STAT 760 Final Project

Jesse Krawiecki, Benjamin Claassen

May, 16, 2019

Introduction

This final project involves the CIFAR 10 dataset. This dataset is provided by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton from the website [https://www.cs.toronto.edu/~kriz/cifar.html]. It consists of 60,000 unique pre-classified images. These images are in color, are 32 by 32 pixels, and are for 10 classes of object. The 10 classes of objects covered are: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images for each class, and the dataset is split into 50,000 images for training a given algorithm, and 10,000 for testing it.

Part 1: Neural Network

The first part of the project involves using a neural network to attempt classification of these images. Before attempting any network experimentation, each image was converted to greyscale from the original red-blue-green (RGB) format. To do this, the RGB pixel values are added, then scaled to be between 0 and 1.

Next, various network architectures were examined using the package Keras and TensorFlow. These are machine learning packages written in Python, but are useable in R and RStudio, which was our approach.

There are multiple hyperparameters that can be tweaked in a neural network. One of these is the number and type of hidden layers in the network. We started with a convolutional layer with a kernel size of 3x3 pixels. We tried networks with a single one of these layers, as well as networks with up to three. Of these, higher numbers of convolutional layers were more accurate by around 1-3 percentage points. We also tried different kernel sizes. Kernels of size 2x2 pixels were notably faster than either 3x3- or 4x4 kernels, but where somewhat less accurate, around 1 percentage point on average. Therefore, we settled on convolutional layers with 3x3 kernel size as a compromise.

Another modification we experimented with how far the kernel moves each iteration. This is called the 'stride'. Higher stride values increased the speed for each epoch, but also in some cases, increased the prediction accuracy. We used a stride of 2 pixels in the first convolutional layer, and a stride of 1 in the second. We also tried multiple numbers and sizes of densely connected hidden layers. We ultimately used just one of these layers, with 512 nodes. Based on in-class discussion and outside research, we decided to use the 'ReLU' or Rectified Linear Unit activation function for the dense

and convolutional layers.

Finally, the dropout technique was implemented in two places within the final network. First, a 25% dropout was instituted after the convolutional layers. This is with the goal of making the connections between the spatial data and the first fully interconnected layers more robust. That is, preventing the network from relying too strongly on a particular part of a given image. Second, a 50% dropout regularization is implemented just before the final classification. This is to prevent overfitting individual classes to certain nodes or collection of nodes in the final hidden layer. All fully connected layers were given a bias term. The final output was a fully connected layer that mapped its' output to ten final nodes. These ten nodes each represent one of the classes.

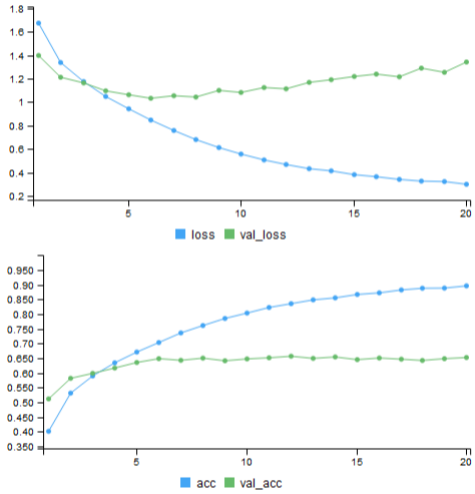
Two additional techniques discussed were used to help convergence of the network. First, we used batch normalization. Very small batches relative to the size of the training dataset resulted in a consistent gain in predictive ability. Using a batch size of 30 was two to three percentage points more accurate for the test data than a batch size of 250.

As well as using batch normalization, different approaches to the learning rate were also experimented with. The first approach involved using an exponentially decaying learning rate. This worked, but was relatively slow to converge. Instead, we ultimately used an option included in the Keras package called 'Adam'.

