Week 5.1
Using Python's Deep Learning Tools to Generate and Evaluate a Convolutional Neural Network

Produced By: Abdullah Abdul Majid*(linkedin)*

Completion Date: August 3, 2024

Email: abdullahpisk@gmail.com

Jupyter Notebook Links:

GitHub

Table of Contents:

Problem Statement:

Use the Python's Deep Learning tools to import the CIFAR-10 dataset, prepare the dataset for model training, and then train a Convolutional Neural Network on it. Also use callbacks and evaluate the model.

Purpose:

The purpose of this assignment is to prepare and preprocess a colour image dataset; CIFAR-10, then perform and optimize a basic Convoluted Neural Network (CNN) model for it to get a better understanding of Deep Learning using the Keras and Tensorflow Python ML tools. The CIFAR-10 dataset has 60k 32x32 colour images with 10 classes containing 6k images each.

Process:

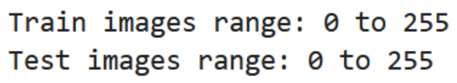
Importing and Loading the Dataset

- Imported all the necessary libraries, i.e pandas, numpy, sklearn, seaborn, matplotlib and, most importantly tensorflow.
- Import the CIFAR-10 dataset from Keras/Tensorfow using the following given command:
  from tensorflow.keras.datasets import cifar10

- Load the dataset using the following given command:
  (train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

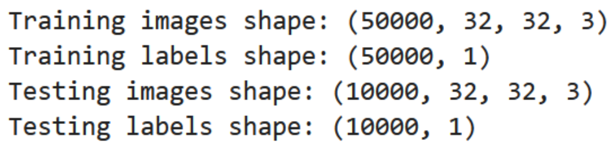
Preprocessing the Dataset

- Check the range of the dataset:



```
Train images range: 0 to 255
Test images range: 0 to 255
```

- Now, normalize the dataset to the range [0,1] to make it easier for the Neural Network to process.
- View the dataset shapes:



```
Training images shape: (50000, 32, 32, 3)
Training labels shape: (50000, 1)
Testing images shape: (10000, 32, 32, 3)
Testing labels shape: (10000, 1)
```

  The dataset has 60k 32x32 pixel images.
- Since our dataset has multiple classes and to avoid misalignment issues with the output, we must apply One-Hot Encoding to the labels (as demonstrated in the notebook.
- Augment the dataset using “ImageDataGenerator”. This randomizes the images’ orientation, shape and rotation to increase dataset diversity.
- Fit the ImageDataGenerator.

