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] 1.1822444 0.9994626 0.6116057 0.5782997 0.5619427 0.5224495 0.4927114
##  [8] 0.4549738 0.4225581 0.4058705
```

Variance

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

```
##  [1] 21.382521 15.281906  5.722521  5.116233  4.830904  4.175735  3.713894
##  [8]  3.166775  2.731601  2.520110
```

Cumsum Variance

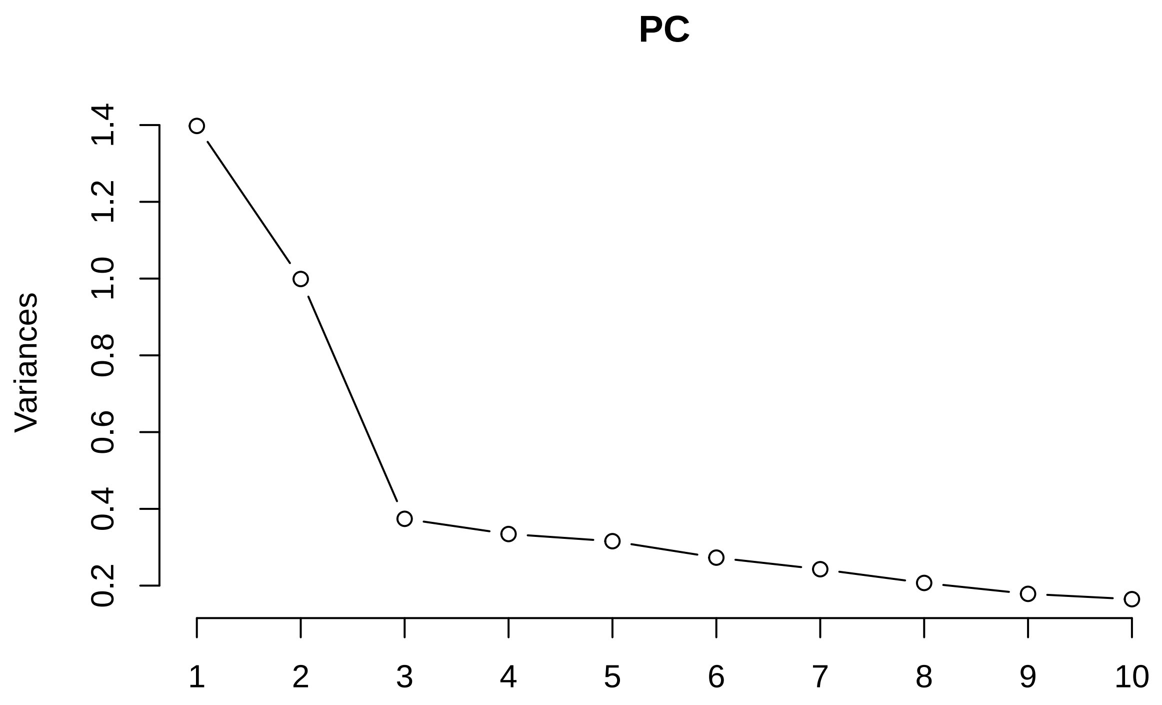
16 PC's to explain 80% of the data 27 PC's to explain 90% of the data 38 PC's to explain 95% of the data and 74 PC's to explain 99% of the data

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

```
##  [1] 21.38252 36.66443 42.38695 47.50318 52.33409 56.50982 60.22372
##  [8] 63.39049 66.12209 68.64220 70.90317 72.99789 74.87628 76.56908
## [15] 78.16068 79.66647 81.06307 82.33578 83.49726 84.55430 85.57742
## [22] 86.52607 87.42483 88.27709 89.00037 89.71120 90.36629 90.98275
## [29] 91.53483 92.06884 92.54679 93.01149 93.42745 93.79282 94.12631
## [36] 94.44965 94.75679 95.04099 95.31609 95.56898 95.80847
```

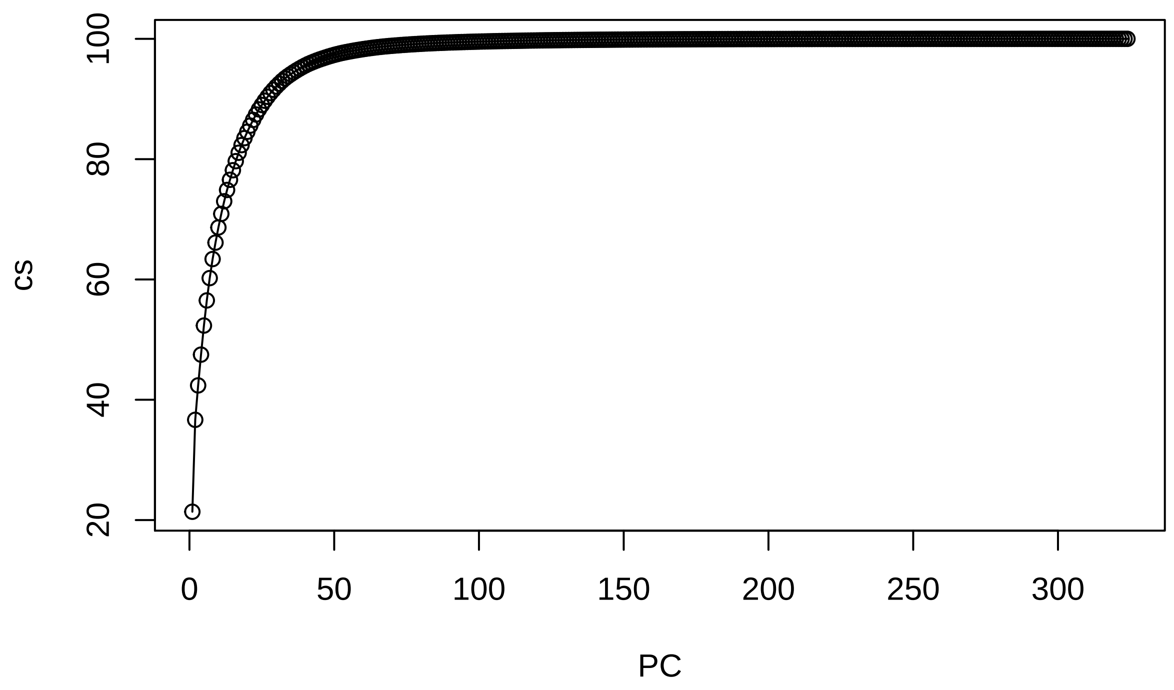
Some quick plots of variance and cumsum

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



The Cummulative variance

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

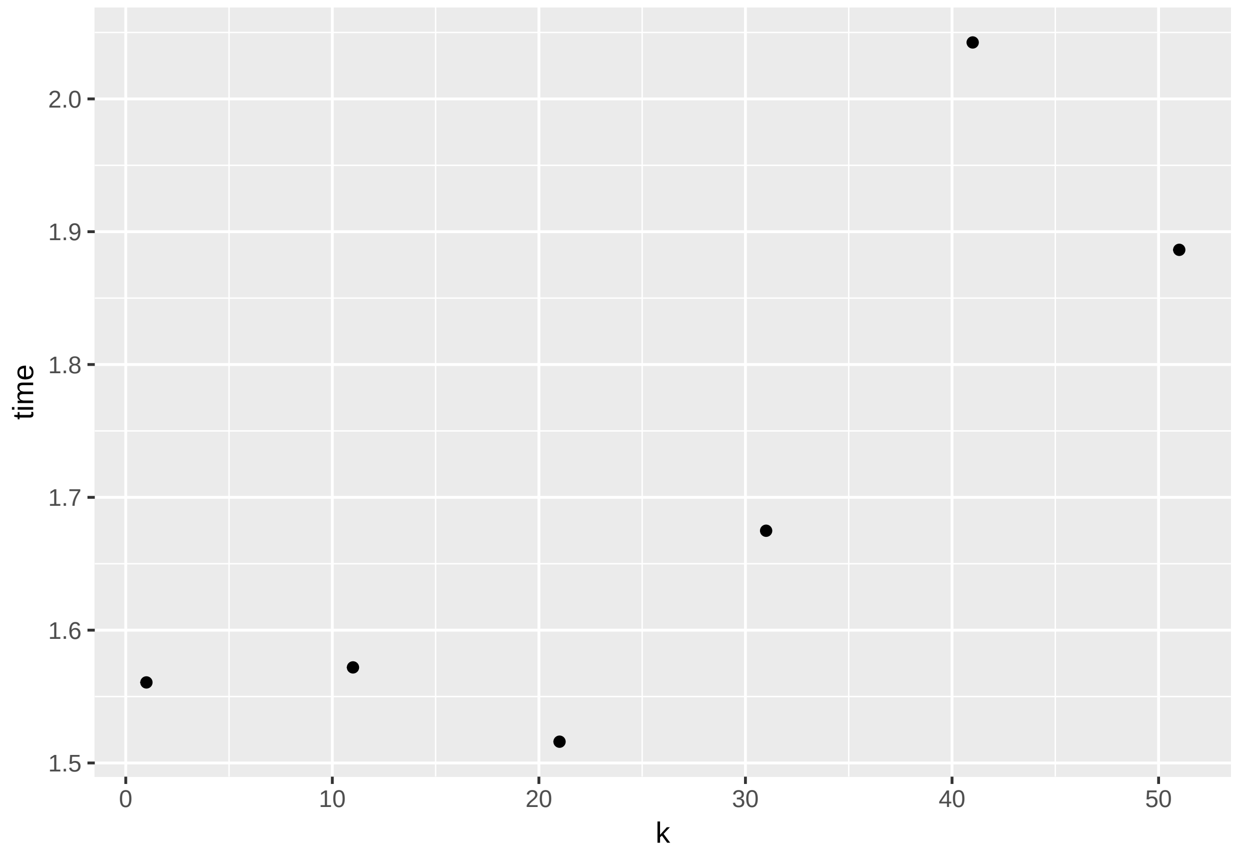


```r
train = inde_pca$x
trainlabels = independent_labels[1:4000]
test = predict(inde_pca,shuffled_inde[4001:8000,])
indeAcc = c()
indeTime = c()

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

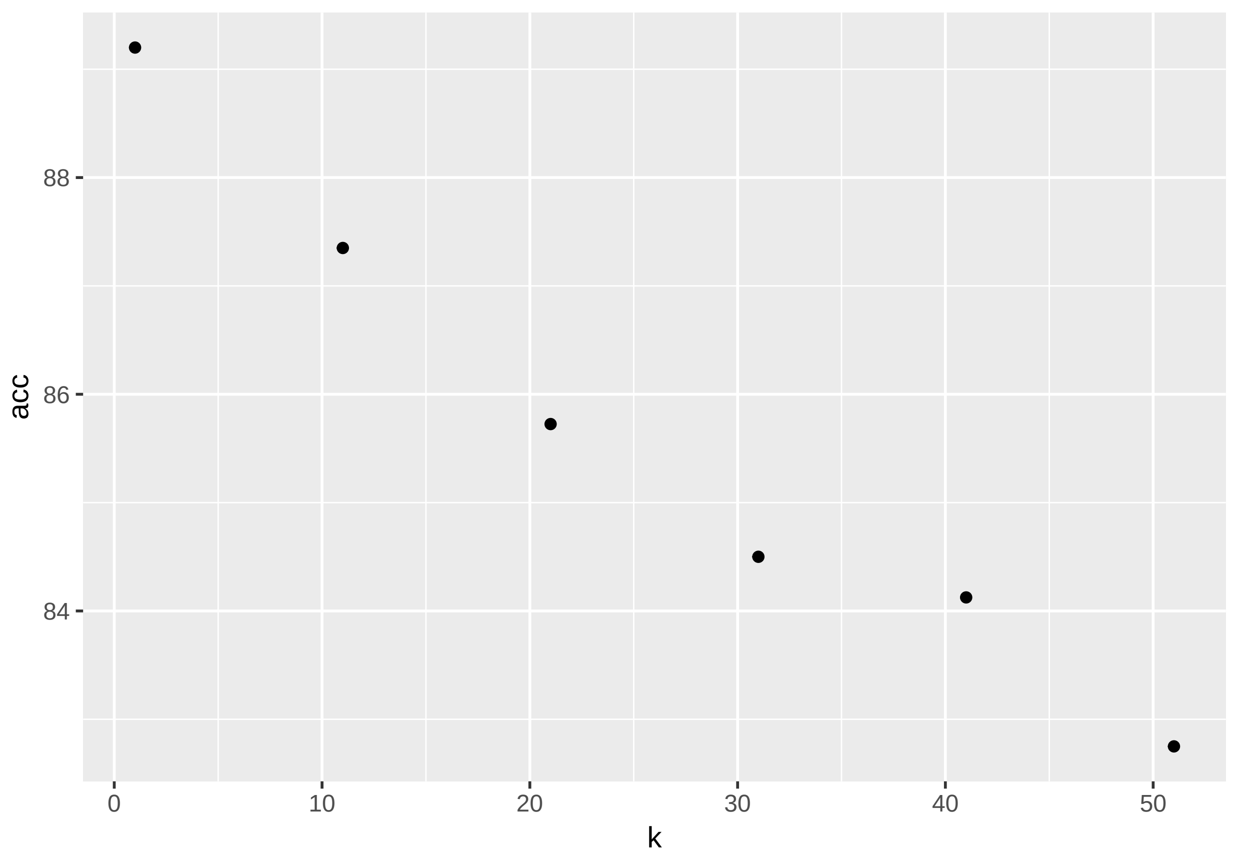
```
## [1] 16 27 38 74
```

```r
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])
}
```

