ADS FINAL PROJECT

TEAM 8

RUI SHEN

HANAN ALSALAMAH

DIVYA MALVIYA

ADBHUT SANGAL

OBJECTIVES

- **PART 1:**

Run neural networks on Prudential Insurance dataset

- **PART 2:**

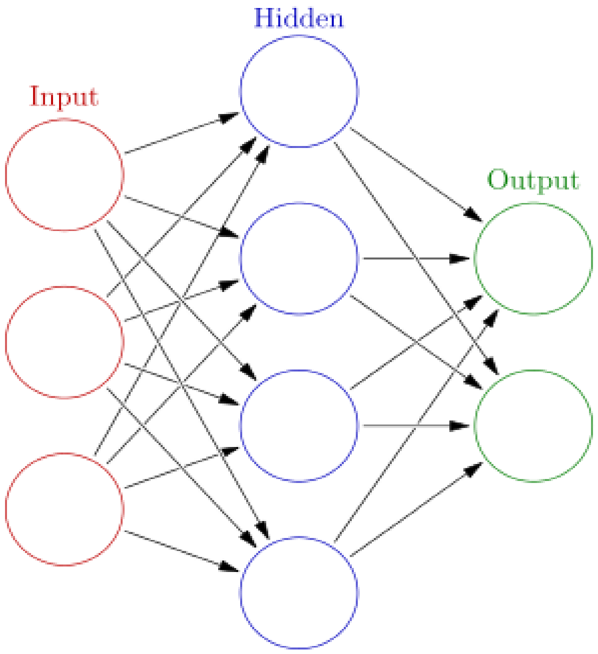
Fit Convolutional Neural Network (CNN) to Semeion Handwritten Digit Data by finding the optimum number of hidden layers and related nodes

PART 1

NEURAL NETWORK
ON
PRUDENTIAL DATASET

NEURAL NETWORKS

Neural Network is a non-linear model characterized by an activation function, which is used by interconnected information processing units to transform input into output.

