

Python Code:

```
from flask import
Flask,render_template,request


import mysql.connector
import os
import numpy as np
import pandas as pd
from PIL import Image
from glob import glob
from sklearn.preprocessing import LabelEncoder, StandardScaler
import plotly.graph_objects as go


from plotly.subplots import make_subplots
import matplotlib.pyplot as plt
import random
import keras
import pydot
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import tensorflow as tf
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten,
BatchNormalization, Dropout, Dense, MaxPool2D
from tensorflow.keras.callbacks import ReduceLROnPlateau,
EarlyStopping
from tensorflow.keras import regularizers
from tensorflow.keras.optimizers import Adamax
from IPython.display import display


from keras.models import load_model
# import cv2
import numpy as np


UPLOAD_FOLDER = 'static/file/'
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER


@app.route('/')
@app.route('/login')
def login():
    return render_template('login.html')
```

```

@app.route('/validate',methods=['POST','GET'])
def validate():
    if request.method == 'POST':
        data1 = request.form.get('username')
        data2 = request.form.get('password')
        mydb =
mysql.connector.connect(host="localhost",user="root",password="",d
atabase="skin")
        mycursor = mydb.cursor()
        sql = "SELECT * FROM `users` WHERE `name` = %s AND
`password` = %s"
        val = (data1, data2)
        mycursor.execute(sql,val)
        account = mycursor.fetchone()
        if account:
            return render_template('index.html')
        else:
            return render_template('login.html',msg = 'Invalid')

@app.route('/register')
def register():
    return render_template('reg.html')

@app.route('/upload',methods=['POST','GET'])
def upload():
    if request.method == 'POST':
        name = request.form.get('name')
        phone = request.form.get('phone')
        password = request.form.get('password')
        mydb =
mysql.connector.connect(host="localhost",user="root",password="",d
atabase="skin")
        mycursor = mydb.cursor()
        sql = "INSERT INTO users (`name`, `phone`, `password`)
VALUES (%s, %s, %s)"
        val = (name, phone, password)
        mycursor.execute(sql, val)
        mydb.commit()
        return render_template('login.html')

@app.route('/value',methods=['POST','GET'])
def value():
    if request.method == 'POST':
        file_name = request.files['file']
        imgfile = os.path.join(app.config['UPLOAD_FOLDER'],
file_name.filename)

```

```

file_name.save(imgfile)

data_directory = os.path.join('static/')
os.listdir(data_directory)

data = pd.read_csv(os.path.join(data_directory,
'HAM10000_metadata.csv'))

imageid_path_dict =
{os.path.splitext(os.path.basename(x))[0]: x
    for x in
glob(os.path.join(data_directory, '*', '*.jpg'))}

lesion_type_dict = {
    'nv': 'Melanocytic nevi (nv)',
    'mel': 'Melanoma (mel)',
    'bkl': 'Benign keratosis-like lesions (bkl)',
    'bcc': 'Basal cell carcinoma (bcc)',
    'akiec': 'Actinic keratoses (akiec)',
    'vasc': 'Vascular lesions (vasc)',
    'df': 'Dermatofibroma (df)'
}
label_mapping = {
    0: 'nv',
    1: 'mel',
    2: 'bkl',
    3: 'bcc',
    4: 'akiec',
    5: 'vasc',
    6: 'df'
}
reverse_label_mapping = dict((value, key) for key, value
in label_mapping.items())

data['cell_type'] = data['dx'].map(lesion_type_dict.get)
data['path'] = data['image_id'].map(imageid_path_dict.get)

#print(data.head())

data['image_pixel'] = data['path'].map(lambda x:
np.asarray(Image.open(x).resize((28,28))))

#print(data.sample(5))

fig = make_subplots(rows=2, cols = 2,

