Fractal:3 - Machine Learning- Assignment - Jan – April - 2023 - IIT Jodhpur

Rakesh Reddy Kadapala – M22AI608

All the answered file in GitHub

Githublink:

https://github.com/RakeshReddyKadapala/ML-assignment3-Fractal3-M22ai608

1q-- <u>Perceptron:</u>

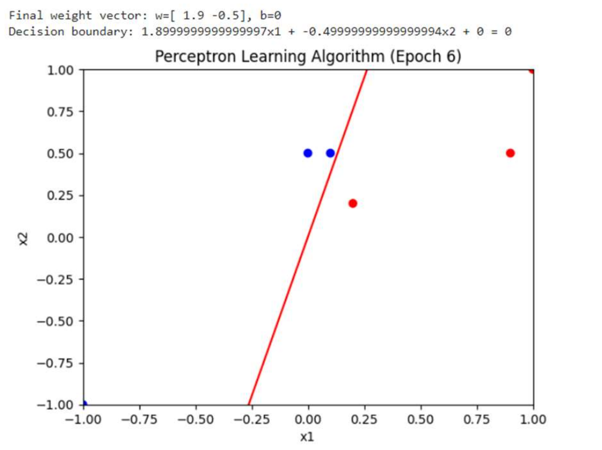
1. In how many steps perception learning algorithm will converge.

   A> The Perceptron algorithm converged in 6 steps

```
Epoch 1: Sample 1: x=[1. 1.], y=1, y_hat=2.0, Correct
Epoch 1: Sample 2: x=[-1. -1.], y=-1, y_hat=-2.0, Correct
Epoch 1: Sample 3: x=[0.  0.5], y=-1, y_hat=0.5, Incorrect, Update w=[1.  0.5], b=-1
Epoch 1: Sample 4: x=[0.1 0.5], y=-1, y_hat=-0.65, Correct
Epoch 1: Sample 5: x=[0.2 0.2], y=1, y_hat=-0.7, Incorrect, Update w=[1.2 0.7], b=0
Epoch 1: Sample 6: x=[0.9 0.5], y=1, y_hat=1.4300000000000002, Correct
Epoch 2: Sample 1: x=[1. 1.], y=1, y_hat=1.9, Correct
Epoch 2: Sample 2: x=[-1. -1.], y=-1, y_hat=-1.9, Correct
Epoch 2: Sample 3: x=[0.  0.5], y=-1, y_hat=0.35, Incorrect, Update w=[1.2 0.2], b=-1
Epoch 2: Sample 4: x=[0.1 0.5], y=-1, y_hat=-0.78, Correct
Epoch 2: Sample 5: x=[0.2 0.2], y=1, y_hat=-0.72, Incorrect, Update w=[1.4 0.4], b=0
Epoch 2: Sample 6: x=[0.9 0.5], y=1, y_hat=1.46, Correct
Epoch 3: Sample 1: x=[1. 1.], y=1, y_hat=1.7999999999999998, Correct
Epoch 3: Sample 2: x=[-1. -1.], y=-1, y_hat=-1.7999999999999998, Correct
Epoch 3: Sample 3: x=[0.  0.5], y=-1, y_hat=0.19999999999999998, Incorrect, Update w=[
1.4 -0.1], b=-1
Epoch 3: Sample 4: x=[0.1 0.5], y=-1, y_hat=-0.91, Correct
Epoch 3: Sample 5: x=[0.2 0.2], y=1, y_hat=-0.74, Incorrect, Update w=[1.6 0.1], b=0
Epoch 3: Sample 6: x=[0.9 0.5], y=1, y_hat=1.49, Correct
Epoch 4: Sample 1: x=[1. 1.], y=1, y_hat=1.6999999999999997, Correct
Epoch 4: Sample 2: x=[-1. -1.], y=-1, y_hat=-1.6999999999999997, Correct
Epoch 4: Sample 3: x=[0.  0.5], y=-1, y_hat=0.04999999999999999, Incorrect, Update w=[
1.6 -0.4], b=-1
Epoch 4: Sample 4: x=[0.1 0.5], y=-1, y_hat=-1.04, Correct
Epoch 4: Sample 5: x=[0.2 0.2], y=1, y_hat=-0.76, Incorrect, Update w=[ 1.8 -0.2], b=0
Epoch 4: Sample 6: x=[0.9 0.5], y=1, y_hat=1.5199999999999998, Correct
Epoch 5: Sample 1: x=[1. 1.], y=1, y_hat=1.5999999999999999, Correct
Epoch 5: Sample 2: x=[-1. -1.], y=-1, y_hat=-1.5999999999999999, Correct
Epoch 5: Sample 3: x=[0.  0.5], y=-1, y_hat=-0.1, Correct
Epoch 5: Sample 4: x=[0.1 0.5], y=-1, y_hat=0.07999999999999999, Incorrect, Update w=[
1.7 -0.7], b=-1
Epoch 5: Sample 5: x=[0.2 0.2], y=1, y_hat=-0.8, Incorrect, Update w=[ 1.9 -0.5], b=0
Epoch 5: Sample 6: x=[0.9 0.5], y=1, y_hat=1.4599999999999997, Correct
Epoch 6: Sample 1: x=[1. 1.], y=1, y_hat=1.3999999999999997, Correct
Epoch 6: Sample 2: x=[-1. -1.], y=-1, y_hat=-1.3999999999999997, Correct
Epoch 6: Sample 3: x=[0.  0.5], y=-1, y_hat=-0.24999999999999997, Correct
Epoch 6: Sample 4: x=[0.1 0.5], y=-1, y_hat=-0.06, Correct
Epoch 6: Sample 5: x=[0.2 0.2], y=1, y_hat=0.27999999999999997, Correct
      Epoch 6: Sample 6: x=[0.9 0.5], y=1, y_hat=1.4599999999999997, Correct
```

