Emerging Technologies

Deep Learning Exercise and Technical Report: Medium-Sized Mammals [#13]

Danica Marie A. Dumalagan & Krystelle U. Timtim

Background

The CIFAR (Canadian Institute for Advanced Research) dataset is a collection of images that are commonly used to train machine learning and computer vision algorithms. CIFAR-100 is a large image dataset with 100 classes, each containing 600 images. It was created to promote research in computer vision and pattern recognition, as well as to provide a benchmark for comparing different deep learning models. CIFAR-100 has been used to evaluate the performance of various deep learning architectures and to advance the state-of-the-art in image classification, object detection, and other related tasks. The dataset has also been used as a testbed for exploring transfer learning and semi-supervised learning techniques, which can help improve model performance when labeled data is limited.

In this exercise, a simple custom-made convolutional neural network (CNN) model will be trained using the classes assigned under the thirteenth group *(coarse class: medium-sized mammals; fine class: fox, porcupine, possum, raccoon, skunk)* in the CIFAR-100 dataset.

Deep Learning Structure and Training Configurations

With the small dataset of 25, a simple CNN model architecture was constructed with these steps:

Import library

```python
import tensorflow as tf
from tensorflow import keras
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Dropout
import matplotlib.pyplot as plt
```

keras

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

matplot.pyplot
A state-based interface to matplotlib. It provides an implicit, MATLAB-like, way of plotting. It also opens figures on your screen, and acts as the figure GUI manager.

Load data from CIFAR-100 dataset
As instructed, the 13th group *(index: 12; coarse class: medium-sized mammals; fine class: fox, porcupine, possum, raccoon, skunk)* in the CIFAR-100 dataset was used as training data in this exercise. The array was then flattened, then, the five fine classes were assigned to labels 0 to 4 respectively *(fox: 0, porcupine: 1, possum: 2, raccoon: 3, skunk: 4)*.

Building the model

```python
# Build the model
model = tf.keras.Sequential()

model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32,
32, 3)))
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(5, activation='softmax'))

model.summary()
```

tf.keras.Sequential()
This is a function in TensorFlow that creates a linear stack of layers to build a neural network model. As a beginner, such was used as it provides a simple and easy-to-use interface for building neural network models in TensorFlow.

Conv2D
The conv2d function in Keras is a 2-dimensional convolutional layer that applies a specified number of filters to the input data, allowing the neural network to learn spatial hierarchies of patterns in images or other 2D data. Using multiple Conv2D layers allows

learning hierarchical and abstract representations of the input data, capturing both low-level and high-level features that can lead to better accuracy in image recognition tasks.

MaxPooling2D
MaxPooling2D is a down-sampling operation that takes the maximum value within a window of pixels in a 2D array, reducing its spatial size while preserving its key features. Using MaxPooling2D reduces the spatial dimensionality of the input while retaining the most important features, making the network more robust to variations in the input and reducing overfitting.

Flatten
Flatten is a function in Keras that converts a multidimensional input tensor into a one-dimensional vector, which is commonly used as the input layer of a neural network's fully connected layers.

Dense
Dense is a function in Keras that creates a fully connected layer where each neuron in the layer is connected to all neurons in the previous layer, and allows customization of the activation function, regularization, and initialization of weights and biases. Dense function enables learning complex representations of the data by stacking multiple layers of fully connected neurons with non-linear activations.

relu
ReLU (Rectified Linear Unit) is an activation function in Keras that sets all negative values to zero, allowing the neural network to learn more efficiently by introducing non-linearity while avoiding the vanishing gradient problem.

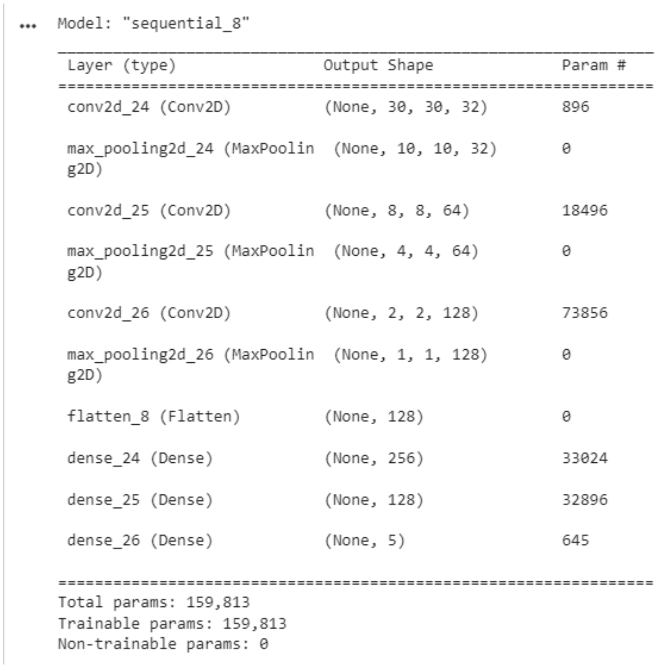
softmax
Softmax is an activation function in Keras that transforms a vector of arbitrary real values into a probability distribution over several classes, making it useful for multi-class classification tasks, compatible with the desired outcome of this exercise.

