A Modern Statistical Method – Convolutional Neural Network (CNN)

ST425 – Summative Project

# Candidate Numbers:

* 21013
* 16137
* 23074
* 20370

Table of Contents

[*Candidate Numbers:* 1](#_Toc56970254)

[Meetings Log 3](#_Toc56970255)

[A Modern Statistical Method 4](#_Toc56970256)

[What is a Convolutional Neural Network? 4](#_Toc56970257)

[History 4](#_Toc56970258)

[Technical Interpretation 5](#_Toc56970259)

[Basic Structure 5](#_Toc56970260)

[Feature Learning Network 5](#_Toc56970261)

[Data Representation 5](#_Toc56970262)

[Convolutional Layers 5](#_Toc56970263)

[Pooling Layers 7](#_Toc56970264)

[Padding 7](#_Toc56970265)

[Classification Network (ANN) 8](#_Toc56970266)

[Activation function 9](#_Toc56970267)

[CNN Training 9](#_Toc56970268)

[Loss Function 9](#_Toc56970269)

[Backpropagation 10](#_Toc56970270)

[Example - AlexNet: 10](#_Toc56970271)

[Multivariate normal distribution 12](#_Toc56970272)

[R code: 12](#_Toc56970273)

[Results: 16](#_Toc56970274)

[Q1 16](#_Toc56970275)

[Q2 16](#_Toc56970276)

[Q3 17](#_Toc56970277)

[Q4 17](#_Toc56970278)

[17](#_Toc56970279)

[Bibliography 18](#_Toc56970280)

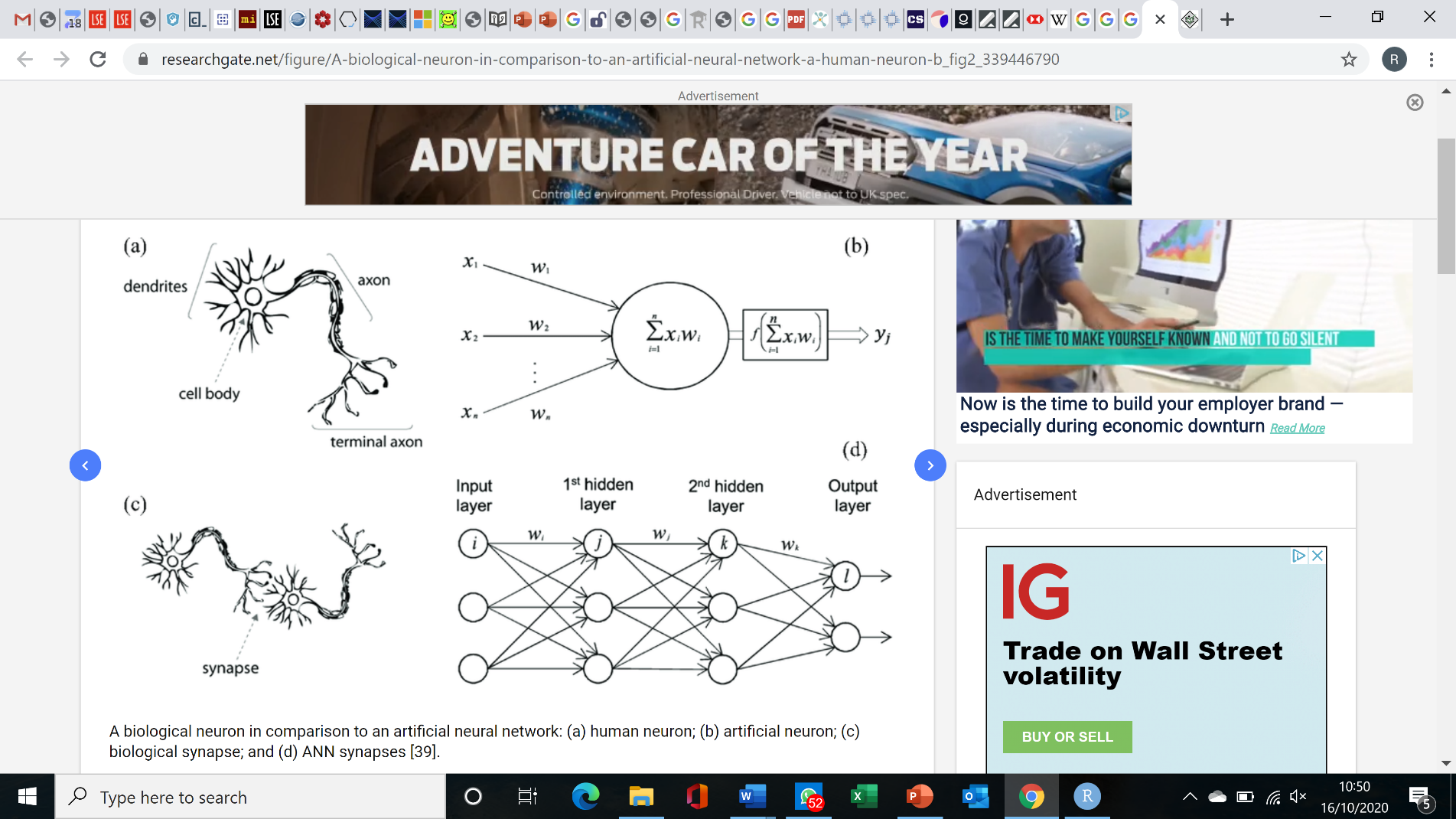
# Meetings Log

|  |  |
| --- | --- |
| Date | Comments |
| 09/10/2020 | * Initial team meeting. * Discussed the work distribution and sub-team allocation. * Discussed the designated deadline for review of works prior to submission. |
| 11/10/2020 | * Quick updates on individual’s understanding on assignment questions and roles |
| 16/10/2020 | * Interim 1 meeting. * Stats update and sharing of knowledge regarding the unclear parts of questions. |
| 24/10/2020 | * Interim 2 meeting. * Stats update and sharing of knowledge regarding the unclear parts of questions. |
| 06/11/2020 | * Interim 3 meeting. * Drafted semi-final version of assignment. * Designation of final review date: 20/11/2020 |
| 20/11/2020 | * Final review meeting * Confirmed the works from each sub-team. * Agreed on submission of assignment. * Quick rehearsal of PPT slides for oral presentation. * Final rehearsal is scheduled at the same day as the actual presentation. |

# A Modern Statistical Method

## What is a Convolutional Neural Network?

An Artificial Neural network (ANN) is a statistical learning technique designed to mimic the neurological processes of a human brain, thus, ‘learning’ the relationship between outcomes and inputs similarly to that of a human. ANN’s are comparable to that of biological neural networks. A simple ANN is comprised of nodes which are connected across ‘layers’ in the manner illustrated by **d** in **figure 1** below.



***Figure 1***

An Illustration comparing a biological neuron and a node in an ANN (a vs b) and a connection between 2 biological neurons and a simple ANN.

Source: Meng, Ancey, Hu (2020)

A convolutional neural network (CNN) is a specific class of ANN, initially designed to capture spatial information from image and video data. Traditional applications include object detection, autonomous driving, medical image classification etc. However, CNN’s are increasingly being used with non-spatial data in fields such as natural language processing and time series forecasting (specifically within healthcare and finance).

