Assignment 2

IMAGE PROCESSING AND PREDICTION

CT70A9600 Intelligent Systems and Services

HARITHA PALOLY SANTHOSHKUMAR

MODEL 1: CNN model

First, I defined the CNN model using 3 Convolutional layers where 1st and 2nd convolutional layers have 32 filters of size (3,3) each, and 3rd convolutional layer with 64 filters. The convolutional layers have relu activation as well to keep non-linearity. I used 2 max-pooling layers with (2,2) size. The dropout layer has been defined with a dropout rate of 0.4, to prevent overfitting. Flatten layer could convert the 2D features to 1D. The fully connected layer with 1024 neurons and relu could imprint the complex features of the images. In the last layer, the softmax function is also added, as there are multiple classes of images are being trained.

During the training of this CNN model, I could observe the accuracy of 72%, but when adding one more convolutional layers has impacted the efficiency by dropping the accuracy to 60%. And I have tried to increase the number of filters to 128 in the fourth convolutional layer, that also had reduced the accuracy to 64%. This might have happened due to the overfitting as the number of layers and filters are being increased.

Fig 1 shows the classification report of the CNN model, in which the test accuracy is 71%. There were 10 epochs used, and the model learned fast after each epoch. After 5-6 epochs, the stability of the accuracy has been attained. The images of automobile and ship (F1-score: 0.83 & 0.82 respectively) are the classes learned by the model most effectively. The model struggled to learn the images from the class of cat, bird, and deer. In addition, as the recall value for the class cat shows very little, there can be chances for more false negatives, as a result, cat images could have been misclassified.

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 33s 19ms/step - accuracy: 0.3635 - loss: 1.7485
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 27s 17ms/step - accuracy: 0.5358 - loss: 1.3011
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 41s 17ms/step - accuracy: 0.5881 - loss: 1.1608
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 27s 17ms/step - accuracy: 0.6208 - loss: 1.0778
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 27s 17ms/step - accuracy: 0.6542 - loss: 0.9838
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 28s 18ms/step - accuracy: 0.6710 - loss: 0.9287
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 28s 18ms/step - accuracy: 0.6883 - loss: 0.8901
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 28s 18ms/step - accuracy: 0.6991 - loss: 0.8604
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 29s 18ms/step - accuracy: 0.7145 - loss: 0.8217
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 39s 17ms/step - accuracy: 0.7195 - loss: 0.8025
```

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

    airplane       0.73      0.78      0.75      1000
  automobile       0.79      0.88      0.83      1000
        bird       0.70      0.56      0.62      1000
         cat       0.73      0.32      0.44      1000
        deer       0.74      0.56      0.64      1000
         dog       0.67      0.64      0.66      1000
        frog       0.60      0.90      0.72      1000
       horse       0.74      0.78      0.76      1000
        ship       0.82      0.81      0.82      1000
       truck       0.65      0.88      0.75      1000

    accuracy                           0.71     10000
   macro avg       0.72      0.71      0.70     10000
weighted avg       0.72      0.71      0.70     10000
```

```
loss, accuracy = CNNmodel.evaluate(x_test, y_test)
print('Test Accuracy =', accuracy)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7080 - loss: 0.8641
Test Accuracy = 0.7102000117301941
```

Fig.1 : Classification Report of CNN model

Using hyperparameters such as the learning rate, epoch, and optimizers, the performance of the model could be enhanced. The number of epochs was 10, and the learning rates were 0.001, 0.01, and 0.1. Among the learning rate, 0.001 is the best learning rate as it could provide a comparatively larger accuracy, of 75% (71% during initial training without optimizers and learning rates). The optimizers Adam and SGD provided good accuracy with learning rate 0.01 and all batch sizes (32,64,128), as it could find the best weights to enhance the performance of the model. The higher learning rates(0.1 and 0.01) are also leading to poor learning of this model. However, there was only 1 epoch used to evaluate the performance of the model on the usage of different hyperparameters and their combinations. See Appendix(Table 1).

MODEL 2: ResNet model

ResNet50 model that uses the image of sizes 224 x 224 pixels by default, as input shape. The pre-trained model of ResNet50 has been used as the backbone for further training of the model. The pre-trained weights are being loaded from ImageNet which is a huge dataset for image classification. For fine-tuning, initially, all the layers are made frozen so that the weights will not be updated during the training, and some layers are unfrozen.