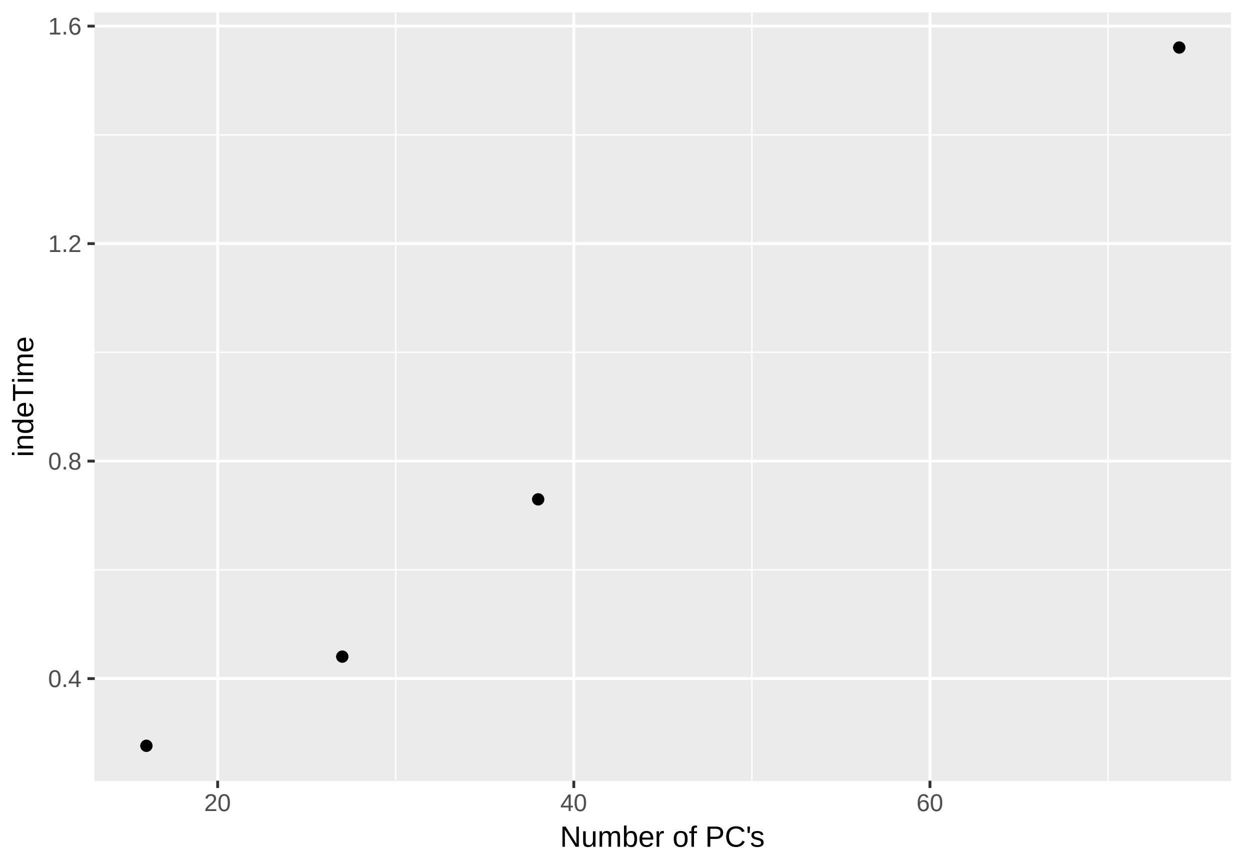
```r
qplot(k,time)
```



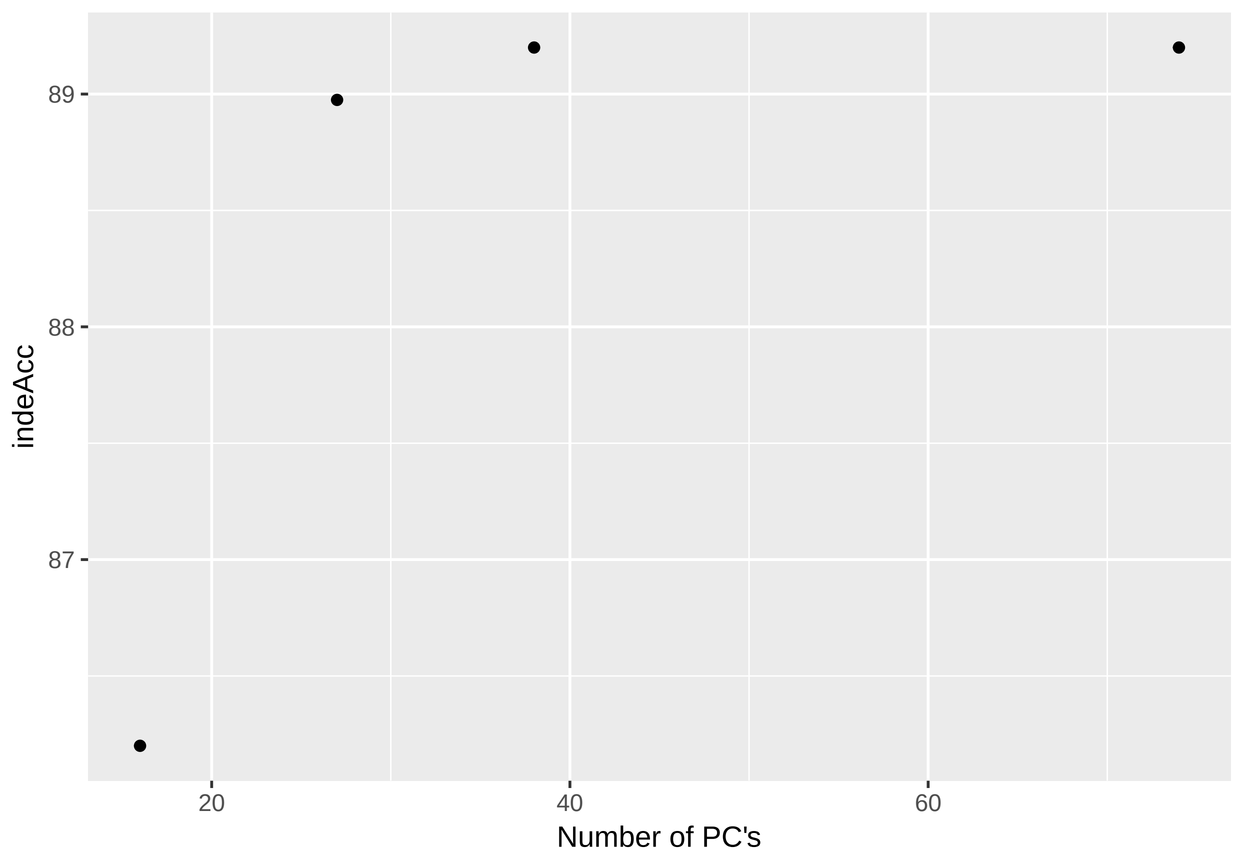
```r
qplot(k,acc)
```



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



```r
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

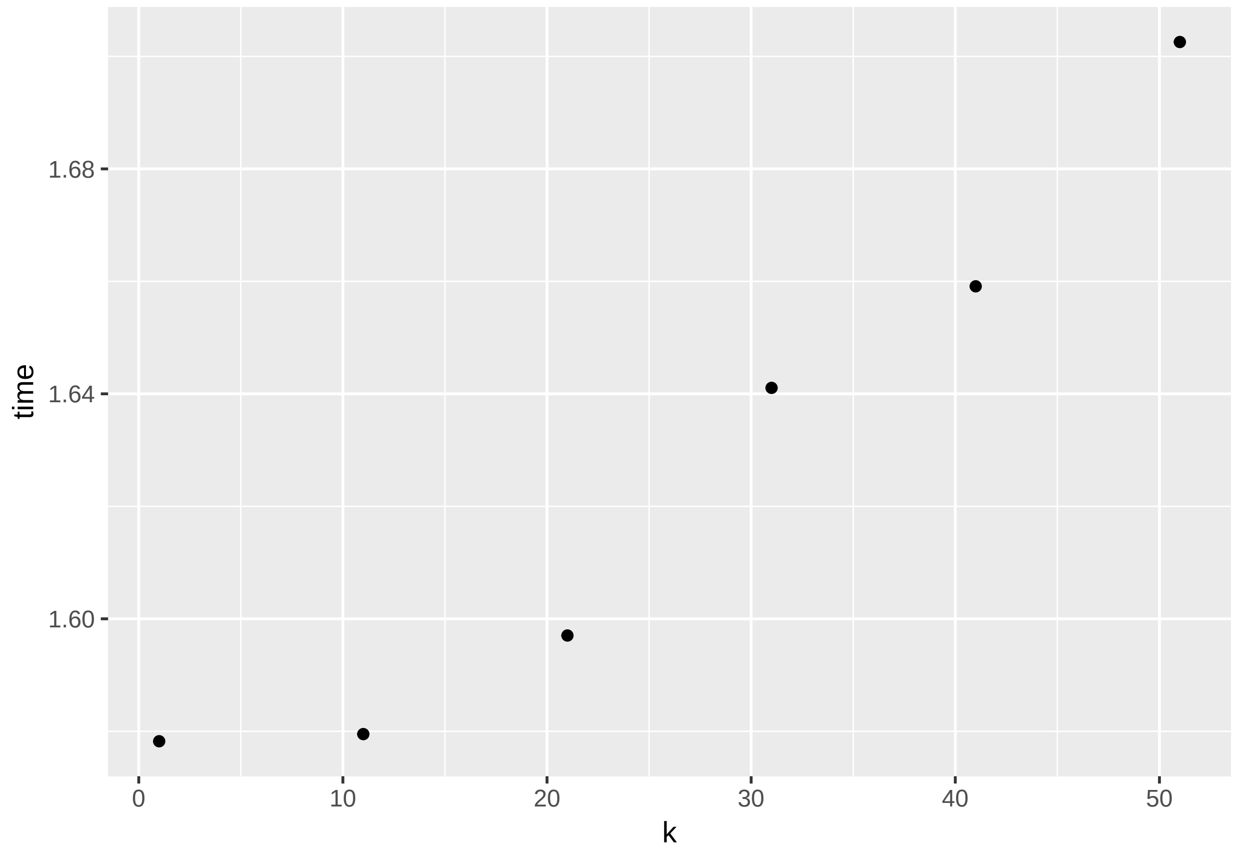
no to do it with person dependent data. This section hasn't provided great answers as i chose to only use 2 persons data to save computational time. this could explain the low percentage of correct predictions due to us writing digits fairly different

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

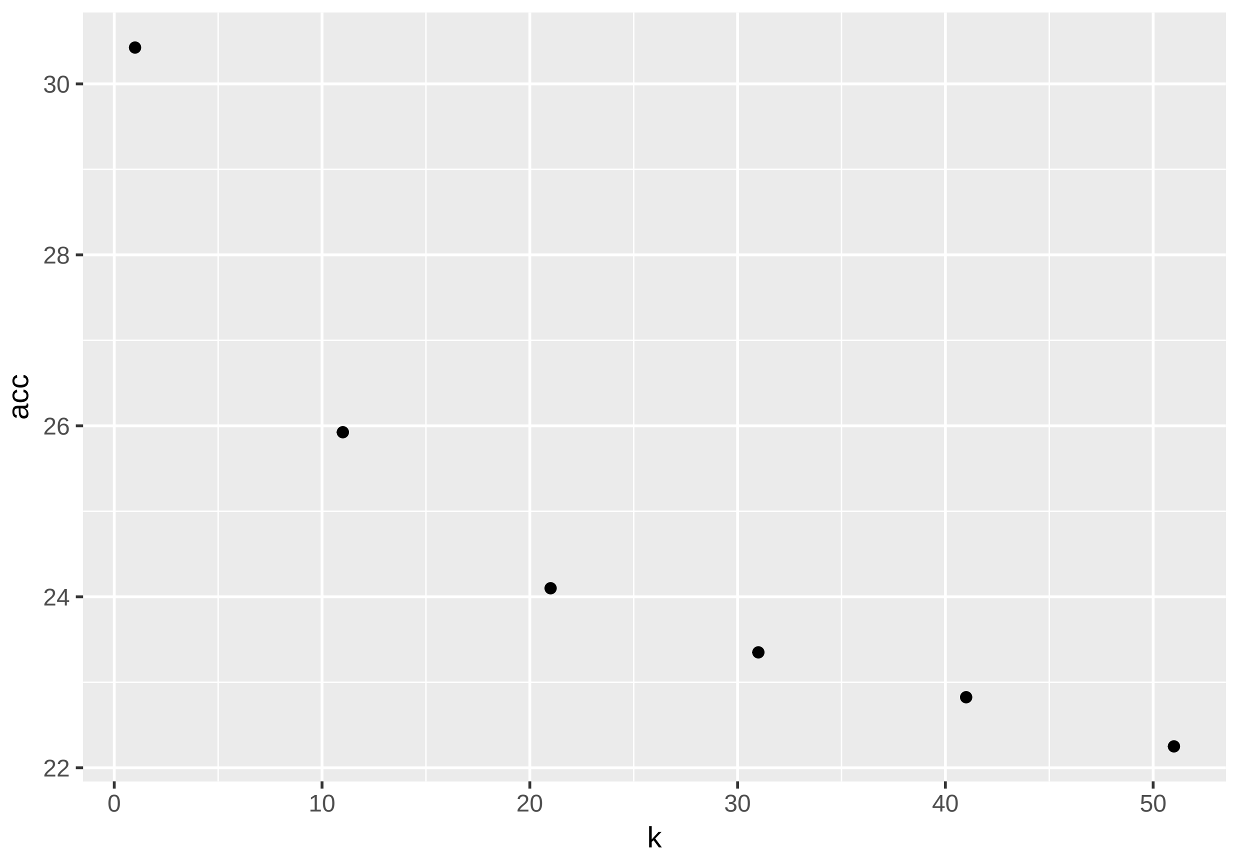
```
train = p1_pca$x
trainlabels = p1_labels
test = predict(p1_pca,p2_shuff)
pacc = c()
ptime = c()