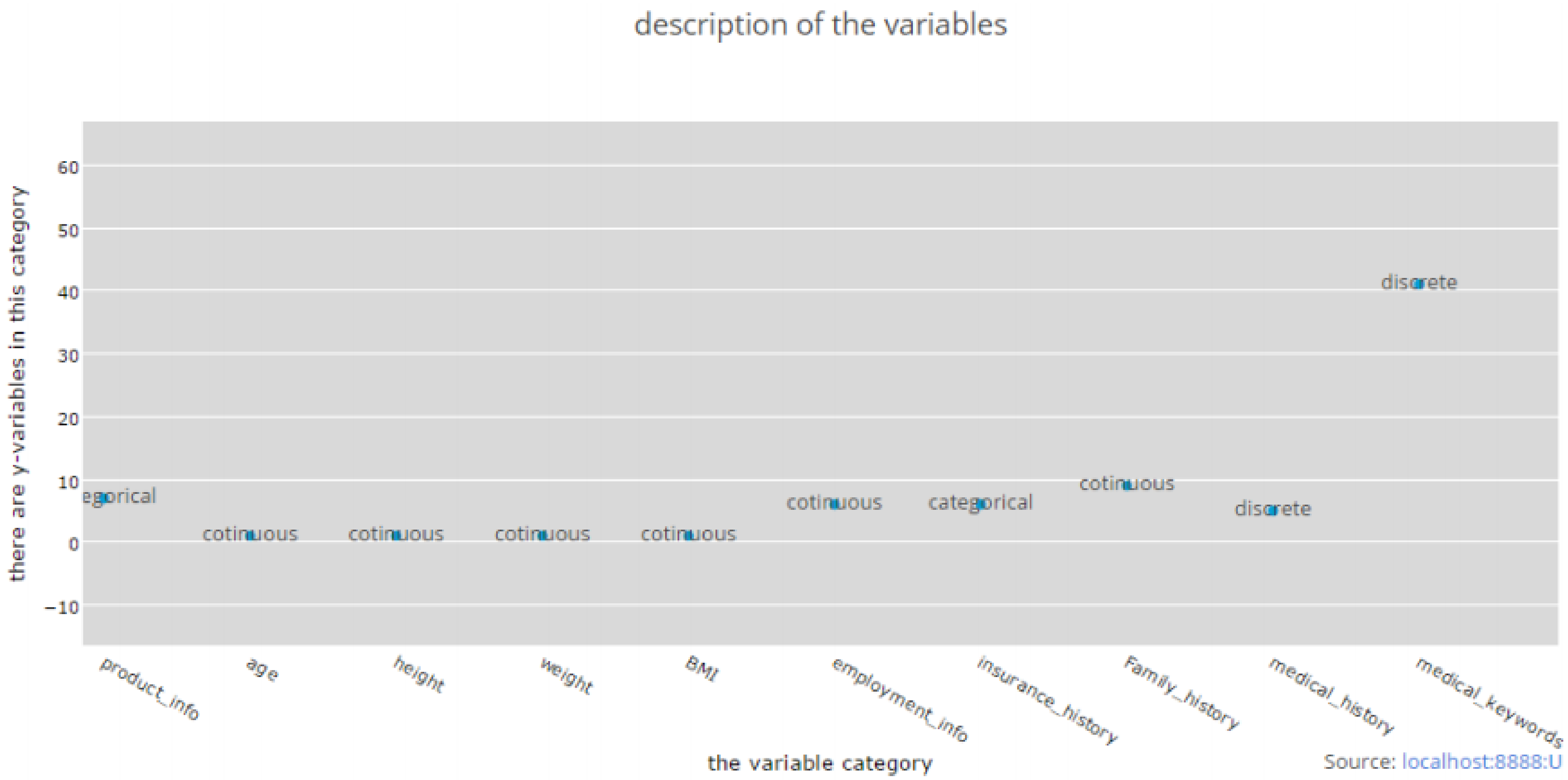


The layers:
The input layer connects with hidden layer/s, which in turn connects to the output layer.

PRUDENTIAL DATA ANALYSIS

- The data contains **110 categorical** variables, **13 continuous** variables and **5 discrete** variables
- Categorical variables contain a finite number of categories, such as (Product_Info_2)
- Discrete variables are numeric variables that have a countable number of values between two values, such as (Medical_History_10)
- Continuous variables are numeric variables that have an infinite number of values between two values, such as (BMI)

PREDICTIVE VARIABLES



DATA PREPARATION AND TRANSFORMATION

- Removing the columns that have more than or equal 70% of null values; and removing unnecessary Id column
- Replacing the null values in each column with the computed mean of that column
- Setting categorical columns to as a factor to get factors with n levels
- Converting the categorical columns into dummy variables using 1 to C method
- Fitting the linear regression model and taking the significant variables only

MODEL 1: NEURAL NETWORK MULTINOMIAL CLASSIFICATION

- Converting each level of "Response" categorical column into dummy variables using 1 to C method

```
# encoding/converting each level of "Response" categorical column to a separate column (converting the categorical columns into dummy variables using 1 to C method)
library(nnet)
Prudential_Data <- cbind(Prudential_final_Data[1:64], class.ind(as.factor(Prudential_final_Data$Response)))
# Set labels name
names(Prudential_Data) <- c(names(Prudential_final_Data)[1:64],"Response_1","Response_2", "Response_3", "Response_4", "Response_5", "Response_6", "Response_7", "Response_8")
```

- Scaling the 10 Continuous variables using Min-Max Normalization to be between 0 and 1

```
# Scaling the 10 Continuous variables using Min-Max Normalization to be between 0 and 1
scale <- function(x){ (x - min(x))/(max(x) - min(x)) }
Prudential_Data[, 1:10] <- data.frame(lapply(Prudential_Data[, 1:10], scale))
```

