NEW YORK INSTITUTE OF TECHNOLOGY

Deep Learning (Spring 2024)
Project Assignment 1

Ramis Rawnak, ID: 1319750

Data: A deep learning CNN that is an already pretrained model available in Keras, a CIFAR-10 Dataset.

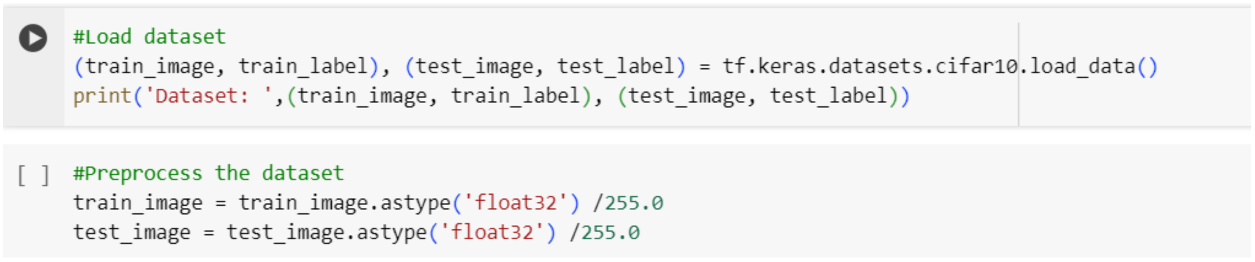
Classification Task: The dataset contains training (50000) instances and use the test instances (10000), each instance belongs to one of the 10 classes.

Metrics: Two convolutional neural network models: (1) SimCNN, a simple CNN models built from scratch, and (2) ResNet50 were used to perform classification tasks on the CIFAR-10 Dataset.

Task: With the given training and testing dataset from CIFAR-10, perform classification with SimCNN and find the classification accuracy of SimCNN. Likewise, using the same dataset, perform classification with ResNet50 and find the classification accuracy of ResNet50.

i) Train and test the dataset with the convolutional neural network models

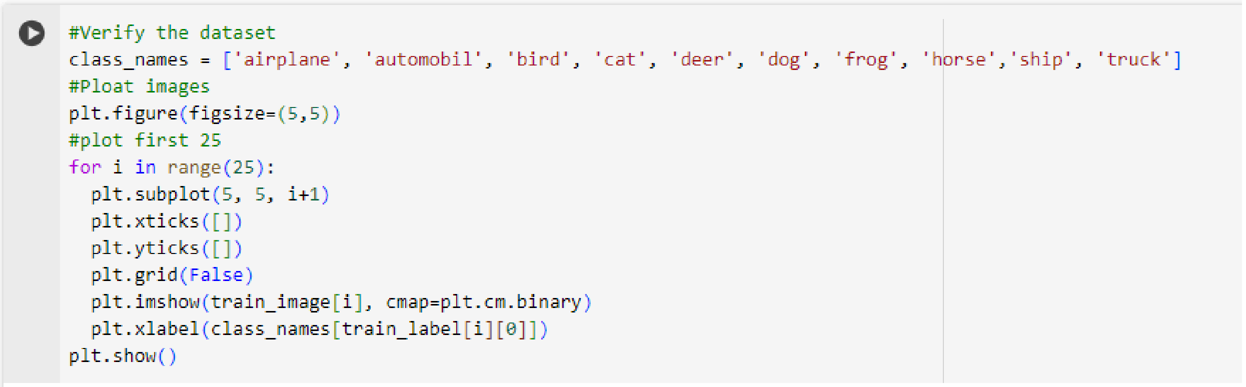
At first I have trained this CIFAR-10 dataset using SimCNN and then evaluated the accuracy of the model using SimCNN



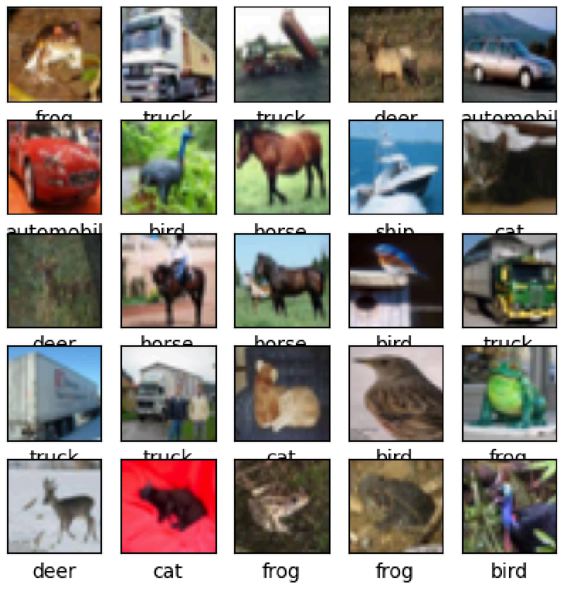
```
#Load dataset
(train_image, train_label), (test_image, test_label) = tf.keras.datasets.cifar10.load_data()
print('Dataset: ',(train_image, train_label), (test_image, test_label))
```

```
[ ] #Preprocess the dataset
train_image = train_image.astype('float32') /255.0
test_image = test_image.astype('float32') /255.0
```