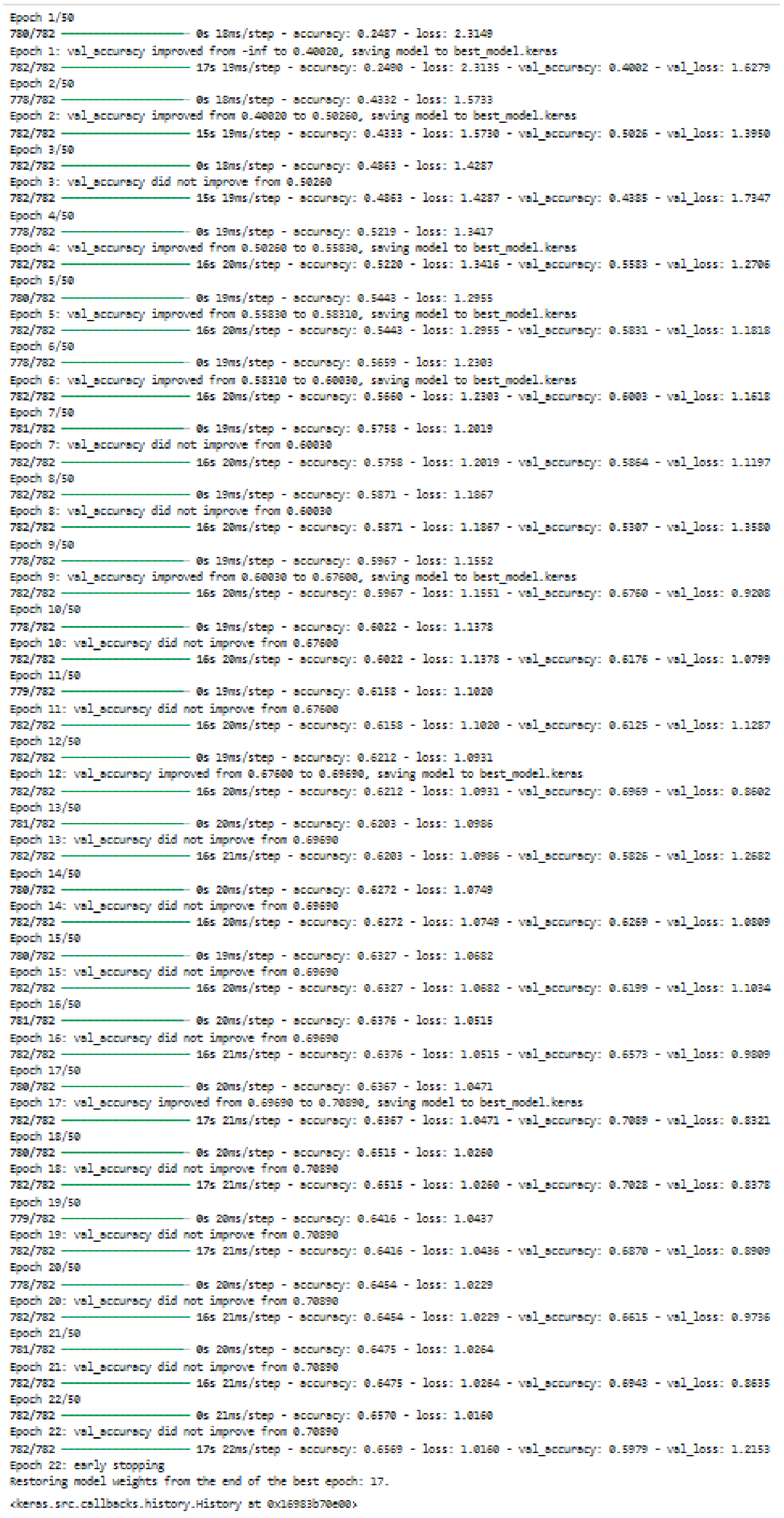
Defining and Creating the Keras CNN Model

- Create the Keras model with three convolutional layers alongside pooling layers. Also include Dropouts (reduces overfitting) and Batch Normalization (stabilizes and accelerates training) to improve model performance.
- Create the model and input the dataset shapes and number of classes.

- Compile the model using the “adam” optimizer (adaptive learning rates for various parameters) and the loss function “CategoricalCrossentropy” (for multi-class performance measurement). Also use the “accuracy” metric.
- Define the Model Checkpoint callback to save the best instance of the model to preserve it and provide a backup of the training progress in case of a failure.
- Define the Early Stopping callback to finish the CNN training as soon as the model stops improving. This speeds up the training progress and saves up on computing resources. In our case, the patience is set to 5. This means that Early Stopping will wait for 5 consecutive epochs in which the model doesn’t improve.