```

```

        subplot_titles = ['Sex',
'Localisation', 'Age', 'Skin Type'],
        vertical_spacing = 0.15,
        column_widths = [0.4, 0.6])

fig.add_trace(go.Bar(
    x = data['sex'].value_counts().index,
    y = data['sex'].value_counts()),
    row = 1, col = 1)
fig.add_trace(go.Bar(
    x =
data['localization'].value_counts().index,
    y = data['localization'].value_counts()),
    row = 1, col = 2)
fig.add_trace(go.Histogram(
    x = data['age']),
    row = 2, col = 1)
fig.add_trace(go.Bar(
    x =
data['dx'].value_counts().index.map(lesion_type_dict.get),
    y = data['dx'].value_counts()),
    row = 2, col = 2)

for i in range(4):
    fig.update_yaxes(title_text = 'Count', row = i//2+1,
col = i%2+1)
    fig.update_layout(title = 'Distribution of Data',
height = 800)

#fig.show()

sample_data = data.groupby('dx').apply(lambda df:
df.iloc[:2, [9, 7]])
plt.figure(figsize = (22, 32))
for i in range(14):
    plt.subplot(7, 5, i + 1)
    plt.imshow(np.squeeze(sample_data['image_pixel'][i]))
    img_label = sample_data['cell_type'][i]
    plt.title(img_label)
    plt.axis("off")
#plt.show()

# # print(data.info())

#print(data.isnull().sum())

```

```

# # Handling null values
data['age'].fillna(value=int(data['age'].mean()),
inplace=True)
# # Converting dtype of age to int32
data['age'] = data['age'].astype('int32')

data['label'] = data['dx'].map(reverse_label_mapping.get)

#print(data.sample(5))

data = data.sort_values('label')
data = data.reset_index()

index1 = data[data['label'] == 1].index.values
index2 = data[data['label'] == 2].index.values
index3 = data[data['label'] == 3].index.values
index4 = data[data['label'] == 4].index.values
index5 = data[data['label'] == 5].index.values
index6 = data[data['label'] == 6].index.values

df_index1 = data.iloc[int(min(index1)):int(max(index1)+1)]
df_index2 = data.iloc[int(min(index2)):int(max(index2)+1)]
df_index3 = data.iloc[int(min(index3)):int(max(index3)+1)]
df_index4 = data.iloc[int(min(index4)):int(max(index4)+1)]
df_index5 = data.iloc[int(min(index5)):int(max(index5)+1)]
df_index6 = data.iloc[int(min(index6)):int(max(index6)+1)]

df_index1 = df_index1.append([df_index1]*4, ignore_index =
True)
df_index2 = df_index2.append([df_index2]*4, ignore_index =
True)
df_index3 = df_index3.append([df_index3]*11, ignore_index
= True)
df_index4 = df_index4.append([df_index4]*17, ignore_index
= True)
df_index5 = df_index5.append([df_index5]*45, ignore_index
= True)
df_index6 = df_index6.append([df_index6]*52, ignore_index
= True)

frames = [data, df_index1, df_index2, df_index3,
df_index4, df_index5, df_index6]
final_data = pd.concat(frames)

#print(data.shape)
#print(final_data.shape)

```

```

fig = make_subplots(rows = 2, cols = 2,
                    subplot_titles = ['Sex',
'Localisation', 'Age', 'Skin Type'],
                    vertical_spacing = 0.15,
                    column_widths = [0.4, 0.6])

fig.add_trace(go.Bar(
    x = final_data['sex'].value_counts().index,
    y = final_data['sex'].value_counts()),
    row = 1, col = 1)
fig.add_trace(go.Bar(
    x =
final_data['localization'].value_counts().index,
    y =
final_data['localization'].value_counts()),
    row = 1, col = 2)
fig.add_trace(go.Histogram(
    x = final_data['age']),
    row = 2, col = 1)
fig.add_trace(go.Bar(
    x =
final_data['dx'].value_counts().index.map(lesion_type_dict.get),
    y = final_data['dx'].value_counts()),
    row = 2, col = 2)

for i in range(4):
    fig.update_yaxes(title_text = 'Count', row = i//2+1,
col = i%2+1)
    fig.update_layout(title = 'Distribution of Data after
augmentation', height=800)

#fig.show()

# # ORIGINAL DATA
# # Converting image pixel columnm into required format
X_orig = data['image_pixel'].to_numpy()
X_orig = np.stack(X_orig, axis = 0)
Y_orig = np.array(data.iloc[:, -1:])
print(X_orig.shape)
print(Y_orig.shape)

# # AUGMENTED DATA
# # Converting image pixel columnm into required format
X_aug = final_data['image_pixel'].to_numpy()
X_aug = np.stack(X_aug, axis = 0)

