Skin Disease Classification

Using ML and DL models

Datasets Used

https://challenge.isic-archive.com/data/#2017 : 2000 lesion images in JPEG format and 2000 corresponding superpixel masks in PNG format, with EXIF data stripped.

HAM10000 : **a large collection of multi-source dermatoscopic images of common pigmented skin lesions**

Need for this study

Skin cancer, a prevalent disease, demands accurate classification for early treatment. This study evaluates ML and DL models, crucial for diagnosis.

The first dataset from ISIC

- 2000 lesion images in JPEG format and 2000 corresponding superpixel masks in PNG format, with EXIF data stripped

Models employed

- SVC
- AdaBoost and DecisionTree
- LDA
- QDA
- K-Neighbors
- Multi-Layer-Perceptron

SVC

SVC is a specific implementation of the Support Vector Machine (SVM) algorithm that is designed specifically for classification tasks

SVM works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable. A separator between the categories is found, then the data are transformed in such a way that the separator could be drawn as a hyperplane.

Our implementation

```python
from sklearn.svm import SVC

# Train SVM model
model = SVC()
model.fit(X_train, Y_train)

# SVM Evaluation
accuracy = model.score(X_test, Y_test)
print("Accuracy:", accuracy)
```

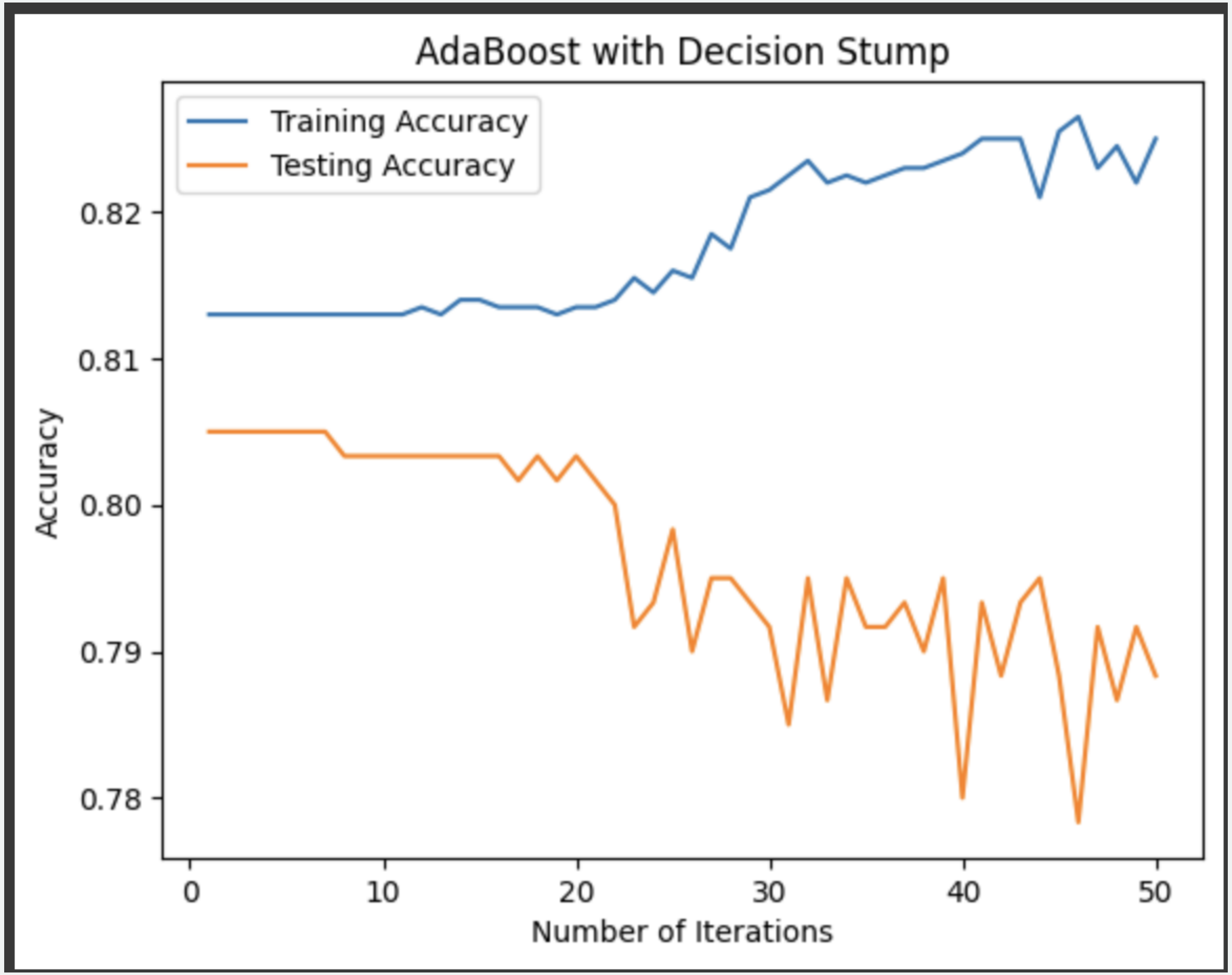
```
Accuracy: 0.805
```

AdaBoost and DecisionTree Classifier

- Here, we used a decision stump as a weak learner and trained an adaboost model

```python
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

# Train AdaBoost with Decision Stump as weak learner
num_iterations = 50
accuracies_train = []
accuracies_test = []
for i in range(1, num_iterations+1):
    # Create AdaBoost classifier with Decision Stump as weak learner
    ada_boost =  AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), algorithm="SAMME", n_estimators=i)
    ada_boost.fit(X_train, Y_train)

    # Calculate accuracy on training set
    accuracy_train = ada_boost.score(X_train, Y_train)
    accuracies_train.append(accuracy_train)

    # Calculate accuracy on testing set
    accuracy_test = ada_boost.score(X_test, Y_test)
    accuracies_test.append(accuracy_test)

    # print(f"Iteration {i}: Training Accuracy = {accuracy_train:.4f}, Testing Accuracy = {accuracy_test:.4f}")
```

AdaBoost and Decision Tree



LDA

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Train LDA model
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, Y_train)

#  Evaluate LDA model
accuracy_lda_train = lda.score(X_train, Y_train)
accuracy_lda_test = lda.score(X_test, Y_test)
print("LDA Training Accuracy:", accuracy_lda_train)
print("LDA Testing Accuracy:", accuracy_lda_test)
```

