Existing Keras Dataset - DNNs/CNNs

Helper Class

```
class myCallback(keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get("accuracy") > 0.99):
      print("\nReached 99% Training Accuracy so stop!")
      self.model.stop_training = True
```

Hyperparameters

model units and layers
batch_size
optimizer
epochs

Get data

```
mnist = keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

DNNs

Preprocess

```
x_train, x_test = x_train/255.0, x_test/255.0
```

Model

```
model = keras.models.Sequential([
  keras.layers.Flatten(),
  keras.layers.Dense(512, activation="relu"),
  keras.layers.Dense(10, activation="softmax")
])

model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])

callbacks = myCallback()
model.fit(x_train, y_train, epochs=10, callbacks=[callbacks])

model.evaluate(x_test, y_test)
```

```
model.save("ex2.h5")
```

Predict

```
mnist = keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

model = keras.models.load_model("models/ex2_recode.h5")
#model.summary()
x_test = x_test/255.0
print(model.evaluate(x_test, y_test))
print(model.predict(x_test[:1]))
print(np.argmax(model.predict(x_test[:1])))
print(y_test[0])
```

CNNs

Preprocess

```
# reshape(number of samples, height, width, number of channels)
x_train = x_train.reshape(60000, 28, 28, 1)
x_train = x_train / 255.0

x_test = x_test.reshape(10000, 28, 28, 1)
x_test = x_test / 255.0
```

Model

```
model = keras.models.Sequential([
  keras.layers.Conv2D(256, (3,3), input_shape=(28, 28, 1)),
  keras.layers.MaxPooling2D(2,2),

  keras.layers.Flatten(),
  keras.layers.Dense(256, activation="relu"),
  keras.layers.Dense(128, activation="relu"),
  keras.layers.Dense(64, activation="relu"),
  keras.layers.Dense(32, activation="relu"),
  keras.layers.Dense(10, activation="softmax")
])

model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])

callbacks = myCallback()
model.fit(x_train, y_train, epochs=20, callbacks=[callbacks], batch_size = 128)

model.evaluate(x_test, y_test)
```

```
model.save("ex3_recode.h5")
```

Predict

```
mnist = keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

model = keras.models.load_model("models/ex3_recode.h5")

x_test = (x_test.reshape(10000, 28, 28, 1 ))
x_test = x_test/255.0

preds = model.predict(x_test[:3])
print(preds)
print(np.argmax(i) for i in preds)
print(y_test[:3])
```

Local Data - Binary and Multi Classification

Import Packages

```
import os
import zipfile
import random
from shutil import copyfile

from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator

%matplotlib inline
import matplotlib.pyplot as plt
```

Hyperparameters

```
model units and layers
batch_size
steps_per_epoch and validation_steps
optimizer
epochs
```

Download, extract data, appropriate train-test directories

Source data - to split

Download and extract data to source folders

```
!wget --no-check-certificate \
    "https://download.microsoft.com/download/3/E/1/"
    + "3E1C3F21-ECDB-4869-8368-6DEBA77B919F/kagglecatsanddogs_3367a.zip" \
    -O "/tmp/cats-and-dogs.zip"

local_zip = '/tmp/cats-and-dogs.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp')
zip_ref.close()

# print number of files
print(len(os.listdir('/tmp/PetImages/Cat/')))
print(len(os.listdir('/tmp/PetImages/Dog/')))
```

Create directories for train-test dataset

```
try:
  # we have to build this from parent to child directories
  os.mkdir('/tmp/cats-v-dogs')

  os.mkdir('/tmp/cats-v-dogs/training')
  os.mkdir('/tmp/cats-v-dogs/testing')

  os.mkdir('/tmp/cats-v-dogs/training/cats')
  os.mkdir('/tmp/cats-v-dogs/training/dogs')

  os.mkdir('/tmp/cats-v-dogs/testing/cats')
  os.mkdir('/tmp/cats-v-dogs/testing/dogs')

except OSError:
  pass
```

Test-train data - already Test split

Download data

```
!wget --no-check-certificate \
    "https://storage.googleapis.com/"
    + "laurencemoroney-blog.appspot.com/rps.zip" \
    -O /tmp/rps.zip

