Image Classification using Deep Learning

Kayla Neal for Pearce Lab

Imports

Using TensorFlow and Keras to build, train, test, and evaluate model.

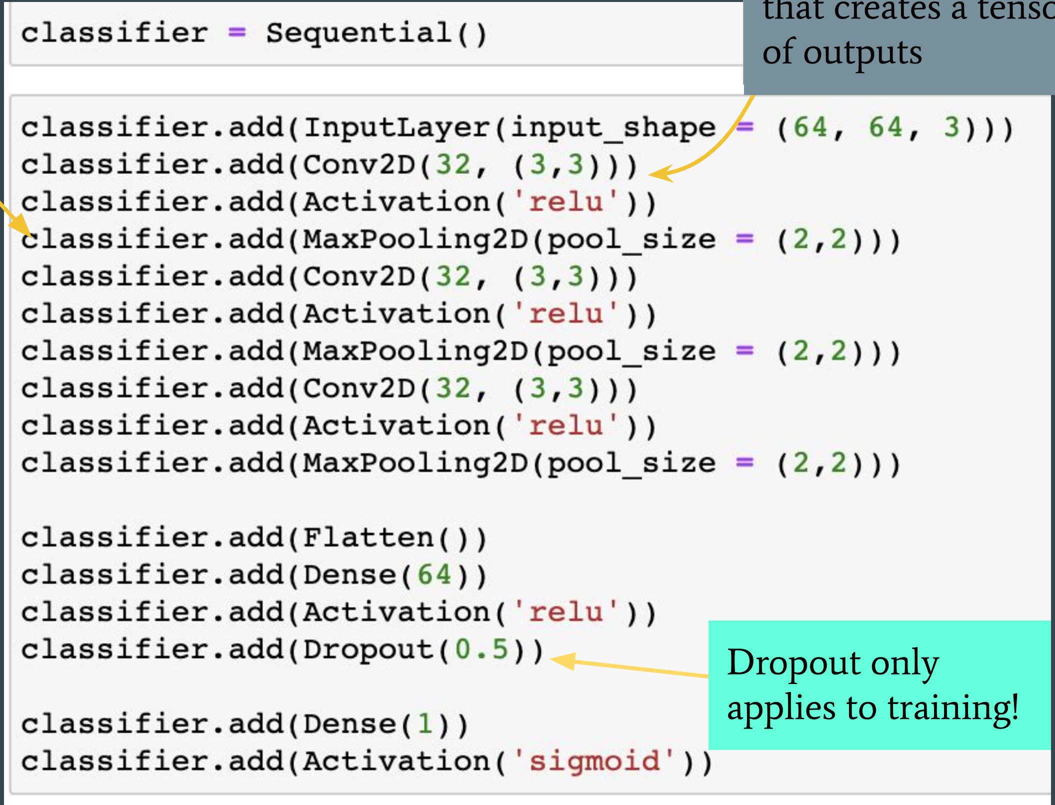
```
import tensorflow as tf

from tensorflow.keras.models
import Sequential

from tensorflow.keras.layers import Conv2D,
Activation, MaxPooling2D,
Dense, Flatten, Dropout, InputLayer

from tensorflow.keras.preprocessing.image
import ImageDataGenerator

import numpy as np

from IPython.display import display

from PIL import Image
```

MaxPooling downsamples the input along the designated dimensions

Building a Classifier

- The classifier is built under the Sequential() model system of tf.keras. The important criteria regarding this decision is the linear stack a Sequential model uses to build and connect layers.
- tf.keras.layers api describes the different layers in detail with what to use each for and what the arguments are.

Conv2D is a convolutional layer that creates a tensor of outputs



```
classifier = Sequential()

classifier.add(InputLayer(input_shape = (64, 64, 3)))
classifier.add(Conv2D(32, (3,3)))
classifier.add(Activation('relu'))
classifier.add(MaxPooling2D(pool_size = (2,2)))
classifier.add(Conv2D(32, (3,3)))
classifier.add(Activation('relu'))
classifier.add(MaxPooling2D(pool_size = (2,2)))
classifier.add(Conv2D(32, (3,3)))
classifier.add(Activation('relu'))
classifier.add(MaxPooling2D(pool_size = (2,2)))

classifier.add(Flatten())
classifier.add(Dense(64))
classifier.add(Activation('relu'))
classifier.add(Dropout(0.5))

classifier.add(Dense(1))
classifier.add(Activation('sigmoid'))
```

Dropout only applies to training!

Using ImageDataGenerator to Create Datasets

- The ImageDataGenerator from keras is a quick, easy, and efficient way to create datasets.
- To prevent overfitting, the training generator has augmentation parameters included to diversify the set. This isn't necessary for the test generator.
- This classifier is being used to distinguish between two types categories -> class mode is binary

```
# Step 1: initialize generators

trainGen = ImageDataGenerator( rescale = 1./255,
                               shear_range = 0.2,
                               zoom_range = 0.2,
                               horizontal_flip = True )

testGen = ImageDataGenerator( rescale = 1./255 )

# Step 2: create datasets

trainSet = trainGen.flow_from_directory( dir_name,
                                         target_size = (64,64),
                                         batch_size = 32,
                                         class_mode = ‘binary’ )

testSet = testGen.flow_from_directory( dir_name,
                                       target_szie = (64,64),
                                       batch_size = 32,
                                       class_mode = ‘binary’ )
```

Compile and Fit Model

- compile the model:

```
classifier.compile( optimizer = ‘rmsprop’,
                    loss = ‘binary_crossentropy’,
                    metrics = [‘accuracy’] )
```

- when distinguishing between 2 classes, binary crossentropy is the preferred loss function
  - for multiple: categorical crossentropy

- there are many different keras optimizers - the most commonly used is Adam, here we used rmsprop.

- train the model:

```
classifier.fit( trainSet, steps_per_epoch = 625,
        epochs = 30, validation_data = testSet,
        validation_steps = 5000 )
```

- to save the model, use mode_namel.save(
                                  ‘model_name.h5’)