```
LDA Training Accuracy: 0.829
LDA Testing Accuracy: 0.785
```

QDA

```python
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

#  Train QDA model
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, Y_train)

#  Evaluate QDA model
accuracy_qda_train = qda.score(X_train, Y_train)
accuracy_qda_test = qda.score(X_test, Y_test)
print("QDA Training Accuracy:", accuracy_qda_train)
print("QDA Testing Accuracy:", accuracy_qda_test)
```

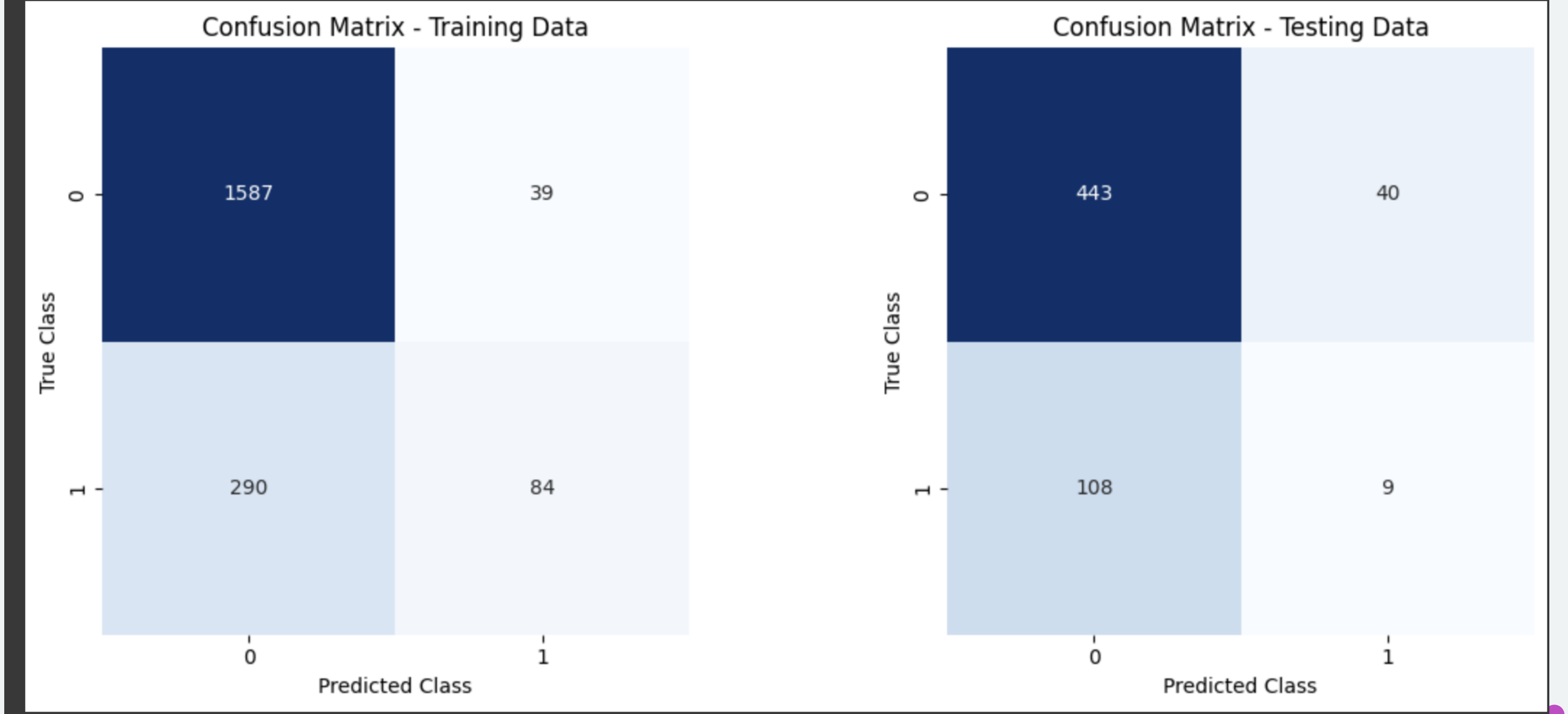
```
QDA Training Accuracy: 0.8845
QDA Testing Accuracy: 0.8033333333333333
```

K-Neighbours

```python
from sklearn.neighbors import KNeighborsClassifier

# Train KNN model
knn = KNeighborsClassifier(n_neighbors=5)  # You can adjust the number of neighbors (k)
knn.fit(X_train, Y_train)

# Predict and evaluate the model
Y_pred_train = knn.predict(X_train)
accuracy_train = accuracy_score(Y_train, Y_pred_train)
print("Training Accuracy:", accuracy_train)

Y_pred_test = knn.predict(X_test)
accuracy_test = accuracy_score(Y_test, Y_pred_test)
print("Testing Accuracy:", accuracy_test)


# Print confusion matrix for training and testing datasets
conf_matrix_train = confusion_matrix(Y_train, Y_pred_train)
conf_matrix_test = confusion_matrix(Y_test, Y_pred_test)
```

K-Neighbours

```
Training Accuracy: 0.8355
Testing Accuracy: 0.7533333333333333
```

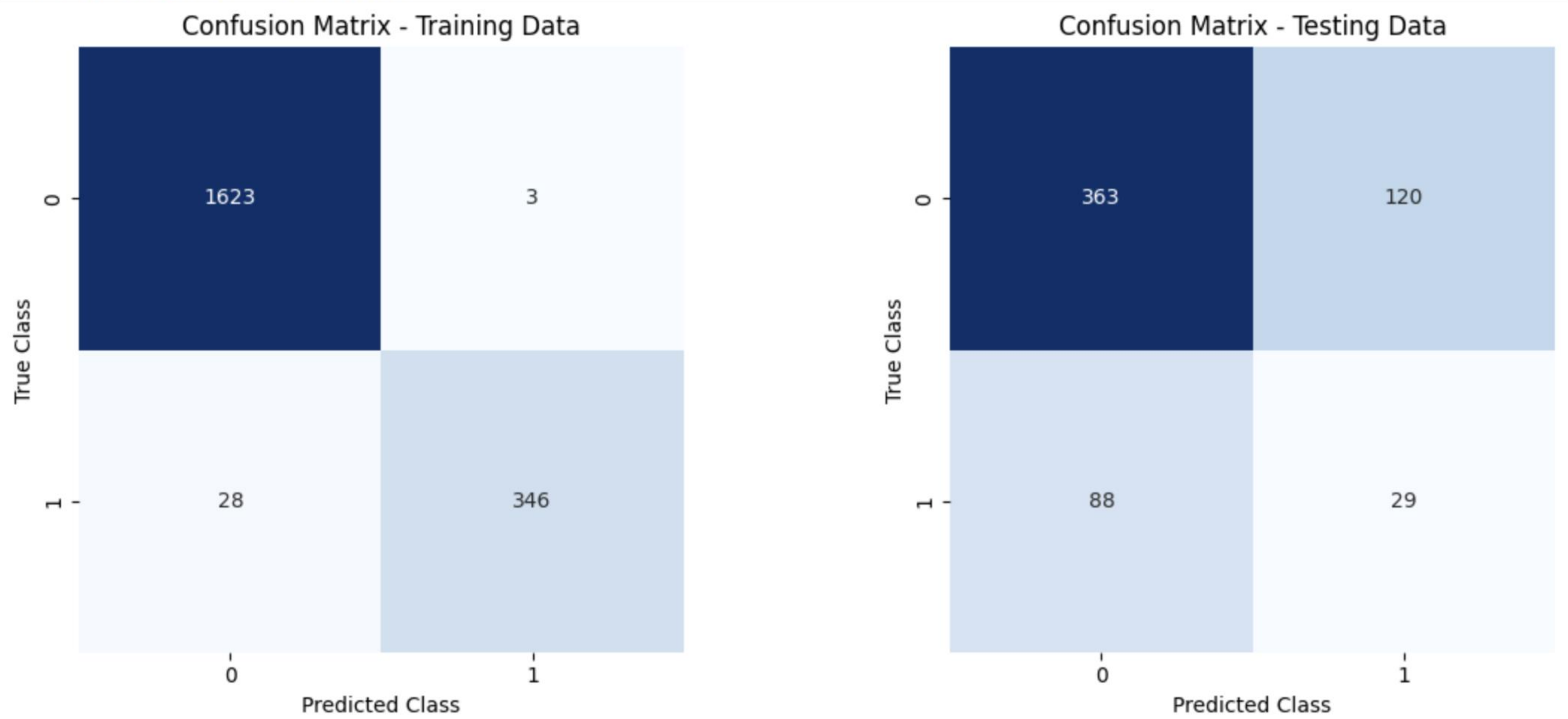


Multi-Layer Perceptron

```
from sklearn.neural_network import MLPClassifier

# Train MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=1000)  # You can adjust the parameters as needed
mlp.fit(X_train, Y_train)