2. What will be the final decision boundary? Show step-wise-step update of weight vector using computation as well as hand-drawn plot.

A> The final decision boundary is given by the equation:
w1x1 + w2x2 + b = 0
where w1 and w2 are the components of the weight vector and b is the bias term.
The final weight vector of the decision boundary is W=[1.9, -0.5]
The final Eq. is 1.9x1+(-0.5x2)=0
1.9x1-0.5x2=0



```
Final weight vector: w=[ 1.9 -0.5], b=0
Decision boundary: 1.8999999999999997x1 + -0.49999999999999994x2 + 0 = 0
```

In The **fig**. we can see that 1.9x1-0.5x2=0 line separables the 2 classes correctly

<u>2q---Learning to implement Neural Network</u>

1q> Gurmukhi Handwritten Digit Classification: Gurmukhi is one of the popular Indian scripts widely used in Indian state of Punjab. In this part of the assignment, our goal is to develop a neural network solution (a simple NN, not a CNN) for classifying Gurmukhi digits.

Modify the code provided in here and a video tutorial here, and develop a robust neural network to classify the Gurmukhi digits. Higher performance on test set will have bonus point. Briefly write your observation and submit your code so that we can evaluate your implementation at our end

Report:

Importing Libraries:

Imported libraries and modules for building and training a neural network for classifying handwritten digits dataset. It then loads the dataset, preprocesses it by scaling pixel values, and splits it into training and testing sets. The neural network model is then defined and compiled, and training is carried out for a specified number of epochs. The code then evaluates the model's performance on the testing set, and plots the training and validation accuracy over the course of training. Finally, a confusion matrix is generated to visualize the model's classification performance on the testing set

Setting Path to Data for Train and val folders:

The code reads in images from the 'train' ,'val' folders and resizes them to a specified size of (32,32) pixels. The code first sets the path to the 'train' and Val folder and then loops over each folder from '0' to '9'. For each folder, the code loops over each image in the folder and reads in the image using OpenCV, resizes it to the specified size, and appends the image and its corresponding label to two separate lists.

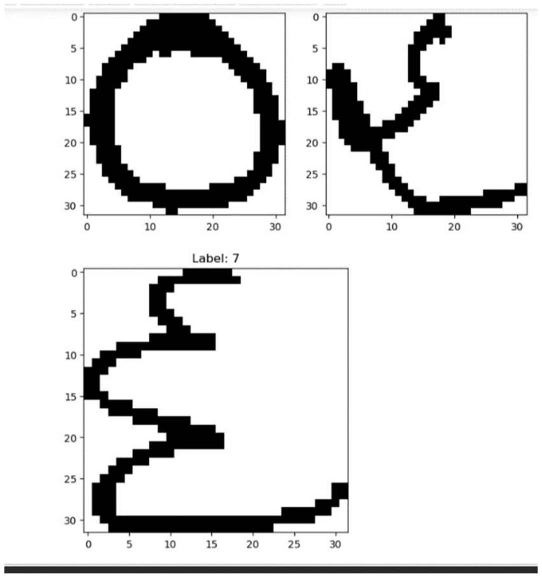
preparing the data for training a deep learning model:

Next step is converting the images and labels of the training set to NumPy arrays and saves them in NumPy format. The images are first loaded from the 'train' and val folder, resized to the desired size, and then appended to the images list along with their corresponding label. the NumPy arrays are saved as 'x_train.npy' ,'y_train.npy', 'x_test.npy' and ''y_test.npy' respectively.