- Splitting dataset into train and test datasets using sample function

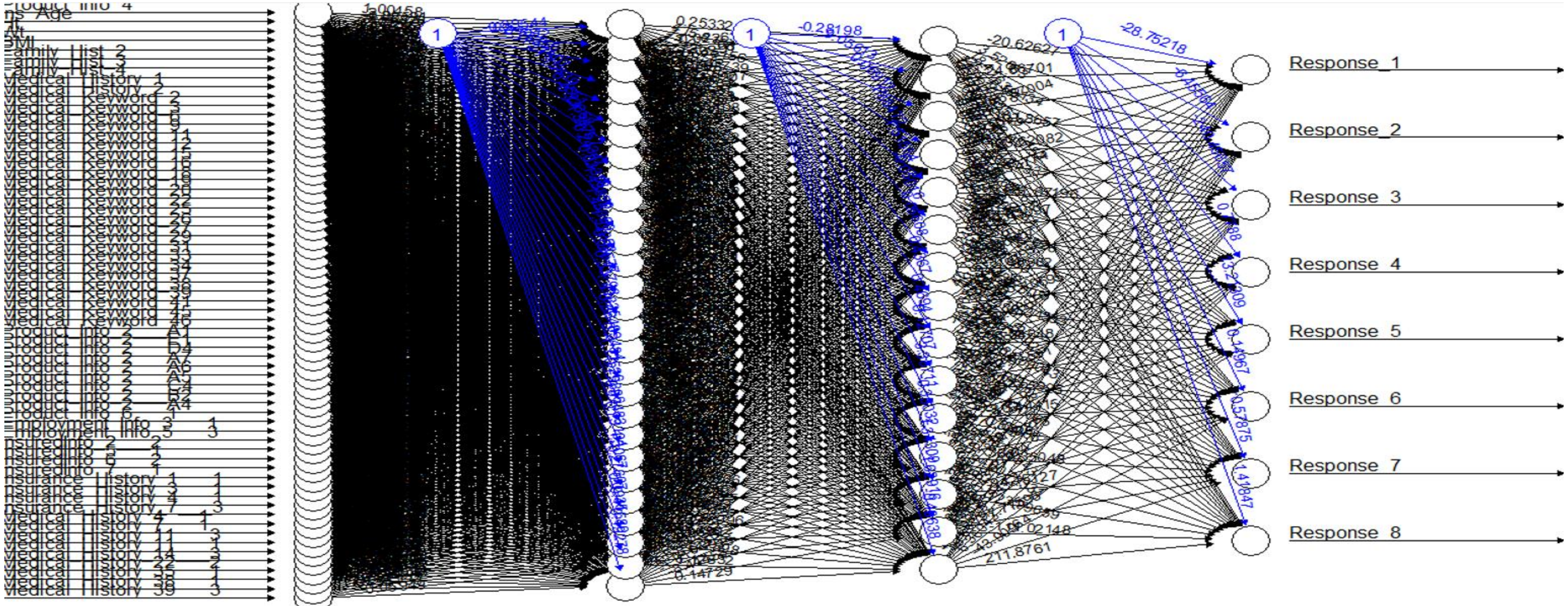
```
# running sample function to select randomly 80% index numbers of the dataset and u
sample_index <- sample(1:nrow(Prudential_Data),round(0.8*nrow(Prudential_Data))) #1
Prudential_train <- Prudential_Data[sample_index,] #80% of dataset is train data
Prudential_test <- Prudential_Data[-sample_index,] #20% of dataset is test data
```

MODEL 1: NEURAL NETWORK MULTINOMIAL CLASSIFICATION

- Fitting Neural Network on train dataset by choosing the hidden layer to be ⅔ of the input size.

```
n <- names(Prudential_train)
f <- as.formula(paste("Response_1 + Response_2 + Response_3 + Response_4 + Response_5 + Response_6 + Response_7 + Response_8 ~",
paste(n[!n %in% c("Response_1", "Response_2", "Response_3", "Response_4", "Response_5", "Response_6", "Response_7", "Response_8")], collapse = " + ")))
#Fitting Neural Network model on train dataset
nn <- neuralnet(f,data=Prudential_train,hidden=c(27,15),linear.output=FALSE, act.fct = "logistic", err.fct ="sse", lifesign = "full", stepmax=1e6) # 'e
# plot the neural network
plot(nn)
```

- Plotting the results



MODEL 1: NEURAL NETWORK MULTINOMIAL CLASSIFICATION

- Computing predictions for multi-class classification.
- The typical approach is to have 'n' output neurons represent the different classes
- In the end, the neuron which has the highest prediction 'wins' and that class is predicted

```
pr.nn <- compute(nn,Prudential_test[,1:64])
# Extract results
pr.nn_ <- pr.nn$net.result
```

