# Exercise 2: Group 10, Jafre13

Accuracy function which will be used later

```
accuracy <- function(results,testlabels) {
  count = 0
  for (i in 1:length(results)){
    if (results[i]==testlabels[i]){
      count = count+1
    }
  }
  return((count/length(results))*100)

}
```

1.1 First load the data into a dataframe

```
source('C:/SML/PCA/loadImage.R')
```

```
## Warning: package 'png' was built under R version 3.2.5
```

```
## Warning: package 'gmodels' was built under R version 3.2.5
```

```
## Warning: package 'ggplot2' was built under R version 3.2.5
```

```
## Warning: package 'caret' was built under R version 3.2.5
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 3.2.5
```

```
library("caret")
rawdata1 = loadSinglePersonsData(100,"group10",1,"C:/SMLIMAGES/2017/")
rawdata2 = loadSinglePersonsData(100,"group12",1,"C:/SMLIMAGES/2017/")
```

Shuffling the data for two persons independent and dependendent, and extracting labels from dataset

```
p1 = data.frame(rawdata1)
p2 = data.frame(rawdata2)

combined = rbind(rawdata1,rawdata2)
doTheShuffle <- function(){
  independent <<- data.frame(combined)
  shuffled_inde <<- independent[sample(nrow(independent)),]
  independent_labels <<- shuffled_inde$X1
  shuffled_inde <<- subset(shuffled_inde,select = -c(X1))
}

p1_shuff = p1[sample(nrow(p1)),]
p2_shuff = p2[sample(nrow(p2)),]

doTheShuffle()

p1_labels = p1_shuff$X1
p1_shuff = subset(p1_shuff,select = -c(X1))
p2_labels = p2_shuff$X1
p2_shuff = subset(p2_shuff,select = -c(X1))
```

Performing independent PCA with first 4000 entries, the other 4000 will be used for testing

```
inde_pca <- prcomp(shuffled_inde[1:4000,])
vari = (((inde_pca$sdev)^2)*100)/sum((inde_pca$sdev)^2)
cs = cumsum(vari)
```

Standard deviation:

```
head(inde_pca$sdev,n=10)
```

```
##  [1] 1.2024243 1.0106029 0.6181484 0.5943693 0.5685322 0.5252023 0.4995392
##  [8] 0.4573615 0.4304183 0.4062745
```

Variance

```
head(vari,n=10)
```

```
##  [1] 21.683304 15.316906  5.730539  5.298131  4.847525  4.136788  3.742391
##  [8]  3.137106  2.778379  2.475421
```

Cumsum Variance

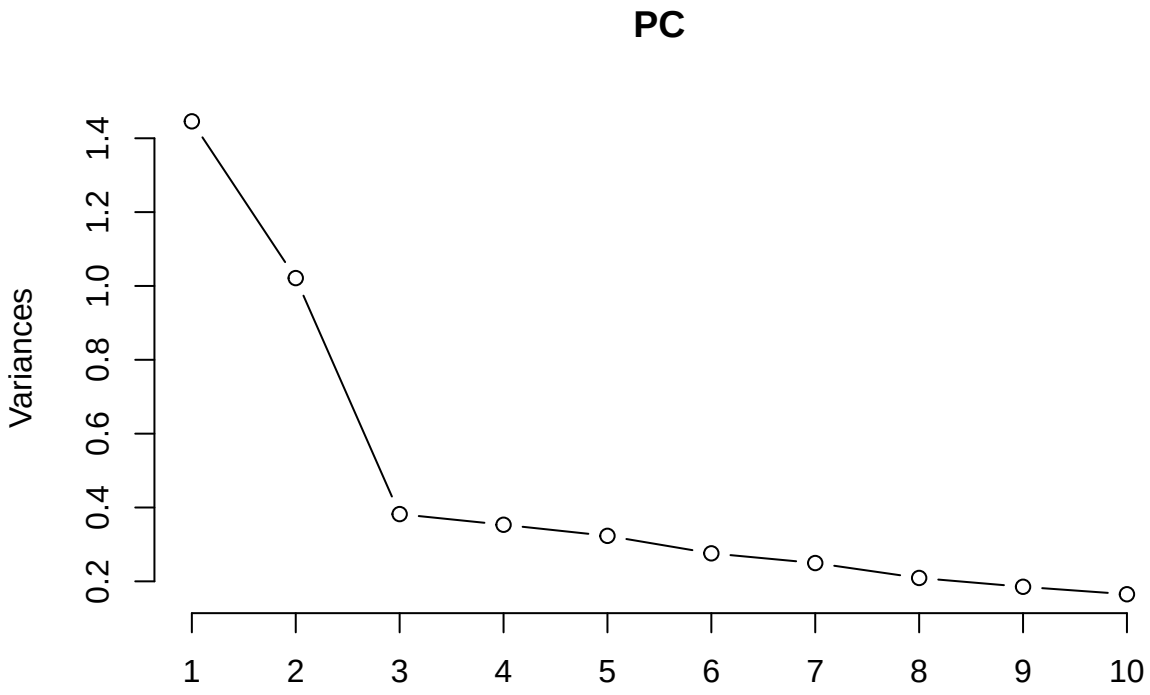
16 PC's to explain 80% of the data 27 PC's to explain 90% of the data 38 PC's to explain 95% of the data and 74 PC's to explain 99% of the data

```
head(cs,n=41)
```

```
##  [1] 21.68330 37.00021 42.73075 48.02888 52.87641 57.01319 60.75559
##  [8] 63.89269 66.67107 69.14649 71.35842 73.38752 75.23794 76.92553
## [15] 78.55452 79.99232 81.35365 82.60465 83.72280 84.73428 85.72922
## [22] 86.64216 87.49930 88.33457 89.04680 89.75450 90.39417 91.00445
## [29] 91.55468 92.07302 92.54608 93.00783 93.42960 93.80213 94.14688
## [36] 94.46229 94.77419 95.06890 95.34688 95.59860 95.83858
```

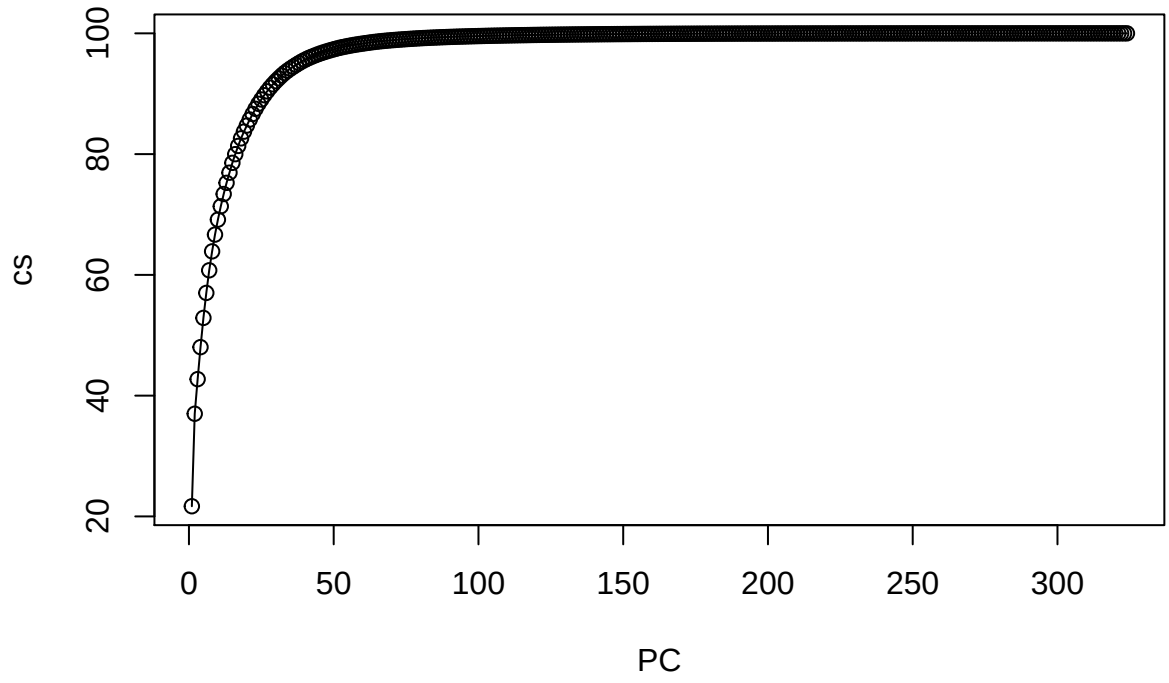
Some quick plots of variance and cumsum

```
plot(inde_pca,type="l",xlab("PC"))
```




The Cummulative variance

```
plot(cs,type="o",xlab="PC")
```

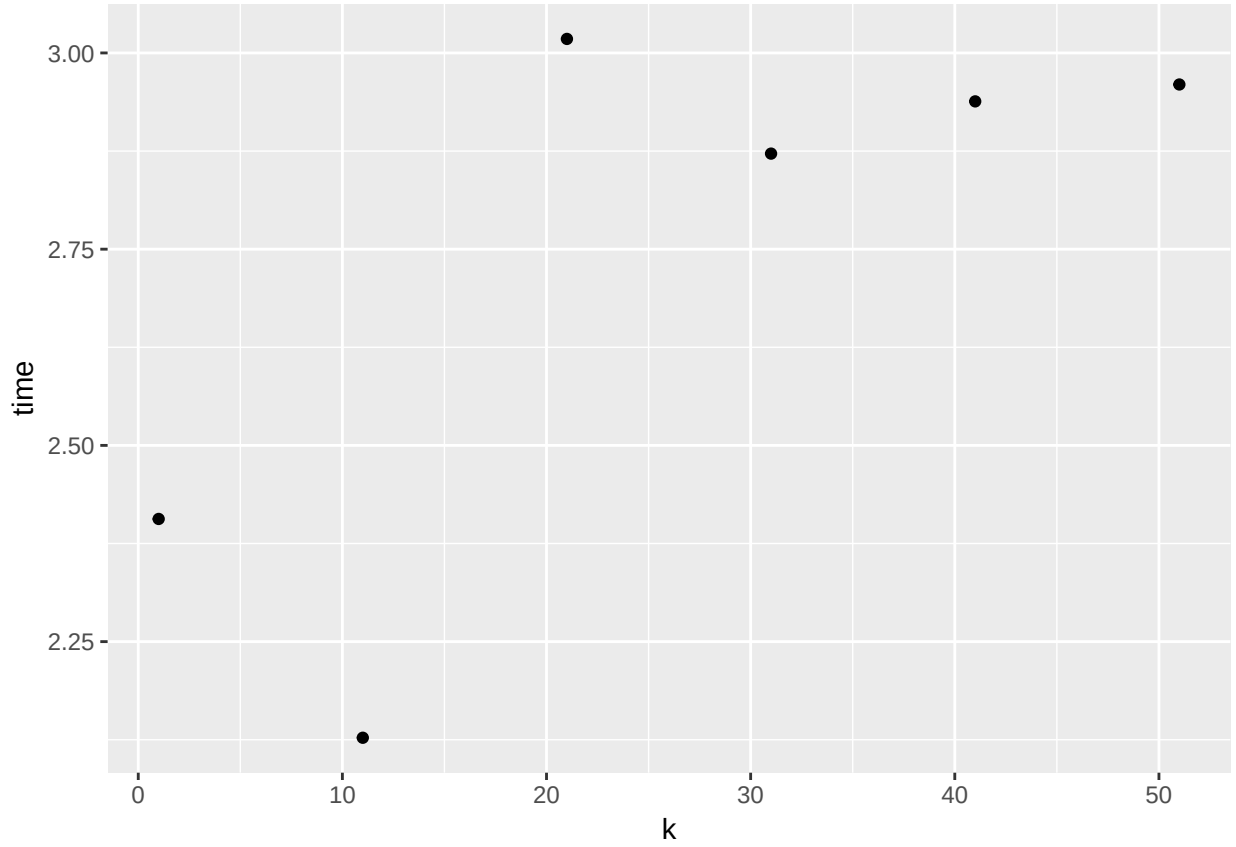



```
train = inde_pca$x
trainlabels = independent_labels[1:4000]
test = predict(inde_pca,shuffled_inde[4001:8000,])
indeAcc = c()
indeTime = c()

testlabels = independent_labels[4001:8000]
variances = c(16,27,38,74)
print(variances)
```

