Robert Kaszubski CSC 578 - Section 710 Homework 7 – Convolution Neural Networks

CSC 578 NN&DL Fall 2021

HW7: Image Classification using a CNN

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

Note that there are **three places** in the code, indicated with **IMPORTANT**, where you have to choose the syntax that works for the version of 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 platform.

```
import matplotlib.pyplot as plt
import tensorflow as tf
print(tf.__version__)  # check the TF version!
```

```
2.7.0
```

```
# 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
```

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 (pre-)divided into 50,000 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 [==============================] - 3s 0us/step
170508288/170498071 [==============================] - 3s 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

# 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],
       [2],
       [8],
       ...,
       [2],
       [4],
       [5]], dtype=uint8)
```

```
valid_labels
```

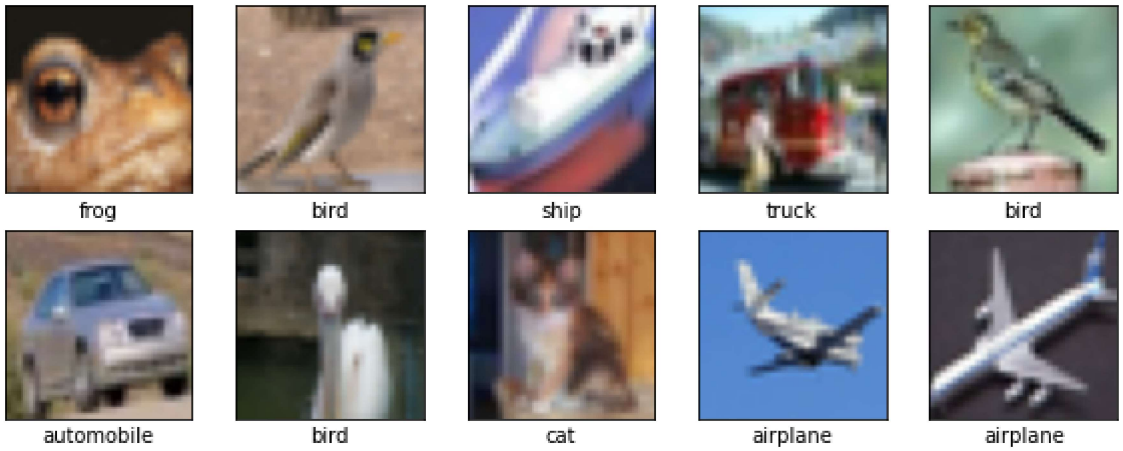
```
array([[1],
       [8],
       [9],
       ...,
       [4],
       [2],
       [1]], dtype=uint8)
```

Verify the data

To verify that the dataset looks correct, plot the first 10 images from the training set and display the class name below each image.

```
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

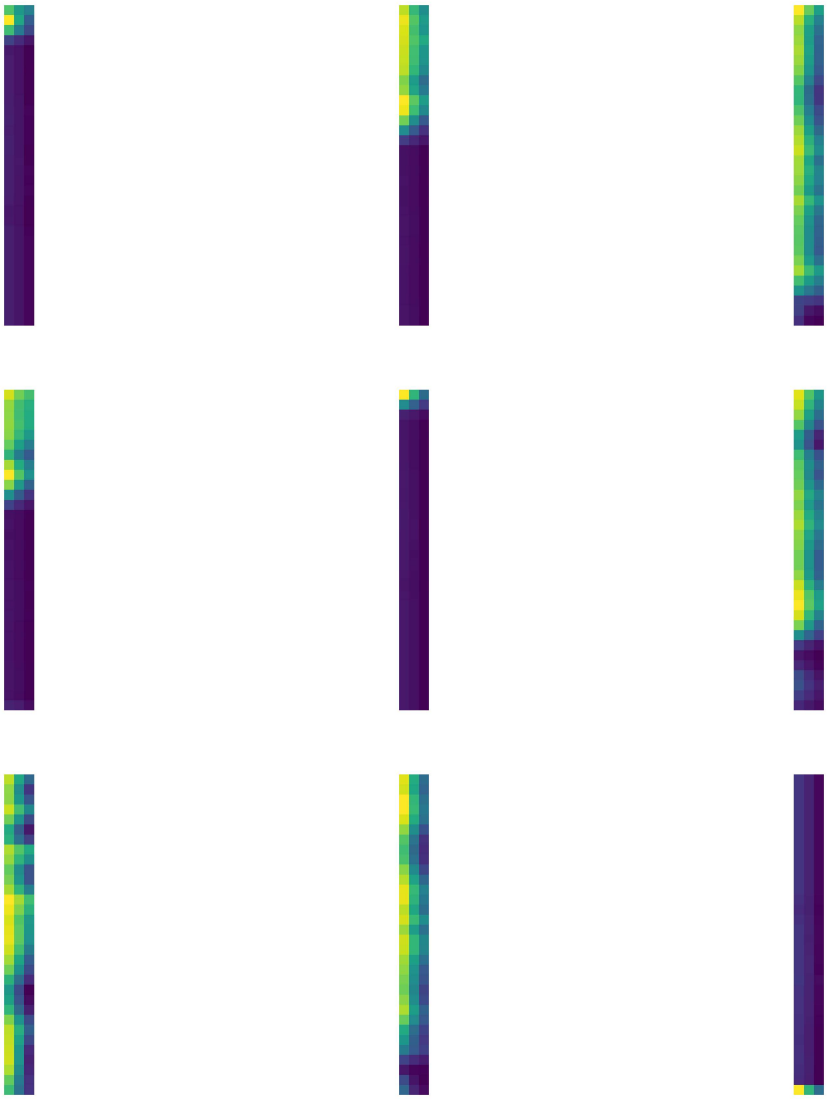
As input, a CNN takes tensors of shape (image_height, image_width, color_channels), ignoring the batch size, where color_channels refers to (R,G,B). The format of CIFAR images is 32 * 32 pixels, so the input shape is (32, 32, 3). The output layer has 10 nodes, corresponding to the number of categories of the images.

