COE59410- Generative Deep Learning

Homework 1

> Imran A. Zualkernan

The purpose of this homework is to learn how to use the GPU machines to run large models. In addition, build familiarity with the Keras framework and the associated tools.

Deliverables

You need to upload the following on ilearn (2 items)

1. The Jupyter notebook in its original format.

2. A PDF of the Jupyter notebook for grading.

Please do not upload a zipped file. Upload each file separately. Each question is worth 25 points.

- Q1. Load and run the large_scale_processing v1.1 Jupyter notebook on the GPU machine and show how you can use tensorboard to monitor the runs remotely on your local machine.

- Q2. Modify the model in large_scale_processing v1.1 so that rather than a CNN, the model is a fully connected feedforward neural network. Fine tune the model to show your best results. Report and discuss all the results that are necessary to determine the goodness of your best model.

> Hint: Use the Reshape Layer in Keras.

- Q3. Use the following two call-backs on your best fully connected model and determine if you are able to improve the results. Clearly explain why or why not.

    i. LearningRateScheduler
    ii. ReduceLROnPlateau

- Q4. Use the Keras Hypertune and Random optimizers (https://keras-team.github.io/keras-tuner/) to determine if you can improve the model by varying the number of layers, neurons in each layer and the learning rate.

i. Plot the precision vs. recall of the best 20 models in one figure.
ii. Show a complete evaluation of the top two models.

Group 2

- Eman ,
- Huangjin Zhou, b00080932
- Mueez ,

1

```
# Useful links
# https://www.hostinger.com/tutorials/ssh/basic-ssh-commands
```

2

```
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 11557652930811666046
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 7491581376
locality {
  bus_id: 1
  links {
  }
}
incarnation: 5153159438992320913
physical_device_desc: "device: 0, name: Quadro RTX 4000, pci bus id: 0000:01:00.0, compute capab
]
```

3

```
import tensorflow as tf
# print(tf.config.list_physical_devices('GPU'))
tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.list_physical_devices(device_type=None)
tf.test.is_gpu_available()
print(tf.test.is_built_with_cuda())
```

```
WARNING:tensorflow:From <ipython-input-3-78f31cea5abc>:5: is_gpu_available (from tensorflow.pytho
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
True
```

4

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
from IPython.display import Image, display
import random
import math
import keras
from keras.preprocessing.text import Tokenizer
from keras.models import Model, Sequential
```

```
from keras.utils import plot_model
from keras.layers import Reshape, Input, Dense, Dropout, Flatten, Activation,Concatenate
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D
from keras.optimizers import Adam
from keras import backend, models
#import tensorflow_addons as tfa
import tensorflow as tf
print(tf.__version__)

# need to add these for the GPU
config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.compat.v1.Session(config=config)
```

```
2.4.1
```

5

```
# import the image generator
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

6

```
#Setting the parameters for training

# batch size and image width to use
batch_size=128
width=100

# all the data directories
train_dir='train/'
test_dir='test/'
valid_dir='valid/'

# the number of epochs
num_epochs=10

# creating an image generator that will feed the data from
# each of the directories

# we use scaling transformation in this generator
generator=ImageDataGenerator(rescale=1./255)

# we specify the size of the input and batch size
# size of the input is necessary because the image
# needs to be rescaled for the neural network

train_data=generator.flow_from_directory(train_dir, target_size=(width,width),batch_size=batch_s
valid_data=generator.flow_from_directory(valid_dir, target_size=(width,width),batch_size=batch_s
test_data=generator.flow_from_directory(test_dir, target_size=(width,width),batch_size=batch_siz

# the number of steps per epoch is samples/batch size
# we need to use these numbers later

train_steps_per_epoch=math.ceil(train_data.samples/batch_size)
valid_steps_per_epoch=math.ceil(valid_data.samples/batch_size)
test_steps_per_epoch=math.ceil(test_data.samples/batch_size)
print(train_steps_per_epoch)
print(valid_steps_per_epoch)
print(test_steps_per_epoch)
```

```
Found 35215 images belonging to 250 classes.
Found 1250 images belonging to 250 classes.
Found 1250 images belonging to 250 classes.
276
10
10
```

