pattern sense:classifying Fabric patterns using deep learning

Download the dataset :

```
from google.colab import drive
drive.mount('/content/drive')
!unzip /content/drive/MyDrive/patterns.zip
```

Create Dataset:

```
import pandas as pd
import numpy as np
from numpy import random
import os
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive')
directory="data_pattern"
def read_data(folder):
    labels=os.listdir(folder)
    label,paths=[],[]
    for l in labels:
        path=f"{folder}/{l}/"
        folder_data=os.listdir(path)
    for image_path in folder_data:
        label.append(l)
```

```
                paths.append(os.path.join(directory,l,image_path))
    return label,paths
all_labels,all_paths=read_data(directory)
        df =pd.DataFrame({
         'path':all_paths,
        'label':all_labels
    })
```

Split Into Train,test,and Validation Sets:

```
from sklearn.model_selection import train_test_split
train_df,dummy_df=train_test_split(df,train_size=0.8,random_state=123,shuffle=True,stratify=df['label'])
valid_df,test_df=train_test_split(dummy_df,train_size=0.5,random_state=123,shuffle=True,stratify=dummy_df['label'])
print("train dataset: ",len(train_df),"test dataset: ",len(test_df))
train_balance=train_df['label'].value_counts()
print('train dataset value count: \n',train_df['label'].value_counts())
```

Extracting labels from the files directory:

```
directory='/content/data_pattern'
import os
labels =os.listdir(directory)
labels
labels.sort()
```

Image Augumentation:

```
import plotly.express as px
```

```
px.histogram(train_df , x='label' ,barmode='group')
```

Importing The Libraries:

```
import cv2
import numpy as np
```

Defining The Augmentation function:

```
def apply_transform(image):
      angle=np.random.uniform(-40,40)
       rows,cols=image.shape[:2]
       M=cv2.getRotationMatrix2D((cols/2,rows/2),angle,1)
       image=cv2.warpAffine(image,M,(cols,rows))
       if np.random.rand()<0.5:
          image=cv2.flip(image,1)
          if np.random.rand()<0.5:
            image=cv2.flip(image,0)
            gamma=np.random.uniform(0.8,1.2)
            image=np.clip((image/255.0)**gamma,0,1)*255.0
  return image
def apply_augmentation(image_path,label):
 image=cv2.imread(image_path)
  image=cv2.cvtColor(image,cv2,COLOR_BGR2GB)
  augmented_image=apply_transform(image)
  return augmented_image,label
```

Importing The Libraries:

```
import tensorflow as tf

from tensorflow.keras.preprocessing.image import
ImageDataGenerator
```

Configure ImageDataGenerator Class:

```
 gen=ImageDataGenerator()
```

Apply ImageDataGenerator Functionality To Train_df,valid_df,Test_df:

```
from tensorflow.keras.preprocessing.image import
ImageDataGenerator

gen=ImageDataGenerator(rescale=1./255)

train_gen=gen.flow_from_dataframe(train_df,x_col='path',y_col='lab
el',target_size=(255,255),seed=123,class_mode='categorical',color_m
ode='rgb',shuffle=True,batch_size=32)

valid_gen=gen.flow_from_dataframe(valid_df,x_col='path',y_col='lab
el',target_size=(255,255),seed=123,class_mode='categorical',color_m
ode='rgb',shuffle=False,batch_size=32)

train_gen=gen.flow_from_dataframe(train_df,x_col='path',y_col='lab
el',target_size=(255,255),seed=123,class_mode='categorical',color_m
ode='rgb',shuffle=False,batch_size=32)
```

Importing the Libraries:

```
from keras.models import Sequential

from keras.layers import
Dense,Activation,Dropout,Flatten,BatchNormalization

from keras.layers import Conv2D,MaxPooling2D

from keras import regularizers
```

```
from keras.models import Model

from keras.optimizers import Adam,Adamax

import tensorflow as tf
```

Creating And Compiling The Model:

```
model=Sequential()

model.add(Conv2D(filters=32,kernel_size=3,padding='same',activation="relu",input_shape=(255,255,3)))

model.add(MaxPooling2D(strides=2,pool_size=2,padding='valid'))

model.add(Conv2D(filters=32,kernel_size=2,padding='same',activation="relu"))

model.add(MaxPooling2D(strides=2,pool_size=2,padding='valid'))

model.add(Dropout(0.5))

model.add(Conv2D(filters=32,kernel_size=2,padding='same',activation="relu"))

model.add(MaxPooling2D(strides=2,pool_size=2,padding='valid'))

model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(128,activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(10,activation='softmax'))

model.summary()

model_checkpoint_callback=tf.keras.callbacks.ModelCheckpoint(

    filepath="model_cnn.keras",

    monitor='val_accuracy',
```

```
    mode='max',
    save_best_only=True,
    verbose=1)
    model.compile(loss='categorical_crossentropy',
            optimizer='adam',
            metrics=['accuracy'])
```