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

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

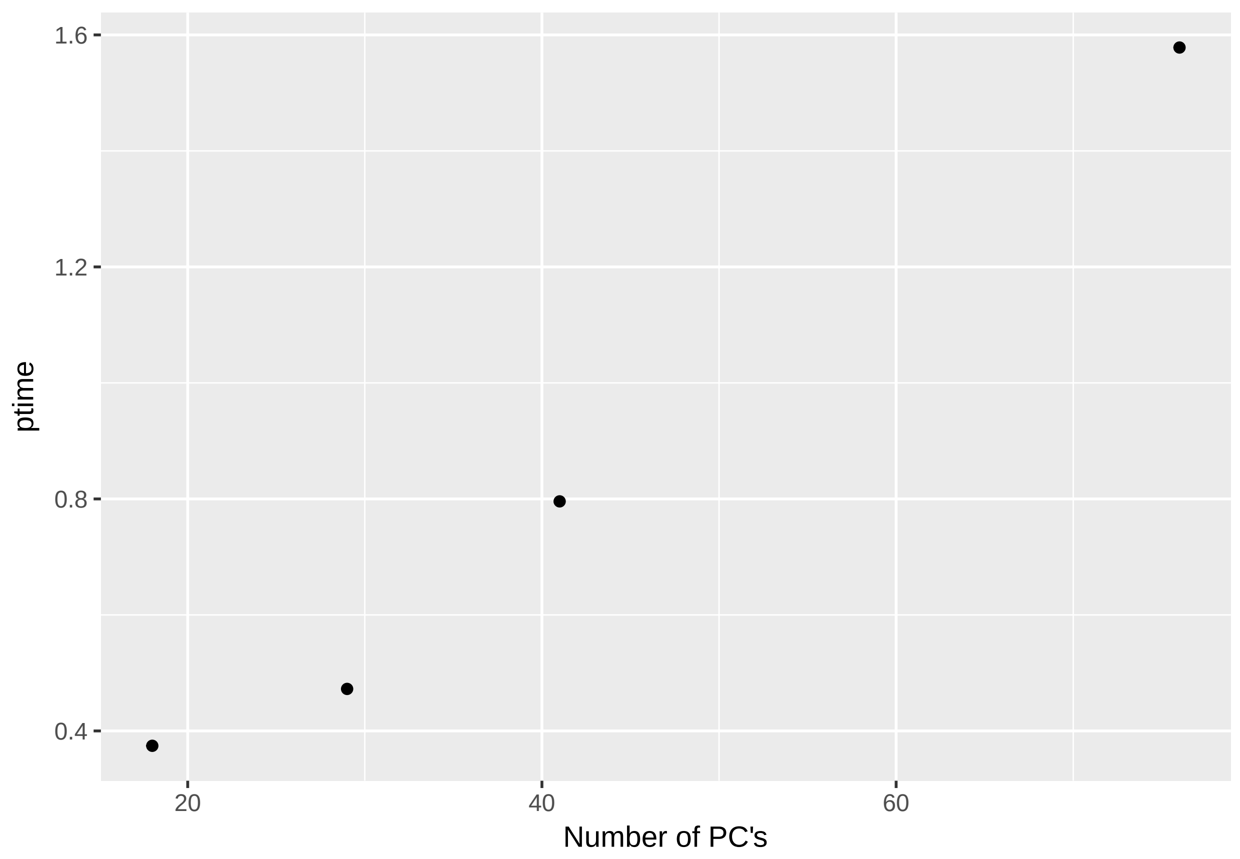
```
k = seq(from=1,to=51,by=10)
for (j in variances){
  time = c()
  acc = c()
  for (i in k){
    startTime = Sys.time()
    results = knn(train[,1:j],test[,1:j],k=i,cl = trainlabels)
    endtime = Sys.time()
    time=c(time,(endtime-startTime))
    acc = c(acc,accuracy(results,testlabels))
  }

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



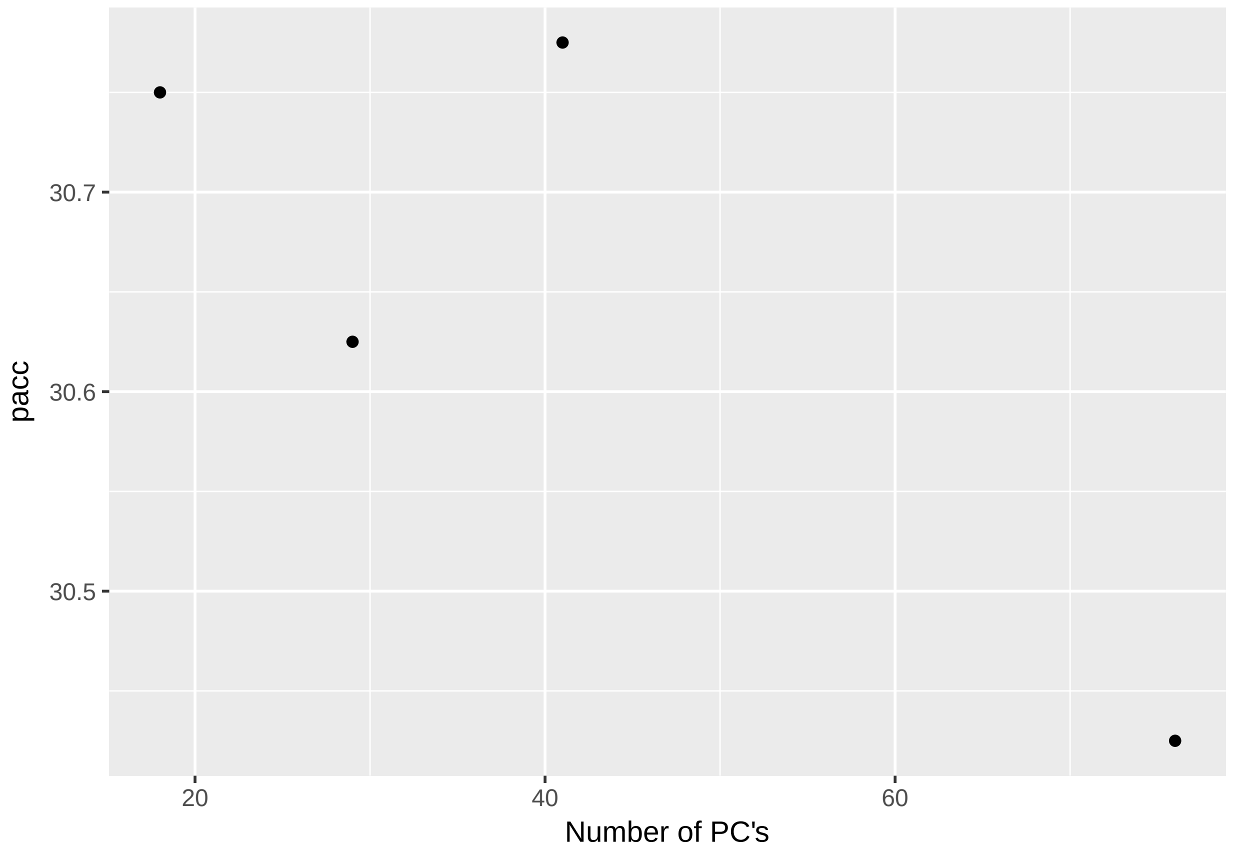
```
qplot(k,acc)
```



```r
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] 0.3281848
```

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

```
## [1] 89.35657
```

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] 0.4019139
```