## History

In 1959 Hubel and Wiesel identified two types of cells in the mammalian visual cortex, “simple cells” and “complex cells”, which are used in pattern recognition. The mechanism behind the interaction of these cells are, complex cells respond to the information collected from a cluster of simple cells, gathering the information from them together, “summing-up” the information (Draelos, 2019).

In the 1980s Dr. Kunihiko Fukushima was inspired by their work, and he has tweaked their model into “neocognitron” model. This model includes non-biological “cells”, that are mathematical equations that work like the visual recognition pattern in biological organisms (Draelos, 2019.

Lecun et al (1998) utilized the first working CNN, in handwritten character recognition. Their CNN clustered simple features into continuously more complicated features. In other words their CNN were designed to identify the individual features of a dataset which, when aggregated, would capture the collective features of the dataset. The dataset they have used, MNIST, is now one of the most recognizable datasets.

## Technical Interpretation

### Basic Structure

A CNN consists of a feature extraction network and a classification network. The feature extraction network identifies unique features of the input data and reduces the dimensionality of the input data whilst the classification network (generally a simple ANN as mentioned above) classifies the output depending on the extracted features.

## Feature Learning Network

### Data Representation

The data used for the feature extraction network in a CNN is represented using 4-dimensional *tensors*, defined as:

Where: indicates the layer the tensor is the output of (when the tensor is the input data); indicates a tensor; is the number of datasets; is the hight of the data; is the width of the data and is the depth/channels of the data after the layer in the network(Michelucci, 2019, Aggarwal, 2018).

After the first layer, the tensors are typically referred to as feature maps. Furthermore, the depth of a feature map depth is often referred to as the number of *channels*. Additionally, is the output of the layer and the input to the layer.

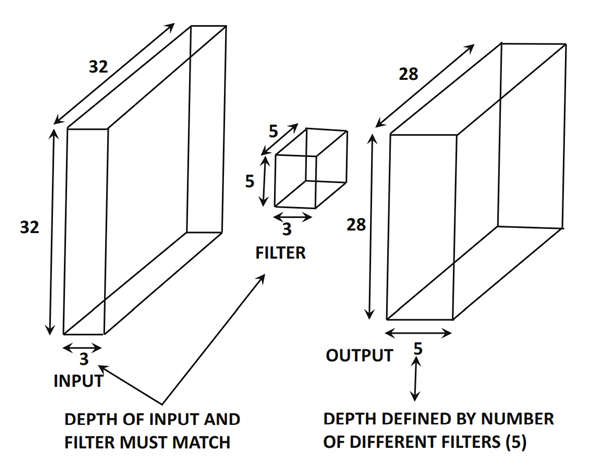
For example, if the input data to a CNN was n grey images with pixels each, the input data would be represented as: .[[1]](#footnote-1)

## Convolutional Layers

A convolutional layer is used to extract individual features of the input data by conducting convolutional operations using a *kernel.* At least 1 kernel will be applied to the input data in each convolutional layer. A kernel is a 3-dimensional ‘filter’ defined by:

Where: indicating the tensor that the kernel is applied on ( for the first layer); represents the kernel in a specific layer; is the height; is the width and is the depth of the kernel. Further =

The depth of a kernel must be equal to the depth of the input tensor that it is being applied to and the depth of the output tensor is equal to the number of applied kernels (Michelucci, 2019, Aggarwal, 2018).



***Figure 2***

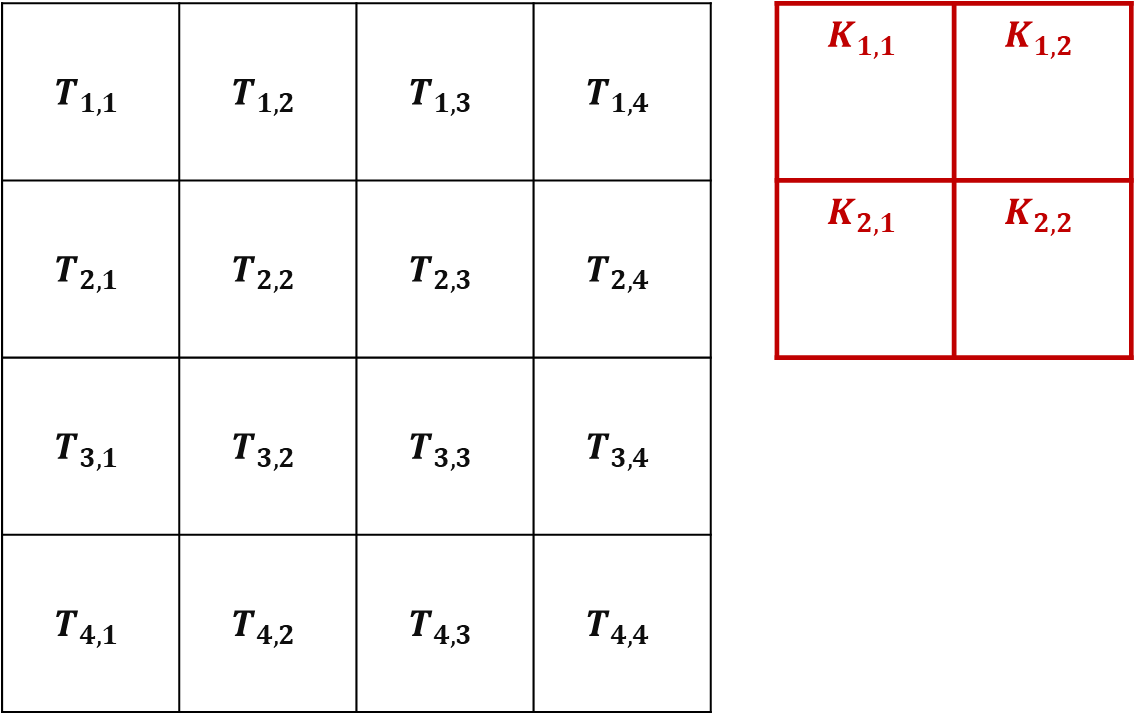
High level overview of a convolutional operation with a stride of 1 and no padding