7

```
# the actual model should go here
hidden_units = 256*2*2*2*2*2
dropout = 0.1
num_labels = train_data.num_classes

model = Sequential()
model.add(Reshape((-1,), input_shape=(width, width, 3)))
model.add(Dense(hidden_units, activation='relu'))
model.add(Dropout(dropout))
model.add(Dense(hidden_units/2, activation='relu'))
model.add(Dropout(dropout))
model.add(Dense(hidden_units/4, activation='relu'))
model.add(Dropout(dropout))
model.add(Dense(num_labels, activation='softmax'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
reshape (Reshape)            (None, 30000)             0         
_________________________________________________________________
dense (Dense)                (None, 8192)              245768192 
_________________________________________________________________
dropout (Dropout)            (None, 8192)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              33558528  
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 2048)              8390656   
_________________________________________________________________
dropout_2 (Dropout)          (None, 2048)              0         
_________________________________________________________________
dense_3 (Dense)              (None, 250)               512250    
=================================================================
Total params: 288,229,626
Trainable params: 288,229,626
Non-trainable params: 0
_________________________________________________________________
```

8

```
# Compile the model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

9

```
# see if the model is good.
print(model)
```

```
<tensorflow.python.keras.engine.sequential.Sequential object at 0x7fd3183dc890>
```

10

```
from tensorflow.keras.callbacks import LearningRateScheduler, ReduceLROnPlateau

def lr_schedule(epoch):
    """Learning rate scheduler - called every epoch"""
    lr = 1e-3
    fold = int(epoch / 10) + 1
    lr /=  fold

    return lr
```

```
lr_scheduler = LearningRateScheduler(lr_schedule)

lr_reducer = ReduceLROnPlateau(factor=np.sqrt(0.1),
                               cooldown=0,
                               patience=5,
                               min_lr=0.5e-6)
```

Q1. Load and run the large_scale_processing v1.1 Jupyter notebook on the GPU machine and show how you can use tensorboard to monitor the runs remotely on your local machine.

```
$ jupyter notebook --port 9999 --NotebookApp.allow_remote_access=True
```

```
[I 20:42:04.633 NotebookApp] Serving notebooks from local directory: /home/group2/
[I 20:42:04.633 NotebookApp] Jupyter Notebook 6.2.0 is running at:
[I 20:42:04.633 NotebookApp] http://localhost:9999/?token=7c9d27bf2c24e9e66eee1da6
[I 20:42:04.633 NotebookApp]  or http://127.0.0.1:9999/?token=7c9d27bf2c24e9e66eee
[I 20:42:04.633 NotebookApp] Use Control-C to stop this server and shut down all k
[W 20:42:04.635 NotebookApp] No web browser found: could not locate runnable brows
[C 20:42:04.635 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///home/group2/.local/share/jupyter/runtime/nbserver-28383-open.html
    Or copy and paste one of these URLs:
        http://localhost:9999/?token=7c9d27bf2c24e9e66eee1da6f5b51d4bd2b8669393e49
     or http://127.0.0.1:9999/?token=7c9d27bf2c24e9e66eee1da6f5b51d4bd2b8669393e49
```

```
$ tensorboard --logdir logs/fit --port=8888
```

```
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --
TensorBoard 2.4.0 at http://localhost:8888/ (Press CTRL+C to quit)
```

11

```
from tensorflow.keras.callbacks import TensorBoard
tensorboard = TensorBoard(log_dir='logs/fit')

print(valid_steps_per_epoch)
num_epochs = 20

callbacks = [lr_reducer, lr_scheduler, tensorboard]
history=model.fit(train_data,
                  steps_per_epoch =train_steps_per_epoch,
                  validation_data=valid_data,
                  epochs=num_epochs,
                  validation_steps=valid_steps_per_epoch, callbacks=callbacks)
```

```
10
Epoch 1/100
276/276 [==============================] - 28s 100ms/step - loss: 17.9556 - accuracy: 0.0076 - va
Epoch 2/100
```