# Predict and evaluate the model
Y_pred_train = mlp.predict(X_train)
accuracy_train = accuracy_score(Y_train, Y_pred_train)
print("Training Accuracy:", accuracy_train)
Y_pred_test = mlp.predict(X_test)
accuracy_test = accuracy_score(Y_test, Y_pred_test)
print("Testing Accuracy:", accuracy_test)

# Print confusion matrix for training and testing datasets
conf_matrix_train = confusion_matrix(Y_train, Y_pred_train)
conf_matrix_test = confusion_matrix(Y_test, Y_pred_test)
```

Multi-Layer Perceptron

```
Training Accuracy: 0.9845
Testing Accuracy: 0.6533333333333333
```



HAM10000 Dataset

- This was a much bigger dataset and allowed us to achieve greater test accuracies

Pre-Processing

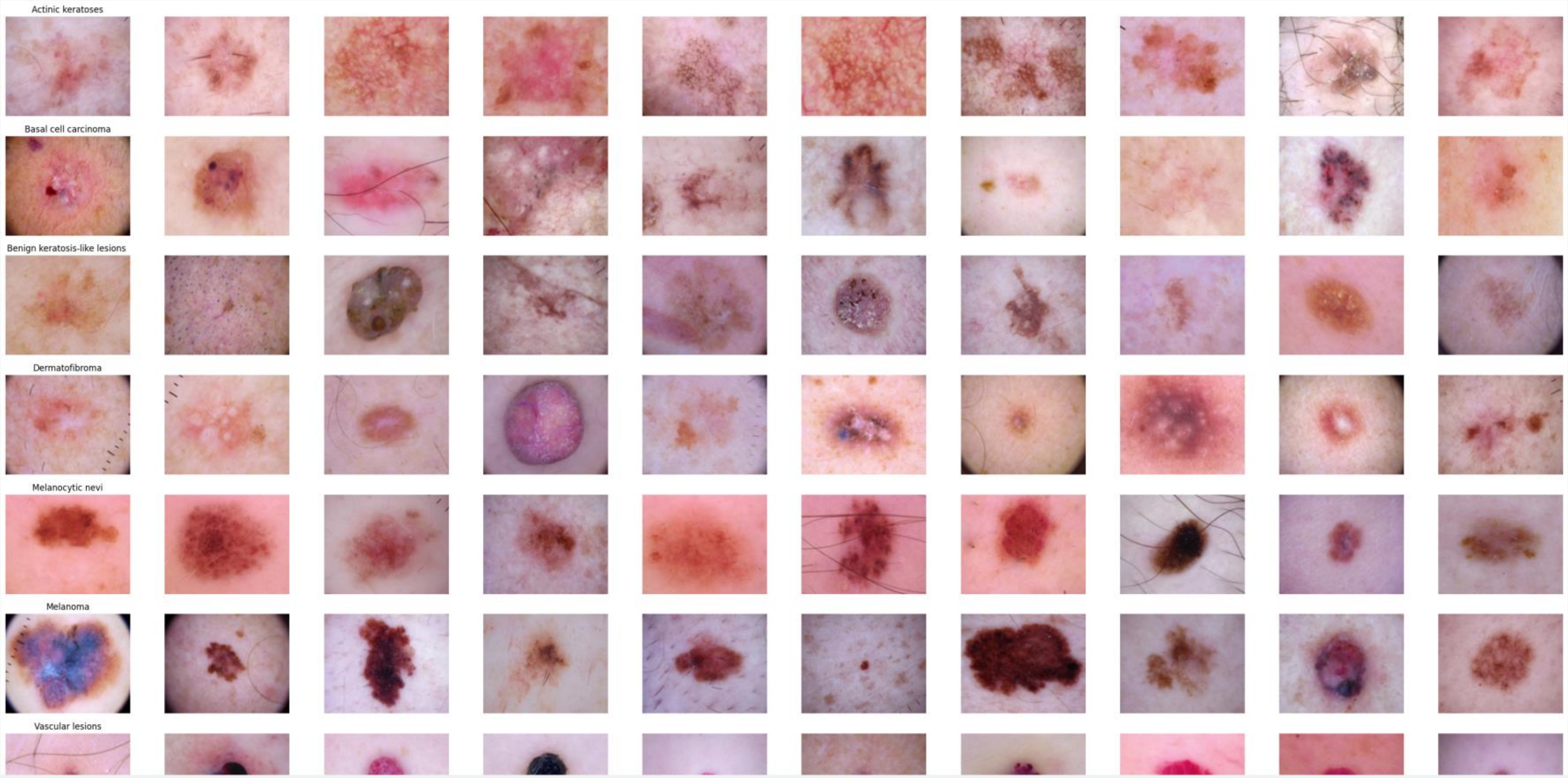
- We had to do certain pre-processing of data, as we found some data with weird values
- We replaced the entries with null age values with the median age
- We also removed all entries with age = 0 and unknown gender

```
df= df[df['age'] != 0]
df= df[df['sex'] != 'unknown']
```

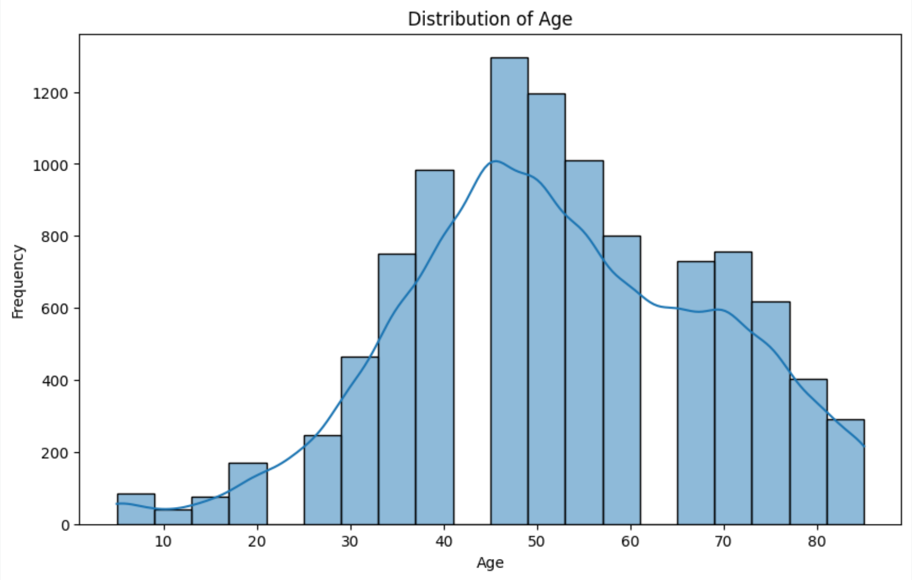
```
df['age'].fillna(int(df['age'].median()),inplace=True)
df.isnull().sum()
```

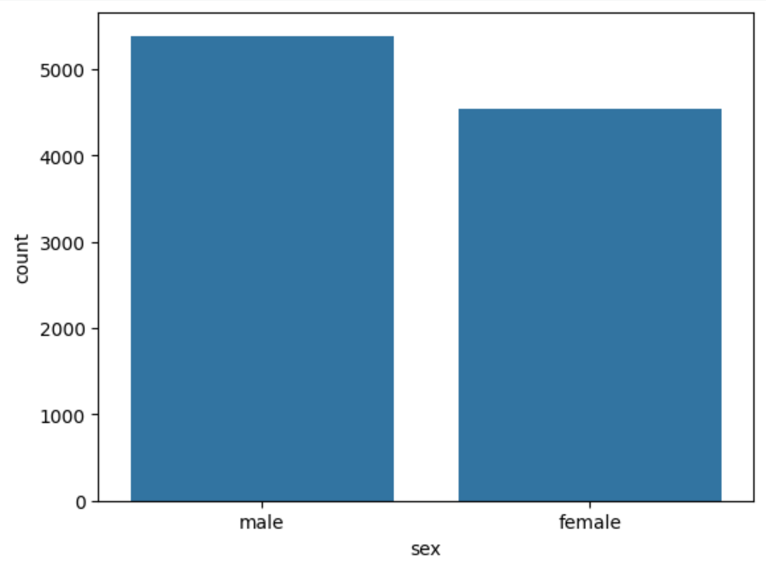
```
C:\Users\tejas\AppData\Local\Temp\ipykernel_2332\355994
The behavior will change in pandas 3.0. This inplace me