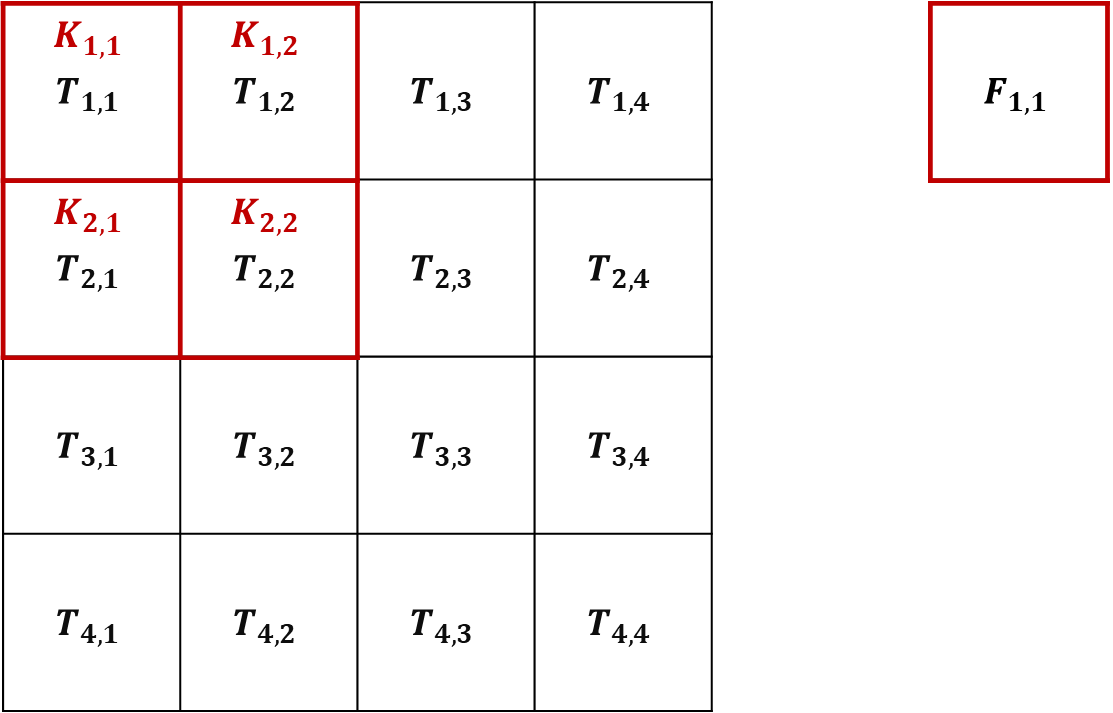
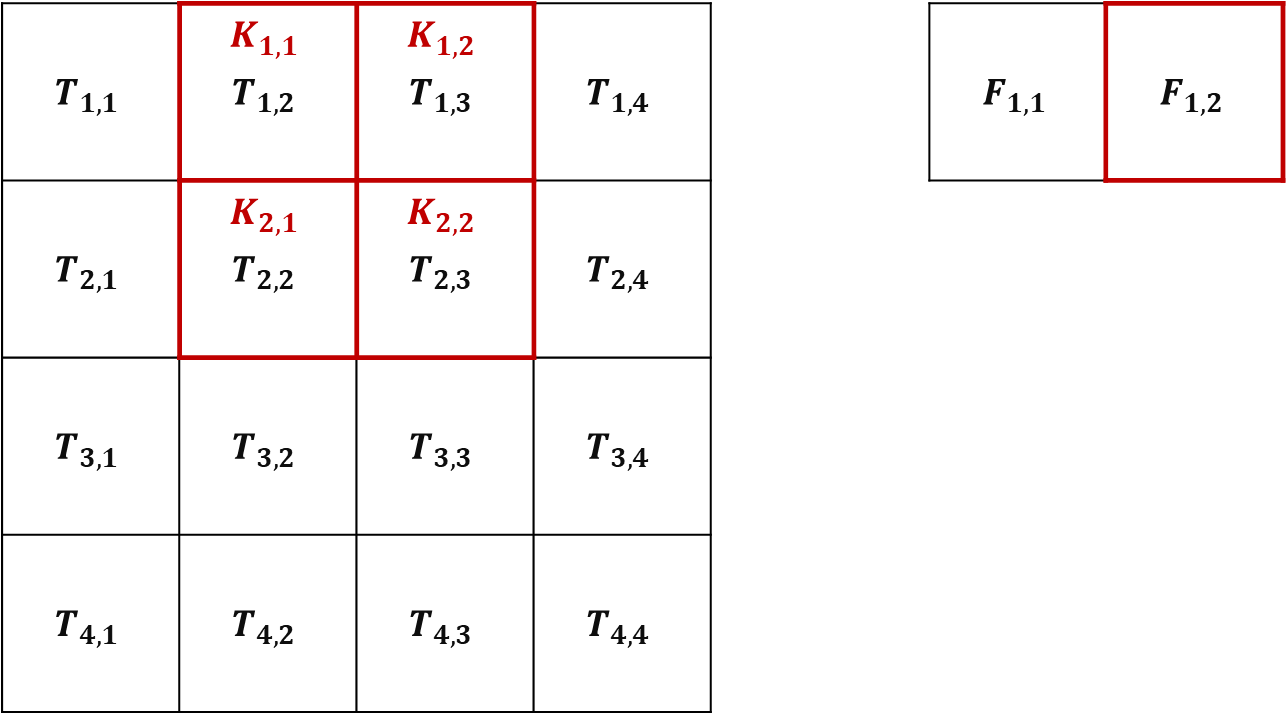
Source: Aggarwal (2018)

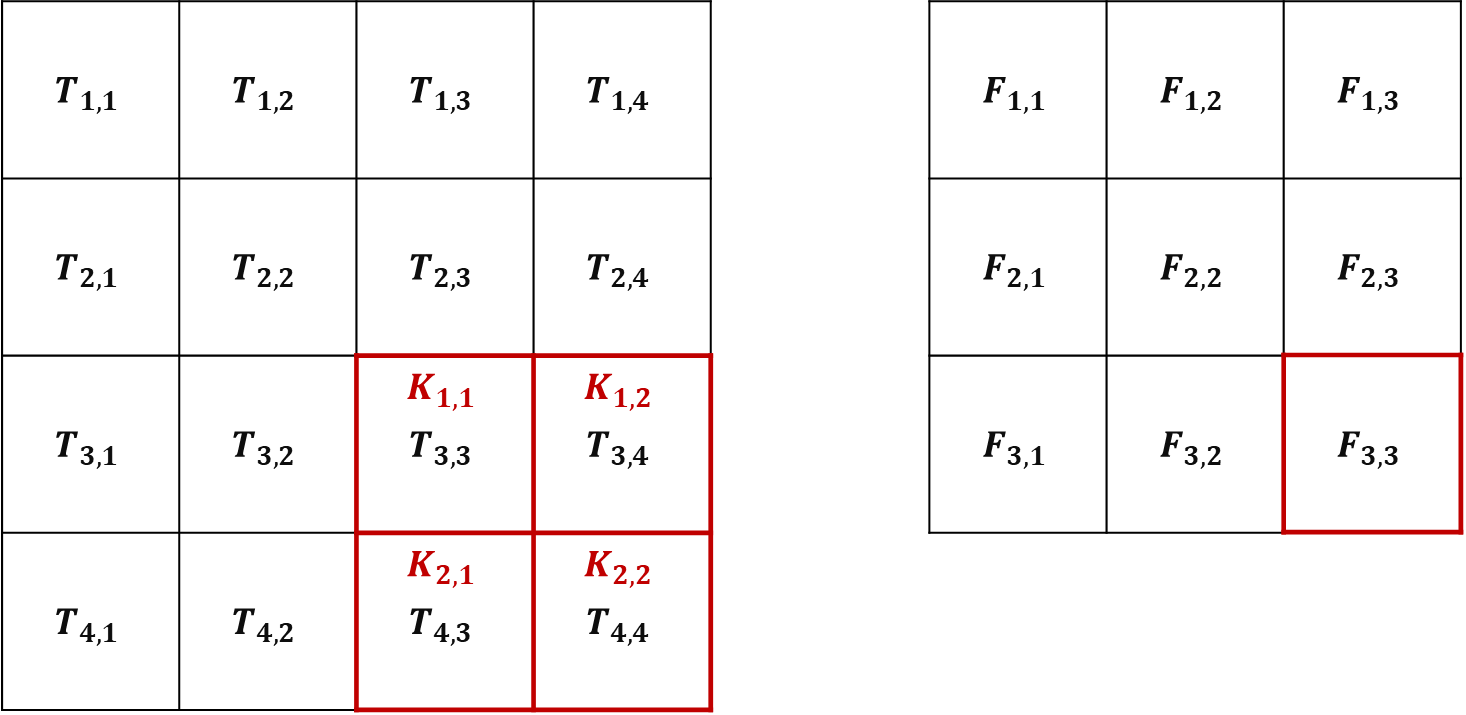
During a convolutional operation, the kernel is ‘moved’ across the input data, where a dot product of the kernel and input data is ‘fed into’ an activation function , producing an element of the output tensor. Consider an input tensor and a convolutional kernel illustrated by:

***Figure 3***

Representation of a one stride, non-padded convolutional operation. Each tensor element represents the input data. The convolutional kernel elements are the kernel weights. The output tensor is represented by .



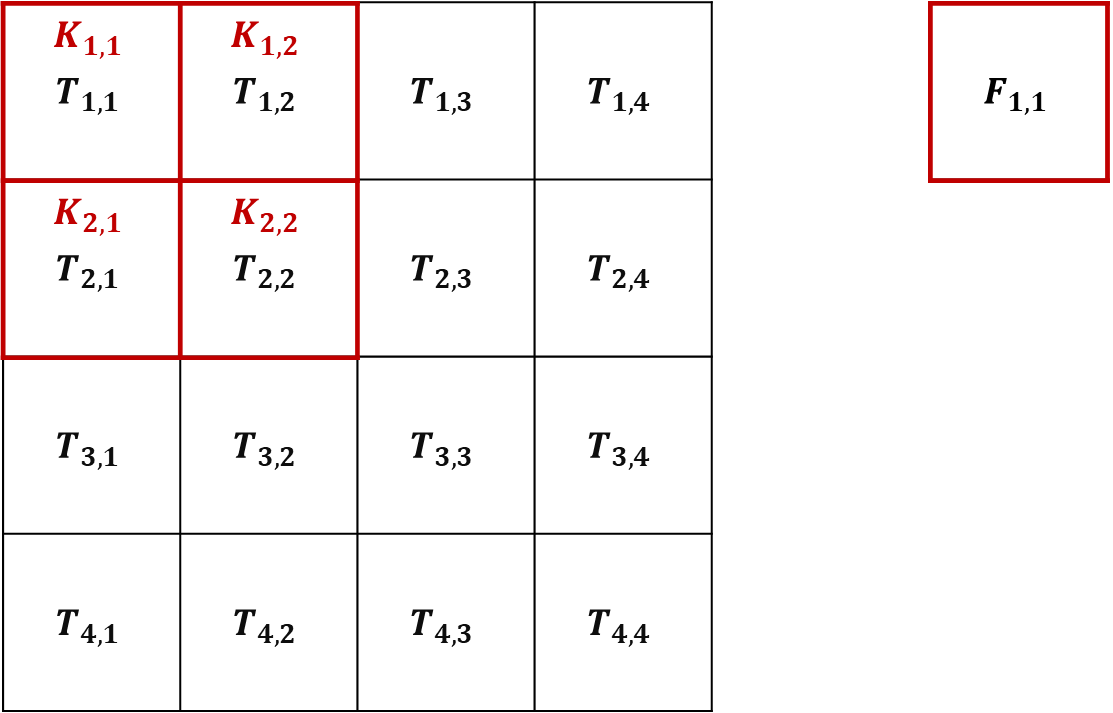
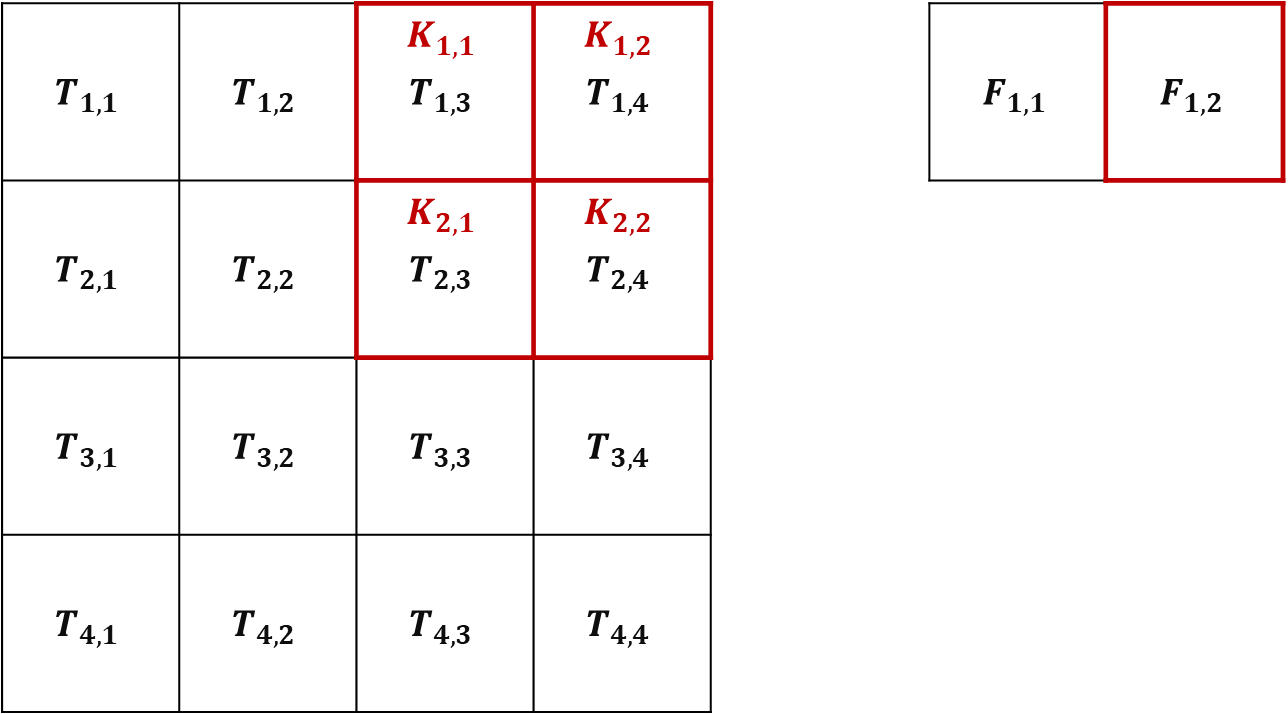




Where:

The height and width of each kernel is generally much smaller than that of the input tensor as smaller kernels can detect smaller features in the data. However, small kernels can be computationally expensive as more dot product operations are performed.