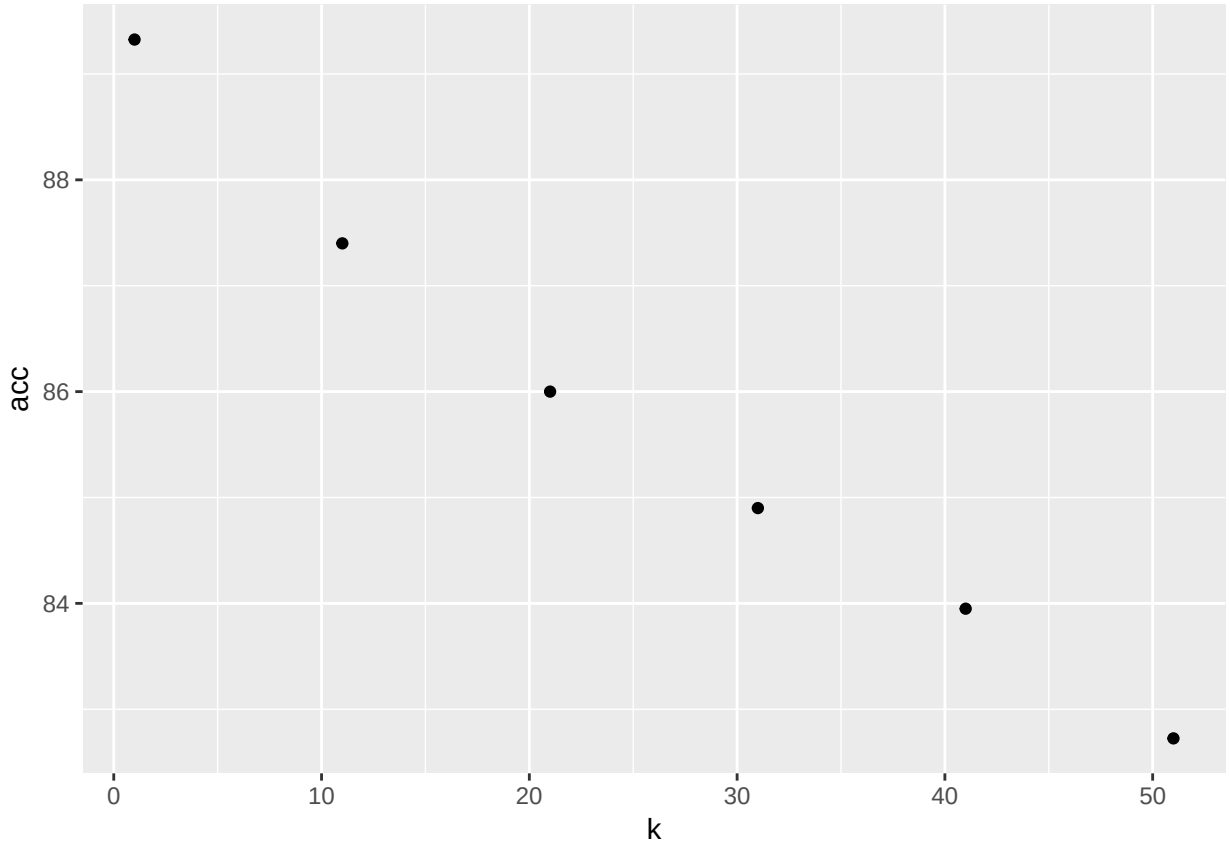
```
## [1] 16 27 38 74
```

```
k = seq(from=1,to=51,by=10)
for (j in variances){
  time = c()
  acc = c()
  for (i in k){
    startTime = Sys.time()
    results = knn(train[,1:j],test[,1:j],k=i,cl = trainlabels)
    endtime = Sys.time()
    time=c(time,(endtime-startTime))
    acc = c(acc,accuracy(results,testlabels))
  }

  indeTime = c(indeTime,time[1])
  indeAcc = c(indeAcc,acc[1])
}
```

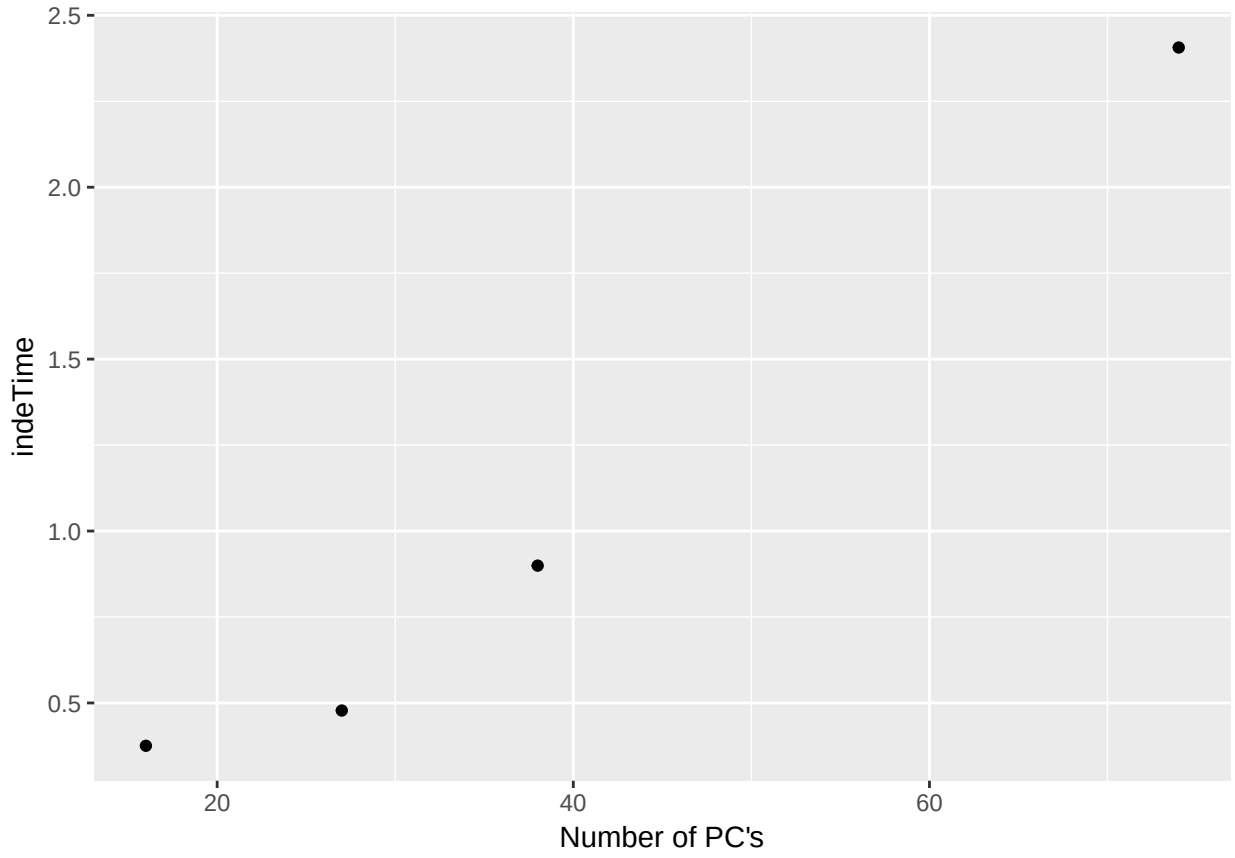
```
qplot(k,time)
```



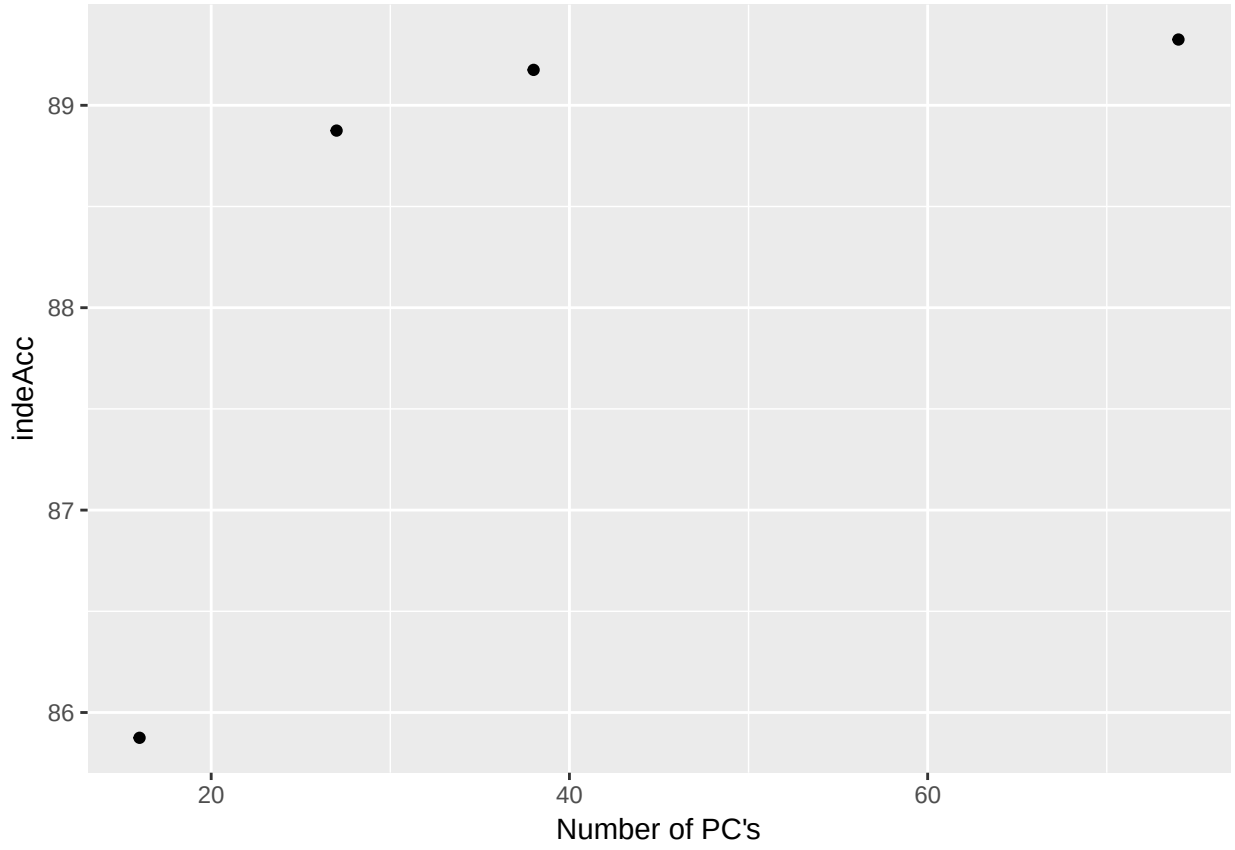

```
qplot(k,acc)
```




```
qplot(variances,indeTime,xlab ="Number of PC's")
```




```
qplot(variances,indeAcc,xlab ="Number of PC's")
```




From the plots we can see that k = 1 provides the best accuracy and time, so that k will be used for the future. we can also see that time taking for amount of PC's rises linearly while the prediction accuracy flats out the more PC's are added. therefore i have chosen to go with 38 pc's for the future

no to do it with person dependent data. This section hasn't provided great answers as i chose to only use 2 persons data to save computational time. this could explain the low percentage of correct predictions due to us writing digits fairly different

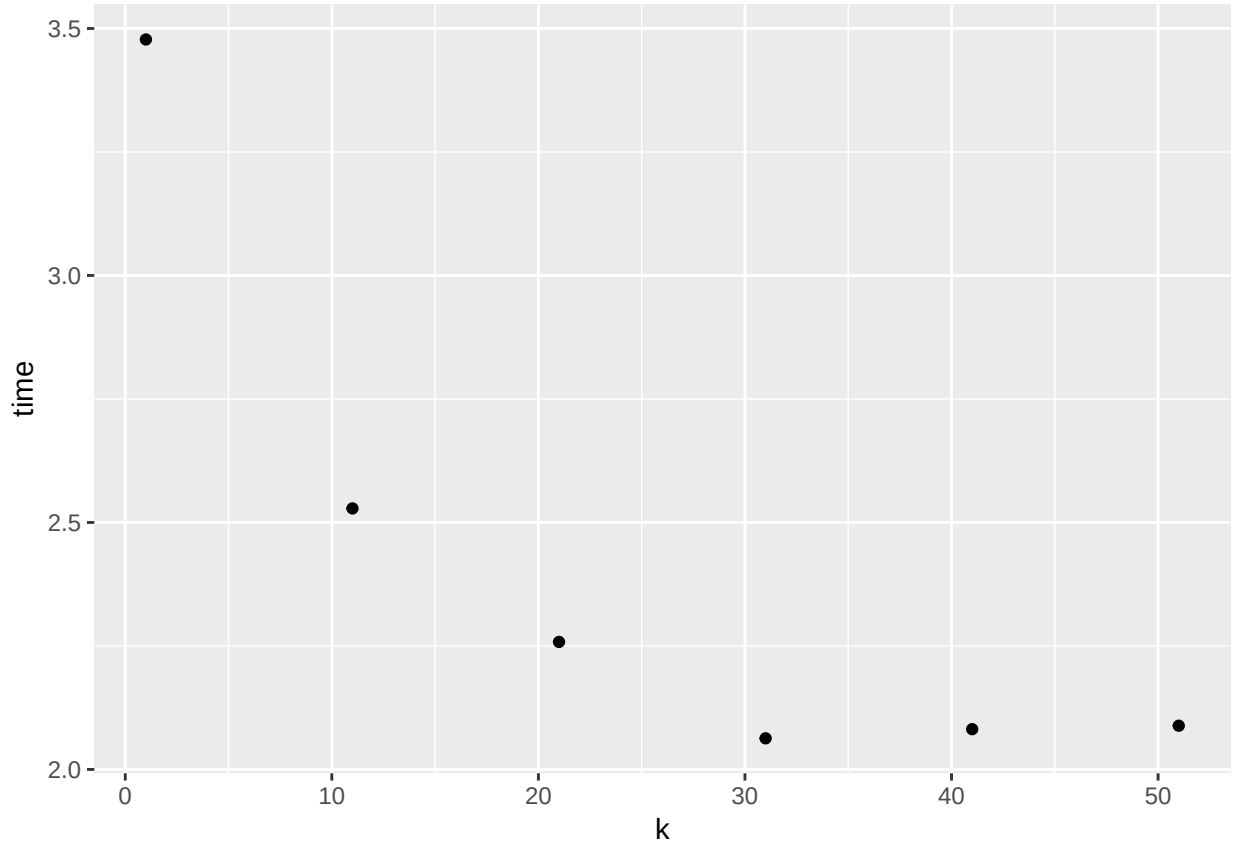
```
p1_pca <- prcomp(p1_shuff)
vari = (((p1_pca$sdev)^2)*100)/sum((p1_pca$sdev)^2)
```

```
train = p1_pca$x
trainlabels = p1_labels
test = predict(p1_pca,p2_shuff)
pacc = c()
ptime = c()

testlabels = p2_labels
variances = c(18,29,41,76)
print(variances)
```