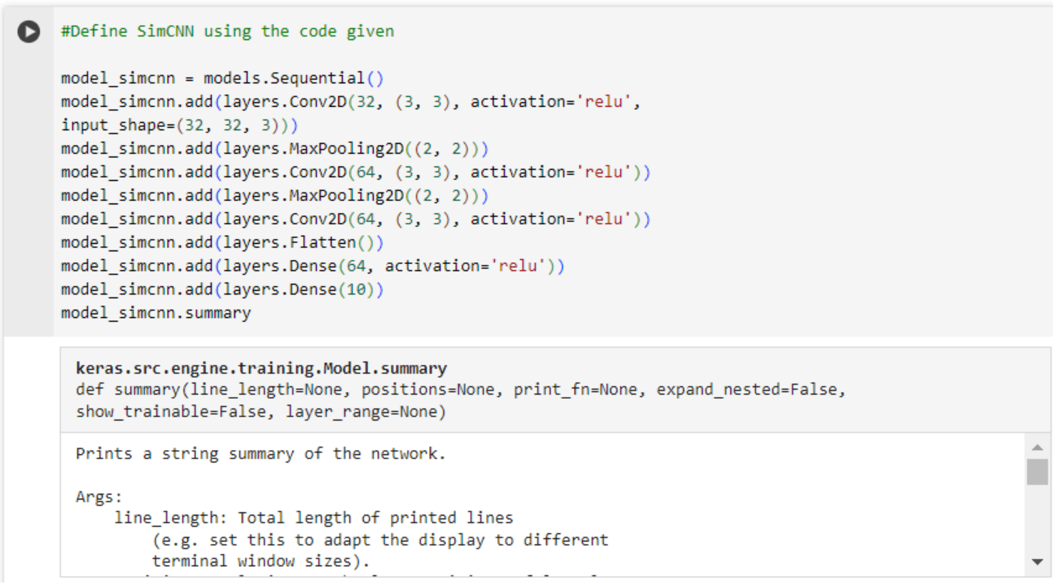
Loaded the dataset into (train image, train_label) and (test_image, test_label)
The preprocessed the dataset as shown above.



```
#Verify the dataset
class_names = ['airplane', 'automobil', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse','ship', 'truck']
#Ploat images
plt.figure(figsize=(5,5))
#plot first 25
for i in range(25):
  plt.subplot(5, 5, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(train_image[i], cmap=plt.cm.binary)
  plt.xlabel(class_names[train_label[i][0]])
plt.show()
```



Verified the dataset using the class names as shown above.

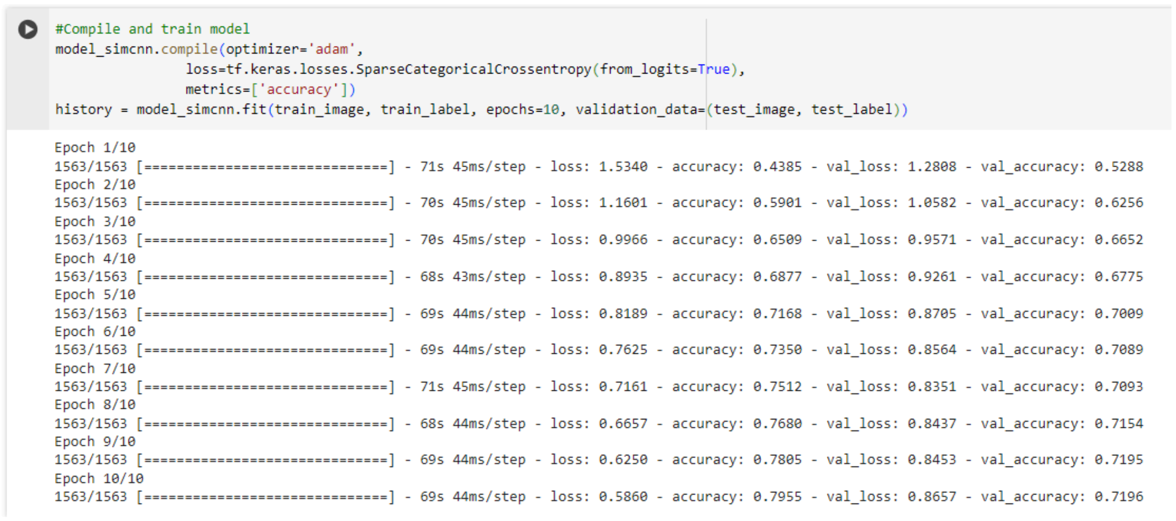


```
#Define SimCNN using the code given

model_simcnn = models.Sequential()
model_simcnn.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(32, 32, 3)))
model_simcnn.add(layers.MaxPooling2D((2, 2)))
model_simcnn.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_simcnn.add(layers.MaxPooling2D((2, 2)))
model_simcnn.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_simcnn.add(layers.Flatten())
model_simcnn.add(layers.Dense(64, activation='relu'))
model_simcnn.add(layers.Dense(10))
model_simcnn.summary
```

```
keras.src.engine.training.Model.summary
def summary(line_length=None, positions=None, print_fn=None, expand_nested=False,
show_trainable=False, layer_range=None)
```

```
Prints a string summary of the network.

Args:
    line_length: Total length of printed lines
        (e.g. set this to adapt the display to different
        terminal window sizes).
```

