Satellite Image Classification using Deep Neural Network with Keras in R with GPU Support (Windows 10)

Zia Ahmed, PhD, University at Buffalo

April 5, 2018

This tutorial will show how to implement Deep Neural Network for pixel based supervised classification of Sentinel-2 multispectral images using keras package in R under Windows 10.

keras is a popular Python package for deep neural networks with multiple backends, including TensorFlow, Microsoft Cognitive Toolkit (CNTK), and Theano. Two R packages allow you to use [Keras[(https://keras.rstudio.com/)] from R: keras and kerasR. The keras package is able to provide a flexible and feature-rich API and can run both CPU and GUP version of TensorFlow in both Windows and Linux. If you want to run this tutorial with GUP version of TensorFlow you need following prerequisites in your system:

*NVIDIA GUP: First, you must make sure weather your computer is running with NVIDIAÂ® GPU or not. Follow the instruction as described here.

*CUDA Toolkit v9.0: If you have an NVIDIAÂ® GPU in your system, you need to download and install CUDA Toolkit v9.0. Detail installation steps can be found here.

*cuDNN v7.0: The download the zip file version cuDNN v7.0 for your CUDA Toolkit v9.0.You need to extract the zip file and add the location where you extracted it to your system PATH. Detail installation steps can be found here here.

Detail installation steps of Keras backend GPU or CUP version of Tensorflow can be found here.

First, we will split "point_data" into a training set (75% of the data), a validation set (12%) and a test set (13%) data.The validation data set will be used to optimize the model parameters during training process.The model's performance will be tested with the data set and then we will predict landuse clasess on grid data set. The point and grid data can be download as rar, 7z and zip format.

```
start_time <- Sys.time()
```

Import packages

```
library(rgdal)
library(raster)
library(dplyr)
library(RStoolbox)
library(plyr)
library(keras)
```

```
library(tfruns)
library(tfestimators)
```

Setworking directory

```
setwd("F:\\My_GitHub\\DNN_keras_R")
```

Load data

```
point<-read.csv("point_data.csv", header=T)
grid<-read.csv("grid_data.csv",header=T)
```

Create a data frame and clean the data

```
point.df<-cbind(point[c(4:13)],Class_ID=point$Class)
grid.df<-cbind(grid[c(4:13)])
grid.xy<-grid[c(3,1:2)]
```

Convert Class to dummy variables

```
point.df[,11] <- as.numeric(point.df[,11]) -1
```

Convert data as matrix

```
point.df<- as.matrix(point.df)
grid.df <- as.matrix(grid.df)
```

Set dimnames to NULL

```
dimnames(point.df) <- NULL
dimnames(grid.df) <- NULL
```

Standardize_the data: ((x-mean(x))/sd(x))

```
point.df[, 1:10] = scale(point.df[, 1:10])
grid.df[, 1:10] = scale(grid.df[, 1:10])
```

Split data

```
##  Determine sample size
ind <- sample(2, nrow(point.df), replace=TRUE, prob=c(0.80, 0.20))
# Split the `Split data
training <- point.df[ind==1, 1:10]
test <- point.df[ind==2, 1:10]
# Split the class attribute
trainingtarget <- point.df[ind==1, 11]
testtarget <- point.df[ind==2, 11]
```

Hyperparameter flag

```
FLAGS <- flags(
  flag_numeric('dropout_1', 0.2, 'First dropout'),
  flag_numeric('dropout_2', 0.2, 'Second dropout'),
  flag_numeric('dropout_3', 0.1, 'Third dropout'),
  flag_numeric('dropout_4', 0.1, 'Forth dropout')
  )
```