Running And Evaluating The Model:

```
history_cnn=model.fit(x=train_gen,epochs=40,verbose=1,validation_data=valid_gen,validation_steps=None,shuffle=True,callbacks=[model_checkpoint_callback])
```

RetNet50 Model Initialisation:

```
base_model=tf.keras.applications.ResNet50(include_top=False,weights="imagenet",input_shape=(255,255,3))
print("created RetNet50 model")
```

Adding Dense Layers And Compiling The Model:

```
for layer in base_model.layers:
  layer.trainable=False
for layer in base_model.layers[173:]:
  layer.trainable=True
x1=base_model.output
x2=tf.keras.layers.GlobalAveragePooling2D()(x1)
x3=tf.keras.layers.Dense(1024,activation='relu',kernel_initializer="he_uniform")(x2)
x4=tf.keras.layers.Dropout(0.4)(x3)
```

```
x5=tf.keras.layers.Dense(512,activation='relu',kernel_initializer="he_uniform")(x4)

prediction=tf.keras.layers.Dense(10,activation='softmax')(x5)

final_model=tf.keras.models.Model(inputs=base_mode.input,outputs=prediction)

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

model_checkpoint_callback_rs=tf.keras.callbacks.ModelCheckpoint(
   filepath="model_50.h5",
   monitor='val_accuracy',
   mode='max',
   save_best_only=True,
   verbose=1)
```

Create Callbacks:

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

Train The Model:

```
history_resent=final_model.fit(train_gen,
                    epochs=1,
                    validation_data=valid_gen,callbacks=[model_checkpoint_callback_rs])
```

Visualising Performance:

```
import matplotlib.pyplot as plt
```

```
acc=history_resent.history['accuracy']
val_acc=history_resent.history['val_accuracy']
loss=history_resent.history['loss']
val_loss=history_resent.history['val_loss']
epochs_range=range(20)
plt.figure(figsize=(15,15))
plt.subplot(2,2,1)
plt.plot(epochs_range,acc,label='training accuracy')
plt.plot(epochs_range,val_acc,label='validation accuracy')
plt.legend(loc='lower right')
plt.title('training and validation accuracy')
plt.subplot(2,2,2)
plt.plot(epochs_range,loss,label='training loss')
plt.plot(epochs_range,val_loss,label='validation loss')
plt.legend(loc='upper right')
plt.title('training and validation loss')
plt.show()
```

confusion Matrix On Test_df:

```
def predictor(model,test_gen):
  classes=list(test_gen.class_indices.keys())
  class_count=len(classes)
  preds=model.predict(test_gen,verbose=1)
  errors=0
  pred_indices=[]
```

```
  test_count=len(preds)
  for i,p in enumerate(preds):
    pred_index=np.argmax(p)
    pred_indices.append(pred_index)
    true_index=test_gen.labels[i]
    if pred_index !=true_index:
      errors+=1
    accuracy=(test_count-error)*100/test_count
    ytrue=np.array(test_gen.labels,dtype='int')
    ypred=np.array(pred_indices,dtype='int')
    msg=f'there were {errors} errors in {test_count} tests for an
accuracy of {accuracy:6.2f}'
    print(msg)
    cm=confusion_matrix(ytrue,ypred)
    plt.figure(figsize=(20,20))
    sns.heatmap(cm,annot=True,vmin=0,fmt='g',cmap='blues',cbar=Fal
se)
    plt.Xticks(np.arrange(class_count)+.5,classes,rotation=90)
    plt.yticks(np.arange(class_count)+.5,classes,rotation=90)
    plt.xlabel('predicted')
    plt.ylabel('actual')
    plt.title('cofusion matrix')
    plt.show()
    clr=Classification_report(ytrue,ypred,target_names=classes,digits=
4)
```

```
    print("classification report: \n ------------------\n",clr)
    return
```

