Objective

The mini-project aims to develop a **Convolutional Neural Network (CNN)** using TensorFlow for **skin cancer classification**. The goal is to create a model that distinguishes between benign and malignant skin lesions, aiding in early detection and diagnosis through automated image analysis.

Dataset Used and Description

The dataset comprises labeled images of skin lesions, sourced from Kaggle's **Skin Cancer MNIST: HAM10000** dataset. It contains over 10,000 high-quality dermatoscopic images belonging to seven classes, covering common pigmented skin lesions. Each image is pre-processed and categorized based on clinical diagnosis, enabling reliable model training and evaluation.

Data Preprocessing

Data preprocessing includes resizing images to uniform dimensions (e.g., 128x128 pixels), normalization to scale pixel values between 0 and 1, and one-hot encoding of labels. The dataset is also split into training, validation, and testing sets, ensuring balanced representation and avoiding data leakage. Augmentation techniques like rotation and flipping are applied to reduce overfitting.

Model Architecture

The CNN architecture consists of **convolutional layers** for feature extraction, followed by **max-pooling layers** for downsampling. It includes **dropout layers** to prevent overfitting and **dense layers** for classification. The final layer uses a softmax activation function for multi-class classification. TensorFlow and Keras libraries are employed to design and compile the model.

Training

The model is trained using the **Adam optimizer** and **categorical crossentropy loss function** for multi-class classification. Training is performed over multiple epochs with a batch size of 32. Validation accuracy and loss are monitored during training to fine-tune hyperparameters and prevent overfitting.

Evaluation

The model is evaluated on a separate test set using metrics like **accuracy, precision, recall, and F1-score**. Confusion matrices and classification reports are generated to assess performance. Results demonstrate the model's capability to classify skin lesions accurately, with high sensitivity and specificity for malignant cases.

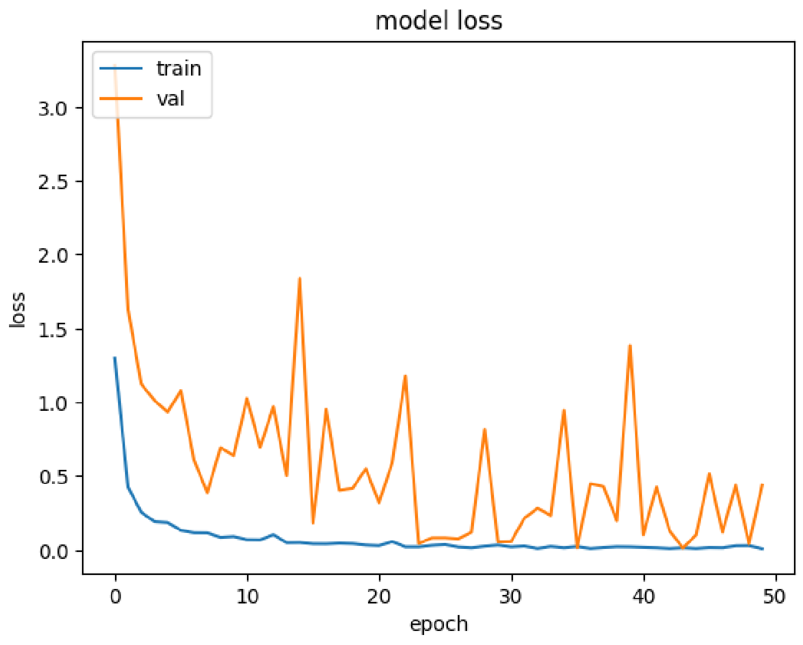
SAMPLE CODE:

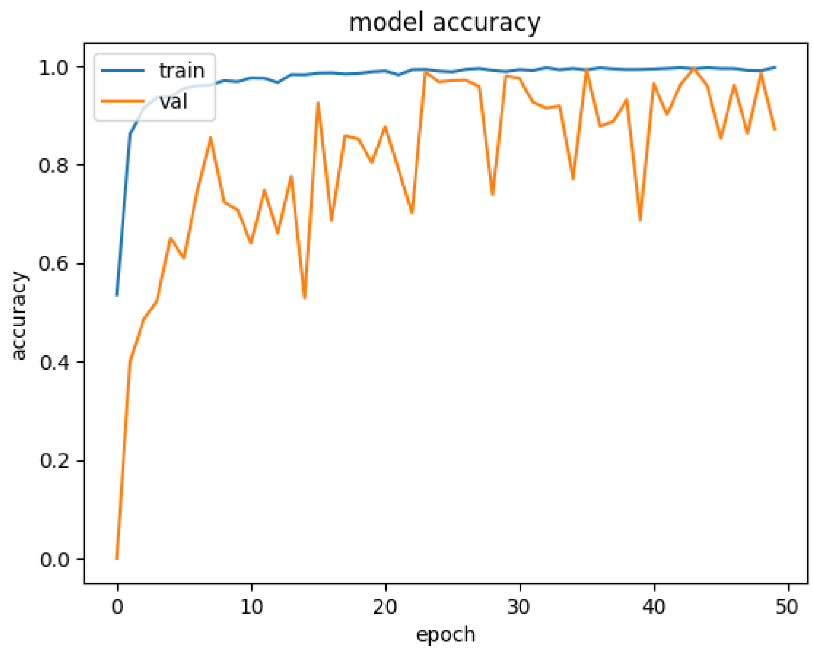
```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
!unzip hmnist_28_28_RGB.csv.zip
path='/kaggle/input/skin-cancer-mnist-ham10000/hmnist_28_28_RGB.csv'
df=pd.read_csv(path)
df.tail()
fractions=np.array([0.8,0.2])
df=df.sample(frac=1)
train_set, test_set = np.array_split(
    df, (fractions[:-1].cumsum() * len(df)).astype(int))
linkcode
print(len(train_set))
classes={0:('akiec', 'actinic keratoses and intraepithelial carcinomae'),
         1:('bcc' , 'basal cell carcinoma'),
         2:('bkl', 'benign keratosis-like lesions'),
         3:('df', 'dermatofibroma'),
         4:('nv', ' melanocytic nevi'),
         5:('vasc', ' pyogenic granulomas and hemorrhage'),
         6:('mel', 'melanoma'),}
y_train=train_set['label']
x_train=train_set.drop(columns=['label'])
y_test=test_set['label']
x_test=test_set.drop(columns=['label'])

columns=list(x_train)
```

```
import seaborn as sns

sns.countplot(train_set['label'])
linkcode
import torch
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
from imblearn.over_sampling import RandomOverSampler
oversample = RandomOverSampler()
x_train,y_train  = oversample.fit_resample(x_train,y_train)
In [16]:
linkcode
sns.countplot(y_train)
import matplotlib.pyplot as plt
import random
num=random.randint(0,8000)
x_train=np.array(x_train, dtype=np.uint8).reshape(-1,28,28,3)

plt.imshow(x_train[num].reshape(28,28,3))
plt.title("Random image from training data")
plt.show()
num=random.randint(0,8000)
plt.imshow(x_train[num].reshape(28,28,3))
plt.title("Random image from training data")
plt.show()

num=random.randint(0,8000)
plt.imshow(x_train[num].reshape(28,28,3))
plt.title("Random image from training data")
plt.show()
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, Dense, MaxPool2D
import tensorflow as tf
#https://keras.io/api/models/sequential/
#https://keras.io/api/layers/core_layers/dense/
#https://keras.io/api/layers/merging_layers/add/
%time

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()
from datetime import datetime
start_time = datetime.now()

history = model.fit(x_train,
                    y_train,
                    validation_split=0.2,
                    batch_size = 128,
                    epochs = 50,
                    shuffle=True,
                    callbacks=[callback])

end_time = datetime.now()
print('Duration: {}'.format(end_time - start_time))
import pickle
```

```
training_history = {
    'loss': history.history['loss'],
    'accuracy': history.history['accuracy'],
    'val_loss': history.history['val_loss'],
    'val_accuracy': history.history['val_accuracy']
}
with open('training_history.pkl', 'wb') as file:
    pickle.dump(training_history, file)
model.save('trained_model.h5')


print("Training history and model saved successfully.")
#plot of accuracy vs epoch
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
import requests
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
from io import BytesIO
'''image_url =
"https://m4b6f3p8.rocketcdn.me/app/uploads/2021/04/basalCellCarcinomaBCC
_6163_lg.jpg"
response = requests.get(image_url)
image = Image.open(BytesIO(response.content))'''
image =
PIL.Image.open('/kaggle/input/testingimage/basalCellCarcinomaBCC_6163_lg.
jpg')
image = image.resize((28, 28))
img = np.array(image)
plt.imshow(img)
plt.axis('off')
plt.show()
img = x_test[1]
img = np.array(image).reshape(-1, 28, 28, 3)
result = model.predict(img)
print(result[0])
result = result.tolist()
max_prob = max(result[0])
class_ind = result[0].index(max_prob)
print(classes[class_ind])
```

OUTPUT:





RESULT:

Thus the Skin Cancer Detection program using CNN was executed successfully