Neuroscience & Computer Science
↓ enable BCI technology.
↓ Advancement using AI.
ML ↘ DL (DNN)

AI - related Visual, literal & motion applications

Visual AI → Computer vision
↓ understand & process visual data: (Recognize patterns & objects in visuals)

Libraries:

1) Open CV (Intel, C++, Python, Java) → fast + efficient
2) Tensor flow → DNN + NN, powerful + Advanced
3) Pytorch → flexible, slow
4) Keras → high level Neural Network + User friendly
5) Dlib → C++, ML, highly optimized, fast

Applications:

1) Object recognition & detection
2) Image classification
3) Facial recognition
4) scene understanding
5) Anomaly detection

Literal AI → ability to handle images, videos, audio and pdf to generate and process.

Libraries:

- Chairlit → Multiplatform, fast, persistant, reasan integrate

① openAI (Hugging face)
② LangChain
③ Mistral

Motion AI → process and detect motion

1) OpenNN → DL, Like brain, learn & make decision, C++
2) Pybrain → ML, educational
3) Theano → Computational AI

→ AI → Human Intelligence
↳ Umbrella of all
↳ reasoning, learn, decision

→ Machine learning → AI that

enables machine to learn on its own by analysing data.

① Mostly regression and classification
Regression → predict outcomes
classification → identify pattern & group data

② Supervised → Input + desired output in training dataset (Regression classification)
③ Unsupervised → No training data. Learn on its own (cluster or reduction)

Supervised

1) Linear regression
→ Find line that fits best the data. (Numeric value) (weighted sum of i/p)

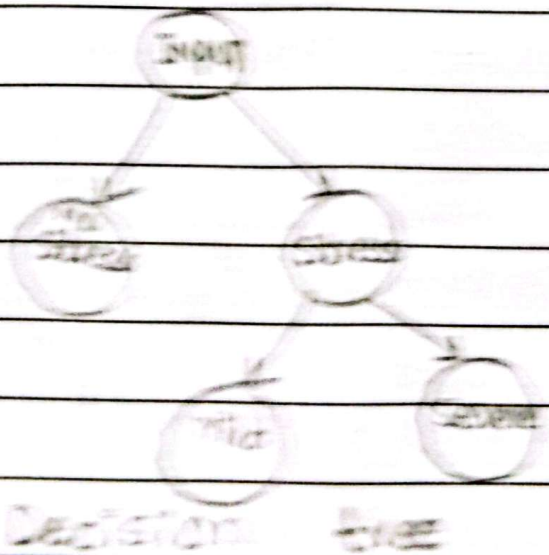
2) Logistic regression
→ Binary output

3) Decision Tree :
→ Complex relationship
Node → Decision & leaf → outcome

1) Regression: value assigned to leaf
2) classification: classification is random forest.

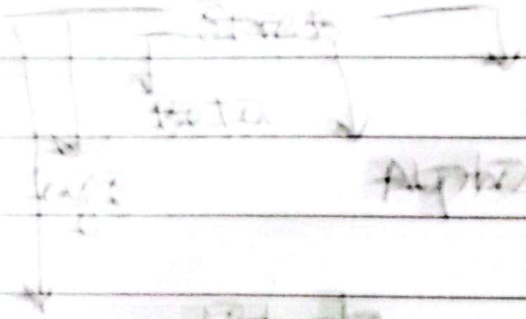
→ Random forest:

Output selected by many trees
work great with training data
but not flexible with classifying
new samples.
So, random forest used for
flexibility to new samples.
Use multiple data variables
from data set.



Decision tree

Random forest



→ support Vector machines

1) text classification
2) image classification
3) spam detection
4) handwriting identification
5) Anomaly detection

represented as point and a hyper-plane is decided by kernel used when non-linear (Gaussian kernel)

→ KNN (K-Nearest Neighbour)

Based on euclidean distance between set point and test point.

→ gradient boost

TERESOL

→ Unsupervised

(1) Clustering → grouping

(2) Association → Relationship

→ Partitioning Clustering

↓ non hierarchical group

Based on data points b/w groups

K-groups on K-pre-define groups.

Distance b/w data points of one cluster is minimum w.r.t to another cluster Centeroid.

K-Clustering

→ Density ...

high dense area in clusters

→ Re-inforcement Learning

control learning
learn from experience
unsupervised learning but feedback
to obtain reward from progress.
choice is correct, incorrect or
neutral.
Based on points and reward from
user.

→ Neural Networks
create software that can learn
decision like humans.
Composed of interconnected processing
nodes or neurons, that can learn to
recognize pattern and decision making.
→ extract identifying features from data,
Lacking pre-programmed understanding
Base of Deep learning.

Neural network → Deep learning
machine learning

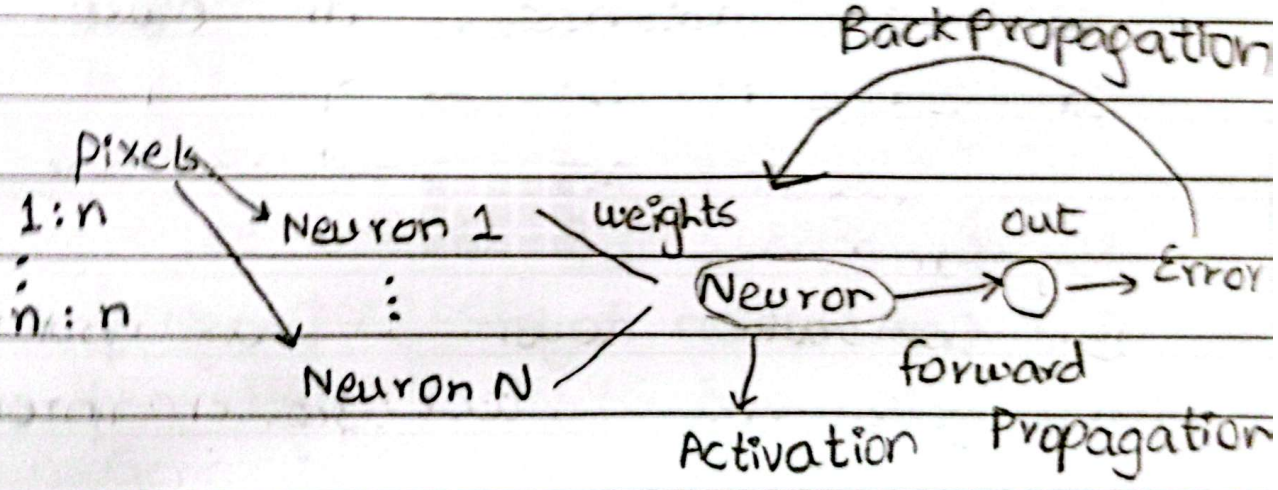
Date.....
Mon | Tue | Wed | Thu | Fri | Sat | Sun

Layers of Neuron
↓
Core processing Units

Input Layer → Hidden Layer → output Layer
↓
Performs Computations needed by Layers

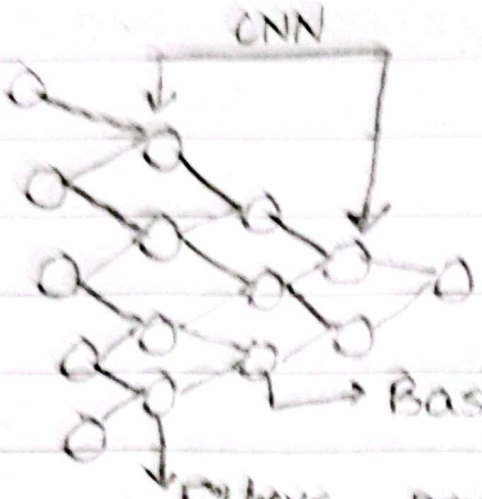
Input data Composed of pixels. Each pixel is passed to the input layer as input to each neuron.
Neurons are Connected to neurons of next layers through channels. Each channel has some assigned weights, neurons are multiplied to corresponding weights, and their sum is input to next layer (Hidden Layer). Each such hidden

Layer is assigned a biased value that is added to sum. This is then passed to an activating function, this tells a particular neuron is activated or not; The higrest value from forward propagation. Then error determined and correction is made and passed using back propagation.