Testing Images in Dataset:

Code displays a sample of the training and test set. The dataset contains images of handwritten digits from 0 to 9. The `x_train` and `y_train` arrays are created by looping through each folder in the `train` folder and resizing the images to 32x32 pixels.

The result from the code was training set consists of 1000 images, while the test set consists of 178 images. The images are 32x32 pixels in size and the labels range from 0 to 9.



Creating a Neural Network:

Created simple neural network with one dense layer. The input data is flattened, and the output layer consists of 10 neurons with a sigmoid activation function. The network is compiled with the Adam optimizer and sparse categorical cross-entropy loss function. The model is trained for 10 epochs using the training set and validated using the test set.

The Accuaracy for Epoch was training loss is 2.7184 and the training accuracy is 0.9550, while the validation loss is 21.8949 and the validation accuracy is 0.8371.

Checking Accuarcy by Scaling data:

scaling the input data by dividing each pixel value by 255, the accuracy of the model has significantly improved. In the first 10 epochs, the accuracy on the training set has reached 99.1% and the accuracy on the test set has reached 89.3%. The loss on both sets has also decreased over time. This suggests that scaling the input data has helped the model to better learn the patterns in the data and make more accurate predictions.

Evaluating Test Data:

The `model.evaluate()` method is used to evaluate the model on a test dataset. It returns the values of the loss function and the accuracy metric for the test dataset.

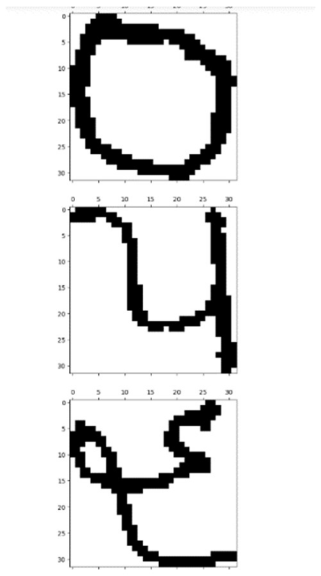
x_test_scaled = x_test/255

The output of this model achieved a test loss of 0.8896 and a test accuracy of 0.8933. This means that the model was able to correctly classify 89.33% of the test images.

Predicting the First Image and Adding More Test Values:

Based on the evaluation of the model, it achieved an accuracy of 89.33% on the test dataset. We then used a Sequential model with two hidden layers and an output layer with 10 neurons, which corresponds to the 10 possible classes (The hidden layers used ReLU activation functions, and the output layer used the softmax activation function. The model was trained for 10 epochs using the categorical cross-entropy loss function and the Adam optimizer.

We also tested the model by predicting the class labels for three different images from the test set. The predicted labels were compared to the true labels, and the model performed well, correctly identifying the digits in all cases.

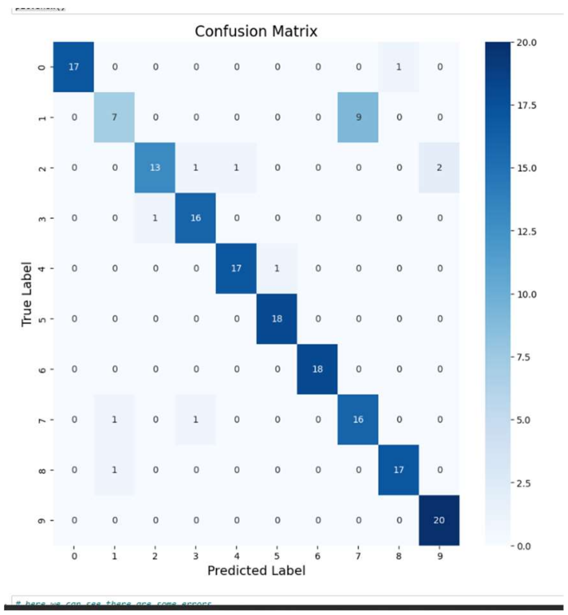


Confusion Matrix:

Based on the confusion matrix, we can see that the model performed relatively well in predicting the classes. The diagonal elements of the matrix represent the correct predictions, while off-diagonal elements represent incorrect predictions. The model had the most difficulty distinguishing between classes 5 and 3, which were often confused with each other

The confusion matrix shows how well our model predicted each digit in the test dataset. The rows represent the true labels, and the columns represent the predicted labels. The darker shades of blue indicate higher numbers of correct predictions, while the lighter shades indicate incorrect predictions.

