Week 2 Assignment: CIFAR-10 Autoencoder

For this week, you will create a convolutional autoencoder for the CIFAR10 dataset. You are free to choose the architecture of your autoencoder provided that the output image has the same dimensions as the input image.

After training, your model should meet loss and accuracy requirements when evaluated with the test dataset. You will then download the model and upload it in the classroom for grading.

Let's begin!

Important: *This colab notebook has read-only access so you won't be able to save your changes. If you want to save your work periodically, please click* `File -> Save a Copy in Drive` *to create a copy in your account, then work from there.*

Imports

```
try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass

import tensorflow as tf
import tensorflow_datasets as tfds

from keras.models import Sequential
```

Load and prepare the dataset

The CIFAR 10 dataset already has train and test splits and you can use those in this exercise. Here are the general steps:

- Load the train/test split from TFDS. Set `as_supervised` to `True` so it will be convenient to use the preprocessing function we provided.
- Normalize the pixel values to the range [0,1], then return `image, image` pairs for training instead of `image, label`. This is because you will check if the output image is successfully regenerated after going through your autoencoder.
- Shuffle and batch the train set. Batch the test set (no need to shuffle).

```
# preprocessing function
def map_image(image, label):
  image = tf.cast(image, dtype=tf.float32)
  image = image / 255.0

  return image, image # dataset label is not used. replaced with the same image input.

# parameters
```

```
BATCH_SIZE = 128
SHUFFLE_BUFFER_SIZE = 1024


### START CODE HERE (Replace instances of `None` with your code) ###

# use tfds.load() to fetch the 'train' split of CIFAR-10
train_dataset = tfds.load('cifar10', as_supervised=True, split="train")

# preprocess the dataset with the `map_image()` function above
train_dataset = train_dataset.map(map_image)

# shuffle and batch the dataset
train_dataset = train_dataset.shuffle(SHUFFLE_BUFFER_SIZE).batch(BATCH_SIZE) #.repeat()


# use tfds.load() to fetch the 'test' split of CIFAR-10
test_dataset = tfds.load('cifar10', as_supervised=True, split="test")

# preprocess the dataset with the `map_image()` function above
test_dataset = test_dataset.map(map_image)

# batch the dataset
test_dataset = test_dataset.batch(BATCH_SIZE) #.repeat()

### END CODE HERE ###
```

Build the Model

Create the autoencoder model. As shown in the lectures, you will want to downsample the image in the encoder layers then upsample it in the decoder path. Note that the output layer should be the same dimensions as the original image. Your input images will have the shape `(32, 32, 3)`. If you deviate from this, your model may not be recognized by the grader and may fail.

We included a few hints to use the Sequential API below but feel free to remove it and use the Functional API just like in the ungraded labs if you're more comfortable with it. Another reason to use the latter is if you want to visualize the encoder output. As shown in the ungraded labs, it will be easier to indicate multiple outputs with the Functional API. That is not required for this assignment though so you can just stack layers sequentially if you want a simpler solution.

```
# suggested layers to use. feel free to add or remove as you see fit.
from keras.layers import Conv2D, MaxPooling2D, UpSampling2D

# use the Sequential API (you can remove if you want to use the Functional API)
model = Sequential()

### START CODE HERE ###
# use `model.add()` to add layers (if using the Sequential API)
model.add(tf.keras.Input(shape=(32,32,3)))
model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(filters=128, kernel_size=(3,3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(3,3), activation='relu', padding='same'))
model.add(Conv2D(filters=128, kernel_size=(3,3), activation='relu', padding='same'))
```

```
model.add(UpSampling2D(size=(2,2)))
model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same'))
model.add(UpSampling2D(size=(2,2)))
model.add(Conv2D(filters=3, kernel_size=(3,3), activation='sigmoid', padding='same'))
### END CODE HERE ###

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 32, 32, 64)        1792
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 8, 8, 256)         295168
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 8, 8, 128)         295040
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 16, 16, 64)        73792
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 32, 32, 3)         1731
=================================================================
Total params: 741,379
Trainable params: 741,379
Non-trainable params: 0
_________________________________________________________________
```

Configure training parameters

We have already provided the optimizer, metrics, and loss in the code below.

```
# Please do not change the model.compile() parameters
model.compile(optimizer='adam', metrics=['accuracy'], loss='mean_squared_error')
```

Training

You can now use model.fit() to train your model. You will pass in the `train_dataset` and you are free to configure the other parameters. As with any training, you should see the loss generally going down and the accuracy going up with each epoch. If not, please revisit the previous sections to find possible bugs.