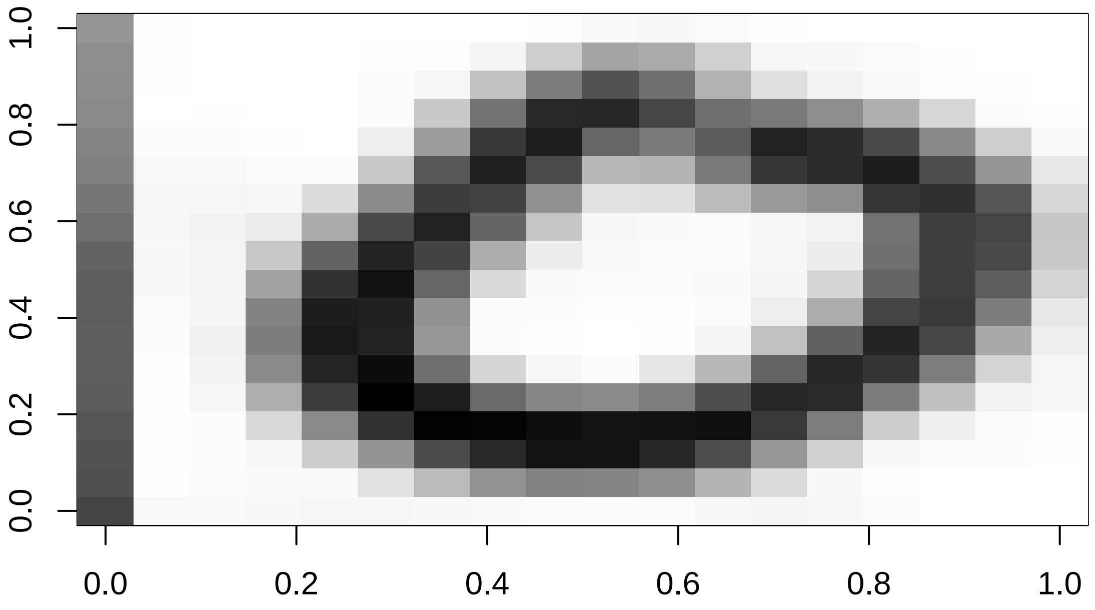
```
print(mean(indeAcc))
```

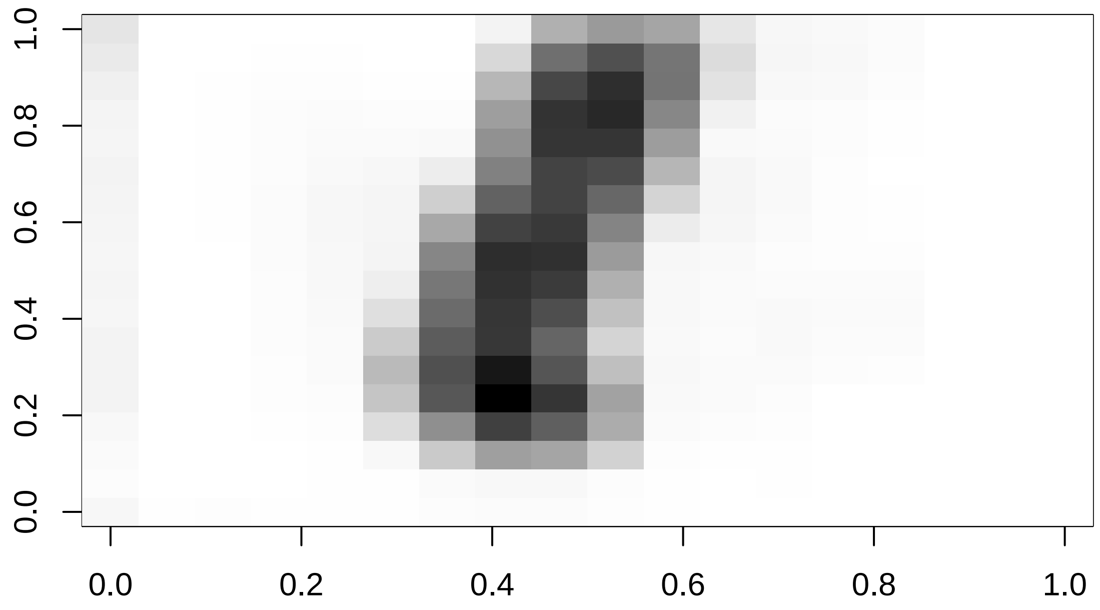
```
## [1] 87.93364
```

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

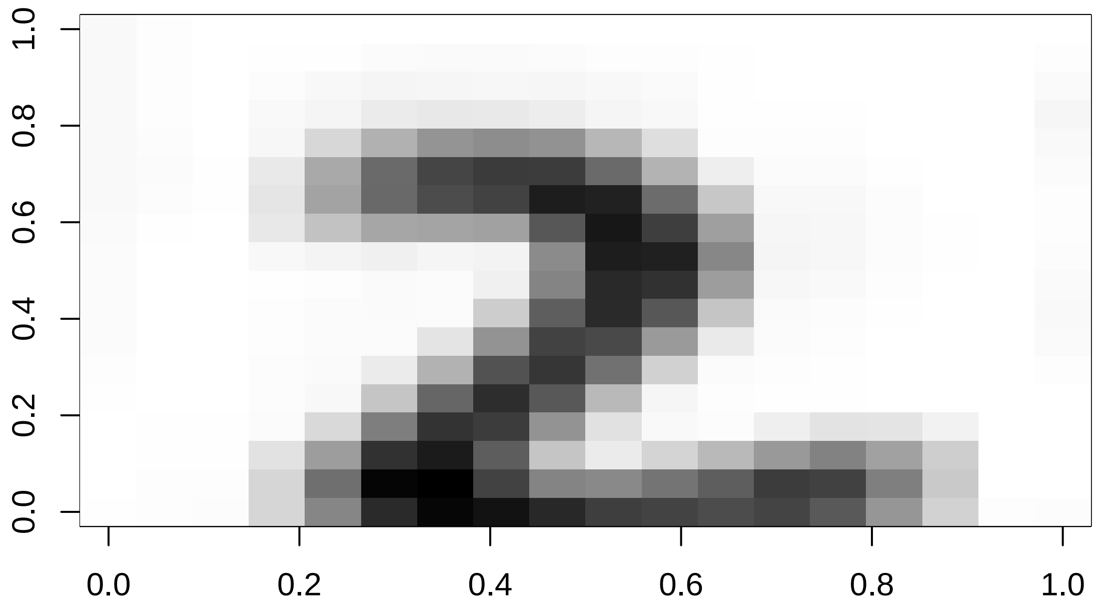
Reconstruction

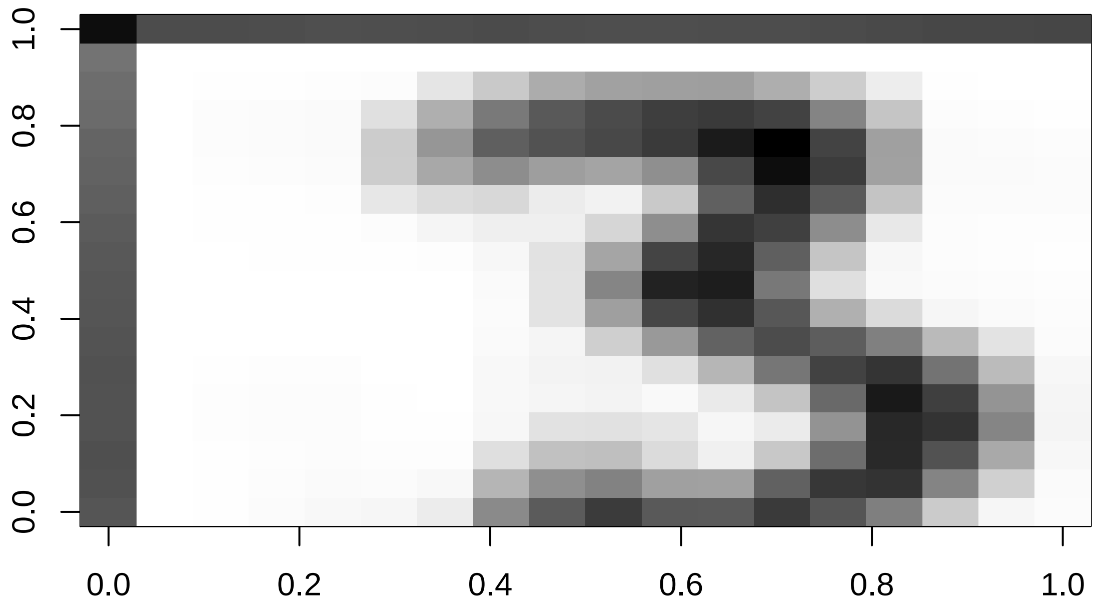
```
for (i in 0:9){
  imageSize = sqrt(ncol(p1)-1)
  imageMatrix <- matrix( p1[1+i*400,2:ncol(p1)], nrow = imageSize, ncol=imageSize, byrow= FALSE)
  imageMatrix <- rotate(imageMatrix, 270)
  image(imageMatrix, col=gray((0:255)/255))

}
```



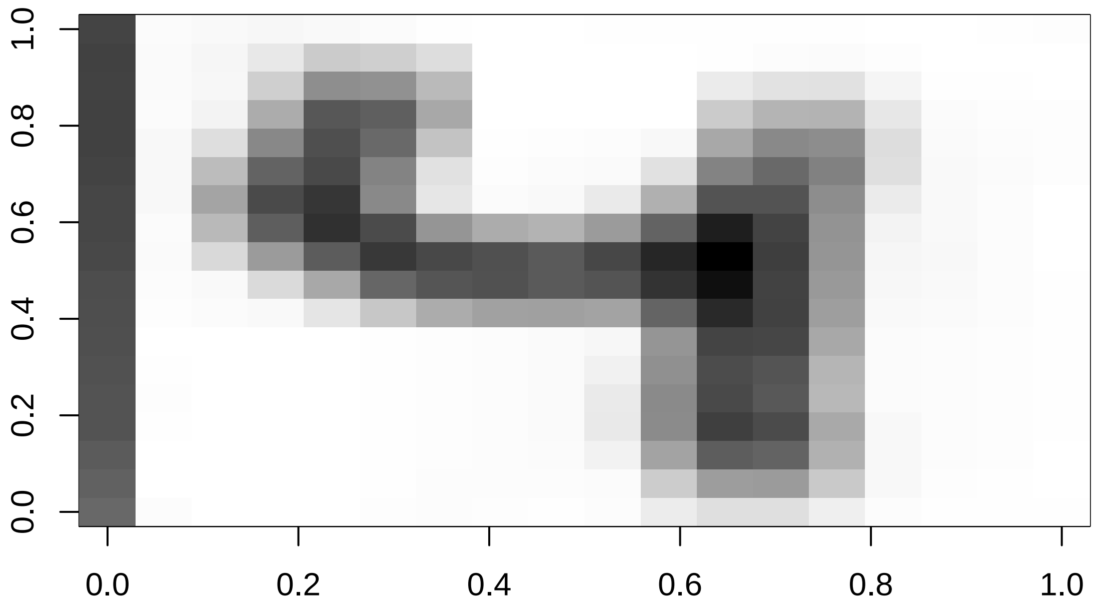


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

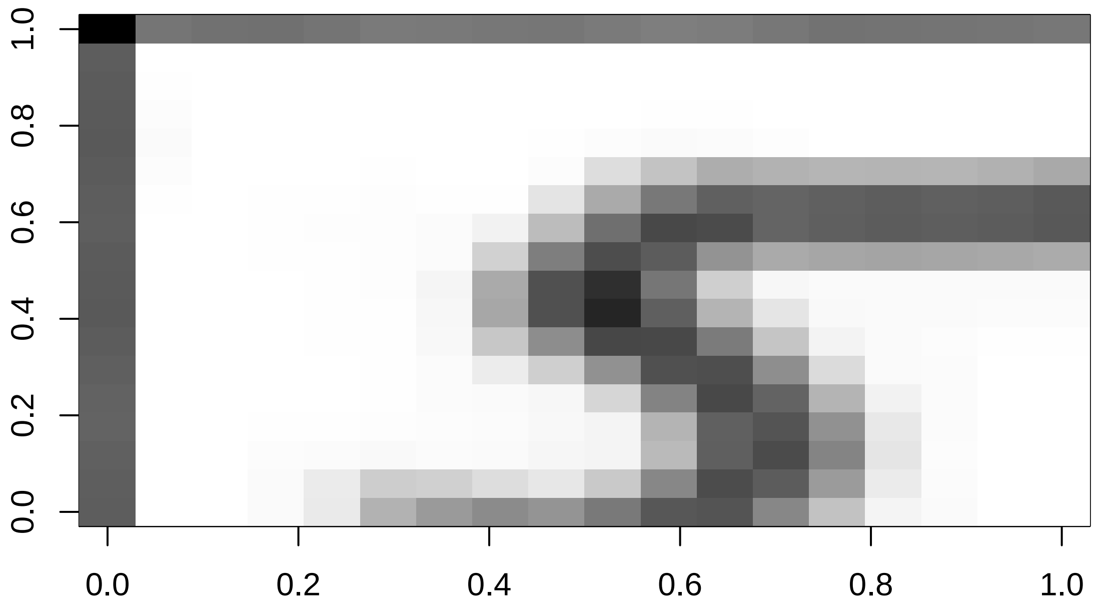

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0



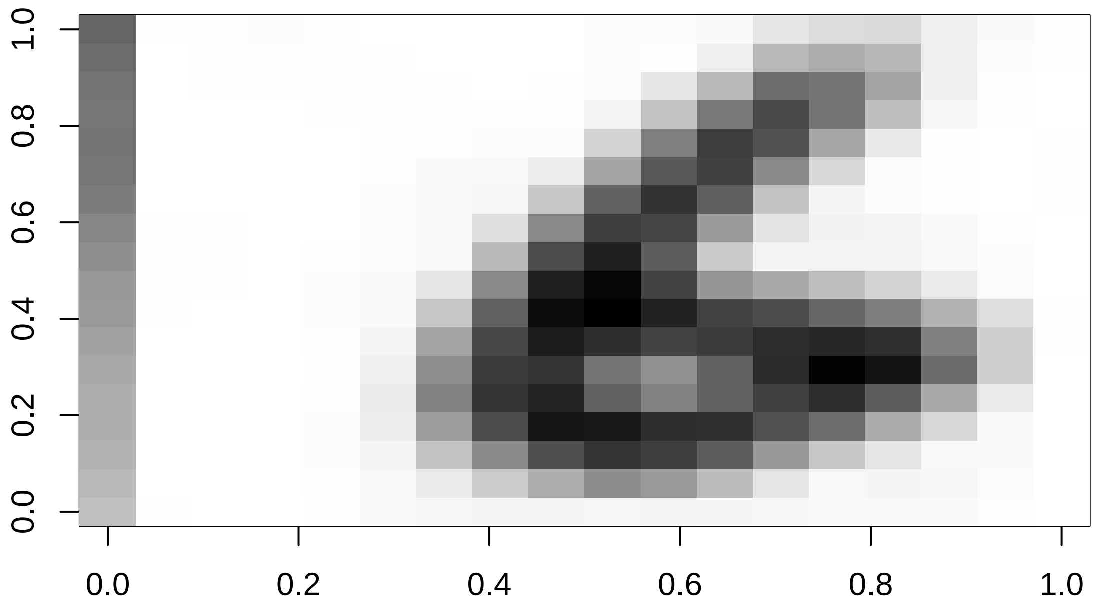
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0