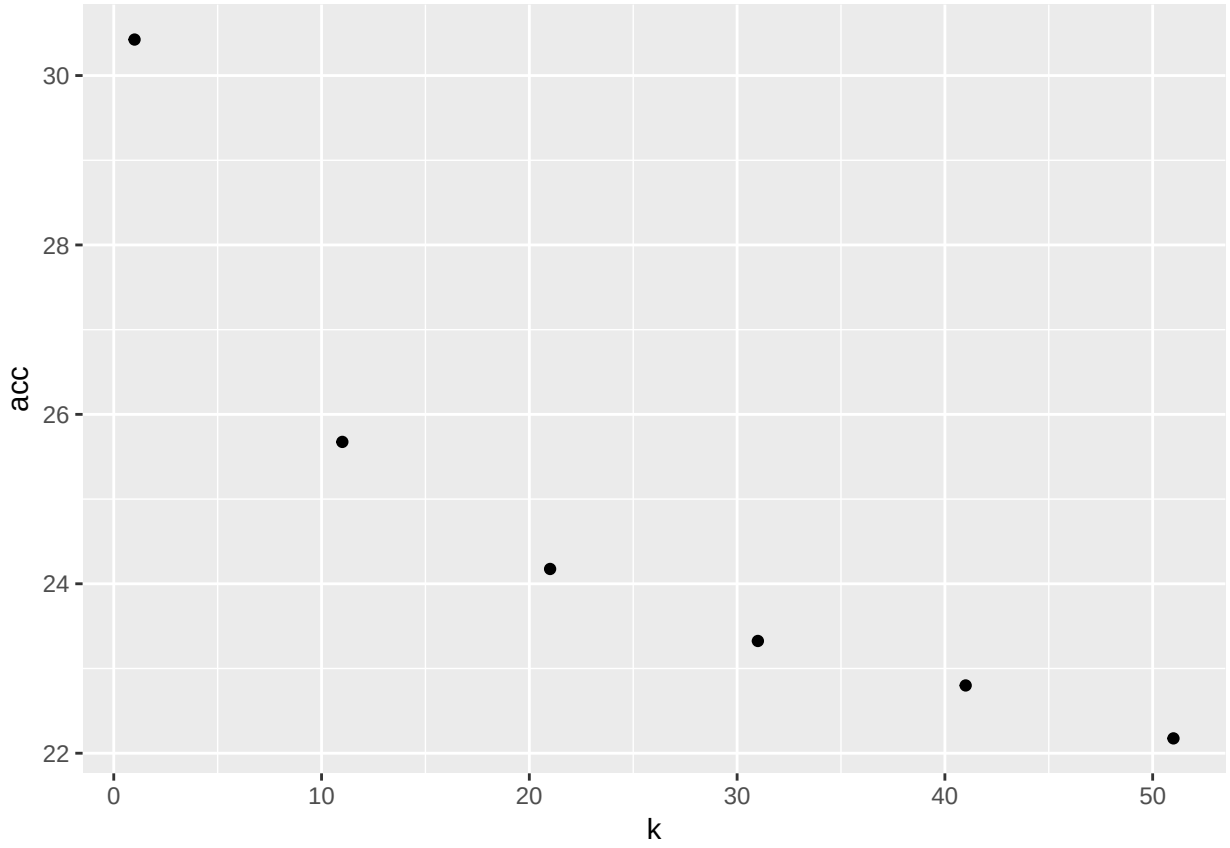
```
## [1] 18 29 41 76
```

```
k = seq(from=1,to=51,by=10)
for (j in variances){
  time = c()
  acc = c()
  for (i in k){
```

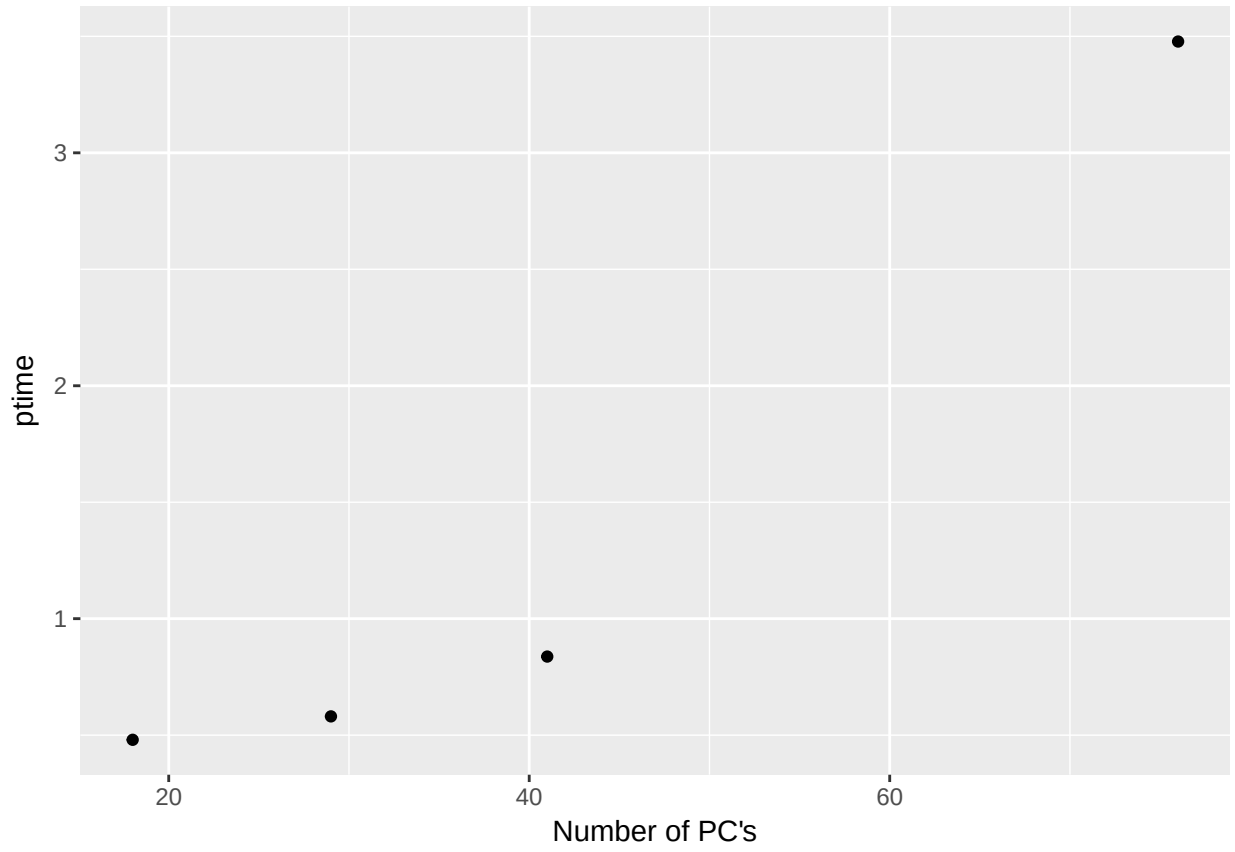
```
    startTime = Sys.time()
    results = knn(train[,1:j],test[,1:j],k=i,cl = trainlabels)
    endtime = Sys.time()
    time=c(time,(endtime-startTime))
    acc = c(acc,accuracy(results,testlabels))
  }

  qplot(k,time)
  qplot(k,acc)
  ptime = c(ptime,time[1])
  pacc = c(pacc,acc[1])
}
qplot(k,time)
```



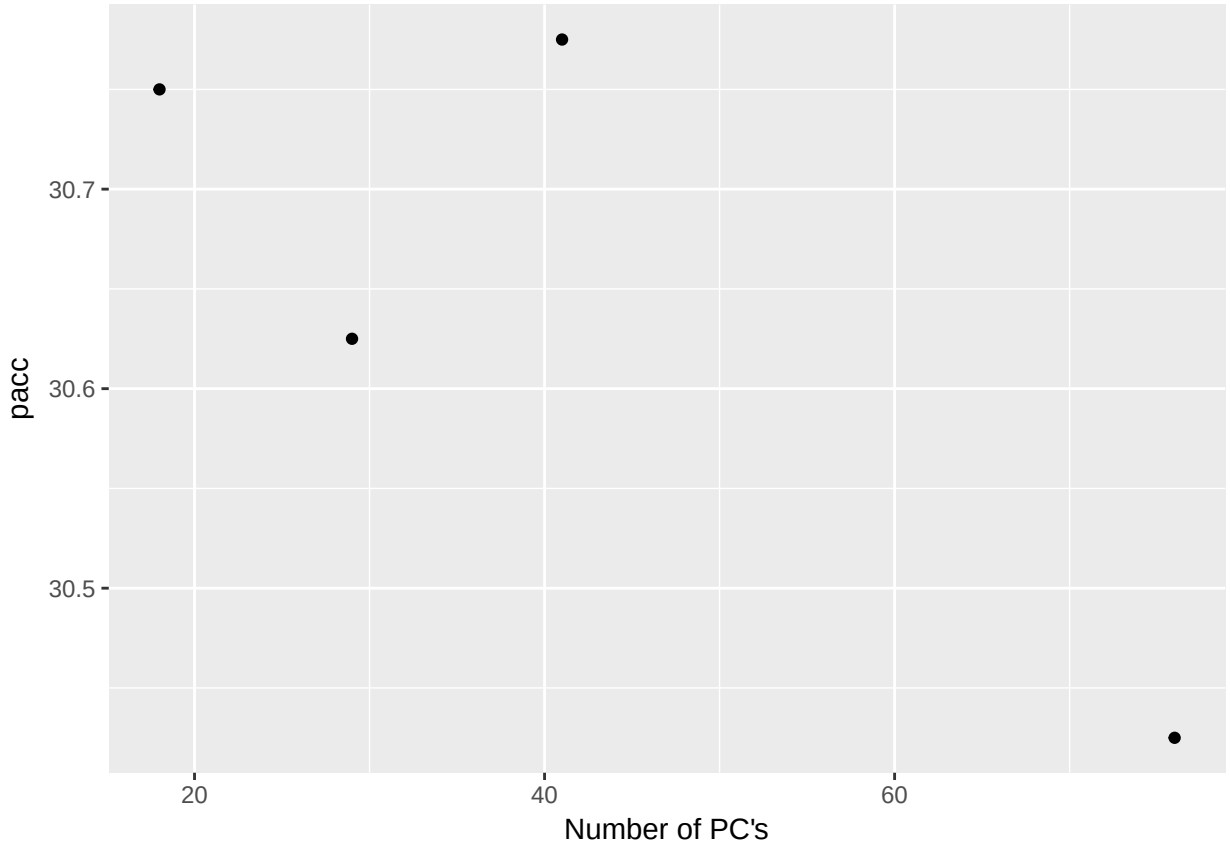

```
qplot(k,acc)
```




```
qplot(variances,ptime,xlab ="Number of PC's")
```




```
qplot(variances,pacc,xlab ="Number of PC's")
```




as the person independent data showed, time and accuracy is best at k=1, although the trend doesn't fit for the accuracy when combining more principal components, but as stated earlier, this could be due to majorly different handwriting

using scale and center arguments for normalization and standardization i decided to use the built in function for scale and centering which as far as i understood, should handle normalization and standardization of the data, i tried by applying my own functions to this, but something went wrong, and i ended up with knn only guessing at 1's.

```
indeAcc = c()
indeTime = c()


k = seq(from=1,to=51,by=10)

time = c()
acc = c()
for (i in 1:10){
  doTheShuffle()
  standard_inde = as.data.frame((shuffled_inde))
  stand_pca = prcomp(standard_inde[1:7200,], scale. = TRUE, center = TRUE)
  train = stand_pca$x
  trainlabels = independent_labels[1:7200]
  test = predict(stand_pca,shuffled_inde[7201:8000,])
  testlabels = independent_labels[7201:8000]

  startTime = Sys.time()
  results = knn(train[,1:41],test[,1:41],k=1,cl = trainlabels)
```

```
  endtime = Sys.time()
  time=c(time,(endtime-startTime))
  acc = c(acc,accuracy(results,testlabels))
  indeTime = c(indeTime,mean(time))
  indeAcc = c(indeAcc,mean(acc))
}

print(mean(indeTime))
```

```
## [1] 0.3915923
```

```
print(mean(indeAcc))
```

```
## [1] 89.61943
```

by applying this before the pca we can see that time rises a bit, while accuracy stays roughly the same

normalazing and standardizing after pca

```
indeAcc = c()
indeTime = c()

print(variances)
```

```
## [1] 18 29 41 76
```

```
k = seq(from=1,to=51,by=10)

time = c()
acc = c()
for (i in 1:10){
  doTheShuffle()
  standard_inde = as.data.frame((shuffled_inde))
  stand_pca = prcomp(standard_inde[1:7200,])
  train = scale(stand_pca$x)
  trainlabels = independent_labels[1:7200]
  test = scale(predict(stand_pca,shuffled_inde[7201:8000,]))
  testlabels = independent_labels[7201:8000]

  startTime = Sys.time()
  results = knn(train[,1:41],test[,1:41],k=1,cl = trainlabels)

  endtime = Sys.time()
  time=c(time,(endtime-startTime))
  acc = c(acc,accuracy(results,testlabels))
  indeTime = c(indeTime,mean(time))
  indeAcc = c(indeAcc,mean(acc))
}

print(mean(indeTime))
```

```
## [1] 0.3881624
```