!wget --no-check-certificate \
    "https://storage.googleapis.com/"
    + "laurencemoroney-blog.appspot.com/rps-test-set.zip" \
    -O /tmp/rps-test-set.zip
```

Extract data

```
local_zip = '/tmp/rps.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp/')
zip_ref.close()

local_zip = '/tmp/rps-test-set.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp/')
zip_ref.close()
```

Split Source data to Train-Test data

```
def split_data(SOURCE_DIR, TRAINING_DIR, TESTING_DIR, SPLIT_SIZE):
  fnames_raw = os.listdir(SOURCE_DIR)
  fnames_shuffled = random.sample(fnames_raw, len(fnames_raw))

  # clean
  # fget uncorrupted data only
  fnames = []
  for fname in fnames_shuffled:
    if os.path.getsize(SOURCE_DIR +fname) !=0:
      fnames.append(fname)

  # split fnames
  train_fnames = fnames[:int(len(fnames)*SPLIT_SIZE)]
  test_fnames = fnames[int(len(fnames)*SPLIT_SIZE):]

  # transfer via copying files
  for fname in train_fnames:
    copyfile(SOURCE_DIR + fname, TRAINING_DIR + fname)

  for fname in test_fnames:
    copyfile(SOURCE_DIR + fname, TESTING_DIR + fname)

split_size = 0.9

CAT_SOURCE_DIR = '/tmp/PetImages/Cat/'
CAT_TRAINING_DIR = '/tmp/cats-v-dogs/training/cats/'
CAT_TESTING_DIR = '/tmp/cats-v-dogs/testing/cats/'

DOG_SOURCE_DIR = '/tmp/PetImages/Dog/'
DOG_TRAINING_DIR = '/tmp/cats-v-dogs/training/dogs/'
DOG_TESTING_DIR = '/tmp/cats-v-dogs/testing/dogs/'

split_data(CAT_SOURCE_DIR, CAT_TRAINING_DIR, CAT_TESTING_DIR, split_size)
split_data(DOG_SOURCE_DIR, DOG_TRAINING_DIR, DOG_TESTING_DIR, split_size)
```

```
# print total number of images per folder
print(len(os.listdir('/tmp/cats-v-dogs/training/cats/')))
print(len(os.listdir('/tmp/cats-v-dogs/training/dogs/')))
print(len(os.listdir('/tmp/cats-v-dogs/testing/cats/')))
print(len(os.listdir('/tmp/cats-v-dogs/testing/dogs/')))
```

Preprocessing - ImageDataGenerator

Binary Classification

```python
TRAIN_DIR = "/tmp/cats-v-dogs/training/"
train_datagen = ImageDataGenerator(
    rescale = 1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
train_generator = train_datagen.flow_from_directory(
    TRAIN_DIR,
    batch_size = 10,
    target_size = (300, 300),
    class_mode = "binary"
)

VALID_DIR = "/tmp/cats-v-dogs/testing/"
valid_datagen = ImageDataGenerator(rescale=1./255)
valid_generator = valid_datagen.flow_from_directory(
    VALID_DIR,
    batch_size = 10,
    target_size = (300, 300),
    class_mode = "binary"
)
```

Multi Classification

```python
import tensorflow as tf
import keras_preprocessing
from keras_preprocessing import image
from keras_preprocessing.image import ImageDataGenerator

TRAINING_DIR = "/tmp/rps/"
training_datagen = ImageDataGenerator(
      rescale = 1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
train_generator = training_datagen.flow_from_directory(
  TRAINING_DIR,
  target_size=(150,150),
  class_mode='categorical',
  batch_size=126
)