In this code, the activation function of the output layer is specified to be softmax for the purpose of aligning the two versions of TensorFlow (TF1 and TF2; in particular to make TF2 compatible with TF1's 'sparse_categorical_crossentropy' loss function).

Attempted to add Data Augmentation

```
'''
data_augmentation = tf.keras.Sequential([
  layers.RandomFlip("horizontal_and_vertical"),
  layers.RandomRotation(0.2),
```

```
])
'''
'''
plt.figure(figsize=(10, 10))
for i in range(9):
  augmented_image = data_augmentation(train_images[0])
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(augmented_image[0])
  plt.axis("off")
'''
```



Best Overall Model - Highest Kaggle Score

Added callback to save best weights from training

```
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=50, restore_best_wei
```

```
model = models.Sequential()
```

```
model.add(layers.Conv2D(64, (2, 2), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(64, (2, 2), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.5))
model.add(layers.Conv2D(128, (2, 2), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (4, 4), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.BatchNormalization())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))  # As noted above
```

Verify the model

```
model.summary()
```

```
Model: "sequential_56"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_191 (Conv2D)         (None, 31, 31, 64)        832

 batch_normalization_17 (Bat  (None, 31, 31, 64)       256
 chNormalization)

 conv2d_192 (Conv2D)         (None, 30, 30, 64)        16448

 max_pooling2d_92 (MaxPoolin  (None, 15, 15, 64)       0
 g2D)

 dropout_98 (Dropout)        (None, 15, 15, 64)        0

 conv2d_193 (Conv2D)         (None, 14, 14, 128)       32896

 max_pooling2d_93 (MaxPoolin  (None, 7, 7, 128)        0
 g2D)

 conv2d_194 (Conv2D)         (None, 4, 4, 128)         262272

 flatten_46 (Flatten)        (None, 2048)              0

 dropout_99 (Dropout)        (None, 2048)              0

 batch_normalization_18 (Bat  (None, 2048)             8192
 chNormalization)

 dense_138 (Dense)           (None, 128)               262272

 dense_139 (Dense)           (None, 64)                8256
```

```
 dense_140 (Dense)           (None, 10)                650

=================================================================
Total params: 592,074
Trainable params: 587,850
Non-trainable params: 4,224
_________________________________________________________________
```

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 TF
              #loss='sparse_categorical_crossentropy', # For TF1
              metrics=['accuracy'])
```

Train the model

```
history = model.fit(train_images, train_labels, epochs=100, callbacks=[callback],
                    validation_data=(valid_images, valid_labels))
```

```
Epoch 38/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6842 - accuracy: 0
Epoch 39/100
1250/1250 [==============================] - 7s 5ms/step - loss: 0.6950 - accuracy: 0
Epoch 40/100
1250/1250 [==============================] - 7s 5ms/step - loss: 0.6806 - accuracy: 0
Epoch 41/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6969 - accuracy: 0
Epoch 42/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6909 - accuracy: 0
Epoch 43/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.7125 - accuracy: 0
Epoch 44/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6896 - accuracy: 0
Epoch 45/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6797 - accuracy: 0
Epoch 46/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6762 - accuracy: 0
Epoch 47/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6555 - accuracy: 0
Epoch 48/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6442 - accuracy: 0
Epoch 49/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6639 - accuracy: 0
Epoch 50/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6528 - accuracy: 0
```