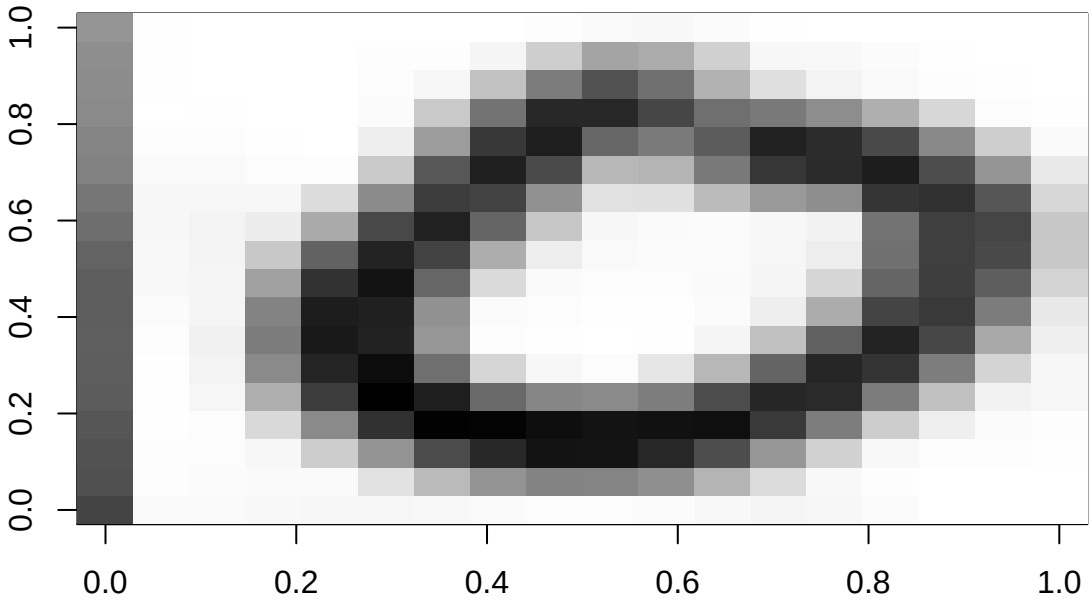
```
print(mean(indeAcc))
```

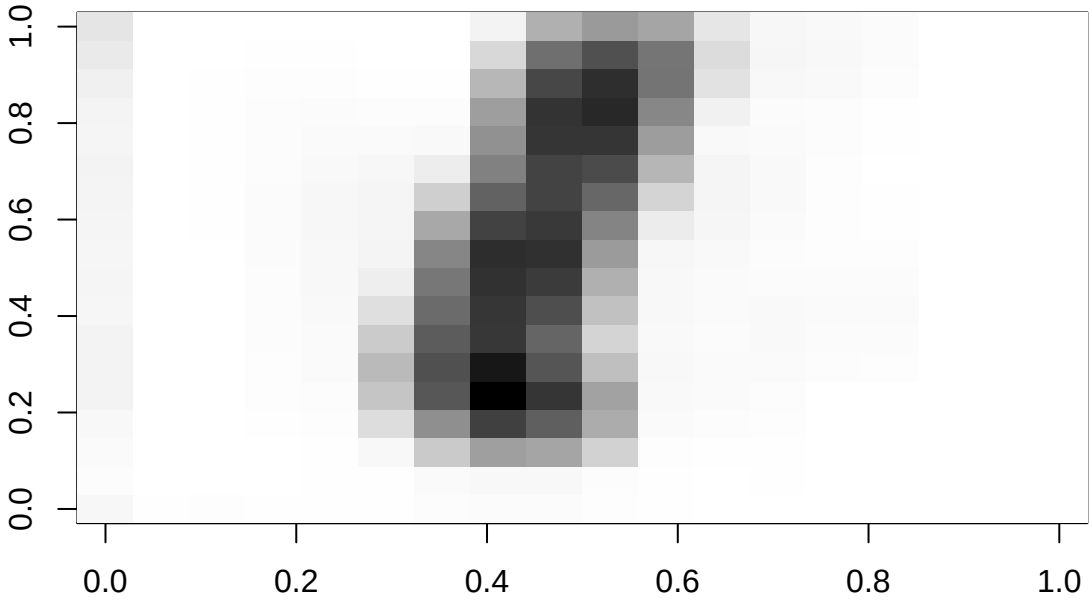
```
## [1] 88.31205
```

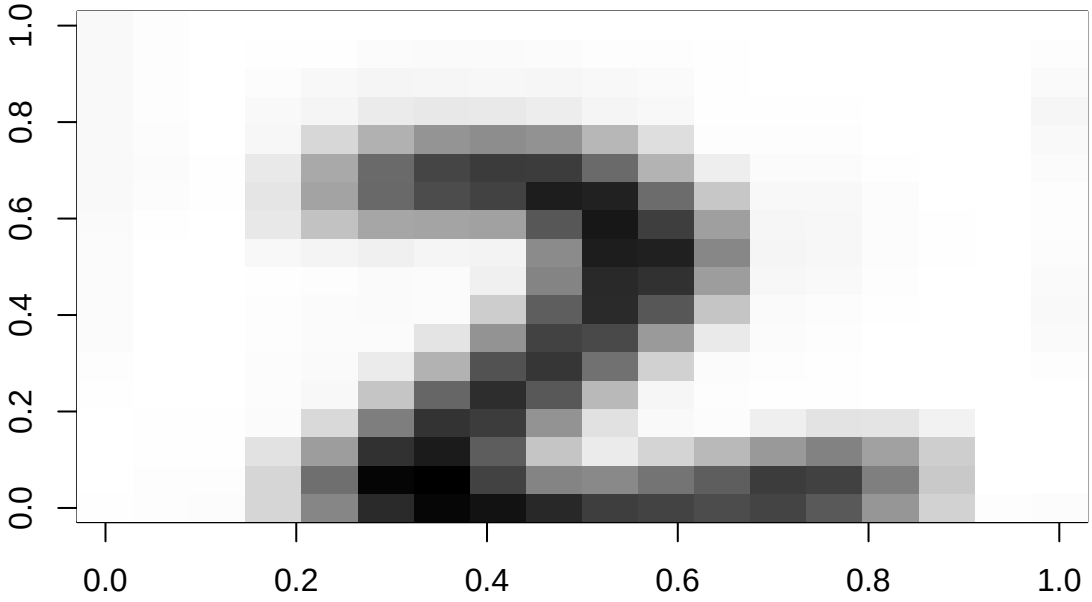
when applying after pca, we can see that we loose a small amount of accuracy, while time taken also rises.

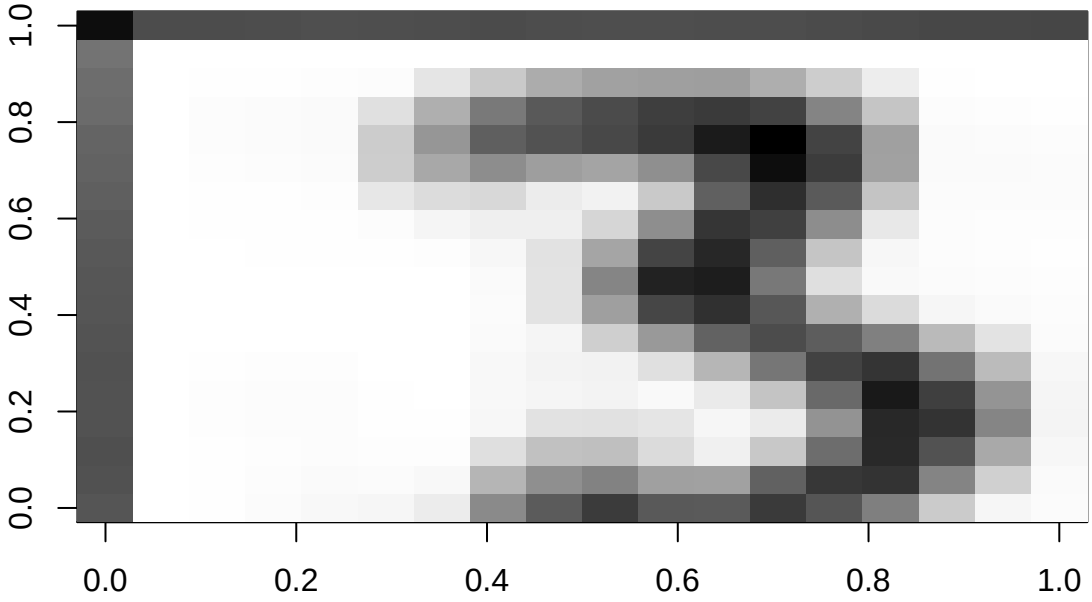
Reconstruction

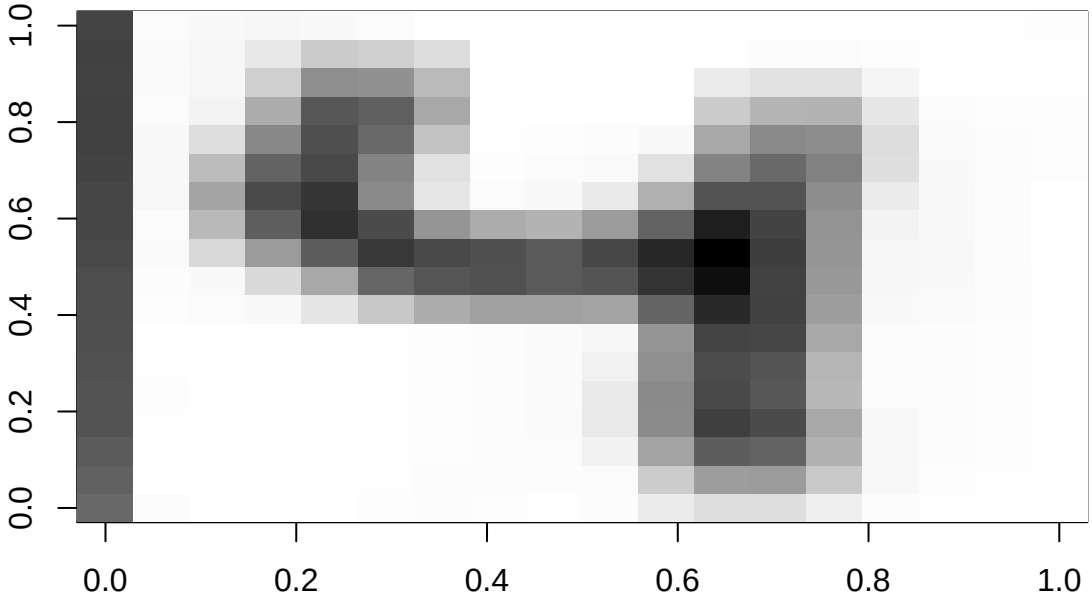
```
for (i in 0:9){
  imageSize = sqrt(ncol(p1)-1)
  imageMatrix <- matrix( p1[1+i*400,2:ncol(p1)], nrow = imageSize, ncol=imageSize, byrow= FALSE)
  imageMatrix <- rotate(imageMatrix, 270)
  image(imageMatrix, col=gray((0:255)/255))

}
```