CNN → Convolutional neural Networks
↳ used object recognition & Video
FNN → feed forward for basic image recognition and pattern recognition
RNN → speech & NLP

Convolutional Neural Network
↓ specialized in pattern recognition

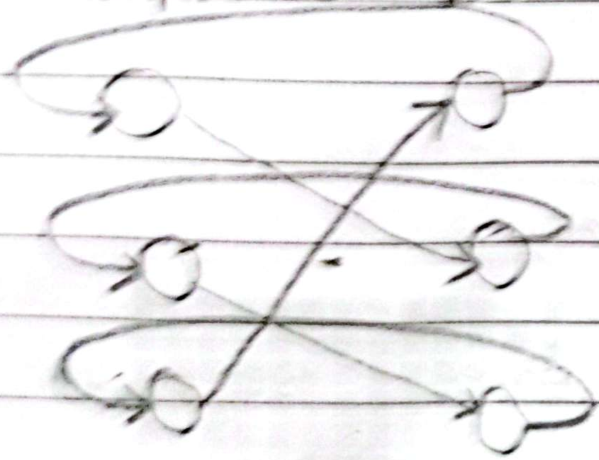


Filters perform pattern recognition, obtain numeric score and then check resemblenes, how close array is.

Key Components of CNN:

① Convolution layer → Image pixel analysis and detect pattern provide score.
② padding → Extra pixels
③ Polling → Reduces dimensions, desample
④ fully connected layers: classify features

2) Recurrent neural networks (RNN):
No input; the output feedback to input. (sequential data)



3) GAN (Generative Adversarial Networks):
generative modelling
It discover and learn pattern automatically in input data and generate output (new examples) from dataset.
Two Neural network
1) Generator (upper layer)
2) Discriminator (Lower Layer)
Generator generate fake pic and discriminator detects and generator corrects itself using error function by discriminator.

LSTM (Long short term memory):

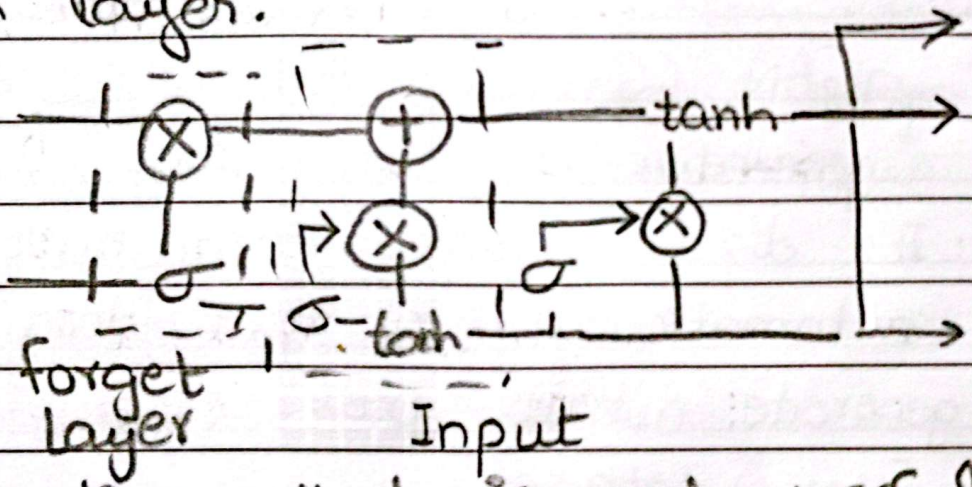
Type of RNN.

RNN has a gated unit.

Four Layers that interact with each other to produce output of cell with a cell state.

RNN has one Layer of tanh.

There are three sigmoid & one tanh layer.



Information that is not useful is removed by forget gate.

→ Addition of useful information is done by input gate.

→ Deep learning subset of machine learning

Date

Mon | Tue | Wed | Thu | Fri | Sat | Sun

based on neural networks with multiple layers of abstraction.

AI → technique to enable machine to mimic human behaviour.

ML → Subset of AI that used statistical methods to learn and experience. Use algorithms. (Manual data)

DL → Subset of ML, data Connection by multiple layers of abs called neural networks. (Automatic data) Next evolution of ML.

Computer Vision → Image & Video
Natural language processing → Textual & spoken human language
Robotics → mechanical humans
AI, ML, DL

AI → operate on data set it is

Innovatively Creative

www.teresol.com

trained.

Strong AI → Everything or task on their own without supervision.

Specialized AI :→ specific task.

Pandas:

Data manipulation and analysis Library for python.

For numerical tables.

Numpy:

matrix or vector operations

Statistical Computing

1) Pandas ✓
2) Seaborn

Signal processing

Scipy

Pywavelets

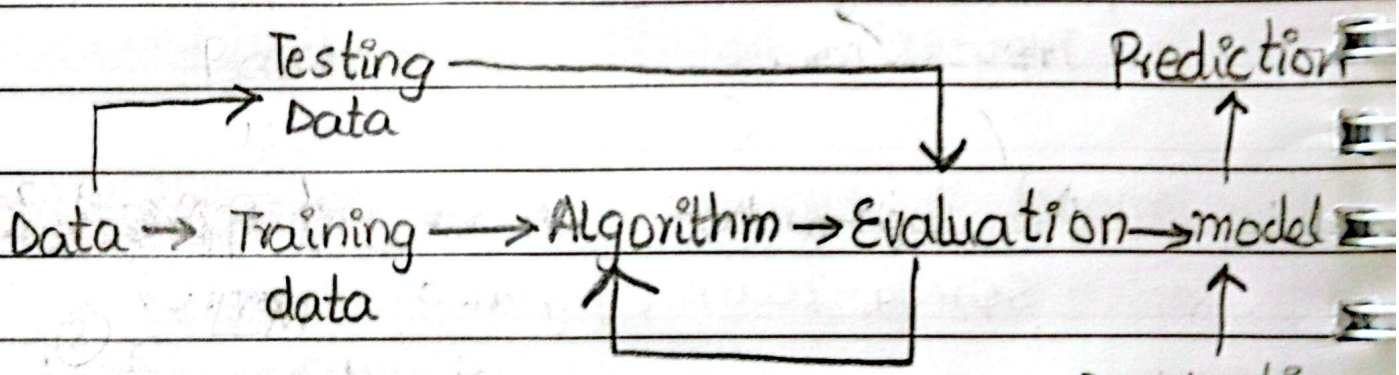
Image processing

scikit - image

light GBM, catboost, yellowbrick, ELi5

GUI
Qt
flask
cherrypy
Panel
streamLit

→ Workflow of machine learning



Three basic stages:

① Gathering data
② Data preprocessing
③ Algorithm selection
④ Training and testing model

5) Evaluation

→ Machine learning model is nothing but a piece of code, which is trained using data to make it smart.

→ Gathering data: Depend on project, wish to make.
Datasets: Kaggle, UCI machine Learning Repository
Kaggle provides Machine Learing Algorithm, a repository.

→ Data preprocessing or data preparation is important in building machine learning models more accurately.
ML has 80/20 rule, 80% time utilization for pre-processing & 20% for analysis.

It is basically cleaning the

raw data, (The data collected from real world), Converted to clean data set.

why?

Real world data is messy:

1) Missing data
Not Continuosly created or technical issue;

2) Noisy data
Outliners, human errors or technical problem of device.

3) Inconsistent data
Duplicate data

Data: Numerical, categorical, textual, Pixel;

→ performing data preprocessing:

1) Conversion of data
ML only handle numeric data hence other are Converted to numeric data.

2) Ignoring missing data

3) Filling missing data:
Manually filling using mean, median or highest frequency value.

4) ML filling dataset by predictions.

5) Outliers detection:
Some error data present in dataset that deviates drastically from other observations in dataset.
human weight = 800 Kg, mistyping of extra 0.