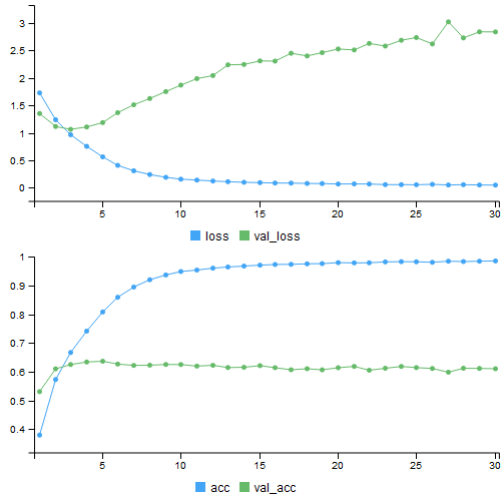
Adam employs both exponential decay and momentum to optimize gradient descent http://ruder.io/optimizing-gradient-descent/index.html#adam. This seems to be a popular choice among practitioners. A computer vision course from Stanford University states that it is "currently recommended as the default algorithm to use" http://cs231n.github.io/, and indeed, achieves the highest accuracy of any of our networks while still converging incredibly quickly.

With this network and learning rate algorithm, iterating over more than 20 epochs was unnecessary when using the Adam approach to the learning rate. Indeed, the final model converged to a relatively fixed test accuracy almost immediately. This is shown in the first image below, and is contrasted with the second image where a standard learning rate decay approach was used.

Adam Learning Rate

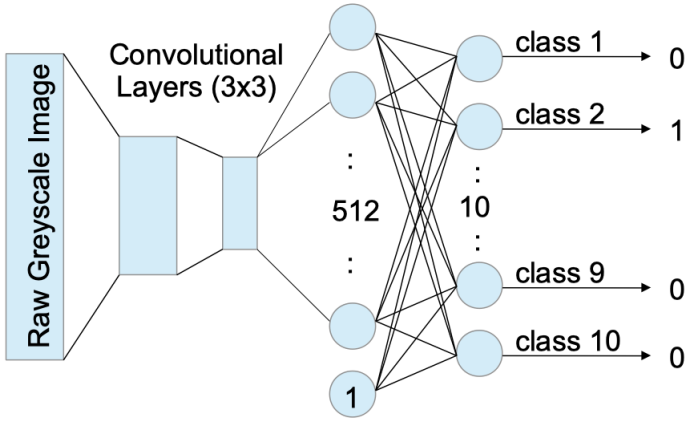


Exponential Decay Learning Rate



Ultimately, our highest test prediction accuracy was 65.2%. This was found using a network with two convolutional layers, each with a kernel size of 3x3 pixels. This was followed by a dropout procedure, and then a fully connected layers with 512. Finally, dropout was implemented again, and a fully connected layer with ten nodes was added. This finally layer has one node for each class. The ReLU activation function was used for each layer expect the last, where we used the softmax function. The softmax activation function is useful for classification as it normalizes to 0 and 1.

Final Network



Confusion Matrix

	airplanes	cars	birds	cats	deer	dogs	frogs	horses	ships	trucks
airplanes	632	17	82	19	48	8	19	26	98	51
cars	12	698	8	20	7	9	23	6	42	175
birds	50	13	505	71	126	95	58	43	23	16
cats	25	12	71	441	95	163	72	60	16	45
deer	17	5	59	89	610	54	53	89	15	9
dogs	14	5	52	157	61	596	27	62	14	12
frogs	9	17	56	70	50	24	730	10	9	25
horses	8	2	31	52	60	84	6	728	9	20
ships	54	32	18	23	17	10	5	4	790	47
trucks	26	63	8	23	13	13	11	13	37	793

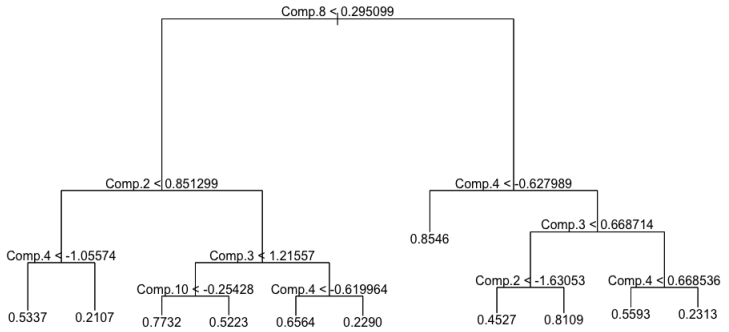
Part 2: Random Forest

The second part of the assignment takes the Cifar 10 dataset and employs a classification random tree model. To do so we used the randomForest package in R. To simplify the classification process we focused on only two classes, horses and cars. We also transformed the data into greyscale and flattened the data into 1024 variables, each one representing a pixel. To get the model to work we also had to add the hot-encoded classes to the data, resulting in a binary response variable. We also had to combine the test and training sets to use cross-validation, and chose to use six cross-validation folds.