For example, when doing 'df[col].method(value, inplace=


  df['age'].fillna(int(df['age'].median()),inplace=True
lesion_id       0
image_id        0
dx              0
dx_type         0
age             0
sex             0
localization    0
dtype: int64
```

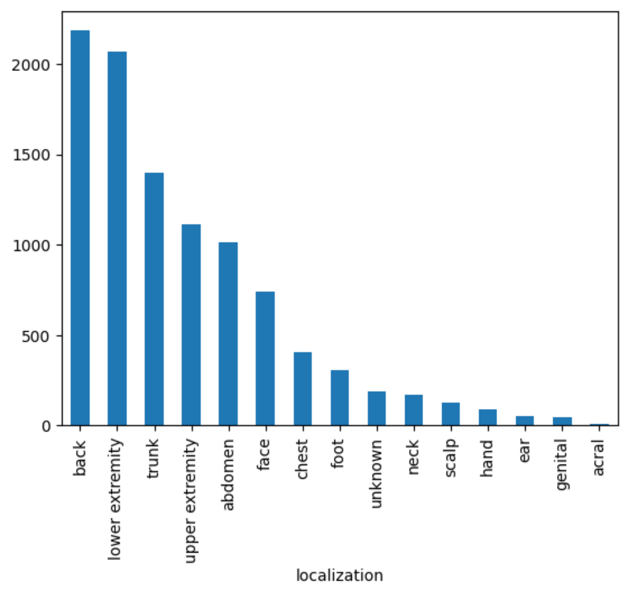
Some images from the dataset



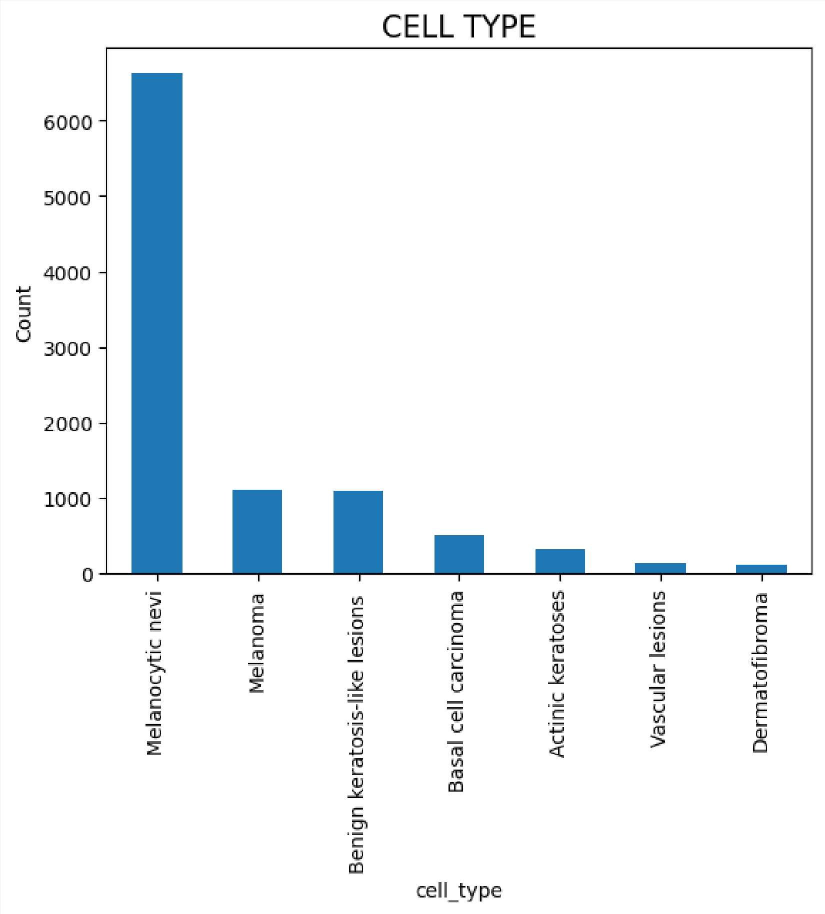
We made certain plots to have an idea of the data distribution

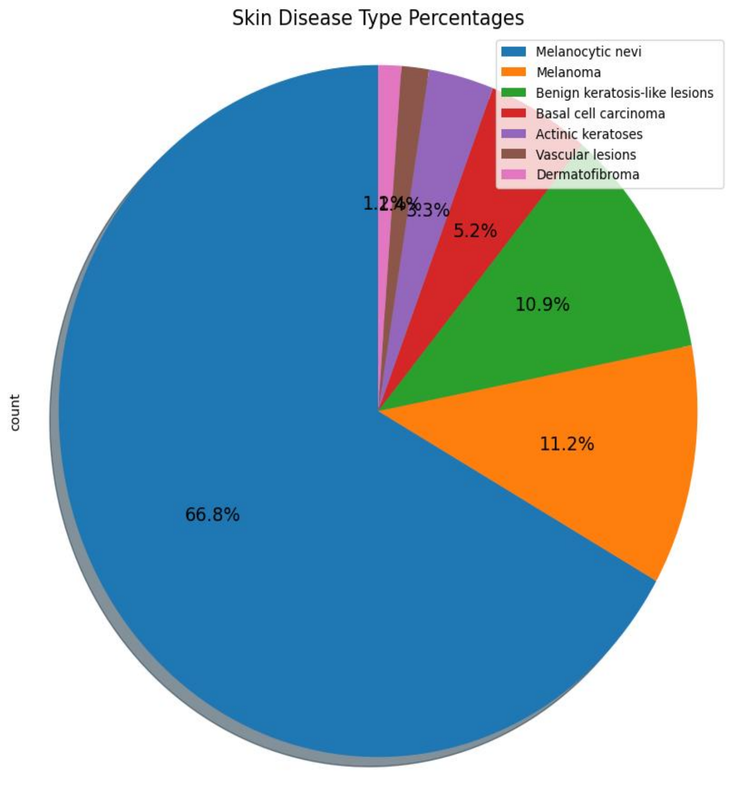


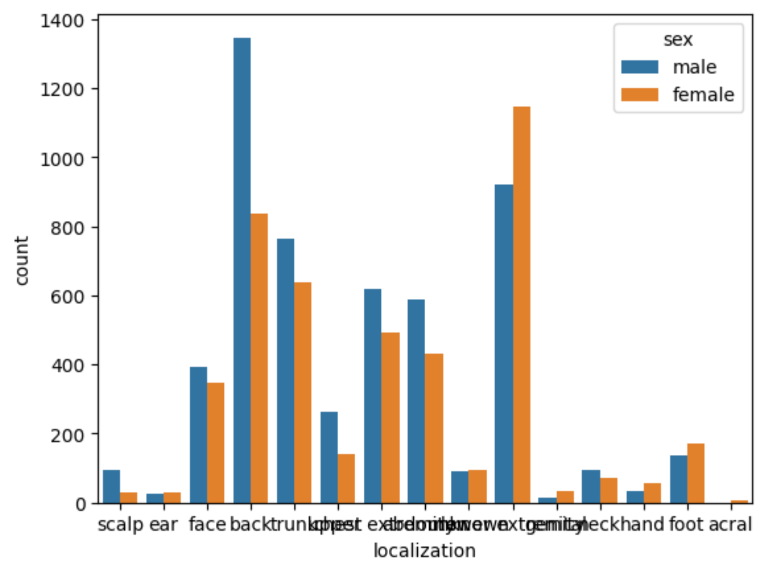


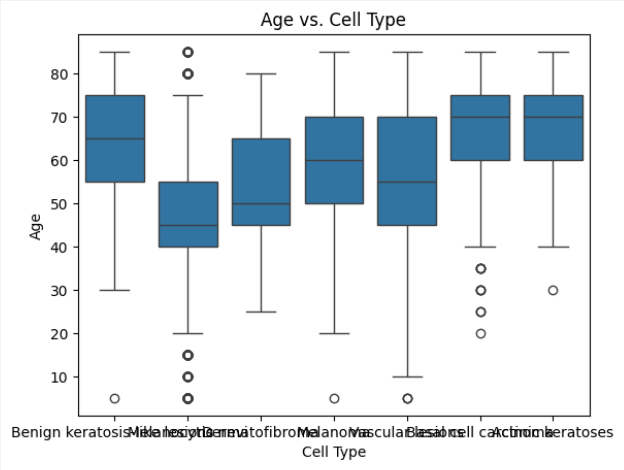


Some more plots









Models employed

- LDA
- AdaBoost (Using decision stumps)
- Decision Tree
- Multi-Layer-Perceptron
- CNN

LDA

