Om Gholap omsagargholap@gmail.com

Title: Cancer Cell Detection by Om Gholap

Problem :
Cancer cell detection and preparing the model based on the dataset

About the Dataset:
The dataset consists of images of seven classes of skin cancer: Melanocytic nevi (nv), Melanoma (mel), Benign keratosis-like lesions (bkl), Basal cell carcinoma (bcc), Actinic keratoses (akiec), Vascular lesions (vas), and Dermatofibroma (df).

Approach:

1. Importing the necessary libraries:
I started by importing the required libraries such as pandas, matplotlib, numpy, os, glob, PIL, seaborn, and keras. These libraries provide various functions and tools that are essential for data analysis, visualization, and building and training the model.

2. Loading the dataset:
I loaded the dataset into a DataFrame called "skin_df" from the "HAM10000_metadata.csv" file. This file contained valuable information about the skin cancer lesions, which I used for analysis and model training.

3. Data exploration and visualization:
To understand the dataset better, I performed data exploration and visualization. I created bar plots to visualize the distribution of different cell types, sex, and localization. Additionally, I plotted a distribution of age to gain insights into the age distribution among the samples.

4. Addressing data imbalance:
I noticed that the dataset was imbalanced, with a higher count of one cell type nv which has 6500 values compared to others.. To address this issue, I balanced the dataset by resampling each class to have an equal number of samples, 500 samples. This ensured that the model doesn't favor any specific cell type during training.

5. Preprocessing and image loading:
To prepare the images for training, I performed preprocessing and image loading. I used the glob function to obtain the paths of the images and added these paths along with their corresponding labels to the balanced DataFrame. Then, I loaded the images using the PIL library, resized them to a specified size, and stored them as numpy arrays.

6. Data preparation:
Before training the model, I prepared the data by scaling the image pixel values between 0 and 1. This step ensures that the data is in an appropriate range for the model to learn effectively. I also converted the labels into categorical form using one-hot encoding. Finally, I split the dataset into training and testing sets to evaluate the model's performance.

Om Gholap omsagargholap@gmail.com

7. Model architecture:
For building the model, I used the Keras library. I created a sequential model consisting of convolutional layers, max pooling layers, dropout layers, and a flatten layer. These layers perform operations on the image data to extract features and learn patterns. The final dense layer with softmax activation was used for multi-class classification. I printed the model summary to understand its architecture.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 30, 30, 256)       7168

 max_pooling2d_6 (MaxPooling  (None, 15, 15, 256)      0
 2D)

 dropout_6 (Dropout)         (None, 15, 15, 256)       0

 conv2d_7 (Conv2D)           (None, 13, 13, 128)       295040

 max_pooling2d_7 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 dropout_7 (Dropout)         (None, 6, 6, 128)         0

 conv2d_8 (Conv2D)           (None, 4, 4, 64)          73792

 max_pooling2d_8 (MaxPooling  (None, 2, 2, 64)         0
 2D)

 dropout_8 (Dropout)         (None, 2, 2, 64)          0

 flatten_2 (Flatten)         (None, 256)               0

 dense_4 (Dense)             (None, 32)                8224

 dense_5 (Dense)             (None, 7)                 231

=================================================================
Total params: 384,455
Trainable params: 384,455
Non-trainable params: 0
_________________________________________________________________
```

8. Model compilation and training:
I compiled the model by specifying the loss function, optimizer, and evaluation metric. I chose categorical cross-entropy as the loss function, the Adam optimizer, and accuracy as the evaluation metric. Then, I trained the model on the training set for a specified number of epochs and with a defined batch size.

Om Gholap omsagargholap@gmail.com

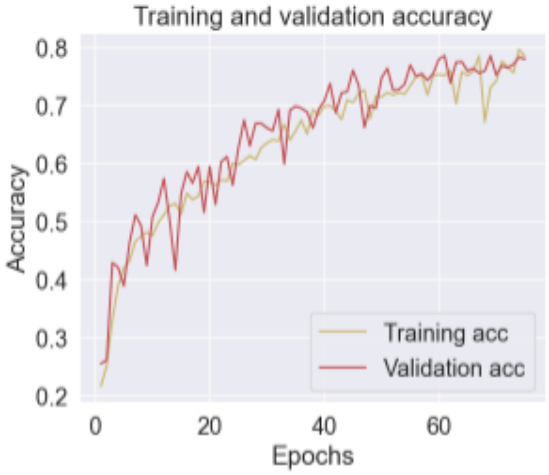
```
batch_size = 32
epochs = 75

history = model.fit(
    x_train, y_train,
    epochs=epochs,
    batch_size = batch_size,
    validation_data=(x_test, y_test),
    verbose=2)
```

9. Model evaluation:

After training the model, I evaluated its performance by plotting the training and validation loss and accuracy. The validation accuracy provided an indication of how well the model could classify unseen images. I observed a validation accuracy of approximately 77%. Increasing the number of epochs and changing the batch may increase the accuracy score more, but may result in the model overfitting to the dataset.

```
Epoch 75/75
83/83 - 58s - loss: 0.5919 - acc: 0.7844 - val_loss: 0.6826 - val_acc: 0.7794 - 58s/epoch - 703ms/step
```



10. Confusion matrix:

Om Gholap omsagargholap@gmail.com

To gain further insights into the model's performance, I used the test set to generate a confusion matrix. This matrix helped visualize the classification results and identify the number of correct and incorrect predictions for each class. I also plotted the fraction of incorrect predictions for each class, highlighting the more challenging classes for the model.

	nv	mel	bkl	bcc	akiec	vas	df
nv	108	10	3	5	1	0	0
mel	23	87	3	11	2	1	2
bkl	19	7	79	3	14	0	0
bcc	3	3	0	119	0	0	0
akiec	11	4	16	0	81	7	0
vas	5	4	17	5	22	63	1
df	0	0	0	0	0	0	136

Potential improvements:
- We can experiment with different model architectures, such as adding more layers or changing the number of filters in the convolutional layers, to improve the model's performance.
- Exploring different optimizers, activation functions, and hyperparameter tuning techniques may also help optimize the model.
- Transfer learning is another avenue to explore, where we can leverage pre-trained models on large datasets and fine-tune them for our specific task.
- Implementing data augmentation techniques using the TensorFlow library can generate additional training data by applying transformations like rotations, flips, and zooms. This can help improve the model's ability to generalize to different variations of the images.

Conclusion:
In conclusion, further research and exploration of alternative techniques are necessary to enhance the performance of the model. By continuously improving and experimenting with different approaches, we can strive to develop a more robust and accurate cancer cell detection model for improved clinical applications.