```
276/276 [==============================] - 28s 100ms/step - loss: 5.0856 - accuracy: 0.0239 - va
Epoch 3/100
276/276 [==============================] - 27s 99ms/step - loss: 4.8207 - accuracy: 0.0412 - val_
Epoch 4/100
276/276 [==============================] - 28s 100ms/step - loss: 4.6110 - accuracy: 0.0584 - va
Epoch 5/100
276/276 [==============================] - 27s 99ms/step - loss: 4.4302 - accuracy: 0.0761 - val_
Epoch 6/100
276/276 [==============================] - 28s 100ms/step - loss: 4.2868 - accuracy: 0.0929 - va
Epoch 7/100
276/276 [==============================] - 27s 99ms/step - loss: 4.1800 - accuracy: 0.1043 - val_
Epoch 8/100
276/276 [==============================] - 28s 100ms/step - loss: 4.1198 - accuracy: 0.1146 - va
Epoch 9/100
276/276 [==============================] - 28s 100ms/step - loss: 4.0230 - accuracy: 0.1205 - va
Epoch 10/100
276/276 [==============================] - 28s 100ms/step - loss: 3.9548 - accuracy: 0.1318 - va
Epoch 11/100
276/276 [==============================] - 28s 99ms/step - loss: 3.7698 - accuracy: 0.1620 - val_
Epoch 12/100
276/276 [==============================] - 28s 100ms/step - loss: 3.7020 - accuracy: 0.1675 - va
Epoch 13/100
276/276 [==============================] - 28s 100ms/step - loss: 3.6439 - accuracy: 0.1771 - va
Epoch 14/100
276/276 [==============================] - 28s 100ms/step - loss: 3.6025 - accuracy: 0.1799 - va
Epoch 15/100
276/276 [==============================] - 27s 99ms/step - loss: 3.5496 - accuracy: 0.1902 - val_
Epoch 16/100
276/276 [==============================] - 28s 100ms/step - loss: 3.5025 - accuracy: 0.2007 - va
Epoch 17/100
276/276 [==============================] - 27s 99ms/step - loss: 3.4508 - accuracy: 0.2055 - val_
Epoch 18/100
276/276 [==============================] - 27s 99ms/step - loss: 3.4350 - accuracy: 0.2077 - val_
Epoch 19/100
276/276 [==============================] - 28s 100ms/step - loss: 3.3848 - accuracy: 0.2167 - va
Epoch 20/100
276/276 [==============================] - 28s 100ms/step - loss: 3.3431 - accuracy: 0.2215 - va
Epoch 21/100
276/276 [==============================] - 28s 99ms/step - loss: 3.2387 - accuracy: 0.2357 - val_
Epoch 22/100
276/276 [==============================] - 28s 100ms/step - loss: 3.1623 - accuracy: 0.2508 - va
Epoch 23/100
276/276 [==============================] - 28s 101ms/step - loss: 3.1490 - accuracy: 0.2579 - va
Epoch 24/100
276/276 [==============================] - 28s 100ms/step - loss: 3.0807 - accuracy: 0.2633 - va
Epoch 25/100
276/276 [==============================] - 28s 100ms/step - loss: 3.0757 - accuracy: 0.2644 - va
Epoch 26/100
276/276 [==============================] - 27s 99ms/step - loss: 3.0393 - accuracy: 0.2738 - val_
Epoch 27/100
276/276 [==============================] - 27s 99ms/step - loss: 3.0046 - accuracy: 0.2759 - val_
Epoch 28/100
276/276 [==============================] - 28s 101ms/step - loss: 2.9574 - accuracy: 0.2868 - va
Epoch 29/100
276/276 [==============================] - 28s 101ms/step - loss: 2.9143 - accuracy: 0.2944 - va
Epoch 30/100
276/276 [==============================] - 28s 100ms/step - loss: 2.8867 - accuracy: 0.3022 - va
Epoch 31/100
276/276 [==============================] - 27s 99ms/step - loss: 2.8026 - accuracy: 0.3138 - val_
Epoch 32/100
276/276 [==============================] - 28s 101ms/step - loss: 2.7576 - accuracy: 0.3221 - va
Epoch 33/100
276/276 [==============================] - 28s 100ms/step - loss: 2.7267 - accuracy: 0.3302 - va
Epoch 34/100
276/276 [==============================] - 28s 100ms/step - loss: 2.7007 - accuracy: 0.3363 - va
```

