CSC 578 NN&DL Spring 2020

HW#6 - Image Classification using CNN

This code is slightly modified from the TensorFlow tutorial "Convolutional Neural Network (CNN)" for the purpose first downloads the data, the CIFAR-10 dataset and partitions the training set into training and validation sets. The network and trains the network with the training set. Finally the code evaluates the network performance using th

Note that there are **three places** in the code, indicated with ****IMPORTANT****, where you choose the syntax that w TensorFlow (1 or 2) installed on your platform.

Import TensorFlow

****IMPORTANT (1)** ** Uncomment either import line(s) for the version of TensorFlow (TF1 or TF2) of your platforr

```
import matplotlib.pyplot as plt

import tensorflow as tf

## For TF version 2 (just one line)
from tensorflow.keras import datasets, layers, models

## For TF version 1 (need both lines)
#from tensorflow import keras
#from keras import datasets, layers, models


print(tf.__version__) # 5/2020 nt: check the TF version!
```

```
2.2.0
```

Download and prepare the CIFAR10 dataset

The CIFAR10 dataset contains 60,000 color images in 10 classes, with 6,000 images in each class. The dataset is training images and 10,000 testing images.

```
# Download the data from the repository site.
(train_all_images, train_all_labels), (test_images, test_labels) = datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 6s 0us/step
```

```
# !! DO NOT REMOVE THIS LINE !!
# Delete test_labels (by making it an empty list) so that we don't accidentally
# use it in the code.
test_labels = []

# Then split the training set ('train_all') into two subsets: train and
# validation.  After that, we have 3 subsets: train, validation and test.
from sklearn.model_selection import train_test_split
```

```
from sklearn.model_selection import train_test_split

# 80% train, 20% validation, and by using stratefied sampling.
train_images, valid_images, train_labels, valid_labels \
  = train_test_split(train_all_images, train_all_labels,
                     stratify=train_all_labels, test_size=0.2)


# Normalize pixel values of images to be between 0 and 1
train_images, valid_images, test_images \
  = train_images / 255.0, valid_images / 255.0, test_images / 255.0


train_labels
```

```
array([[6],
       [6],
       [4],
       ...,
       [9],
       [9],
       [5]], dtype=uint8)
```

```
valid_labels
```

```
array([[8],
       [9],
       [1],
       ...,
       [7],
       [7],
       [2]], dtype=uint8)
```

Verify the data

To verify that the dataset looks correct, plot the first 10 images from the training set and display the class name b

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(10):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

Create a convolutional Network

As input, a CNN takes tensors of shape (image_height, image_width, color_channels), ignoring the batch size, wh
(R,G,B). The format of CIFAR images is 32 * 32 pixels, so the input shape is (32, 32, 3). The output layer has 10 no
number of categories of the images.
In this code, the activation function of the output layer is specified to be softmax for the purpose of aligning the t
(TF1 and TF2; in particular to make TF2 compatible with TF1's 'sparse_categorical_crossentropy' loss function).

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax')) # 5/2020 nt: as noted above
```

Verify the model.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

Compile the model

****IMPORTANT (2) **** Uncomment either loss function for the version of TensorFlow (TF1 or TF2) of your platform

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), # For TF2
              #loss='sparse_categorical_crossentropy', # For TF1
              metrics=['accuracy'])
```

Train the model

```
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(valid_images, valid_labels)) # 5/2020 nt: use validation set
```

```
Epoch 1/10
1250/1250 [==============================] - 5s 4ms/step - loss: 1.6045 - accuracy: 0.4134 - val_lc
Epoch 2/10
1250/1250 [==============================] - 4s 3ms/step - loss: 1.2238 - accuracy: 0.5639 - val_lc
Epoch 3/10
1250/1250 [==============================] - 4s 3ms/step - loss: 1.0721 - accuracy: 0.6202 - val_lc
Epoch 4/10
1250/1250 [==============================] - 4s 3ms/step - loss: 0.9650 - accuracy: 0.6608 - val_lc
Epoch 5/10
1250/1250 [==============================] - 4s 4ms/step - loss: 0.8838 - accuracy: 0.6885 - val_lc
Epoch 6/10
1250/1250 [==============================] - 4s 4ms/step - loss: 0.8217 - accuracy: 0.7084 - val_lc
Epoch 7/10
1250/1250 [==============================] - 4s 3ms/step - loss: 0.7708 - accuracy: 0.7282 - val_lc
Epoch 8/10
1250/1250 [==============================] - 4s 3ms/step - loss: 0.7181 - accuracy: 0.7476 - val_lc
Epoch 9/10
1250/1250 [==============================] - 4s 3ms/step - loss: 0.6759 - accuracy: 0.7613 - val_lc
Epoch 10/10
1250/1250 [==============================] - 4s 3ms/step - loss: 0.6393 - accuracy: 0.7747 - val_lc
```

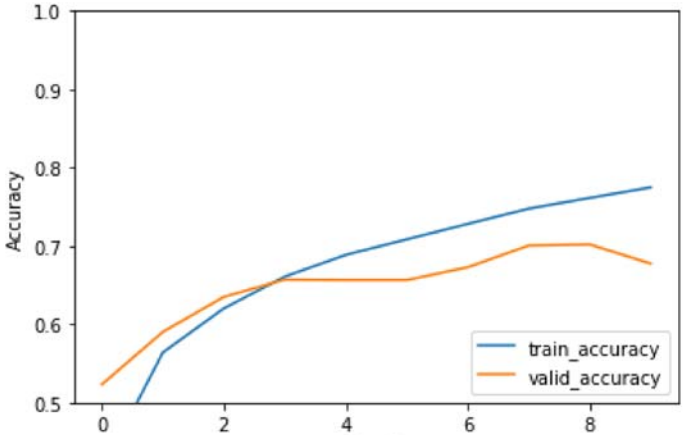
Evaluate the model

****IMPORTANT (3) **** Uncomment either syntax for the version of TensorFlow (TF1 or TF2) of your platform.

```
plt.plot(history.history['accuracy'], label='train_accuracy') # For TF2
#plt.plot(history.history['acc'], label='train_accuracy') # For TF1
plt.plot(history.history['val_accuracy'], label = 'valid_accuracy') # For TF2
#plt.plot(history.history['val_acc'], label = 'valid_accuracy') # For TF1
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

# Evaluate the learned model with validation set
valid_loss, valid_acc = model.evaluate(valid_images, valid_labels, verbose=2) # 5/2020 nt: use validatic
print ("valid_accuracy=%s, valid_loss=%s" % (valid_acc, valid_loss))
```

```
313/313 - 1s - loss: 0.9725 - accuracy: 0.6774
valid_accuracy=0.6773999929428101, valid_loss=0.9725407361984253
```



TO DO -- Make Predictions

Apply the learned network to '**test_images**' and generate predictions.

Look at the code from HW#4 or other tutorial code for the syntax. You generate predictions and create/write a KA