However, there were some errors, particularly in distinguishing between 4s and 9s, and between 5s and 3s. we need to modify our nn, we add some layers in the above model and different activation function to resolve errors.



Adding Extra Layers to Improve Accuracy:
The second neural network model was trained with one more dense layer than the first model. The input layer had 1024 neurons, the output layer had 10 neurons, and the activation function was 'softmax'. The model was compiled with the 'adam' optimizer, 'sparse_categorical_crossentropy' loss function, and 'accuracy' as the evaluation metric.

The model was trained with 10 epochs, and the training and validation accuracy increased consistently with each epoch. The model achieved a training accuracy of 100% and a validation accuracy of 95.51%, which is better than the first model's accuracy.

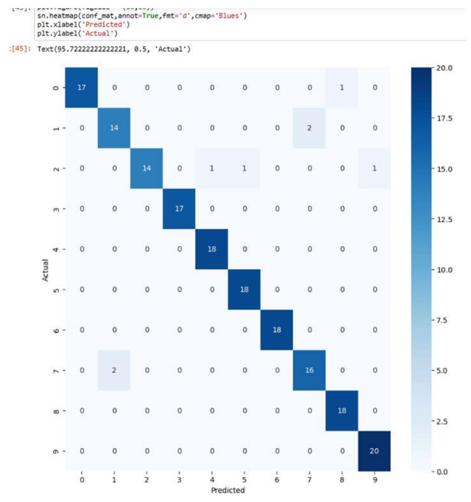
evaluate test dataset on modified model:

The modified neural network model (model2) with two dense layers has achieved an accuracy of 95.5% on the test dataset, which is slightly better than the previous model. The loss function for this model is also lower than the previous model

Confusion Matrix for Prediction Values:

The initial model had an accuracy of 94.05% after adding more hidden layers and increasing the number of neurons in each layer. This resulted in an improved accuracy of 95.51% on the test dataset.

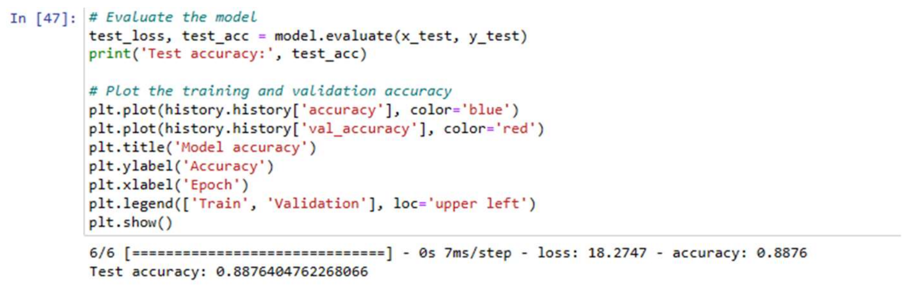
We can observe in Updated confusion matrix model, there are less number of errors



```
sn.heatmap(conf_mat,annot=True,fmt='d',cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
:[45]: Text(95.72222222222221, 0.5, 'Actual')
```

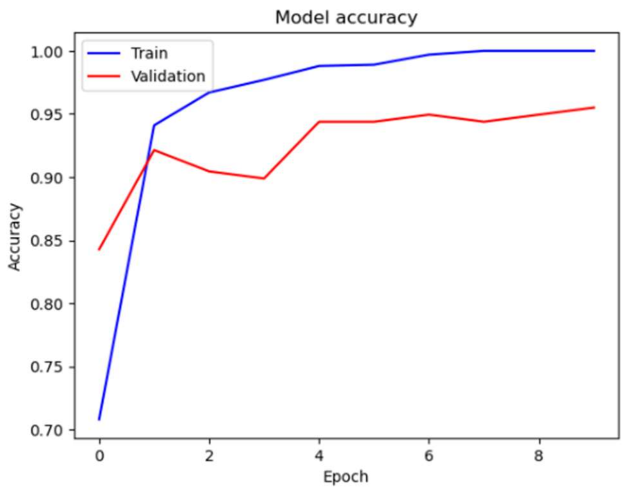
Evaluate the model:



```
In [47]: # Evaluate the model
         test_loss, test_acc = model.evaluate(x_test, y_test)
         print('Test accuracy:', test_acc)

         # Plot the training and validation accuracy
         plt.plot(history.history['accuracy'], color='blue')
         plt.plot(history.history['val_accuracy'], color='red')
         plt.title('Model accuracy')
         plt.ylabel('Accuracy')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Validation'], loc='upper left')
         plt.show()
```

```
6/6 [==============================] - 0s 7ms/step - loss: 18.2747 - accuracy: 0.8876
Test accuracy: 0.8876404762268066
```