```
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score
import numpy as np


# Create and fit the LDA model
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)

# Predict the labels for the test data
y_pred = lda.predict(X_test)

# Calculate the accuracy of the predictions
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.6325159556600605
```

AdaBoost with Decision Stumps

- We had to reduce train this on a smaller dataset in order to be able to effectively test it, otherwise this took way too long. Even running this on 100 elements took 5 minutes.

```python
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

smallX_train = X_train[:100]
smallY_train = y_train[:100]
smallX_test = X_test[:100]
smallY_test = y_test[:100]

# Initialize the AdaBoost classifier with decision stumps as weak learners
clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),
                        n_estimators=400,
                        learning_rate=1)

# Fit the classifier to the training data
clf.fit(smallX_train, smallY_train)

# Predict the labels for the test data
y_pred = clf.predict(X_test)

# Calculate and print the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of the AdaBoost classifier with decision stumps:", accuracy)
```

```
c:\Users\tejas\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklear
  warnings.warn(
Accuracy of the AdaBoost classifier with decision stumps: 0.6076587168290225
```

Decision Tree

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Separate features and target


# Create a decision tree classifier
dt_model = DecisionTreeClassifier()

# Train the model
dt_model.fit(X_train, y_train)

# Predict on the test set
y_pred = dt_model.predict(X_test)

# Calculate and print accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of the decision tree classifier:", accuracy)
```

```
Accuracy of the decision tree classifier: 0.6123614376889486
```

Multi-Layer Perceptron

- This was taking way too long to train and we thought it isn't working so we halted it midway, and yet achieved satisfactory results. Again, allowing this to run completely wasn't allowed by colab as it exceeded the ram limits and so this would have to do.

```
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import numpy as np

# Assuming df['image'] contains the image data in the form of numpy arrays

# Convert image data to flattened numpy arrays


# Create and train the MLP classifier
mlp = MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='adam', max_iter=1000)
mlp.fit(X_train, y_train)

# Evaluate the model
y_pred = mlp.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print("Test Accuracy of MLP :", accuracy) #it was taking too long so I interrupted it. But the po
```

```
c:\Users\tejas\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\neural_network\_
  warnings.warn("Training interrupted by user.")
Test Accuracy of MLP : 0.6627477326167283
```

CNN without Dropout

- Finally, we used CNN, without dropout and the results were unsatisfactory, to say the least.

```
model = Sequential()

model.add(Dense(units= 64, kernel_initializer = 'uniform', activation = 'relu', input_dim = 37500))
model.add(Dense(units= 64, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dense(units= 64, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dense(units= 64, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dense(units = 7, kernel_initializer = 'uniform', activation = 'softmax'))

optimizer = tf.keras.optimizers.Adam(learning_rate = 0.00075,
                                    beta_1 = 0.9,
                                    beta_2 = 0.999,
                                    epsilon = 1e-8)

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

# fit the keras model on the dataset
history = model.fit(x_train, y_train, batch_size = 10, epochs = 5)

accuracy = model.evaluate(x_test, y_test, verbose=1)[1]
print("Test: accuracy = ",accuracy*100,"%")
```

```
Epoch 1/5
670/670 ━━━━━━━━━━━━━━━━━━━━ 20s 28ms/step - accuracy: 0.6586 - loss: 1.0731
Epoch 2/5
670/670 ━━━━━━━━━━━━━━━━━━━━ 20s 30ms/step - accuracy: 0.6836 - loss: 0.9026
Epoch 3/5
670/670 ━━━━━━━━━━━━━━━━━━━━ 23s 34ms/step - accuracy: 0.6872 - loss: 0.8767
Epoch 4/5
670/670 ━━━━━━━━━━━━━━━━━━━━ 23s 34ms/step - accuracy: 0.7029 - loss: 0.8412
Epoch 5/5
670/670 ━━━━━━━━━━━━━━━━━━━━ 23s 34ms/step - accuracy: 0.7175 - loss: 0.7993
78/78 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7104 - loss: 0.8315
Test: accuracy =  70.8182156085968 %
```

Model Summary

```
model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
dense_13 (Dense)	(None, 64)	2,400,064
dense_14 (Dense)	(None, 64)	4,160
dense_15 (Dense)	(None, 64)	4,160
dense_16 (Dense)	(None, 64)	4,160
dense_17 (Dense)	(None, 7)	455

Total params: 7,238,999 (27.61 MB)
Trainable params: 2,412,999 (9.20 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 4,826,000 (18.41 MB)

CNN (with dropout)

- Having studied that dropout helps in increasing test accuracy, we tried this. Unfortunately, it didn't work out

```
Epoch 1/5
76/76 ━━━━━━━━━━━━━━━━━━━━ 60s 760ms/step - accuracy: 0.6665 - loss: 1.1530 - val_accuracy: 0.6924 - val_loss: 1.1242 - learning_rate: 1.0000e-04
Epoch 2/5
 1/76 ━━━━━━━━━━━━━━━━━━━━ 50s 673ms/step - accuracy: 0.6719 - loss: 0.9920c:\Users\tejas\AppData\Local\Programs\Python\Python311\Lib\contextlib.py:155: UserWarnin
  self.gen.throw(typ, value, traceback)
76/76 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.6719 - loss: 0.9920 - val_accuracy: 0.6924 - val_loss: 1.1120 - learning_rate: 1.0000e-04
Epoch 3/5
76/76 ━━━━━━━━━━━━━━━━━━━━ 63s 811ms/step - accuracy: 0.6555 - loss: 1.0251 - val_accuracy: 0.6998 - val_loss: 0.9643 - learning_rate: 1.0000e-04
Epoch 4/5
76/76 ━━━━━━━━━━━━━━━━━━━━ 3s 32ms/step - accuracy: 0.6406 - loss: 0.9435 - val_accuracy: 0.6998 - val_loss: 0.9598 - learning_rate: 1.0000e-04
Epoch 5/5
76/76 ━━━━━━━━━━━━━━━━━━━━ 73s 941ms/step - accuracy: 0.6578 - loss: 0.9950 - val_accuracy: 0.6998 - val_loss: 0.9279 - learning_rate: 1.0000e-04
```

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