In order to train the model using SimCNN, we established SimCNN using the code shown above.

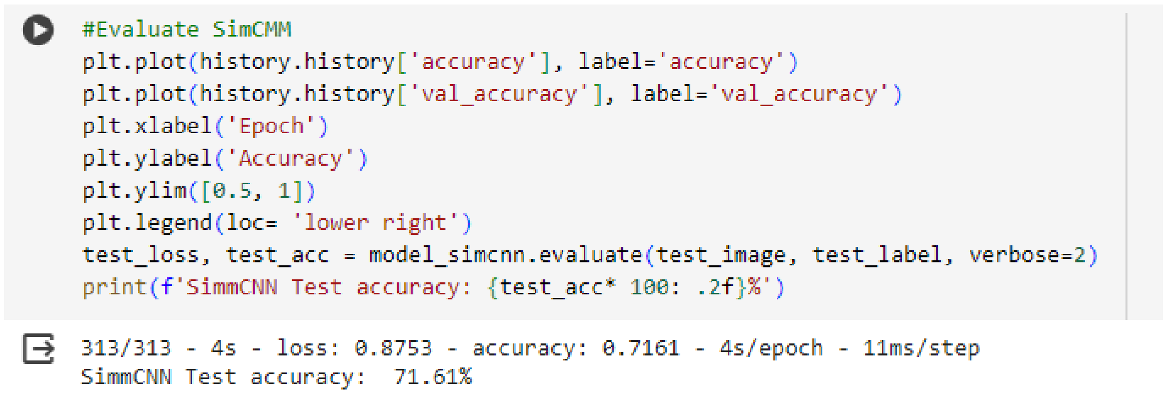


```
#Compile and train model
model_simcnn.compile(optimizer='adam',
                loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])
history = model_simcnn.fit(train_image, train_label, epochs=10, validation_data=(test_image, test_label))
```

```
Epoch 1/10
1563/1563 [==============================] - 71s 45ms/step - loss: 1.5340 - accuracy: 0.4385 - val_loss: 1.2808 - val_accuracy: 0.5288
Epoch 2/10
1563/1563 [==============================] - 70s 45ms/step - loss: 1.1601 - accuracy: 0.5901 - val_loss: 1.0582 - val_accuracy: 0.6256
Epoch 3/10
1563/1563 [==============================] - 70s 45ms/step - loss: 0.9966 - accuracy: 0.6509 - val_loss: 0.9571 - val_accuracy: 0.6652
Epoch 4/10
1563/1563 [==============================] - 68s 43ms/step - loss: 0.8935 - accuracy: 0.6877 - val_loss: 0.9261 - val_accuracy: 0.6775
Epoch 5/10
1563/1563 [==============================] - 69s 44ms/step - loss: 0.8189 - accuracy: 0.7168 - val_loss: 0.8705 - val_accuracy: 0.7009
Epoch 6/10
1563/1563 [==============================] - 69s 44ms/step - loss: 0.7625 - accuracy: 0.7350 - val_loss: 0.8564 - val_accuracy: 0.7089
Epoch 7/10
1563/1563 [==============================] - 71s 45ms/step - loss: 0.7161 - accuracy: 0.7512 - val_loss: 0.8351 - val_accuracy: 0.7093
Epoch 8/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.6657 - accuracy: 0.7680 - val_loss: 0.8437 - val_accuracy: 0.7154
Epoch 9/10
1563/1563 [==============================] - 69s 44ms/step - loss: 0.6250 - accuracy: 0.7805 - val_loss: 0.8453 - val_accuracy: 0.7195
Epoch 10/10
1563/1563 [==============================] - 69s 44ms/step - loss: 0.5860 - accuracy: 0.7955 - val_loss: 0.8657 - val_accuracy: 0.7196
```