Define model parameters with 4 hidden layers with 200 neuron

```
model <- keras_model_sequential()
```

```
## Warning in normalizePath(path.expand(path), winslash, mustWork):
## path[1]="C:\Users\zua3\AppData\Local\conda\conda\envs\py27/python.exe": The
## system cannot find the file specified
```

```
model %>%
  # Imput layer
  layer_dense(units = 200, activation = 'relu',
              kernel_regularizer =regularizer_l1_l2(l1 = 0.00001, l2 =
0.00001),input_shape = c(10)) %>%
  layer_dropout(rate = FLAGS$dropout_1,seed = 1) %>%
  # Hidden layers
  layer_dense(units = 200, activation = 'relu',
              kernel_regularizer = regularizer_l1_l2(l1 = 0.00001, l2 = 0.00001)) %>%
  layer_dropout(rate = FLAGS$dropout_2,seed = 1) %>%
  layer_dense(units = 200, activation = 'relu',
              kernel_regularizer = regularizer_l1_l2(l1 = 0.00001, l2 = 0.00001)) %>%
  layer_dropout(rate = FLAGS$dropout_3,seed = 1) %>%
  layer_dense(units = 200, activation = 'relu',
              kernel_regularizer = regularizer_l1_l2(l1 = 0.0001, l2 = 0.00001)) %>%
  layer_dropout(rate = FLAGS$dropout_4) %>%
  # Output layer
  layer_dense(units = 5, activation = 'softmax')
summary(model)
```

```
## ___________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ===========================================================================
## dense_1 (Dense)                  (None, 200)                   2200
## ___________________________________________________________________________
## dropout_1 (Dropout)              (None, 200)                   0
## ___________________________________________________________________________
## dense_2 (Dense)                  (None, 200)                   40200
## ___________________________________________________________________________
## dropout_2 (Dropout)              (None, 200)                   0
## ___________________________________________________________________________
## dense_3 (Dense)                  (None, 200)                   40200
## ___________________________________________________________________________
## dropout_3 (Dropout)              (None, 200)                   0
## ___________________________________________________________________________
## dense_4 (Dense)                  (None, 200)                   40200
## ___________________________________________________________________________
## dropout_4 (Dropout)              (None, 200)                   0
## ___________________________________________________________________________
## dense_5 (Dense)                  (None, 5)                     1005
## ===========================================================================
## Total params: 123,805
## Trainable params: 123,805
## Non-trainable params: 0
##
```

Define an optimizer (Stochastic gradient descent optimizer)

```
optimizer <- optimizer_sgd(lr=0.01)
```

Compile the model

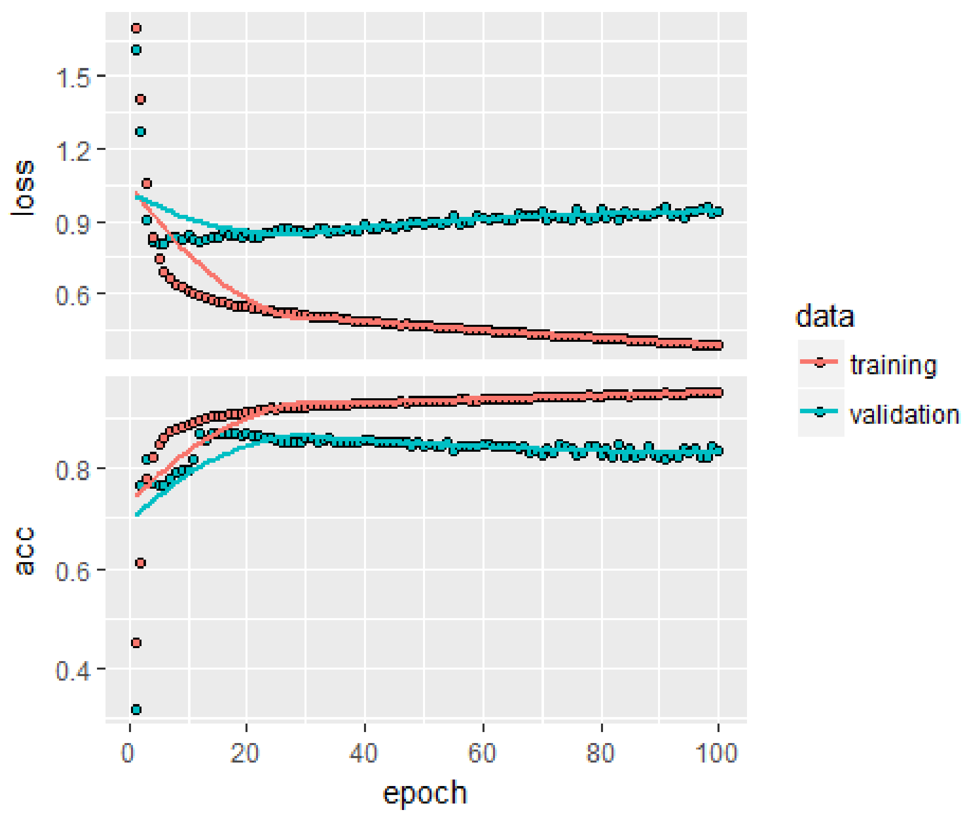
```
model %>% compile(
  loss = 'sparse_categorical_crossentropy',
  optimizer = optimizer,
  metrics = 'accuracy'
)
```

Fit the model to the data

```
history<-model %>% fit(
  training, trainingtarget,
  epochs = 100,
  batch_size = 100,
  shuffle = TRUE,
  validation_split = 0.2,
  callbacks = callback_tensorboard()
)
```

Plot history

```
plot(history)
```



Evaluate the model

```
score <- model %>% evaluate(test, testtarget, batch_size = 100)
cat('Test loss:', score[[1]], '\n')

## Test loss: 0.4655384

cat('Test accuracy:', score[[2]], '\n')
```

```
## Test accuracy: 0.935908
```