```
78/78 ━━━━━━━━━━━━━━━━━━━━ 5s 66ms/step - accuracy: 0.6871 - loss: 0.9559
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 0.7004 - loss: 0.8942
Validation: accuracy = 0.699816  ;  loss_v = 0.927865
Test: accuracy = 0.683595  ;  loss = 0.960181
```

Our implementation of CNN with dropout

Model: "sequential_8"

Layer (type)	Output Shape	Param #
conv2d_30 (Conv2D)	(None, 100, 125, 32)	896
conv2d_31 (Conv2D)	(None, 100, 125, 32)	9,248
max_pooling2d_15 (MaxPooling2D)	(None, 50, 62, 32)	0
dropout_20 (Dropout)	(None, 50, 62, 32)	0

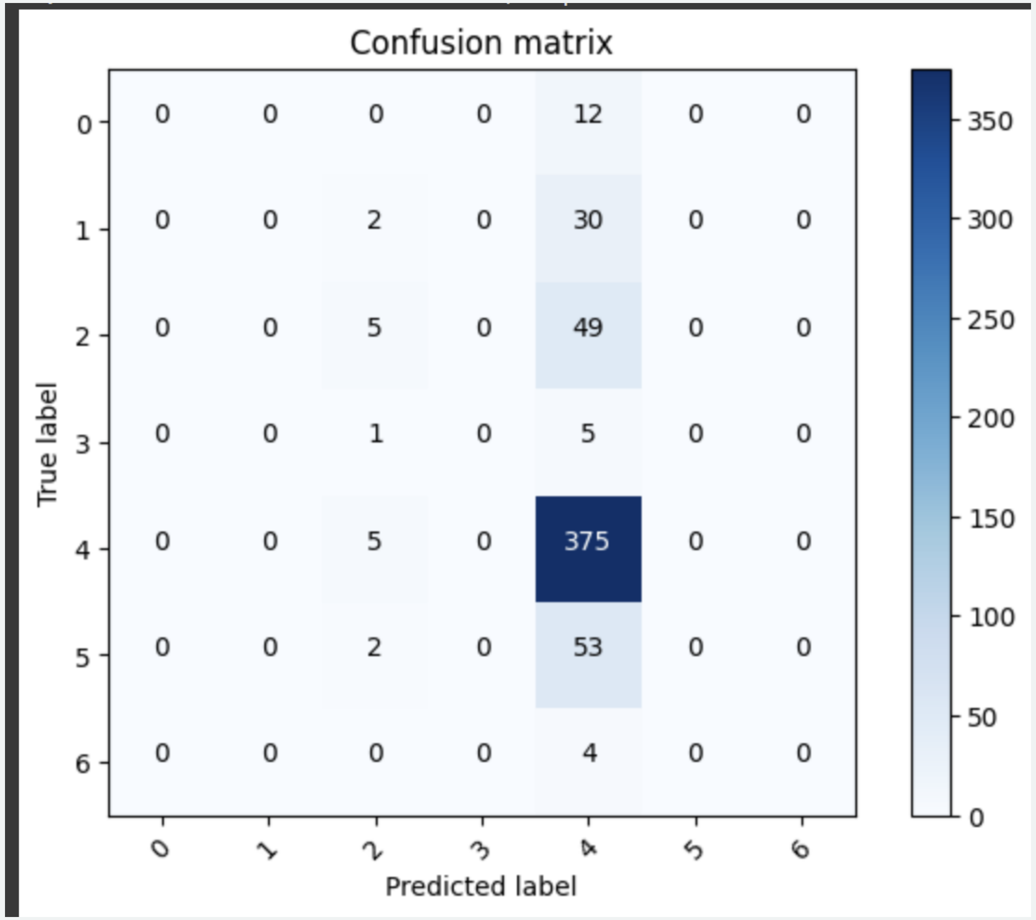
```
loss, accuracy = model.evaluate(x_test, y_test, verbose=1)
loss_v, accuracy_v = model.evaluate(x_validate, y_validate, verbose=1)
print("Validation: accuracy = %f  ;  loss_v = %f" % (accuracy_v, loss_v))
print("Test: accuracy = %f  ;  loss = %f" % (accuracy, loss))
```

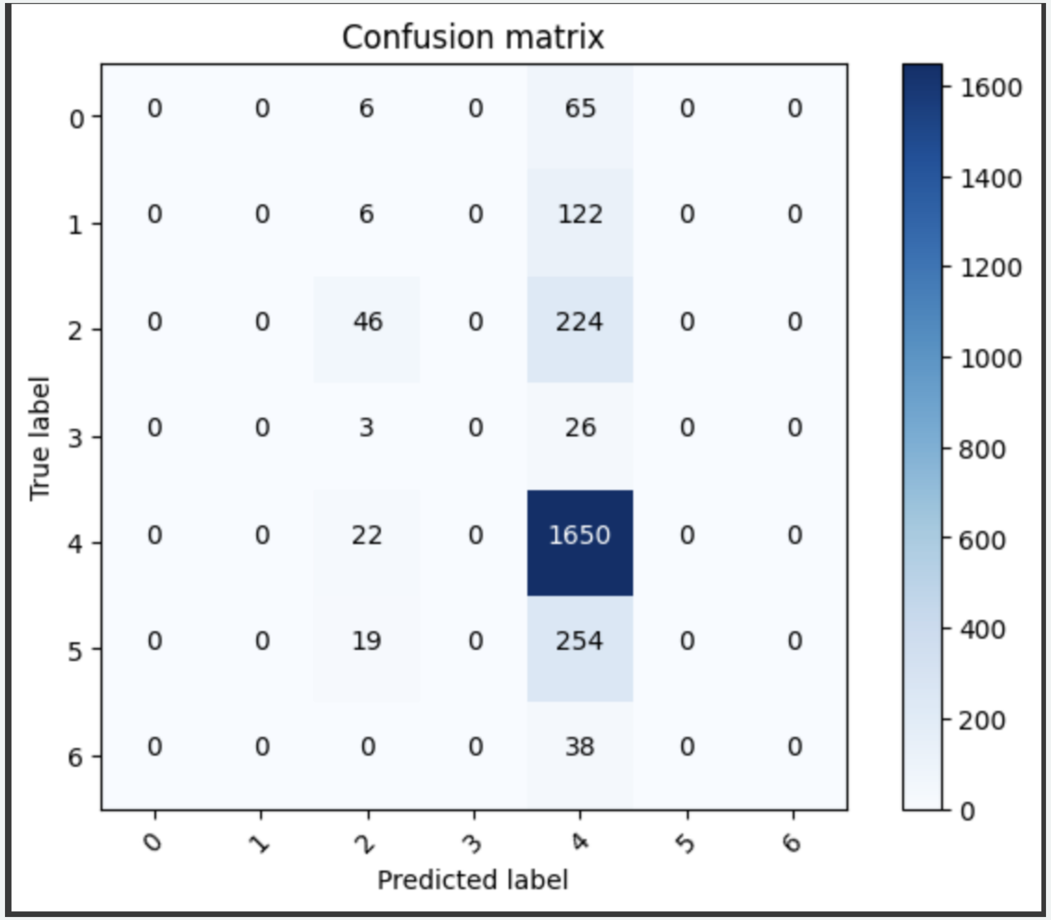
```
78/78 ━━━━━━━━━━━━━━━━━━━━ 5s 66ms/step - accuracy: 0.6871 - loss: 0.9559
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 0.7004 - loss: 0.8942
Validation: accuracy = 0.699816  ;  loss_v = 0.927865
Test: accuracy = 0.683595  ;  loss = 0.960181
```

