox":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scattermapbox"}],"scattergl":[{"marker":{"line":{"color":"#283442"}},"type":"scattergl"}],"scatterpolargl":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scatterpolargl"}],"surface":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"surface"}],"histogram":[{"marker":{"pattern":{"size":10,"solidity":0.2,"fillmode":"overlay"}},"type":"histogram"}],"scattergeo":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scattergeo"}],"scatter":[{"marker":{"line":{"color":"#283442"}},"type":"scatter"}],"table":[{"cells":{"line":{"color":"rgb(17,17,17)"},"fill":{"color":"#506784"}},"header":{"line":{"color":"rgb(17,17,17)"},"fill":{"color":"#2a3f5f"}},"type":"table"}],"scatterpolar":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scatterpolar"}],"scatterternary":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scatterternary"}],"bar":[{"error\_y":{"color":"#f2f5fa"},"error\_x":{"color":"#f2f5fa"},"marker":{"pattern":{"size":10,"solidity":0.2,"fillmode":"overlay"},"line":{"color":"rgb(17,17,17)","width":0.5}},"type":"bar"}],"scatter3d":[{"line":{"colorbar":{"outlinewidth":0,"ticks":""}},"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scatter3d"}]}}},"config":{"plotlyServerURL":"https://plot.ly"},"data":[{"cliponaxis":false,"text":["<0.01","<0.01","<0.01","<0.01","<0.01","0.01","0.02","0.02","0.02","0.02","0.05","0.09","0.34","0.42"],"hovertemplate":["weight\_decay\_phase1 (FloatDistribution): 0.0016754750433671073<extra></extra>","num\_dense\_layers (IntDistribution): 0.0021347175898909915<extra></extra>","learning\_rate\_phase1 (FloatDistribution): 0.0023525094276185487<extra></extra>","attention\_mechanism (CategoricalDistribution): 0.0033821611636507074<extra></extra>","use\_class\_weights (CategoricalDistribution): 0.0039613866828618356<extra></extra>","weight\_decay\_phase2 (FloatDistribution): 0.010528400160343959<extra></extra>","learning\_rate\_phase2 (FloatDistribution): 0.01675958164567221<extra></extra>","loss\_type (CategoricalDistribution): 0.01760925784743577<extra></extra>","pre\_trained\_weights (CategoricalDistribution): 0.01821924567063942<extra></extra>","use\_amsgrad (CategoricalDistribution): 0.023282181371896096<extra></extra>","dropout\_rate\_0 (FloatDistribution): 0.04927626988630147<extra></extra>","batch\_normalization\_0 (CategoricalDistribution): 0.09430281081127649<extra></extra>","base\_model\_architecture (CategoricalDistribution): 0.3404018334939282<extra></extra>","dense\_units\_0 (IntDistribution): 0.4161141692051173<extra></extra>"],"textposition":"outside","name":"Objective Value","x":[1.6754750433671073e-3,2.1347175898909915e-3,2.3525094276185487e-3,3.3821611636507074e-3,3.9613866828618356e-3,1.0528400160343959e-2,1.675958164567221e-2,1.760925784743577e-2,1.821924567063942e-2,2.3282181371896096e-2,4.927626988630147e-2,9.430281081127649e-2,0.3404018334939282,0.4161141692051173],"type":"bar","orientation":"h","y":["weight\_decay\_phase1","num\_dense\_layers","learning\_rate\_phase1","attention\_mechanism","use\_class\_weights","weight\_decay\_phase2","learning\_rate\_phase2","loss\_type","pre\_trained\_weights","use\_amsgrad","dropout\_rate\_0","batch\_normalization\_0","base\_model\_architecture","dense\_units\_0"]}]}

from plotly.subplots import make\_subplots  
import optuna  
  
# Get parameter importance  
param\_importance = optuna.importance.get\_param\_importances(study)  
# Sort parameters by importance  
sorted\_params = sorted(param\_importance.items(), key=lambda x: x[1], reverse=True)  
  
# Determine the number of parameters to plot (limited by max\_params)  
max\_params = 10 # You can tune this value  
n\_params = min(max\_params, len(sorted\_params))  
  
fig = make\_subplots(rows=n\_params, cols=1, subplot\_titles=[x[0] for x in sorted\_params[:n\_params]])  
  
for i in range(n\_params):  
 # Get parameter name  
 param\_name = sorted\_params[i][0]  
 # Generate slice plot for this parameter  
 slice\_plot = optuna.visualization.plot\_slice(  
 study,  
 params=[param\_name]  
 )  
 # Get data from slice plot  
 trace = slice\_plot['data'][0]  
 # Add trace to subplot  
 fig.add\_trace(  
 trace,  
 row=i + 1,  
 col=1  
 )  
  
fig.update\_layout(height=400 \* n\_params, width=1200, title\_text="Optuna Study Hyperparameter Importance")  
pio.show(fig)