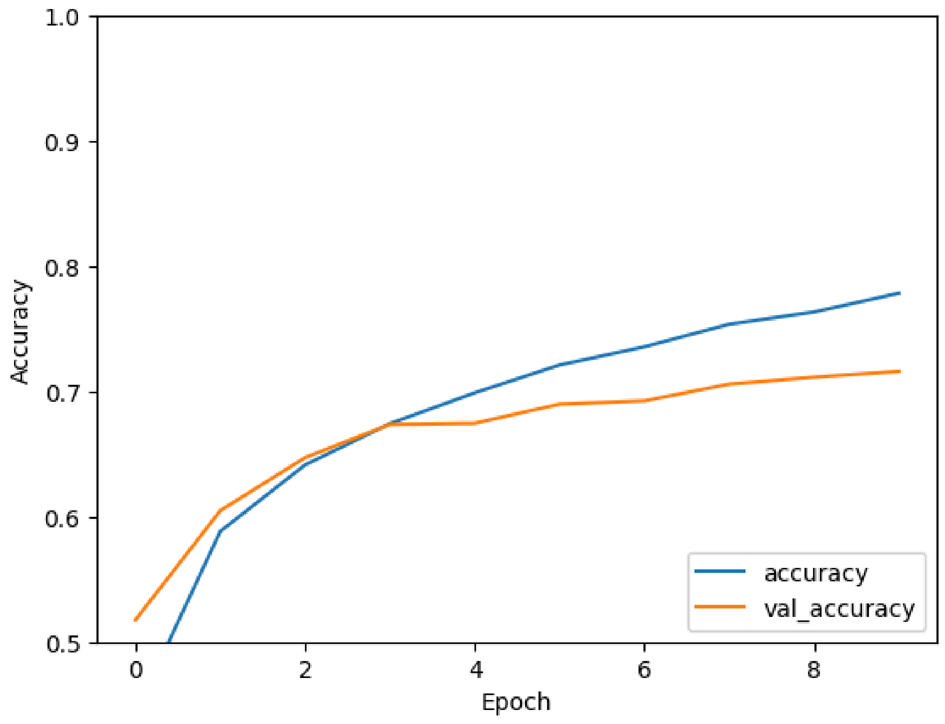
Then, the model was compiled and trained. We can observe that the loss is decreasing showing that the model is actually able to learn from the training dataset. The validation loss is also decreasing at the beginning from epochs 1-7 and then increasing slightly which could be due to an underlying overfitting issue.

The accuracy is increasing which shows that the model is improving on the training dataset and the validation accuracy is also increasing showing that the model is also performing well on unseen data but towards the end it is increasing little bit, this could be due to an overfitting issue as mentioned above.



```
#Evaluate SimCMM
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc= 'lower right')
test_loss, test_acc = model_simcnn.evaluate(test_image, test_label, verbose=2)
print(f'SimmCNN Test accuracy: {test_acc* 100: .2f}%')
```