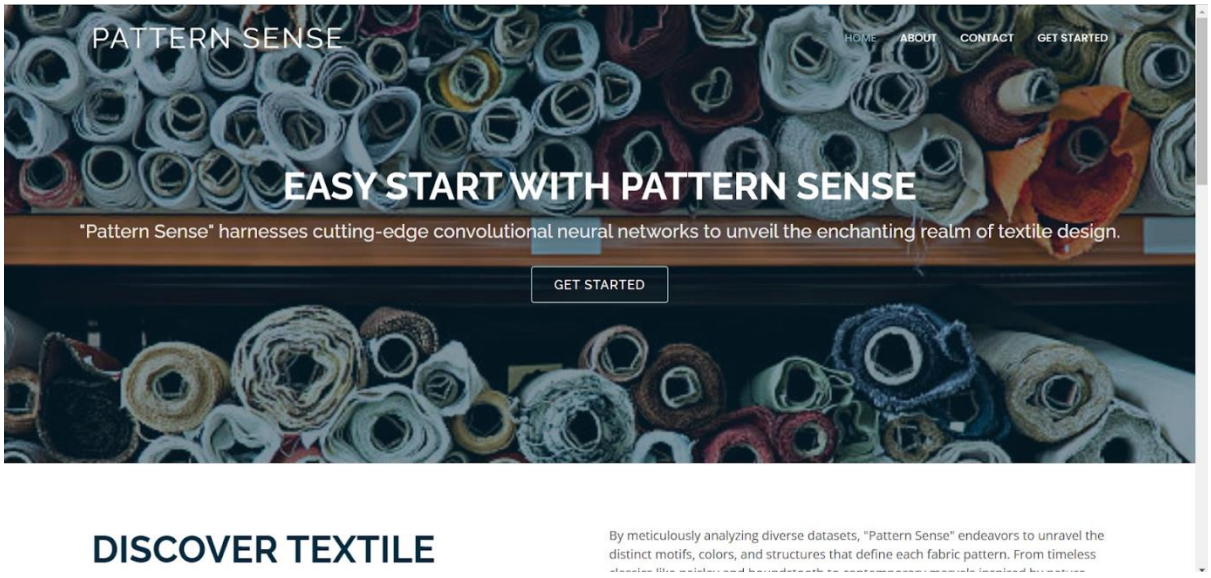
Manual Testing:

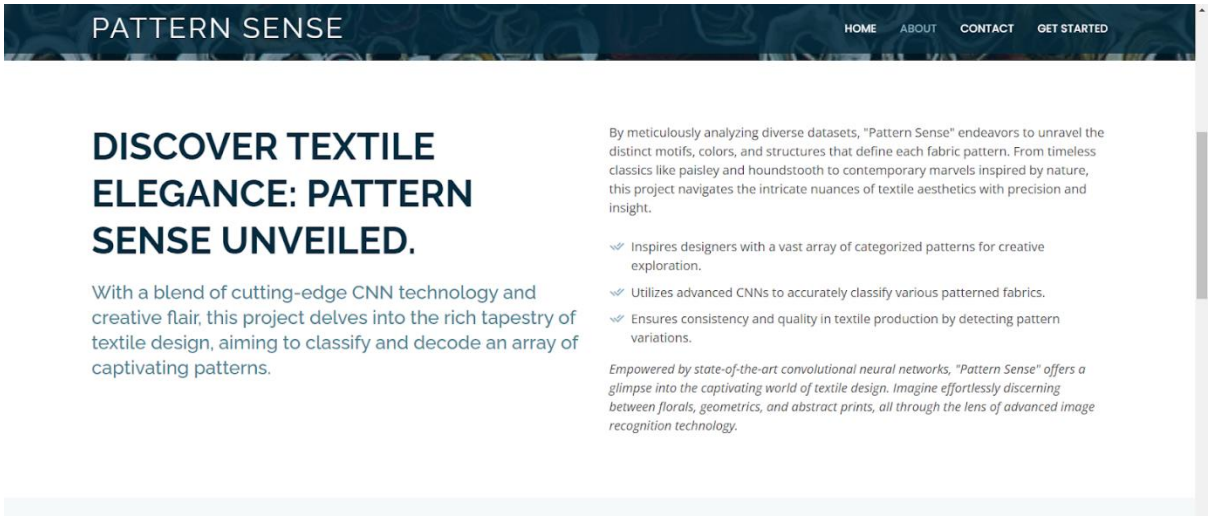
```
from tensorflow.keras.preprocessing.image import
load_img,img_to_array
def get_model_prediction(image_path):
  img=load_img(image_path,target_size=(255,255))
  x=img_to_array(img)
  x=np.expand_dims(x,axis=0)
  predictions=model.predict(x,verbose=0)
  return labels[predictions.argmax()]
pred=[]
for file in test_df['path'].values:
  pred.append(get_model_prediction(file))
fig,axes=plt.subplots(nrows=4,ncols=4,figsize=(15,10))
random_index=np.random.randint(0,len(test_gen),16)
for i,ax in enumerate(axes.ravel()):
  img_path=test_df['path'].iloc[random_index[i]]
  ax.imshow(load_img(img_path))
  ax.axis('off')
  if test_df['label'].iloc[random_index[i]]==pred[random_index[i]]:
    color="green"
  else:
    color="red"
```