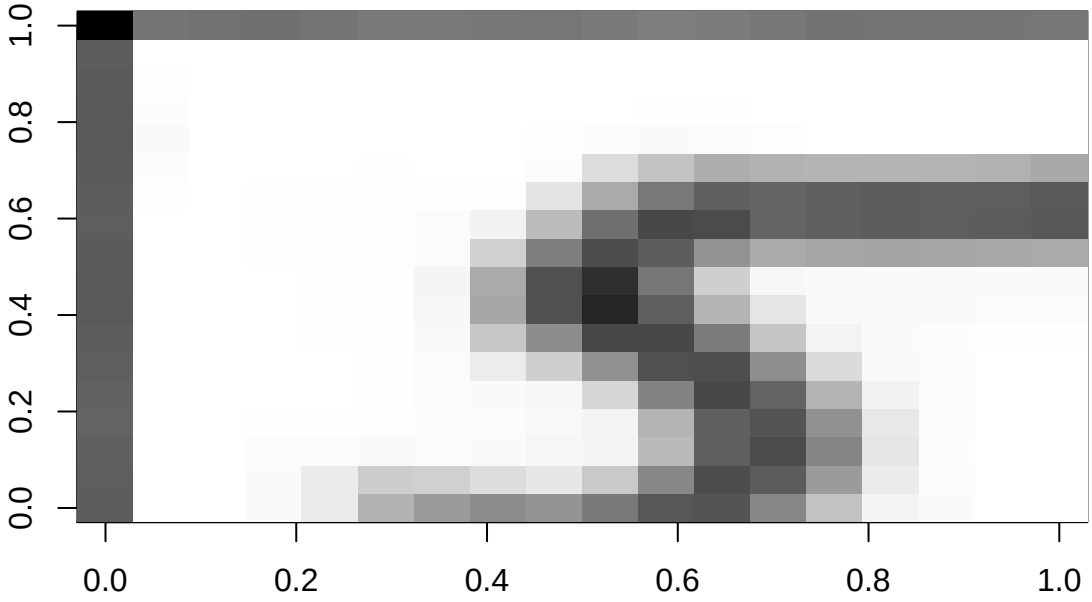


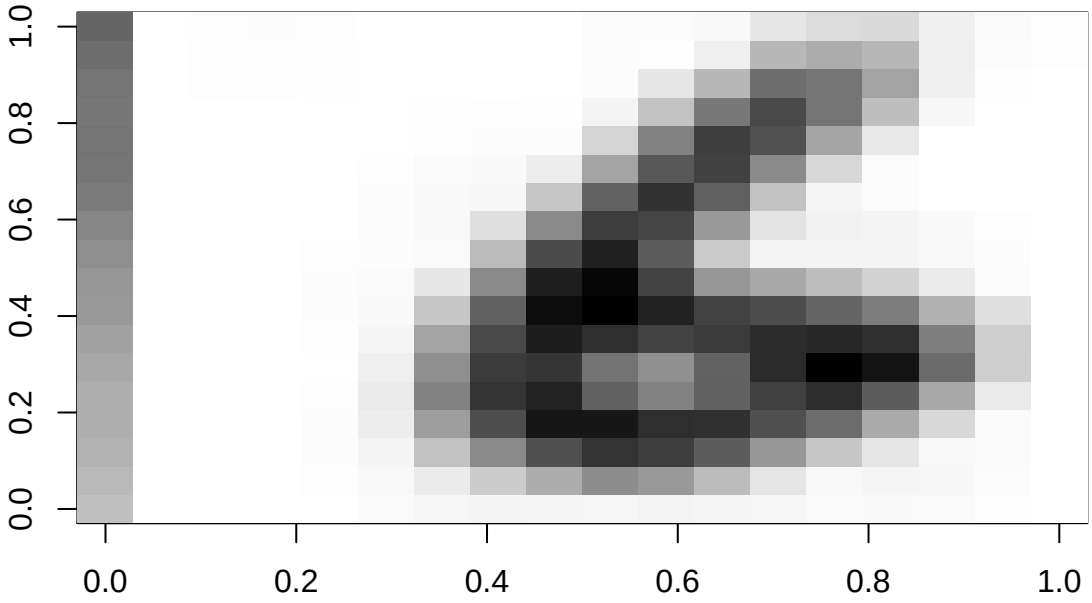

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

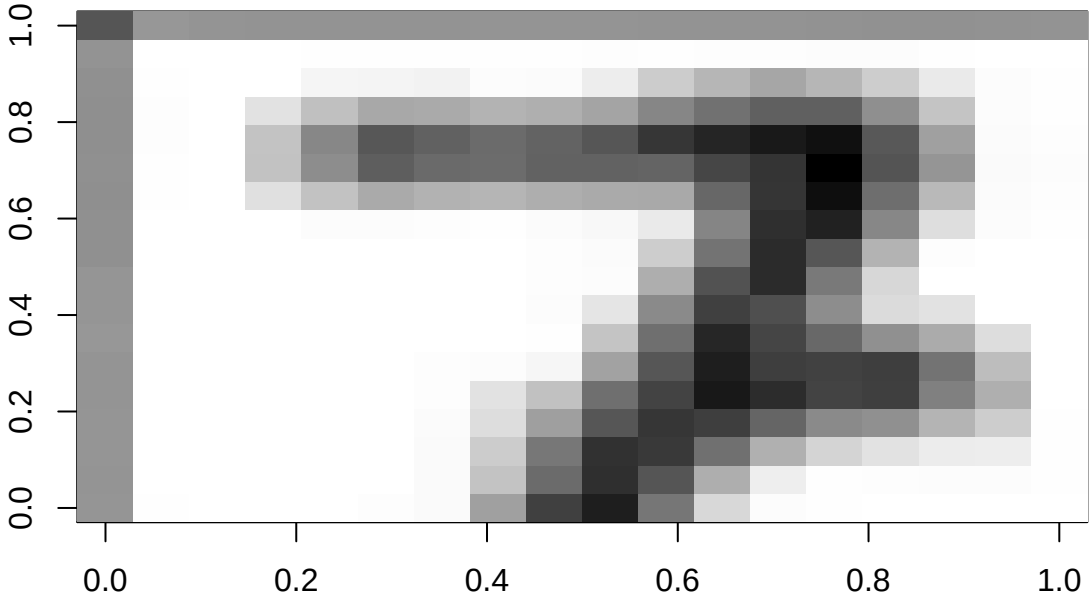

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

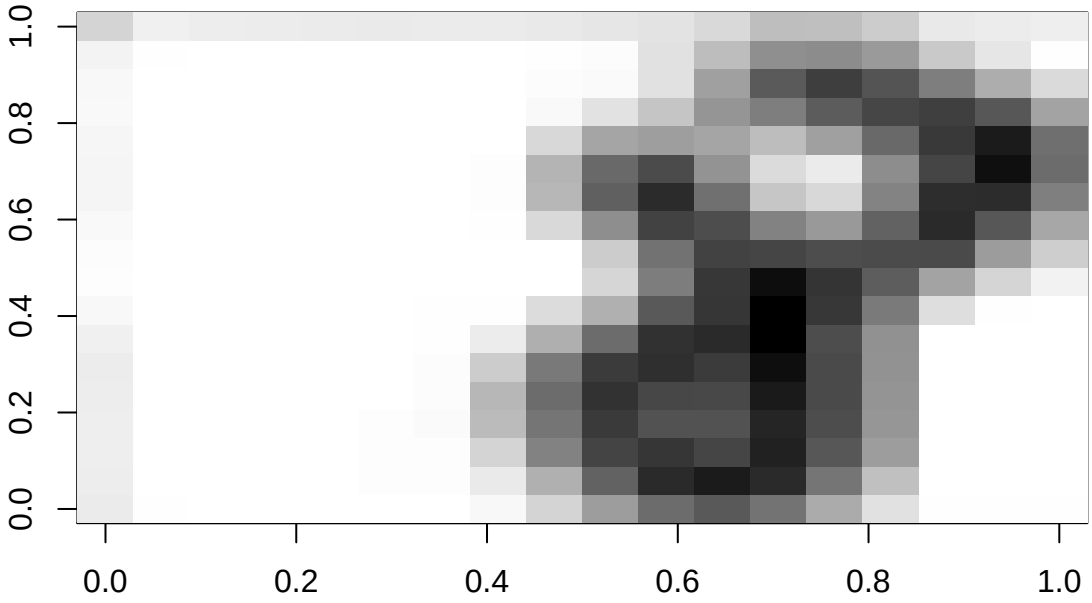

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

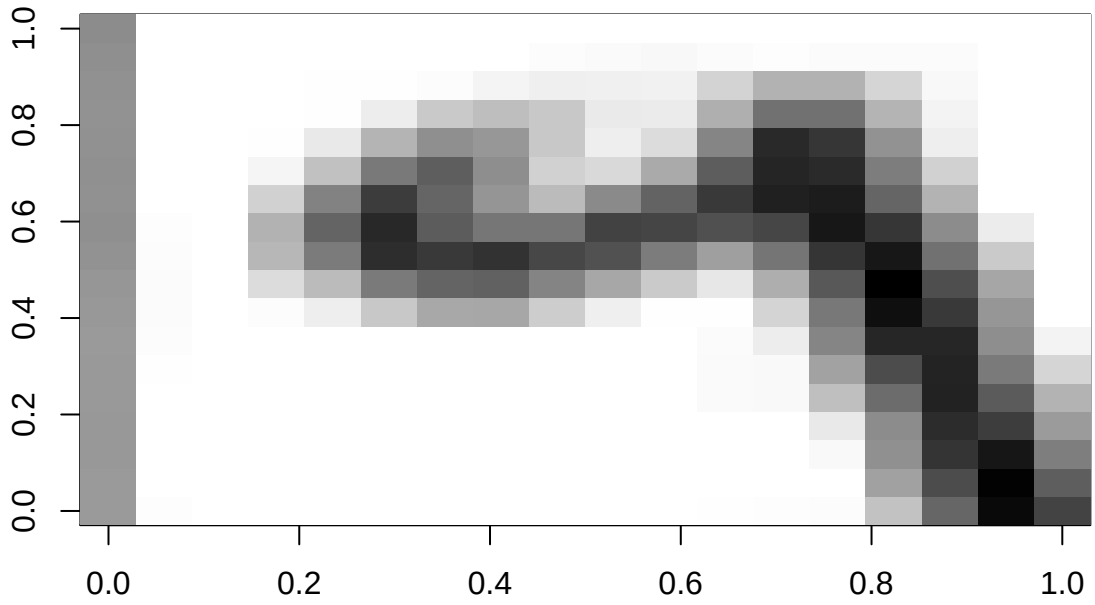

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0



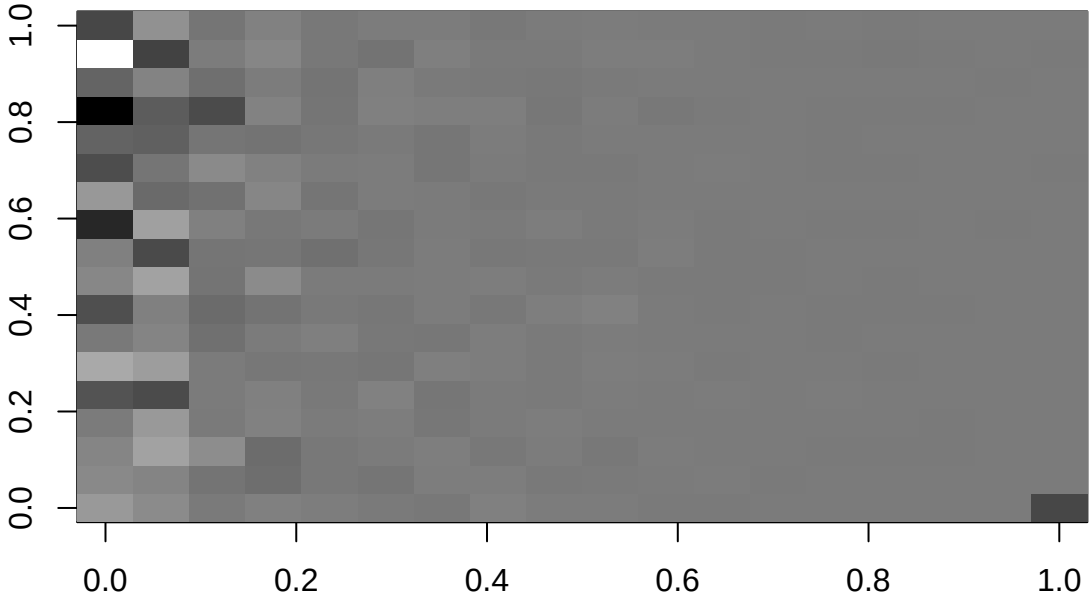

When printing out the images, it can be seen that the corners might not have been set properly, which could provide more details at to why some accuracy percentages have been low.

```
ncol(p1_pca$x)-1
```

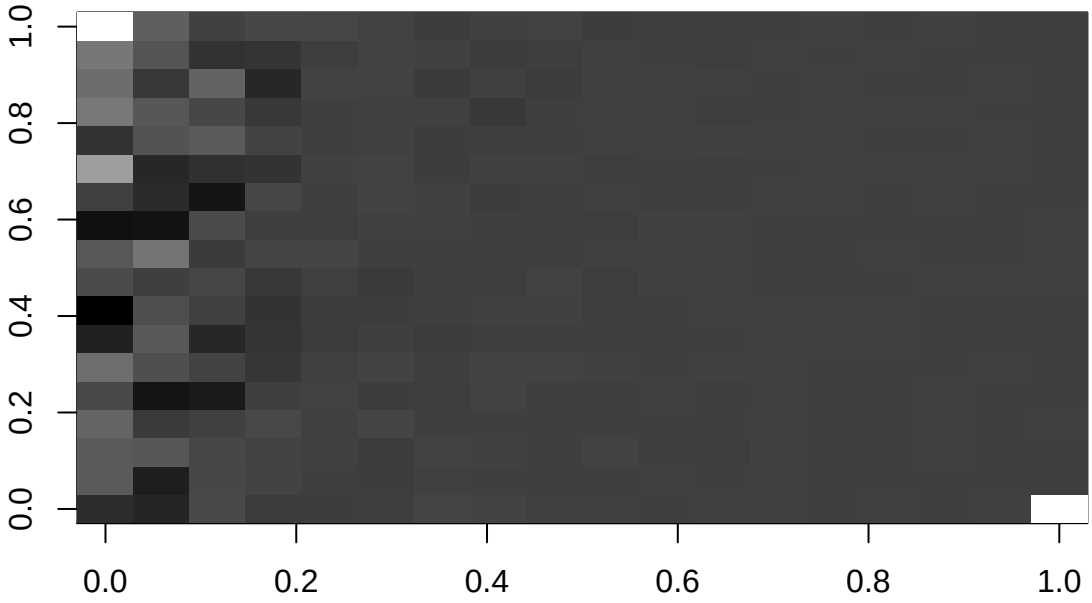
```
## [1] 323
```

```
for (i in 1:10){
  imageSize = 18
  imageMatrix <- matrix( p1_pca$x[i*399+i,1:nrow(p1_pca$rotation)-1], nrow = imageSize, ncol=imageSize,
  imageMatrix <- rotate(imageMatrix, 270)
  image(imageMatrix, col=gray((0:255)/255))

}
```