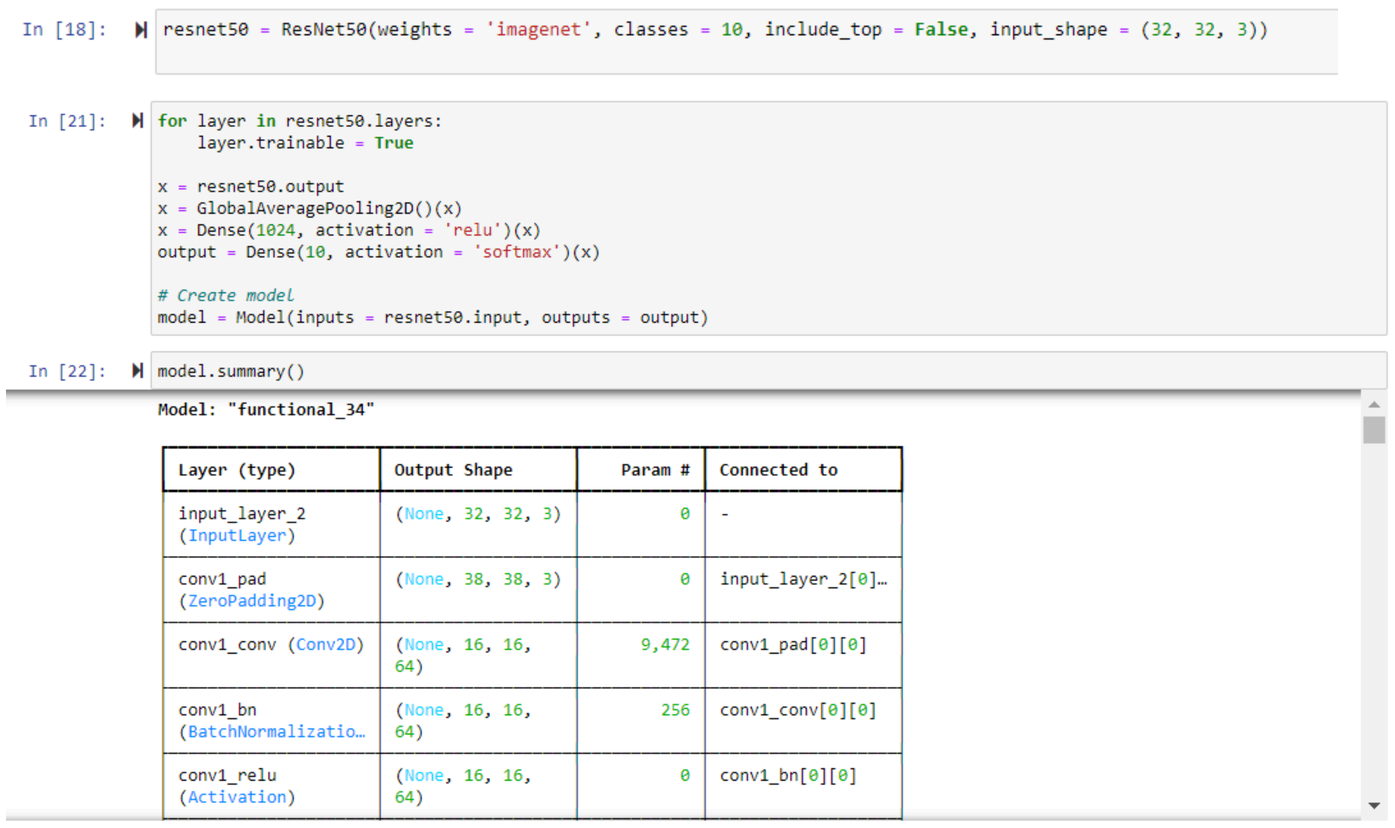
```
313/313 - 4s - loss: 0.8753 - accuracy: 0.7161 - 4s/epoch - 11ms/step
SimmCNN Test accuracy:  71.61%
```



After conducting an evaluation of the model using SimCNN, we can conclude that the final accuracy of the model to correctly classify the images given in the dataset is 71.61%. As per the plot, we can observe that accuracy is increasing in a steady manner while the val_accuracy increases and decreases in a steady manner indicating an overfitting issue.

While reflecting on the overall performance of the SimCNN model, it showed proficiency in learning the underlying patterns that are present in the CIFAR-10 dataset. However, we did observe some overfitting issues and necessary improvements can be made to avoid in the future by the implementation of regularization techniques i.e. data augmentation or dropout layers. This can help improve the model's performance further.

For the next task , we have trained the dataset using ResNet50:

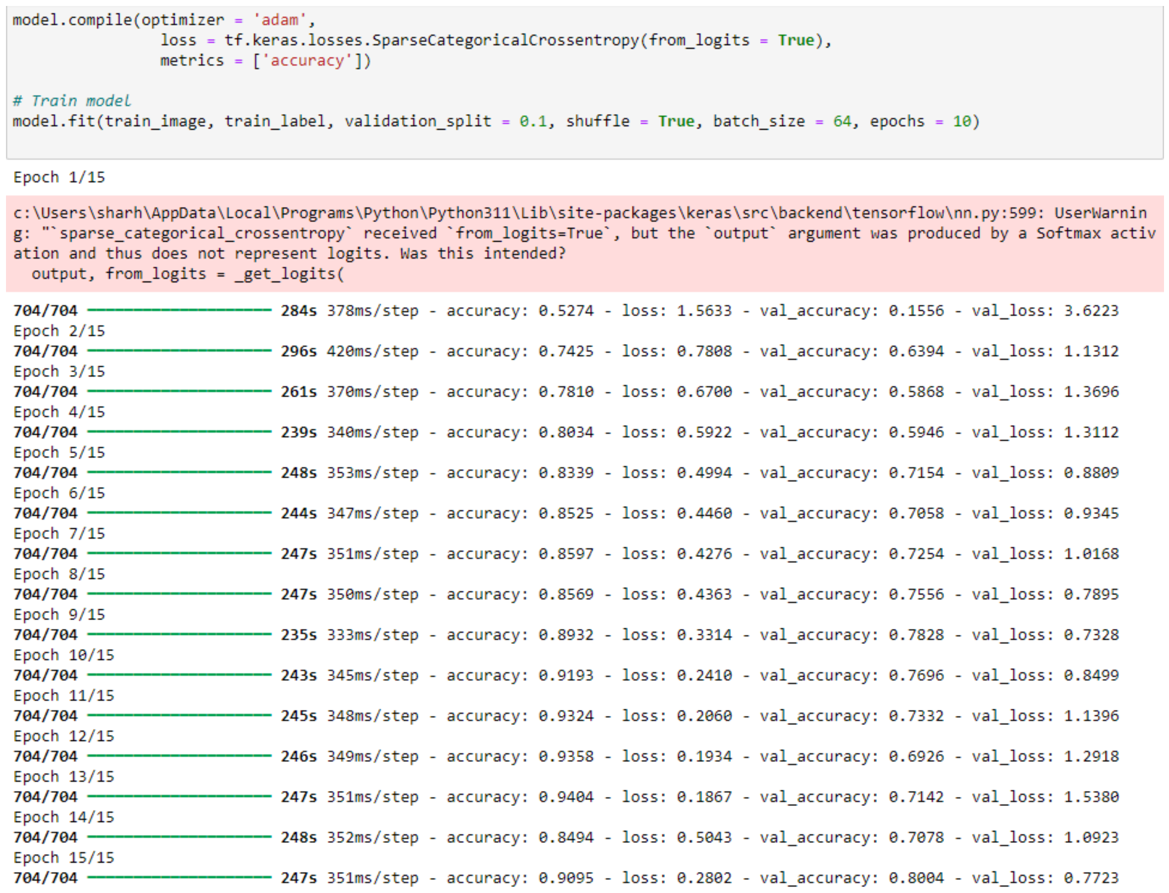


```
In [18]:  resnet50 = ResNet50(weights = 'imagenet', classes = 10, include_top = False, input_shape = (32, 32, 3))
```

```
In [21]:  for layer in resnet50.layers:
              layer.trainable = True

          x = resnet50.output
          x = GlobalAveragePooling2D()(x)
          x = Dense(1024, activation = 'relu')(x)
          output = Dense(10, activation = 'softmax')(x)

          # Create model
          model = Model(inputs = resnet50.input, outputs = output)
```

```
In [22]:  model.summary()
```

```
Model: "functional_34"

┌─────────────────────────────┬───────────────────────┬────────────┬────────────────────┐
│ Layer (type)                │ Output Shape          │    Param # │ Connected to       │
├─────────────────────────────┼───────────────────────┼────────────┼────────────────────┤
│ input_layer_2               │ (None, 32, 32, 3)     │          0 │ -                  │
│ (InputLayer)                │                       │            │                    │
├─────────────────────────────┼───────────────────────┼────────────┼────────────────────┤
│ conv1_pad                   │ (None, 38, 38, 3)     │          0 │ input_layer_2[0]…  │
│ (ZeroPadding2D)             │                       │            │                    │
├─────────────────────────────┼───────────────────────┼────────────┼────────────────────┤
│ conv1_conv (Conv2D)         │ (None, 16, 16,        │      9,472 │ conv1_pad[0][0]    │
│                             │ 64)                   │            │                    │
├─────────────────────────────┼───────────────────────┼────────────┼────────────────────┤
│ conv1_bn                    │ (None, 16, 16,        │        256 │ conv1_conv[0][0]   │
│ (BatchNormalizatio…         │ 64)                   │            │                    │
├─────────────────────────────┼───────────────────────┼────────────┼────────────────────┤
│ conv1_relu                  │ (None, 16, 16,        │          0 │ conv1_bn[0][0]     │
│ (Activation)                │ 64)                   │            │                    │
├─────────────────────────────┼───────────────────────┼────────────┼────────────────────┤
```