6) Data standardization and Normalization
Improves data performance and ensure features of different scales or units or units are treated equitably.
Normalization is process of scaling feature values to a specific range between 0 and 1.

$$X_{norm} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

Standardization involves transforming feature values to have mean 0 and standard deviation of 1.

$$z = \frac{x_i - \mu}{\sigma}$$

Normalization used in distance based algorithms, K-Means, KNN & Neural networks.

Standardization used when data is supposed to be centred at mean. Like Linear regression, SVM

Implementation in python.

```
from sklearn.preprocessing import
MinMaxScaler, standardScaler
  # Normalization
  scaler = MinMaxScaler()
  X_norm = Scaler.fit_transform(X)
```

standardization

```
scaler = StandardScaler()
X-standardized = Scaler.fit_transform(X)
```

Epoch or pass or iteration or batch

→ Epoch is the one complete pass of the entire training dataset.
Epoch makes model to make prediction on its current states, compares these predictions to the actual outcomes, calculates error function and then updates its internal parameter in order to reduce error.

→ Batch is the subset of training dataset.
It is 16 or n(16).

→ Iterations refers to one update of model parameters, which occurs after processing one batch.

The number of iterations needed to complete one epoch depends on total size of training dataset and batch size.

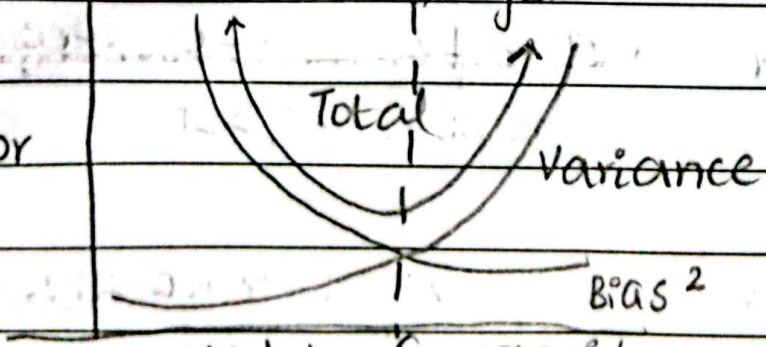
→ Underfitting
Underlying trends cannot be captured
high biase + Low Variance

→ Overfitting
Fits data too well
Low bias + high Variance

→ Bias is difference between average prediction of model and Correct value

→ Variance Shows Spread of value.

→ Low bias, Low Variance is good model, generalized.



→ (4) Training & Testing:

3 section

→ Training data

Train the classifier

→ Validation data

Tune the parameters

→ Test data

Test performance of your classifier over unseen data.

By Confusion matrix, a Confusion matrix can have 4 parameters,

① True positive (predict true output ✓)

② True Negatives (predict Fake output ✓)

③ False positives (predict True output Fal

④ False Negatives (predict False output Tru

$$\text{Accuracy} = \frac{(\text{True pos} + \text{True Neg})}{\text{Total number}}$$

→ LLM (Large Language models):

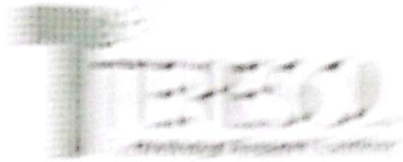
understand and generate human like text.

→ Data analysis
Inspecting, cleansing, transforming, modelling data with the goal of discovering useful information, conclusion and decision making.

→ Data Visualization
① Graphical or pictorial view

→ Libraries:
① Numpy
② stats model
③ matplotlib
④ Scipy
⑤ Scikit-learn → visualization
⑥ pandas → Data manipulation
⑦ Seaborn

→ Pandas



→ Passing dictionary or tuple to pandas Series

```
dic = {"name":['python','C','C++'],
"por": [12, 13, 15], "rank":[1,4,3]}

Var1 = pd.Series(dic).
```

→ Duplicating over indexes

```
var = Py.Series(12, index=[12,13,15])
```

→ Addition of arrays:

$$\begin{bmatrix}12\\13\end{bmatrix} + \begin{bmatrix}12\end{bmatrix} = \begin{bmatrix}24\\NAN\end{bmatrix}$$

It work even with missing data but Numpy generate error.

→ DataFrame (2D - structures)

Pd.DataFrame

→ Extract Column

Date:

pd.DataFrame (d, columns=[ ])

→ setting data
var[column][rows]

→ Nested list
lst = [[ ], [ ], [ ]]

→ s1 = {"x": pd.Series([ ]),
"y": pd.Series([ ])}

pd.DataFrame (s1)

→ var = pd.DataFrame( )
var["a"] = var["b"]

→ var["x"] = var["y"] <= value

→ Insertion
var.insert(At index, "name", data)
column

→ Limited data

var [A] = var [column to be copied] [:index to stop+1]

→ Deletion:

var.pop ("column to be deleted")

or

→ Creating csv file

var.to_csv ("file.csv")

→ For removing or including index:

var.to_csv ("file.csv", index=False)

→ changing header

var.to_csv ("file.csv", index=False, header = [values]);

→ Reading csv into python

p = pd.read_csv("File path with double //")

→ Reading rows

csv_a = pd.read_csv("filepath", nrows=

number of rows from start);

→ Getting column
use, usecols = ["It "] after file path in read, It can be Column number or name

→ Removing rows
use skiprows = ["row to skip"]

→ Removing index column
use index_col = "column name"

→ Row to header
use, header = "row number"

→ Add header
names = ["Col"]

→ Remove header
header = None

Then adding header
prefix = " name" x

→ change data type of a Specific row
dtype = {"Column Name": "Datatype"},

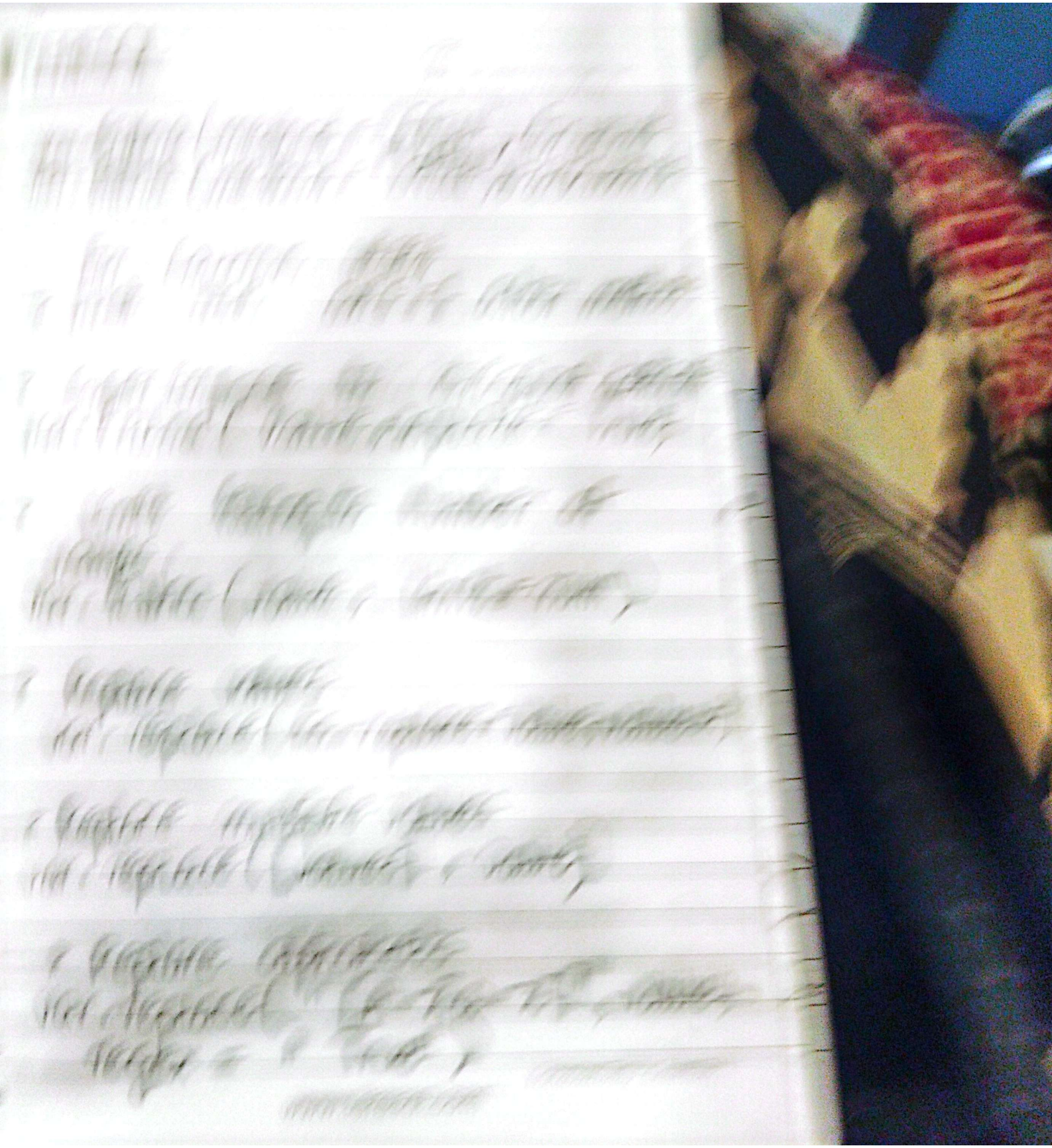
→ obtaining range of indexes
csv-1.index