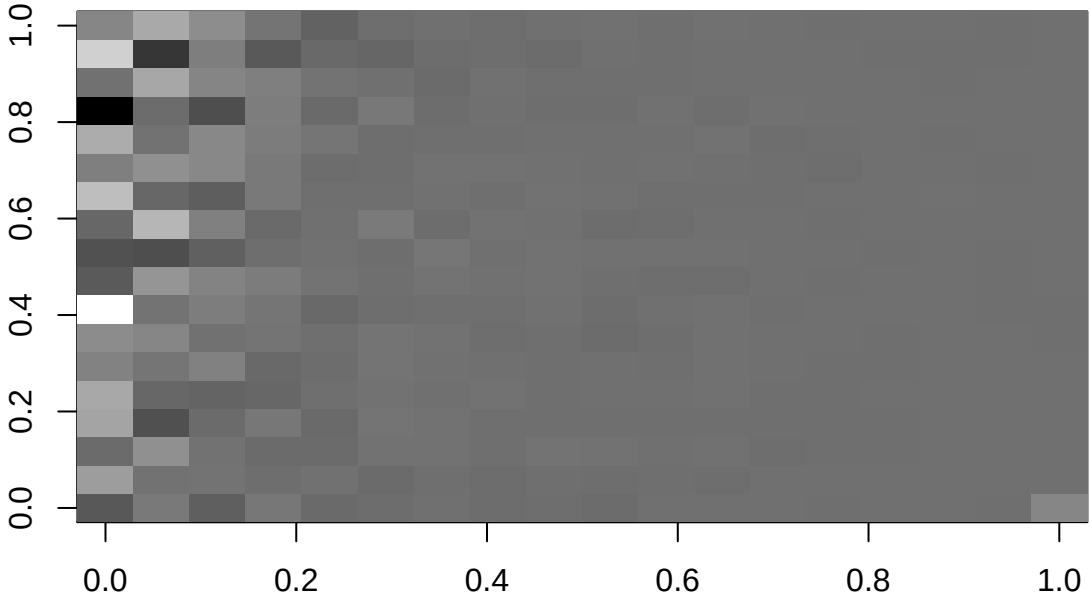
```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]

## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]
```



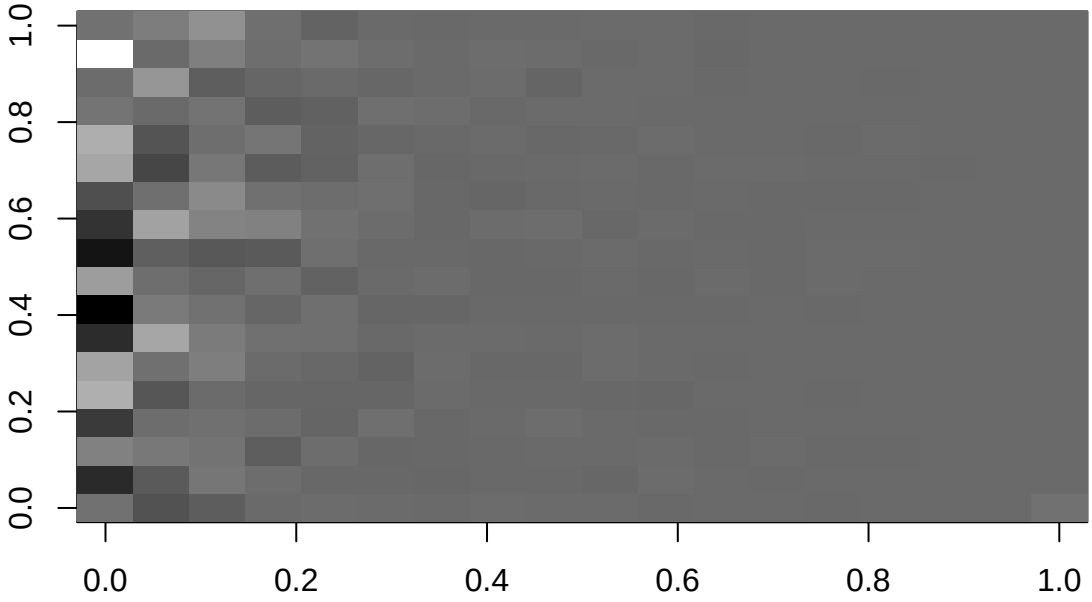

```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]
```



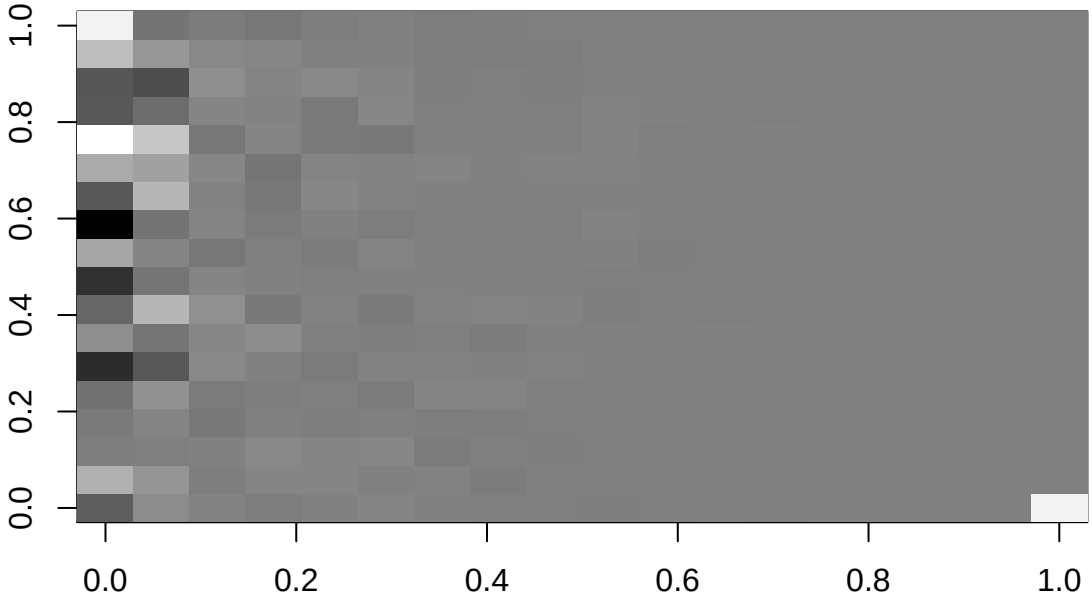

```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]
```



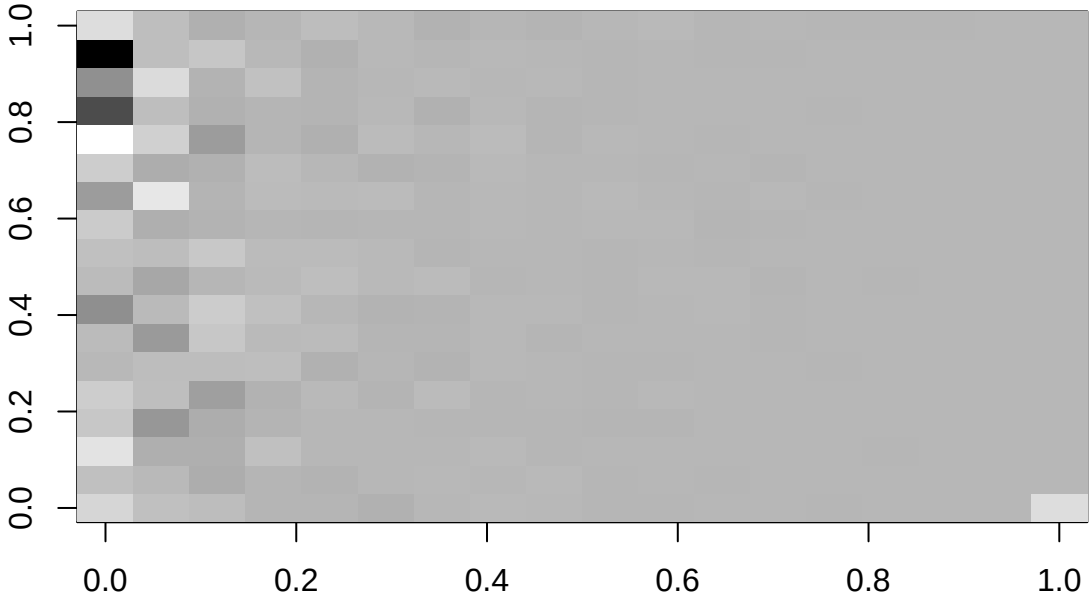

```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]
```



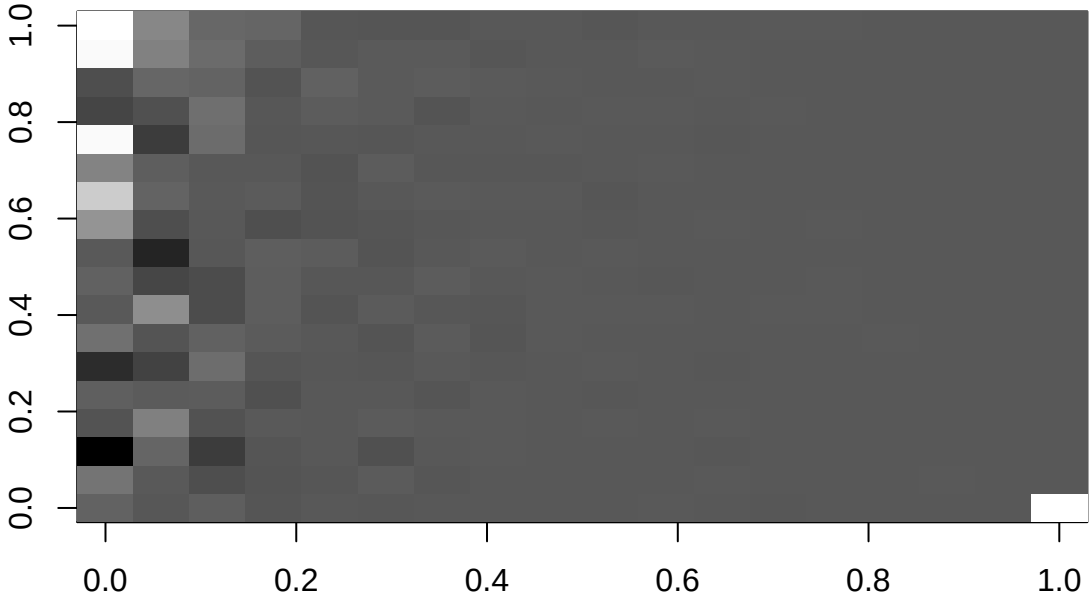

```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]
```



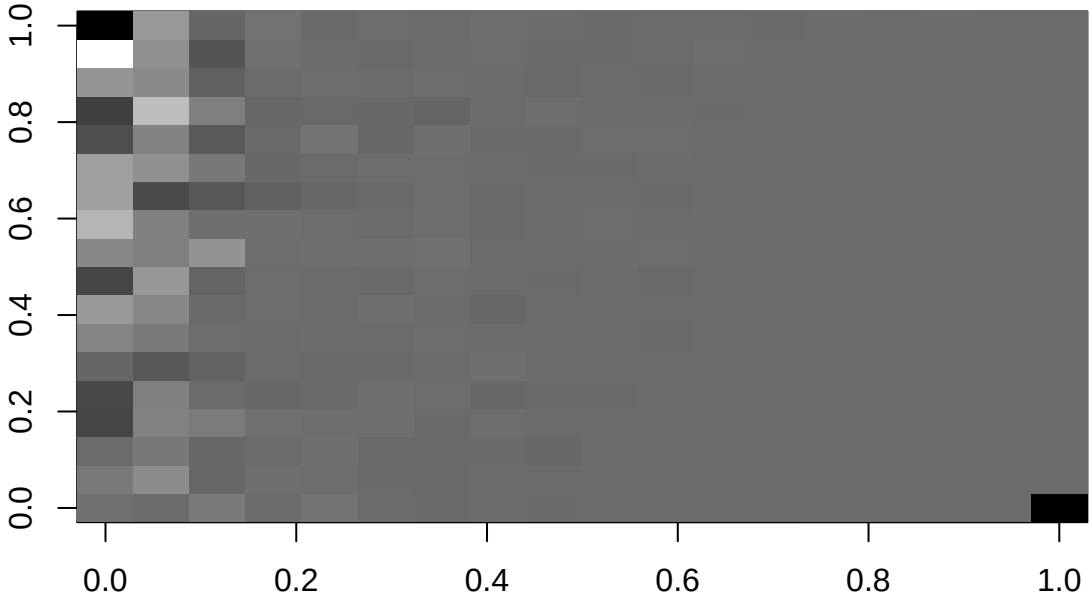

```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]
```