Compiling the model

```python
history= model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

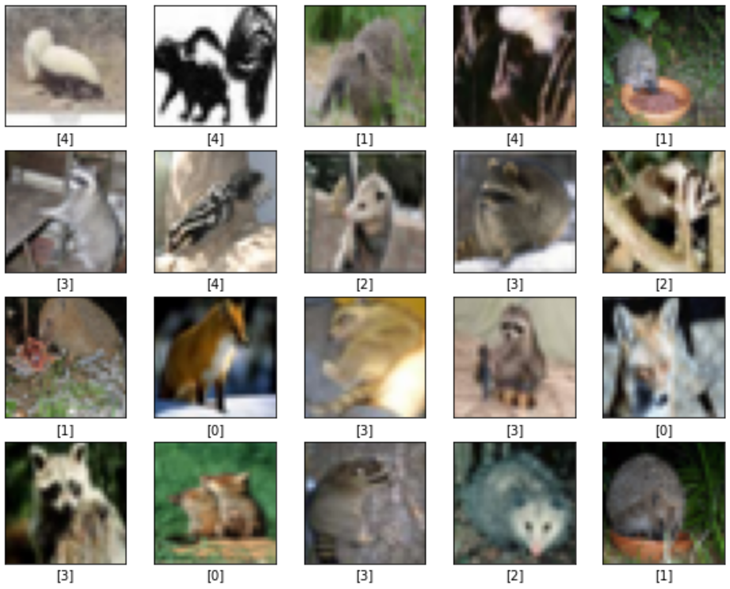
adam
Adam is an optimization algorithm in Keras that uses adaptive learning rates and momentum to update the weights during training, allowing for faster convergence and better performance.

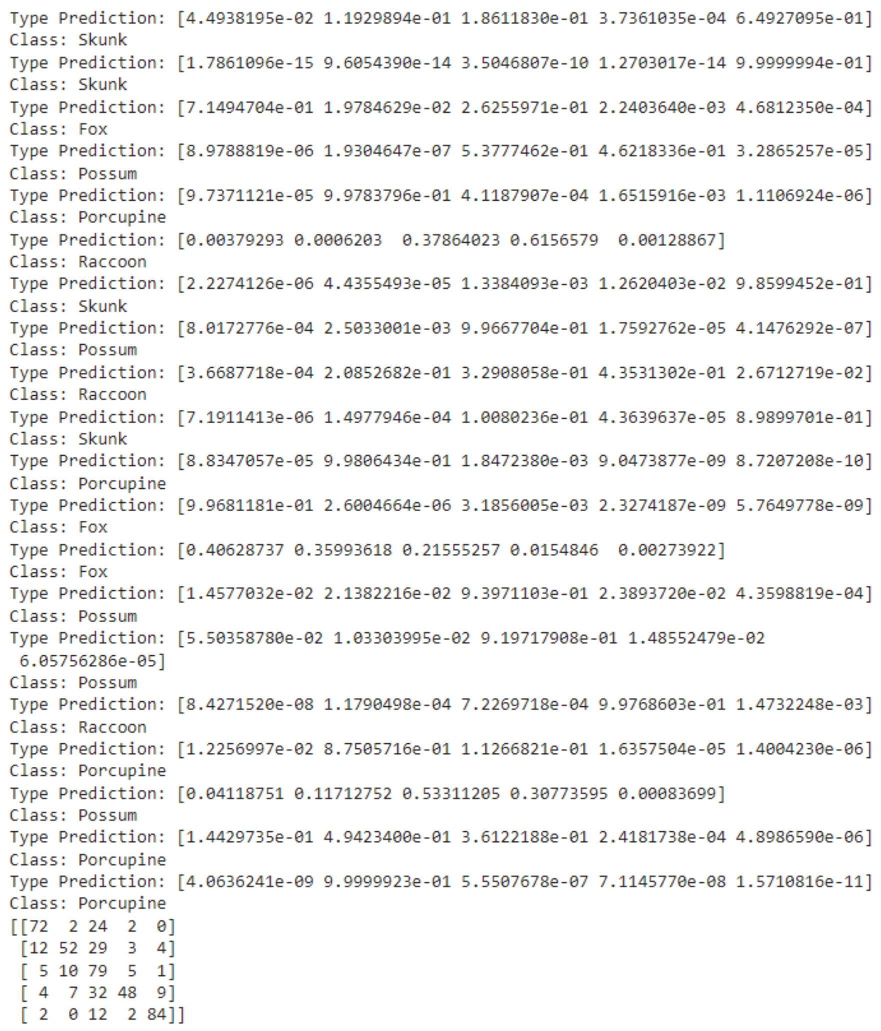
sparse_categorical_crossentropy
This is a loss function in Keras that is used for multi-class classification problems with integer labels, where the output is a probability distribution over several classes, making it suitable for tasks with a large number of classes such as in this exercise.