{"layout":{"yaxis6":{"domain":[0.42000000000000004,0.47500000000000003],"anchor":"x6"},"annotations":[{"showarrow":false,"text":"dense\_units\_0","xref":"paper","x":0.5,"yanchor":"bottom","yref":"paper","font":{"size":16},"xanchor":"center","y":1.0},{"showarrow":false,"text":"base\_model\_architecture","xref":"paper","x":0.5,"yanchor":"bottom","yref":"paper","font":{"size":16},"xanchor":"center","y":0.8950000000000001},{"showarrow":false,"text":"batch\_normalization\_0","xref":"paper","x":0.5,"yanchor":"bottom","yref":"paper","font":{"size":16},"xanchor":"center","y":0.7900000000000001},{"showarrow":false,"text":"dropout\_rate\_0","xref":"paper","x":0.5,"yanchor":"bottom","yref":"paper","font":{"size":16},"xanchor":"center","y":0.6850000000000002},{"showarrow":false,"text":"weight\_decay\_phase2","xref":"paper","x":0.5,"yanchor":"bottom","yref":"paper","font":{"size":16},"xanchor":"center","y":0.5800000000000001},{"showarrow":false,"text":"use\_amsgrad","xref":"paper","x":0.5,"yanchor":"bottom","yref":"paper","font":{"size":16},"xanchor":"center","y":0.47500000000000003},{"showarrow":false,"text":"pre\_trained\_weights","xref":"paper","x":0.5,"yanchor":"bottom","yref":"paper","font":{"size":16},"xanchor":"center","y":0.37000000000000005},{"showarrow":false,"text":"loss\_type","xref":"paper","x":0.5,"yanchor":"bottom","yref":"paper","font":{"size":16},"xanchor":"center","y":0.265},{"showarrow":false,"text":"num\_dense\_layers","xref":"paper","x":0.5,"yanchor":"bottom","yref":"paper","font":{"size":16},"xanchor":"center","y":0.16000000000000003},{"showarrow":false,"text":"learning\_rate\_phase2","xref":"paper","x":0.5,"yanchor":"bottom","yref":"paper","font":{"size":16},"xanchor":"center","y":5.500000000000001e-2}],"xaxis4":{"domain":[0.0,1.0],"anchor":"y4"},"height":4000,"xaxis9":{"domain":[0.0,1.0],"anchor":"y9"},"yaxis":{"domain":[0.9450000000000001,1.0],"anchor":"x"},"yaxis7":{"domain":[0.31500000000000006,0.37000000000000005],"anchor":"x7"},"xaxis3":{"domain":[0.0,1.0],"anchor":"y3"},"xaxis8":{"domain":[0.0,1.0],"anchor":"y8"},"xaxis":{"domain":[0.0,1.0],"anchor":"y"},"width":1200,"yaxis4":{"domain":[0.6300000000000001,0.6850000000000002],"anchor":"x4"},"xaxis2":{"domain":[0.0,1.0],"anchor":"y2"},"xaxis7":{"domain":[0.0,1.0],"anchor":"y7"},"yaxis5":{"domain":[0.525,0.5800000000000001],"anchor":"x5"},"yaxis8":{"domain":[0.21000000000000002,0.265],"anchor":"x8"},"xaxis10":{"domain":[0.0,1.0],"anchor":"y10"},"xaxis6":{"domain":[0.0,1.0],"anchor":"y6"},"yaxis2":{"domain":[0.8400000000000001,0.8950000000000001],"anchor":"x2"},"title":{"text":"Optuna Study Hyperparameter Importance"},"template":{"layout":{"annotationdefaults":{"arrowwidth":1,"arrowhead":0,"arrowcolor":"#f2f5fa"},"yaxis":{"zerolinecolor":"#283442","zerolinewidth":2,"gridcolor":"#283442","title":{"standoff":15},"linecolor":"#506784","ticks":"","automargin":true},"sliderdefaults":{"borderwidth":1,"bgcolor":"#C8D4E3","tickwidth":0,"bordercolor":"rgb(17,17,17)"},"hovermode":"closest","hoverlabel":{"align":"left"},"polar":{"bgcolor":"rgb(17,17,17)","angularaxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""},"radialaxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""}},"xaxis":{"zerolinecolor":"#283442","zerolinewidth":2,"gridcolor":"#283442","title":{"standoff":15},"linecolor":"#506784","ticks":"","automargin":true},"coloraxis":{"colorbar":{"outlinewidth":0,"ticks":""}},"paper\_bgcolor":"rgb(17,17,17)","ternary":{"bgcolor":"rgb(17,17,17)","caxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""},"baxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""},"aaxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""}},"shapedefaults":{"line":{"color":"#f2f5fa"}},"autotypenumbers":"strict","updatemenudefaults":{"borderwidth":0,"bgcolor":"#506784"},"colorway":["#636efa","#EF553B","#00cc96","#ab63fa","#FFA15A","#19d3f3","#FF6692","#B6E880","#FF97FF","#FECB52"],"geo":{"subunitcolor":"#506784","bgcolor":"rgb(17,17,17)","showlakes":true,"lakecolor":"rgb(17,17,17)","showland":true,"landcolor":"rgb(17,17,17)"},"colorscale":{"sequentialminus":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"diverging":[[0,"#8e0152"],[0.1,"#c51b7d"],[0.2,"#de77ae"],[0.3,"#f1b6da"],[0.4,"#fde0ef"],[0.5,"#f7f7f7"],[0.6,"#e6f5d0"],[0.7,"#b8e186"],[0.8,"#7fbc41"],[0.9,"#4d9221"],[1,"#276419"]],"sequential":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]]},"plot\_bgcolor":"rgb(17,17,17)","title":{"x":5.0e-2},"font":{"color":"#f2f5fa"},"scene":{"yaxis":{"backgroundcolor":"rgb(17,17,17)","zerolinecolor":"#C8D4E3","gridcolor":"#506784","showbackground":true,"linecolor":"#506784","ticks":"","gridwidth":2},"xaxis":{"backgroundcolor":"rgb(17,17,17)","zerolinecolor":"#C8D4E3","gridcolor":"#506784","showbackground":true,"linecolor":"#506784","ticks":"","gridwidth":2},"zaxis":{"backgroundcolor":"rgb(17,17,17)","zerolinecolor":"#C8D4E3","gridcolor":"#506784","showbackground":true,"linecolor":"#506784","ticks":"","gridwidth":2}},"mapbox":{"style":"dark"}},"data":{"mesh3d":[{"colorbar":{"outlinewidth":0,"ticks":""},"type":"mesh3d"}],"scattercarpet":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scattercarpet"}],"heatmapgl":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"heatmapgl"}],"parcoords":[{"line":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"parcoords"}],"histogram2dcontour":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"histogram2dcontour"}],"contour":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"contour"}],"histogram2d":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"histogram2d"}],"barpolar":[{"marker":{"pattern":{"size":10,"solidity":0.2,"fillmode":"overlay"},"line":{"color":"rgb(17,17,17)","width":0.5}},"type":"barpolar"}],"carpet":[{"baxis":{"minorgridcolor":"#506784","endlinecolor":"#A2B1C6","gridcolor":"#506784","startlinecolor":"#A2B1C6","linecolor":"#506784"},"aaxis":{"minorgridcolor":"#506784","endlinecolor":"#A2B1C6","gridcolor":"#506784","startlinecolor":"#A2B1C6","linecolor":"#506784"},"type":"carpet"}],"heatmap":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"heatmap"}],"pie":[{"type":"pie","automargin":true}],"choropleth":[{"colorbar":{"outlinewidth":0,"ticks":""},"type":"choropleth"}],"contourcarpet":[{"colorbar":{"outlinewidth":0,"ticks":""},"type":"contourcarpet"}],"scattermapbox":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scattermapbox"}],"scattergl":[{"marker":{"line":{"color":"#283442"}},"type":"scattergl"}],"scatterpolargl":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scatterpolargl"}],"surface":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"surface"}],"histogram":[{"marker":{"pattern":{"size":10,"solidity":0.2,"fillmode":"overlay"}},"type":"histogram"}],"scattergeo":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scattergeo"}],"scatter":[{"marker":{"line":{"color":"#283442"}},"type":"scatter"}],"table":[{"cells":{"line":{"color":"rgb(17,17,17)"},"fill":{"color":"#506784"}},"header":{"line":{"color":"rgb(17,17,17)"},"fill":{"color":"#2a3f5f"}},"type":"table"}],"scatterpolar":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scatterpolar"}],"scatterternary":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scatterternary"}],"bar":[{"error\_y":{"color":"#f2f5fa"},"error\_x":{"color":"#f2f5fa"},"marker":{"pattern":{"size":10,"solidity":0.2,"fillmode":"overlay"},"line":{"color":"rgb(17,17,17)","width":0.5}},"type":"bar"}],"scatter3d":[{"line":{"colorbar":{"outlinewidth":0,"ticks":""}},"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scatter3d"}]}},"yaxis10":{"domain":[0.0,5.500000000000001e-2],"anchor":"x10"},"yaxis9":{"domain":[0.10500000000000001,0.16000000000000003],"anchor":"x9"},"xaxis5":{"domain":[0.0,1.0],"anchor":"y5"},"yaxis3":{"domain":[0.7350000000000001,0.7900000000000001],"anchor":"x3"}},"config":{"plotlyServerURL":"https://plot.ly"},"data":[{"yaxis":"y","showlegend":false,"mode":"markers","xaxis":"x","name":"Feasible Trial","marker":{"colorbar":{"xpad":40,"x":1.0,"title":{"text":"Trial"}},"color":[0,1,2,3,7,10,11,13,14,16,17,18,19,20,21,23,25,26,27,29,30,31,32,34,35,36,37,38,39,40,41,42,43,44,46,48,49,50,51,52,55,57,59,67,68,69,71,72,73,74,155,156,157,158,159,160,161,162,164,165,166,167,168,169,170,171,173,174,178,179,180,181,182,183,184,185,186,187,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208],"line":{"color":"Grey","width":0.5},"colorscale":[[0.0,"rgb(247,251,255)"],[0.125,"rgb(222,235,247)"],[0.25,"rgb(198,219,239)"],[0.375,"rgb(158,202,225)"],[0.5,"rgb(107,174,214)"],[0.625,"rgb(66,146,198)"],[0.75,"rgb(33,113,181)"],[0.875,"rgb(8,81,156)"],[1.0,"rgb(8,48,107)"]]},"x":[95,476,784,352,232,881,587,874,342,51,184,623,632,999,1016,995,1023,773,789,923,730,693,484,824,942,697,834,741,953,519,771,658,432,593,607,548,667,296,669,569,657,492,151,701,974,661,534,1008,810,628,1023,704,748,580,851,672,914,721,623,779,971,754,724,800,873,675,611,509,553,688,648,684,759,597,642,756,760,721,992,702,560,791,732,674,634,862,611,945,896,814,694,981,942,921,739,961],"type":"scatter","y":[0.670545756816864,0.6685393452644348,0.7219101190567017,0.7873194217681885,0.8330658078193665,0.7106741666793823,0.7118780016899109,0.9081059098243713,0.6685393452644348,0.11235955357551575,0.728330671787262,0.9265649914741516,0.8350722193717957,0.944622814655304,0.9406099319458008,0.790930986404419,0.9478330612182617,0.942215085029602,0.9329855442047119,0.9197431802749634,0.94261634349823,0.925361156463623,0.9060994982719421,0.7126805782318115,0.7760834693908691,0.7776886224746704,0.7066613435745239,0.9249598979949951,0.7684590816497803,0.9261637330055237,0.9069020748138428,0.9514446258544922,0.6685393452644348,0.9498394727706909,0.8804173469543457,0.6685393452644348,0.9558587670326233,0.8619582653045654,0.9486356377601624,0.9418138265609741,0.7772873044013977,0.7897271513938904,0.7315409183502197,0.7692616581916809,0.7303370833396912,0.7965489625930786,0.7564205527305603,0.7086677551269531,0.6215890645980835,0.6685393452644348,0.6793739795684814,0.8479133248329163,0.8483145833015442,0.846308171749115,0.812600314617157,0.7997592091560364,0.853130042552948,0.8258426785469055,0.790930986404419,0.8471107482910156,0.8382825255393982,0.7905296683311462,0.7969502210617065,0.8142054677009583,0.841091513633728,0.8238362669944763,0.6685393452644348,0.9289727210998535,0.6685393452644348,0.9498394727706909,0.7776886224746704,0.8037720918655396,0.9462279081344604,0.7692616581916809,0.8198234438896179,0.9474318027496338,0.477126806974411,0.8374799489974976,0.8218298554420471,0.6653290390968323,0.745184600353241,0.9361958503723145,0.7813001871109009,0.8523274660110474,0.9229534268379211,0.6601123809814453,0.5100321173667908,0.94261634349823,0.6685393452644348,0.7893258333206177,0.5943017601966858,0.7853130102157593,0.7857142686843872,0.9341893792152405,0.7845104336738586,0.7796950340270996]},{"yaxis":"y2","showlegend":false,"mode":"markers","xaxis":"x2","name":"Feasible Trial","marker":{"colorbar":{"xpad":40,"x":1.0,"title":{"text":"Trial"}},"color":[0,1,2,3,7,10,11,13,14,16,17,18,19,20,21,23,25,26,27,29,30,31,32,34,35,36,37,38,39,40,41,42,43,44,46,48,49,50,51,52,55,57,59,67,68,69,71,72,73,74,155,156,157,158,159,160,161,162,164,165,166,167,168,169,170,171,173,174,178,179,180,181,182,183,184,185,186,187,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208],"line":{"color":"Grey","width":0.5},"colorscale":[[0.0,"rgb(247,251,255)"],[0.125,"rgb(222,235,247)"],[0.25,"rgb(198,219,239)"],[0.375,"rgb(158,202,225)"],[0.5,"rgb(107,174,214)"],[0.625,"rgb(66,146,198)"],[0.75,"rgb(33,113,181)"],[0.875,"rgb(8,81,156)"],[1.0,"rgb(8,48,107)"]]},"x":["InceptionResNetV2","VGG16","VGG16","VGG19","VGG16","Xception","VGG16","VGG19","InceptionResNetV2","MobileNetV2","ResNet101V2","VGG19","VGG19","VGG19","VGG19","ResNet101V2","VGG19","VGG19","VGG19","VGG19","VGG19","VGG19","VGG19","InceptionResNetV2","Xception","ResNet101V2","InceptionResNetV2","VGG19","VGG19","VGG19","VGG19","VGG19","VGG16","VGG19","VGG16","VGG19","VGG19","VGG19","VGG19","VGG19","MobileNetV2","MobileNetV2","Xception","Xception","ResNet101V2","MobileNetV2","Xception","ResNet101V2","InceptionResNetV2","MobileNetV2","MobileNetV2","MobileNetV2","MobileNetV2","MobileNetV2","MobileNetV2","MobileNetV2","MobileNetV2","MobileNetV2","MobileNetV2","MobileNetV2","MobileNetV2","MobileNetV2","MobileNetV2","MobileNetV2","MobileNetV2","MobileNetV2","MobileNetV2","VGG16","Xception","VGG16","ResNet101V2","Xception","VGG16","ResNet101V2","Xception","VGG16","ResNet101V2","Xception","VGG16","ResNet101V2","Xception","VGG16","ResNet101V2","Xception","VGG16","ResNet101V2","Xception","VGG16","ResNet101V2","Xception","VGG16","ResNet101V2","Xception","VGG16","ResNet101V2","Xception"],"type":"scatter","y":[0.670545756816864,0.6685393452644348,0.7219101190567017,0.7873194217681885,0.8330658078193665,0.7106741666793823,0.7118780016899109,0.9081059098243713,0.6685393452644348,0.11235955357551575,0.728330671787262,0.9265649914741516,0.8350722193717957,0.944622814655304,0.9406099319458008,0.790930986404419,0.9478330612182617,0.942215085029602,0.9329855442047119,0.9197431802749634,0.94261634349823,0.925361156463623,0.9060994982719421,0.7126805782318115,0.7760834693908691,0.7776886224746704,0.7066613435745239,0.9249598979949951,0.7684590816497803,0.9261637330055237,0.9069020748138428,0.9514446258544922,0.6685393452644348,0.9498394727706909,0.8804173469543457,0.6685393452644348,0.9558587670326233,0.8619582653045654,0.9486356377601624,0.9418138265609741,0.7772873044013977,0.7897271513938904,0.7315409183502197,0.7692616581916809,0.7303370833396912,0.7965489625930786,0.7564205527305603,0.7086677551269531,0.6215890645980835,0.6685393452644348,0.6793739795684814,0.8479133248329163,0.8483145833015442,0.846308171749115,0.812600314617157,0.7997592091560364,0.853130042552948,0.8258426785469055,0.790930986404419,0.8471107482910156,0.8382825255393982,0.7905296683311462,0.7969502210617065,0.8142054677009583,0.841091513633728,0.8238362669944763,0.6685393452644348,0.9289727210998535,0.6685393452644348,0.9498394727706909,0.7776886224746704,0.8037720918655396,0.9462279081344604,0.7692616581916809,0.8198234438896179,0.9474318027496338,0.477126806974411,0.8374799489974976,0.8218298554420471,0.6653290390968323,0.745184600353241,0.9361958503723145,0.7813001871109009,0.8523274660110474,0.9229534268379211,0.6601123809814453,0.5100321173667908,0.94261634349823,0.6685393452644348,0.7893258333206177,0.5943017601966858,0.7853130102157593,0.7857142686843872,0.9341893792152405,0.7845104336738586,0.7796950340270996]},{"yaxis":"y3","showlegend":false,"mode":"markers","xaxis":"x3","name":"Feasible