# Classification on CIFAR-10 Dataset

**Kourosh Hassanzadeh**

1. Dataset:
   - The mentioned dataset is CIFAR-10, which includes 10 classes. The data distribution is as follows:

| | |
|---|---|
| 'test' | 10,000 |
| 'train' | 50,000 |

*Figure 1 - Initial dataset distribution*

   - with the last 5,000 samples of the set assigned to validation, as mentioned earlier.

   The dimensions of the images are 32 × 32 × 3.

```
x_train shape: (45000, 32, 32, 3)
y_train shape: (45000, 1)
x_val shape: (5000, 32, 32, 3)
y_val shape: (5000, 1)
x_test shape: (10000, 32, 32, 3)
y_test shape: (10000, 1)
num classes: 10
```

*Figure 2 - Dataset distribution after adding validation*

2. Preprocessing:
   - For this section, I divided the data values by 255.0 to normalize them and converted the image labels to a one-hot encoded vector to pass them into the model.
3. Model Construction:
   - Two images of the model's architecture are sent along with this report. In general, the architecture of the model is as follows:

```
model.add(Conv2D(filters=32, kernel_size=KERNEL_SIZE, input_shape=INPUT_SHAPE, activation='relu', padding='same', kernel_regularizer=l2(0.0001)))
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=KERNEL_SIZE, activation='relu', padding='same', kernel_regularizer=l2(0.0001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=64, kernel_size=KERNEL_SIZE, activation='relu', padding='same', kernel_regularizer=l2(0.0001)))
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=KERNEL_SIZE, activation='relu', padding='same', kernel_regularizer=l2(0.0001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=128, kernel_size=KERNEL_SIZE, activation='relu', padding='same', kernel_regularizer=l2(0.0001)))
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=KERNEL_SIZE, activation='relu', padding='same', kernel_regularizer=l2(0.0001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.0001)))
model.add(Dropout(0.25))
model.add(Dense(10, activation='softmax'))
```

*Figure 3 - model architecture*

- The explanations for each layer and its parameters are provided in the notebook file.
- This model is very simple, containing 551,786 parameters.
- For improving the model's performance, I used **L2 Regularization**, **Dropout**, and **Batch Normalization**, which slightly improved the accuracy.
- I also attempted to use the idea of **residual blocks** from the **ResNet** architecture. However, this approach not only increased the training time significantly but also decreased accuracy, leading me to decide to remove it.

4. Data Augmentation:
   - To increase the training data and avoid overfitting, I used the following data augmentation techniques, which also slightly improved the accuracy:

```
width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True
```

5. Compile and Train:
   - After constructing the model, I compiled it, defined the optimizer and loss function, and then trained the model (parameter explanations are given in the code). I trained the model with a batch size of 32 for 50 epochs.
   - Additionally, I used reduce_lr to adjust the learning rate during training, which gave better results.
   - As can be seen, there is not much difference between the accuracy on the training and validation data, indicating that the model has not overfit. Also, the accuracy on the test data, discussed in the next section, confirms that the model has been well-trained.