VALIDATION_DIR = "/tmp/rps-test-set/"
validation_datagen = ImageDataGenerator(rescale = 1./255)
validation_generator = validation_datagen.flow_from_directory(
  VALIDATION_DIR,
  target_size=(150,150),
  class_mode='categorical',
  batch_size=126
)

```

Model

Binary Classification

```
model = keras.models.Sequential([
  keras.layers.Conv2D(16, (3,3), activation="relu", input_shape=(300,300, 3)),
  keras.layers.MaxPooling2D(2,2),
  keras.layers.Conv2D(32, (3,3), activation="relu"),
  keras.layers.MaxPooling2D(2,2),
  keras.layers.Conv2D(64, (3,3), activation="relu"),
  keras.layers.MaxPooling2D(2,2),
  keras.layers.Conv2D(64, (3,3), activation="relu"),
  keras.layers.MaxPooling2D(2,2),
  keras.layers.Conv2D(64, (3,3), activation="relu"),
  keras.layers.MaxPooling2D(2,2),

  keras.layers.Flatten(),
  keras.layers.Dense(512, activation = "relu"),
  keras.layers.Dense(128, activation="relu"),
  keras.layers.Dense(128, activation="relu"),
  keras.layers.Dense(64, activation="relu"),
  keras.layers.Dense(1, activation="sigmoid"),
])

model.compile(loss = "binary_crossentropy",
              optimizer = RMSprop(lr=0.001),
              metrics=["accuracy"])

history = model.fit(
    train_generator,
    epochs = 15,
    verbose =2,
    validation_data = valid_generator
)
```

Multi Classification

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])


model.summary()

model.compile(loss = 'categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

history = model.fit(
    train_generator,
    epochs=25,
    steps_per_epoch=20, # = total train samples/ train batch_size
    validation_data = validation_generator,
    verbose = 1,
    validation_steps=3 # = total valid samples/ valid batch_size
)
```

Plot Loss and Accuracy

```
train_loss = history.history['loss']
val_loss = history.history['val_loss']

train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

epochs = range(len(train_acc))

plt.plot(epochs, train_loss, color="b", label="train_loss")
plt.plot(epochs, val_loss, color="r", label="valid_loss")
plt.title('Training and validation accuracy')
plt.figure()

plt.plot(epochs, train_acc, color="b",label="train_acc")
plt.plot(epochs, val_acc, color="r",label="valid_acc")
plt.title('Training and validation loss')

plt.show()
```

Predict

```
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=10)
  print(fn)
  print(classes)
```

Transfer Learning (Course 2 ex3)

Import Packages

```
import os
import zipfile

from tensorflow.keras import layers
from tensorflow.keras import Model

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import RMSprop
```

Download, extract data, appropriate train-test directories

Source data - to split

Download and extract data to source folders

```
!wget --no-check-certificate \
    "https://download.microsoft.com/download/3/E/1/"
    + "3E1C3F21-ECDB-4869-8368-6DEBA77B919F/kagglecatsanddogs_3367a.zip" \
    -O "/tmp/cats-and-dogs.zip"

local_zip = '/tmp/cats-and-dogs.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp')
zip_ref.close()

# print number of files
print(len(os.listdir('/tmp/PetImages/Cat/')))
print(len(os.listdir('/tmp/PetImages/Dog/')))
```

Create directories for train-test dataset

```
try:
  # we have to build this from parent to child directories
  os.mkdir('/tmp/cats-v-dogs')

  os.mkdir('/tmp/cats-v-dogs/training')
  os.mkdir('/tmp/cats-v-dogs/testing')

  os.mkdir('/tmp/cats-v-dogs/training/cats')
  os.mkdir('/tmp/cats-v-dogs/training/dogs')

  os.mkdir('/tmp/cats-v-dogs/testing/cats')
  os.mkdir('/tmp/cats-v-dogs/testing/dogs')

except OSError:
  pass
```