To increase the efficiency of the modeling process we also reduced the dimensionality of the data by implementing Principal Component Analysis. We ultimately settled on using just 50 principal components. These 50 components explained 84.5% of the variance and greatly sped up the time it took to perform classification. This did not significantly affect the accuracy of the prediction.

Our random forest model also incorporated bagging. We experimented with a multiple numbers of trees. Our benchmark number of trees is 50. Using frequency of the classes in the training data the baseline naive prediction is correct 50% of the time. We achieved an accuracy of 83.98% with 50 trees. The cross-validated error associated with this model is 0.1396.

Example of tree



The below confusion matrices correspond to 50 bagged trees and six cross-validation fold:

Confusion Matrix 1

	horses	cars
horses	823	151
cars	148	878

Confusion Matrix 2

	horses	cars
horses	835	157
cars	143	865

Confusion Matrix 3

	horses	cars
horses	836	159
cars	166	839

Confusion Matrix 4

	horses	cars
horses	840	147
cars	191	822

Confusion Matrix 5

	horses	cars
horses	851	165
cars	174	810

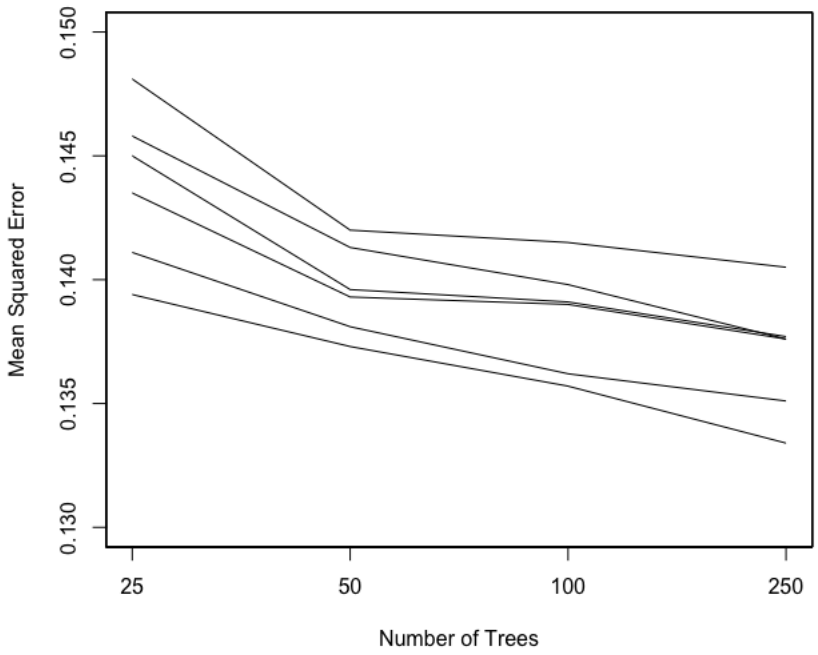
Confusion Matrix 6

	horses	cars
horses	829	157
cars	164	850

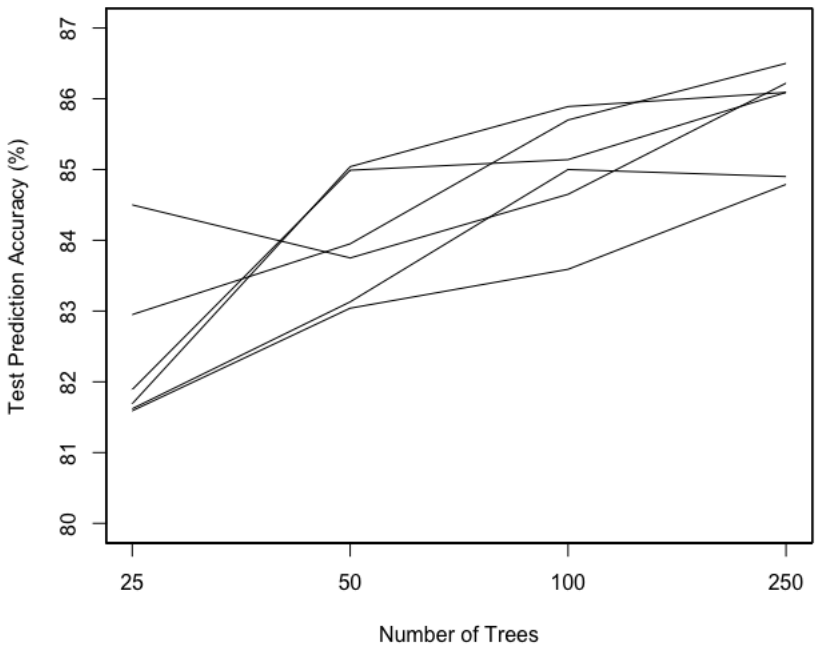
Bagging accuracy and error

number of trees	ave. accuracy	cross-validated error
25	82.37%	0.1438
50	83.98%	0.1396
100	85.00%	0.1386