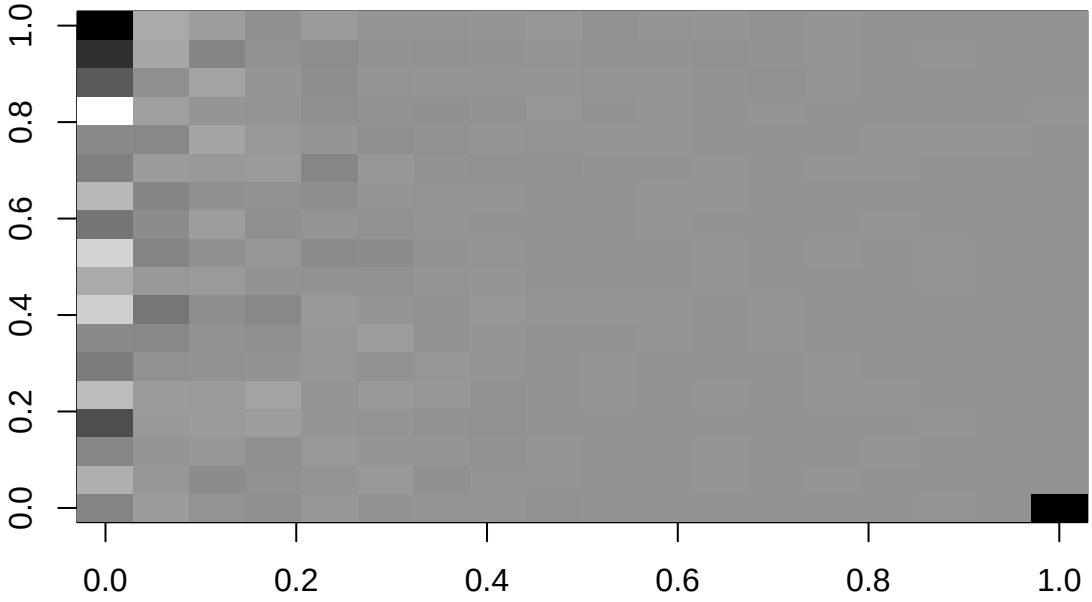

```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]
```

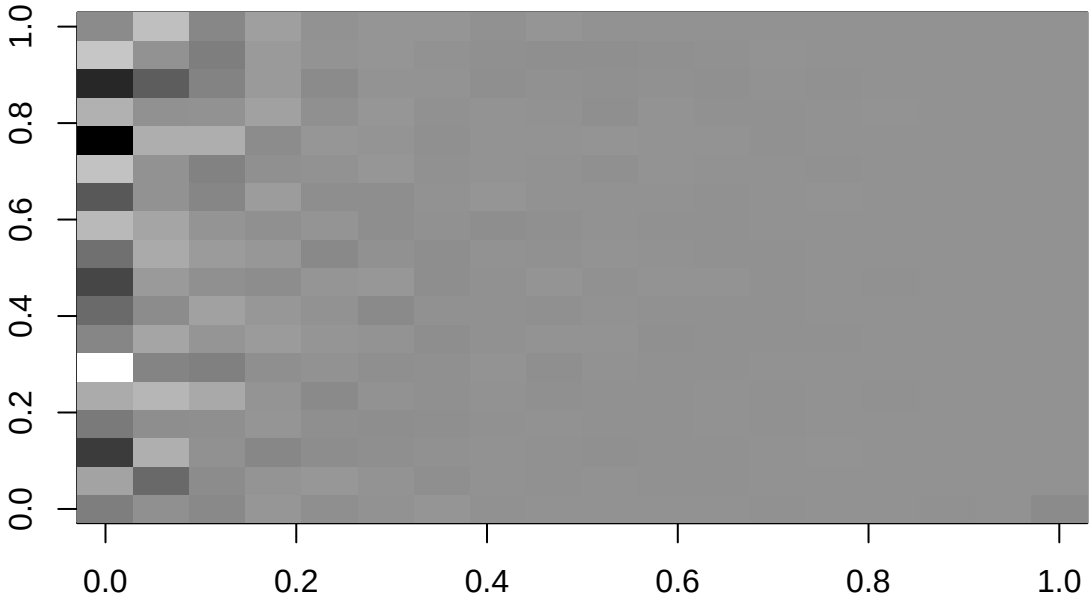



```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]
```




```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]
```

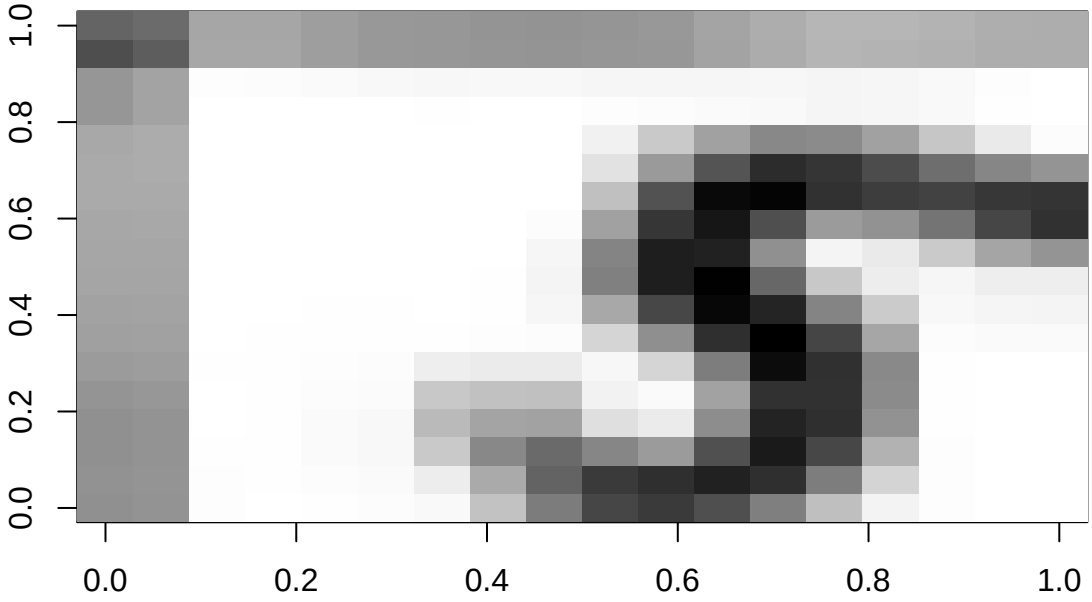

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

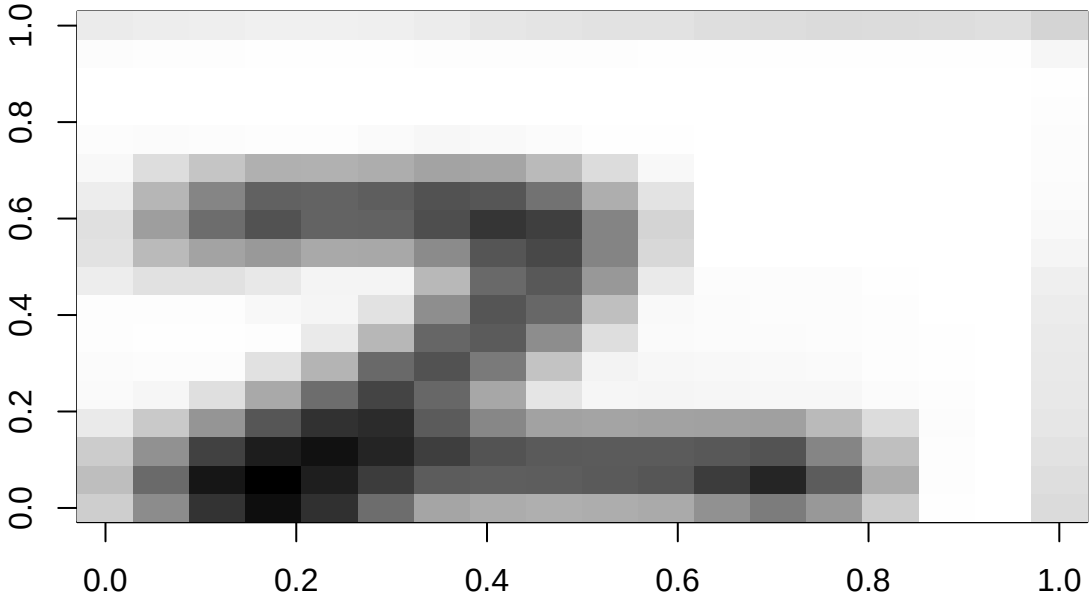


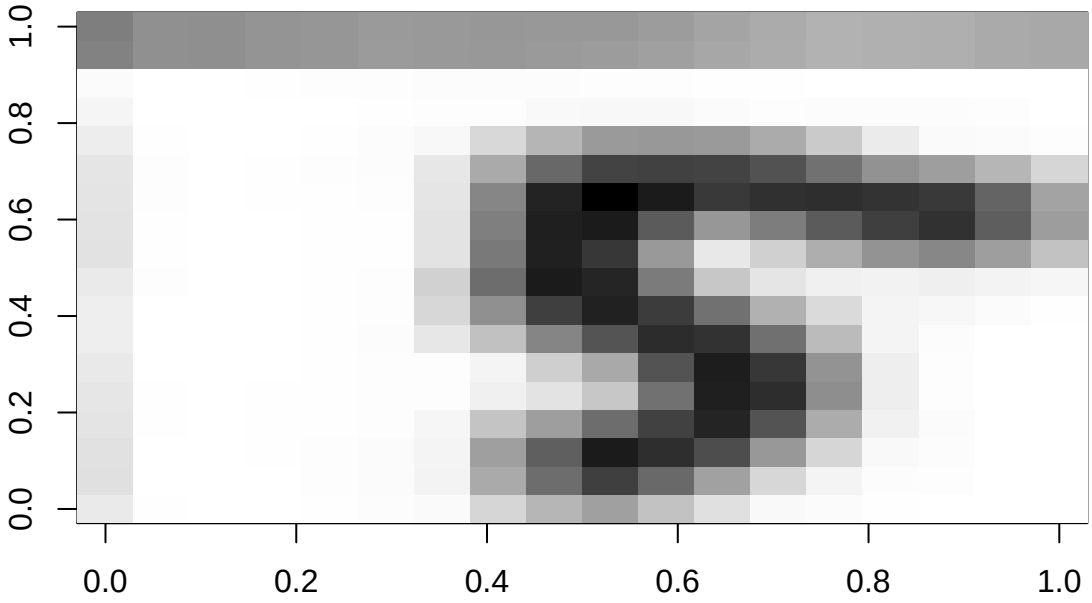

To be honest, i have no idea what's happening here, or if i'm doing it right but i'm guessing it's PCA trying to do it's own thing

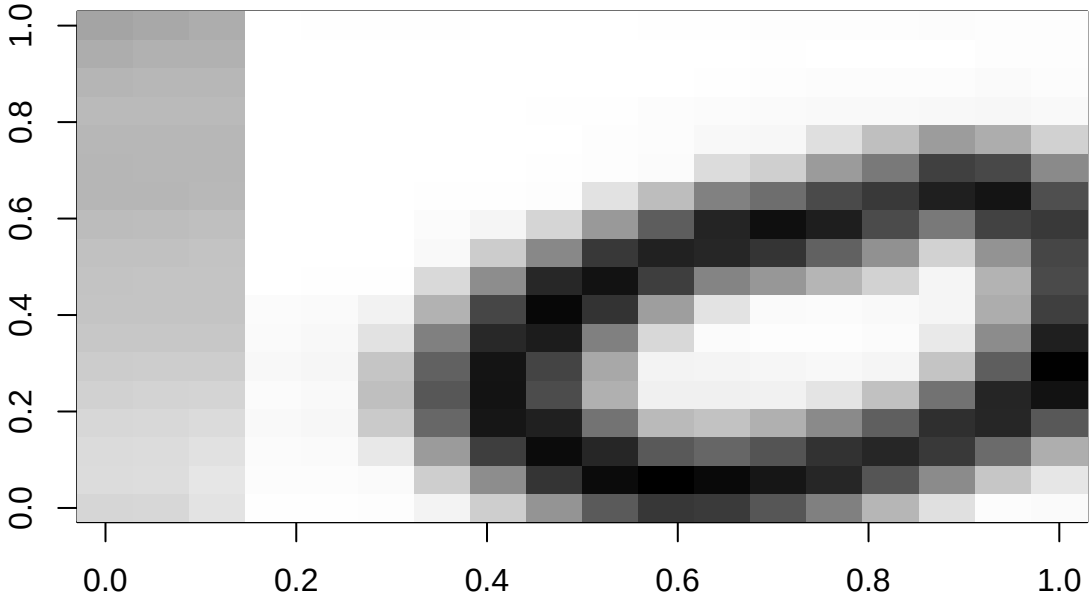
When reconstructing the image from the principal components, we can see the numbers starting to reappear a bit blurry, but for some reason i can't explain, the images keep getting darker.