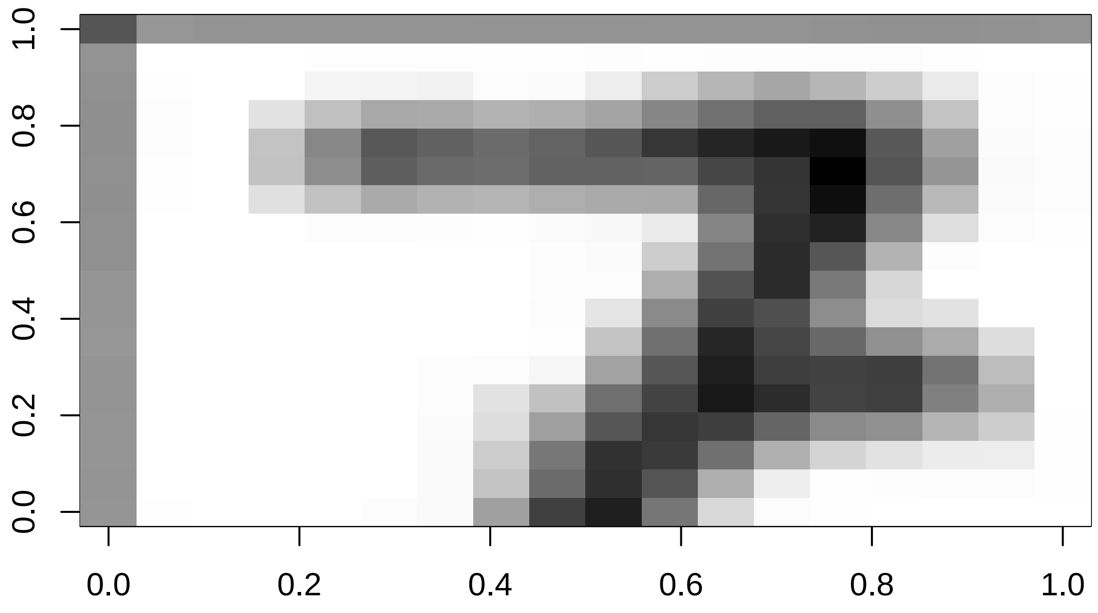
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

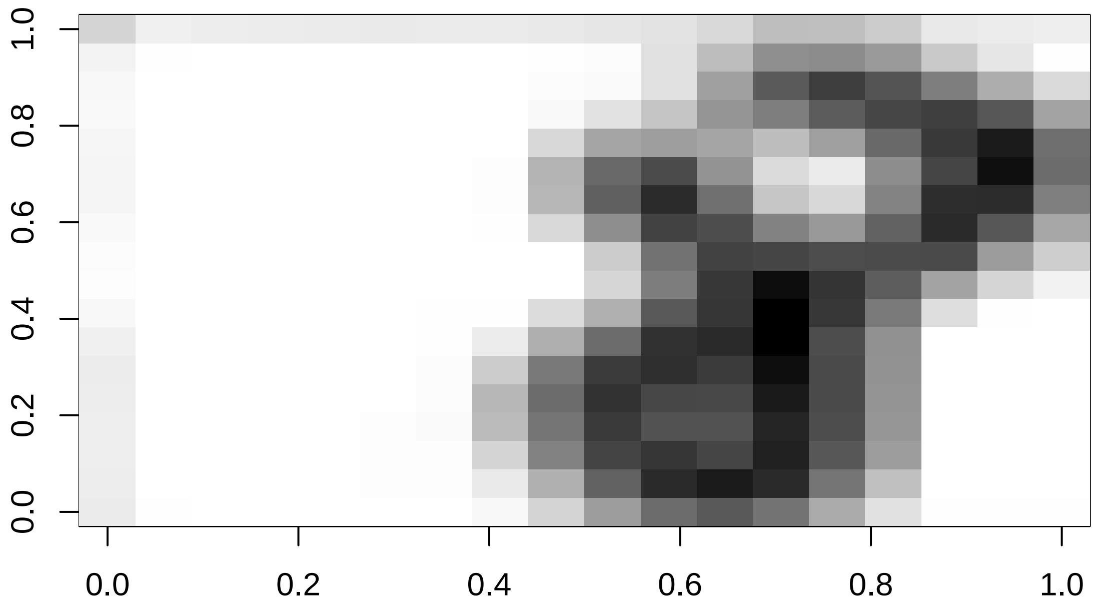


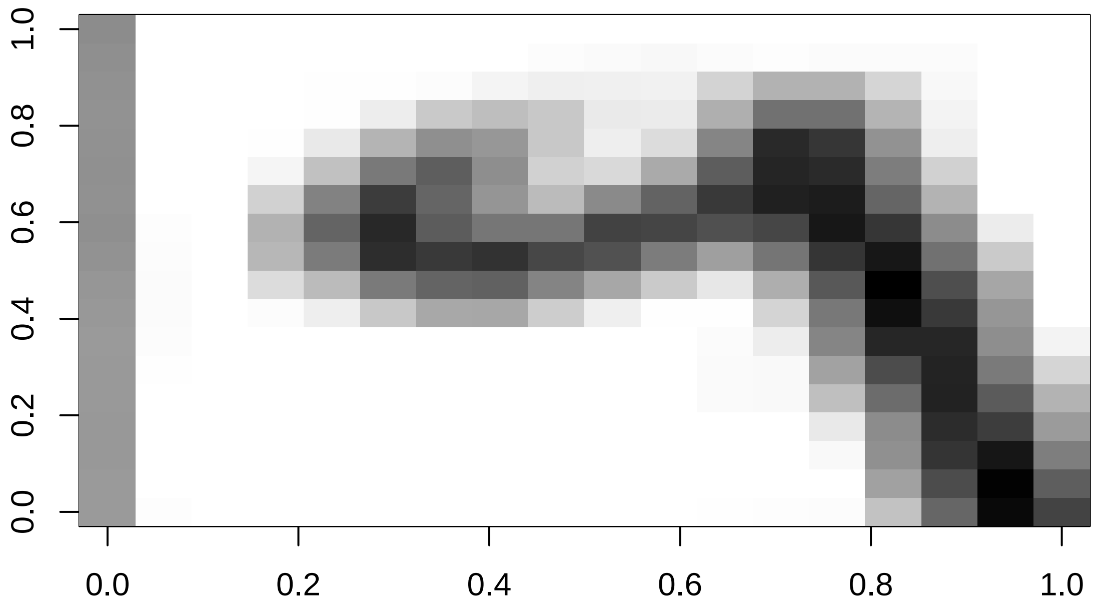
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0



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



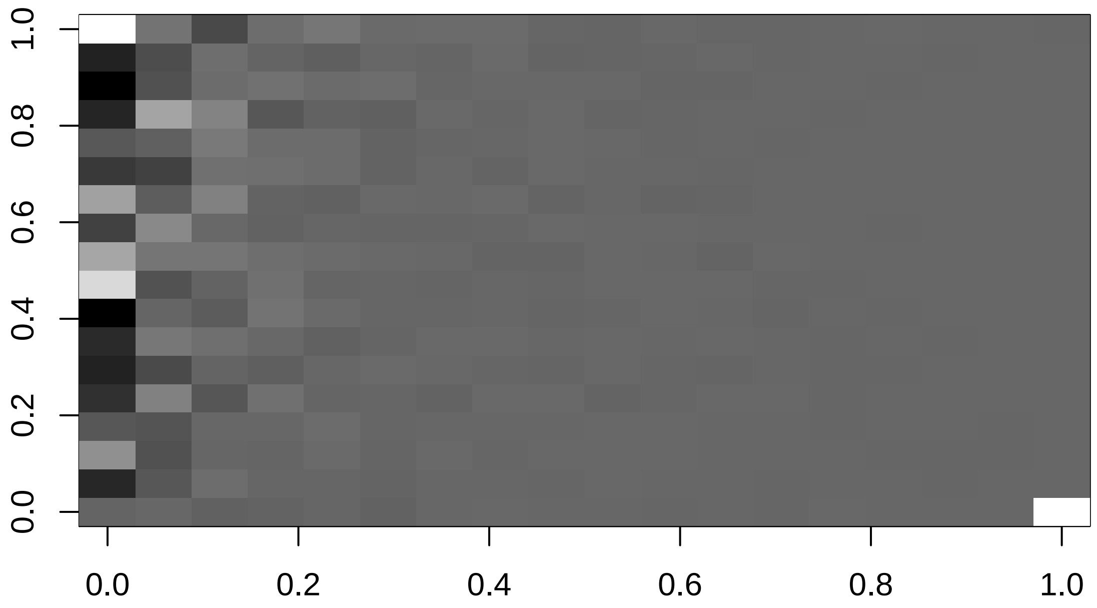
When printing out the images, it can be seen that the corners might not have been set properly, which could provide more details at to why some accuracy percentages have been low.

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

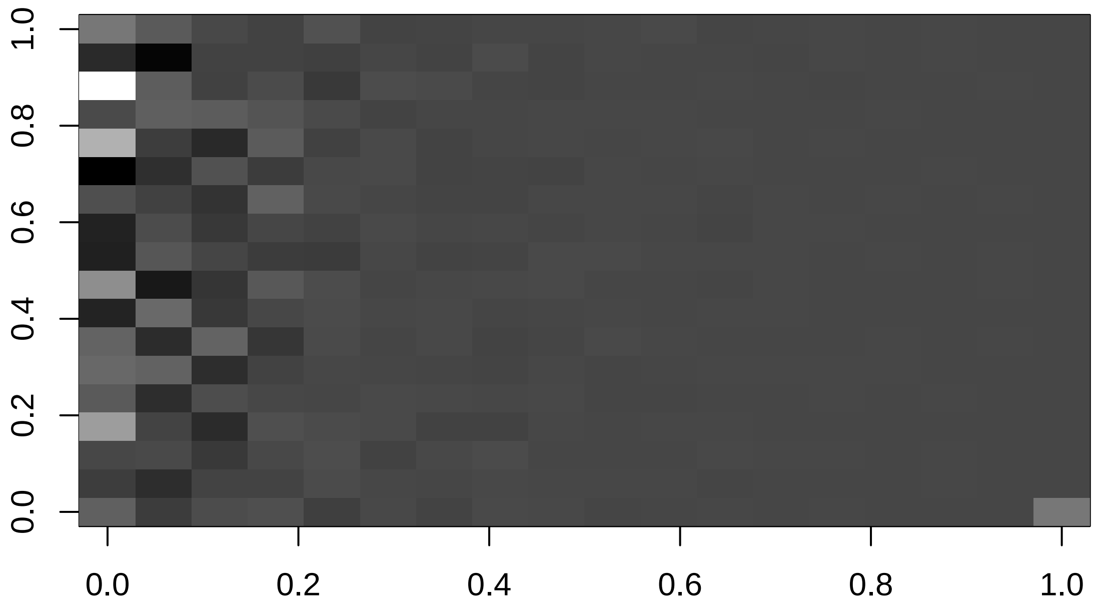
```
## [1] 323
```

```
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))

}
```

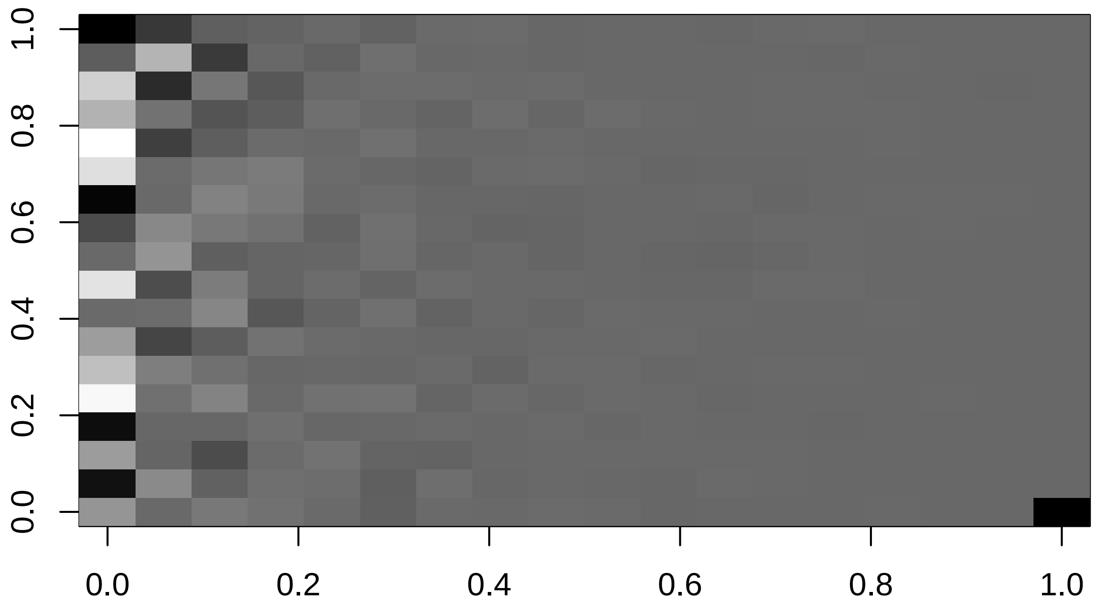
```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]

## 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]
```