Trial","marker":{"colorbar":{"xpad":40,"x":1.0,"title":{"text":"Trial"}},"color":[0,1,2,3,7,10,11,13,14,16,17,18,19,20,21,23,25,26,27,29,30,31,32,34,35,36,37,38,39,40,41,42,43,44,46,48,49,50,51,52,55,57,59,67,68,69,71,72,73,74,155,156,157,158,159,160,161,162,164,165,166,167,168,169,170,171,173,174,178,179,180,181,182,183,184,185,186,187,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208],"line":{"color":"Grey","width":0.5},"colorscale":[[0.0,"rgb(247,251,255)"],[0.125,"rgb(222,235,247)"],[0.25,"rgb(198,219,239)"],[0.375,"rgb(158,202,225)"],[0.5,"rgb(107,174,214)"],[0.625,"rgb(66,146,198)"],[0.75,"rgb(33,113,181)"],[0.875,"rgb(8,81,156)"],[1.0,"rgb(8,48,107)"]]},"x":[true,false,false,false,false,false,true,true,false,true,true,true,true,true,true,true,true,true,true,true,true,true,true,true,true,true,true,true,true,true,true,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,true,true,true,true,false,true,true,false,true,true,true,true,true,true,true,false,true,false,true,true,true,true,true,true,true,false,true,false,true,false,true,true,true,true,true,false,true,false,true,false,true,true,true,true,true],"type":"scatter","y":[0.670545756816864,0.6685393452644348,0.7219101190567017,0.7873194217681885,0.8330658078193665,0.7106741666793823,0.7118780016899109,0.9081059098243713,0.6685393452644348,0.11235955357551575,0.728330671787262,0.9265649914741516,0.8350722193717957,0.944622814655304,0.9406099319458008,0.790930986404419,0.9478330612182617,0.942215085029602,0.9329855442047119,0.9197431802749634,0.94261634349823,0.925361156463623,0.9060994982719421,0.7126805782318115,0.7760834693908691,0.7776886224746704,0.7066613435745239,0.9249598979949951,0.7684590816497803,0.9261637330055237,0.9069020748138428,0.9514446258544922,0.6685393452644348,0.9498394727706909,0.8804173469543457,0.6685393452644348,0.9558587670326233,0.8619582653045654,0.9486356377601624,0.9418138265609741,0.7772873044013977,0.7897271513938904,0.7315409183502197,0.7692616581916809,0.7303370833396912,0.7965489625930786,0.7564205527305603,0.7086677551269531,0.6215890645980835,0.6685393452644348,0.6793739795684814,0.8479133248329163,0.8483145833015442,0.846308171749115,0.812600314617157,0.7997592091560364,0.853130042552948,0.8258426785469055,0.790930986404419,0.8471107482910156,0.8382825255393982,0.7905296683311462,0.7969502210617065,0.8142054677009583,0.841091513633728,0.8238362669944763,0.6685393452644348,0.9289727210998535,0.6685393452644348,0.9498394727706909,0.7776886224746704,0.8037720918655396,0.9462279081344604,0.7692616581916809,0.8198234438896179,0.9474318027496338,0.477126806974411,0.8374799489974976,0.8218298554420471,0.6653290390968323,0.745184600353241,0.9361958503723145,0.7813001871109009,0.8523274660110474,0.9229534268379211,0.6601123809814453,0.5100321173667908,0.94261634349823,0.6685393452644348,0.7893258333206177,0.5943017601966858,0.7853130102157593,0.7857142686843872,0.9341893792152405,0.7845104336738586,0.7796950340270996]},{"yaxis":"y4","showlegend":false,"mode":"markers","xaxis":"x4","name":"Feasible Trial","marker":{"colorbar":{"xpad":40,"x":1.0,"title":{"text":"Trial"}},"color":[0,1,2,3,7,10,11,13,14,16,17,18,19,20,21,23,25,26,27,29,30,31,32,34,35,36,37,38,39,40,41,42,43,44,46,48,49,50,51,52,55,57,59,67,68,69,71,72,73,74,155,156,157,158,159,160,161,162,164,165,166,167,168,169,170,171,173,174,178,179,180,181,182,183,184,185,186,187,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208],"line":{"color":"Grey","width":0.5},"colorscale":[[0.0,"rgb(247,251,255)"],[0.125,"rgb(222,235,247)"],[0.25,"rgb(198,219,239)"],[0.375,"rgb(158,202,225)"],[0.5,"rgb(107,174,214)"],[0.625,"rgb(66,146,198)"],[0.75,"rgb(33,113,181)"],[0.875,"rgb(8,81,156)"],[1.0,"rgb(8,48,107)"]]},"x":[0.4960775050872815,0.3026410358036335,0.6590546669958817,0.350212300784031,0.5944279849172035,0.3238901864230077,0.49358888886611596,0.42410597546078144,0.49611520928869857,0.6012885221889803,0.58043950548171,0.5969208509163735,0.689831237241465,0.42507702860908003,0.5412251035743053,0.542851800492867,0.38396432757733534,0.37813141021300145,0.3880343205435778,0.38402120891516994,0.35281854190708906,0.3473670785826005,0.4601717679734084,0.34653320953033806,0.47234453849923214,0.41588718320616175,0.31148767829791335,0.37365790890001427,0.36070673447364415,0.40414287001926597,0.33206343534364413,0.3678370567468727,0.31222161504486157,0.36354056159501674,0.32990061741757115,0.46047441054702976,0.362800923727492,0.3659493486034012,0.40096586869525297,0.39553455712580554,0.3421683441972399,0.4046973033147832,0.32776406359585253,0.39468344669744165,0.4154389709537496,0.3773784837313172,0.45476591622233686,0.3585713556301417,0.47865415102712233,0.42473259709950195,0.38001126078815967,0.3529584405565984,0.3399540771743265,0.39888189163031323,0.3853590306221154,0.35778613573501256,0.3156593931735219,0.40996837809254666,0.37103736201036963,0.3514996519333837,0.3381088980252816,0.3813332155080514,0.44384582581423604,0.390079422803016,0.36676076955966813,0.323515666880192,0.4178791131158042,0.4002884384729484,0.369629387672061,0.34801853889002116,0.3546937969090064,0.3475954203190637,0.33425407698430915,0.30366019536489985,0.3177771812610969,0.32899758985564903,0.3355287660050659,0.3422046036729765,0.32777836022083157,0.5702207358212126,0.3653794640250848,0.3586361596155612,0.3871994295420844,0.3213152829592995,0.33333482555993943,0.3470853397746965,0.3763855171928615,0.3035712462226969,0.508398524834934,0.4259642034366777,0.4093787565481985,0.30241495389281553,0.3116976412845023,0.3257966002623914,0.3625255737565286,0.33946464744532107],"type":"scatter","y":[0.670545756816864,0.6685393452644348,0.7219101190567017,0.7873194217681885,0.8330658078193665,0.7106741666793823,0.7118780016899109,0.9081059098243713,0.6685393452644348,0.11235955357551575,0.728330671787262,0.9265649914741516,0.8350722193717957,0.944622814655304,0.9406099319458008,0.790930986404419,0.9478330612182617,0.942215085029602,0.9329855442047119,0.9197431802749634,0.94261634349823,0.925361156463623,0.9060994982719421,0.7126805782318115,0.7760834693908691,0.7776886224746704,0.7066613435745239,0.9249598979949951,0.7684590816497803,0.9261637330055237,0.9069020748138428,0.9514446258544922,0.6685393452644348,0.9498394727706909,0.8804173469543457,0.6685393452644348,0.9558587670326233,0.8619582653045654,0.9486356377601624,0.9418138265609741,0.7772873044013977,0.7897271513938904,0.7315409183502197,0.7692616581916809,0.7303370833396912,0.7965489625930786,0.7564205527305603,0.7086677551269531,0.6215890645980835,0.6685393452644348,0.6793739795684814,0.8479133248329163,0.8483145833015442,0.846308171749115,0.812600314617157,0.7997592091560364,0.853130042552948,0.8258426785469055,0.790930986404419,0.8471107482910156,0.8382825255393982,0.7905296683311462,0.7969502210617065,0.8142054677009583,0.841091513633728,0.8238362669944763,0.6685393452644348,0.9289727210998535,0.6685393452644348,0.9498394727706909,0.7776886224746704,0.8037720918655396,0.9462279081344604,0.7692616581916809,0.8198234438896179,0.9474318027496338,0.477126806974411,0.8374799489974976,0.8218298554420471,0.6653290390968323,0.745184600353241,0.9361958503723145,0.7813001871109009,0.8523274660110474,0.9229534268379211,0.6601123809814453,0.5100321173667908,0.94261634349823,0.6685393452644348,0.7893258333206177,0.5943017601966858,0.7853130102157593,0.7857142686843872,0.9341893792152405,0.7845104336738586,0.7796950340270996]},{"yaxis":"y5","showlegend":false,"mode":"markers","xaxis":"x5","name":"Feasible Trial","marker":{"colorbar":{"xpad":40,"x":1.0,"title":{"text":"Trial"}},"color":[0,1,2,3,7,10,11,13,14,16,17,18,19,20,21,23,25,26,27,29,30,31,32,34,35,36,37,38,39,40,41,42,43,44,46,48,49,50,51,52,55,57,59,67,68,69,71,72,73,74,155,156,157,158,159,160,161,162,164,165,166,167,168,169,170,171,173,174,178,179,180,181,182,183,184,185,186,187,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208],"line":{"color":"Grey","width":0.5},"colorscale":[[0.0,"rgb(247,251,255)"],[0.125,"rgb(222,235,247)"],[0.25,"rgb(198,219,239)"],[0.375,"rgb(158,202,225)"],[0.5,"rgb(107,174,214)"],[0.625,"rgb(66,146,198)"],[0.75,"rgb(33,113,181)"],[0.875,"rgb(8,81,156)"],[1.0,"rgb(8,48,107)"]]},"x":[7.644090545864702e-5,1.0425042791215845e-5,1.080697450160222e-4,1.754338661275091e-6,4.127671085735859e-6,4.1015131019290326e-5,9.407201974338876e-4,8.8959545710159e-3,9.367882794420708e-6,5.549022953877958e-3,3.1347267213647655e-4,1.1151055589013968e-6,1.7147535679283997e-3,3.799311903542394e-4,2.847125071831052e-4,3.4763416083864615e-4,1.470612272096408e-3,1.759649261042102e-3,1.8953922435601784e-3,7.853489265089114e-4,2.369852854012148e-3,3.7630490000354596e-3,7.894208841696096e-4,1.1015861908130227e-4,9.280076186734558e-4,4.781943965994695e-4,1.2874711736140165e-4,2.1656176646882006e-3,1.644532436914805e-3,4.2428365208944585e-3,1.8088469807929958e-4,5.90335883988068e-4,6.214975922641371e-4,1.117251688625468e-3,1.1260164855398285e-3,4.863556824799373e-4,1.2188886659969672e-3,1.2230636041551644e-3,1.6367865182508187e-4,3.970912040890466e-5,6.557885750387884e-3,7.427790802947653e-5,1.7067997963192438e-4,2.481756420940997e-4,1.3145663620028078e-3,4.6085640475964483e-5,6.865341828727055e-4,1.3244522704051263e-3,2.6899028178269838e-5,9.192271182327469e-4,5.55115796832547e-3,2.5272369998691467e-3,4.343226939052277e-6,2.129302124886797e-3,2.455725597805224e-4,5.434746031616164e-4,1.5807032182731742e-3,3.34906532353831e-4,8.437820738594531e-4,3.7527028546560284e-3,7.733832106204712e-3,1.854587556770487e-3,1.0717312720192396e-3,2.4859206467216016e-3,4.410618819719139e-3,1.62262267562355e-3,4.1351117271583207e-4,7.739903908494326e-4,1.089341654966649e-3,2.948075080650043e-3,1.329094926815321e-4,2.9596977679617192e-3,1.4240351666761869e-3,2.188612649268277e-3,5.561022598222291e-4,1.4991870837214873e-3,1.5123583462056634e-3,2.0575809770785865e-4,9.579261314237906e-4,6.344256481544043e-4,1.335434264194725e-3,3.2127085718321336e-3,4.814070927201941e-3,1.95696481174114e-3,7.269572757638529e-4,2.5774169063122955e-3,1.3339691522616282e-3,3.3802911279991314e-3,2.9664670355945046e-4,4.89651984831912e-4,1.4416657797947322e-4,3.827196787744189e-3,5.753136038054284e-3,9.445838536929257e-4,9.379485848814512e-5,2.0998815311574336e-3],"type":"scatter","y":[0.670545756816864,0.6685393452644348,0.7219101190567017,0.7873194217681885,0.8330658078193665,0.7106741666793823,0.7118780016899109,0.9081059098243713,0.6685393452644348,0.11235955357551575,0.728330671787262,0.9265649914741516,0.8350722193717957,0.944622814655304,0.9406099319458008,0.790930986404419,0.9478330612182617,0.942215085029602,0.9329855442047119,0.9197431802749634,0.94261634349823,0.925361156463623,0.9060994982719421,0.7126805782318115,0.7760834693908691,0.7776886224746704,0.7066613435745239,0.9249598979949951,0.7684590816497803,0.9261637330055237,0.9069020748138428,0.9514446258544922,0.6685393452644348,0.9498394727706909,0.8804173469543457,0.6685393452644348,0.9558587670326233,0.8619582653045654,0.9486356377601624,0.9418138265609741,0.7772873044013977,0.7897271513938904,0.7315409183502197,0.7692616581916809,0.7303370833396912,0.7965489625930786,0.7564205527305603,0.7086677551269531,0.6215890645980835,0.6685393452644348,0.6793739795684814,0.8479133248329163,0.8483145833015442,0.846308171749115,0.812600314617157,0.7997592091560364,0.853130042552948,0.8258426785469055,0.790930986404419,0.8471107482910156,0.8382825255393982,0.7905296683311462,0.7969502210617065,0.8142054677009583,0.841091513633728,0.8238362669944763,0.6685393452644348,0.9289727210998535,0.6685393452644348,0.9498394727706909,0.7776886224746704,0.8037720918655396,0.9462279081344604,0.7692616581916809,0.8198234438896179,0.9474318027496338,0.477126806974411,0.8374799489974976,0.8218298554420471,0.6653290390968323,0.745184600353241,0.9361958503723145,0.7813001871109009,0.8523274660110474,0.9229534268379211,0.6601123809814453,0.5100321173667908,0.94261634349823,0.6685393452644348,0.7893258333206177,0.5943017601966858,0.7853130102157593,0.7857142686843872,0.9341893792152405,0.7845104336738586,0.7796950340270996]},{"yaxis":"y6","showlegend":false,"mode":"markers","xaxis":"x6","name":"Feasible Trial","marker":{"colorbar":{"xpad":40,"x":1.0,"title":{"text":"Trial"}},"color":[0,1,2,3,7,10,11,13,14,16,17,18,19,20,21,23,25,26,27,29,30,31,32,34,35,36,37,38,39,40,41,42,43,44,46,48,49,50,51,52,55,57,59,67,68,69,71,72,73,74,155,156,157,158,159,160,161,162,164,165,166,167,168,169,170,171,173,174,178,179,180,181,182,183,184,185,186,187,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208],"line":{"color":"Grey","width":0.5},"colorscale":[[0.0,"rgb(247,251,255)"],[0.125,"rgb(222,235,247)"],[0.25,"rgb(198,219,239)"],[0.375,"rgb(158,202,225)"],[0.5,"rgb(107,174,214)"],[0.625,"rgb(66,146,198)"],[0.75,"rgb(33,113,181)"],[0.875,"rgb(8,81,156)"],[1.0,"rgb(8,48,107)"]]},"x":[true,false,true,false,false,true,true,false,true,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,true,false,false,false,false,false,false,true,false,true,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,true,false,false,false,false,false,false,false,false,false,true,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,false,true,false,false,false,false,false,false,false,false,false],"type":"scatter","y":[0.670545756816864,0.6685393452644348,0.7219101190567017,0.7873194217681885,0.8330658078193665,0.7106741666793823,0.7118780016899109,0.9081059098243713,0.6685393452644348,0.11235955357551575,0.728330671787262,0.9265649914741516,0.8350722193717957,0.944622814655304,0.9406099319458008,0.790930986404419,0.9478330612182617,0.942215085029602,0.9329855442047119,0.9197431802749634,0.94261634349823,0.925361156463623,0.9060994982719421,0.7126805782318115,0.7760834693908691,0.7776886224746704,0.7066613435745239,0.9249598979949951,0.7684590816497803,0.9261637330055237,0.9069020748138428,0.9514446258544922,0.6685393452644348,0.9498394727706909,0.8804173469543457,0.6685393452644348,0.9558587670326233,0.8619582653045654,0.9486356377601624,0.9418138265609741,0.7772873044013977,0.7897271513938904,0.7315409183502197,0.7692616581916809,0.7303370833396912,0.7965489625930786,0.7564205527305603,0.7086677551269531,0.6215890645980835,0.6685393452644348,0.6793739795684814,0.8479133248329163,0.8483145833015442,0.846308171749115,0.812600314617157,0.7997592091560364,0.853130042552948,0.8258426785469055,0.790930986404419,0.8471107482910156,0.8382825255393982,0.7905296683311462,0.7969502210617065,0.8142054677009583,0.841091513633728,0.8238362669944763,0.6685393452644348,0.9289727210998535,0.6685393452644348,0.9498394727706909,0.7776886224746704,0.8037720918655396,0.9462279081344604,0.7692616581916809,0.8198234438896179,0.9474318027496338,0.477126806974411,0.8374799489974976,0.8218298554420471,0.6653290390968323,0.745184600353241,0.9361958503723145,0.7813001871109009,0.8523274660110474,0.9229534268379211,0.6601123809814453,0.5100321173667908,0.94261634349823,0.6685393452644348,0.7893258333206177,0.5943017601966858,0.7853130102157593,0.7857142686843872,0.9341893792152405,0.7845104336738586,0.7796950340270996]},{"yaxis":"y7","showlegend":false,"mode":"markers","xaxis":"x7","name":"Feasible