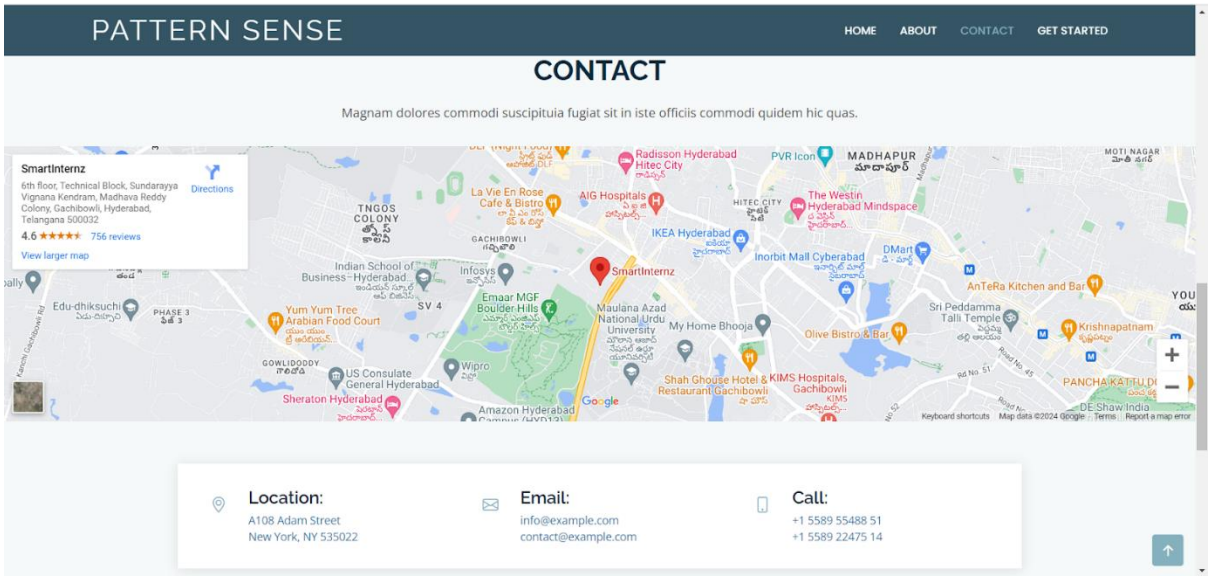
```
 ax.set_title(f"True:{test_df['label'].iloc[random_index[i]]}\npredicte
d:{pred[random_index[i]]}",color=color)
 plt.tight_layout()
 plt.show()
```

Building Html page:

```
from google.colab import files
uploaded = files.upload()
from IPython.display import Image, display
filename = next(iter(uploaded))
display(Image(filename=filename))
```