Training a Keras CNN Model

- Now train the model using the callbacks and 50 epochs on the augmented data.
- This may take some time, but our model will stop before 50 epochs if the Early Stopping callback takes effect.
- In our case, the model stopped at 41 epochs, the best model is stored in best_model.keras.
- Load the best model.
- The training process:



```
Epoch 1/50
780/782 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.2487 - loss: 2.3149
Epoch 1: val_accuracy improved from -inf to 0.40020, saving model to best_model.keras
782/782 ━━━━━━━━━━━━━━━━━━━━ 17s 19ms/step - accuracy: 0.2490 - loss: 2.3135 - val_accuracy: 0.4002 - val_loss: 1.6279
Epoch 2/50
778/782 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.4332 - loss: 1.5733
Epoch 2: val_accuracy improved from 0.40020 to 0.50260, saving model to best_model.keras
782/782 ━━━━━━━━━━━━━━━━━━━━ 15s 19ms/step - accuracy: 0.4333 - loss: 1.5730 - val_accuracy: 0.5026 - val_loss: 1.3950
Epoch 3/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.4863 - loss: 1.4287
Epoch 3: val_accuracy did not improve from 0.50260
782/782 ━━━━━━━━━━━━━━━━━━━━ 15s 19ms/step - accuracy: 0.4863 - loss: 1.4287 - val_accuracy: 0.4385 - val_loss: 1.7347
Epoch 4/50
778/782 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.5219 - loss: 1.3417
Epoch 4: val_accuracy improved from 0.50260 to 0.55830, saving model to best_model.keras
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 20ms/step - accuracy: 0.5220 - loss: 1.3416 - val_accuracy: 0.5583 - val_loss: 1.2706
Epoch 5/50
780/782 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.5443 - loss: 1.2955
Epoch 5: val_accuracy improved from 0.55830 to 0.58310, saving model to best_model.keras
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 20ms/step - accuracy: 0.5443 - loss: 1.2955 - val_accuracy: 0.5831 - val_loss: 1.1818
Epoch 6/50
778/782 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.5659 - loss: 1.2303
Epoch 6: val_accuracy improved from 0.58310 to 0.60030, saving model to best_model.keras
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 20ms/step - accuracy: 0.5660 - loss: 1.2303 - val_accuracy: 0.6003 - val_loss: 1.1618
Epoch 7/50
781/782 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.5758 - loss: 1.2019
Epoch 7: val_accuracy did not improve from 0.60030
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 20ms/step - accuracy: 0.5758 - loss: 1.2019 - val_accuracy: 0.5864 - val_loss: 1.1197
Epoch 8/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.5871 - loss: 1.1867
Epoch 8: val_accuracy did not improve from 0.60030
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 20ms/step - accuracy: 0.5871 - loss: 1.1867 - val_accuracy: 0.5307 - val_loss: 1.3580
Epoch 9/50
778/782 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.5967 - loss: 1.1552
Epoch 9: val_accuracy improved from 0.60030 to 0.67600, saving model to best_model.keras
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 20ms/step - accuracy: 0.5967 - loss: 1.1551 - val_accuracy: 0.6760 - val_loss: 0.9208
Epoch 10/50
778/782 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.6022 - loss: 1.1378
Epoch 10: val_accuracy did not improve from 0.67600
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 20ms/step - accuracy: 0.6022 - loss: 1.1378 - val_accuracy: 0.6176 - val_loss: 1.0799
Epoch 11/50
779/782 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.6158 - loss: 1.1020
Epoch 11: val_accuracy did not improve from 0.67600
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 20ms/step - accuracy: 0.6158 - loss: 1.1020 - val_accuracy: 0.6125 - val_loss: 1.1287
Epoch 12/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.6212 - loss: 1.0931
Epoch 12: val_accuracy improved from 0.67600 to 0.69690, saving model to best_model.keras
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 20ms/step - accuracy: 0.6212 - loss: 1.0931 - val_accuracy: 0.6969 - val_loss: 0.8602
Epoch 13/50
781/782 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.6203 - loss: 1.0986
Epoch 13: val_accuracy did not improve from 0.69690
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 21ms/step - accuracy: 0.6203 - loss: 1.0986 - val_accuracy: 0.5826 - val_loss: 1.2682
Epoch 14/50
780/782 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.6272 - loss: 1.0749
Epoch 14: val_accuracy did not improve from 0.69690
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 20ms/step - accuracy: 0.6272 - loss: 1.0749 - val_accuracy: 0.6269 - val_loss: 1.0809
Epoch 15/50
780/782 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.6327 - loss: 1.0682
Epoch 15: val_accuracy did not improve from 0.69690
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 20ms/step - accuracy: 0.6327 - loss: 1.0682 - val_accuracy: 0.6199 - val_loss: 1.1034
Epoch 16/50
781/782 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.6376 - loss: 1.0515
Epoch 16: val_accuracy did not improve from 0.69690
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 21ms/step - accuracy: 0.6376 - loss: 1.0515 - val_accuracy: 0.6573 - val_loss: 0.9809
Epoch 17/50
780/782 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.6367 - loss: 1.0471
Epoch 17: val_accuracy improved from 0.69690 to 0.70890, saving model to best_model.keras
782/782 ━━━━━━━━━━━━━━━━━━━━ 17s 21ms/step - accuracy: 0.6367 - loss: 1.0471 - val_accuracy: 0.7089 - val_loss: 0.8321
Epoch 18/50
780/782 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.6515 - loss: 1.0260
Epoch 18: val_accuracy did not improve from 0.70890
782/782 ━━━━━━━━━━━━━━━━━━━━ 17s 21ms/step - accuracy: 0.6515 - loss: 1.0260 - val_accuracy: 0.7028 - val_loss: 0.8378
Epoch 19/50
779/782 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.6416 - loss: 1.0437
Epoch 19: val_accuracy did not improve from 0.70890
782/782 ━━━━━━━━━━━━━━━━━━━━ 17s 21ms/step - accuracy: 0.6416 - loss: 1.0436 - val_accuracy: 0.6870 - val_loss: 0.8909
Epoch 20/50
778/782 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.6454 - loss: 1.0229
Epoch 20: val_accuracy did not improve from 0.70890
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 21ms/step - accuracy: 0.6454 - loss: 1.0229 - val_accuracy: 0.6615 - val_loss: 0.9736
Epoch 21/50
781/782 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.6475 - loss: 1.0264
Epoch 21: val_accuracy did not improve from 0.70890
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 21ms/step - accuracy: 0.6475 - loss: 1.0264 - val_accuracy: 0.6943 - val_loss: 0.8635
Epoch 22/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.6570 - loss: 1.0160
Epoch 22: val_accuracy did not improve from 0.70890
782/782 ━━━━━━━━━━━━━━━━━━━━ 17s 22ms/step - accuracy: 0.6569 - loss: 1.0160 - val_accuracy: 0.5979 - val_loss: 1.2153
Epoch 22: early stopping
Restoring model weights from the end of the best epoch: 17.
<keras.src.callbacks.history.History at 0x16983b70e00>
```