- Computing the accuracy and the misclassification error

```
> #Computing accuracy
> actual_values <- max.col(Prudential_test[, 65:72]) # Find the maximum position for each row of a matrix.
> predicted_values <- max.col(pr.nn_) # Find the maximum position for each row of a matrix.
> Accuracy <- mean(predicted_values == actual_values)
> Accuracy
[1] 0.4
> #Computing Misclassification error
> Misclass_error <- 1-mean(predicted_values == actual_values)
> Misclass_error
[1] 0.6
```

MODEL 1: NEURAL NETWORK MULTINOMIAL CLASSIFICATION

- **Hidden Layers:**

Hidden Layers	(27,15)	(23,19)
Accuracy	40%	41%
Misclassification Error	60%	59%

MODEL 2: NEURAL NETWORK REGRESSION

- Scale the dataset and split it into train and test data

- Set parameters and train the train dataset

```
f <- as.formula(paste("Response ~",
                      paste(names(data_from_midterm)[!names(data_from_midterm) %in% "Response"], collapse = " + ")))
nn <- neuralnet(f, data = train_dataset, hidden=11, lifesign = "full",
                stepmax = 1e7, rep=1,
                learningrate.factor = list(minus = 0.5, plus = 1.2), threshold = 0.23,
                algorithm = "rprop+")
```

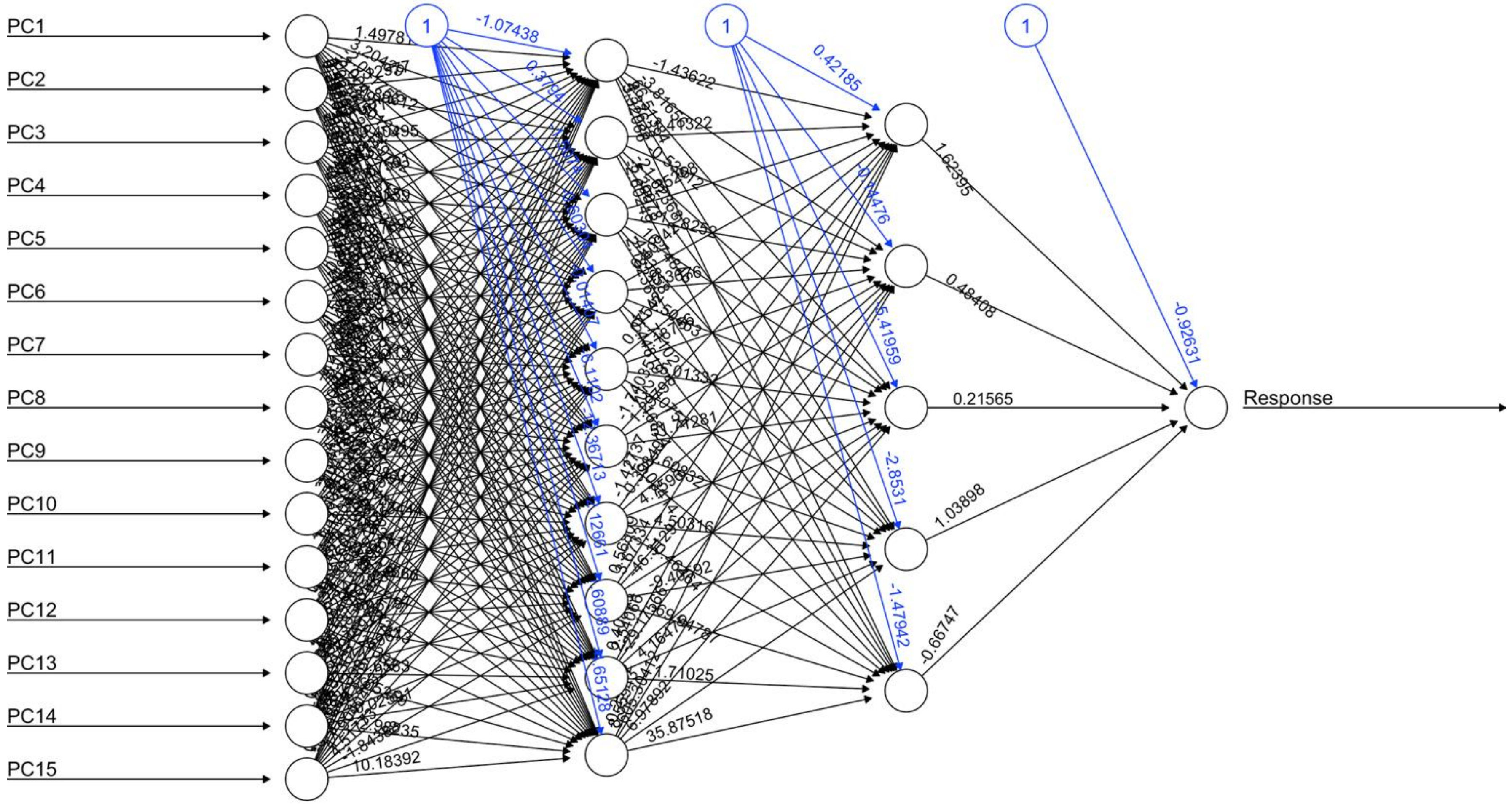
- Evaluate the performance

```
MSE.nn <- rmse(pr.nn - test.r); MAE.nn <- mae(pr.nn - test.r)
print(MSE.nn)
#[1] 2.320669455
print(MAE.nn)
#[1] 1.925377265
```