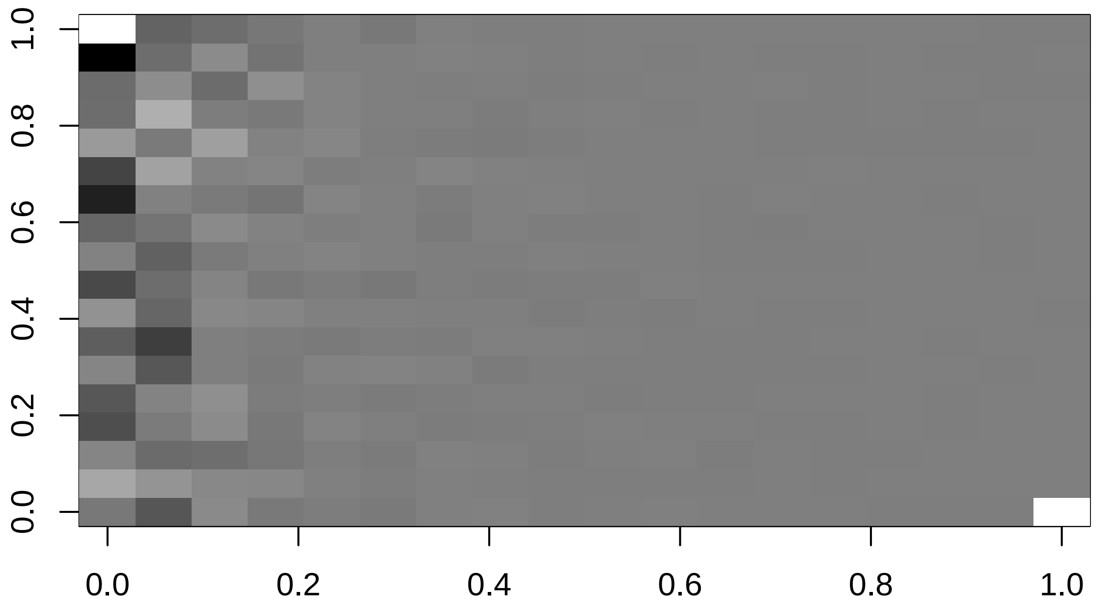
```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]
```



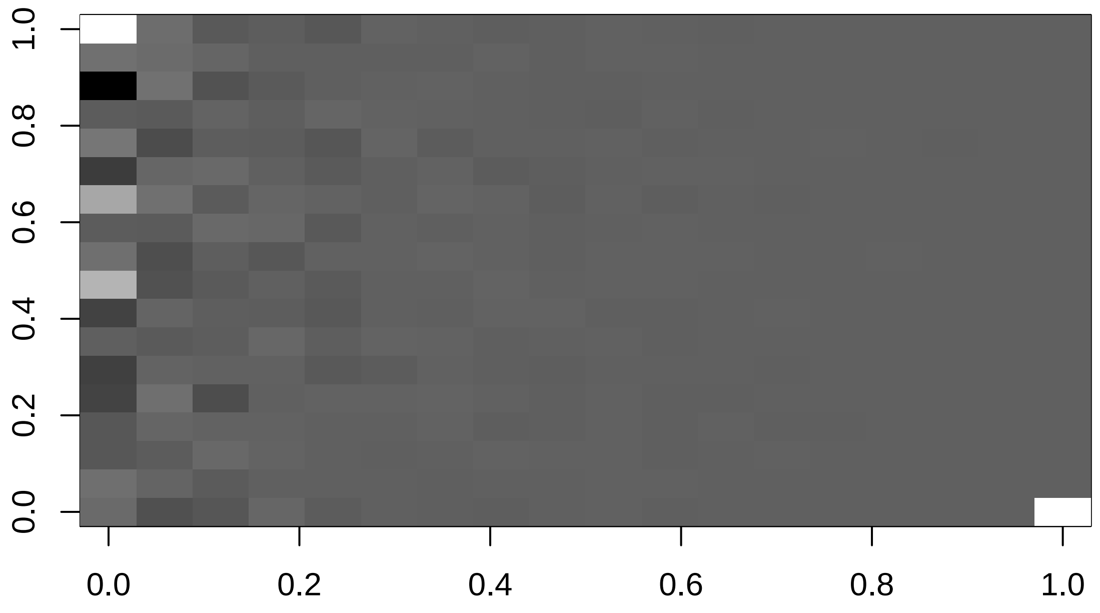
```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]
```



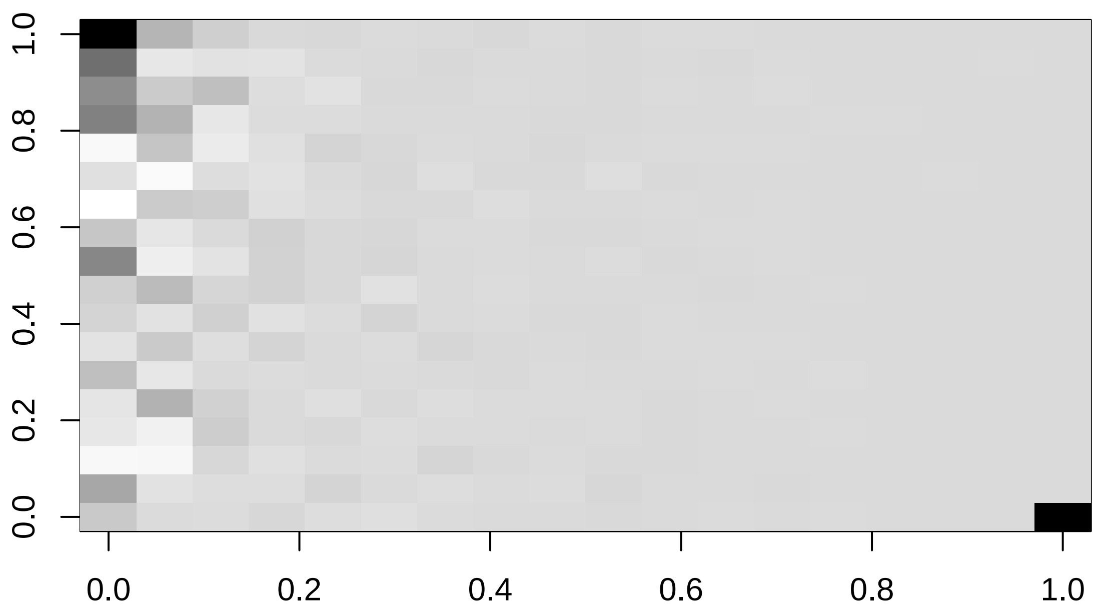
```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]
```



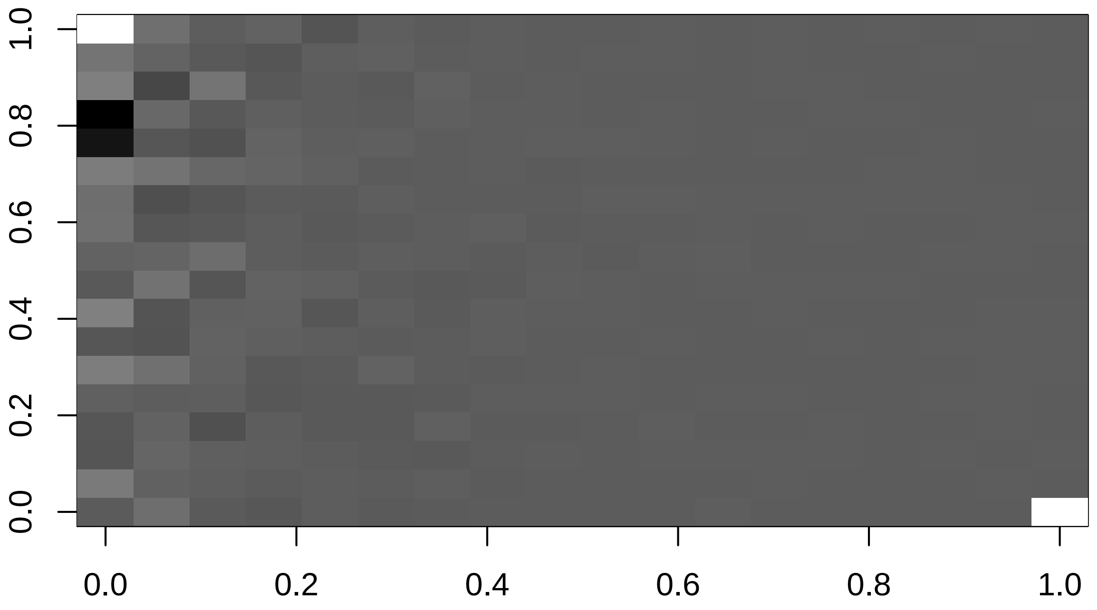
```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]
```



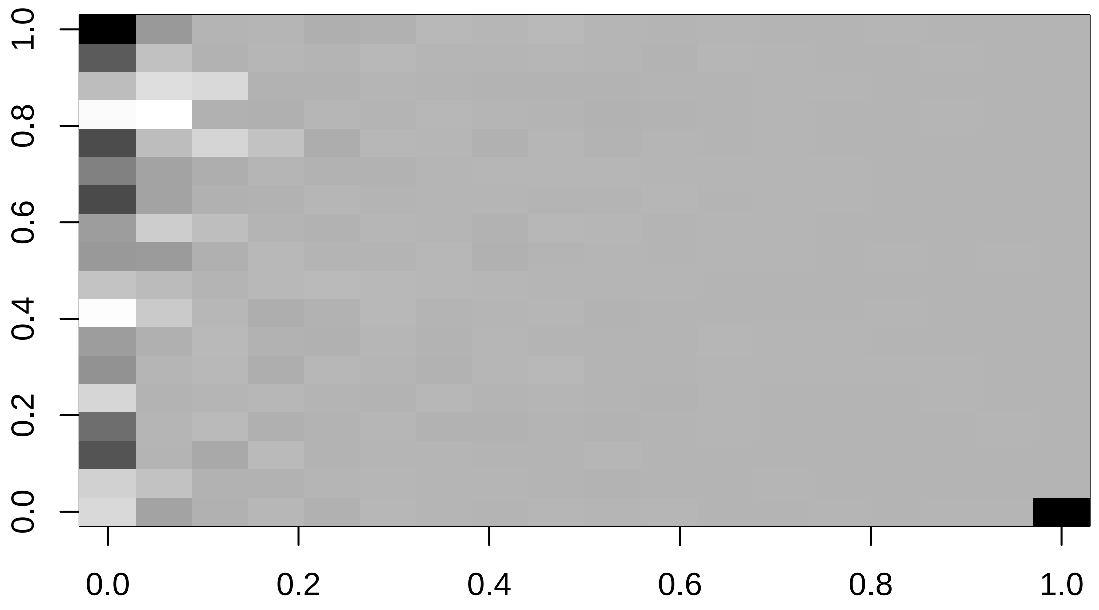
```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]
```



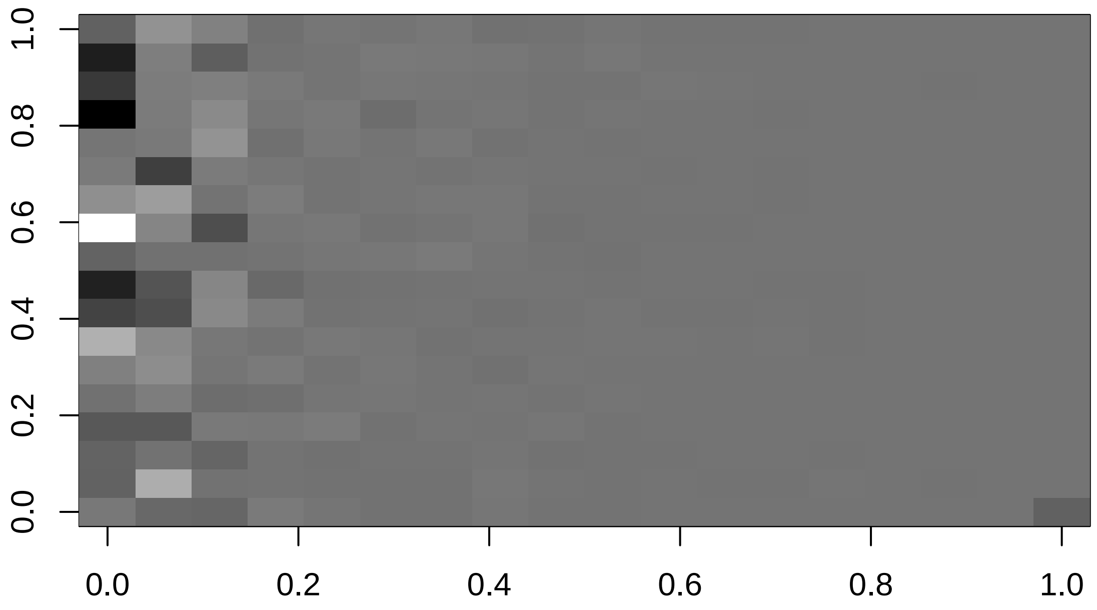
```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]
```