```
Epoch 35/100
276/276 [==============================] - 28s 100ms/step - loss: 2.6681 - accuracy: 0.3413 - va
Epoch 36/100
276/276 [==============================] - 28s 100ms/step - loss: 2.6279 - accuracy: 0.3470 - va
Epoch 37/100
276/276 [==============================] - 28s 100ms/step - loss: 2.6259 - accuracy: 0.3531 - va
Epoch 38/100
276/276 [==============================] - 27s 99ms/step - loss: 2.5915 - accuracy: 0.3535 - val_
Epoch 39/100
276/276 [==============================] - 28s 100ms/step - loss: 2.5432 - accuracy: 0.3658 - va
Epoch 40/100
276/276 [==============================] - 28s 100ms/step - loss: 2.4871 - accuracy: 0.3771 - va
Epoch 41/100
276/276 [==============================] - 28s 100ms/step - loss: 2.4600 - accuracy: 0.3798 - va
Epoch 42/100
276/276 [==============================] - 28s 100ms/step - loss: 2.4297 - accuracy: 0.3875 - va
Epoch 43/100
276/276 [==============================] - 28s 100ms/step - loss: 2.4035 - accuracy: 0.4010 - va
Epoch 44/100
276/276 [==============================] - 28s 100ms/step - loss: 2.3760 - accuracy: 0.4014 - va
Epoch 45/100
276/276 [==============================] - 28s 100ms/step - loss: 2.3580 - accuracy: 0.4062 - va
Epoch 46/100
276/276 [==============================] - 28s 100ms/step - loss: 2.3046 - accuracy: 0.4170 - va
Epoch 47/100
276/276 [==============================] - 28s 100ms/step - loss: 2.2868 - accuracy: 0.4199 - va
Epoch 48/100
276/276 [==============================] - 28s 100ms/step - loss: 2.2572 - accuracy: 0.4276 - va
Epoch 49/100
276/276 [==============================] - 28s 100ms/step - loss: 2.2403 - accuracy: 0.4357 - va
Epoch 50/100
276/276 [==============================] - 28s 100ms/step - loss: 2.2445 - accuracy: 0.4334 - va
Epoch 51/100
276/276 [==============================] - 27s 99ms/step - loss: 2.1690 - accuracy: 0.4480 - val_
Epoch 52/100
276/276 [==============================] - 28s 100ms/step - loss: 2.1456 - accuracy: 0.4507 - va
Epoch 53/100
276/276 [==============================] - 28s 100ms/step - loss: 2.1320 - accuracy: 0.4572 - va
Epoch 54/100
276/276 [==============================] - 28s 100ms/step - loss: 2.1133 - accuracy: 0.4655 - va
Epoch 55/100
276/276 [==============================] - 28s 100ms/step - loss: 2.0881 - accuracy: 0.4685 - va
Epoch 56/100
276/276 [==============================] - 28s 100ms/step - loss: 2.0831 - accuracy: 0.4695 - va
Epoch 57/100
276/276 [==============================] - 28s 100ms/step - loss: 2.0491 - accuracy: 0.4786 - va
Epoch 58/100
276/276 [==============================] - 28s 100ms/step - loss: 2.0318 - accuracy: 0.4788 - va
Epoch 59/100
276/276 [==============================] - 28s 100ms/step - loss: 2.0020 - accuracy: 0.4866 - va
Epoch 60/100
276/276 [==============================] - 28s 99ms/step - loss: 1.9888 - accuracy: 0.4939 - val_
Epoch 61/100
276/276 [==============================] - 28s 100ms/step - loss: 1.9852 - accuracy: 0.4909 - va
Epoch 62/100
276/276 [==============================] - 28s 100ms/step - loss: 1.9580 - accuracy: 0.5003 - va
Epoch 63/100
276/276 [==============================] - 28s 100ms/step - loss: 1.9306 - accuracy: 0.5053 - va
Epoch 64/100
276/276 [==============================] - 28s 100ms/step - loss: 1.9132 - accuracy: 0.5073 - va
Epoch 65/100
276/276 [==============================] - 28s 100ms/step - loss: 1.8943 - accuracy: 0.5151 - va
Epoch 66/100
276/276 [==============================] - 28s 99ms/step - loss: 1.8958 - accuracy: 0.5164 - val_
Epoch 67/100
```