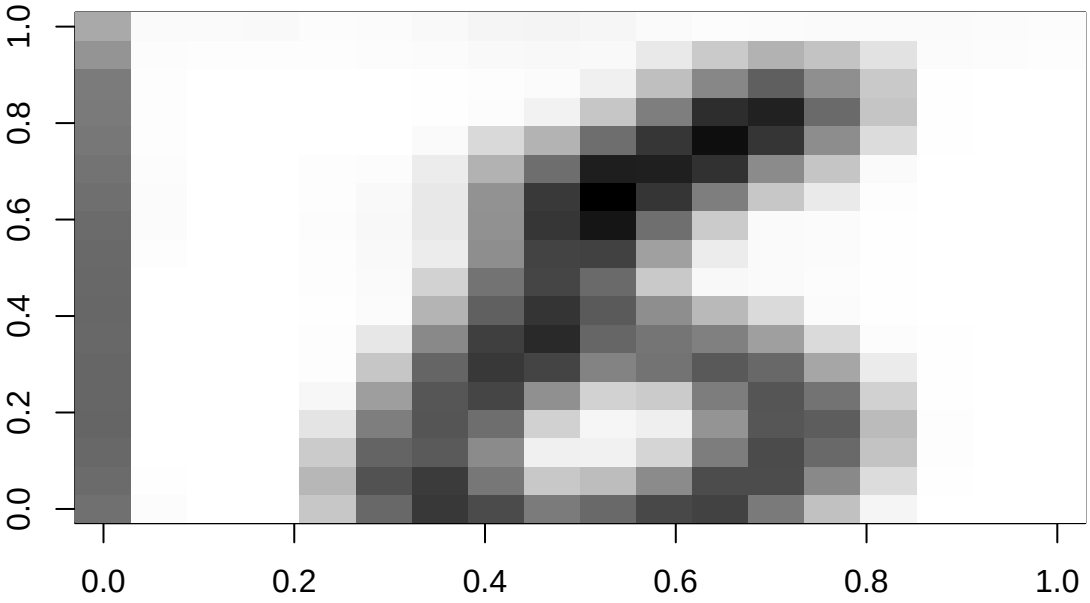
```
for (i in 1:10){

  reco = p1_pca$x[,1:nrow(p1_pca$rotation)] %*% t(p1_pca$rotation[,1:nrow(p1_pca$rotation)])
  reco <- scale(reco, center = -1 * p1_pca$center, scale=FALSE)
  dim(reco)
  imageSize = 18
  imageMatrix <- matrix(reco[i*399+i,1:ncol(reco)], nrow = imageSize, ncol=imageSize, byrow= FALSE)
  imageMatrix <- rotate(imageMatrix, 270)
  image(imageMatrix, col=gray((0:255)/255))

}
```

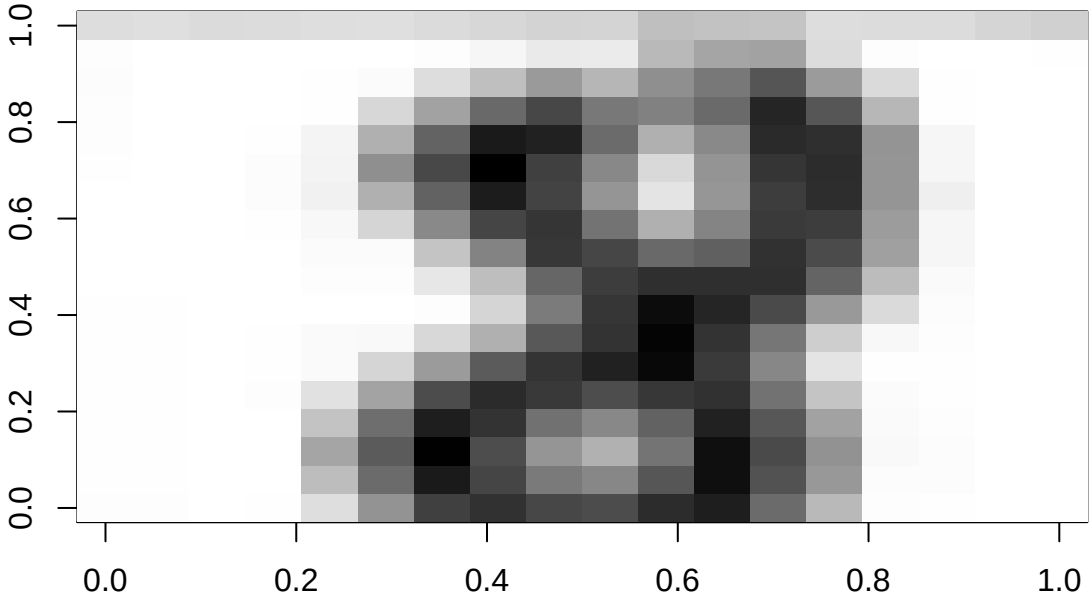

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

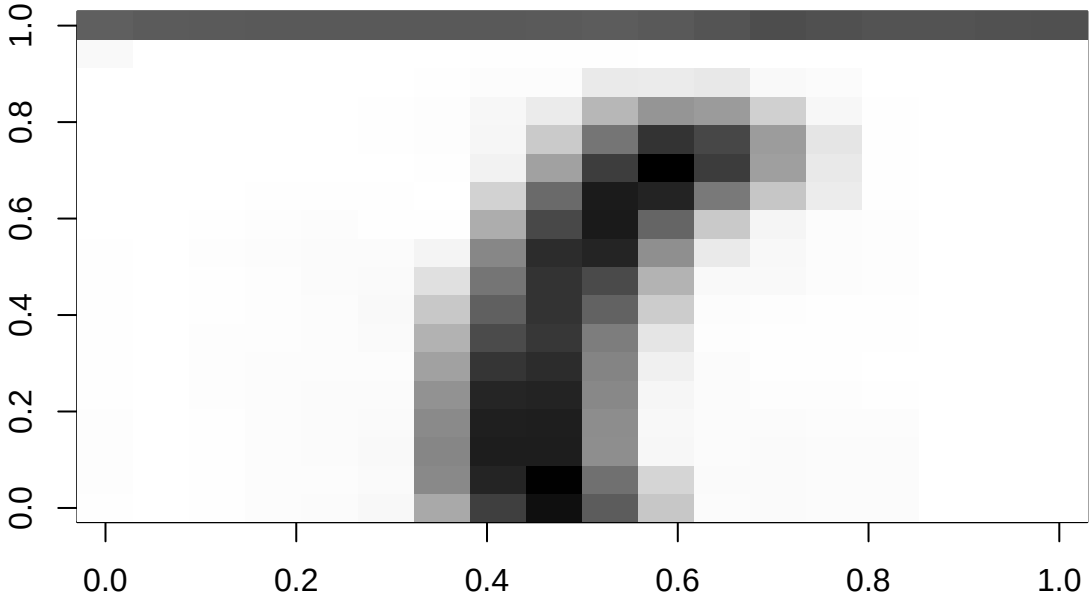

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

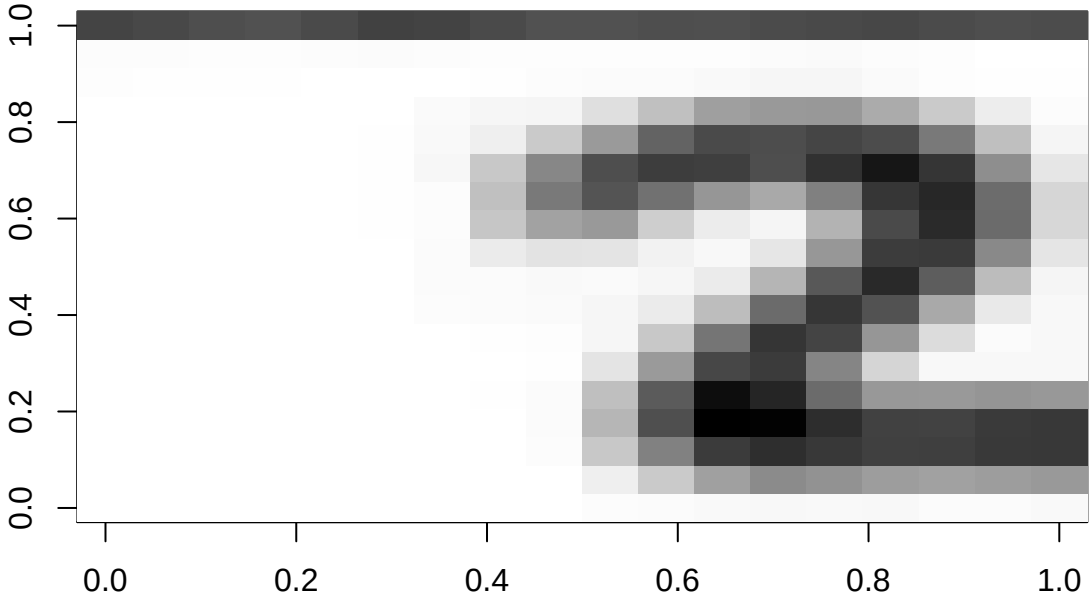

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

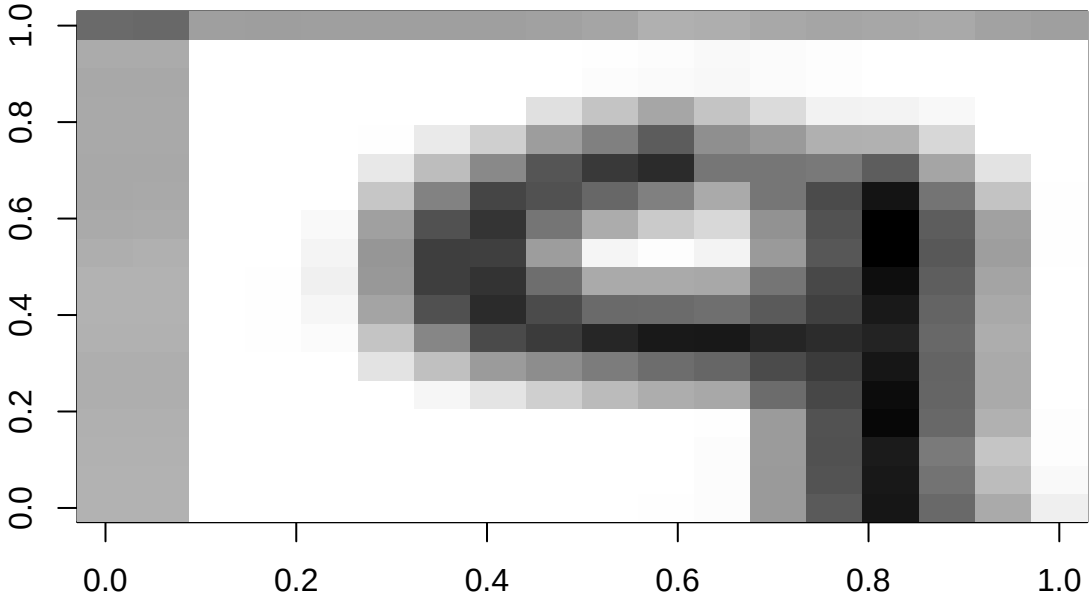

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

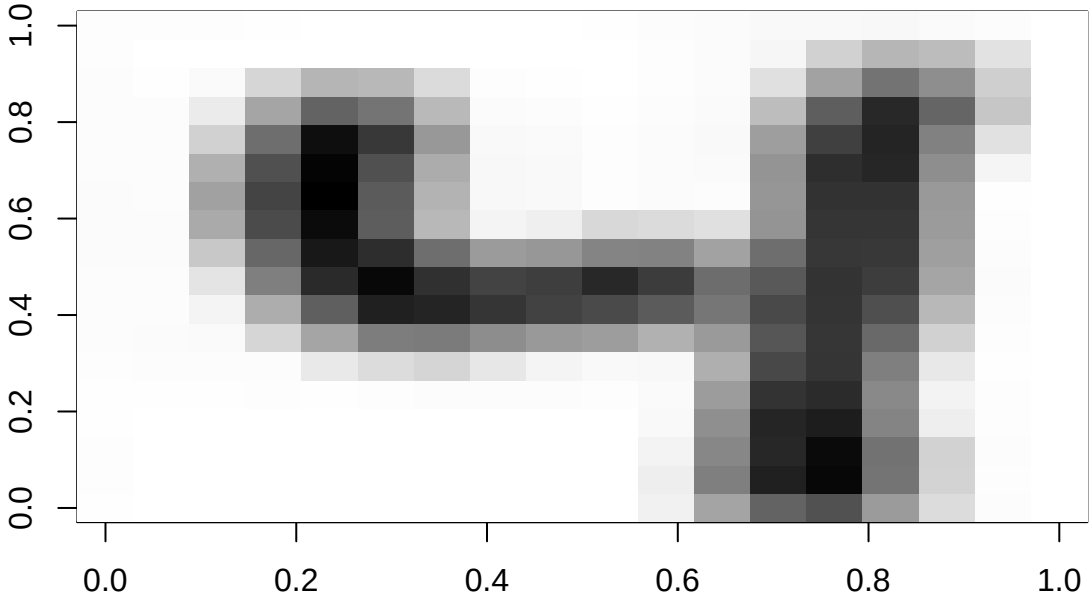

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0