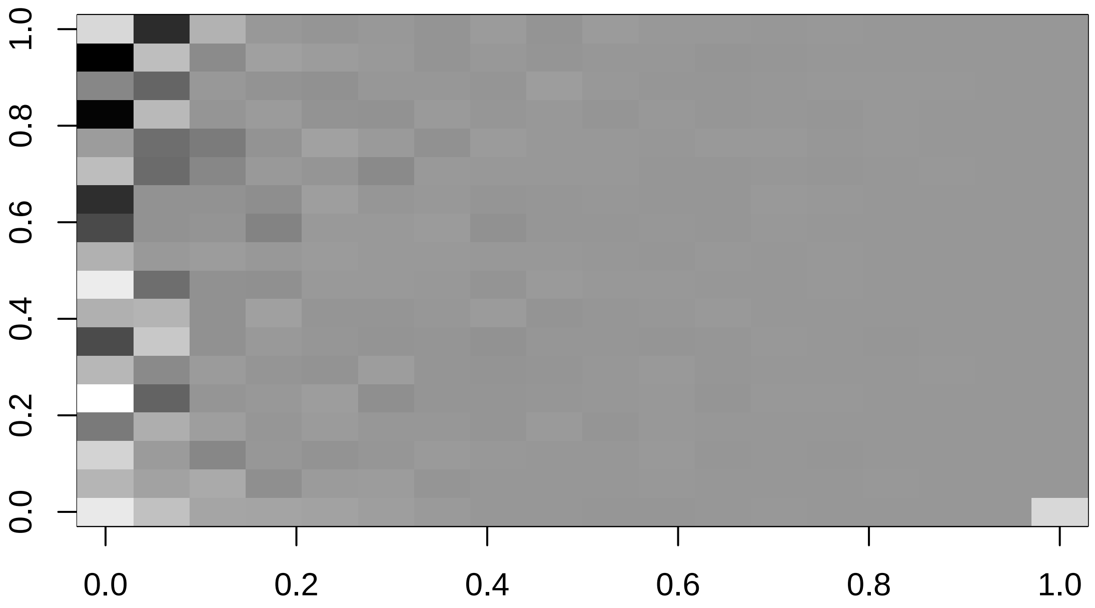


```
## Warning in matrix(p1_pca$x[i * 399 + i, 1:nrow(p1_pca$rotation) - 1], nrow
## = imageSize, : data length [323] is not a sub-multiple or multiple of the
## number of rows [18]
```



```
## 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]
```

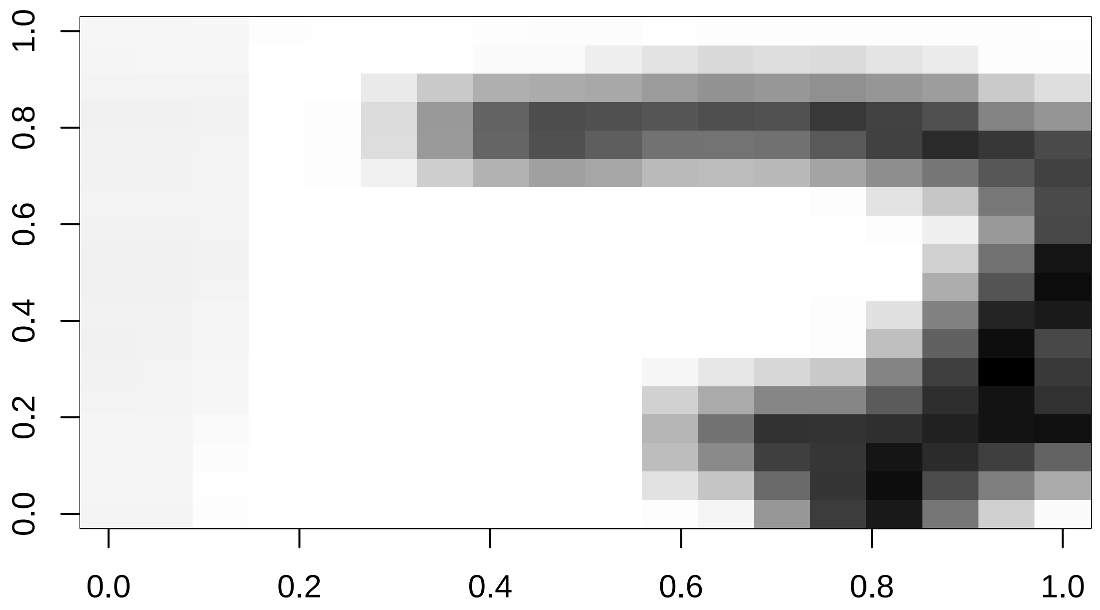

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

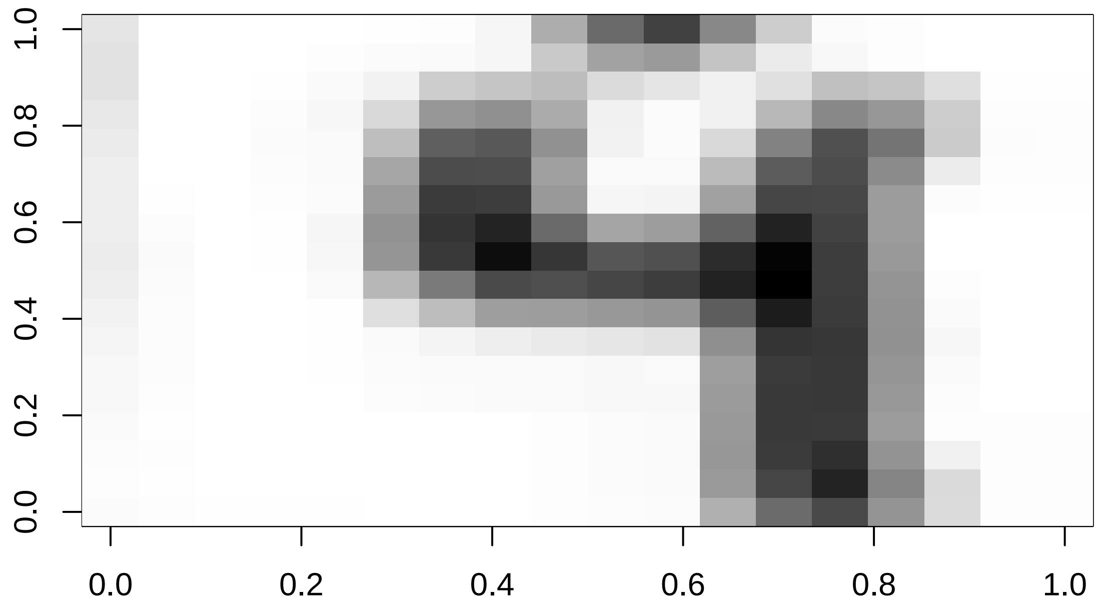


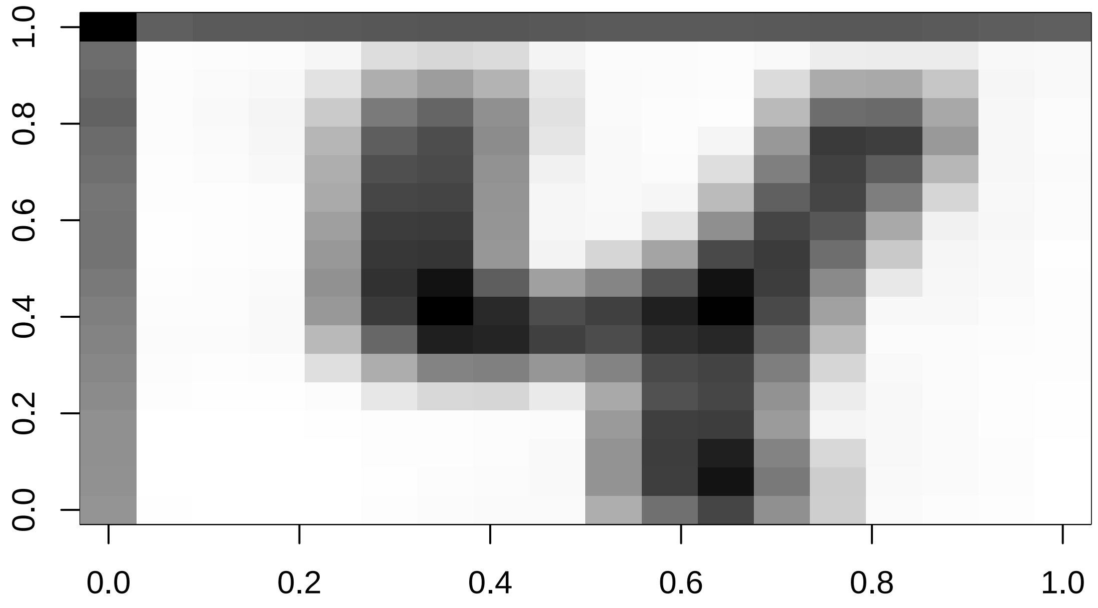
To be honest, i have no idea what's happening here, or if i'm doing it right but i'm guessing it's PCA trying to do it's own thing

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.

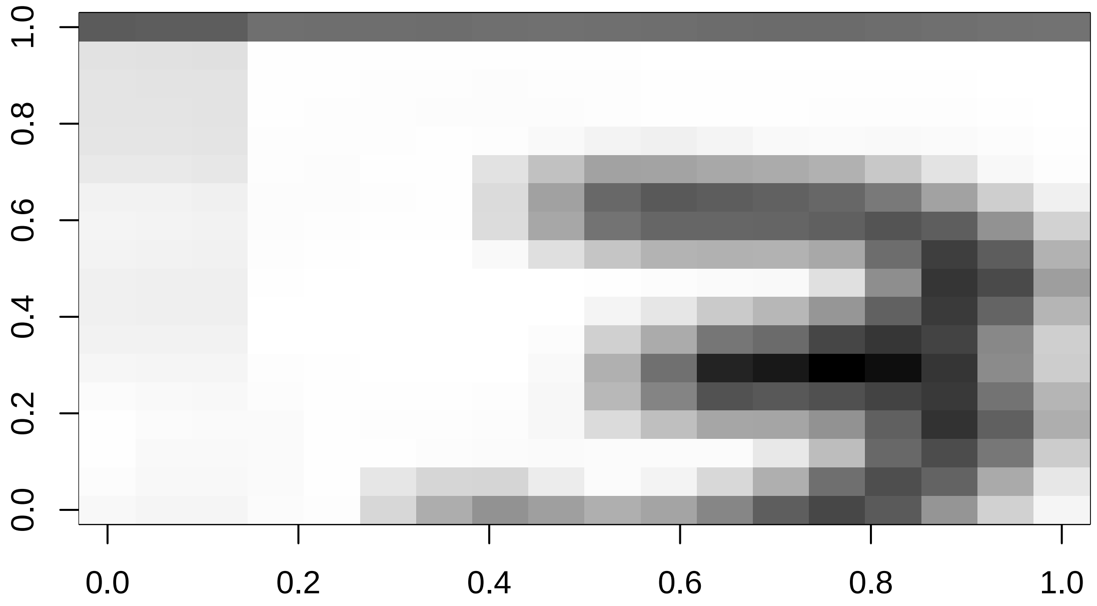
```
for (i in 1:10){

  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))

}
```