```
276/276 [==============================] - 28s 100ms/step - loss: 1.8562 - accuracy: 0.5239 - va
Epoch 68/100
276/276 [==============================] - 28s 100ms/step - loss: 1.8596 - accuracy: 0.5245 - va
Epoch 69/100
276/276 [==============================] - 27s 99ms/step - loss: 1.8423 - accuracy: 0.5284 - val_
Epoch 70/100
276/276 [==============================] - 28s 100ms/step - loss: 1.8089 - accuracy: 0.5361 - va
Epoch 71/100
276/276 [==============================] - 28s 100ms/step - loss: 1.7696 - accuracy: 0.5419 - va
Epoch 72/100
276/276 [==============================] - 28s 100ms/step - loss: 1.7929 - accuracy: 0.5411 - va
Epoch 73/100
276/276 [==============================] - 28s 100ms/step - loss: 1.7757 - accuracy: 0.5498 - va
Epoch 74/100
276/276 [==============================] - 27s 99ms/step - loss: 1.7553 - accuracy: 0.5510 - val_
Epoch 75/100
276/276 [==============================] - 28s 100ms/step - loss: 1.7105 - accuracy: 0.5555 - va
Epoch 76/100
276/276 [==============================] - 28s 100ms/step - loss: 1.7375 - accuracy: 0.5497 - va
Epoch 77/100
276/276 [==============================] - 28s 100ms/step - loss: 1.7055 - accuracy: 0.5609 - va
Epoch 78/100
276/276 [==============================] - 28s 99ms/step - loss: 1.6559 - accuracy: 0.5726 - val_
Epoch 79/100
276/276 [==============================] - 28s 100ms/step - loss: 1.6777 - accuracy: 0.5704 - va
Epoch 80/100
276/276 [==============================] - 28s 101ms/step - loss: 1.6612 - accuracy: 0.5706 - va
Epoch 81/100
276/276 [==============================] - 28s 100ms/step - loss: 1.6391 - accuracy: 0.5783 - va
Epoch 82/100
276/276 [==============================] - 28s 100ms/step - loss: 1.6321 - accuracy: 0.5789 - va
Epoch 83/100
276/276 [==============================] - 28s 100ms/step - loss: 1.6149 - accuracy: 0.5870 - va
Epoch 84/100
276/276 [==============================] - 28s 100ms/step - loss: 1.5950 - accuracy: 0.5865 - va
Epoch 85/100
276/276 [==============================] - 28s 100ms/step - loss: 1.5674 - accuracy: 0.5956 - va
Epoch 86/100
276/276 [==============================] - 27s 99ms/step - loss: 1.5804 - accuracy: 0.5909 - val_
Epoch 87/100
276/276 [==============================] - 28s 100ms/step - loss: 1.5799 - accuracy: 0.5927 - va
Epoch 88/100
276/276 [==============================] - 28s 100ms/step - loss: 1.5608 - accuracy: 0.5998 - va
Epoch 89/100
276/276 [==============================] - 28s 100ms/step - loss: 1.5501 - accuracy: 0.6031 - va
Epoch 90/100
276/276 [==============================] - 28s 100ms/step - loss: 1.5289 - accuracy: 0.6047 - va
Epoch 91/100
276/276 [==============================] - 28s 100ms/step - loss: 1.5120 - accuracy: 0.6124 - va
Epoch 92/100
276/276 [==============================] - 28s 100ms/step - loss: 1.5022 - accuracy: 0.6164 - va
Epoch 93/100
276/276 [==============================] - 28s 100ms/step - loss: 1.4978 - accuracy: 0.6143 - va
Epoch 94/100
276/276 [==============================] - 28s 100ms/step - loss: 1.4972 - accuracy: 0.6140 - va
Epoch 95/100
276/276 [==============================] - 28s 100ms/step - loss: 1.4752 - accuracy: 0.6153 - va
Epoch 96/100
276/276 [==============================] - 28s 100ms/step - loss: 1.4629 - accuracy: 0.6232 - va
Epoch 97/100
276/276 [==============================] - 28s 100ms/step - loss: 1.4579 - accuracy: 0.6228 - va
Epoch 98/100
276/276 [==============================] - 28s 100ms/step - loss: 1.4504 - accuracy: 0.6279 - va
Epoch 99/100
276/276 [==============================] - 28s 100ms/step - loss: 1.4177 - accuracy: 0.6328 - va
```