Note: If you get a `dataset length is infinite` error. Please check how you defined `train_dataset`. You might have included a method that repeats the dataset indefinitely.

```
# parameters (feel free to change this)
train_steps = len(train_dataset) // BATCH_SIZE
val_steps = len(test_dataset) // BATCH_SIZE
```

```
### START CODE HERE ###
model.fit(train_dataset, steps_per_epoch=train_steps, validation_data=test_dataset, validation_steps
### END CODE HERE ###
```

```
Epoch 1/100
3/3 [==============================] - 0s 106ms/step - loss: 0.0041 - accuracy: 0.8047
Epoch 2/100
3/3 [==============================] - 0s 95ms/step - loss: 0.0040 - accuracy: 0.8017
Epoch 3/100
3/3 [==============================] - 0s 94ms/step - loss: 0.0040 - accuracy: 0.8052
Epoch 4/100
3/3 [==============================] - 0s 89ms/step - loss: 0.0043 - accuracy: 0.8091
Epoch 5/100
3/3 [==============================] - 0s 88ms/step - loss: 0.0037 - accuracy: 0.8084
Epoch 6/100
3/3 [==============================] - 0s 89ms/step - loss: 0.0039 - accuracy: 0.8063
Epoch 7/100
3/3 [==============================] - 0s 82ms/step - loss: 0.0040 - accuracy: 0.8038
Epoch 8/100
3/3 [==============================] - 0s 87ms/step - loss: 0.0049 - accuracy: 0.7981
Epoch 9/100
3/3 [==============================] - 0s 82ms/step - loss: 0.0043 - accuracy: 0.8111
Epoch 10/100
3/3 [==============================] - 0s 83ms/step - loss: 0.0049 - accuracy: 0.7807
Epoch 11/100
3/3 [==============================] - 0s 83ms/step - loss: 0.0045 - accuracy: 0.8037
Epoch 12/100
3/3 [==============================] - 0s 81ms/step - loss: 0.0042 - accuracy: 0.7848
Epoch 13/100
3/3 [==============================] - 0s 89ms/step - loss: 0.0043 - accuracy: 0.7939
Epoch 14/100
3/3 [==============================] - 0s 81ms/step - loss: 0.0040 - accuracy: 0.7959
Epoch 15/100
3/3 [==============================] - 0s 87ms/step - loss: 0.0040 - accuracy: 0.8054
Epoch 16/100
3/3 [==============================] - 0s 79ms/step - loss: 0.0041 - accuracy: 0.7918
Epoch 17/100
3/3 [==============================] - 0s 77ms/step - loss: 0.0041 - accuracy: 0.7688
Epoch 18/100
3/3 [==============================] - 0s 80ms/step - loss: 0.0042 - accuracy: 0.7892
Epoch 19/100
3/3 [==============================] - 0s 76ms/step - loss: 0.0039 - accuracy: 0.7936
Epoch 20/100
3/3 [==============================] - 0s 80ms/step - loss: 0.0039 - accuracy: 0.7960
Epoch 21/100
3/3 [==============================] - 0s 81ms/step - loss: 0.0036 - accuracy: 0.8019
Epoch 22/100
3/3 [==============================] - 0s 83ms/step - loss: 0.0037 - accuracy: 0.8117
Epoch 23/100
3/3 [==============================] - 0s 81ms/step - loss: 0.0039 - accuracy: 0.8079
Epoch 24/100
3/3 [==============================] - 0s 80ms/step - loss: 0.0039 - accuracy: 0.8092
Epoch 25/100
3/3 [==============================] - 0s 82ms/step - loss: 0.0036 - accuracy: 0.8128
Epoch 26/100
3/3 [==============================] - 0s 90ms/step - loss: 0.0036 - accuracy: 0.8050
Epoch 27/100
3/3 [==============================] - 0s 81ms/step - loss: 0.0037 - accuracy: 0.8134
Epoch 28/100
3/3 [==============================] - 0s 84ms/step - loss: 0.0038 - accuracy: 0.8042
Epoch 29/100
3/3 [==============================] - 0s 82ms/step - loss: 0.0038 - accuracy: 0.8166
Epoch 30/100
```

Model evaluation

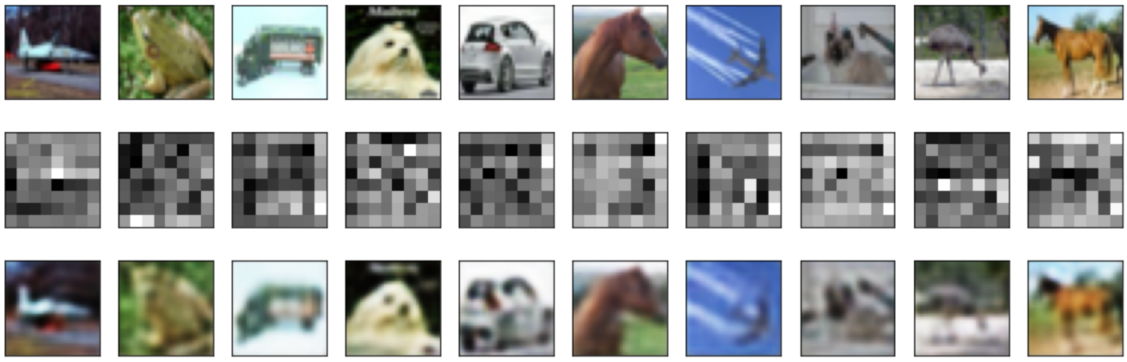
You can use this code to test your model locally before uploading to the grader. To pass, your model needs to satisfy these two requirements:

- loss must be less than 0.01
- accuracy must be greater than 0.6

```
result = model.evaluate(test_dataset, steps=10)
```

```
    10/10 [==============================] - 1s 33ms/step - loss: 0.0033 - accuracy: 0.8164
```

If you did some visualization like in the ungraded labs, then you might see something like the gallery below. This part is not required.



```
def display_one_row(disp_images, offset, shape=(28, 28)):
  '''Display sample outputs in one row.'''
  for idx, test_image in enumerate(disp_images):
    plt.subplot(3, 10, offset + idx + 1)
    plt.xticks([])
    plt.yticks([])
    test_image = np.reshape(test_image, shape)
    plt.imshow(test_image, cmap='gray')


def display_results(disp_input_images, disp_encoded, disp_predicted, enc_shape=(8,4)):
  '''Displays the input, encoded, and decoded output values.'''
  plt.figure(figsize=(15, 5))
  display_one_row(disp_input_images, 0, shape=(32,32,3))
  display_one_row(disp_encoded, 10, shape=enc_shape)
  display_one_row(disp_predicted, 20, shape=(32,32,3))


import numpy as np
import matplotlib.pylab as plt

# take 1 batch of the dataset
test_dataset = test_dataset.take(1)

# take the input images and put them in a list
output_samples = []
for input_image, image in tfds.as_numpy(test_dataset):
      output_samples = input_image
```

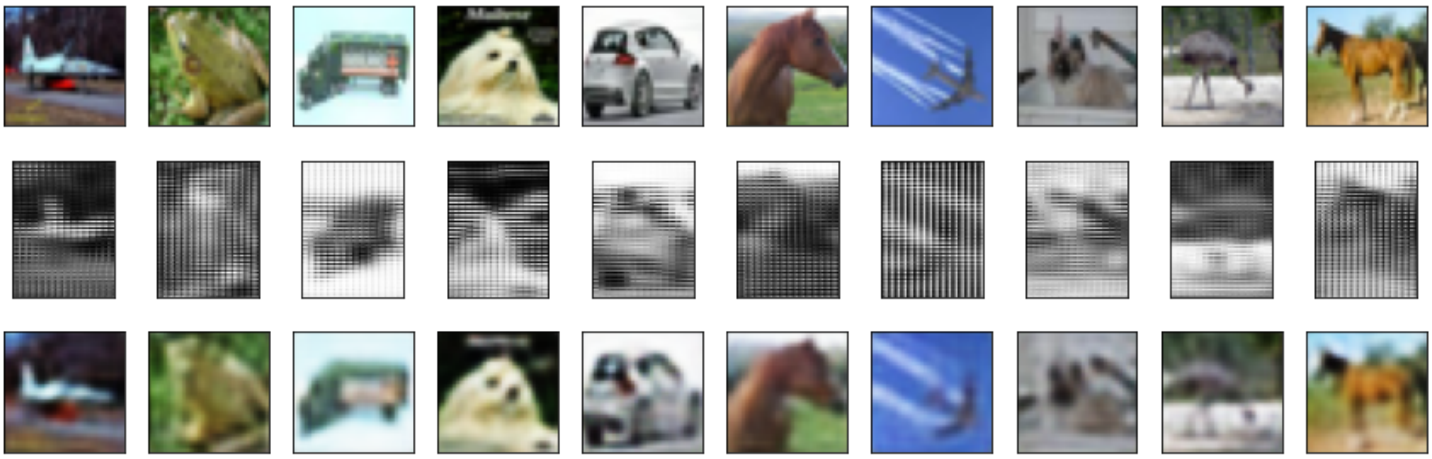
```
# pick 10 indices
idxs = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# prepare test samples as a batch of 10 images
conv_output_samples = np.array(output_samples[idxs])
conv_output_samples = np.reshape(conv_output_samples, (10, 32, 32, 3))

# get the encoder ouput
encoded = model.predict(conv_output_samples)

# get a prediction for some values in the dataset
predicted = model.predict(conv_output_samples)

# display the samples, encodings and decoded values!
display_results(conv_output_samples, encoded, predicted, enc_shape=(64,48))
```

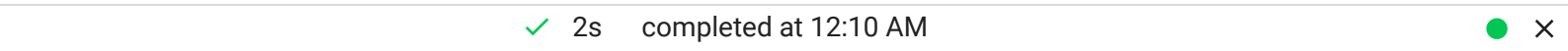


Save your model

Once you are satisfied with the results, you can now save your model. Please download it from the Files window on the left and go back to the Submission portal in Coursera for grading.

```
model.save('mymodel.h5')
```

Congratulations on completing this week's assignment!


2s
completed at 12:10 AM