```
)
datagen.fit(x_train)
```

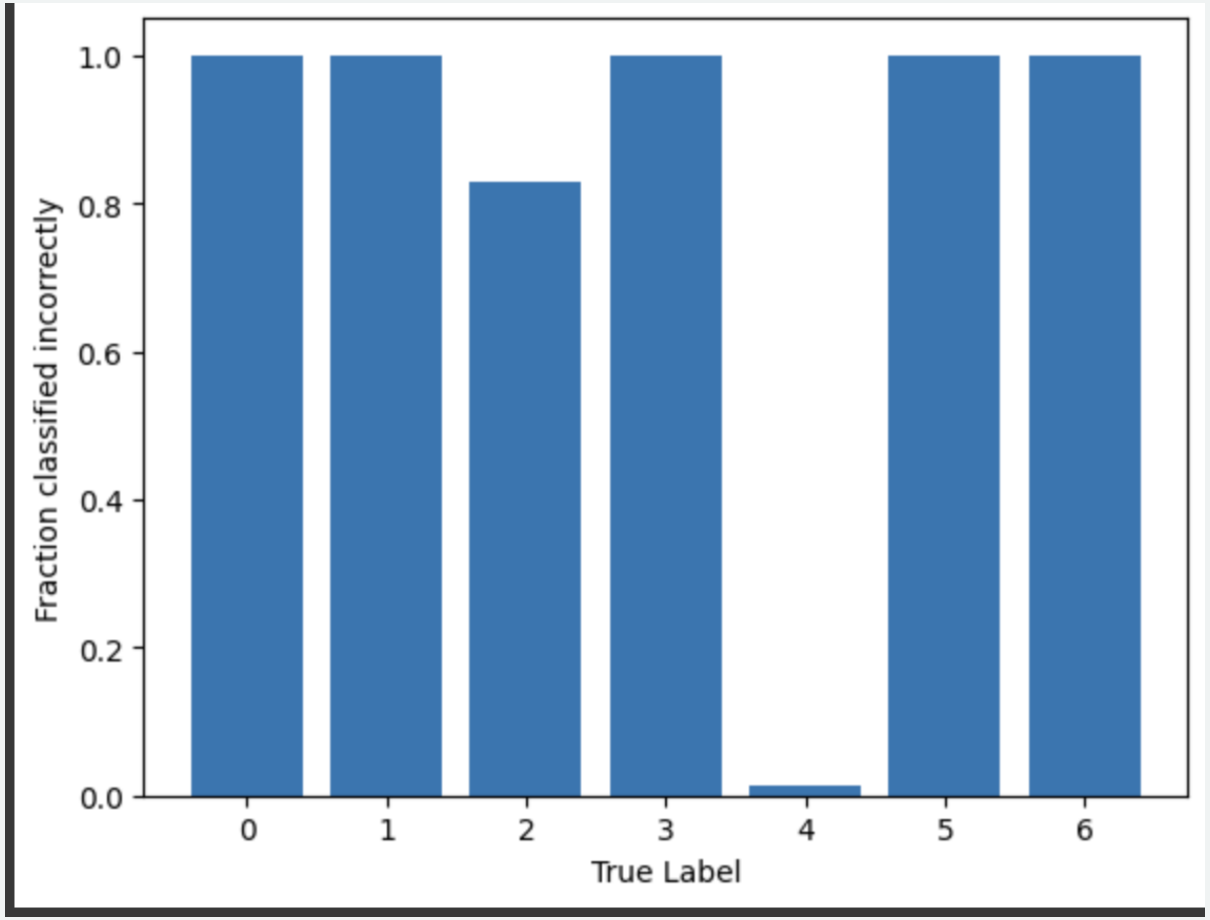
```
[ ] epochs = 5
    batch_size = 64

    # Train the model
    history = model.fit(
        datagen.flow(x_train, y_train, batch_size=batch_size),
        epochs=epochs,
        validation_data=(x_validate, y_validate),
        verbose=1,
        steps_per_epoch=x_train.shape[0] // batch_size,
        callbacks=[learning_rate_reduction]
    )
```

Confusion Matrices (val and test)





Misclassified points



Observation

There was a very clear bias of our model towards the label ‘4’ which was Actinic keratoses

This was a turning out to be a disaster and so we realised that out approach of reducing the number of epochs isn’t working out, and we need to achieve faster training times somehow. So, we reduced the dimensions of the input images even further and hoped things would work out.

A new start

- We tried the same thing with the smaller images

```
model = Sequential()
model.add(Conv2D(16, kernel_size = (3,3), input_shape = (28, 28, 3), activation='relu', padding = 'same'))
model.add(MaxPool2D(pool_size = (2,2)))
model.add(tf.keras.layers.BatchNormalization())

model.add(Conv2D(32, kernel_size = (3,3), activation = 'relu'))
model.add(Conv2D(64, kernel_size = (3,3), activation = 'relu'))

model.add(MaxPool2D(pool_size = (2,2)))

model.add(tf.keras.layers.BatchNormalization())

model.add(Conv2D(128, kernel_size = (3,3), activation = 'relu'))
model.add(Conv2D(256, kernel_size = (3,3), activation = 'relu'))

model.add(Flatten())
model.add(tf.keras.layers.Dropout(0.2))
model.add(Dense(256,activation='relu'))

model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.2))
model.add(Dense(128,activation='relu'))

model.add(tf.keras.layers.BatchNormalization())
model.add(Dense(64,activation='relu'))

model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.2))
model.add(Dense(32,activation='relu'))

model.add(tf.keras.layers.BatchNormalization())
model.add(Dense(7,activation='softmax'))

model.summary()
```

Model: "sequential_9"

Layer (type)	Output Shape	Param #
conv2d_36 (Conv2D)	(None, 28, 28, 16)	448
max_pooling2d_18 (MaxPooling2D)	(None, 14, 14, 16)	0
batch_normalization (BatchNormalization)	(None, 14, 14, 16)	64
conv2d_37 (Conv2D)	(None, 12, 12, 32)	4,640
conv2d_38 (Conv2D)	(None, 10, 10, 64)	18,496
max_pooling2d_19 (MaxPooling2D)	(None, 5, 5, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 5, 5, 64)	256
conv2d_39 (Conv2D)	(None, 3, 3, 128)	73,856
conv2d_40 (Conv2D)	(None, 1, 1, 256)	295,168
flatten_6 (Flatten)	(None, 256)	0
dropout_24 (Dropout)	(None, 256)	0
dense_33 (Dense)	(None, 256)	65,792
batch_normalization_2 (BatchNormalization)	(None, 256)	1,024
dropout_25 (Dropout)	(None, 256)	0
dense_34 (Dense)	(None, 128)	32,896
batch_normalization_3 (BatchNormalization)	(None, 128)	512
dense_35 (Dense)	(None, 64)	8,256
batch_normalization_4 (BatchNormalization)	(None, 64)	256
dropout_26 (Dropout)	(None, 64)	0
dense_36 (Dense)	(None, 32)	2,080
batch_normalization_5 (BatchNormalization)	(None, 32)	128
dense_37 (Dense)	(None, 7)	231

Total params: 504,103 (1.92 MB)
Trainable params: 502,983 (1.92 MB)
Non-trainable params: 1,120 (4.38 KB)