Test-train data - already Test split

Download data

```
!wget --no-check-certificate \
    "https://storage.googleapis.com/"
    + "laurencemoroney-blog.appspot.com/rps.zip" \
    -O /tmp/rps.zip

!wget --no-check-certificate \
    "https://storage.googleapis.com/"
    + "laurencemoroney-blog.appspot.com/rps-test-set.zip" \
    -O /tmp/rps-test-set.zip
```

Extract data

```
local_zip = '/tmp/rps.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp/')
zip_ref.close()

local_zip = '/tmp/rps-test-set.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp/')
zip_ref.close()
```

Preprocess

```python
# Add our data-augmentation parameters to ImageDataGenerator
train_datagen = ImageDataGenerator(
    rescale = 1./255.,
    rotation_range = 40,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True)
# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size = 20,
                                                    class_mode = 'binary',
                                                    target_size = (150, 150))

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator( rescale = 1.0/255. )
# Flow validation images in batches of 20 using test_datagen generator
validation_generator =  test_datagen.flow_from_directory( validation_dir,
                                                          batch_size  = 20,
                                                          class_mode  = 'binary',
                                                          target_size = (150, 150))
```

Transfer learning key parts

Get data

```
!wget --no-check-certificate \
    https://storage.googleapis.com/mledu-datasets/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5 \
    -O /tmp/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
```

Setup and load pretrained model

```
from tensorflow.keras.applications.inception_v3 import InceptionV3

local_weights_file = '/tmp/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'

pre_trained_model = InceptionV3(input_shape = (150, 150, 3), # desired input shape for the data
                                include_top = False, # ignore fully convolutional layers and get straight to convs
                                weights = None # do not use built-in weights but the snapshot downloaded
)

pre_trained_model.load_weights(local_weights_file)
```

Freeze layers

```
# iterate through layers and lock them
# for them not to be trainable
for layer in pre_trained_model.layers:
  layer.trainable = False

# print summary - caution: it's looooooongg
pre_trained_model.summary()
```

Pick where to cutoff the pretrained model

```
# get the last layer output
last_layer = pre_trained_model.get_layer('mixed7')
print('last layer output shape: ', last_layer.output_shape)
last_output = last_layer.output
```

Model (normal parts but using Layers API instead of Sequential)

```
# build model from that last layer
# this is just a different way of using the Layers API

from tensorflow.keras.optimizers import RMSprop

x = layers.Flatten()(last_output)
x = layers.Dense(1024, activation="relu")(x)
x = layers.Dropout(0.2)(x)
x = layers.Dense(1, activation="sigmoid")(x)

model = Model(pre_trained_model.input, x) # build the model

model.compile(loss = "binary_crossentropy",
              optimizer = RMSprop(lr=0.0001),
              metrics = ['accuracy'])

history = model.fit(
            train_generator,
            validation_data = validation_generator,
            steps_per_epoch = 100,
            epochs = 3,
            validation_steps = 50,
            verbose = 2)
```

```
model.summary() # to check if the layers changed
```

Plot Loss and Accuracy

```
train_loss = history.history['loss']
val_loss = history.history['val_loss']

train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

epochs = range(len(train_acc))

plt.plot(epochs, train_loss, color="b", label="train_loss")
plt.plot(epochs, val_loss, color="r", label="valid_loss")
plt.title('Training and validation accuracy')
plt.figure()

plt.plot(epochs, train_acc, color="b",label="train_acc")
plt.plot(epochs, val_acc, color="r",label="valid_acc")
plt.title('Training and validation loss')

plt.show()
```

Predict

```
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=10)
  print(fn)
  print(classes)
```

Loading from CSV (course 2 - ex4)