→ Obtaining Column names
csv_1. Columns

→ obtaining information
csv-1.describe

→ Starting values
csv-1.head(x)
If x not given, then 5 by default.

→ End values



'akima', 'pchip', 'akima'}

.interpolate(method=" ", axis=1)
1 = 4, 1 — optional

→ Filling limited NaN
.interpolate(limit=2)

→ Filling limited NaN, in direction
.interpolate(limit_direction=" ", limit=)
direction can be forward, backward, both

→ Filling in specific area
.interpolate(limit_area=" ")
area can be outside, inside

→ Combining two dataframes
pd.merge(Frame1, Frame2)

→ with same column 'A'.

pd. merge (Frame1, Frame2, on = "A")

→ Combining same data Frames with one different values
default is inner

pd. merge(var1, var2, how = " ")

how = inner, left, right, outer

→ Provide indicator
use indicator = True on previous Command after how.

→ Merge two same dataframes with Completely different value
pd. merge (Var1, Var2, left_index = True, right_index = True)
To change heading, use Suffixes = (" 1st " , " 2nd) in above Command.

```
var1 = var.groupby("column")
```

For viewing :

use for loop

```
for x,y in var1:
    print(x)
    print(y)
```

→ Specific data accessing

```
var1.get_group("data of required")
```

→ mean, mode, median, max, min

```
var1.min()
var1.mean()
Var1.max()
```

→ Convert Grouped data into list

```
l1 = list(var1)
```

→ Grouping data set;

```
var1.join(var2)
```

→ join by specific side
a.join(b, how = " ")
how = left, right, inner, outer
inner → intersection
outer → union

→ join elements by same column name
a.join(b, lsuffix = "Name", rsuffix = "Name")
used when same column

→ join elements of same column
a.append(b)

→ show sequence, ignore index
a.append(b, ignore_index = True)

→ Remove duplicates
drop-duplicates()

→ Form a vertical table
Pd.melt(c)

Date
Mon | Tue | Wed | Thu | Fri | Sat | Sun

→ Indexing or Id using column
pd.melt(c, id_vars = "group")

→ Change Variable name
pd.melt(c, id_vars = " ", var_name = " ")
use value_name for value name change

→ Data in arranged order
var.pivot(index = "col", columns = " ")

→ To access particular data
use value = "group" after columns

→ Arithematic operations in pivot
c.pivot_table(index = " ", columns = " ", aggfunc = " ")

aggfunc = mean, Sum,

→ Mean
use margin = True

→ NUMPY

import numpy as nd

→ Creating an array

a = np.array([list])

→ Dimension & datatype

print(a.shape) → size
Print(a.dtype) → type
print(a.ndim) → dimension
print(a.size) → No of element
Print(a.itemsize) → Number of bits

→ Specific element print

print(a[i])

→ Assignment

a[i] = value

→ Product

b = value * a

TENSOR

a.T → Transpose

→ Inverse:
np.linalg.inv(a)

→ Determinant
np.linalg.det(a)

→ Diagonal
c = np.diag(a)

→ Generating diagonal matrix
np.diag(c)

→ Generating n dimension array
np.array([[1,2],[3,4]])

→ Boolean index
booli = (Matrix / Array) < value
a = a[booli] → return elements that were true

→ Matrix Multiplication
np.matmul(a,b)

→ Conditional matrixing
np.where(condition, for true, for false)
Let,
np.where(a>2, a, -1)

→ Fancy indicies
b = [ ]
a[b]

→ Even numbers in array indicies
np.argwhere(a%2==0).flatten()
↓ Single List

→ Create array
a = np.arange(start, stop+1)

→ Reshaping
b = a.reshape((rows, column))

→ New axis (shaping in a certain manner)
b = a[np.newaxis, :]

Numpy

np methods

Concatenation
np.concatenate((a,b))
along column use
np.concatenate((a,b), axis=" ")

axis = 0 → rows
axis = 1 → column
axis = None → one-D array

Horizontal appending (One-D resulting). (For 1-D)
np.hstack((a,b))

Vertical appending (Results in 2-D) (For 1-D)
np.vstack((a,b))

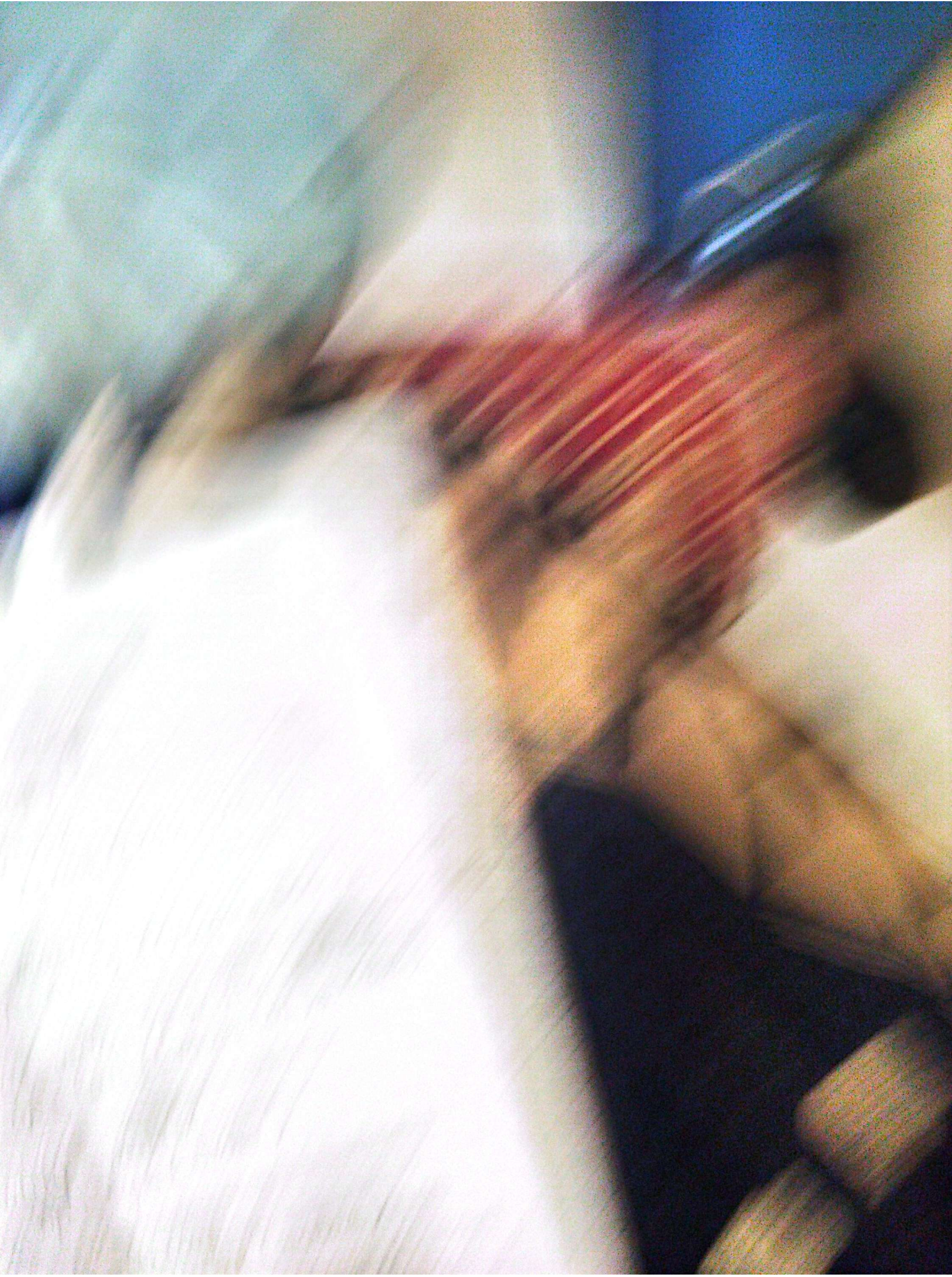
Broadcasting
Adding each element to its equivalent in other matrix