A *stride* is a hyperparameter describing the movement of kernels at a layer in the network (kernels from the same layer must have the same stride). A single stride kernel is one which moves across and down the input tensor one element at a time, whilst a stride kernel skips elements between dot product operations. Below is a 2-stride convolutional operation:



## Pooling Layers

Pooling operations are used to reduce the dimensionality of the input tensor. A pooling kernel is defined as:

Where: indicating the tensor that the kernel is applied on ( for the first layer); represents the kernel in a specific layer; is the hight; is the width.

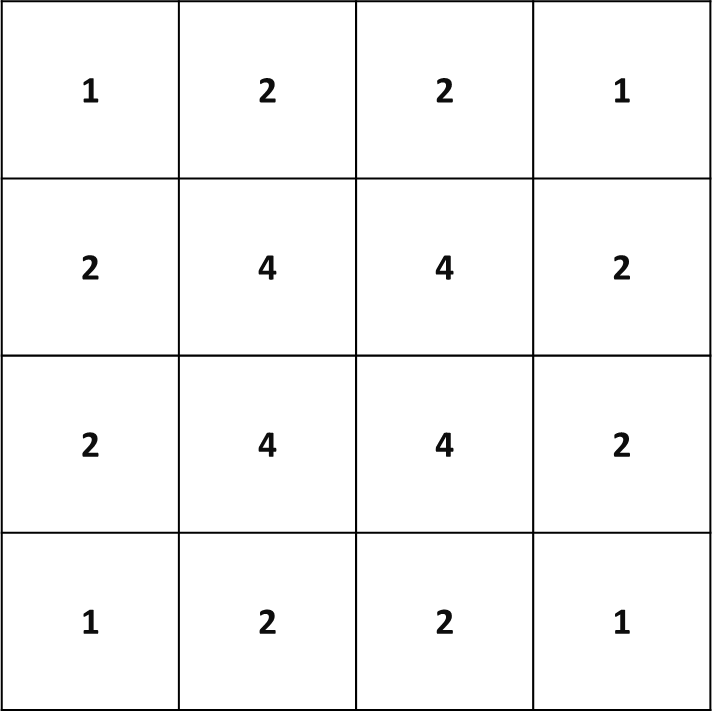
Similarly, to convolutional operations, pooling kernels ‘move’ across and down the input tensor. However, unlike convolutional kernels which have a depth dimension, pooling kernels do not. Consequently, pooling operations are also applied to each ‘layer’ of a tensor separately. Thus, output tensors have the same depth as input tensors.

The main types of pooling operations are max-pooling and average-pooling. Consider max-pooling, rather than performing a dot product operation (like a convolutional layer) the pooling operation applied will select the maximum value of all the elements from the input tensor in the dimensions of the pooling kernel and assign that value to the output tensor element. Average pooling will find the average of the tensor elements rather than the maximum. Whether max or average pooling is used is dependent on the data set.

## Padding

One can notice a shortcoming of the above convolutional operation. Some input tensor elements appear in the dot products of the output tensor more so than other elements.

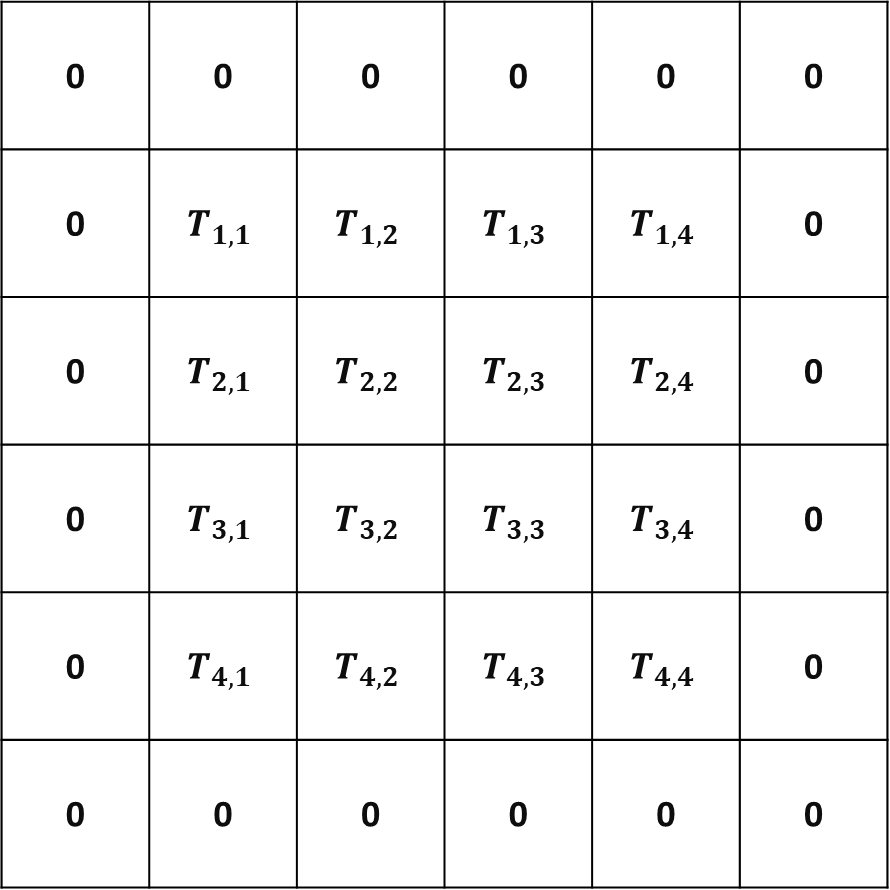
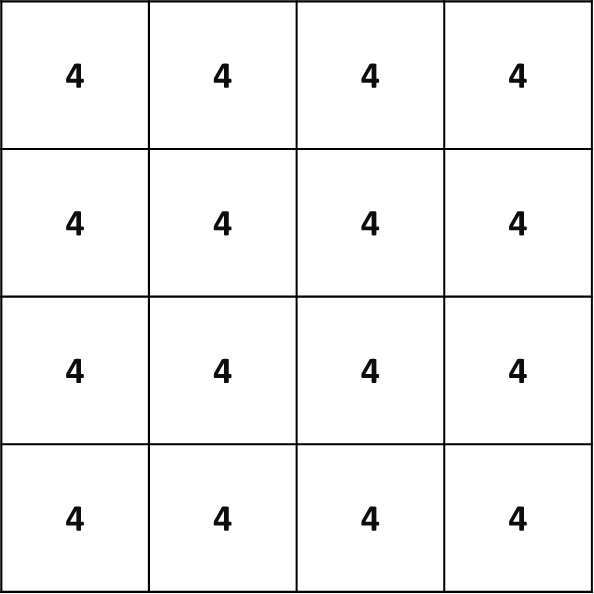
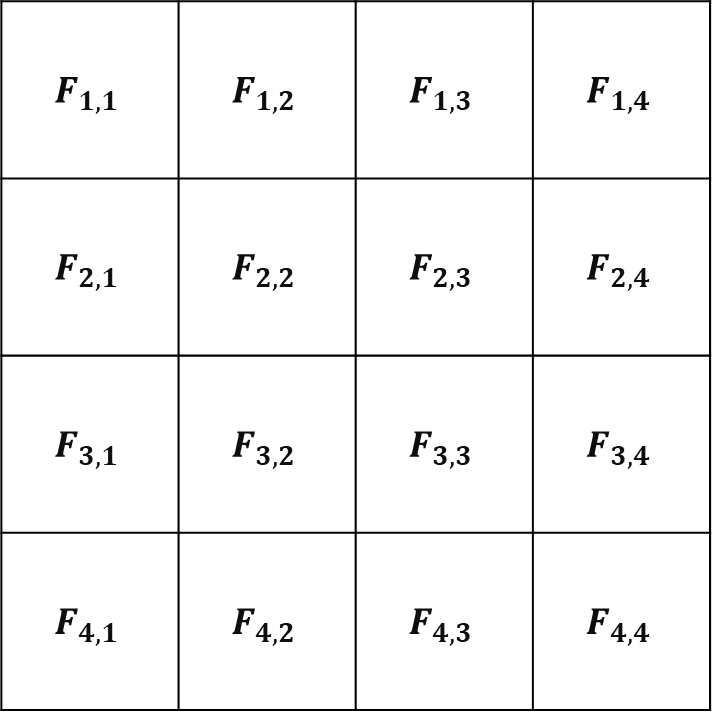
Consequently, features on the outside of the input tensor are less represented in the output tensor, potentially reducing the predictability of the CNN.