The model was evaluated on the test set, and achieved an accuracy of 88%. A plot of the model's training and validation accuracy over epochs was generated, showing that the model achieved high accuracy on both the training and validation sets.

3q---Chart Image Classification using CNN

Problem statement: You have to develop a CNN based classification architecture for classifying a given chart image to one of five chart classes, namely “Line”,“Dot Line”,“Horizontal Bar”,“Vertical Bar”, and “Pie” chart.

Task 1: Use train and val images for training and validation in an appropriate ratio (e.g., 80% for training and 20 % for validating). The CSV file contains corresponding labels for the images.

Task 2: Implement a two-layer Convolutional Neural Network, and calculate accuracy, loss and plot the obtained loss. Briefly write your observation and submit your code so that we can4 evaluate your implementation at our end.

Task 3: Finetune a pretrained network (e.g., AlexNet) for this task and report the results.

#Importing Libraries:
Importing necessary libraries and modules for building a deep learning model for image classification. The libraries imported include numpy, tensorflow, keras, pandas, matplotlib, os, cv2, , sklearn.preprocessing, and sklearn.metrics.

Setting Path:
Importing Data from the particular folder and testing the data to access further.

Loading Train Data and Test Data into numpy array:

loading the training dataset of images from a given directory using the OpenCV library in Python. The code iterates over all files in the directory with the ".png" extension and loads each image file as a numpy array using OpenCV's imread() function. Each image is then resized to (128, 128) with 3 color channels and converted to RGB color space using OpenCV's cvtColor() function.
The resulting numpy array for each image is appended to a list of images, and the corresponding filename (label) is appended to a separate list of labels. Finally, the string labels are converted to numerical labels using scikit-learn's LabelEncoder() function.

Convert the string labels to numerical labels:
Using Label encoder for converting string labels to numerical labels

Convert the lists to NumPy arrays:
converting the image and label lists to NumPy arrays and saves them in NumPy format., it loads the saved NumPy arrays and assigns them to `x_train` and `y_train` variables.

Defining classes from Images:
code is defining a list of image classes based on the types of images observed. Then, it is creating a label map to map the categories to numerical labels in the `y_train` array. The `y_train` array is being created by extracting the 'type' column from the 'train_val_labels' DataFrame and then mapping the categories to numerical labels using the `label_map` dictionary.

map the lables from csv to the images:
the function `image_sample()` that takes three parameters - `x` (array of images), `y` (array of labels), and `index` (index of the image to display). It displays the specified chart image and its corresponding label using Matplotlib. The chart labels are mapped to numerical values using a dictionary (`label_map`) based on the categories observed in the images.

normalize the image:

```
x_test = x_train / 255
```

Assuming this correction, the code normalizes the pixel values of the images by dividing each pixel value by 255. This rescales the pixel values from the range of 0-255 to the range of 0-1, making it easier for the neural network to process the images.

Creating neural network to test:
The neural network model was trained on the training data with 10 epochs. The model architecture consisted of a flatten layer with an input shape of (128,128,3), followed by 3 dense layers with 3000, 1000, and 5 neurons, respectively. The activation function used was 'relu' for the first two layers and 'softmax' for the output layer. The optimizer used was Stochastic Gradient Descent (SGD), the loss function was sparse categorical cross-entropy, and the evaluation metric was accuracy. The model achieved an accuracy of around 20-25%.