Upsampling layers are being used here to enlarge the input images from the CIFAR-10 dataset which are 32x32 pixels, as ResNet50 expects the input images to be 224x224 by default.

```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset
  self._warn_if_super_not_called()
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 516s 289ms/step - accuracy: 0.5347 - loss: 1.3277
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 436s 279ms/step - accuracy: 0.7422 - loss: 0.7451
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 436s 279ms/step - accuracy: 0.7952 - loss: 0.5976
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 442s 279ms/step - accuracy: 0.8253 - loss: 0.4996
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 436s 279ms/step - accuracy: 0.8506 - loss: 0.4326
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 436s 279ms/step - accuracy: 0.8685 - loss: 0.3835
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 436s 279ms/step - accuracy: 0.8826 - loss: 0.3454
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 436s 279ms/step - accuracy: 0.8935 - loss: 0.3070
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 435s 278ms/step - accuracy: 0.9073 - loss: 0.2716
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 434s 278ms/step - accuracy: 0.9151 - loss: 0.2457
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 33s 92ms/step
Classification Report:
              precision    recall  f1-score   support

    airplane       0.73      0.95      0.83      1000
  automobile       0.96      0.81      0.88      1000
        bird       0.91      0.70      0.79      1000
         cat       0.78      0.67      0.72      1000
        deer       0.77      0.91      0.84      1000
         dog       0.87      0.74      0.80      1000
        frog       0.80      0.96      0.87      1000
       horse       0.95      0.84      0.89      1000
        ship       0.93      0.87      0.90      1000
       truck       0.80      0.94      0.87      1000

    accuracy                           0.84     10000
   macro avg       0.85      0.84      0.84     10000
weighted avg       0.85      0.84      0.84     10000
```

```
loss, accuracy = Resnet50_model.evaluate(x_test, y_test)
print('Test Accuracy =', accuracy)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 31s 85ms/step - accuracy: 0.8452 - loss: 0.5343
Test Accuracy = 0.8396000266075134
```

Fig.2 : Classification Report of ResNet model

ResNet using optimizers has exhibited better accuracy along with good learning rates. Fig 2 shows the classification report of the ResNet model. The images of ships, trucks, and automobiles, are being learned by the model effectively. The images of dogs and cats have been challenging for the ResNet model for learning. I could also observe that fast learning is being undergone in the model after each epoch. During the 1st epoch, the model exhibited around 50% accuracy, and after seeing the whole image dataset repeatedly the accuracy also gradually improved to attain the accuracy of about 84-90% towards the final epochs.

While performing fine-tuning of the model, I could observe that by avoiding the learning of weights through the freezing of layers, and later by unfreezing them there was almost two times increase in the accuracy. This indicates that the model is trying to learn effectively and efficiently.

Hyperparameters such as learning rates and optimizers have been used in different batch sizes. The learning rates 0.001 and 0.01 seem to be providing good accuracy for the RMSprop optimizer. Using optimizer SGD also a higher accuracy can be achieved.

Moreover, compared to CNN, the model accuracy is greater for ResNet. The resNet model is a quick learner when compared to the CNN model. When CNN learns the dataset with 36% accuracy in the initial epochs, reaching up to 71% whereas ResNet attained an accuracy of 50% in the very first epochs led to a greater accuracy of the model. While comparing the classification reports of these models, I could observe that the image classes of cats have exhibited lower F1 scores in both of these models. However, the CNN had the lowest results of recall and F1-score, indicating that its ability to identify the features of cats and classify according to respective class is less when compared to ResNet. As there are images of both cats and dogs in the dataset, the CNN could not identify the complex features or might have thrown more false positives than the ResNet model.