Get Data and output arrays for labels and images

```python
def get_data(filename):
  images=[]
  labels=[]
  with open(filename) as files:
    csvfile = csv.reader(files, delimiter=',')
    for row in csvfile:
      label, image = np.split(row, [1])

      if label == 'label': # first rows are column headers
        continue

      # must be an array and values should be not 'str' but int
      label = np.array(label).astype(int)
      image = np.array(image).astype(int)

      image = np.reshape(image, (28,28))

      labels.append(label)
      images.append(image)

  # should return two arrays
  labels = np.array(labels).ravel() # ravel - to return contiguous flattened array (27455,) and not (27455, 1)
  images = np.array(images)
  return images, labels

training_images, training_labels = get_data('/content/drive/MyDrive/study/tf-specialization/course2_convnets/sign_mnist_train.csv')
testing_images, testing_labels = get_data('/content/drive/MyDrive/study/tf-specialization/course2_convnets/sign_mnist_test.csv')

# Keep these
print(training_images.shape)
print(training_labels.shape)
print(testing_images.shape)
print(testing_labels.shape)

# Their output should be:
# (27455, 28, 28)
# (27455,)
# (7172, 28, 28)
# (7172,)
```

Preprocess

```
# In this section you will have to add another dimension to the data
# So, for example, if your array is (10000, 28, 28)
# You will need to make it (10000, 28, 28, 1)
# Hint: np.expand_dims

training_images = np.expand_dims(training_images, axis=3)
testing_images = np.expand_dims(testing_images, axis=3)

# Create an ImageDataGenerator and do Image Augmentation
train_datagen = ImageDataGenerator(
    rescale = 1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
train_generator = train_datagen.flow(
    training_images,
    training_labels,
    batch_size = 289
)


validation_datagen = ImageDataGenerator(rescale=1./255)
validation_generator = validation_datagen.flow(
    testing_images,
    testing_labels,
    batch_size = 163
)

# Keep These
print(training_images.shape)
print(testing_images.shape)

# Their output should be:
# (27455, 28, 28, 1)
# (7172, 28, 28, 1)
```

Model

```python
# Define the model
# Use no more than 2 Conv2D and 2 MaxPooling2D
model = tf.keras.models.Sequential([
    keras.layers.Conv2D(32, (3,3), activation="relu", input_shape=(28,28,1)),
    keras.layers.MaxPool2D(2,2),
    keras.layers.Conv2D(64, (3,3), activation="relu"),
    keras.layers.MaxPool2D(2,2),

    keras.layers.Flatten(),
    keras.layers.Dense(512, activation="relu"),
    keras.layers.Dense(len(set(training_labels.tolist())) + 1, activation="softmax")
])
# len(set(training_labels.tolist())) + 1 = 25 classes
# len(set(training_labels.tolist())) this returns 24 but the classes are 0-24 without 9
# so the range of the classes is still 25 since it's range is from 0 to 24

# Compile Model.
model.compile( loss = "sparse_categorical_crossentropy", optimizer="rmsprop", metrics=["accuracy"])

# Train the Model
history = model.fit(
    train_generator,
    epochs=10,
    steps_per_epoch = len(training_labels)/289,
    validation_data = validation_generator,
    validation_steps = len(testing_labels)/163
)

model.evaluate(testing_images, testing_labels)

# The output from model.evaluate should be close to:
#[6.92426086682151, 0.56609035]
```

Plot Loss and Accuracy

```
train_loss = history.history['loss']
val_loss = history.history['val_loss']

train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

epochs = range(len(train_acc))

plt.plot(epochs, train_loss, color="b", label="train_loss")
plt.plot(epochs, val_loss, color="r", label="valid_loss")
plt.title('Training and validation accuracy')
plt.figure()

plt.plot(epochs, train_acc, color="b",label="train_acc")
plt.plot(epochs, val_acc, color="r",label="valid_acc")
plt.title('Training and validation loss')

plt.show()
```

Predict

```
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=10)
  print(fn)
  print(classes)
```