```
... Model: "sequential_8"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     conv2d_24 (Conv2D)          (None, 30, 30, 32)        896

     max_pooling2d_24 (MaxPoolin  (None, 10, 10, 32)       0
     g2D)

     conv2d_25 (Conv2D)          (None, 8, 8, 64)          18496

     max_pooling2d_25 (MaxPoolin  (None, 4, 4, 64)         0
     g2D)

     conv2d_26 (Conv2D)          (None, 2, 2, 128)         73856

     max_pooling2d_26 (MaxPoolin  (None, 1, 1, 128)        0
     g2D)

     flatten_8 (Flatten)         (None, 128)               0

     dense_24 (Dense)            (None, 256)               33024

     dense_25 (Dense)            (None, 128)               32896

     dense_26 (Dense)            (None, 5)                 645

    =================================================================
    Total params: 159,813
    Trainable params: 159,813
    Non-trainable params: 0
```

Testing the model





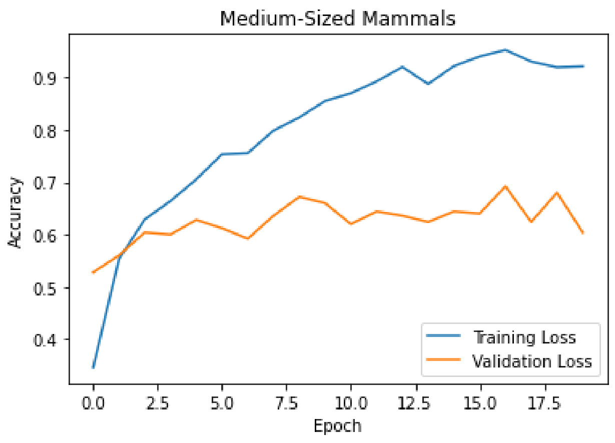
```
Type Prediction: [4.4938195e-02 1.1929894e-01 1.8611830e-01 3.7361035e-04 6.4927095e-01]
Class: Skunk
Type Prediction: [1.7861096e-15 9.6054390e-14 3.5046807e-10 1.2703017e-14 9.9999994e-01]
Class: Skunk
Type Prediction: [7.1494704e-01 1.9784629e-02 2.6255971e-01 2.2403640e-03 4.6812350e-04]
Class: Fox
Type Prediction: [8.9788819e-06 1.9304647e-07 5.3777462e-01 4.6218336e-01 3.2865257e-05]
Class: Possum
Type Prediction: [9.7371121e-05 9.9783796e-01 4.1187907e-04 1.6515916e-03 1.1106924e-06]
Class: Porcupine
Type Prediction: [0.00379293 0.0006203  0.37864023 0.6156579  0.00128867]
Class: Raccoon
Type Prediction: [2.2274126e-06 4.4355493e-05 1.3384093e-03 1.2620403e-02 9.8599452e-01]
Class: Skunk
Type Prediction: [8.0172776e-04 2.5033001e-03 9.9667704e-01 1.7592762e-05 4.1476292e-07]
Class: Possum
Type Prediction: [3.6687718e-04 2.0852682e-01 3.2908058e-01 4.3531302e-01 2.6712719e-02]
Class: Raccoon
Type Prediction: [7.1911413e-06 1.4977946e-04 1.0080236e-01 4.3639637e-05 8.9899701e-01]
Class: Skunk
Type Prediction: [8.8347057e-05 9.9806434e-01 1.8472380e-03 9.0473877e-09 8.7207208e-10]
Class: Porcupine
Type Prediction: [9.9681181e-01 2.6004664e-06 3.1856005e-03 2.3274187e-09 5.7649778e-09]
Class: Fox
Type Prediction: [0.40628737 0.35993618 0.21555257 0.0154846  0.00273922]
Class: Fox
Type Prediction: [1.4577032e-02 2.1382216e-02 9.3971103e-01 2.3893720e-02 4.3598819e-04]
Class: Possum
Type Prediction: [5.50358780e-02 1.03303995e-02 9.19717908e-01 1.48552479e-02
 6.05756286e-05]
Class: Possum
Type Prediction: [8.4271520e-08 1.1790498e-04 7.2269718e-04 9.9768603e-01 1.4732248e-03]
Class: Raccoon
Type Prediction: [1.2256997e-02 8.7505716e-01 1.1266821e-01 1.6357504e-05 1.4004230e-06]
Class: Porcupine
Type Prediction: [0.04118751 0.11712752 0.53311205 0.30773595 0.00083699]
Class: Possum
Type Prediction: [1.4429735e-01 4.9423400e-01 3.6122188e-01 2.4181738e-04 4.8986590e-06]
Class: Porcupine
Type Prediction: [4.0636241e-09 9.9999923e-01 5.5507678e-07 7.1145770e-08 1.5710816e-11]
Class: Porcupine
[[72  2 24  2  0]
 [12 52 29  3  4]
 [ 5 10 79  5  1]
 [ 4  7 32 48  9]
 [ 2  0 12  2 84]]
```

The matrix above shows the how likely the Medium-Sized Mammals appear. Each column matches the *fine class: fox, porcupine, possum, raccoon, skunk.*

Fox = 72% accuracy
Porcupine = 52% accuracy
Possum = 79% accuracy
Racoon = 48% accuracy
Skunk = 84% accuracy

Since the accuracy percentage of each animals per row has greater value, this then shows that the output will likely give the right picture per prediction. The greater the accuracy of each column, the higher the possibility that the right picture will be predicted.

Training Loss and Validation Loss Graph



The training loss and validation loss are commonly used to monitor the performance of machine learning models during training. The training loss is the error calculated on the training dataset during each epoch of the model training, while the validation loss is the error calculated on a separate validation dataset, which is usually a subset of the training data or a completely separate dataset.