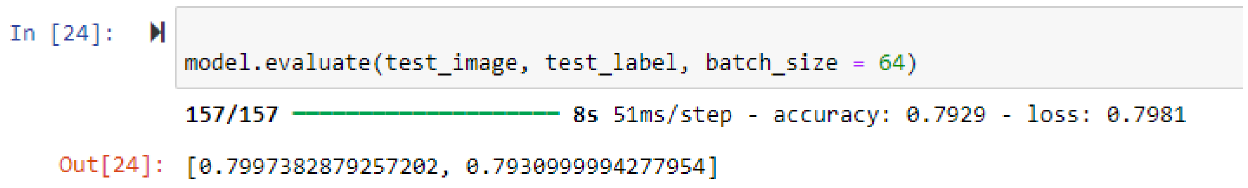


```
model.compile(optimizer = 'adam',
              loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True),
              metrics = ['accuracy'])

# Train model
model.fit(train_image, train_label, validation_split = 0.1, shuffle = True, batch_size = 64, epochs = 10)
```

```
Epoch 1/15
c:\Users\sharh\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\backend\tensorflow\nn.py:599: UserWarnin
g: "`sparse_categorical_crossentropy` received `from_logits=True`, but the `output` argument was produced by a Softmax activ
ation and thus does not represent logits. Was this intended?
  output, from_logits = _get_logits(
704/704 ━━━━━━━━━━━━━━━━━━━━ 284s 378ms/step - accuracy: 0.5274 - loss: 1.5633 - val_accuracy: 0.1556 - val_loss: 3.6223
Epoch 2/15
704/704 ━━━━━━━━━━━━━━━━━━━━ 296s 420ms/step - accuracy: 0.7425 - loss: 0.7808 - val_accuracy: 0.6394 - val_loss: 1.1312
Epoch 3/15
704/704 ━━━━━━━━━━━━━━━━━━━━ 261s 370ms/step - accuracy: 0.7810 - loss: 0.6700 - val_accuracy: 0.5868 - val_loss: 1.3696
Epoch 4/15
704/704 ━━━━━━━━━━━━━━━━━━━━ 239s 340ms/step - accuracy: 0.8034 - loss: 0.5922 - val_accuracy: 0.5946 - val_loss: 1.3112
Epoch 5/15
704/704 ━━━━━━━━━━━━━━━━━━━━ 248s 353ms/step - accuracy: 0.8339 - loss: 0.4994 - val_accuracy: 0.7154 - val_loss: 0.8809
Epoch 6/15
704/704 ━━━━━━━━━━━━━━━━━━━━ 244s 347ms/step - accuracy: 0.8525 - loss: 0.4460 - val_accuracy: 0.7058 - val_loss: 0.9345
Epoch 7/15
704/704 ━━━━━━━━━━━━━━━━━━━━ 247s 351ms/step - accuracy: 0.8597 - loss: 0.4276 - val_accuracy: 0.7254 - val_loss: 1.0168
Epoch 8/15
704/704 ━━━━━━━━━━━━━━━━━━━━ 247s 350ms/step - accuracy: 0.8569 - loss: 0.4363 - val_accuracy: 0.7556 - val_loss: 0.7895
Epoch 9/15
704/704 ━━━━━━━━━━━━━━━━━━━━ 235s 333ms/step - accuracy: 0.8932 - loss: 0.3314 - val_accuracy: 0.7828 - val_loss: 0.7328
Epoch 10/15
704/704 ━━━━━━━━━━━━━━━━━━━━ 243s 345ms/step - accuracy: 0.9193 - loss: 0.2410 - val_accuracy: 0.7696 - val_loss: 0.8499
Epoch 11/15
704/704 ━━━━━━━━━━━━━━━━━━━━ 245s 348ms/step - accuracy: 0.9324 - loss: 0.2060 - val_accuracy: 0.7332 - val_loss: 1.1396
Epoch 12/15
704/704 ━━━━━━━━━━━━━━━━━━━━ 246s 349ms/step - accuracy: 0.9358 - loss: 0.1934 - val_accuracy: 0.6926 - val_loss: 1.2918
Epoch 13/15
704/704 ━━━━━━━━━━━━━━━━━━━━ 247s 351ms/step - accuracy: 0.9404 - loss: 0.1867 - val_accuracy: 0.7142 - val_loss: 1.5380
Epoch 14/15
704/704 ━━━━━━━━━━━━━━━━━━━━ 248s 352ms/step - accuracy: 0.8494 - loss: 0.5043 - val_accuracy: 0.7078 - val_loss: 1.0923
Epoch 15/15
704/704 ━━━━━━━━━━━━━━━━━━━━ 247s 351ms/step - accuracy: 0.9095 - loss: 0.2802 - val_accuracy: 0.8004 - val_loss: 0.7723
```

Then, the model was compiled and trained. We can observe that the loss is decreasing significantly but increases for the last two epochs. This shows it is performing well on the training dataset but shifts due to some overfitting issues. The val_loss on the other hand, increases on epoch 2 and then decreases until epoch 4. Overall, there is inconsistency in the val accuracy loss indicating that there are more overfitting issues here and thus is not performing well on unseen data.