MODELS AND SUMMARY

Models	hidden layer	rep	learningrate	threshold	algorithm	MSE	MAE
model1	11	1	minus = 0.5, plus = 1.2	0.23	rprop+	2.320669455	1.925377265
model2	c(10,5)	1	minus = 0.5, plus = 1.2	0.15	rprop+	2.324929278	1.927565246
model3	c(8,5)	1	minus = 0.5, plus = 1.2	0.1	rprop+	2.322958041	1.926729142
model4	8	1	minus = 0.5, plus = 1.2	0.1	rprop+	2.322529671	1.927285048
model5	c(10,3)	10	0.005	0.003	backprop	3.384031069	2.340350218

DEMO PLOT OUTPUT



PART 2

CONVOLUTIONAL NEURAL NETWORK

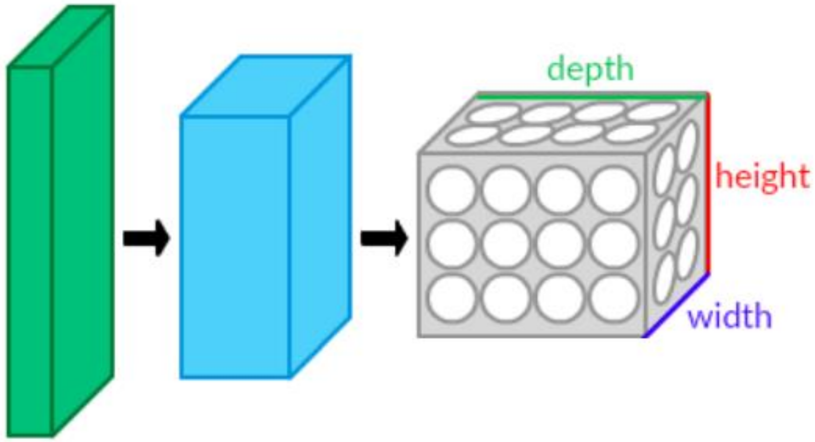
CONVOLUTIONAL NEURAL NETWORK

Convolutional neural network is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery.

A CNN consists of an input and an output layer, as well as multiple hidden layers

The layers of a CNN typically consist of

- **Convolutional Layer**
- **Pooling Layer**
- **Dropout Layer**
- **Fully connected Layer**



SEMEION DATA ANALYSIS

- Data Set Characteristics: Multivariate
- Number of Instances: 1593
- Number of Attributes: 256
- Associated Tasks: Classification

1593 handwritten digits from around 80 persons were scanned, stretched in a rectangular box 16x16 in a gray scale of 256 values.
Then each pixel of each image was scaled into a boolean (1/0) value using a fixed threshold.