In the illustration shown above, the validation loss *(orange line)* shows how well the model fits new data, while the training loss *(blue line)* shows how well it matches training data.

The illustration above shows that the training loss is consistently higher than the validation loss. This usually indicates that the model is not overfitting the training data, and it may be underfitting instead. This means that the model is not complex enough to capture the underlying patterns in the training data, and as a result, it is not able to fit the data well.

There could be several reasons why a model may underfit the data. For instance, the model architecture may be too simple or not deep enough, the learning rate may be too low, or the training data may not be representative of the problem space.

Underfitting is a common problem in machine learning, and it can lead to poor performance and low accuracy. To address underfitting, one could try increasing the complexity of the model by adding more layers or neurons, increasing the number of training epochs, adjusting the learning rate, or collecting more training data to cover a more diverse range of examples. It's important to note that finding the right balance between model complexity and generalization ability is key to building a high-performing and robust machine learning model.

Challenges

Complete newbies. Implementing our ideas into code has proven to be the greatest challenge faced in this exercise. Upon further reading on the dataset in hopes of finding clues to improve our current implementation, data preparation techniques such as data augmentation, reshaping, and normalization were considered but were eventually dropped due to the (relative) difficulty in manipulating the dataset due to much inexperience in the language and the errors introduced after the addition of such. Much more familiarity with the Python programming language and the available functions of the Keras could have significantly improve the existing model.

High accuracy. Due to having limited skills to strategize a more effective CNN model architecture, much difficulty was experienced in raising the accuracy from the maximum yielded accuracy of 71.200009 after 100 epochs using the current architecture.

Relatively limited dataset. The limited dataset size can make it challenging to train models that generalize well to new examples.

Small image size. The images are only 32x32 pixels, which made it harder to distinguish images accurately, even upon visual inspection of the proponents themselves.

Wrong labels. Some of the labels were incorrect, negatively affecting the accuracy of the training results.

Conclusion

Despite the various hurdles encountered throughout the exercise, this has still been a good experience to get acquainted with basic deep learning for image processing using a convolutional neural network (CNN), one among the many subsets of artificial intelligence.

Nowadays, artificial intelligence has played integral roles in various fields in society, notably in automation, prediction, and classification applications. In business, AI can help businesses to better understand their customers' preferences and behaviors by analyzing large datasets, including visual data such as images.

In manufacturing, and logistics, AI models trained on CIFAR-10 and CIFAR-100 can be used to classify images of products, and equipment. For example, AI models can be trained to detect defective products in a manufacturing plant or to identify the contents of a package for efficient sorting in a warehouse.

Another great potential application of image processing is the improvement of the detection and diagnosis of medical abnormalities in medical images such as cancer cells, tumors, and the like, ideally leading to technologies that could detect early signs of life-threatening diseases.

In brief, studying deep learning can equip us with the skills and knowledge to create intelligent systems that can analyze and interpret complex data, enabling you to develop innovative solutions to real-world problems and potentially advance the field of artificial intelligence.

Appendix