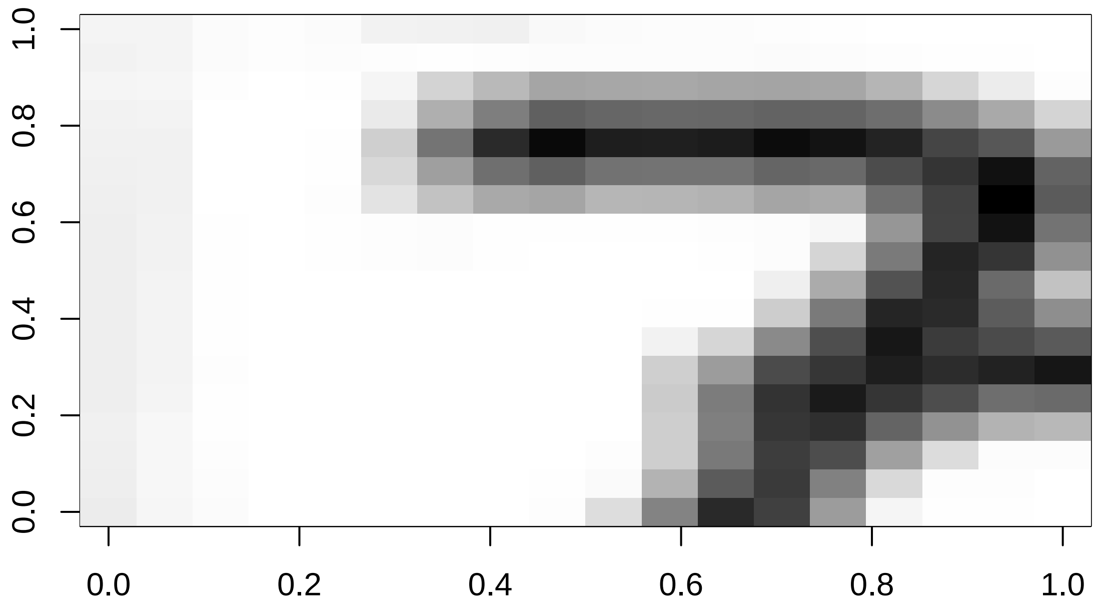

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


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

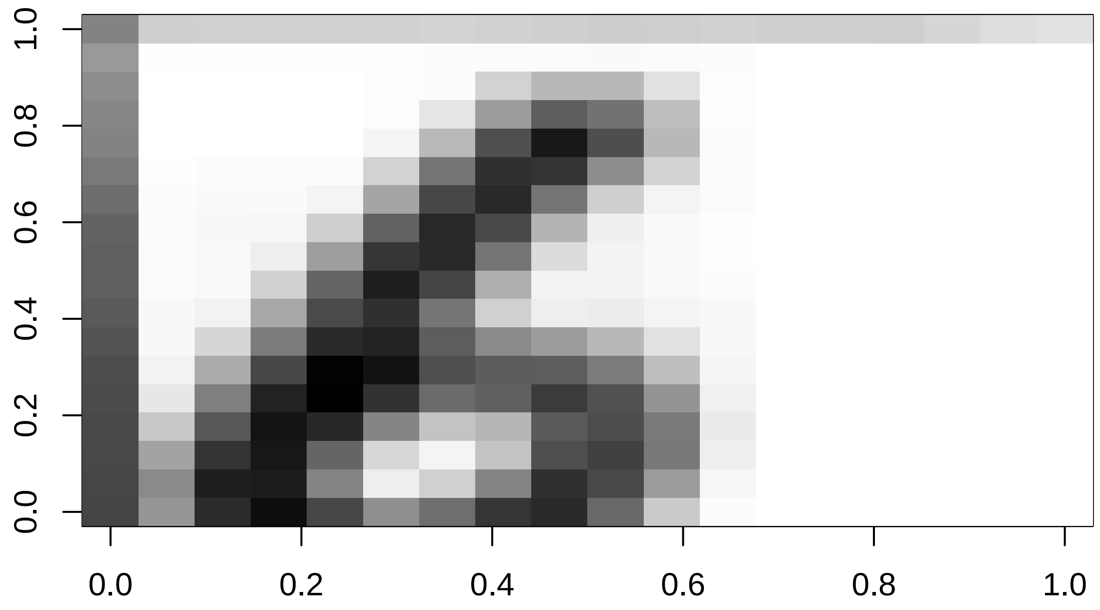


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

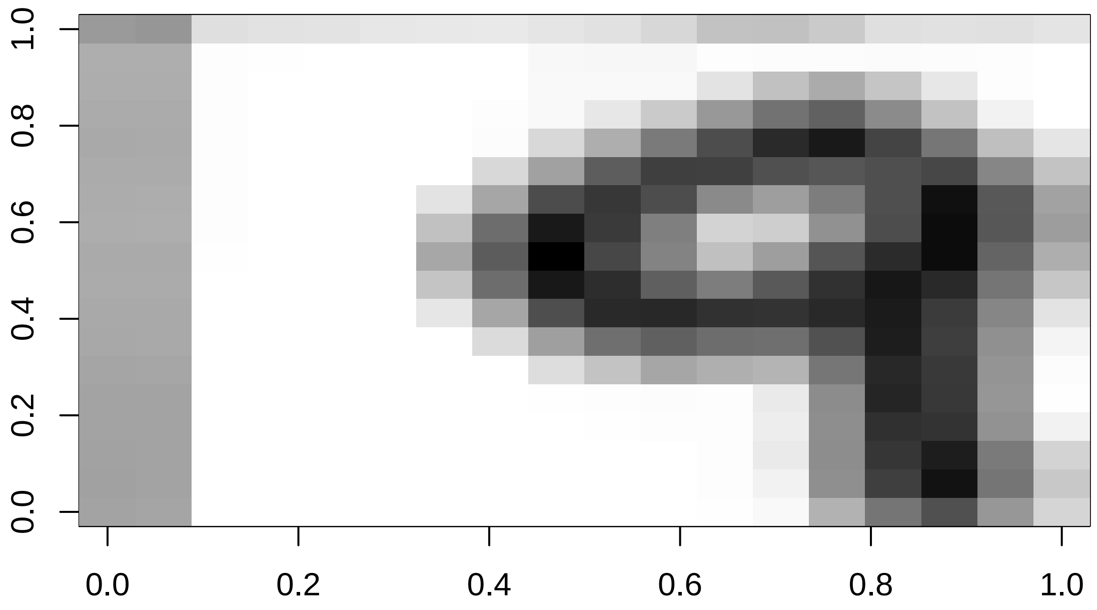

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

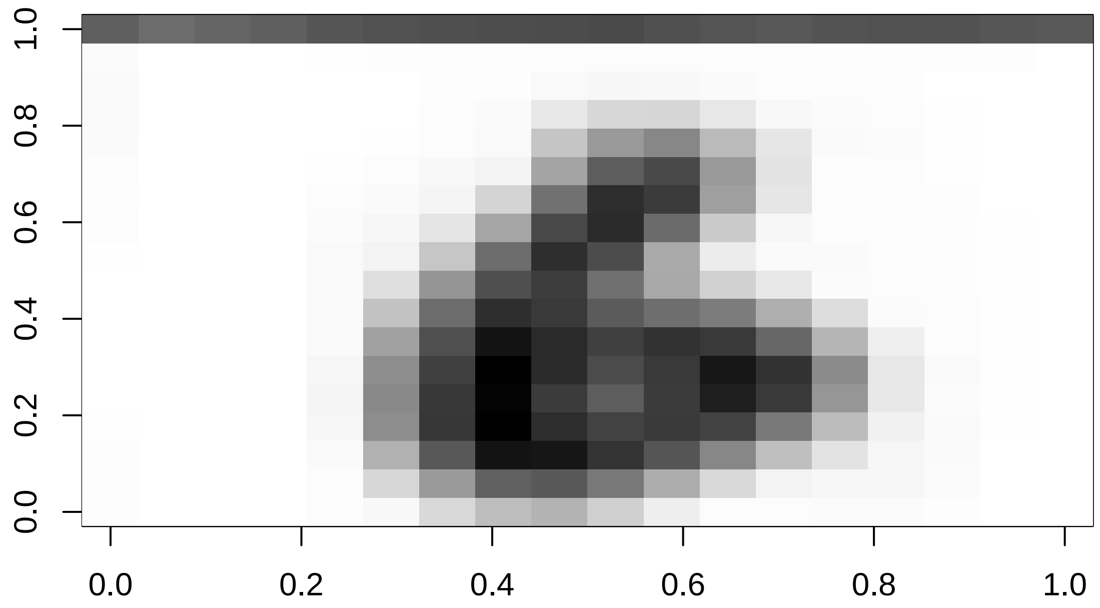


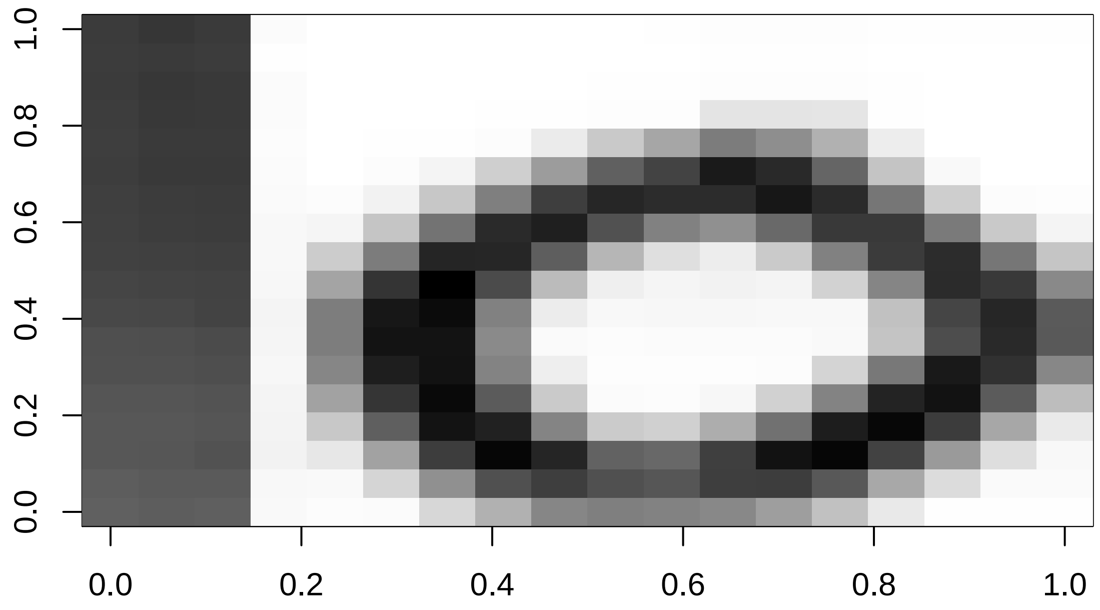
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

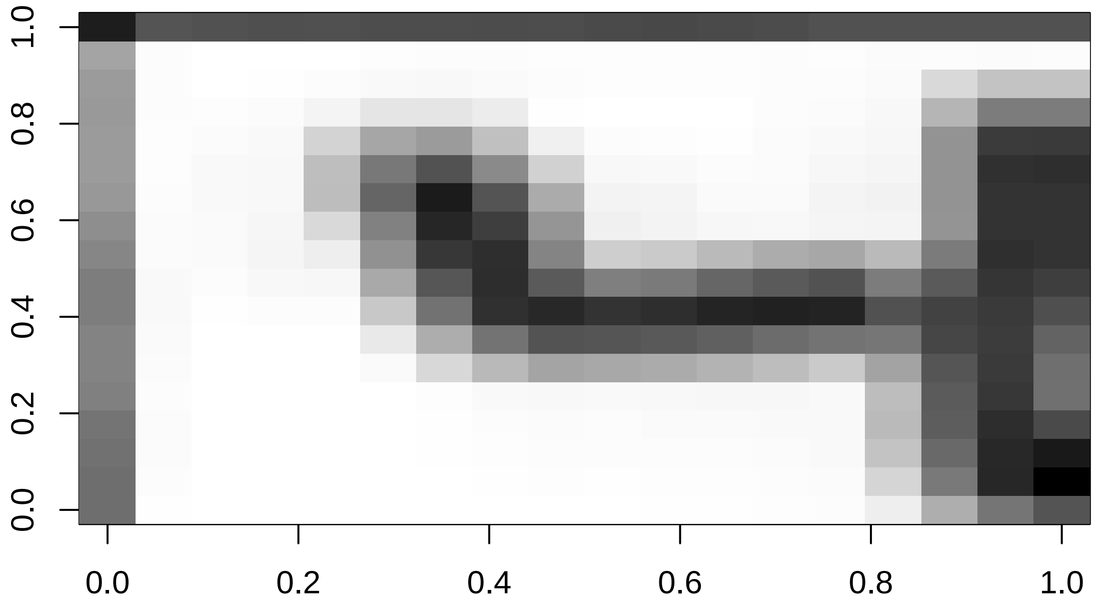


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
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