***Figure 4***

Tally of input tensor elements that appear in the output tensor dot-products. appears only once in the output

*Padding* adds elements to the outside of the input tensor, assigning them a value of 0. Subsequently, padded elements will not affect the dot product of the kernel and the re-adjusted input tensor, thus not affecting whether features are added or not. **Figure 5** illustrates full padding with single stride convolutions.



***Figure 5***

The results of full padding for the above convolution. is now represented 4 times in various output elements.

## Classification Network (ANN)

A simple ANN (as described in the introduction and illustrated in **figure 1**) is generally used as the structure of the classification network. The classification network is used to classify the output tensor of the feature extraction network in different categories. Generally, a CNN is used to make univariate classifications, which means that the ultimate objective of the CNN is to classify the input data into a single *class*.

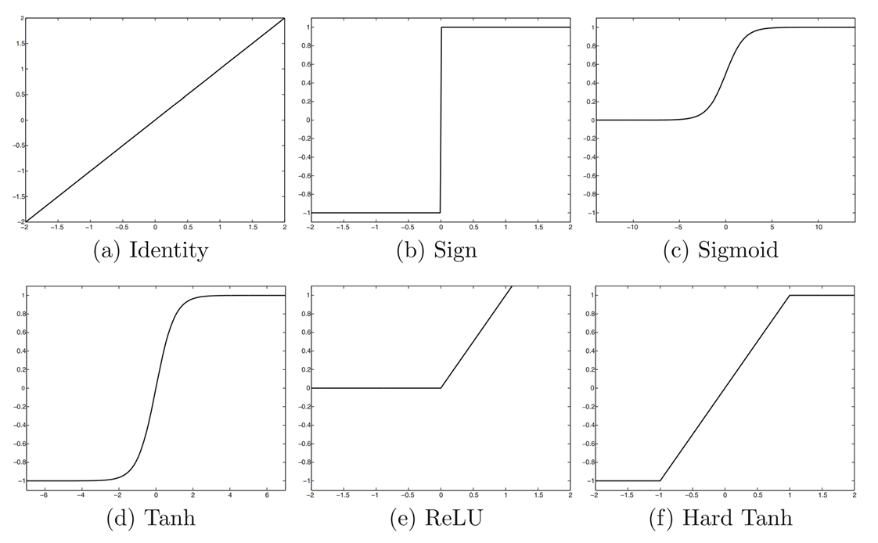
The output layer of an ANN must have the same number of nodes as there are classes. Additionally, it is beneficial for the output node of a univariate classification CNN to be represented as a ‘probability of classification’ . As such, a *softmax* transformation is used in the output layer to transform the output node value to a probability as below.

Where is the total number of classification categories; is the estimated probability of output node and is the estimated value of output node .

Consequently, the entire CNN takes the form of **figure 7**.

### Activation function

The activation function is a very important component of any neural network, commonly used activation functions are shown in figure 6.



***Figure 6***

ReLU is commonly used in CNN’s and closely replicates the biological activation as the function only has an output if a certain threshold is achieved.

## CNN Training

ANN’s are typically trained by minimising a *loss function* through a process known as *backpropagation* over all the training data.

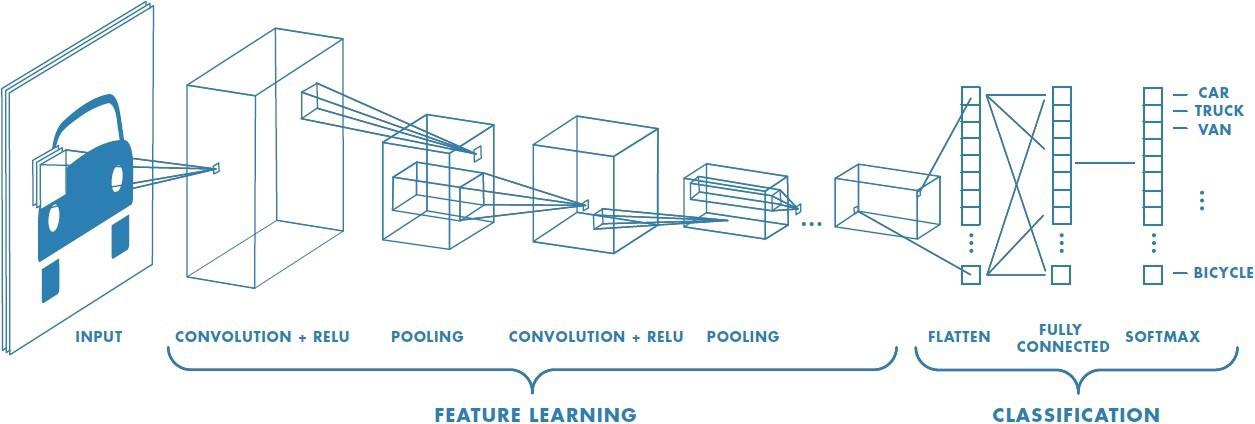
### Loss Function

In the case of univariate classification, the loss function will generally be a *cross-entropy loss* function:

Where is the total number of classification categories; is the estimated value of output node (after the softmax transformation) and depending on the class that the input data is assigned to.

For instance, if the input data is classified as a car (as in **figure 6**) then (where represents the car class) will have a value of 1 and for . Therefore, the loss function simplifies to:

Thus, the loss function will be increased if the estimated probability of classification is low for the class that the input data is associated with.