```python
import tensorflow as tf
from tensorflow import keras
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Dropout
import matplotlib.pyplot as plt

# Loading data
(train_images, train_labels), (test_images, test_labels) =
keras.datasets.cifar100.load_data(label_mode='coarse')
(fine_train, fine_Trlabels), (fine_test, fine_TsLabels) =
keras.datasets.cifar100.load_data(label_mode='fine')

print('Class: {}'.format(len(np.unique(train_labels))))
a = train_images
b = test_images
coarse_TrLabels = train_labels
coarse_TsLabels = test_labels

idx = []
for i in range(len(coarse_TrLabels)):
  if coarse_TrLabels[i] == 12:
      idx.append(i)

print('Total: {}'.format(len(idx)))
idx = np.array(idx)
train_images, train_labels = fine_train[idx], fine_Trlabels[idx]
print("Shape of image: {}".format(train_images.shape))

idx = []
for i in range(len(coarse_TsLabels)):
    if coarse_TsLabels[i] == 12:
        idx.append(i)

idx = np.array(idx)
test_images, test_labels = fine_test[idx], fine_TsLabels[idx]

#Testing and Changing number labels
for i in range(len(test_labels)):
    if(test_labels[i] == 34):
        test_labels[i] = 0
    if(test_labels[i] == 63):
        test_labels[i] = 1
    if(test_labels[i] == 64):
        test_labels[i] = 2
    if(test_labels[i] == 66):
        test_labels[i] = 3
    if(test_labels[i] == 75):
        test_labels[i] = 4

#Training and Changing number labels
for i in range(len(train_labels)):
    if(train_labels[i] == 34):
        train_labels[i] = 0
    if(train_labels[i] == 63):
        train_labels[i] = 1
    if(train_labels[i] == 64):
        train_labels[i] = 2
    if(train_labels[i] == 66):
        train_labels[i] = 3
    if(train_labels[i] == 75):
        train_labels[i] = 4

#Plotting the Output
plt.figure(figsize=(10,2))
for i in range(5):
  plt.subplot(1,5,i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(test_images[i], cmap=plt.cm.binary)
  plt.xlabel(test_labels[i])


#Printing Unique Train Labels
print(np.unique(train_labels))

# Building the model
model = tf.keras.Sequential()

model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32,
32, 3)))
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(5, activation='softmax'))

model.summary()

# Compiling the model
history= model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

metricInfo = model.fit(train_images, train_labels, epochs=10,
validation_split=0.1)

# Testing the model
print(test_images.shape)
test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)

# Checking the output
from sklearn.metrics import confusion_matrix

classes = ['Fox', 'Porcupine', 'Possum', 'Raccoon', 'Skunk']
prediction = model.predict(test_images)


plt.figure(figsize=(10,10))
for image in range(0, 20):
  i = image
  plt.subplot(5,5,i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  j=i+0
  data_plot = test_images[j]
  plt.imshow(data_plot)
  plt.xlabel(test_labels[j])
plt.show()
```

```python
for i in range(20):
  print("Type Prediction: {}".format(prediction[i]))
  class_idx = np.argmax(prediction[i])
  print("Class: {}".format(classes[class_idx]))

class_idx2 = []
for val in prediction:
  class_idx2.append(np.argmax(val))

cm = confusion_matrix(test_labels, class_idx2)
print(cm)

# Plotting the model
import keras
from matplotlib import pyplot as plt
plt.plot(metricInfo.history['accuracy'])
plt.plot(metricInfo.history['val_accuracy'])
plt.title('Medium-Sized Mammals')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Validation Loss'], loc='lower right')
plt.show()
```