```
Epoch 100/100
276/276 [==============================] - 28s 99ms/step - loss: 1.4391 - accuracy: 0.6336 - val_
```

11

12

```
# Compile the model
from keras import metrics

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy',
                       metrics.AUC(name='my_auc'),
                       F1_Score])
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-12-da8c9a7433e9> in <module>
      6               metrics=['accuracy',
      7                        metrics.AUC(name='my_auc'),
----> 8                        F1_Score])
      9

NameError: name 'F1_Score' is not defined
```

```
# https://keras.io/api/callbacks/
# We can use a variety of pre-defined callbacks.
# Experiment with ReduceLROnPlateuau()

import tensorflow_addons as tfa

from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, CSVLogger, Learni

# We can also do a modelcheck point
# https://machinelearningmastery.com/check-point-deep-learning-models-keras/

# checkpoint to save the model with best validation accuracy
checkpoint = ModelCheckpoint(filepath='model.{epoch:02d}-{val_loss:.2f}.h5',
                             monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')

# We can also stop the model early
#https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-
# val_loss
early_stopping_callback = EarlyStopping(monitor='accuracy', mode='min', verbose=1, patience=200)


# initialize TimeStopping callback
# https://www.tensorflow.org/addons/tutorials/time_stopping
# note that it will still run a minimum of 1 epoch
time_stopping_callback = tfa.callbacks.TimeStopping(seconds=600, verbose=1)

# We can also use CVSLogger to log information in a CSV
csvlogger = CSVLogger("logfile.csv",separator=',',append=False)


# ** IMPORTANT ** - please make sure that csvlogger is the last call back
# in the list.

my_callbacks = [time_stopping_callback,early_stopping_callback,checkpoint,csvlogger]
```

```
# Fitting the model with call-backs

num_epochs = 1

history=model.fit(train_data,
                  steps_per_epoch =train_steps_per_epoch,
                  validation_data=valid_data,
                  epochs=num_epochs,
                  validation_steps=valid_steps_per_epoch,
                  callbacks=my_callbacks)
```

```
# Compile the model
from keras import metrics
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy',
                        metrics.AUC(name='auc'),
                        metrics.Precision(name='precision'),
                        metrics.Recall(name='recall')])

# Fitting the model with more metrics

num_epochs = 1

history=model.fit(train_data,
                  steps_per_epoch =train_steps_per_epoch,
                  validation_data=valid_data,
                  epochs=num_epochs,
                  validation_steps=valid_steps_per_epoch,
                  callbacks=my_callbacks)
```

```
# Defining custom metrics to record while running
from keras import backend as K

def F1_Score(y_true, y_pred): #taken from old keras source code
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val

def my_metric_fn(y_true, y_pred):
    squared_difference = tf.square(y_true - y_pred)
    return tf.reduce_mean(squared_difference, axis=-1)  # Note the `axis=-1`
```

```
# Compile the model
from keras import metrics
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
```

```
              metrics=['accuracy',
                        metrics.AUC(name='auc'),
                        metrics.Precision(name='precision'),
                        metrics.Recall(name='recall'),
                        F1_Score])

# Fitting the model with more metrics

num_epochs = 1

history=model.fit(train_data,
                  steps_per_epoch =train_steps_per_epoch,
                  validation_data=valid_data,
                  epochs=num_epochs,
                  validation_steps=valid_steps_per_epoch,
                  callbacks=my_callbacks)



# Defining custom call backs

# https://www.tensorflow.org/guide/keras/custom_callback
# https://keras.io/guides/writing_your_own_callbacks/

from keras.callbacks import Callback
import time

class TimingCallback(keras.callbacks.Callback):
    def __init__(self):
        super(TimingCallback, self).__init__()
    def on_batch_begin(self, epoch, logs=None):
        self.starttime=time.time()
    def on_batch_end(self, epoch, logs=None):
        logs['epoch_time'] = (time.time()-self.starttime)
        print('\nepoch_time(sec)=',logs['epoch_time'],'\n')