np.std(a, axis = " ")
np.min(a) ; axis same
np.max(a)

Changing datatype

np.array([float values],
dtype = np.int64)

↳ can be 32



→ Random number generation

a = np.random.random((row, col))

→ Gaussian / normal distributed

0 mean $\sigma^2 = 1$

a = np.random.randn(row, col)

a = np.random.randn(numbers)

→ Rand int

np.random.randint(start, stop, size = (row, col))

np.random.randint(end, size=(row, col))

→ Random choice

np.random.choice(end, size = num)

np.random.choice([list], size = N)

→ Eigen values

eigenvalue, eigenvectors = np.linalg.eig(a)

→ Title

```
plt.xlabel(" ")
plt.ylabel(" ")
plt.title(" ")
```

→ Font size

```
plt.xlabel(" ", fontsize=" ")
```

→ Bar width

```
plt.bar(x, y, width=" ")
```

→ changing color

```
plt.bar(x, y, color=" ")
```

y = yellow

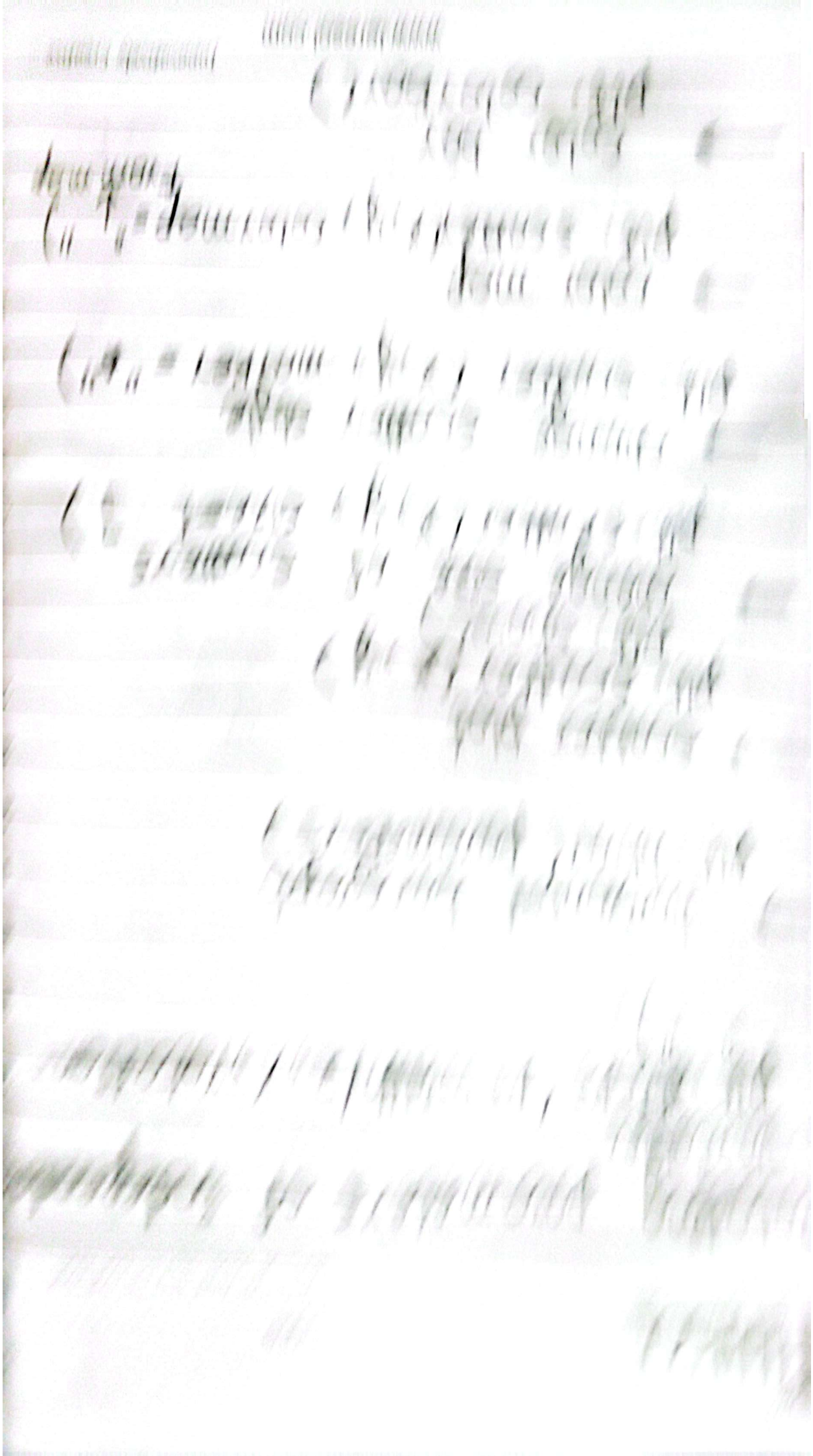
→ color of every element

```
c = ["y", "b", "m", "g"]
plt.bar(x, y, color = c)
```

→ Alignment

```
plt.bar(x, y, align="edge")
```

align = edge, centre



→ Histogram

```
plt.hist(list)
plt.show()
```

→ Set configuration for bar

```
L = [10, 20, 30, 40, 50, 60]
plt.hist(list, bin=L)
```

→ Graph range

```
plt.hist(List, "auto", (0, 100))
```

"auto" ← Bin

(0, 100) ← x-axis range

→ Reversing frequency graph

```
plt.hist(List, cumulative = -1)
```

→ Scaling y-axis

```
plt.hist(List, bottom = 20)
```

bottom = 20 ← start y-axis

→ Alignment of histogram

```
plt.hist(List, align = " ")
```

left, right, mid

→ Types of histo-gram
plt.hist(no, histtype = "step")
step filled, barstacked, bar, step

→ orientation
plt.hist (no, orientation = "horizontal")
vertical, horizontal

→ width of bar
plt.hist(no, rwidth = " ")

→ Logarithmic frequencies
plt.hist (no, log = True)

→ Axis Color & line
plt.axvline(pos, color = " ", label = " ");

→ Grid view
plt.grid()

Pie Chart

```
plt.pie(x)
plt.show()
```

→ Add labels

```
plt.pie(x, labels = y)
```

x & y = List

→ Extrude a parameter

```
ex = [0.4, 0.0, 0.0, ...]
```

↓ extruded part to that extend

```
plt.pie(x, explode = ex)
```

→ Showing %.age in pi-chart

```
plt.pie(x, autopct = "%.0.1f%%")
```

↓ No of decimal

→ Shadow

```
plt.pie(x, shadow = True)
```

→ Radius of pi-chart

```
plt.pie(x, radius = value)
```

Date

→ Label distance

plt.pie(x, labeldistance = val)