APPENDIX

Optimizer	Learning rate	Batch size	Accuracy %
Adam	0.001	32	73.59
Adam	**0.001**	**64**	**74.87**
Adam	**0.001**	**128**	**75.62**
SGD	**0.001**	**32**	**75.66**
SGD	0.001	64	75.37
SGD	0.001	**128**	**75.81**
RMSprop	0.001	32	73.46
RMSprop	0.001	64	72.85
RMSprop	0.001	128	74.21
Adam	0.01	32	41.76
Adam	0.01	64	49.12
Adam	0.01	128	53.11
SGD	0.01	32	57.23
SGD	0.01	64	56.48
SGD	0.01	128	58.38
RMSprop	0.01	32	34.09
RMSprop	0.01	64	48.41
RMSprop	0.01	128	46.41
Adam	***0.1***	32	***10***
Adam	***0.1***	64	***10***
Adam	***0.1***	128	***10***
SGD	***0.1***	32	***10***
SGD	***0.1***	64	***10***
SGD	***0.1***	128	***10***
RMSprop	***0.1***	32	***10***
RMSprop	***0.1***	64	***10***
RMSprop	***0.1***	128	***10***

```
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 31s 18ms/step - accuracy: 0.7305 - loss: 0.7705
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7359 - loss: 0.7827
Learning Rate: 0.001, Optimizer: adam, Batch Size: 32, Test Accuracy: 0.7358999848365784
782/782 ━━━━━━━━━━━━━━━━━━━━ 30s 33ms/step - accuracy: 0.7575 - loss: 0.6951
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7448 - loss: 0.7407
Learning Rate: 0.001, Optimizer: adam, Batch Size: 64, Test Accuracy: 0.7487000226974487
391/391 ━━━━━━━━━━━━━━━━━━━━ 30s 68ms/step - accuracy: 0.7771 - loss: 0.6309
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7562 - loss: 0.7240
Learning Rate: 0.001, Optimizer: adam, Batch Size: 128, Test Accuracy: 0.7562999725341797
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 29s 18ms/step - accuracy: 0.7926 - loss: 0.5967
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7567 - loss: 0.7265
Learning Rate: 0.001, Optimizer: SGD, Batch Size: 32, Test Accuracy: 0.7566999793052673
782/782 ━━━━━━━━━━━━━━━━━━━━ 27s 32ms/step - accuracy: 0.8050 - loss: 0.5715
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7531 - loss: 0.7205
Learning Rate: 0.001, Optimizer: SGD, Batch Size: 64, Test Accuracy: 0.7537000179290771
391/391 ━━━━━━━━━━━━━━━━━━━━ 27s 62ms/step - accuracy: 0.7956 - loss: 0.5824
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7586 - loss: 0.7147
Learning Rate: 0.001, Optimizer: SGD, Batch Size: 128, Test Accuracy: 0.7581999897956848
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 31s 18ms/step - accuracy: 0.7555 - loss: 0.7073
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7349 - loss: 0.7977
Learning Rate: 0.001, Optimizer: RMSprop, Batch Size: 32, Test Accuracy: 0.7346000075340271
782/782 ━━━━━━━━━━━━━━━━━━━━ 28s 33ms/step - accuracy: 0.7626 - loss: 0.6804
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7263 - loss: 0.8314
Learning Rate: 0.001, Optimizer: RMSprop, Batch Size: 64, Test Accuracy: 0.7285000085830688
391/391 ━━━━━━━━━━━━━━━━━━━━ 27s 63ms/step - accuracy: 0.7837 - loss: 0.6318
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.7420 - loss: 0.7683
Learning Rate: 0.001, Optimizer: RMSprop, Batch Size: 128, Test Accuracy: 0.7421000003814697
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 30s 18ms/step - accuracy: 0.3615 - loss: 1.8643
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4212 - loss: 1.6652
Learning Rate: 0.01, Optimizer: adam, Batch Size: 32, Test Accuracy: 0.41769999265670776
782/782 ━━━━━━━━━━━━━━━━━━━━ 28s 32ms/step - accuracy: 0.4308 - loss: 1.5916
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4893 - loss: 1.4375
Learning Rate: 0.01, Optimizer: adam, Batch Size: 64, Test Accuracy: 0.4912000000476837
391/391 ━━━━━━━━━━━━━━━━━━━━ 27s 62ms/step - accuracy: 0.4802 - loss: 1.4459
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5310 - loss: 1.3023
Learning Rate: 0.01, Optimizer: adam, Batch Size: 128, Test Accuracy: 0.5311999917030334
```