MSE for each fold of cross-validation



Accuracy for each fold of cross-validation



```
#############
# Data Prep #
#############

###############
# Training Dat

#cifar10 <- dataset_cifar10()
dat10_mat_R <- cifar10[[1]][[1]][,,,1]
dat10_mat_G <- cifar10[[1]][[1]][,,,2]
dat10_mat_B <- cifar10[[1]][[1]][,,,3]

numObs <- dim(dat10_mat_R)[1]
dat10_R <- matrix(,numObs, (32*32) )
dat10_G <- matrix(,numObs, (32*32) )
dat10_B <- matrix(,numObs, (32*32) )
dat10_greyscale <- matrix(,numObs, (32*32) )


for(i in 1:numObs)
{
  print(i)
  dat10_greyscale[i,] <- as.vector(t(dat10_mat_R[i,,])) + as.vector(t(dat10_mat_G[i,,])) +
as.vector(t(dat10_mat_B[i,,]))
  dat10_greyscale[i,] <- dat10_greyscale[i,] / 765
  #dat10_R[i,] <- as.vector(t(dat10_mat_R[i,,]))
  #dat10_G[i,] <- as.vector(t(dat10_mat_G[i,,]))
  #dat10_B[i,] <- as.vector(t(dat10_mat_B[i,,]))
}

save(dat10_R, file = "dat10_R.Rda")
save(dat10_G, file = "dat10_G.Rda")
save(dat10_B, file = "dat10_B.Rda")
save(dat10_greyscale, file = "dat10_greyscale.Rda")



horseIDs <- which(cifar10[[1]][[2]] == 7)
carIDs <- which(cifar10[[1]][[2]] == 1)
dat2_greyscale <- dat10_greyscale[ c(horseIDs,carIDs), ]

tmpIDs <- c( rep(0, length(horseIDs)), rep(1, length(carIDs)) )
dat2_greyscale <- cbind(tmpIDs, dat2_greyscale)

save(dat2_greyscale, file = "dat2_greyscale.Rda")

####################################

############
# Test Dat

dat10_mat_R_test <- cifar10[[2]][[1]][,,,1]
dat10_mat_G_test <- cifar10[[2]][[1]][,,,2]
dat10_mat_B_test <- cifar10[[2]][[1]][,,,3]

numObs_test <- dim(dat10_mat_R_test)[1]
dat10_R_test <- matrix(,numObs_test, (32*32) )
dat10_G_test <- matrix(,numObs_test, (32*32) )
dat10_B_test <- matrix(,numObs_test, (32*32) )
dat10_greyscale_test <- matrix(,numObs_test, (32*32) )


for(i in 1:numObs_test)
{
  print(i)
  dat10_greyscale_test[i,] <- as.vector(t(dat10_mat_R_test[i,,])) + as.vector(t(dat10_mat_G_test[i,,]))
+ as.vector(t(dat10_mat_B_test[i,,]))
  dat10_greyscale_test[i,] <- dat10_greyscale_test[i,] / 765

  dat10_R[i,] <- as.vector(t(dat10_mat_R[i,,]))
  dat10_G[i,] <- as.vector(t(dat10_mat_G[i,,]))
  dat10_B[i,] <- as.vector(t(dat10_mat_B[i,,]))
}
```

```
save(dat10_R_test, file = "dat10_R_test.Rda")
save(dat10_G_test, file = "dat10_G_test.Rda")
save(dat10_B_test, file = "dat10_B_test.Rda")
save(dat10_greyscale_test, file = "dat10_greyscale_test.Rda")

horseIDs_test <- which(cifar10[[2]][[2]] == 7)
carIDs_test <- which(cifar10[[2]][[2]] == 1)
dat2_greyscale_test <- dat10_greyscale_test[ c(horseIDs_test,carIDs_test), ]

save(dat2_greyscale_test, file = "dat2_greyscale_test.Rda")


#########################################################################################################
# Convolutional Neural Network #
################################
library(keras)
library(tensorflow)

batch_size <- 32
epochs <- 30

cifar10 <- dataset_cifar10()

x_train <- array(dat10_greyscale, dim = c(50000,32,32,1))