```
#reference: https://www.kaggle.com/dhruv1234/ham10000-skin-disease-classification
callback = tf.keras.callbacks.ModelCheckpoint(filepath='best_model.keras',
                                              monitor='val_acc',
                                              mode='max',
                                              verbose=1,
                                              save_best_only=True)
```

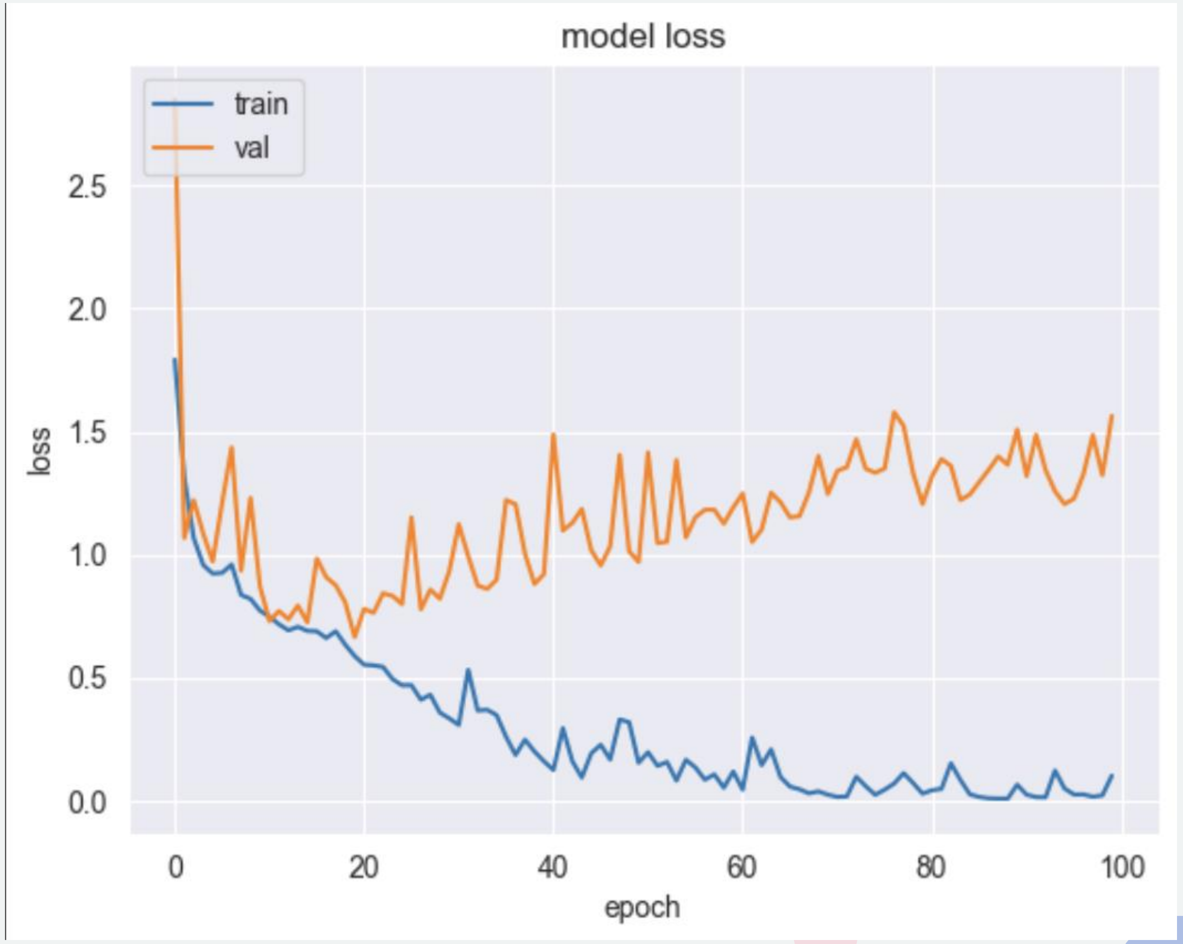
```
optimizer=tf.keras.optimizers.Adam(learning_rate=0.001)

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

```
history = model.fit(x_train,
                    y_train,
                    validation_split=0.2,
                    batch_size = 128,
                    epochs = 100,
                    shuffle=True,
                    callbacks=[callback])
```

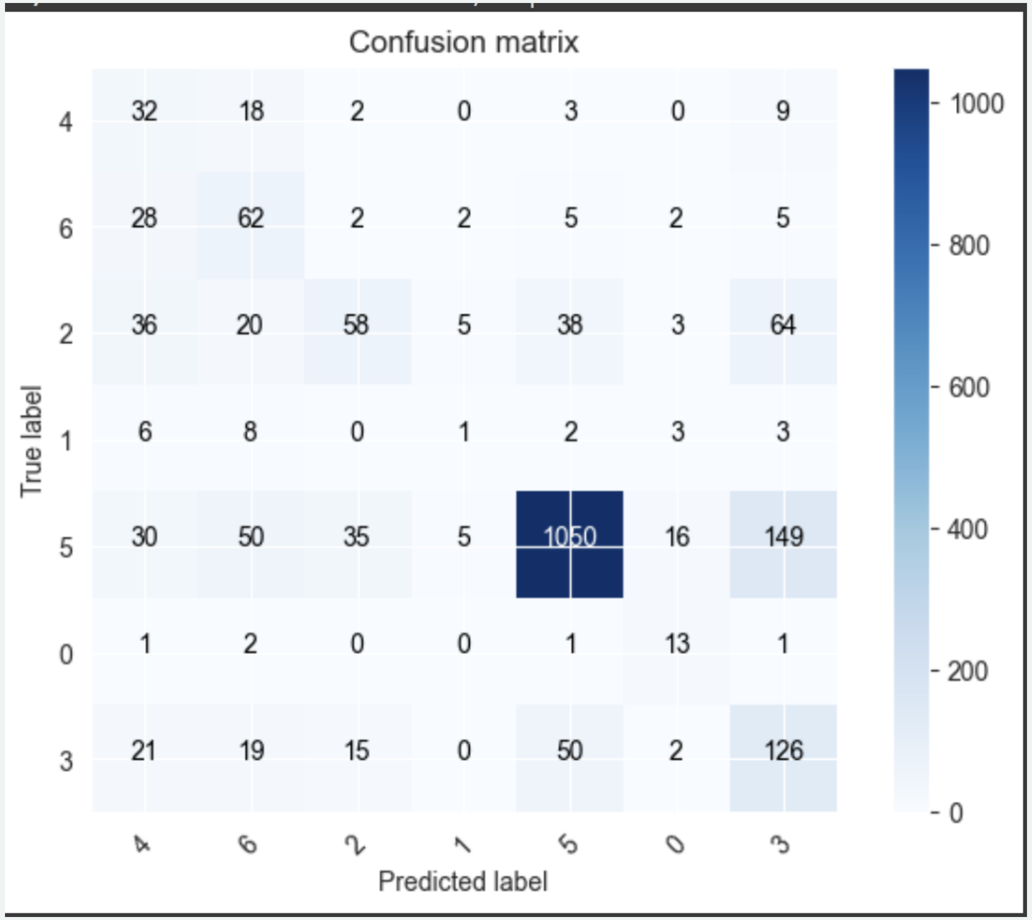
This time around, we succeeded

```
Epoch 98/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - accuracy: 0.9948 - loss: 0.0168 - val_accuracy: 0.7367 - val_loss: 1.4865
Epoch 99/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.9946 - loss: 0.0174 - val_accuracy: 0.7455 - val_loss: 1.3228
Epoch 100/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.9718 - loss: 0.0906 - val_accuracy: 0.6825 - val_loss: 1.5634
```

Some plots to go with our new model



Confusion Matrix



Input

- We can give it images and the model can now predict what sort of skin disease it is. Of course, it's not very accurate, but we think, this should be satisfactory for the scope of this course.

```
#Prediction
import PIL
image = PIL.Image.open('skin-cancer-mnist-ham10000\HAM10000_images_part_1\ISIC_0024306.jpg')
image = image.resize((28, 28))
img = np.array(image)
y_pred = model.predict(np.array([img]))
# print(y_pred)
print("Predicted Class:",classes[np.argmax(y_pred, axis=1)[0]] )
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
Predicted Class: melanocytic nevi
```

Thank You