**Neural Networks and Deep Learning – ICP 7**

**GitHub Link:** [**https://github.com/rumanathaskeen22/Neural-Networks-and-Deep-Learning-ICP-7**](https://github.com/rumanathaskeen22/Neural-Networks-and-Deep-Learning-ICP-7)

**Video Link:** [**https://drive.google.com/file/d/1xvqZps7aj61g8uIX4fn\_P982M-A-r6N\_/view?usp=sharing**](https://drive.google.com/file/d/1xvqZps7aj61g8uIX4fn_P982M-A-r6N_/view?usp=sharing)

**Name: Lnu Rumana Thaskeen**

**Student ID: 700742859**

**CRN: 23216**

**Deep Learning Image Classification with CNN**

Use Case Description:

Image Classification with CNN

1. Training the model

2. Evaluating the model

This code creates a simple Convolutional Neural Network (CNN) model for the CIFAR-10 dataset, which is a collection of 60,000 32x32 color images in 10 classes. The code first loads the dataset and preprocesses the data by normalizing it to the range of 0.0 to 1.0 and one-hot encoding the output labels. The model includes a convolutional input layer, dropout layers, convolutional layers, max-pooling layers, and fully connected layers. The model is compiled with stochastic gradient descent (SGD) optimizer and categorical cross-entropy loss function. The model is then trained on the dataset for 25 epochs with a batch size of 32, and the accuracy of the model is printed.

**Text

Description automatically generated**

(X\_train, y\_train) represents the training set, while (X\_test, y\_test) represents the test set. The training set is used to train the model, while the test set is used to evaluate the performance of the model on unseen data. The code given by you upon correction and execution gives the accuracy as 71.08% as shown below:

**Text, timeline

Description automatically generated**

**Text

Description automatically generated**

**Graphical user interface, text

Description automatically generated**

1. **Follow the instruction below and then report how the performance changed. (Apply all at once).**

To the above code I have made changes according to the question and added the following and applied them all at once as shown below.

Conv2D layer with 32 feature maps, kernel size of 3x3, padding='same', and activation function='relu'. The input\_shape is (32, 32, 3) which corresponds to the dimensions of the input image (height, width, depth). This layer applies 32 filters to the input image and each filter generates a feature map.

Dropout layer with a rate of 0.2. This layer randomly sets 20% of the input units to 0 at each update during training, which helps to prevent overfitting.

Conv2D layer with 32 feature maps, kernel size of 3x3, padding='same', and activation function='relu'. This layer applies another set of 32 filters to the input image, generating a new set of feature maps.

MaxPooling2D layer with pool\_size of 2x2. This layer downsamples the feature maps by taking the maximum value within each 2x2 block.

Conv2D layer with 64 feature maps, kernel size of 3x3, padding='same', and activation function='relu'. This layer applies 64 filters to the input image and generates 64 feature maps.

Dropout layer with a rate of 0.2.

Conv2D layer with 64 feature maps, kernel size of 3x3, padding='same', and activation function='relu'. This layer applies another set of 64 filters to the input image, generating a new set of feature maps.

MaxPooling2D layer with pool\_size of 2x2.

Conv2D layer with 128 feature maps, kernel size of 3x3, padding='same', and activation function='relu'. This layer applies 128 filters to the input image and generates 128 feature maps.

Dropout layer with a rate of 0.2.

Conv2D layer with 128 feature maps, kernel size of 3x3, padding='same', and activation function='relu'. This layer applies another set of 128 filters to the input image, generating a new set of feature maps.

MaxPooling2D layer with pool\_size of 2x2.

Flatten layer. This layer flattens the output of the previous layer, creating a 1D vector that can be used as input to a fully connected layer.

Dropout layer with a rate of 0.2.

Dense layer with 1024 units and activation function='relu'. This layer is fully connected and applies a rectified linear activation function to its inputs.

Dropout layer with a rate of 0.2.

Dense layer with 512 units and activation function='relu'.

Dropout layer with a rate of 0.2.

Dense layer with num\_classes=10 units and activation function='softmax'. This layer is the output layer and applies the softmax activation function to produce the output probabilities for each class.

**Text

Description automatically generated**

Then upon compiling the model to 25 epochs:

**Text

Description automatically generated**

**Graphical user interface, text

Description automatically generated**

The validation data (X\_test, y\_test) is used to evaluate the model after each epoch to monitor the training progress and to prevent overfitting. Upon final evaluation of the model, my code prints the accuracy as 80.58%. So, this is the accuracy of the model after applying the changes.

**Text

Description automatically generated**

**Text

Description automatically generated**

I have now printed the accuracy from above before and after adding the layers. I am also calculating the difference in accuracy to view the performance change. We can see that the performance improved by 9.50% after making the changes to the model.

There could be many reasons why the performance of the model changed. Here are some possibilities:

Data: The data could be different, for example, it could have different features, distributions, or outliers. This could affect the ability of the model to learn from the data and make accurate predictions.

Model: The model architecture, hyperparameters, or initialization could be different, which could affect the model's ability to learn and generalize to new data.

Environment: The environment in which the model was trained could be different. For example, the version of the libraries used, the hardware or the operating system could affect the performance of the model.

Randomness: Machine learning models often have some level of randomness in them, for example, in the way the data is split into training and testing sets, in the initialization of the weights, or in the shuffling of the data. As a result, even if the same data and model are used, the performance could be different due to different random seeds.

The performance change is as shown below:

**Text

Description automatically generated**

1. **Predict the first 4 images of the test data using the above model. Then, compare with the actual label for those 4 images to check whether or not the model has predicted correctly.**

I have now written the following code which is creating a prediction for the first four images in the test set using a trained neural network model.

First, a pandas DataFrame named prediction is created. Then an empty list called imageid is initialized. The for-loop iterates over the first four images in the test set, and for each iteration, the index i is incremented by 1 and appended to the imageid list.

After that, two new columns are added to the prediction DataFrame: "ImageId" and "Label". The "ImageId" column contains the imageid list, while the "Label" column contains the predicted labels for the corresponding images in the test set. The predicted labels are obtained by calling the argmax() function on the output of the model's predict() method, with the axis parameter set to -1.

Finally, the DataFrame is printed using the head() method, displaying the first five rows. Additionally, the actual labels for the first four images in the test set are printed using the argmax() function on the first four elements of y\_test array.

**Text

Description automatically generated**

**3. Visualize Loss and Accuracy using the history object**

I am now plotting the accuracy and loss of a trained neural network model during training and validation.

The history object returned by the fit() method contains information about the training process, including the accuracy and loss at each epoch for both training and validation datasets.

The code plots the accuracy and loss for both training and validation datasets over the course of the training process using plt.plot(). The title(), xlabel(), and ylabel() methods are used to set the title and axis labels of the plot. The legend() method is used to create a legend for the plot, indicating which lines correspond to which metrics. Finally, plt.show() is called to display the plot.

In summary, this code creates a visualization that allows us to see how the accuracy and loss of our neural network model changeover the course of training and validation.

**Text

Description automatically generated**

Upon executing the above code, the visualization of accuracy and loss is displayed as shown below:

**Chart, line chart

Description automatically generated**

**Chart, line chart

Description automatically generated**