Table 1: CNN model's performance results on using hyperparameters

Optimizer	Learning rate	Batch size	Accuracy %
Adam	0.001	32	64.03
SGD	0.001	32	86.28
RMSProp	0.001	32	83.12
Adam	0.001	64	89.51
SGD	0.001	64	93.59
RMSProp	**0.001**	**64**	**95.26**
Adam	0.001	128	59.70
SGD	0.001	128	69.34
RMSProp	0.001	128	71.49
Adam	0.01	32	68.82
SGD	0.01	32	83.39
RMSProp	0.01	32	86.21
Adam	0.01	64	86.81
SGD	0.01	64	92.22
RMSProp	**0.01**	**64**	**94.52**
Adam	0.01	128	17.43
SGD	0.01	128	34.64
RMSProp	0.01	128	36.88

```
Training with LR=0.001, Batch Size=32, Optimizer=Adam
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 553s 314ms/step - accuracy: 0.6403 - loss: 1.0822 - val_accuracy: 0.7182 - val_loss: 0.8671
Training with LR=0.001, Batch Size=32, Optimizer=SGD
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 477s 290ms/step - accuracy: 0.8628 - loss: 0.4050 - val_accuracy: 0.8545 - val_loss: 0.4257
Training with LR=0.001, Batch Size=32, Optimizer=RMSprop
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 503s 298ms/step - accuracy: 0.8312 - loss: 0.4882 - val_accuracy: 0.7534 - val_loss: 0.7207
Training with LR=0.001, Batch Size=64, Optimizer=Adam
782/782 ━━━━━━━━━━━━━━━━━━━━ 514s 574ms/step - accuracy: 0.8951 - loss: 0.3104 - val_accuracy: 0.8084 - val_loss: 0.5710
Training with LR=0.001, Batch Size=64, Optimizer=SGD
782/782 ━━━━━━━━━━━━━━━━━━━━ 451s 546ms/step - accuracy: 0.9359 - loss: 0.1910 - val_accuracy: 0.8803 - val_loss: 0.3489
Training with LR=0.001, Batch Size=64, Optimizer=RMSprop
782/782 ━━━━━━━━━━━━━━━━━━━━ 474s 558ms/step - accuracy: 0.9526 - loss: 0.1344 - val_accuracy: 0.8662 - val_loss: 0.4380

Training with LR=0.01, Batch Size=32, Optimizer=Adam
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 575s 335ms/step - accuracy: 0.6882 - loss: 0.9253 - val_accuracy: 0.5600 - val_loss: 1.7814
Training with LR=0.01, Batch Size=32, Optimizer=SGD
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 504s 308ms/step - accuracy: 0.8389 - loss: 0.4740 - val_accuracy: 0.8231 - val_loss: 0.5134
Training with LR=0.01, Batch Size=32, Optimizer=RMSprop
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 537s 320ms/step - accuracy: 0.8621 - loss: 0.4018 - val_accuracy: 0.7489 - val_loss: 0.8282
Training with LR=0.01, Batch Size=64, Optimizer=Adam
782/782 ━━━━━━━━━━━━━━━━━━━━ 575s 647ms/step - accuracy: 0.8681 - loss: 0.3805 - val_accuracy: 0.7907 - val_loss: 0.6297
Training with LR=0.01, Batch Size=64, Optimizer=SGD
782/782 ━━━━━━━━━━━━━━━━━━━━ 504s 615ms/step - accuracy: 0.9222 - loss: 0.2244 - val_accuracy: 0.8686 - val_loss: 0.3863
Training with LR=0.01, Batch Size=64, Optimizer=RMSprop
782/782 ━━━━━━━━━━━━━━━━━━━━ 514s 610ms/step - accuracy: 0.9452 - loss: 0.1614 - val_accuracy: 0.8338 - val_loss: 0.6030

Training with LR=0.01, Batch Size=128, Optimizer=Adam
391/391 ━━━━━━━━━━━━━━━━━━━━ 569s 1s/step - accuracy: 0.1743 - loss: 2.7410 - val_accuracy: 0.1000 - val_loss: 6.1773
Training with LR=0.01, Batch Size=128, Optimizer=SGD
391/391 ━━━━━━━━━━━━━━━━━━━━ 462s 1s/step - accuracy: 0.3464 - loss: 1.7289 - val_accuracy: 0.2256 - val_loss: 2.0669
Training with LR=0.01, Batch Size=128, Optimizer=RMSprop
391/391 ━━━━━━━━━━━━━━━━━━━━ 481s 1s/step - accuracy: 0.3688 - loss: 1.6766 - val_accuracy: 0.4065 - val_loss: 1.8880
Training with LR=0.001, Batch Size=128, Optimizer=Adam
391/391 ━━━━━━━━━━━━━━━━━━━━ 525s 1s/step - accuracy: 0.5970 - loss: 1.1081 - val_accuracy: 0.6325 - val_loss: 1.0213
Training with LR=0.001, Batch Size=128, Optimizer=SGD
391/391 ━━━━━━━━━━━━━━━━━━━━ 467s 1s/step - accuracy: 0.6934 - loss: 0.8490 - val_accuracy: 0.6839 - val_loss: 0.8902
Training with LR=0.001, Batch Size=128, Optimizer=RMSprop
391/391 ━━━━━━━━━━━━━━━━━━━━ 500s 1s/step - accuracy: 0.7149 - loss: 0.7935 - val_accuracy: 0.5779 - val_loss: 1.2420
```

Table 2 : ResNet model's performance results on using hyperparameters

AI DISCLOSURE

I acknowledge the use of ChatGPT in assisting the programming syntax and libraries. This document was developed with the support of Grammarly which helped in improving grammar, vocabulary, and style according to the assignment.