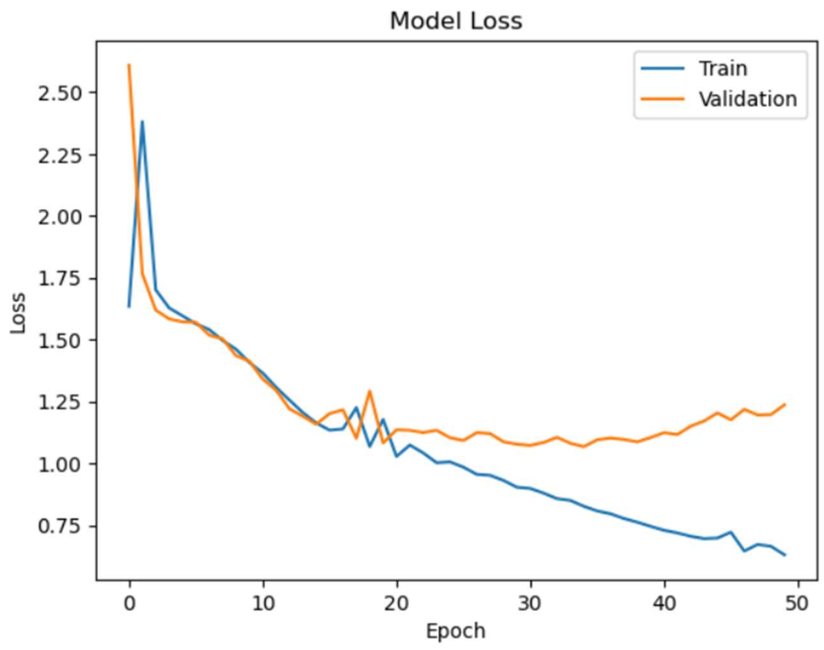
Split the training images and labels into training and validation sets:
The initial neural network model that was trained on the chart image data did not perform well, with a final validation accuracy of only 20%. The model was compiled with stochastic gradient descent optimizer and sparse categorical crossentropy loss.

After splitting the data into training and validation sets using sklearn's train_test_split function with a test size of 0.2, the final evaluation metrics for the model showed a loss of 1.61 and an accuracy of only 16.5% on the test set.

The predicted classes for the test set were generated using the trained model and argmax function. However, classification report was not generated due to the low accuracy of the model.

shapes of the arrays:
the training set has 800 images with a shape of (128, 128, 3) representing the dimensions of the image and the number of color channels, while the training label array has 800 labels. Similarly, the test set has 200 images with the same shape as the training set, and the test label array has 200 labels.

modify the model architecture to cmnn:
The model seems to be improving the validation accuracy as the number of epochs increase until it reaches around 0.56 accuracy. However, the training accuracy is not improving and is even decreasing at some points, indicating that the model might be overfitting the training data.



The output of the cnn_model.evaluate() function indicates that the model was evaluated on the test set and achieved a loss of 1.2362 and an accuracy of 0.5850. This means that the model was able to correctly classify 58.5% of the samples in the test set.

Confusion matrix:
The confusion matrix is a useful tool to evaluate the performance of a classification model. In this case, the confusion matrix is visualized using a heatmap with annotations for each cell. The x-axis represents the predicted labels and the y-axis represents the actual labels. The diagonal cells indicate the correct predictions, while the off-diagonal cells represent the misclassifications.

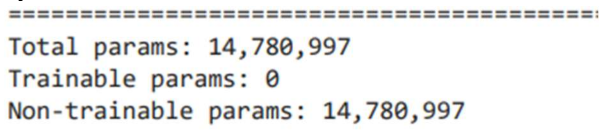
The confusion matrix shows that the model performed better in correctly predicting class 0 (negative sentiment) with 65 correct predictions and 35 misclassifications. However, the model struggled with class 1 (positive sentiment) with only 40 correct predictions and 60 misclassifications. The overall accuracy of the model is 0.585, which means that the model correctly classified 58.5% of the test data.

Based on this evaluation, it is clear that the model could benefit from further tuning or refinement to improve its performance on predicting positive sentiment.

for 50 iterations, we can see some promising accuracy, more training will be required for better accuracy and in the confusion matrix, whatever is not in diagonal is a error

Loading Pre-trained model and Replacing Classification layer:
The code defines a new model for image classification using transfer learning. The pre-trained VGG16 model, pre-trained on the ImageNet dataset, is loaded and the final classification layer is replaced with a new layer consisting of a global average pooling layer, a dense layer with 128 units and ReLU activation, and a dense layer with 5 units and softmax activation, corresponding to the 5 classes of images in the dataset. The new model is named pt_model. This approach allows for leveraging the pre-trained model's learned features, which can improve performance and reduce training time.

Summary:



```
==========================================
Total params: 14,780,997
Trainable params: 0
Non-trainable params: 14,780,997
```