# create an instance of the timingcallback
timing_call = TimingCallback()

# We can also use other metrics
# https://keras.io/api/metrics/
class PrintBatchCallback(keras.callbacks.Callback):
    def on_train_batch_end(self, batch, logs=None):
        print("For batch {}, loss is {:7.2f}.".format(batch, logs["loss"]))
        print("For batch {}, accuracy is {:7.2f}.".format(batch, logs["accuracy"]))
        print("For batch {}, AUC is {:7.2f}.".format(batch, logs["auc"]))

print_batch_call = PrintBatchCallback()

# add to the callback list
my_callbacks = [time_stopping_callback,early_stopping_callback,checkpoint,print_batch_call, timi




# Compile the model
from keras import metrics
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy',
                        metrics.AUC(name='auc'),
                        metrics.Precision(name='precision'),
                        metrics.Recall(name='recall'),
                        F1_Score])

# Fitting the model with more metrics
```

```
num_epochs = 1

history=model.fit(train_data,
                  steps_per_epoch =train_steps_per_epoch,
                  validation_data=valid_data,
                  epochs=num_epochs,
                  validation_steps=valid_steps_per_epoch,
                  callbacks=my_callbacks)



# https://neptune.ai/blog/keras-metrics

# How to save batch level data in a file

import os
from keras.callbacks import Callback
import numpy as np


class SaveBatchLevelDataCallback(keras.callbacks.Callback):
    def __init__(self, validation_data, save_dir):
        super().__init__()
        self.validation_data = validation_data
        os.makedirs(save_dir, exist_ok=True)
        self.save_dir = save_dir
        self.f = None

    def on_epoch_begin(self, epoch, logs=None):
        # create a file
        self.f= open(os.path.join(self.save_dir, f'epoch_{epoch}.csv'),'w+')
        line = "batch,loss,accuracy,auc\n"
        self.f.write(line)

    def on_epoch_end(self, batch, logs=None):
        self.f.close()

    def on_train_batch_end(self, batch, logs=None):
        line = "{},{:7.2f},{:7.2f},{:7.2f}\n".format(batch, logs["loss"], logs["accuracy"],logs[
        self.f.write(line)


batch_write_cbk = SaveBatchLevelDataCallback(validation_data=valid_data,save_dir='batch_data')

# add to the callback list
my_callbacks = [time_stopping_callback,early_stopping_callback,checkpoint,batch_write_cbk, CSVLo





# # Compile the model
from keras import metrics
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy',
                       metrics.AUC(name='auc'),
                       metrics.Precision(name='precision'),
                       metrics.Recall(name='recall'),
                       F1_Score])

# Fitting the model with more metrics

num_epochs = 10

history=model.fit(train_data,
```

```
                  steps_per_epoch =train_steps_per_epoch,
                  validation_data=valid_data,
                  epochs=num_epochs,
                  validation_steps=valid_steps_per_epoch,
                  callbacks=my_callbacks)


# print history
print(history.history)


#plot accuracy vs epoch
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validate'], loc='upper left')
plt.show()

# Plot loss values vs epoch
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validate'], loc='upper left')
plt.show()

# Plot loss values vs epoch
plt.plot(history.history['F1_Score'])
plt.plot(history.history['val_F1_Score'])
plt.title('Model F1-Score')
plt.ylabel('F1_Score')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validate'], loc='upper left')
plt.show()

# Plot accuracy vs. prevision
plt.plot(history.history['precision'],label='precision')
plt.plot(history.history['val_precision'],label='val_precision')
plt.plot(history.history['recall'],label='recall')
plt.plot(history.history['val_recall'],label='val_precision')
plt.title('Model Precision and Recall')
plt.ylabel('Precision and Recall')
plt.xlabel('Epoch')
plt.legend()
plt.show()

# Plot accuracy vs. prevision
plt.plot(history.history['precision'],history.history['recall'],'o', color='black',label='precis
plt.plot(history.history['recall'],history.history['val_recall'],'o', color='red',label='val_pre
plt.title('Model Precision and Recall')
plt.ylabel('Precision')
plt.xlabel('Recall')
plt.legend()
plt.show()

# Evaluate against test data.
scores = model.evaluate(test_data, verbose=1)

print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
print('Test AUC:', scores[1])
```

```
print('Test precision:', scores[1])
print('Test recall:', scores[1])
print('Test F1-Score:', scores[1])




# For evaluation first, we will create the actual and predicted labels
# We can then use these to generate all the reports we need.