Trial","marker":{"colorbar":{"xpad":40,"x":1.0,"title":{"text":"Trial"}},"color":[0,1,2,3,7,10,11,13,14,16,17,18,19,20,21,23,25,26,27,29,30,31,32,34,35,36,37,38,39,40,41,42,43,44,46,48,49,50,51,52,55,57,59,67,68,69,71,72,73,74,155,156,157,158,159,160,161,162,164,165,166,167,168,169,170,171,173,174,178,179,180,181,182,183,184,185,186,187,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208],"line":{"color":"Grey","width":0.5},"colorscale":[[0.0,"rgb(247,251,255)"],[0.125,"rgb(222,235,247)"],[0.25,"rgb(198,219,239)"],[0.375,"rgb(158,202,225)"],[0.5,"rgb(107,174,214)"],[0.625,"rgb(66,146,198)"],[0.75,"rgb(33,113,181)"],[0.875,"rgb(8,81,156)"],[1.0,"rgb(8,48,107)"]]},"x":[true,false,true,true,true,true,false,true,true,true,true,true,false,true,true,true,true,true,true,true,true,true,true,true,true,true,true,true,true,true,true,true,false,true,true,false,true,true,true,true,true,true,true,true,true,true,true,true,true,true,false,true,true,true,true,true,true,true,true,true,false,true,true,true,true,true,true,true,true,true,true,true,true,true,true,true,true,true,false,true,true,true,true,true,true,true,true,true,true,true,false,true,true,true,true,true],"type":"scatter","y":[0.670545756816864,0.6685393452644348,0.7219101190567017,0.7873194217681885,0.8330658078193665,0.7106741666793823,0.7118780016899109,0.9081059098243713,0.6685393452644348,0.11235955357551575,0.728330671787262,0.9265649914741516,0.8350722193717957,0.944622814655304,0.9406099319458008,0.790930986404419,0.9478330612182617,0.942215085029602,0.9329855442047119,0.9197431802749634,0.94261634349823,0.925361156463623,0.9060994982719421,0.7126805782318115,0.7760834693908691,0.7776886224746704,0.7066613435745239,0.9249598979949951,0.7684590816497803,0.9261637330055237,0.9069020748138428,0.9514446258544922,0.6685393452644348,0.9498394727706909,0.8804173469543457,0.6685393452644348,0.9558587670326233,0.8619582653045654,0.9486356377601624,0.9418138265609741,0.7772873044013977,0.7897271513938904,0.7315409183502197,0.7692616581916809,0.7303370833396912,0.7965489625930786,0.7564205527305603,0.7086677551269531,0.6215890645980835,0.6685393452644348,0.6793739795684814,0.8479133248329163,0.8483145833015442,0.846308171749115,0.812600314617157,0.7997592091560364,0.853130042552948,0.8258426785469055,0.790930986404419,0.8471107482910156,0.8382825255393982,0.7905296683311462,0.7969502210617065,0.8142054677009583,0.841091513633728,0.8238362669944763,0.6685393452644348,0.9289727210998535,0.6685393452644348,0.9498394727706909,0.7776886224746704,0.8037720918655396,0.9462279081344604,0.7692616581916809,0.8198234438896179,0.9474318027496338,0.477126806974411,0.8374799489974976,0.8218298554420471,0.6653290390968323,0.745184600353241,0.9361958503723145,0.7813001871109009,0.8523274660110474,0.9229534268379211,0.6601123809814453,0.5100321173667908,0.94261634349823,0.6685393452644348,0.7893258333206177,0.5943017601966858,0.7853130102157593,0.7857142686843872,0.9341893792152405,0.7845104336738586,0.7796950340270996]},{"yaxis":"y8","showlegend":false,"mode":"markers","xaxis":"x8","name":"Feasible Trial","marker":{"colorbar":{"xpad":40,"x":1.0,"title":{"text":"Trial"}},"color":[0,1,2,3,7,10,11,13,14,16,17,18,19,20,21,23,25,26,27,29,30,31,32,34,35,36,37,38,39,40,41,42,43,44,46,48,49,50,51,52,55,57,59,67,68,69,71,72,73,74,155,156,157,158,159,160,161,162,164,165,166,167,168,169,170,171,173,174,178,179,180,181,182,183,184,185,186,187,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208],"line":{"color":"Grey","width":0.5},"colorscale":[[0.0,"rgb(247,251,255)"],[0.125,"rgb(222,235,247)"],[0.25,"rgb(198,219,239)"],[0.375,"rgb(158,202,225)"],[0.5,"rgb(107,174,214)"],[0.625,"rgb(66,146,198)"],[0.75,"rgb(33,113,181)"],[0.875,"rgb(8,81,156)"],[1.0,"rgb(8,48,107)"]]},"x":["focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","categorical\_crossentropy","focal\_loss","focal\_loss","categorical\_crossentropy","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","categorical\_crossentropy","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","categorical\_crossentropy","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","categorical\_crossentropy","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","categorical\_crossentropy","categorical\_crossentropy","categorical\_crossentropy","categorical\_crossentropy","categorical\_crossentropy","categorical\_crossentropy","categorical\_crossentropy","categorical\_crossentropy","categorical\_crossentropy","categorical\_crossentropy","categorical\_crossentropy","categorical\_crossentropy","categorical\_crossentropy","categorical\_crossentropy","categorical\_crossentropy","categorical\_crossentropy","categorical\_crossentropy","categorical\_crossentropy","focal\_loss","focal\_loss","categorical\_crossentropy","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss","focal\_loss"],"type":"scatter","y":[0.670545756816864,0.6685393452644348,0.7219101190567017,0.7873194217681885,0.8330658078193665,0.7106741666793823,0.7118780016899109,0.9081059098243713,0.6685393452644348,0.11235955357551575,0.728330671787262,0.9265649914741516,0.8350722193717957,0.944622814655304,0.9406099319458008,0.790930986404419,0.9478330612182617,0.942215085029602,0.9329855442047119,0.9197431802749634,0.94261634349823,0.925361156463623,0.9060994982719421,0.7126805782318115,0.7760834693908691,0.7776886224746704,0.7066613435745239,0.9249598979949951,0.7684590816497803,0.9261637330055237,0.9069020748138428,0.9514446258544922,0.6685393452644348,0.9498394727706909,0.8804173469543457,0.6685393452644348,0.9558587670326233,0.8619582653045654,0.9486356377601624,0.9418138265609741,0.7772873044013977,0.7897271513938904,0.7315409183502197,0.7692616581916809,0.7303370833396912,0.7965489625930786,0.7564205527305603,0.7086677551269531,0.6215890645980835,0.6685393452644348,0.6793739795684814,0.8479133248329163,0.8483145833015442,0.846308171749115,0.812600314617157,0.7997592091560364,0.853130042552948,0.8258426785469055,0.790930986404419,0.8471107482910156,0.8382825255393982,0.7905296683311462,0.7969502210617065,0.8142054677009583,0.841091513633728,0.8238362669944763,0.6685393452644348,0.9289727210998535,0.6685393452644348,0.9498394727706909,0.7776886224746704,0.8037720918655396,0.9462279081344604,0.7692616581916809,0.8198234438896179,0.9474318027496338,0.477126806974411,0.8374799489974976,0.8218298554420471,0.6653290390968323,0.745184600353241,0.9361958503723145,0.7813001871109009,0.8523274660110474,0.9229534268379211,0.6601123809814453,0.5100321173667908,0.94261634349823,0.6685393452644348,0.7893258333206177,0.5943017601966858,0.7853130102157593,0.7857142686843872,0.9341893792152405,0.7845104336738586,0.7796950340270996]},{"yaxis":"y9","showlegend":false,"mode":"markers","xaxis":"x9","name":"Feasible Trial","marker":{"colorbar":{"xpad":40,"x":1.0,"title":{"text":"Trial"}},"color":[0,1,2,3,7,10,11,13,14,16,17,18,19,20,21,23,25,26,27,29,30,31,32,34,35,36,37,38,39,40,41,42,43,44,46,48,49,50,51,52,55,57,59,67,68,69,71,72,73,74,155,156,157,158,159,160,161,162,164,165,166,167,168,169,170,171,173,174,178,179,180,181,182,183,184,185,186,187,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208],"line":{"color":"Grey","width":0.5},"colorscale":[[0.0,"rgb(247,251,255)"],[0.125,"rgb(222,235,247)"],[0.25,"rgb(198,219,239)"],[0.375,"rgb(158,202,225)"],[0.5,"rgb(107,174,214)"],[0.625,"rgb(66,146,198)"],[0.75,"rgb(33,113,181)"],[0.875,"rgb(8,81,156)"],[1.0,"rgb(8,48,107)"]]},"x":[3,3,3,2,2,1,2,2,1,2,2,2,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,3,1,1,2,2,2,2,2,2,2,2,2,3,2,2,3,2,1,2,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,2,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,3,2,2,2,2,2,2,2,2,2],"type":"scatter","y":[0.670545756816864,0.6685393452644348,0.7219101190567017,0.7873194217681885,0.8330658078193665,0.7106741666793823,0.7118780016899109,0.9081059098243713,0.6685393452644348,0.11235955357551575,0.728330671787262,0.9265649914741516,0.8350722193717957,0.944622814655304,0.9406099319458008,0.790930986404419,0.9478330612182617,0.942215085029602,0.9329855442047119,0.9197431802749634,0.94261634349823,0.925361156463623,0.9060994982719421,0.7126805782318115,0.7760834693908691,0.7776886224746704,0.7066613435745239,0.9249598979949951,0.7684590816497803,0.9261637330055237,0.9069020748138428,0.9514446258544922,0.6685393452644348,0.9498394727706909,0.8804173469543457,0.6685393452644348,0.9558587670326233,0.8619582653045654,0.9486356377601624,0.9418138265609741,0.7772873044013977,0.7897271513938904,0.7315409183502197,0.7692616581916809,0.7303370833396912,0.7965489625930786,0.7564205527305603,0.7086677551269531,0.6215890645980835,0.6685393452644348,0.6793739795684814,0.8479133248329163,0.8483145833015442,0.846308171749115,0.812600314617157,0.7997592091560364,0.853130042552948,0.8258426785469055,0.790930986404419,0.8471107482910156,0.8382825255393982,0.7905296683311462,0.7969502210617065,0.8142054677009583,0.841091513633728,0.8238362669944763,0.6685393452644348,0.9289727210998535,0.6685393452644348,0.9498394727706909,0.7776886224746704,0.8037720918655396,0.9462279081344604,0.7692616581916809,0.8198234438896179,0.9474318027496338,0.477126806974411,0.8374799489974976,0.8218298554420471,0.6653290390968323,0.745184600353241,0.9361958503723145,0.7813001871109009,0.8523274660110474,0.9229534268379211,0.6601123809814453,0.5100321173667908,0.94261634349823,0.6685393452644348,0.7893258333206177,0.5943017601966858,0.7853130102157593,0.7857142686843872,0.9341893792152405,0.7845104336738586,0.7796950340270996]},{"yaxis":"y10","showlegend":false,"mode":"markers","xaxis":"x10","name":"Feasible Trial","marker":{"colorbar":{"xpad":40,"x":1.0,"title":{"text":"Trial"}},"color":[0,1,2,3,7,10,11,13,14,16,17,18,19,20,21,23,25,26,27,29,30,31,32,34,35,36,37,38,39,40,41,42,43,44,46,48,49,50,51,52,55,57,59,67,68,69,71,72,73,74,155,156,157,158,159,160,161,162,164,165,166,167,168,169,170,171,173,174,178,179,180,181,182,183,184,185,186,187,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208],"line":{"color":"Grey","width":0.5},"colorscale":[[0.0,"rgb(247,251,255)"],[0.125,"rgb(222,235,247)"],[0.25,"rgb(198,219,239)"],[0.375,"rgb(158,202,225)"],[0.5,"rgb(107,174,214)"],[0.625,"rgb(66,146,198)"],[0.75,"rgb(33,113,181)"],[0.875,"rgb(8,81,156)"],[1.0,"rgb(8,48,107)"]]},"x":[3.719099947854293e-4,2.5148175155375052e-6,2.2084539821603144e-6,3.0530562644958884e-6,1.4564381323424583e-5,2.90367258430538e-4,5.403726812968865e-5,1.787949721810223e-5,3.0379367437559432e-5,1.429307145666793e-5,1.1433701369583976e-5,1.2055584510507954e-4,1.2707541402927697e-4,1.8306947121149775e-4,6.346546090846041e-4,9.565953680352569e-4,3.620934982895585e-4,1.8903145440708112e-4,2.459759885063566e-4,2.0585201478480285e-4,5.0990161099835333e-5,5.651769195738681e-5,6.211809959692228e-5,2.6681318481592302e-5,5.562530020061215e-6,4.8531791467450495e-4,8.762149524014564e-5,1.5703467075945394e-4,1.1140614818746737e-6,3.246081814263154e-4,1.8143932806181562e-4,4.221751458923657e-5,3.273196296310207e-5,4.153154809652371e-5,6.802443957620954e-6,2.2546374039332758e-5,4.427308850392865e-5,3.978193363502007e-5,7.044320530140698e-5,4.046779716367932e-5,2.027093742784837e-5,1.1876714899126725e-4,1.562992578383465e-5,6.31676757380743e-5,2.898918285976599e-4,2.9121757392379145e-5,6.427350303258348e-4,1.0189163975781635e-4,4.8088360962498394e-5,2.4262415747499454e-4,1.6677054054952756e-4,5.324620562766805e-5,3.448416928376019e-5,2.2603451078984874e-5,6.715840550873993e-5,3.734009660402447e-4,4.924203841404809e-5,1.3848296161066183e-4,2.7074268435177287e-5,9.858239018314565e-5,7.071990711149281e-4,2.1475410859356307e-4,3.3532361614622405e-4,4.6394457671447646e-4,7.791799662084754e-5,3.638516421026646e-5,4.7316921741882696e-5,1.4845630103598964e-4,2.071417288685285e-4,1.0928590664086039e-4,8.145420964156894e-5,5.943448772215874e-5,2.593619748097705e-4,2.662791842220211e-4,1.063554167185056e-4,3.9088863530156167e-4,4.0589860493264084e-4,8.339678629730929e-4,5.242319057864118e-4,3.2397746224528993e-4,2.29652675087322e-4,4.398695677322417e-5,1.7862238077540285e-4,5.609682035350665e-4,5.5922275219688976e-5,1.1282843099160059e-4,3.1652461326971105e-5,4.2595347964107785e-4,2.3810267509974917e-5,6.78908254763398e-5,2.9462304450438156e-4,3.560144765718682e-4,4.172149363025072e-4,1.4176435400626203e-4,2.6083183713507615e-4,3.700625074215402e-5],"type":"scatter","y":[0.670545756816864,0.6685393452644348,0.7219101190567017,0.7873194217681885,0.8330658078193665,0.7106741666793823,0.7118780016899109,0.9081059098243713,0.6685393452644348,0.11235955357551575,0.728330671787262,0.9265649914741516,0.8350722193717957,0.944622814655304,0.9406099319458008,0.790930986404419,0.9478330612182617,0.942215085029602,0.9329855442047119,0.9197431802749634,0.94261634349823,0.925361156463623,0.9060994982719421,0.7126805782318115,0.7760834693908691,0.7776886224746704,0.7066613435745239,0.9249598979949951,0.7684590816497803,0.9261637330055237,0.9069020748138428,0.9514446258544922,0.6685393452644348,0.9498394727706909,0.8804173469543457,0.6685393452644348,0.9558587670326233,0.8619582653045654,0.9486356377601624,0.9418138265609741,0.7772873044013977,0.7897271513938904,0.7315409183502197,0.7692616581916809,0.7303370833396912,0.7965489625930786,0.7564205527305603,0.7086677551269531,0.6215890645980835,0.6685393452644348,0.6793739795684814,0.8479133248329163,0.8483145833015442,0.846308171749115,0.812600314617157,0.7997592091560364,0.853130042552948,0.8258426785469055,0.790930986404419,0.8471107482910156,0.8382825255393982,0.7905296683311462,0.7969502210617065,0.8142054677009583,0.841091513633728,0.8238362669944763,0.6685393452644348,0.9289727210998535,0.6685393452644348,0.9498394727706909,0.7776886224746704,0.8037720918655396,0.9462279081344604,0.7692616581916809,0.8198234438896179,0.9474318027496338,0.477126806974411,0.8374799489974976,0.8218298554420471,0.6653290390968323,0.745184600353241,0.9361958503723145,0.7813001871109009,0.8523274660110474,0.9229534268379211,0.6601123809814453,0.5100321173667908,0.94261634349823,0.6685393452644348,0.7893258333206177,0.5943017601966858,0.7853130102157593,0.7857142686843872,0.9341893792152405,0.7845104336738586,0.7796950340270996]}]}