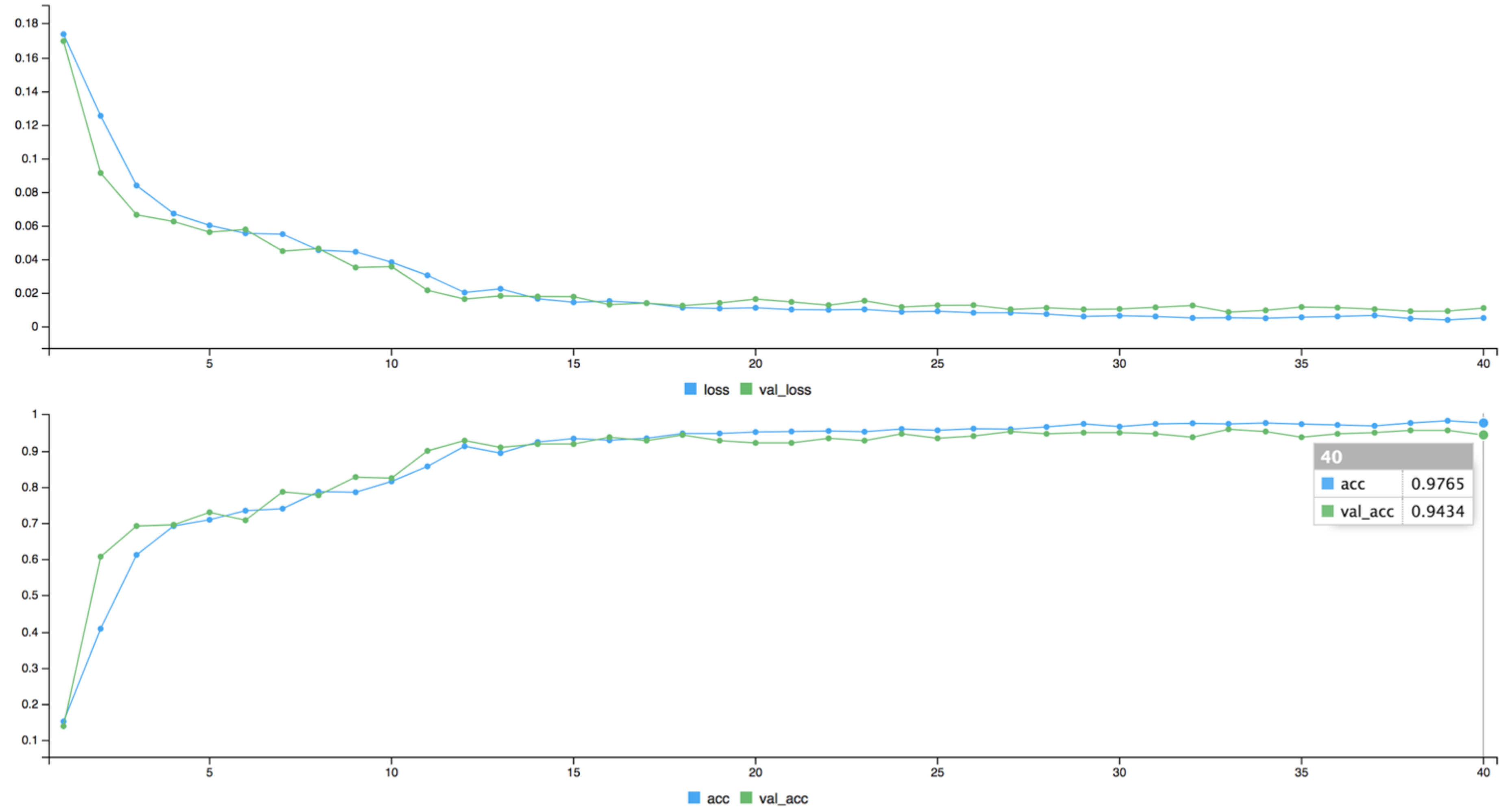
Each person wrote on a paper all the digits from 0 to 9, twice- first time in the normal way and the second time in a fast way