```
Epoch 51/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6412 - accuracy: 0
Epoch 52/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6364 - accuracy: 0
Epoch 53/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6297 - accuracy: 0
Epoch 54/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6338 - accuracy: 0
Epoch 55/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6312 - accuracy: 0
Epoch 56/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6960 - accuracy: 0
Epoch 57/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6536 - accuracy: 0
Epoch 58/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6684 - accuracy: 0
Epoch 59/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6436 - accuracy: 0
Epoch 60/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6278 - accuracy: 0
Epoch 61/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6317 - accuracy: 0
Epoch 62/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6189 - accuracy: 0
Epoch 63/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6077 - accuracy: 0
Epoch 64/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6112 - accuracy: 0
Epoch 65/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6137 - accuracy: 0
Epoch 66/100
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6096 - accuracy: 0
Epoch 67/100
```

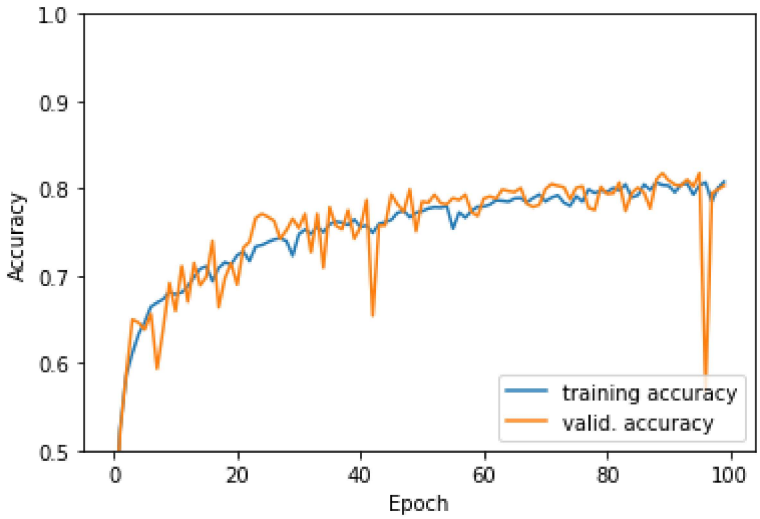
Evaluate the model

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

```
plt.plot(history.history['accuracy'], label='training accuracy') # For TF2
#plt.plot(history.history['acc'], label='training 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)
print ("valid_accuracy=%s, valid_loss=%s" % (valid_acc, valid_loss))
```

```
313/313 - 1s - loss: 0.5685 - accuracy: 0.8033 - 787ms/epoch - 3ms/step
valid_accuracy=0.8033000230789185, valid_loss=0.5685405731201172
```



TO DO (by you): 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 should generate predictions and create/write a KAGGLE submission file.

```
predictions = model.predict(test_images)
```

```
predictions[0]
```

```
array([3.4954617e-04, 5.3431617e-05, 3.3582603e-03, 8.3703762e-01,
       1.4302431e-03, 1.2573761e-01, 2.9787323e-02, 6.4202416e-04,
       1.5763802e-03, 2.7440994e-05], dtype=float32)
```

```
#output list
output = []
output.append(['id','cat0','cat1','cat2','cat3','cat4','cat5','cat6','cat7','cat8','cat9'])

#add id to each row
count = 1
for row in predictions:
  prediction = [count] + row.tolist()
  output.append(prediction)
  count += 1
```

```
output[1]
```

```
[1,
 0.00034954617149196565,
 5.3431616834132e-05,
```

```
       0.0033582602627575397,
       0.8370376229286194,
       0.0014302431372925 64,
       0.12573760747909546,
       0.029787322506308556,
       0.0006420241552405059,
       0.0015763802221044898,
       2.7440994017524645e-05]


import csv

#export output to csv
with open('predictions.csv', 'w') as f:
  writer = csv.writer(f)
  for row in output:
    writer.writerow(row)
```

Interesting Model found while experimenting

```
#interesting model
'''
model = models.Sequential()
model.add(layers.Conv2D(64, (2, 2), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(64, (2, 2), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.15))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(128, (2, 2), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(128, (4, 4), activation='relu'))
model.add(layers.Flatten())
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.15))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))  # As noted above
'''
```

```
'\nmodel = models.Sequential()\nmodel.add(layers.Conv2D(64, (2, 2), activation='relu',
input_shape=(32, 32, 3)))\nmodel.add(layers.Conv2D(64, (2, 2), activation='relu'))\nmod
el.add(layers.MaxPooling2D((2, 2)))\nmodel.add(layers.Dropout(0.15))\nmodel.add(layers.
BatchNormalization())\nmodel.add(layers.Conv2D(128, (2, 2), activation='relu'))\nmodel.
add(layers.MaxPooling2D((2, 2)))\nmodel.add(layers.BatchNormalization())\nmodel.add(lay
ers.Conv2D(128, (4, 4), activation='relu'))\nmodel.add(layers.Flatten())\nmodel.add(lay
```

Other Model when training with only 10 epochs

```
#other interesting model
'''
```

```
model = models.Sequential()
model.add(layers.Conv2D(64, (2, 2), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(64, (2, 2), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.15))
model.add(layers.Conv2D(128, (2, 2), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (4, 4), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dropout(0.15))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))  # As noted above
'''
```

```
'\nmodel = models.Sequential()\nmodel.add(layers.Conv2D(64, (2, 2), activation='relu',
input_shape=(32, 32, 3)))\nmodel.add(layers.Conv2D(64, (2, 2), activation='relu'))\nmod
el.add(layers.MaxPooling2D((2, 2)))\nmodel.add(layers.Dropout(0.15))\nmodel.add(layers.
Conv2D(128, (2, 2), activation='relu'))\nmodel.add(layers.MaxPooling2D((2, 2)))\nmodel.
add(layers.Conv2D(128, (4, 4), activation='relu'))\nmodel.add(layers.Flatten())\nmodel.
```