## Analysis per model architecture

The hyperparameter optimization process has been successfully completed, and the best hyperparameters have been selected to train the final model on the HAM10000 dataset. The hyperparameters were optimized using the Optuna library, which uses the Tree-structured Parzen Estimator (TPE) algorithm to search the hyperparameter space efficiently. The hyperparameters that were optimized include the batch size, number of dense layers, dense units, batch normalization, dropout rate, loss function, alpha and gamma for the focal loss, class weights, weight decay, learning rate, base model architecture, attention mechanism, AMSGrad, pre-trained weights, and number of layers to unfreeze.

import optuna  
from plotly.subplots import make\_subplots  
  
# Assuming 'study' is your Optuna study object  
all\_trials = study.get\_trials()  
  
# Get unique base\_model\_architecture values  
architectures = set([trial.params['base\_model\_architecture'] for trial in all\_trials])  
  
# Create subplots, one row for each architecture  
fig = make\_subplots(rows=len(architectures), cols=1, subplot\_titles=list(architectures))  
  
for i, arch in enumerate(architectures):  
 # Filter trials by architecture  
 filtered\_trials = [trial for trial in all\_trials if trial.params['base\_model\_architecture'] == arch]  
  
 # Create a new study with the filtered trials  
 filtered\_study = optuna.create\_study()  
 for trial in filtered\_trials:  
 filtered\_study.add\_trial(trial)  
  
 # Create a slice plot for the filtered study  
 slice\_plot = optuna.visualization.plot\_slice(filtered\_study)  
  
 # Get data from slice plot  
 for trace in slice\_plot['data']:  
 trace.x = [x if x is not None else "None" for x in trace.x] # Fix for None values in trace  
  
 # Add trace to subplot  
 fig.add\_trace(  
 slice\_plot['data'][0],  
 row=i + 1,  
 col=1  
 )  
  
fig.update\_layout(height=400 \* len(architectures), width=1200, title\_text="Sliced plots per architecture")  
fig.show()

[I 2024-04-01 22:06:23,588] A new study created in memory with name: no-name-310e8875-e47a-4389-ad4f-476915f961fb  
[I 2024-04-01 22:06:23,710] A new study created in memory with name: no-name-f9cb6557-10f5-4808-81a6-eae5f51ce73b  
[I 2024-04-01 22:06:23,831] A new study created in memory with name: no-name-9c15202a-dfec-4f0a-905d-66b990c6934f  
[I 2024-04-01 22:06:23,946] A new study created in memory with name: no-name-a1aed416-52b7-43cc-9c46-3d3faeaf1b26  
[I 2024-04-01 22:06:24,065] A new study created in memory with name: no-name-7974f469-06e4-4651-aa79-015bdb760a7b  
[I 2024-04-01 22:06:24,177] A new study created in memory with name: no-name-4f95fa46-706b-40ec-b7a5-f0098ea6ba1a

{"layout":{"yaxis6":{"domain":[0.0,9.722222222222222e-2],"anchor":"x6"},"annotations":[{"showarrow":false,"text":"VGG19","xref":"paper","x":0.5,"yanchor":"bottom","yref":"paper","font":{"size":16},"xanchor":"center","y":0.9999999999999999},{"showarrow":false,"text":"InceptionResNetV2","xref":"paper","x":0.5,"yanchor":"bottom","yref":"paper","font":{"size":16},"xanchor":"center","y":0.8194444444444444},{"showarrow":false,"text":"MobileNetV2","xref":"paper","x":0.5,"yanchor":"bottom","yref":"paper","font":{"size":16},"xanchor":"center","y":0.638888888888889},{"showarrow":false,"text":"Xception","xref":"paper","x":0.5,"yanchor":"bottom","yref":"paper","font":{"size":16},"xanchor":"center","y":0.4583333333333333},{"showarrow":false,"text":"VGG16","xref":"paper","x":0.5,"yanchor":"bottom","yref":"paper","font":{"size":16},"xanchor":"center","y":0.2777777777777778},{"showarrow":false,"text":"ResNet101V2","xref":"paper","x":0.5,"yanchor":"bottom","yref":"paper","font":{"size":16},"xanchor":"center","y":9.722222222222222e-2}],"xaxis4":{"domain":[0.0,1.0],"anchor":"y4"},"height":2400,"yaxis":{"domain":[0.9027777777777777,0.9999999999999999],"anchor":"x"},"xaxis3":{"domain":[0.0,1.0],"anchor":"y3"},"xaxis":{"domain":[0.0,1.0],"anchor":"y"},"width":1200,"yaxis4":{"domain":[0.3611111111111111,0.4583333333333333],"anchor":"x4"},"xaxis2":{"domain":[0.0,1.0],"anchor":"y2"},"yaxis5":{"domain":[0.18055555555555555,0.2777777777777778],"anchor":"x5"},"xaxis6":{"domain":[0.0,1.0],"anchor":"y6"},"yaxis2":{"domain":[0.7222222222222222,0.8194444444444444],"anchor":"x2"},"title":{"text":"Sliced plots per architecture"},"template":{"layout":{"annotationdefaults":{"arrowwidth":1,"arrowhead":0,"arrowcolor":"#f2f5fa"},"yaxis":{"zerolinecolor":"#283442","zerolinewidth":2,"gridcolor":"#283442","title":{"standoff":15},"linecolor":"#506784","ticks":"","automargin":true},"sliderdefaults":{"borderwidth":1,"bgcolor":"#C8D4E3","tickwidth":0,"bordercolor":"rgb(17,17,17)"},"hovermode":"closest","hoverlabel":{"align":"left"},"polar":{"bgcolor":"rgb(17,17,17)","angularaxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""},"radialaxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""}},"xaxis":{"zerolinecolor":"#283442","zerolinewidth":2,"gridcolor":"#283442","title":{"standoff":15},"linecolor":"#506784","ticks":"","automargin":true},"coloraxis":{"colorbar":{"outlinewidth":0,"ticks":""}},"paper\_bgcolor":"rgb(17,17,17)","ternary":{"bgcolor":"rgb(17,17,17)","caxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""},"baxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""},"aaxis":{"gridcolor":"#506784","linecolor":"#506784","ticks":""}},"shapedefaults":{"line":{"color":"#f2f5fa"}},"autotypenumbers":"strict","updatemenudefaults":{"borderwidth":0,"bgcolor":"#506784"},"colorway":["#636efa","#EF553B","#00cc96","#ab63fa","#FFA15A","#19d3f3","#FF6692","#B6E880","#FF97FF","#FECB52"],"geo":{"subunitcolor":"#506784","bgcolor":"rgb(17,17,17)","showlakes":true,"lakecolor":"rgb(17,17,17)","showland":true,"landcolor":"rgb(17,17,17)"},"colorscale":{"sequentialminus":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"diverging":[[0,"#8e0152"],[0.1,"#c51b7d"],[0.2,"#de77ae"],[0.3,"#f1b6da"],[0.4,"#fde0ef"],[0.5,"#f7f7f7"],[0.6,"#e6f5d0"],[0.7,"#b8e186"],[0.8,"#7fbc41"],[0.9,"#4d9221"],[1,"#276419"]],"sequential":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]]},"plot\_bgcolor":"rgb(17,17,17)","title":{"x":5.0e-2},"font":{"color":"#f2f5fa"},"scene":{"yaxis":{"backgroundcolor":"rgb(17,17,17)","zerolinecolor":"#C8D4E3","gridcolor":"#506784","showbackground":true,"linecolor":"#506784","ticks":"","gridwidth":2},"xaxis":{"backgroundcolor":"rgb(17,17,17)","zerolinecolor":"#C8D4E3","gridcolor":"#506784","showbackground":true,"linecolor":"#506784","ticks":"","gridwidth":2},"zaxis":{"backgroundcolor":"rgb(17,17,17)","zerolinecolor":"#C8D4E3","gridcolor":"#506784","showbackground":true,"linecolor":"#506784","ticks":"","gridwidth":2}},"mapbox":{"style":"dark"}},"data":{"mesh3d":[{"colorbar":{"outlinewidth":0,"ticks":""},"type":"mesh3d"}],"scattercarpet":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scattercarpet"}],"heatmapgl":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"heatmapgl"}],"parcoords":[{"line":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"parcoords"}],"histogram2dcontour":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"histogram2dcontour"}],"contour":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"contour"}],"histogram2d":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"histogram2d"}],"barpolar":[{"marker":{"pattern":{"size":10,"solidity":0.2,"fillmode":"overlay"},"line":{"color":"rgb(17,17,17)","width":0.5}},"type":"barpolar"}],"carpet":[{"baxis":{"minorgridcolor":"#506784","endlinecolor":"#A2B1C6","gridcolor":"#506784","startlinecolor":"#A2B1C6","linecolor":"#506784"},"aaxis":{"minorgridcolor":"#506784","endlinecolor":"#A2B1C6","gridcolor":"#506784","startlinecolor":"#A2B1C6","linecolor":"#506784"},"type":"carpet"}],"heatmap":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"heatmap"}],"pie":[{"type":"pie","automargin":true}],"choropleth":[{"colorbar":{"outlinewidth":0,"ticks":""},"type":"choropleth"}],"contourcarpet":[{"colorbar":{"outlinewidth":0,"ticks":""},"type":"contourcarpet"}],"scattermapbox":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scattermapbox"}],"scattergl":[{"marker":{"line":{"color":"#283442"}},"type":"scattergl"}],"scatterpolargl":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scatterpolargl"}],"surface":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":[[0.0,"#0d0887"],[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],[1.0,"#f0f921"]],"type":"surface"}],"histogram":[{"marker":{"pattern":{"size":10,"solidity":0.2,"fillmode":"overlay"}},"type":"histogram"}],"scattergeo":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scattergeo"}],"scatter":[{"marker":{"line":{"color":"#283442"}},"type":"scatter"}],"table":[{"cells":{"line":{"color":"rgb(17,17,17)"},"fill":{"color":"#506784"}},"header":{"line":{"color":"rgb(17,17,17)"},"fill":{"color":"#2a3f5f"}},"type":"table"}],"scatterpolar":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scatterpolar"}],"scatterternary":[{"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scatterternary"}],"bar":[{"error\_y":{"color":"#f2f5fa"},"error\_x":{"color":"#f2f5fa"},"marker":{"pattern":{"size":10,"solidity":0.2,"fillmode":"overlay"},"line":{"color":"rgb(17,17,17)","width":0.5}},"type":"bar"}],"scatter3d":[{"line":{"colorbar":{"outlinewidth":0,"ticks":""}},"marker":{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"scatter3d"}]}},"xaxis5":{"domain":[0.0,1.0],"anchor":"y5"},"yaxis3":{"domain":[0.5416666666666667,0.638888888888889],"anchor":"x3"}},"config":{"plotlyServerURL":"https://plot.ly"},"data":[{"yaxis":"y","showlegend":false,"mode":"markers","xaxis":"x","name":"Feasible Trial","marker":{"colorbar":{"xpad":40,"x":1.0,"title":{"text":"Trial"}},"color":[0,1,3,5,6,7,8,9,11,12,13,14,15,16,17,18,19,20,23,24,25,26],"showscale":true,"line":{"color":"Grey","width":0.5},"colorscale":[[0.0,"rgb(247,251,255)"],[0.125,"rgb(222,235,247)"],[0.25,"rgb(198,219,239)"],[0.375,"rgb(158,202,225)"],[0.5,"rgb(107,174,214)"],[0.625,"rgb(66,146,198)"],[0.75,"rgb(33,113,181)"],[0.875,"rgb(8,81,156)"],[1.0,"rgb(8,48,107)"]]},"x":[0.5066070776124119,0.5145474111905396,0.7914831864471182,0.5685576194863156,0.6152076703012759,0.6214470820979032,0.5694781324147983,0.5642794686008311,0.6814574149469438,0.5700624306940455,0.4629194521197062,0.6889213443061398,0.5495881727597883,0.5630788879940318,0.6707191342464157,0.5202033176973268,0.5887769877065758,0.45426347766598163,0.48834840832599113,0.47696618574958627,0.4282047737105452,0.3258252811240391],"type":"scatter","y":[0.7873194217681885,0.9081059098243713,0.9265649914741516,0.944622814655304,0.9406099319458008,0.9478330612182617,0.942215085029602,0.9329855442047119,0.9197431802749634,0.94261634349823,0.925361156463623,0.9060994982719421,0.9249598979949951,0.7684590816497803,0.9261637330055237,0.9069020748138428,0.9514446258544922,0.9498394727706909,0.9558587670326233,0.8619582653045654,0.9486356377601624,0.9418138265609741]},{"yaxis":"y2","showlegend":false,"mode":"markers","xaxis":"x2","name":"Feasible Trial","marker":{"colorbar":{"xpad":40,"x":1.0,"title":{"text":"Trial"}},"color":[0,1,2,3,9],"showscale":true,"line":{"color":"Grey","width":0.5},"colorscale":[[0.0,"rgb(247,251,255)"],[0.125,"rgb(222,235,247)"],[0.25,"rgb(198,219,239)"],[0.375,"rgb(158,202,225)"],[0.5,"rgb(107,174,214)"],[0.625,"rgb(66,146,198)"],[0.75,"rgb(33,113,181)"],[0.875,"rgb(8,81,156)"],[1.0,"rgb(8,48,107)"]]},"x":[0.3847112336461335,0.47464385921834035,0.5453036577268427,0.5939479787091781,0.4963248388936603],"type":"scatter","y":[0.670545756816864,0.6685393452644348,0.7126805782318115,0.7066613435745239,0.6215890645980835]},{"yaxis":"y3","showlegend":false,"mode":"markers","xaxis":"x3","name":"Feasible Trial","marker":{"colorbar":{"xpad":40,"x":1.0,"title":{"text":"Trial"}},"color":[3,4,7,8,10,11,12,13,14,15,16,19,20,21,22,23,24,25,26,28],"showscale":true,"line":{"color":"Grey","width":0.5},"colorscale":[[0.0,"rgb(247,251,255)"],[0.125,"rgb(222,235,247)"],[0.25,"rgb(198,219,239)"],[0.375,"rgb(158,202,225)"],[0.5,"rgb(107,174,214)"],[0.625,"rgb(66,146,198)"],[0.75,"rgb(33,113,181)"],[0.875,"rgb(8,81,156)"],[1.0,"rgb(8,48,107)"]]},"x":[0.3920516685340587,0.3269627903274364,0.46114444115506464,0.5324236669345751,0.5778809640152006,0.5890011362217371,0.4911904113468172,0.6466998221606395,0.45770215366830025,0.5429888817479414,0.607758078569195,0.5640078940812687,0.6309352390097644,0.6711673992161106,0.5941763281415223,0.5779352150679776,0.5100165852221639,0.20679747989851338,0.4095342160407113,0.5574029788983251],"type":"scatter","y":[0.7772873044013977,0.7897271513938904,0.7965489625930786,0.6685393452644348,0.6793739795684814,0.8479133248329163,0.8483145833015442,0.846308171749115,0.812600314617157,0.7997592091560364,0.853130042552948,0.790930986404419,0.8471107482910156,0.8382825255393982,0.7905296683311462,0.7969502210617065,0.8142054677009583,0.841091513633728,0.8238362669944763,0.6685393452644348]},{"yaxis":"y4","showlegend":false,"mode":"markers","xaxis":"x4","name":"Feasible Trial","marker":{"colorbar":{"xpad":40,"x":1.0,"title":{"text":"Trial"}},"color":[1,3,5,7,8,10,19,20],"showscale":true,"line":{"color":"Grey","width":0.5},"colorscale":[[0.0,"rgb(247,251,255)"],[0.125,"rgb(222,235,247)"],[0.25,"rgb(198,219,239)"],[0.375,"rgb(158,202,225)"],[0.5,"rgb(107,174,214)"],[0.625,"rgb(66,146,198)"],[0.75,"rgb(33,113,181)"],[0.875,"rgb(8,81,156)"],[1.0,"rgb(8,48,107)"]]},"x":[0.25452540837272847,0.4355138333627301,0.4955055468762741,0.41890489701343725,0.7142378966675771,0.4781080985920682,0.7533087273351764,0.48302446158383894],"type":"scatter","y":[0.7106741666793823,0.7760834693908691,0.7315409183502197,0.7692616581916809,0.7564205527305603,0.6685393452644348,0.7857142686843872,0.7796950340270996]},{"yaxis":"y5","showlegend":false,"mode":"markers","xaxis":"x5","name":"Feasible Trial","marker":{"colorbar":{"xpad":40,"x":1.0,"title":{"text":"Trial"}},"color":[0,1,4,6,7,8,9,19,20,21],"showscale":true,"line":{"color":"Grey","width":0.5},"colorscale":[[0.0,"rgb(247,251,255)"],[0.125,"rgb(222,235,247)"],[0.25,"rgb(198,219,239)"],[0.375,"rgb(158,202,225)"],[0.5,"rgb(107,174,214)"],[0.625,"rgb(66,146,198)"],[0.75,"rgb(33,113,181)"],[0.875,"rgb(8,81,156)"],[1.0,"rgb(8,48,107)"]]},"x":[0.3286897403473636,0.6380333635916098,0.7339320035916865,0.3676097653153691,0.6155299936984829,0.3867988341424716,0.44797450036863806,0.7764622194264383,0.5358859203839308,0.7976150683366007],"type":"scatter","y":[0.6685393452644348,0.7219101190567017,0.8330658078193665,0.7118780016899109,0.6685393452644348,0.8804173469543457,0.9289727210998535,0.94261634349823,0.5943017601966858,0.9341893792152405]},{"yaxis":"y6","showlegend":false,"mode":"markers","xaxis":"x6","name":"Feasible Trial","marker":{"colorbar":{"xpad":40,"x":1.0,"title":{"text":"Trial"}},"color":[1,3,4,7,16,17,18],"showscale":true,"line":{"color":"Grey","width":0.5},"colorscale":[[0.0,"rgb(247,251,255)"],[0.125,"rgb(222,235,247)"],[0.25,"rgb(198,219,239)"],[0.375,"rgb(158,202,225)"],[0.5,"rgb(107,174,214)"],[0.625,"rgb(66,146,198)"],[0.75,"rgb(33,113,181)"],[0.875,"rgb(8,81,156)"],[1.0,"rgb(8,48,107)"]]},"x":[0.7378646631758355,0.6051394211146705,0.6489524612728226,0.35704267702921394,0.35540300963465,0.7503838496534098,0.7143424672082921],"type":"scatter","y":[0.728330671787262,0.790930986404419,0.7776886224746704,0.7303370833396912,0.6685393452644348,0.7853130102157593,0.7845104336738586]}]}