PROCESS AND CODE

- Setup keras environment in R
- Read semeion dataset
- Split into train and test dataset
- Add CNN layers

```
##Define a keras sequential model
model <- keras_model_sequential()

##Add CNN layer - our input is a 1D array
layer_conv_1d(model, nb_filters, kernel_size, padding='same', activation='relu',
              input_shape = input_shape) ###Convolutional layer1
layer_max_pooling_1d(model, pool_size) ###Pooling layer1

layer_conv_1d(model, nb_filters, kernel_size, activation='relu') ###Convolution layer2
layer_max_pooling_1d(model, pool_size) ###Pooling layer1
layer_dropout(model, 0.25) ###Dropout (Pervent overfitting)

layer_conv_1d(model, nb_filters, kernel_size, activation='relu') ###Convolution layer3
layer_max_pooling_1d(model, pool_size) ###Pooling layer3

layer_flatten(model) ###Always flatten the model before going into fully connected layer

layer_dense(model, 128, activation='relu')###Fully connected layer1
layer_dropout(model, 0.35) ###Dropout (Pervent overfitting)
layer_dense(model, 50, activation='relu')###Fully connected layer2
layer_dense(model, nb_classes, activation = 'softmax')###Fully connected layer3

#Check model summary
summary(model)

#Complile the model
compile(model, optimizer='RMSprop', loss='mean_absolute_error', metrics='accuracy')

#Train the model
fit(model, x_train, y_train, batch_size, epochs, validation_data=list(x_test, y_test))
```