Model Evaluation

- Get the accuracy and test loss by comparing with the one-hot encoded test set:

```
313/313 - 1s - 2ms/step - accuracy: 0.7089 - loss: 0.8321
Test Loss: 0.8320946097373962
Test Accuracy: 0.708899974822998
```

  The test loss and accuracy are acceptable.

- Generate predictions and print the classification report:

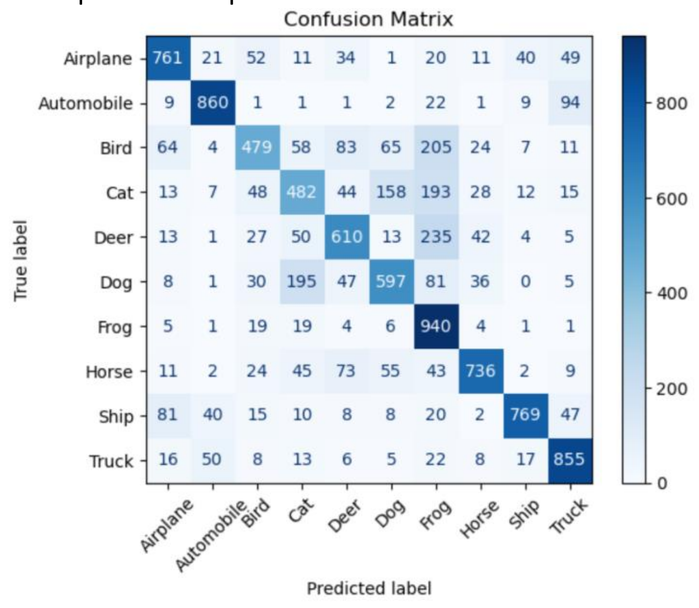
```
Classification Report:
              precision    recall  f1-score   support

    Airplane       0.78      0.76      0.77      1000
  Automobile       0.87      0.86      0.87      1000
        Bird       0.68      0.48      0.56      1000
         Cat       0.55      0.48      0.51      1000
        Deer       0.67      0.61      0.64      1000
         Dog       0.66      0.60      0.63      1000
        Frog       0.53      0.94      0.68      1000
       Horse       0.83      0.74      0.78      1000
        Ship       0.89      0.77      0.83      1000
       Truck       0.78      0.85      0.82      1000

    accuracy                           0.71     10000
   macro avg       0.72      0.71      0.71     10000
weighted avg       0.72      0.71      0.71     10000
```

  The Classification results are acceptable, but not great.

- Compute and plot the Confusion Matrix:



  Our CNN model is pretty good as diagonal elements are the hottest.

Conclusion

In this assignment, I was able to experiment more with Deep Learning using the Tensorflow and Keras tools in Python. I imported the CIFAR-10 dataset, preprocessed it and used callbacks to improve the Neural Network's performance. I also augmented the dataset to increase diversity. In the end, I got an accurate model as evident by the Confusion Matrix and Classification Report evaluations.