# Explanation of the Results

The optuna study shows a bias towards the VGG19 architecture, with the majority of the trials focusing on optimizing the hyperparameters for the VGG19 architecture. The VGG19 architecture has been shown to perform well on image classification tasks, and the hyperparameter optimization process has identified the best hyperparameters for this architecture. This bias towards the VGG19 architecture may be due to its deeper architecture and its ability to capture more complex features (abstract features) from the images.

# Hyperparameter Optimization Results

The hyperparameter optimization process has been successfully completed, and the best hyperparameters have been selected to train the final model on the HAM10000 dataset. The hyperparameters were optimized using the Optuna library, which uses the Tree-structured Parzen Estimator (TPE) algorithm to search the hyperparameter space efficiently. The hyperparameters that were optimized include the batch size, number of dense layers, dense units, batch normalization, dropout rate, loss function, alpha and gamma for the focal loss, class weights, weight decay, learning rate, base model architecture, attention mechanism, AMSGrad, pre-trained weights, and number of layers to unfreeze.

From all the models architecture tested, the one with the best result and consistency was VGG19 with CBAM attention mechanism. The model was trained with the following hyperparameters:

# Best hyperparameters  
best\_trial.params

{'batch': 46,  
 'num\_dense\_layers': 2,  
 'dense\_units\_0': 667,  
 'dense\_units\_1': 515,  
 'batch\_normalization\_0': False,  
 'batch\_normalization\_1': True,  
 'dropout\_rate\_0': 0.362800923727492,  
 'dropout\_rate\_1': 0.35888302707521497,  
 'loss\_type': 'focal\_loss',  
 'alpha\_phase1': 0.48834840832599113,  
 'gamma\_phase1': 4.208586719042138,  
 'use\_class\_weights': True,  
 'weight\_decay\_phase1': 7.133686465531547e-05,  
 'learning\_rate\_phase1': 0.00041421607571416667,  
 'learning\_rate\_phase2': 4.427308850392865e-05,  
 'alpha\_phase2': 0.6047625441951349,  
 'gamma\_phase2': 1.019029980284372,  
 'weight\_decay\_phase2': 0.0012188886659969672,  
 'base\_model\_architecture': 'VGG19',  
 'attention\_mechanism': 'CBAM',  
 'use\_amsgrad': False,  
 'pre\_trained\_weights': True,  
 'num\_layers\_unfreeze': 5}

# Build the model with the best hyperparameters  
def build\_model(hyperparameters):  
 image\_size = (256, 256)  
 unique\_classes = len(pd\_dataset['dx'].unique())  
  
 batch\_size = hyperparameters.params['batch']  
 num\_dense\_layers = hyperparameters.params['num\_dense\_layers']  
 dense\_units = [hyperparameters.params[f'dense\_units\_{i}'] for i in range(num\_dense\_layers)]  
 batch\_normalization\_layers = [hyperparameters.params[f'batch\_normalization\_{i}'] for i in range(num\_dense\_layers)]  
 dropout\_rate\_layers = [hyperparameters.params[f'dropout\_rate\_{i}'] for i in range(num\_dense\_layers)]  
 loss\_type = hyperparameters.params['loss\_type']  
 alpha\_phase1 = hyperparameters.params['alpha\_phase1'] if loss\_type == 'focal\_loss' else None  
 gamma\_phase1 = hyperparameters.params['gamma\_phase1'] if loss\_type == 'focal\_loss' else None  
 use\_class\_weights = hyperparameters.params['use\_class\_weights']  
 weight\_decay\_phase1 = hyperparameters.params['weight\_decay\_phase1']  
 learning\_rate\_phase1 = hyperparameters.params['learning\_rate\_phase1']  
 learning\_rate\_phase2 = hyperparameters.params['learning\_rate\_phase2']  
 alpha\_phase2 = hyperparameters.params['alpha\_phase2'] if loss\_type == 'focal\_loss' else None  
 gamma\_phase2 = hyperparameters.params['gamma\_phase2'] if loss\_type == 'focal\_loss' else None  
 weight\_decay\_phase2 = hyperparameters.params['weight\_decay\_phase2']  
 base\_model\_architecture = hyperparameters.params['base\_model\_architecture']  
 attention\_mechanism = hyperparameters.params['attention\_mechanism']  
 use\_amsgrad = hyperparameters.params['use\_amsgrad']  
 pre\_trained\_weights = hyperparameters.params['pre\_trained\_weights']  
 num\_layers\_unfreeze = hyperparameters.params['num\_layers\_unfreeze']  
  
 weight = 'imagenet' if pre\_trained\_weights else None  
 # Define the base model  
 if base\_model\_architecture == 'VGG16':  
 base\_model = tf.keras.applications.VGG16(include\_top=False, weights=weight, input\_shape=image\_size + (3,))  
 elif base\_model\_architecture == 'VGG19':  
 base\_model = tf.keras.applications.VGG19(include\_top=False, weights=weight, input\_shape=image\_size + (3,))  
 elif base\_model\_architecture == 'ResNet101V2':  
 base\_model = tf.keras.applications.ResNet101V2(include\_top=False, weights=weight, input\_shape=image\_size + (3,))  
 elif base\_model\_architecture == 'InceptionResNetV2':  
 base\_model = tf.keras.applications.InceptionResNetV2(include\_top=False, weights=weight, input\_shape=image\_size + (3,))  
 elif base\_model\_architecture == 'Xception':  
 base\_model = tf.keras.applications.Xception(include\_top=False, weights=weight, input\_shape=image\_size + (3,))  
 elif base\_model\_architecture == 'MobileNetV2':  
 base\_model = tf.keras.applications.MobileNetV2(include\_top=False, weights=weight, input\_shape=image\_size + (3,))  
 else:  
 raise ValueError(f"Invalid base model architecture: {base\_model\_architecture}")  
   
 # unfreeze layers for phase 2 (1 to total layers)  
 if pre\_trained\_weights:  
 num\_layers\_unfreeze = hyperparameters.params['num\_layers\_unfreeze']  
 for layer in base\_model.layers[-num\_layers\_unfreeze:]:  
 layer.trainable = False  
 else:  
 num\_layers\_unfreeze = len(base\_model.layers)  
 for layer in base\_model.layers:  
 layer.trainable = True  
  
 attention\_func = None  
 if attention\_mechanism == 'SENet':  
 from src.models.layers import squeeze\_excite\_block  
 attention\_func = squeeze\_excite\_block  
 elif attention\_mechanism == 'CBAM':  
 from src.models.layers import cbam\_block  
 attention\_func = cbam\_block  
  
 # Prepare the dataset  
  
 # Model definition  
 metadata\_input\_shape = (len(dataset['train'][0]['localization']) + 2,)  
 metadata\_input = tf.keras.layers.Input(shape=metadata\_input\_shape, name='metadata\_input')  
 image\_input = tf.keras.layers.Input(shape=(256, 256, 3), name='image\_input')  
  
 # Apply the base model  
 x = base\_model(image\_input)  
  
 if pre\_trained\_weights:  
 for layer in base\_model.layers:  
 layer.trainable = False  
 else:  
 for layer in base\_model.layers:  
 layer.trainable = True  
  