***Figure 7***

The full structure of a CNN

Source: Convolutional Neural Network, Mathworks

## Backpropagation

An ANN is trained through a process called *backpropagation*. The objective of backpropagation is to minimise the loss function by adjusting the weights by the derivative of the loss function with respect to the specific weights multiplied by a constant:

Where: is the weight of the network; is the loss function; and is some constant.

Before any training data is used, the network weights are assigned values either randomly or through human estimation. The first batch of training data is then fed into the network and the loss function is calculated. The derivative of the loss function with respect to each weight is calculated and the weights are adjusted as outlined above. The process is repeated until the network is reasonably trained/the training data. The training data is often ‘recycled’ again and again until loss function of the network ceases to be improved.

## Example - AlexNet:

AlexNet is one of the most recognised and cited CNN’s developed. It was developed to illustrate the importance of a CNN’s depth in computer vision accuracy. Further, it was one of the first CNN’s to use GPUs during training, making it computationally feasible to train a network of AlexNet’s depth in a timely manner. The model consists of eight layers, with the first five layers being the feature extraction network, followed by three classification layers Aggarwal (2018).

The input tensor to AlexNet is an image of . Five convolutional operations are then performed in the feature extraction network:

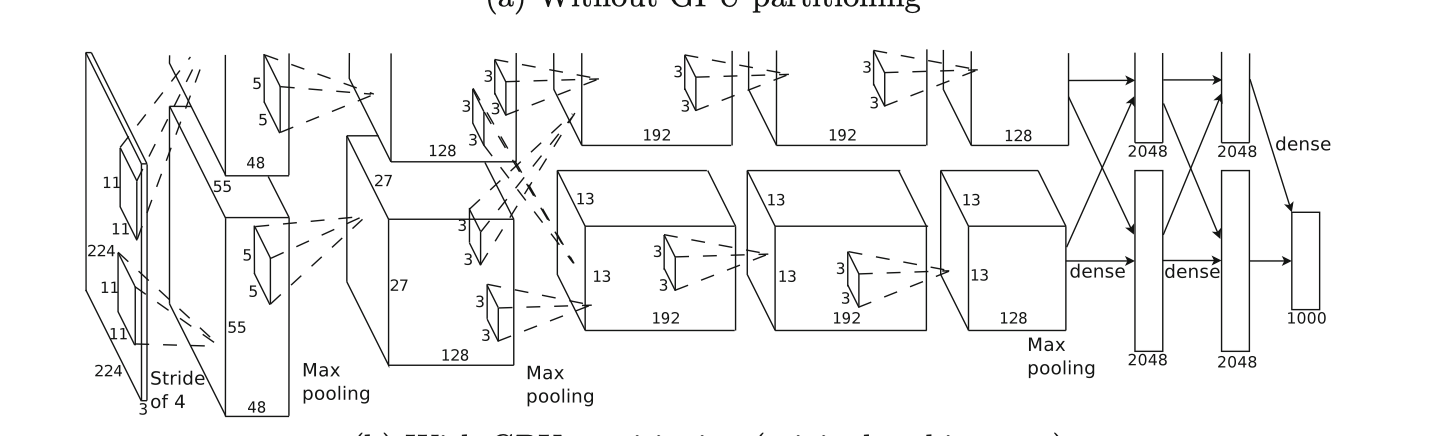
1st layer: x 96 kernels, with a stride of 4.

2nd layer: x 256 kernels.

3rd layer: x 384 kernels.

4th layer: x 384 kernels.

5th layer: x 256 kernels.



***Figure 8***

Simpilified tructure of AlexNet

Source: Aggarwal (2018)

The ReLU activation function was shown to produce the best results and was thus was used in each convolution operation.

The feature extraction network is divided into 2 parts so that the CNN could be trained using parallel computing techniques, reducing training time. After the 1st, 2nd and 5th convolution operations, max-pooling was applied with kernels of size and a stride of 2, to reduce dimensionality.

The classification network was comprised of 3 fully connected layers. The first two layers had 4096 neurons each. The final layer of AlexNet used 1000-way softmax to perform classification.