→ Rotate pi-chart and change starting angle

plt.pie(x, startangle = val)

→ Text Size of labels

plt.pie(x, textprops = {"fontsize" : val})

→ Rotation direction of pie chart

plt.pie(x, counterclock = False)

→ Linewidth of pie charts

plt.pie(x, wedgeprops = {"linewidth" : val})

→ Rotate labels

plt.pie(x, rotatelabels = True)

→ Circle

```
Cr = plt.Circle (xy=(0,0), radius=1,
facecolor = "W")          ↓centre
plt.gca().add_artist (Cr)
```

→ Stem plot

```
plt.stem (x,y)
```

→ line style

```
plt.stem (x,y, linefmt=":")
```

→ Market color and stem

```
plt.stem (x,y, markerfmt="r+")
```

→ Shifting y - axis

```
plt.stem (x,y, bottom = 1)
```

→ Changing base color

```
plt.stem (x,y, basefmt = "g")
```

→ Color change each line

```
plt.stem (x,y, use_line_collection
    = False)
```

→ Box plot

plt.boxplot(x)

→ changing width

plt.boxplot(x, widths=0.2)

→ Color the box

plt.boxplot(x, patch_artist=True)

→ Show mean

plt.boxplot(x, showmeans=True)

→ Join whisk to outlier

plt.boxplot(x, whis=3.5)

→ change outlier color & style

plt.boxplot(x, sym="g+")

→ change box property

plt.boxplot(x, boxprops={"color":"y"}, capprops={"color":"y"})

— for whiskey color:
use: whiskeyprops = {"color": "y"}
in previous.

→ Two box plot (Avoid overlapping)
y = [ ]
x1 = [ ]
x = [x, x1]
plt.boxplot(y, labels = [" ", " "])

→ Stack plot
plt.stackplot(x, area)
List 1 → List 2

→ Multiple stack plot
plt.stackplot(x, area, area1, area2)

→ Base Change
plt.stackplot(x, area, ..., baseline = "Sym")
baseline = sym, zero

→ Step plot

plt.step(x,y)

→ Modify

plt.step(x,y, color = "r", marker = "o", ms = 10, mfc = "g")

ms ↓ size
mfc ↓ color
marker ↓

→ Linear plot

plt.plot(x,y)

→ Color fill

plt.fill_between(x=val, y=val)

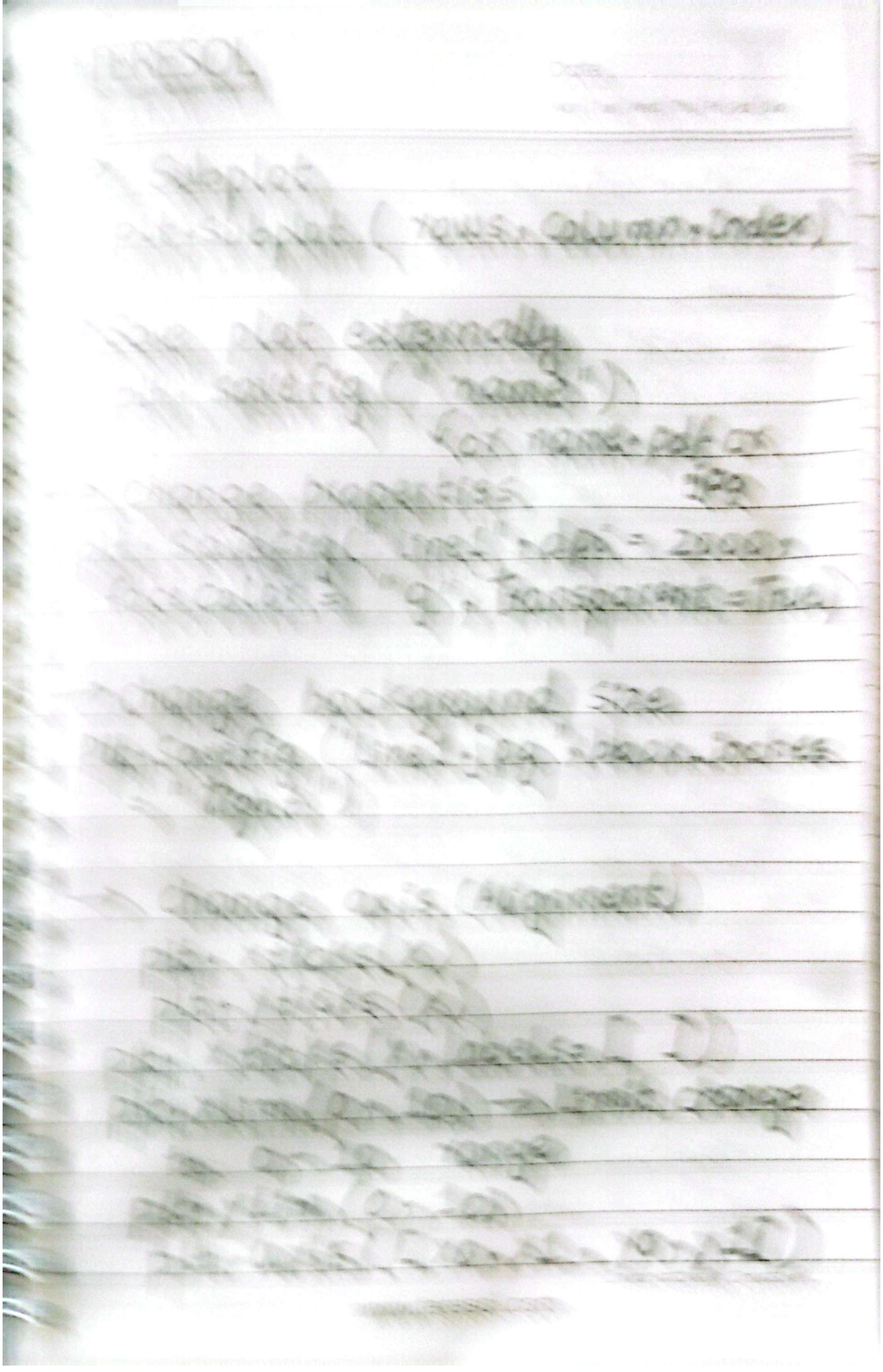
Like,

plt.fill_between(x=(2,4), y1=2, y2=4)

→ Boundary

use numpy...

plt.fill_between(x,y, color = "g", where = (x>=2) & (x<=4), alpha = 0.5)



```
plt.imshow(img)
plt.show()
```

→ Sine, Cos

```
np.sin(x)
np.cos(x)
```

→ fft

```
f = np.fft.fft(x)
```

→ real and imag

```
np.real(f)
np.imag(f)
```

→ fft shift

```
np.fft.fftshift(np.abs(f))
np.fft.fftshift(np.angle(f))
```

→ Understanding the Learning Curves

① Look up model training loss

High training loss?

High sign of struggle

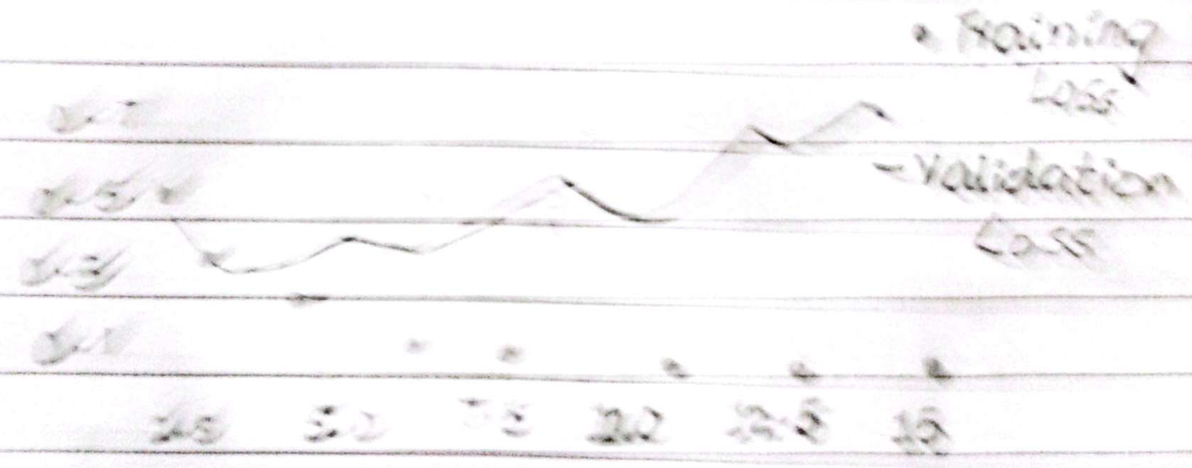
network is underfitting, not learning effectively.