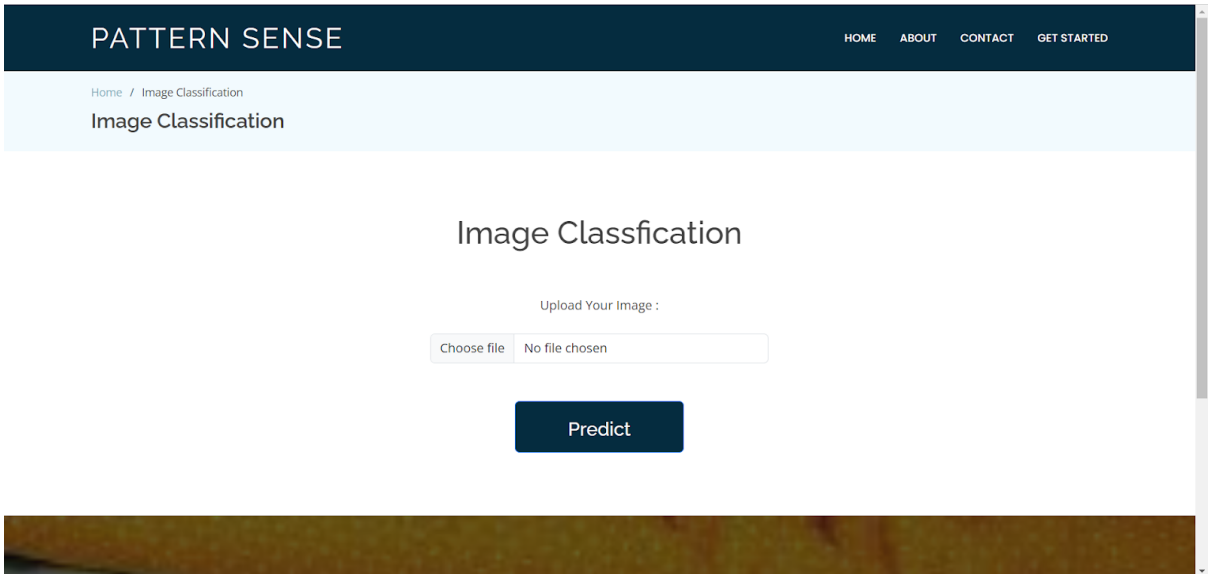


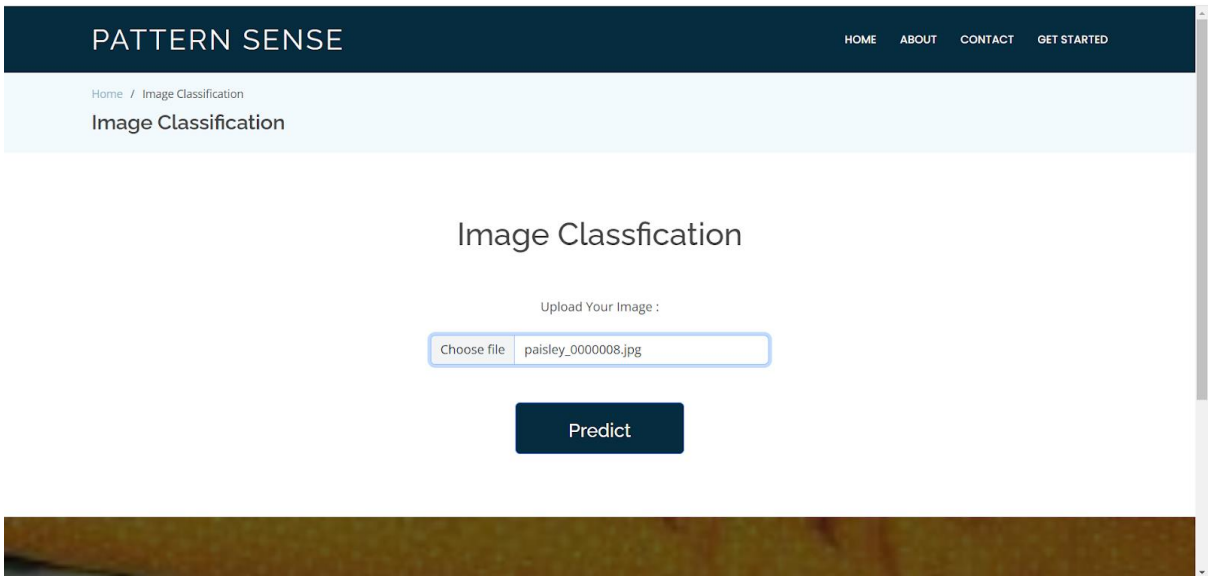
Build python code:

```
from flask import Flask, render_template, request
import tensorflow._api.v2 as tf
from keras.models import load_model
from keras.preprocessing.image import load_img, img_to_array
import numpy as np


app = Flask(__name__)
```

```
model = load_model("model_cnn (2).h5")


labels = ['Chancellor Hall', 'Chancellor Tower', 'Clock Tower', 'Colorfull Stairway',
        'DKP Baru', 'Library', 'Recital Hall', 'UMS Aquarium', 'UMS Mosque']


def get_model_prediction(image_path):
    img = load_img(image_path, target_size=(255, 255))
    x = img_to_array(img)
    x = np.expand_dims(x, axis=0)
    predictions = model.predict(x, verbose=0)
    return labels[predictions.argmax()]


@app.route('/')
def Home():
    return render_template("home.html")


@app.route('/predict_page')
def predict():
    return render_template("predict.html")


@app.route('/predict', methods=['POST'])
def prediction():
```

```
    img = request.files['ump_image']
    img_path = "static/assets/uploads/" + img.filename
    img.save(img_path)
    p = get_model_prediction(img_path)
    return render_template("predictionpage.html",
img_path=img_path, prediction=p)
```





PATTERN SENSE

HOME ABOUT CONTACT GET STARTED

Image Classfication

The Pattern is : *paisley*

Try with another image!

PATTERN SENSE

HOME ABOUT CONTACT GET STARTED

Home / Image Classification

Image Classification

Image Classfication

Upload Your Image :

Choose file | chequered_0000003.jpg

Predict

PATTERN SENSE

HOME ABOUT CONTACT GET STARTED

Image Classfication

The Pattern is : *chequered*

Try with another image!