SAMPLE OUTPUT



MODELS AND SUMMARY

model	model shape	batch_size	nb_filters	epochs	c1	c2	c3	fc1	fc2	fc3	compile optimizer	compile loss	d1	d2	accu_train	accu_test
1	c-c-mp-d-mp-f-fc-d-fc	4	32	20	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.2	0.9939	0.9434
2	c-c-mp-d-mp-f-fc-d-fc	8	32	20	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.2	0.9937	0.956
3	c-c-mp-d-mp-f-fc-d-fc	12	32	20	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.2	0.9922	0.9591
4	c-c-mp-d-mp-f-fc-d-fc	24	32	20	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.2	0.9945	0.9497
5	c-c-mp-d-mp-f-fc-d-fc	32	32	20	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.2	0.9929	0.9497
6	c-c-mp-d-mp-f-fc-d-fc	12	6	20	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.2	0.9624	0.9403
7	c-c-mp-d-mp-f-fc-d-fc	12	12	20	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.2	0.9804	0.9465
8	c-c-mp-d-mp-f-fc-d-fc	12	16	20	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.2	0.9969	0.9465
9	c-c-mp-d-mp-f-fc-d-fc	12	24	20	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.2	0.9914	0.9686
10	c-c-mp-d-mp-f-fc-d-fc	12	48	20	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.2	0.9929	0.9591
11	c-c-mp-d-mp-f-fc-d-fc	256	48	35	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.35	0.9663	0.956
12	c-c-mp-d-mp-f-fc-d-fc	256	32	35	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.35	0.96	0.9528
13	c-c-mp-d-mp-f-fc-d-fc	256	24	35	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.35	0.9459	0.9245
14	c-c-mp-d-mp-f-fc-d-fc	1275	48	35	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.35	0.8365	0.84
15	c-c-mp-d-mp-f-fc-d-fc	1275	24	35	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.35	0.84	0.8774
16	c-mp-d-c-mp-f-fc-d-fc	128	32	35	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.35	0.978	0.9434
17	c-mp-d-c-mp-f-fc-d-fc	128	48	35	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.35	0.9851	0.9528
18	c-mp-d-c-mp-f-fc-d-fc	256	32	35	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.35	0.9616	0.9497
19	c-mp-d-c-mp-f-fc-d-fc	256	48	35	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.35	0.9678	0.9371
20	c-mp-d-c-mp-f-fc-d-fc	12	24	35	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.35	0.9851	0.9497
21	c-mp-d-c-mp-f-fc-d-fc	24	24	35	relu	relu		relu	softmax		adadelta	categorical_crossentropy	0.2	0.35	0.9867	0.9528
22	c-mp-d-c-mp-f-fc-d-fc	128	48	35	elu	elu		elu	softmax		adadelta	categorical_crossentropy	0.2	0.35	0.9969	0.9434
23	c-mp-d-c-mp-f-fc-d-fc	128	48	35	LeakyReLU	LeakyReLU		LeakyReLU	softmax		adadelta	categorical_crossentropy	0.2	0.35	0.9937	0.9528
24	c-mp-d-c-mp-f-fc-d-fc	128	48	35	tanh	tanh		tanh	softmax		adadelta	categorical_crossentropy	0.2	0.35	1	0.9434
25	c-mp-d-c-mp-f-fc-d-fc	128	48	35	sigmoid	sigmoid		sigmoid	softmax		adadelta	categorical_crossentropy	0.2	0.35	0.0965	0.0849
26	c-mp-c-mp-d-c-mp-f-fc-d-fc-fc	128	32	40	relu	relu	relu	relu	relu	softmax	adadelta	categorical_crossentropy	0.2	0.35	0.9851	0.9528
27	c-mp-c-mp-d-c-mp-f-fc-d-fc-fc	128	32	40	tanh	tanh	tanh	tanh	tanh	softmax	adadelta	categorical_crossentropy	0.2	0.35	0.9922	0.934
28	c-mp-c-mp-d-c-mp-f-fc-d-fc-fc	128	32	40	sigmoid	sigmoid	sigmoid	sigmoid	sigmoid	softmax	adadelta	categorical_crossentropy	0.2	0.35	0.0881	0.0863
29	c-mp-c-mp-d-c-mp-f-fc-d-fc-fc	128	32	40	hard_sigmoic	hard_sigmoid	hard_sigr	hard_sigmoid	hard_sigmoid	softmax	adadelta	categorical_crossentropy	0.2	0.35	0.1122	0.0849
30	c-mp-c-mp-d-c-mp-f-fc-d-fc-fc	128	32	40	linear	linear	linear	linear	linear	softmax	adadelta	categorical_crossentropy	0.2	0.35	0.9867	0.9371
31	c-mp-c-mp-d-c-mp-f-fc-d-fc-fc	128	32	100	relu	relu	relu	relu	relu	softmax	SGD	categorical_crossentropy	0.2	0.35	0.9371	0.8902
32	c-mp-c-mp-d-c-mp-f-fc-d-fc-fc	64	128	100	relu	relu	relu	relu	relu	softmax	SGD	categorical_crossentropy	0.2	0.35	0.9631	0.9403
33	c-mp-c-mp-d-c-mp-f-fc-d-fc-fc	128	128	100	relu	relu	relu	relu	relu	softmax	SGD	categorical_crossentropy	0.2	0.35	0.9278	0.9308
34	c-mp-c-mp-d-c-mp-f-fc-d-fc-fc	128	32	40	relu	relu	relu	relu	relu	softmax	RMSprop	categorical_crossentropy	0.2	0.35	0.9843	0.9654
35	c-mp-c-mp-d-c-mp-f-fc-d-fc-fc	128	64	40	relu	relu	relu	relu	relu	softmax	RMSprop	categorical_crossentropy	0.2	0.35	0.9914	0.9403
36	c-mp-c-mp-d-c-mp-f-fc-d-fc-fc	128	128	40	relu	relu	relu	relu	relu	softmax	RMSprop	categorical_crossentropy	0.2	0.35	0.9922	0.9465
37	c-mp-c-mp-d-c-mp-f-fc-d-fc-fc	128	32	40	relu	relu	relu	relu	relu	softmax	RMSprop	mean_squared_error	0.2	0.35	0.9867	0.9497
38	c-mp-c-mp-d-c-mp-f-fc-d-fc-fc	128	128	40	relu	relu	relu	relu	relu	softmax	RMSprop	mean_squared_error	0.2	0.35	0.9937	0.9497
39	c-mp-c-mp-d-c-mp-f-fc-d-fc-fc	128	32	40	relu	relu	relu	relu	relu	softmax	RMSprop	mean_absolute_error	0.2	0.35	0.9671	0.956
40	c-mp-c-mp-d-c-mp-f-fc-d-fc-fc	128	128	40	relu	relu	relu	relu	relu	softmax	RMSprop	mean_absolute_error	0.2	0.35	0.9765	0.9434

CONCLUSION & FUTURE WORK

- **NEURAL NETWORK:**

MULTINOMIAL CLASSIFICATION	REGRESSION
Takes longer time to process	Processes in shorter time
Result is straightforward	Requires additional operations

- Less variables reduce the runtime
- Each dataset has different parameters and algorithms that works best

CONVOLUTIONAL NEURAL NETWORK:

- Data is the key

FUTURE WORK:

- deepnet and mxnet Package
- h20 package