x_test <- array(dat10_greyscale_test, dim = c(10000,32,32,1))
y_train <- to_categorical(cifar10[[1]][[2]], num_classes = 10)
y_test <- to_categorical(cifar10[[2]][[2]], num_classes = 10)



model <- keras_model_sequential()

model %>%
 # Start with hidden 2D convolutional layer being fed 32x32 pixel images
 layer_conv_2d(
 filter = 32, kernel_size = c(3,3), strides = c(2,2),
 input_shape = c(32,32,1)
 ) %>%
 layer_activation("relu") %>%

 # Second hidden layer
 layer_conv_2d(filter = 32, kernel_size = c(3,3)) %>%
 layer_activation("relu") %>%

 layer_dropout(0.25) %>%

 # Flatten max filtered output into feature vector

 # and feed into dense layer
 layer_flatten() %>%
 layer_dense(512) %>%
 layer_activation("relu") %>%
 layer_dropout(0.5) %>%

 # Outputs from dense layer are projected onto 10 unit output layer
 layer_dense(10) %>%
 layer_activation("softmax")

model %>% compile(
 loss = "categorical_crossentropy",
 optimizer = "adam",
 metrics = "accuracy"
)




history_0 <- model %>% fit(
  x_train, y_train,
  batch_size = batch_size,
  epochs = epochs,
  validation_data = list(x_test, y_test),
```

```
  shuffle = TRUE
)
###########################################################################################################
# Random Forest: Bagging #
##########################
library(randomForest)
library(MASS)
library(tree)
library(dplyr)

load("dat2_greyscale.Rda")
load("dat2_greyscale_test.Rda")

tmpIDs <- c( rep(0, 5000), rep(1, 5000) )
dat2_greyscale <- cbind(tmpIDs, dat2_greyscale)
dat2_greyscale <- as.data.frame(dat2_greyscale)
names(dat2_greyscale) <- c("Class", paste0("pixel_", 1:1024) )

tmpIDs_test <- c( rep(0, 1000), rep(1, 1000) )
dat2_greyscale_test <- cbind(tmpIDs_test, dat2_greyscale_test)
dat2_greyscale_test <- as.data.frame(dat2_greyscale_test)
names(dat2_greyscale_test) <- c("Class", paste0("pixel_", 1:1024) )


cifarDat <- rbind(dat2_greyscale_test, dat2_greyscale)

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
cifarPCA <- princomp(cifarDat[,-1])
sum(head(cifarPCA$sdev^2 / sum(cifarPCA$sdev^2), n = 50))

cifarScores <- as.matrix(cifarPCA$scores)[,1:50]
cifarScores <- as.data.frame(cbind(cifarDat[,1], cifarScores))

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
set.seed(1)
cifarScores <- sample_frac(cifarScores, 1)
fold1 <- 1:2000
fold2 <- 2001:4000
fold3 <- 4001:6000
fold4 <- 6001:8000
fold5 <- 8001:10000
fold6 <- 10001:12000
foldList <- list(fold1, fold2, fold3, fold4, fold5, fold6)

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
for(f in foldList)
{
  tmpObs <- c(1:12000)[-f]
  bag.CIFAR_tmp=randomForest(cifarScores[tmpObs,1] ~.,data=cifarScores[tmpObs,-1], mtry=5,
importance=TRUE, ntree=250)
  #bag.CIFAR_tmp
  yhat.bag = predict(bag.CIFAR_tmp, newdata=cifarScores[f,-1])

  print(table(round(yhat.bag,0), cifarScores[f,1]))
  print(mean((yhat.bag-cifarScores[f,1])^2))
}



treeCIFAR <- tree(cifarScores[,1] ~., cifarScores[,-1])
plot(treeCIFAR)
text(treeCIFAR)
```