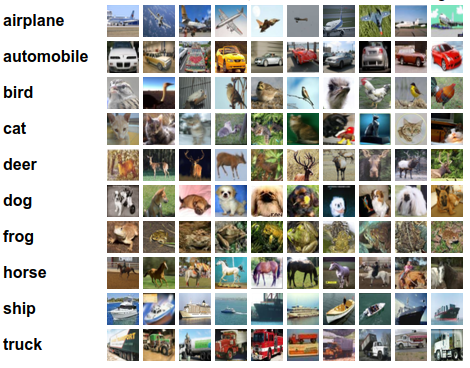
Exercise 6: Playing with Deep NN

Task 1: Create a deep learning model. Train and test it on CIFAR-10. Report on the training and testing accuracy.

In this assignment, we are assigned to train Deep NN model to for CIFAR-10 dataset, which is image dataset. It consists of 60000, 32x32 color images, 10 classes (6000 images each). Examples of the dataset are shown below.



Firstly, include dependencies, load the dataset, and normalize it into [0,1].

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

# Load CIFAR-10 dataset

(train\_images, train\_labels), (test\_images, test\_labels) = datasets.cifar10.load\_data()

train\_images, test\_images = train\_images / 255.0, test\_images / 255.0

The model is a Sequential model, which is a linear stack of layers. It starts with a 2D convolutional layer with 32 filters, each of size 3x3, and ReLU activation, followed by a max pooling layer with a pool size of 2x2. This pattern is repeated with 64 filters in the next two convolutional layers, each followed by a max pooling layer. The model then flattens the 3D output into 1D and adds a dense (fully connected) layer with 64 units and ReLU activation. The final layer is a dense layer with ten units representing the ten classes of the CIFAR-10 dataset. The model is shown below

# Create the model

model = models.Sequential([

layers.Conv2D(32, (3, 3), *activation*='relu', *input\_shape*=(32, 32, 3)),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), *activation*='relu'),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), *activation*='relu'),

layers.Flatten(),

layers.Dense(64, *activation*='relu'),

layers.Dense(10)

])

The optimizer I used is “adam”, specified below.

model.compile(*optimizer*='adam',

*loss*=tf.keras.losses.SparseCategoricalCrossentropy(*from\_logits*=True),

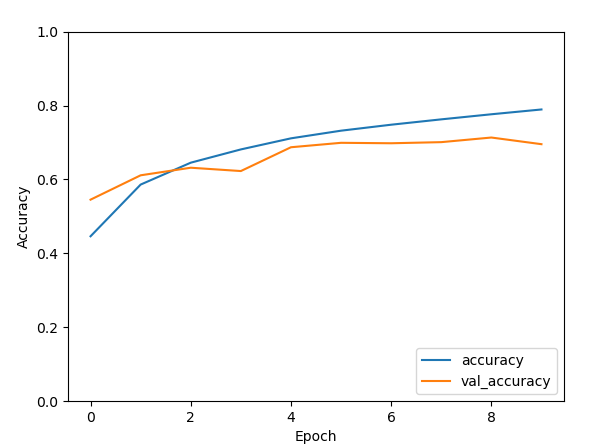
*metrics*=['accuracy'])

Then, train it using “fit” function. The number of epochs is set to 10.

history = model.fit(train\_images, train\_labels, *epochs*=10,

*validation\_data*=(test\_images, test\_labels))

The accuracy of train set (blue), and validation set (orange) are shown in the figure below. The validation accuracy at the last epoch is around 70%.



Lastly, I saved the model for the second task in h5 format.

model.save('cifar10\_model.h5')

Task 2: Report the accuracy of the same model learned on CIFAR in a slightly modified CIFAR-10 dataset. This dataset should introduce some sort of noise to the images.

First, noise is added to the test images by creating function “add\_noise.”

*def* add\_noise(*images*, *noise\_factor*=0.1):

noisy\_images = *images* + *noise\_factor* \* np.random.randn(\**images*.shape)

noisy\_images = np.clip(noisy\_images, 0., 1.)

return noisy\_images

test\_images\_noisy = add\_noise(test\_images)

Load the model that I have saved in the previous task and applied the test imaged with noise.

# Load the trained model

model = models.load\_model('cifar10\_model.h5')

# Evaluate the model on noisy data

test\_loss\_noisy, test\_acc\_noisy = model.evaluate(test\_images\_noisy, test\_labels, *verbose*=2)

My model's test accuracy when I used a noisy dataset was 51%, which is very low. We can improve accuracy by using the existing pre-trained model.

Task 3: Train on this new dataset with noise and report the results (accuracy and testing accuracy).

In this task, I used the same model and optimizer as the first task. Also, same number of epoch. But the dataset is changed by adding noise using “add\_noise” function.

train\_images\_noisy = add\_noise(train\_images)

test\_images\_noisy = add\_noise(test\_images)

# Compile the model

model\_noisy.compile(*optimizer*='adam',

*loss*=tf.keras.losses.SparseCategoricalCrossentropy(*from\_logits*=True),

*metrics*=['accuracy'])

# Train the model on noisy data

history\_noisy = model\_noisy.fit(train\_images\_noisy, train\_labels, *epochs*=10,

*validation\_data*=(test\_images\_noisy, test\_labels))

# Evaluate the model on noisy data

test\_loss\_noisy, test\_acc\_noisy = model\_noisy.evaluate(test\_images\_noisy, test\_labels, *verbose*=2)

After training, the accuracy of validation set of noisy images is 66.66%.