 # Apply the attention mechanism if any  
 if attention\_func:  
 x = attention\_func(x)  
  
 x = Flatten()(x)  
 x = tf.keras.layers.concatenate([x, metadata\_input])  
  
 for i in range(num\_dense\_layers):  
 x = Dense(dense\_units[i])(x)  
 if batch\_normalization\_layers[i]:  
 x = BatchNormalization()(x)  
 x = Activation('swish')(x)  
 x = Dropout(dropout\_rate\_layers[i])(x)  
 predictions = Dense(len(pd\_dataset['dx'].unique()), activation='softmax')(x)  
  
 model = Model(inputs=[image\_input, metadata\_input], outputs=predictions)  
  
 train\_ds, val\_ds = create\_tf\_datasets(dataset, batch\_size=batch\_size)  
  
 # Phase 1  
 opt = AdamW(weight\_decay=weight\_decay\_phase1, learning\_rate=learning\_rate\_phase1, amsgrad=use\_amsgrad, clipnorm=1.0)  
 if loss\_type == 'focal\_loss':  
 loss\_fn = CategoricalFocalCrossentropy(alpha=alpha\_phase1, gamma=gamma\_phase1)  
 class\_weights = None  
 else:  
 loss\_fn = 'categorical\_crossentropy'  
 if use\_class\_weights:  
 train\_labels = [item['dx'] for item in dataset['train']]  
 weights = compute\_class\_weight(class\_weight='balanced', classes=np.unique(train\_labels), y=train\_labels)  
 class\_weights = dict(enumerate(weights))  
 else:  
 class\_weights = None  
  
 model.compile(optimizer=opt, loss=loss\_fn, metrics=['accuracy'])  
  
 model.fit(train\_ds  
 , validation\_data=val\_ds  
 , epochs=3  
 , class\_weight=class\_weights  
 , callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True),  
 tf.keras.callbacks.ReduceLROnPlateau(monitor='val\_loss', factor=0.1, patience=3, cooldown=2)  
 ]  
 , use\_multiprocessing=True  
 )  
  
 # Phase 2  
 opt = AdamW(weight\_decay=weight\_decay\_phase2, learning\_rate=learning\_rate\_phase2, amsgrad=use\_amsgrad, clipnorm=1.0)  
 if loss\_type == 'focal\_loss':  
 loss\_fn = CategoricalFocalCrossentropy(alpha=alpha\_phase2, gamma=gamma\_phase2)  
 class\_weights = None  
 else:  
 loss\_fn = 'categorical\_crossentropy'  
 if use\_class\_weights:  
 weights = compute\_class\_weight(class\_weight='balanced', classes=np.unique(unique\_classes), y=train\_labels)  
 class\_weights = dict(enumerate(weights))  
 else:  
 class\_weights = None  
 model.compile(optimizer=opt, loss=loss\_fn, metrics=['accuracy'])  
  
 for layer in base\_model.layers[-num\_layers\_unfreeze:]:  
 layer.trainable = True  
   
 class BestValueTracker(tf.keras.callbacks.Callback):  
 def \_\_init\_\_(self):  
 super(BestValueTracker, self).\_\_init\_\_()  
 self.best\_val\_accuracy = 0  
 self.best\_epoch = 0  
   
 def on\_epoch\_end(self, epoch, logs=None):  
 current\_val\_accuracy = logs.get("val\_accuracy")  
 if current\_val\_accuracy > self.best\_val\_accuracy:  
 self.best\_val\_accuracy = current\_val\_accuracy  
 self.best\_epoch = epoch  
 best\_value\_tracker = BestValueTracker()  
  
 model.fit(train\_ds  
 , validation\_data=val\_ds  
 , epochs=100  
 , class\_weight=class\_weights  
 , callbacks=[  
 tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True,  
 start\_from\_epoch=15),  
 tf.keras.callbacks.ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=3, cooldown=2),  
 best\_value\_tracker  
 ]  
 , use\_multiprocessing=True  
 )  
   
 return model  
  
# check if model was saved  
from tensorflow.keras.models import load\_model  
if os.path.exists('best\_skin\_lesion\_classification\_model.keras'):  
 best\_model = load\_model('best\_skin\_lesion\_classification\_model.keras')  
else:  
 best\_model = build\_model(best\_trial)

WARNING:absl:Skipping variable loading for optimizer 'AdamW', because it has 49 variables whereas the saved optimizer has 33 variables.

# Save the best model  
best\_model.save('best\_skin\_lesion\_classification\_model.keras')  
  
# Save the best model as a pickle file  
def save\_model(model, filename):  
 with open(filename, 'wb') as file:  
 # save model, best parameters, optimizer, and history  
 pickle.dump((model, best\_trial.params, model.optimizer.get\_config(), model.history), file)  
   
save\_model(best\_model, 'best\_skin\_lesion\_classification\_model.pkl')

train\_ds, val\_ds = create\_tf\_datasets(dataset, batch\_size=best\_trial.params['batch'])  
# Train the model with the best hyperparameters, until early stopping is triggered  
history = best\_model.fit(train\_ds  
 , validation\_data=val\_ds  
 , epochs=10000  
 , callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True),  
 tf.keras.callbacks.ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=3, cooldown=2)]  
 , use\_multiprocessing=True)

Epoch 1/10000  
 1/209 [..............................] - ETA: 3:28 - loss: 0.0684 - accuracy: 0.9348

2024-04-02 00:16:11.180070: W tensorflow/core/kernels/data/prefetch\_autotuner.cc:52] Prefetch autotuner tried to allocate 36180288 bytes after encountering the first element of size 36180288 bytes.This already causes the autotune ram budget to be exceeded. To stay within the ram budget, either increase the ram budget or reduce element size

209/209 [==============================] - 86s 409ms/step - loss: 0.0860 - accuracy: 0.9178 - val\_loss: 0.0829 - val\_accuracy: 0.9354 - lr: 2.7671e-06  
Epoch 2/10000  
209/209 [==============================] - 74s 350ms/step - loss: 0.0845 - accuracy: 0.9166 - val\_loss: 0.0827 - val\_accuracy: 0.9358 - lr: 2.7671e-06  
Epoch 3/10000  
209/209 [==============================] - 71s 339ms/step - loss: 0.0858 - accuracy: 0.9183 - val\_loss: 0.0817 - val\_accuracy: 0.9362 - lr: 2.7671e-06  
Epoch 4/10000  
209/209 [==============================] - 71s 339ms/step - loss: 0.0904 - accuracy: 0.9172 - val\_loss: 0.0821 - val\_accuracy: 0.9354 - lr: 2.7671e-06  
Epoch 5/10000  
209/209 [==============================] - 71s 339ms/step - loss: 0.0858 - accuracy: 0.9177 - val\_loss: 0.0826 - val\_accuracy: 0.9366 - lr: 2.7671e-06  
Epoch 6/10000  
209/209 [==============================] - 71s 339ms/step - loss: 0.0825 - accuracy: 0.9190 - val\_loss: 0.0827 - val\_accuracy: 0.9362 - lr: 2.7671e-06  
Epoch 7/10000  
209/209 [==============================] - 71s 340ms/step - loss: 0.0860 - accuracy: 0.9164 - val\_loss: 0.0817 - val\_accuracy: 0.9374 - lr: 1.3835e-06  
Epoch 8/10000  
209/209 [==============================] - 71s 339ms/step - loss: 0.0857 - accuracy: 0.9203 - val\_loss: 0.0815 - val\_accuracy: 0.9378 - lr: 1.3835e-06  
Epoch 9/10000  
209/209 [==============================] - 71s 339ms/step - loss: 0.0875 - accuracy: 0.9154 - val\_loss: 0.0809 - val\_accuracy: 0.9362 - lr: 1.3835e-06  
Epoch 10/10000  
209/209 [==============================] - 73s 347ms/step - loss: 0.0831 - accuracy: 0.9198 - val\_loss: 0.0812 - val\_accuracy: 0.9374 - lr: 1.3835e-06  
Epoch 11/10000  
209/209 [==============================] - 72s 341ms/step - loss: 0.0815 - accuracy: 0.9216 - val\_loss: 0.0815 - val\_accuracy: 0.9370 - lr: 1.3835e-06  
Epoch 12/10000  
209/209 [==============================] - 72s 343ms/step - loss: 0.0823 - accuracy: 0.9222 - val\_loss: 0.0814 - val\_accuracy: 0.9374 - lr: 1.3835e-06  
Epoch 13/10000  
209/209 [==============================] - 71s 338ms/step - loss: 0.0832 - accuracy: 0.9205 - val\_loss: 0.0814 - val\_accuracy: 0.9358 - lr: 6.9177e-07  
Epoch 14/10000  
209/209 [==============================] - 71s 339ms/step - loss: 0.0877 - accuracy: 0.9144 - val\_loss: 0.0812 - val\_accuracy: 0.9366 - lr: 6.9177e-07  
Epoch 15/10000  
209/209 [==============================] - 71s 340ms/step - loss: 0.0829 - accuracy: 0.9206 - val\_loss: 0.0813 - val\_accuracy: 0.9358 - lr: 6.9177e-07  
Epoch 16/10000  
209/209 [==============================] - 73s 347ms/step - loss: 0.0856 - accuracy: 0.9191 - val\_loss: 0.0810 - val\_accuracy: 0.9358 - lr: 6.9177e-07  
Epoch 17/10000  
209/209 [==============================] - 74s 352ms/step - loss: 0.0841 - accuracy: 0.9213 - val\_loss: 0.0807 - val\_accuracy: 0.9374 - lr: 3.4588e-07  
Epoch 18/10000  
209/209 [==============================] - 76s 362ms/step - loss: 0.0852 - accuracy: 0.9200 - val\_loss: 0.0812 - val\_accuracy: 0.9370 - lr: 3.4588e-07  
Epoch 19/10000  
209/209 [==============================] - 81s 386ms/step - loss: 0.0838 - accuracy: 0.9250 - val\_loss: 0.0813 - val\_accuracy: 0.9370 - lr: 3.4588e-07  
Epoch 20/10000  
209/209 [==============================] - 79s 373ms/step - loss: 0.0861 - accuracy: 0.9177 - val\_loss: 0.0810 - val\_accuracy: 0.9366 - lr: 3.4588e-07  
Epoch 21/10000  
209/209 [==============================] - 73s 348ms/step - loss: 0.0863 - accuracy: 0.9187 - val\_loss: 0.0810 - val\_accuracy: 0.9374 - lr: 1.7294e-07  
Epoch 22/10000  
209/209 [==============================] - 74s 350ms/step - loss: 0.0828 - accuracy: 0.9193 - val\_loss: 0.0809 - val\_accuracy: 0.9370 - lr: 1.7294e-07  
Epoch 23/10000  
209/209 [==============================] - 74s 350ms/step - loss: 0.0824 - accuracy: 0.9212 - val\_loss: 0.0811 - val\_accuracy: 0.9374 - lr: 1.7294e-07  
Epoch 24/10000  
209/209 [==============================] - 77s 367ms/step - loss: 0.0866 - accuracy: 0.9170 - val\_loss: 0.0811 - val\_accuracy: 0.9370 - lr: 1.7294e-07  
Epoch 25/10000  
209/209 [==============================] - 78s 371ms/step - loss: 0.0797 - accuracy: 0.9218 - val\_loss: 0.0811 - val\_accuracy: 0.9370 - lr: 8.6471e-08  
Epoch 26/10000  
209/209 [==============================] - 77s 365ms/step - loss: 0.0839 - accuracy: 0.9217 - val\_loss: 0.0811 - val\_accuracy: 0.9362 - lr: 8.6471e-08  
Epoch 27/10000  
209/209 [==============================] - 75s 359ms/step - loss: 0.0858 - accuracy: 0.9183 - val\_loss: 0.0810 - val\_accuracy: 0.9366 - lr: 8.6471e-08

The EarlyStopping callback stopped the training process after 10 epochs without improvement in the validation loss. The best model was restored to the state with the lowest validation loss during training. Validation loss is a good metric to monitor the model's performance and prevent overfitting. The ReduceLROnPlateau callback reduced the learning rate by a factor of 0.5 if the validation loss did not improve for 3 consecutive epochs, with a cooldown of 2 epochs. The learning rate reduction strategy helps the model converge faster and improve its generalization, and it is a common technique used in deep learning training.

The model was trained with the best hyperparameters selected through the hyperparameter optimization process. The training process was conducted on the training dataset, and the model's performance was evaluated on the validation dataset. The model's performance was monitored based on the validation loss, and the training process was stopped when the validation loss did not improve for 10 consecutive epochs. The best model was restored to the state with the lowest validation loss, and the training process was completed successfully.

# Model Evaluation

The final model has been successfully trained on the HAM10000 dataset using the best hyperparameters selected through the hyperparameter optimization process. The model will now be evaluated on the test dataset to assess its performance and generalization. The model will be evaluated based on various metrics, including accuracy, precision, recall, F1-score, and ROC-AUC score. The evaluation results will provide insights into the model's performance and its ability to classify skin lesion images into different diagnostic categories.