# make predictions on the testing images, finding the index of the
# label with the corresponding largest predicted probability

predicted = model.predict(x=test_data, steps=test_steps_per_epoch)

# create predited IDs
predicted = np.argmax(predicted, axis=1)

# create test labels from the generator
actual = []
for i in range(0,int(test_steps_per_epoch)):
    actual.extend(np.array(test_data[i][1]))

# create actual IDs
actual = np.asarray(actual).argmax(axis=1)

# make sure predicted and actual are the same size and shape
print(predicted.shape)
print(actual.shape)


from sklearn.metrics import classification_report

print("[INFO] evaluating network...")
print(classification_report(actual, predicted))


# Now we can determine the confusion matrix
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(actual,predicted)

def print_cm(cm, frm, to,abs_or_relative=0):
    import seaborn as sns
    import matplotlib.pylab as plt

    cm = cm[frm:to+1,frm:to+1]
    # create labels
    x_axis_labels = np.arange(frm,to+1)
    y_axis_labels = np.arange(frm,to+1)

    plt.xticks(rotation=45)
    plt.yticks(rotation=-45)

    if(abs_or_relative==0):
        sns.heatmap(cm, annot=True,xticklabels=x_axis_labels, yticklabels=y_axis_labels)
    else:
        sns.heatmap(cm/np.sum(cm), annot=True,
           fmt='.2%', cmap='Blues',
           xticklabels=x_axis_labels, yticklabels=y_axis_labels)

print_cm(cm,1 ,20,0)


# we already have actual and predicted
```

```
# also see https://www.dlology.com/blog/simple-guide-on-how-to-generate-roc-plot-for-keras-class
# for micro-average ROC curves as well

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

fpr = dict()
tpr = dict()
roc_auc = dict()

#extract the actual labels from the test data
Y_test = []
for i in range(0,int(test_steps_per_epoch)):
    Y_test.extend(np.array(test_data[i][1]))
Y_test = np.array(Y_test)
n_classes = Y_test.shape[1]  # one hot encoded

# create actual output from the model using test_data
y_score=model.predict(x=test_data, steps=test_steps_per_epoch)

print(Y_test.shape)
print(y_score.shape)



print(n_classes)
# compare each class's probabilities one by one
# each acts like a single column
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(Y_test[:,i], y_score[:,i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Print the AUC scores
from IPython.display import display
import pandas as pd
auc_array = np.array(list(roc_auc.items()))
df = pd.DataFrame(auc_array[:,1])
df.columns = ['AUC']
display(df)



# plot the ROC for the ith class cls
import matplotlib.pyplot as plt
import os

def plot_roc(cls,roc_dir):
    plt.plot(fpr[cls], tpr[cls], lw=2,label='ROC curve of class {0} (area = {1:0.3f})'
    ''.format(cls, roc_auc[cls]))
    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC')
    plt.legend(loc="lower right")
    plt.tight_layout()
    plt.savefig(os.path.join(roc_dir, f'ROC_{cls}.png'))
    plt.show()


# make sure directory exists
def make_directory(roc_dir):
    try:
        os.mkdir(roc_dir)
```

```
    except OSError:
        print ("Creation of the directory %s failed" % roc_dir)
    else:
        print ("Successfully created the directory %s " % roc_dir)

# print the roc curve for 0

make_directory('rocs')

for i in range(n_classes):
    plot_roc(i,'rocs')



# Using tensorflow extension
# Load the TensorBoard notebook extension
%load_ext tensorboard
import datetime



# Define tensorboard callback

log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

# Using remote tensorboard
#https://blog.yyliu.net/remote-tensorboard/



# Compile the model
from keras import metrics
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy',
                        metrics.AUC(name='auc'),
                        metrics.Precision(name='precision'),
                        metrics.Recall(name='recall')])

# Fitting the model with more metrics

num_epochs = 10

history=model.fit(train_data,
                  steps_per_epoch =train_steps_per_epoch,
                  validation_data=valid_data,
                  epochs=num_epochs,
                  validation_steps=valid_steps_per_epoch,
                  callbacks=[tensorboard_callback])



#%tensorboard --logdir logs/fit
```