6. Model Evaluation and Testing:
   - The final accuracy on the training data is 89.23% with a loss of 45.69%.
   - The final accuracy on the validation data is 88.90% with a loss of 48.61%.
   - Finally, the model's accuracy on the test data is 88.33%.
   - I also calculated the F1 Score, precision, and recall, which are shown below:

```
Precision: 0.88
Recall: 0.88
F1 Score: 0.88
              precision    recall  f1-score   support

           0       0.89      0.89      0.89      1000
           1       0.94      0.96      0.95      1000
           2       0.87      0.81      0.84      1000
           3       0.83      0.72      0.77      1000
           4       0.88      0.87      0.88      1000
           5       0.84      0.83      0.83      1000
           6       0.82      0.96      0.89      1000
           7       0.91      0.92      0.92      1000
           8       0.94      0.93      0.94      1000
           9       0.91      0.95      0.93      1000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

*Figure 4 - Evaluation Metrics*

7. Graphs:
   - The following graphs were plotted during training: acc_loss graphs, ROC curve and Confusion Matrix.

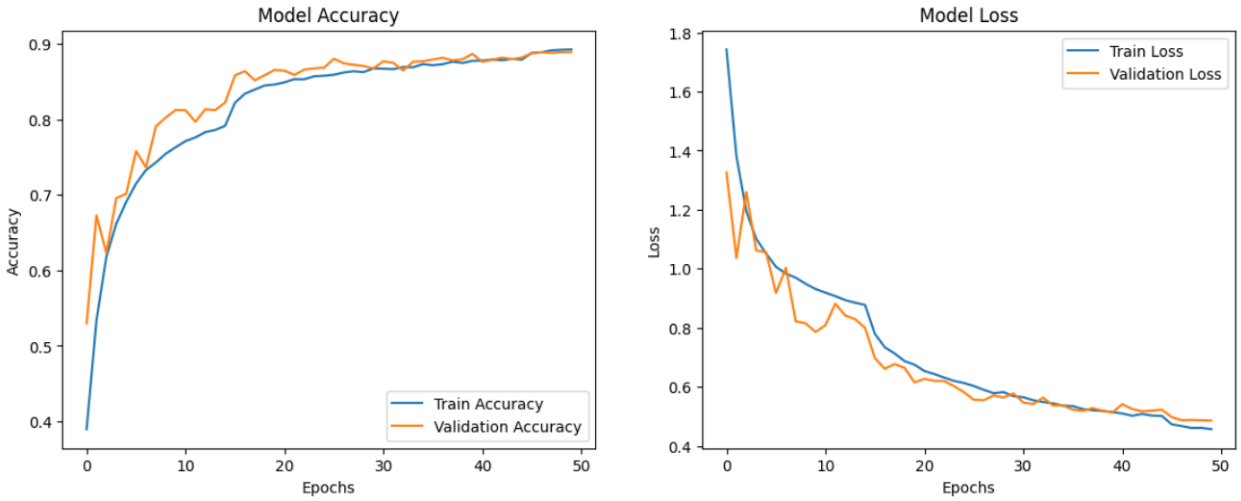


*Figure 5 - Accuracy and Loss Learning Curves*

   - As can be seen, the accuracy improved well during the training process, and the loss decreased, with no signs of overfitting in the validation chart.

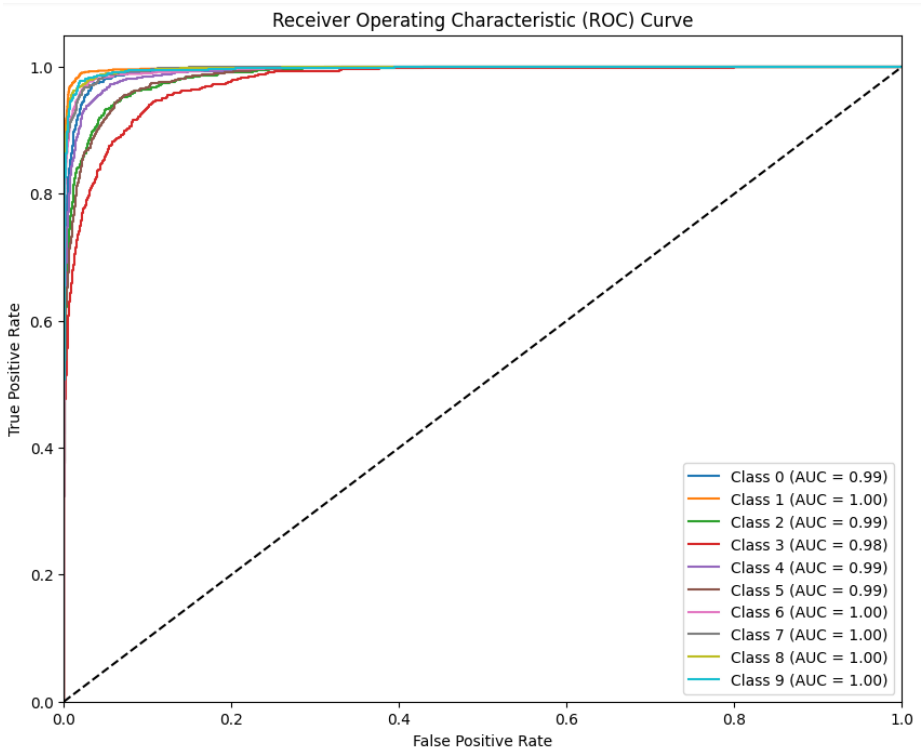


*Figure 6 - ROC Curve*

- The ROC curve is very high, indicating good model performance, with the TP rate being high and the FP rate low.
- The confusion matrix is as follows:

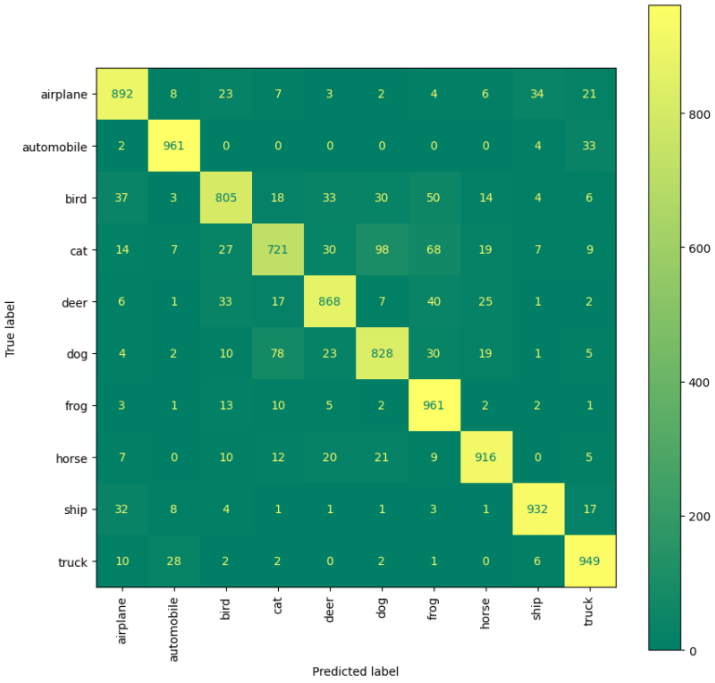


*Figure 7 - Confusion Matrix*

- As can be seen, the model performed worst on class 4.

## 8. Visualizing Images:

- Finally, I visualized some of the images with their predicted labels.

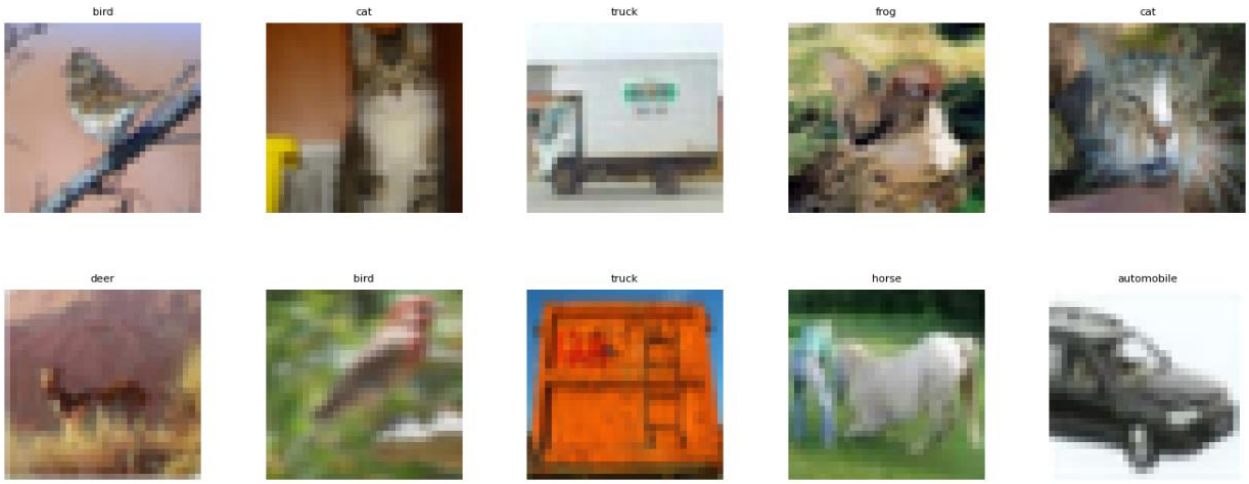


*Figure 8 - Analysis of Image Labels*