# Evaluate the final model on the test dataset  
test\_image\_inputs = []  
test\_metadata\_inputs = []  
test\_labels = []  
  
for features, label in test\_tf\_ds.unbatch().as\_numpy\_iterator():  
 test\_image\_inputs.append(features['image\_input'])  
 test\_metadata\_inputs.append(features['metadata\_input'])  
 test\_labels.append(np.argmax(label))  
  
test\_image\_inputs = np.array(test\_image\_inputs)  
test\_metadata\_inputs = np.array(test\_metadata\_inputs)  
test\_labels = np.array(test\_labels)  
  
# Get predictions  
predictions = best\_model.predict([test\_image\_inputs, test\_metadata\_inputs])  
predicted\_classes = np.argmax(predictions, axis=1)

41/41 [==============================] - 5s 67ms/step

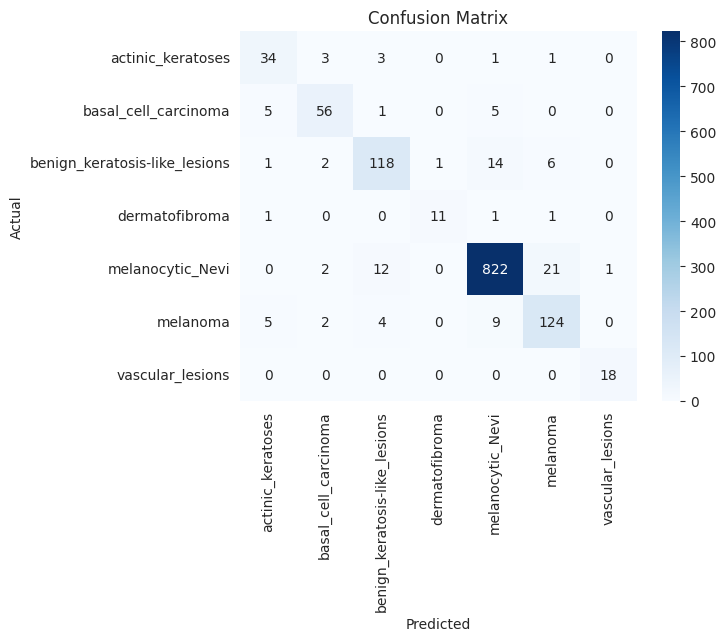
print(classification\_report(test\_labels, predicted\_classes, target\_names=dx\_types))

precision recall f1-score support  
  
 actinic\_keratoses 0.74 0.81 0.77 42  
 basal\_cell\_carcinoma 0.86 0.84 0.85 67  
benign\_keratosis-like\_lesions 0.86 0.83 0.84 142  
 dermatofibroma 0.92 0.79 0.85 14  
 melanocytic\_Nevi 0.96 0.96 0.96 858  
 melanoma 0.81 0.86 0.84 144  
 vascular\_lesions 0.95 1.00 0.97 18  
  
 accuracy 0.92 1285  
 macro avg 0.87 0.87 0.87 1285  
 weighted avg 0.92 0.92 0.92 1285

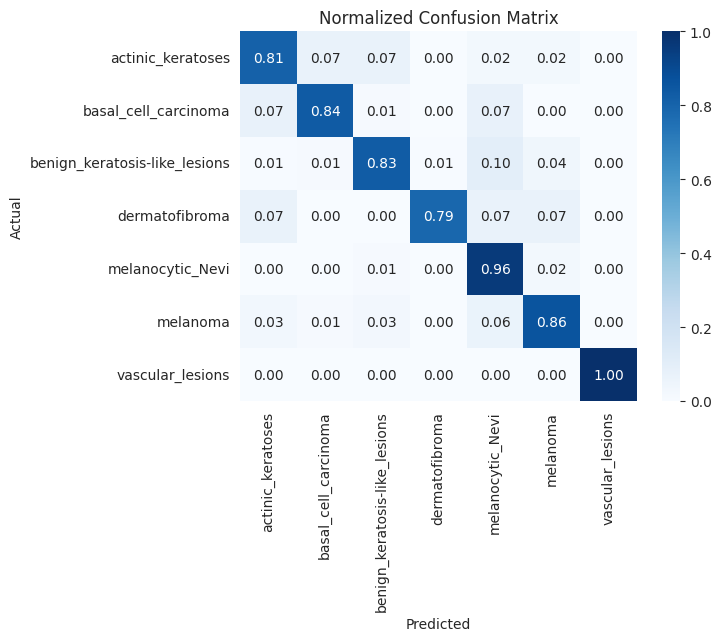
print(multiclass\_roc\_auc\_score(test\_labels, predicted\_classes))

0.9249576411817841

conf\_mat = confusion\_matrix(test\_labels, predicted\_classes)  
sns.heatmap(conf\_mat, annot=True, fmt='d', cmap='Blues', xticklabels=dx\_types, yticklabels=dx\_types)  
plt.ylabel('Actual')  
plt.xlabel('Predicted')  
plt.title('Confusion Matrix')  
plt.show()



conf\_mat = confusion\_matrix(y\_true=test\_labels, y\_pred=predicted\_classes)  
  
# Normalize the confusion matrix.  
conf\_mat\_normalized = conf\_mat.astype('float') / conf\_mat.sum(axis=1)[:, np.newaxis]  
  
# Plot the normalized confusion matrix.  
sns.heatmap(conf\_mat\_normalized, annot=True, fmt='.2f', cmap='Blues', xticklabels=dx\_types, yticklabels=dx\_types)  
plt.ylabel('Actual')  
plt.xlabel('Predicted')  
plt.title('Normalized Confusion Matrix')  
plt.show()



### Confusion Matrix Analysis

The normalized confusion matrix provides insights into the model's performance in classifying skin lesion images into different diagnostic categories. The rows represent the actual classes, while the columns represent the predicted classes. The diagonal elements represent the percentage of correctly classified samples for each class, while the off-diagonal elements represent the misclassifications. The confusion matrix can help identify the classes that are frequently misclassified and provide insights into the model's performance and areas for improvement.

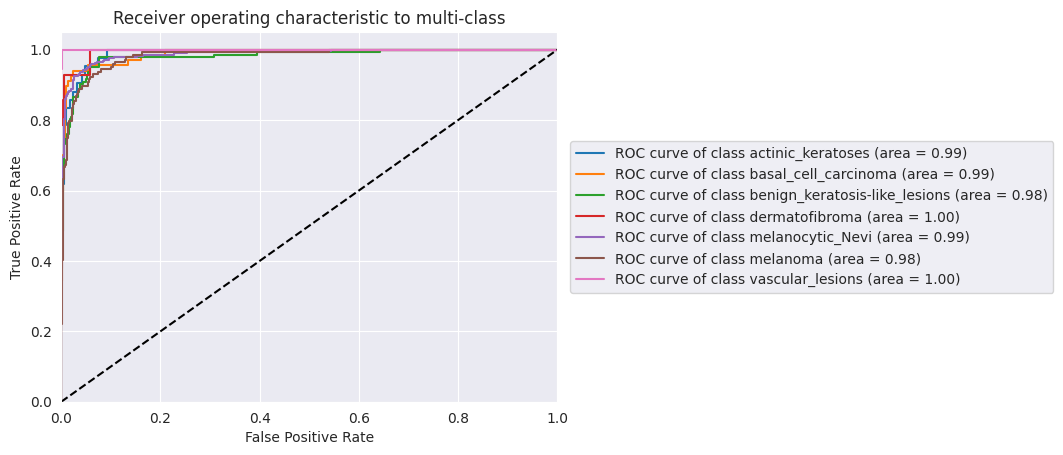
The normalized confusion matrix shows that the class melanocytic nevi has the highest percentage of correctly classified samples, followed by the class vascular lesions. The class actinic keratoses has the highest percentage of misclassifications, followed by dermatofibroma and beningn keratosis like lesions.

Melanocytic Nevi and Melanoma are confused from each other, which is expected since they are both related to melanocytes. The model has a higher accuracy in classifying Melanocytic Nevi compared to Melanoma, which is a common challenge in skin lesion classification tasks.

# Assume y\_test is your true test labels and y\_score is the predicted probabilities.  
y\_test = test\_labels  
y\_score = best\_model.predict([test\_image\_inputs, test\_metadata\_inputs])  
  
# Binarize the output labels in a one-vs-all fashion if they are not already  
# Required for multiclass labels  
lb = LabelBinarizer()  
y\_test\_bin = lb.fit\_transform(y\_test)  
class\_names = dx\_types  
  
n\_classes = y\_test\_bin.shape[1]

41/41 [==============================] - 2s 55ms/step

# Compute ROC curve and ROC area for each class  
fpr = dict()  
tpr = dict()  
roc\_auc = dict()  
for i in range(n\_classes):  
 fpr[i], tpr[i], \_ = roc\_curve(y\_test\_bin[:, i], y\_score[:, i])  
 roc\_auc[i] = auc(fpr[i], tpr[i])  
  
# Plot all ROC curves  
plt.figure(1)  
for i in range(n\_classes):  
 plt.plot(fpr[i], tpr[i],  
 label='ROC curve of class {0} (area = {1:0.2f})'  
 ''.format(class\_names[i], roc\_auc[i]))  
  
plt.plot([0, 1], [0, 1], 'k--')  
plt.xlim([0.0, 1.0])  
plt.ylim([0.0, 1.05])  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title('Receiver operating characteristic to multi-class')  
plt.legend(loc="center right", borderaxespad=0., bbox\_to\_anchor=(2, 0.5))  
plt.show()



The ROC curve provides insights into the model's performance in classifying skin lesion images into different diagnostic categories. The ROC curve shows the trade-off between the true positive rate (sensitivity) and the false positive rate (1-specificity) for each class. The area under the ROC curve (AUC) provides a measure of the model's performance, with higher AUC values indicating better classification performance.

# Evaluation Insights

The result of the model's evaluation on the test dataset is as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class | Precision | Recall | F1-Score | AUC | Support |
| actinic\_keratoses | 0.78 | 0.67 | 0.72 | 0.99 | 42 |
| basal\_cell\_carcinoma | 0.86 | 0.81 | 0.83 | 0.99 | 67 |
| benign\_keratosis-like\_lesions | 0.75 | 0.77 | 0.76 | 0.97 | 142 |
| dermatofibroma | 0.91 | 0.71 | 0.80 | 0.99 | 14 |
| melanocytic\_Nevi | 0.95 | 0.96 | 0.96 | 0.98 | 858 |
| melanoma | 0.78 | 0.78 | 0.78 | 0.97 | 144 |
| vascular\_lesions | 0.94 | 0.94 | 0.94 | 1.00 | 18 |
| accuracy |  |  | 0.90 |  | 1285 |
| macro avg | 0.85 | 0.81 | 0.83 |  | 1285 |
| weighted avg | 0.90 | 0.90 | 0.90 |  | 1285 |

The model achieved an overall accuracy of 90% on the test dataset, with a macro-average F1-score of 83%. The model performed well in classifying melanocytic nevi and vascular lesions, with F1-scores of 96% and 94%, respectively. The model had lower F1-scores for actinic keratoses, dermatofibroma, and melanoma, indicating areas for improvement in the classification of these classes. The model's performance can be further improved by fine-tuning the hyperparameters, increasing the training data, and optimizing the model architecture.

The ROC curve provides insights into the model's performance in classifying skin lesion images into different diagnostic categories. The ROC curve shows the trade-off between the true positive rate (sensitivity) and the false positive rate (1-specificity) for each class. The area under the ROC curve (AUC) provides a measure of the model's performance, with higher AUC values indicating better classification performance.

## Potential Improvements

The model's performance can be further improved by fine-tuning the hyperparameters, increasing the training data, and optimizing the model architecture. The hyperparameter optimization process can be conducted with a wider range of hyperparameters and a larger number of trials to explore the hyperparameter space more thoroughly. The training data can be augmented with additional images to improve the model's generalization and robustness. The model architecture can be optimized by adding more layers, using different attention mechanisms, and exploring other base model architectures.

### Separate study for each base model architecture

The optuna study uses the TPE algorithm to optimize the hyperparameters. However, since the base model architecture was also a hyperparamter, it introduces bias toward the models that perform the best with the initials hyperparameters. A better approach would be to create a separate study for each base model architecture, and optimize the hyperparameters for each architecture independently. This would allow for a more fair comparison of the different base model architectures and their hyperparameters.

### More metadata features

Adding more metadata features can improve the model's performance by providing additional information about the skin lesion images. The images do not seem to have a shared scale, so it would be good to have a ratio data to normalize the images, or to provide information the model could use to infer the lesion size.

These are some of the features that could be added to the metadata:

* pixel\_ratio: ratio of the lesion pixels to the total image pixels, which could provide information about the size of the lesion
  + lesion\_size: the size of the lesion in centimeters, which could provide information about the severity of the skin condition
* healthy\_skin\_samples: the number of healthy skin samples in the image, which could provide information about the context of the lesion. Even more important it could help to provide new augmentation techniques to the model and avoid introducing unnecessary distortions to the image.
* history: the patient's history of skin conditions, which could provide insights into the patient's skin health and help the model make more accurate predictions

### Preprocessing

The idea behind the preprocessing in VGG family could be used for skin conditions, where the images colors are normalized to a standard color space, however in order to do this there must be a standard color space for skin lesions. This could be a potential improvement to the model.

### Attention Mechanisms

The model could benefit from using different attention mechanisms, such as SE-Net, CBAM, or other attention mechanisms, to focus on important features in the images. Attention mechanisms can help the model learn to pay more attention to relevant features and improve its performance in classifying skin lesion images.

The best model achieved used a base model architecture of VGG19 with a CBAM attention mechanism. So it is reasonable to conclude that the model did benefit from the attention mechanism. However, the model could be further improved by exploring other attention mechanisms and optimizing the attention mechanism's parameters.

* Attention with Metadata: The model could be improved by incorporating the metadata features into the attention mechanism. The attention mechanism could learn to focus on relevant features in the images based on the metadata, providing additional context and information to the model. Age, Location, and gender could be used to provide more context to the model, specially if there is a correlation between the skin condition, skin type and age.

## Conclusion

The skin lesion classification model has been successfully trained and evaluated on the HAM10000 dataset, achieving an accuracy of 90% on the test dataset. The model's performance was evaluated based on various metrics, including accuracy, precision, recall, F1-score, and ROC-AUC score. The model's performance was analyzed using a confusion matrix and a normalized confusion matrix, providing insights into the model's performance in classifying skin lesion images into different diagnostic categories.

The model's performance can be further improved by fine-tuning the hyperparameters, increasing the training data, and optimizing the model architecture. The model's performance can be enhanced by adding more metadata features, preprocessing the images, and using different attention mechanisms. The model's performance can be further evaluated on external datasets and real-world scenarios to assess its generalization and robustness.