The accuracy is increasing significantly showing the model is performing well on the trained data. However for val_loss, there is discrepancy as it increases and also decreases, it is not consistent, at the end it decreases a lot at epoch 15, thus indicating an overfitting issue.



```
In [24]:  model.evaluate(test_image, test_label, batch_size = 64)

          157/157 ━━━━━━━━━━━━━━━━━━━━ 8s 51ms/step - accuracy: 0.7929 - loss: 0.7981

Out[24]:  [0.7997382879257202, 0.7930999994277954]
```

After training the dataset, the model was evaluated on the test dataset. Yielding a loss of 0.7981 and a test accuracy of 79.29%. The accuracy rate of the model indicated that it has performed pretty well on the test dataset. However, increased test loss value indicates it has not performed well on the unseen dataset; this could be due to overfitting issues. To get more accurate results and low test loss results in the future, the dataset can be refined more so the model can perform better on the test dataset by the implementation of regularization techniques i.e. data augmentation or dropout layers.

ii) Compare and contrast, Analyze the two models:

The parameters used:

- Batch size: A batch size of 32 was used for SimCNN which is a default setting for CNN models.
  A batch size of 64 was used for ResNet50, as it is a common use. A larger batch size can lead to supporting computational efficiency of GPUs.

- Steps per epoch: 1562 steps per epoch (50000/32 for SimCNN), this is the total instances divided by the batch size.
  781 steps per epoch(50000/64 for ResNet50).

- Epochs: 10 (SimCNN)
- 15(ResNet50)
  The epochs were chosen based on practical observations and experiment to avoid Risk of overfitting and underfitting during the training phase.

- Validation Steps: 0.1
- The 10% of the training data were chosen to see the model's
- Performance on unseen data during training. However, it is to be noted that the validation step has not been distinctly specified in the provided code snippets.

- Optimizer: Adam was used as the optimizer as it is a common preference used for deep learning task due to its high learning rate potentials.

- Learning rate: default

- Decay: default

- Momentum: default

The above 3 parameters are set to default to minimize the configuration process and make it time efficient.

Training and evaluation: In this report, as mentioned above with steps and code snippets, SimCNN and ResNet 50 models were trained on the CIFAR-10 dataset using the above parameters. At the same time the accuracy of the models were also evaluated.

The Classification Accuracies of both models are given below:

- SimCNN: 71.61%

- ResNet50: 79.29%

Analysis:

The classification accuracies of SimCNN and ResNet 50 models evaluated by utilizing the above hyperparameters taken and numbers modified form Table of the paper “Transfer Learning Based on ResNet50”. The hyperparameters include a batch size of 32 for SimCNN and a batch size of 64 for ResNet50 respectively. The steps per epochs for SimCNN is 1562 And 781 for ResNet 50, 10 epochs set for SimCNN and 15 epochs set for ResNet50. In addition, both Models were optimized with Adam optimizer, while the learning rate, decay and momentum are set to default.

Regardless of having similar hyperparameters and techniques, the performance results of Both the models displayed significant differences. As per the classification accuracy, ResNet50 is performing better at 79.29% accuracy while SimCNN is at 71.61%. However, training the SimCNN the model did not take much longer and it was pretty straightforward. But the ResNet 50 model took many days to train and often crash as ResNet50 is a big model. At first the ResNet 50 was being trained in google collab, it was not providing with all the outputs even after using GPU setting as per collab’s suggestion. It was switched to Jupyter Notebook and after 6 hours it provided the outputs. This could be the due the complex architecture of ResNet 50 while SimCNN is a simpler architecture. ResNet50 lacks practicality due to its high computational demand and this can also lead to expenses, which might not always be feasible.

In contrast, SimCNN has offered more time efficient and straightforward training process, requiring less computational demands. While its accuracy rate was lower than ResNet50, it is more of a practical model to use especially for times when there is limited computation resources and the time to train is constrained.

To conclude, in choosing between the two models would depend on many factors i.e. computational demands, training time, external computational resources and our classification accuracy goal. Further research can be conducted to optimize the training process of ResNet50 to address issues such as crashing problems, time efficiency and to maximize its potential to achieve excellent performance.

References:

1. Transfer Learning Based Plant Diseases Detection Using ResNet50[PDF]
2. ResNet50 Transfer Learning CIFAR-10 Beginner. Kaggle. [https://www.kaggle.com/code/kutaykutlu/resnet50-transfer-learning-cifar-10-beginner]
3. Convolutional Neural Network (CNN). Tensorflow. [https://www.tensorflow.org/tutorials/images/cnn]