Words: 1,999

# Multivariate normal distribution

## R code:

|  |
| --- |
| require(reshape2)  require(MASS)  # Group 12  # PART II  #### QUESTION 1  set.seed(5) # Identical result for random sequence generation.  rMNorm = function(n, mu, vars, rho) {  #' Generate p-variate normal distribution with mean 'mu' and variance 'vars  #' with correlation between the i-th and the j-th components ρ^(|j−i|),  #' where |ρ| ≤ 1 and 1 ≤ i, j ≤ p.  #'  #' @param n (int): Number of repetition.  #' @param mu (array): p-dimensional array of mean value of n-variate.  #' @param vars (array): p-dimensional array of variance of n-variate.  #' @param rho (float): Correlation value for the n-variate functions.  #'  #' @return The n x p variate normal distribution matrix.  dim = length(mu)  # Generating empty correlation matrix.  corr = matrix(0,  nrow=dim,  ncol=dim)  # Correlation matrix that has ρ^([j-i]) for each i-th and j-th component.  for (i in 1:dim) {  for (j in 1:dim) {  corr[i,j] = rho^(abs(i-j))  }  }  # print("Correlation matrix.")  # print(corr)  # cat("\n")  Sigma = vars^0.5 %\*% t(vars^0.5) \* corr # Covariance matrix.  # print("Covariance matrix.")  # print(Sigma)  # cat("\n")  eigsys = eigen(Sigma, symmetric=T) # eigen system of Sigma.  evals = eigsys$values # eigen values.  Gamma = as.matrix(eigsys$vectors) # matrix of eigen vectors.  D = Gamma\*0  for(i in 1:dim) D[i,i] = sqrt(evals[i])  rootSigma = Gamma %\*% D %\*% t(Gamma) #matrix root of Σ = ΓΛΓ′  Z = matrix(rnorm(n\*dim), byrow=T, ncol=dim) # Z is n\*dim matrix with elements drawn from N(0,1)  X = matrix(rep(mu, n), byrow=T, ncol=dim) + Z %\*% rootSigma  # each row of eX is N(mu, Sigma)  return (X)  }  # p-variate normal distribution (p=3).  n = 1000000 # number of replications  mu = c(0, 0, 0) # An array of mean for p-variate.  vars = c(1,2,3) # An array of variance for p-variate.  rho = 0.5 # Correlation value (Condition written on question).  random\_seq = rMNorm(n, mu, vars, rho) # Generating p-variate normal distribution.  var(random\_seq)  # Correlation of each i-th and j-th corresponds to the condition outlined on the question 1.  cor(random\_seq)  #### QUESTION 2  # Arguments & Conditions  n = 1000000 # number of replications  p = 8  mu = c()  for (i in 1:p){  mu[i] = (-1)\*\*i  } # An array of mean for p-variate.  vars = rep(1.5,p) # An array of variance for p-variate.  rho = seq(-0.95, 0.95, by=0.05) # ρ ∈ [−0.95, 0.95]  # P1  estprob\_1 = c() # Vector for estimated probability results.  for (i in seq\_along(rho)){  X = rMNorm(n, mu, vars, rho[i])  mask = list() # A list of masks for Xj ∈ (−1,4), j = 1,··· ,8.  for (j in 1:p){  mask[j] = list(-1<X[,j]& X[,j]<4) # Xj ∈ (−1,4), j = 1,··· ,8  }  estprob\_1[i] = sum(Reduce("&", mask))/n  cat("Estimated probability for ρ:",rho[i], "=", estprob\_1[i], "\n")  }  plot(rho, estprob\_1, col="red",  main="P1", xlab="ρ", ylab="Probability")  # P2  estprob\_2 = c() # Vector for estimated probability results.  for (i in seq\_along(rho)){  X = rMNorm(n, mu, vars, rho[i])  t = rowSums(abs(X)) # Absolute sum of all the p variate distributions.  estprob\_2[i] = length(t[t<=8])/n # Probability of Abs sum less or equal to 8.  cat("Estimated probability for ρ:",rho[i], "=", estprob\_2[i], "\n")  }  plot(rho, estprob\_2, col="red",  main="P2", xlab="ρ", ylab="Probability")  #### QUESTION 3  # Arguments & Conditions  n = 1000000 # number of replications  p = 3  mu = rep(0,p)  vars = rep(0.5,p) # An array of variance for p-variate.  rho = seq(-1, 1, by=0.05) # Simulation of correlation coefficient increasing.  estprob\_3 = c()  for (i in seq\_along(rho)){  X = rMNorm(n, mu, vars, rho[i])  # P3 : P(ρ)=P(X1 >0,X3 >0|X2 >0)  # Bayes' theorem  denom =sum(X[,1]>0 &  X[,2]>0 &  X[,3]>0)/n  nom = sum(X[,2]>0) / n  estprob\_3[i] = denom/nom  cat("Estimated probability for ρ:",rho[i] , "=", estprob\_3[i], "\n")  }  plot(rho, estprob\_3, col="red",  main="P3", xlab="ρ", ylab="Probability")  estprob\_4 = c()  for (i in seq\_along(rho)){  X = rMNorm(n, mu, vars, rho[i])  # P4 : P(ρ)=P(X1 >0,X3 >0|X2 <0)  # Bayes' theorem  denom =sum(X[,1]>0 &  X[,2]>0 &  X[,3]<0)/n  nom = sum(X[,2]<0) / n  estprob\_4[i] = denom/nom  cat("Estimated probability for ρ:",rho[i], "=", estprob\_4[i], "\n")  }  plot(rho, estprob\_4, col="red",  main="P4", xlab="ρ", ylab="Probability")  #### QUESTION 4  # Arguments  mnData = read.csv("https://raw.githubusercontent.com/Anko-Jipsa/statistics/master/ST425/mnData.txt", sep="")  mnData = data.matrix(mnData)  mle\_est = function(data){  #' Run grid search MLE to find optimal parameters.  #'  #' @param data (array): Normally distributed sequences.  #' @return (vector): A vector contains optimal parameter values.  # Sample  y = data  # MLE for mu  # Likelihood function is maximized if mu is equal to the sample mean (e.g. mu = sum(x)/n)  mu = colMeans(y)  # Parameters setup  fixed\_variance = rep(1,length(mu)) # Sigma^2 = 1  dim = length(fixed\_variance)  corr = matrix(0, nrow=dim, ncol=dim)  k = length(mu)  n = nrow(y)  # MLE estimation for varying "ρ"  # Grid search method to find optimal "ρ" for constructing covariance matrix that maximises Log-likelihood  rho\_list = seq(from=-0.99, to=0.99, by=0.01)  ll\_list = rep(0, length(rho\_list)) # LL values for each grid search  for (x in seq\_along(rho\_list)){  iter\_rho = rho\_list[x]  # Constructing covariance matrix  for (i in 1:dim) {  for (j in 1:dim) {  corr[i,j] = iter\_rho^(abs(i-j))  }  }  cov\_mat = sqrt(fixed\_variance) %\*% t(sqrt(fixed\_variance)) \* corr  LL\_third\_comp = rep(0, n)  for (j in 1:n){  LL\_third\_comp[j] = t(y[j,]-mu) %\*% ginv(cov\_mat) %\*% (y[j,]-mu)  }  ll\_list[x] = -.5\*n\*k\*log(2\*pi) -.5\*n\*log(det(cov\_mat)) - .5\*sum(LL\_third\_comp)  print("Correlation value:")  print(iter\_rho)  print("Log-likelihood")  print(ll\_list[x])  cat("\n")  }  # Index of maximum log-likelihood value  mle\_index = which.max(ll\_list)  # Graphical illustration  plot(rho\_list, ll\_list, main="Grid search MLE",  type="l", xlab="ρ", ylab="Log-Likelihood")  abline(v=rho\_list[mle\_index], col="red", lwd=3, lty=2)  coeff\_hat = c(array(mu), rho\_list[mle\_index], ll\_list[mle\_index])  names(coeff\_hat) = c("μ1", "μ2", "μ3", "μ4", "μ5", "μ6", "μ7", "μ8", "ρ","MLE")  return(coeff\_hat)  }  coeff\_hat = mle\_est(mnData)  print(coeff\_hat) |

## Results:

### Q1

Text

Description automatically generated

### Q2 Chart Description automatically generatedChart, histogram Description automatically generated

### Q3

### Chart Description automatically generatedChart Description automatically generatedChart Description automatically generatedQ4

## Text Description automatically generated

# Bibliography

Aggarwal, C,C., 2018, Neural Networks and Deep Learning, Springer Nature, doi.org/10.1007/978-3-319-94463-0

Artificial Intelligence - Neural Networks, viewed 20 September 2020, https://www.tutorialspoint.com/artificial\_intelligence/artificial\_intelligence\_neural\_networks.html

Brain and Artificial Neural Networks: Differences and Similarities, ERA-IITK, viewed 20 September 2020, https://medium.com/@eraiitk/brain-and-artificial-neural-networks-differences-and-similarities1d337fe50168#:~:text=Biological%20neural%20networks%20have%20a%20different%20kind%20of%20learning%20method.&text=One%20major%20point%20of%20difference,but%20the%20brain%20may%20falter

Convolutional Neural Network, viewed 27 September 2020, https://ch.mathworks.com/discovery/convolutional-neural-network-matlab.html

Draelos, R., 2019, The history of convolutional neural networks, retrieved 3 November 2020, https://glassboxmedicine.com/2019/04/13/a-short-history-of-convolutional-neural-networks/

Michelucci, U., 2019, Advanced Applied Deep Learning: Convolutional Neural Networks and Object Detection, Apress

Venkatesan, R., Li, B., 2018, Convolutional Neural Networks in Visual Computing A Concise Guide, CRC Press

What is the loss function used for CNN, viewed 20 September 2020, https://stats.stackexchange.com/questions/432896/what-is-the-loss-function-used-for-cnn

Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

Zhenzhu Meng, Yating Hu, Christophe Ancey, 2020, Using a data driven approach to predict waves generated by gravity driven mass flows, Water, 2020, 12, 600

1. The intensity of grey image pixels are represented by a number 0-1 where white is 1 and black is 0. [↑](#footnote-ref-1)