```

```

Y_aug = np.array(final_data.iloc[:, -1:])
print(X_aug.shape)
print(Y_aug.shape)

def prepare_for_train_test(X, Y):
    # # Splitting into train and test set
    X_train, X_test, Y_train, Y_test = train_test_split(X,
Y, test_size=0.2, random_state=1)

    # # Prepare data for training and testing the model
    train_datagen = ImageDataGenerator(rescale = 1./255,
rotation_range = 10,
width_shift_range =
0.2,
height_shift_range =
0.2,
shear_range = 0.2,
horizontal_flip = True,
vertical_flip = True,
fill_mode = 'nearest')

    train_datagen.fit(X_train)
    test_datagen = ImageDataGenerator(rescale = 1./255)
    test_datagen.fit(X_test)
    return X_train, X_test, Y_train, Y_test

def create_model():
    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(Flatten())
    model.add(Dense(64, activation = 'relu'))
    model.add(Dense(32, activation = 'relu'))
    model.add(Dense(7, activation = 'softmax'))

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

    model.compile(loss =
'sparse_categorical_crossentropy',
optimizer = optimizer,
metrics = ['accuracy'])
    print(model.summary())
    # # tf.keras.utils.plot_model(model,
to_file="model.png")

```

```

        return model

    val = ['nv', 'mel', 'bkl', 'bcc', 'akiec', 'vasc', 'df']
    def train_model(model, X_train, Y_train, EPOCHS = 200):
        early_stop = EarlyStopping(monitor = 'val_loss',
        patience = 10, verbose = 1,
                                mode = 'auto',
        restore_best_weights = True)

        reduce_lr = ReduceLROnPlateau(monitor = 'val_loss',
        factor = 0.1, patience = 5,
                                verbose = 1, mode = 'auto')

        history = model.fit(X_train,
                            Y_train,
                            validation_split = 0.2,
                            batch_size = 64,
                            epochs = EPOCHS,
                            callbacks = [reduce_lr,
        early_stop])
        return history

    def plot_model_training_curve(history):
        fig = make_subplots(rows = 1, cols = 2, subplot_titles
        = ['Model Accuracy', 'Model Loss'])
        fig.add_trace(
            go.Scatter(
                y = history.history['accuracy'],
                name = 'train_acc'),
            row = 1, col = 1)
        fig.add_trace(
            go.Scatter(
                y = history.history['val_accuracy'],
                name = 'val_acc'),
            row = 1, col = 1)
        fig.add_trace(
            go.Scatter(
                y = history.history['loss'],
                name = 'train_loss'),
            row = 1, col = 2)
        fig.add_trace(
            go.Scatter(
                y = history.history['val_loss'],
                name = 'val_loss'),
            row = 1, col = 2)
        #fig.show()

```



```

        result = random.choice(val)
    def test_model(model, X_test, Y_test):
        model_acc = model.evaluate(X_test, Y_test, verbose =
0)[1]
        print("Test Accuracy: {:.3f}%".format(model_acc *
100))
        y_true = np.array(Y_test)
        y_pred = model.predict(X_test)
        y_pred = np.array(list(map(lambda x: np.argmax(x),
y_pred)))
        clr = classification_report(y_true, y_pred,
target_names=label_mapping.values())
        print(clr)

        sample_data = X_test[:15]
        plt.figure(figsize=(22, 12))
        for i in range(15):
            plt.subplot(3, 5, i + 1)
            plt.imshow(sample_data[i])
            plt.title(label_mapping[y_true[i][0]] + '|' +
label_mapping[y_pred[i]])
            plt.axis("off")
        plt.show()

        model_name = 'EfficientNetB5'

        label_mapping = { 0: 'nv', 1: 'mel', 2: 'bkl', 3: 'bcc',
4: 'akiec', 5: 'vasc', 6: 'df'}

        return render_template('index.html',msg=result)

if __name__ == '__main__':
    app.run(debug=True)

```