Prediction & confusion matrix - test data

```
class.test <- model %>%
  predict_classes(test, batch_size = 100)
# Confusion matrix
table(testtarget,class.test)

##           class.test
## testtarget    0    1    2    3    4
##          0 1019   78   24    1    0
##          1   59  770   53    4    0
##          2   41   11 1597   15    0
##          3    0    0   22  985    0
##          4    0    0    4    0  185
```

Predicted Class Probability

```
prob.test <- model %>%
  predict_proba(test, batch_size = 100)
```

Prediction at grid locations

```
Class.grid <- model %>%
  predict_classes(grid.df, batch_size = 100)
```

Detach keras, tfruns, tftestimators

```
detach(package:keras, unload=TRUE)
detach(package:tfruns, unload=TRUE)

## Warning: 'tfruns' namespace cannot be unloaded:
##   namespace 'tfruns' is imported by 'tensorflow', 'tfestimators' so cannot be
unloaded

detach(package:tfestimators, unload=TRUE)
```

Change column name

```
class<-as.data.frame(Class.grid)
new.grid<-cbind(x=grid.xy$x, y=grid.xy$y,Class_ID=class )
names(new.grid)

## [1] "x"          "y"          "Class.grid"

colnames(new.grid)[3]<-"Class_ID"
new.grid.na<-na.omit(new.grid)
```

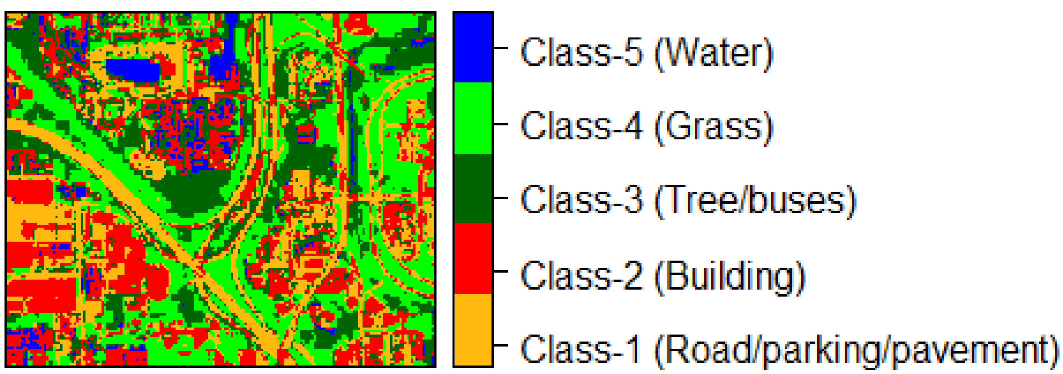
Load landuse ID file

```
#### Join Class Id Column
ID<-read.csv("Landuse_ID_keras.csv", header=TRUE)
ID

##   Class_ID   Class            Description
## 1        0 Class_1 Parking/road/pavement
## 2        1 Class_2              Building
## 3        2 Class_3           Tree/bushes
```

```
## 4         3 Class_4                    Grass
## 5         4 Class_5                    Water
```

Convert to raster

```
#### Convert to raster
x<-SpatialPointsDataFrame(as.data.frame(new.grid.na)[, c("x", "y")], data =
new.grid.na)
r <- rasterFromXYZ(as.data.frame(x)[, c("x", "y", "Class_ID")])

myPalette <- colorRampPalette(c("darkgoldenrod1","red", "darkgreen","green", "blue"))
spplot(r,"Class_ID",
      colorkey = list(space="right",tick.number=1,height=1, width=1.5,
              labels = list(at = seq(0,3.8,length=5),cex=1.0,
              lab = c("Class-1 (Road/parking/pavement)" ,"Class-2 (Building)",
"Class-3 (Tree/buses)", "Class-4 (Grass)", "Class-5 (Water)"))),
              col.regions=myPalette,cut=4)
```




```
writeRaster(r,"predicted_Landuse.tiff","GTiff",overwrite=TRUE)
```

Run time

```
end_time <- Sys.time()
end_time - start_time

## Time difference of 2.194659 mins
```

Conclusions

This simple pixel-based satellite image classification algorithm with deep neural network in R with keras able to identify urban objects with high accuracy. It may be use full for landuse classification for urban environment monitoring as well as planning purpose. Also, may use full for agricultural landuse classification.

Clean everyrhing

```
gc()

##           used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 3202700 171.1    4703850 251.3  4703850 251.3
## Vcells 5987154  45.7   10178327  77.7  8411888  64.2
```