3. Increase learning

Increase model capacity, allowing it to learn more complex pattern

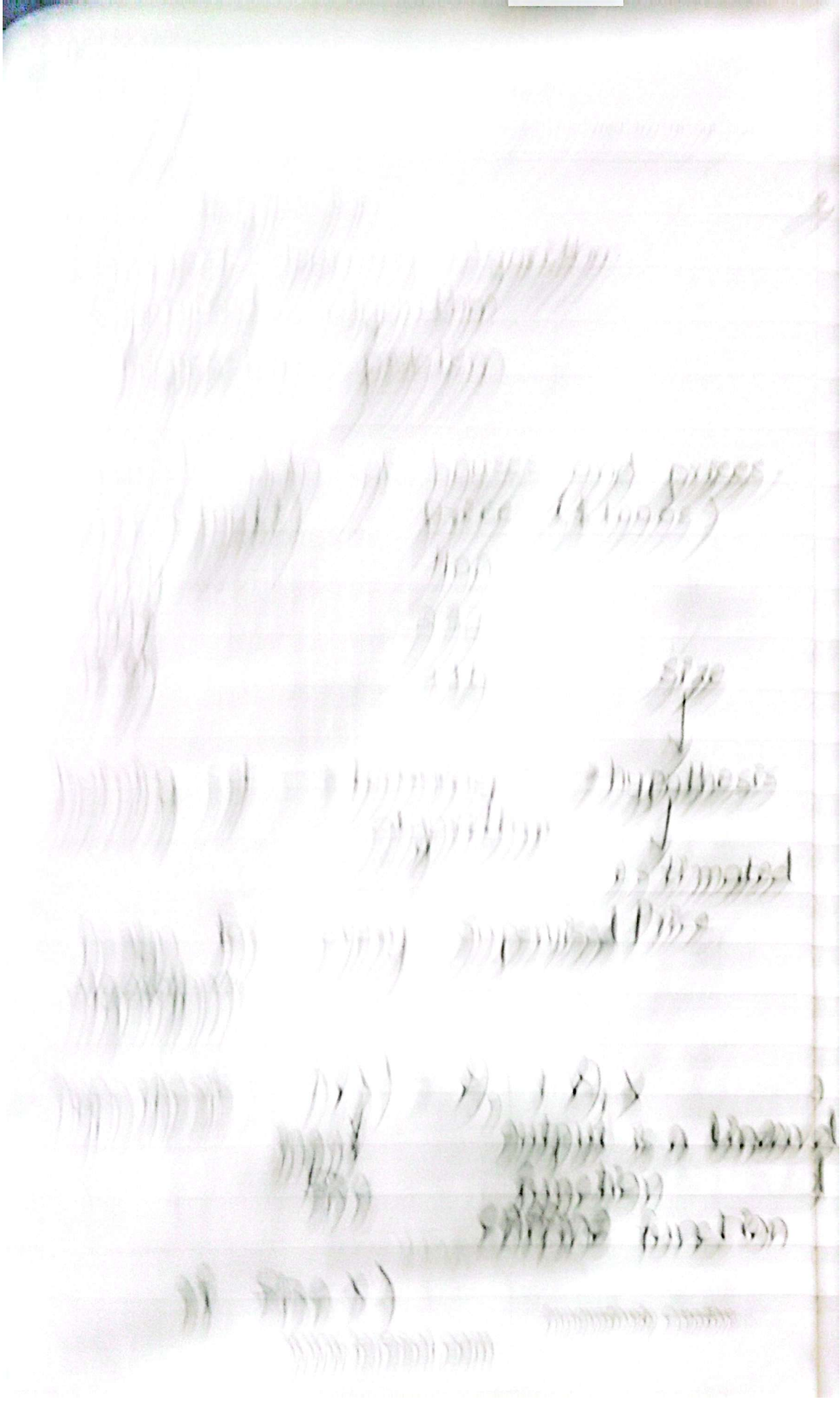
4. Validation

Evaluate against validation set.
observe validation loss.



To combat overfitting, we introduce regularization.

Both training and validation Loss go down.



Bedroom	Price (1000)
3	400
2	232
3	315
2	178

are three two inputs

x_1 = size

x_2 = no. of bedrooms

$= \theta_0 + \theta_1 x_1 + \theta_2 x_2$

$= \theta^T x$

Linear regression is also known as ordinary least squares.

Minimize, the square difference between hypothesis outputs,

$(h_\theta(x) - y)^2$

↳ Correct price

For m examples:

$$\frac{1}{2}\sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 = J(\theta)$$

$J(\theta)$ ← Costfunction

error

- To minimize Cost function, we need gradient descent.

Start with some θ (say $\theta = \vec{0}$)

Keep changing θ to reduce $J(\theta)$

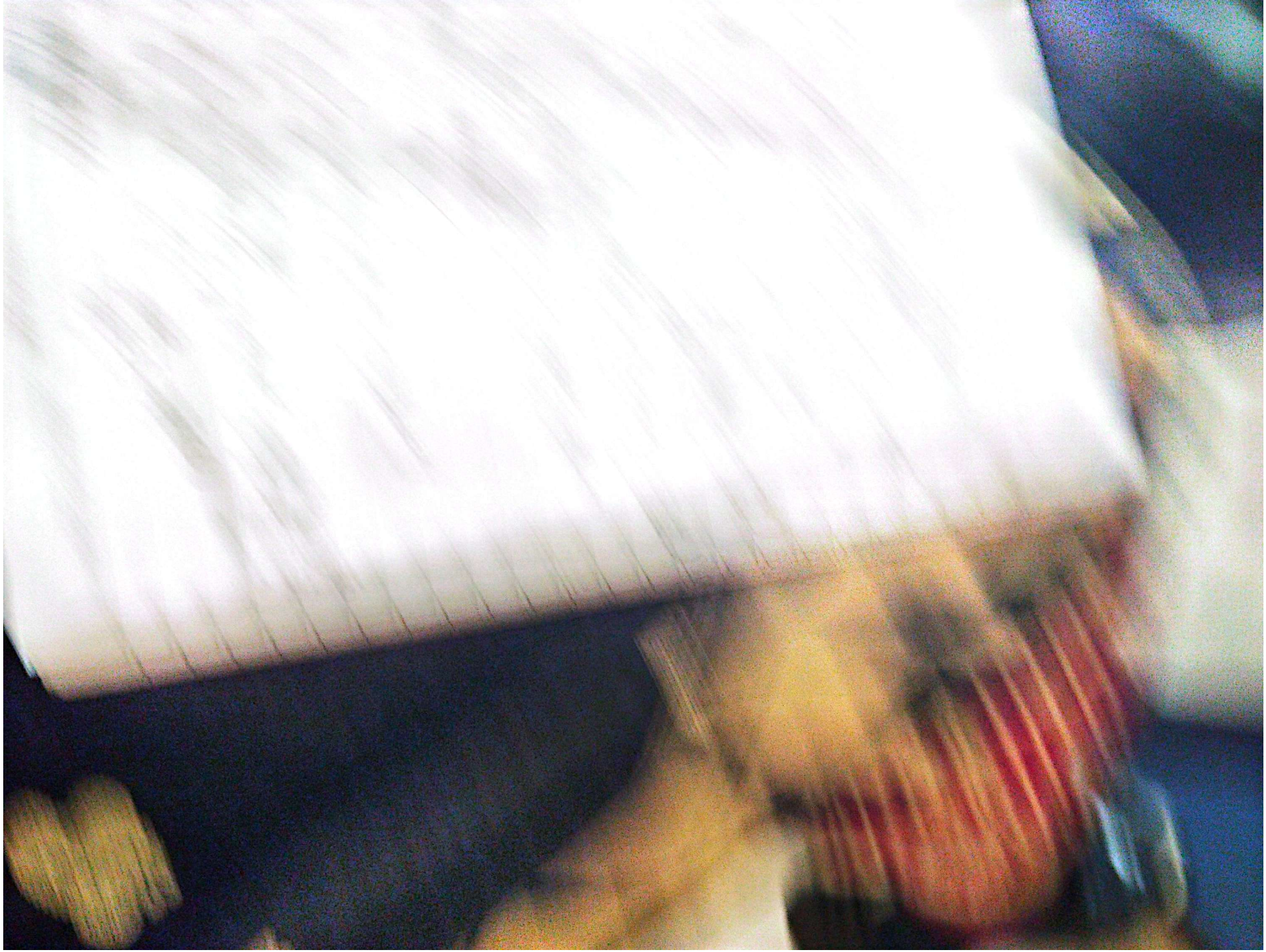
Modify θ:

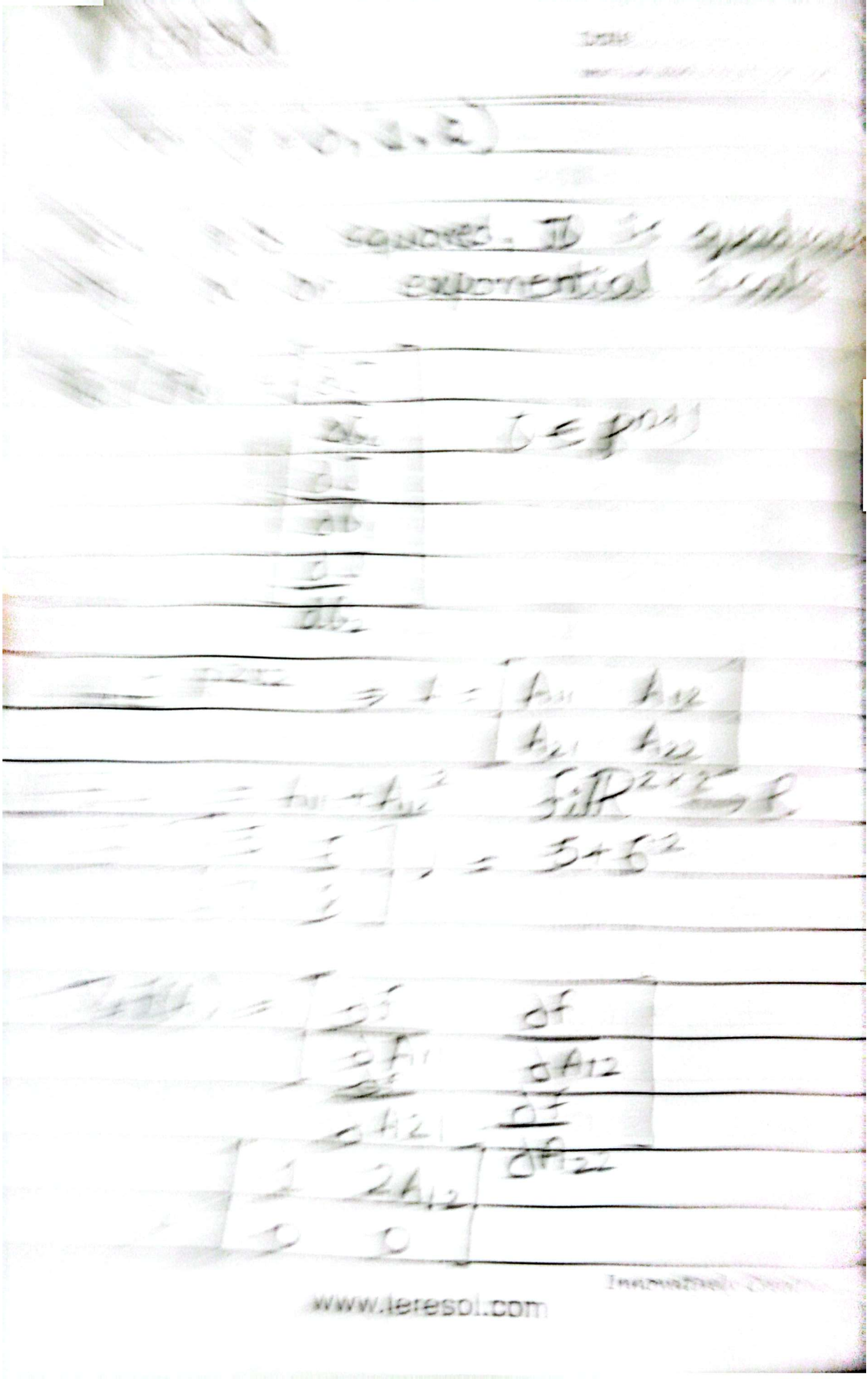
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

α ← Learning rate

$j = 0, 1, 2$

Derivative defines the direction





sum of residuals must be less than others.

→ Hypothesis / predictions
thetha . Transpose * X
↳ data
↗ Mean square error features

→ Compute Cost (MSE)
$\frac{1}{2} m * \text{sum(errors)}^2$ ✓

error = prediction − actual

→ Learning parameters:
features + 1
Number of rows = No of feature
Columns = No of elements.

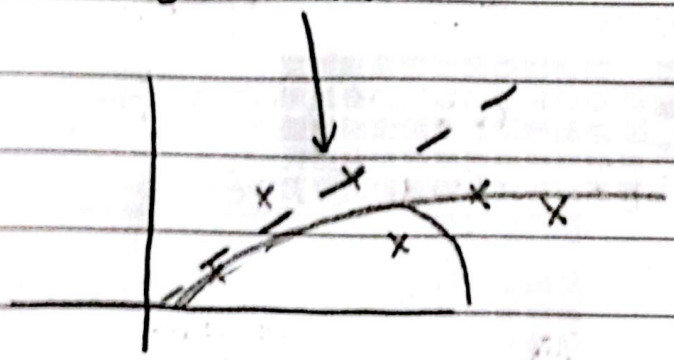
Obtain data
data ["col"] . values → np values
↳ Matrix / Pd List

We need two rows for training examples.

X = np.c_[np.ones(len(x)), X]
X.isnull().sum()

→ Logistic regression
↓ Non Linear functions
Newton's Method
Modification of Linear regression

→ $\theta_0 + \theta_1 x$



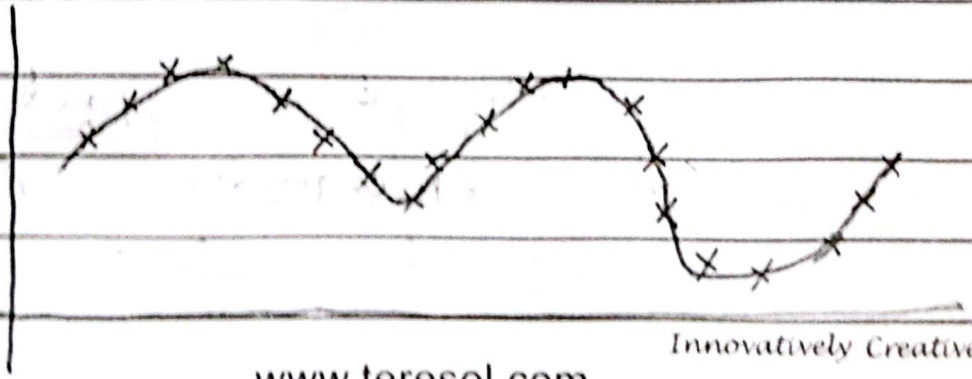
or $\theta_0 + \theta_1 x + \theta_2 x^2$

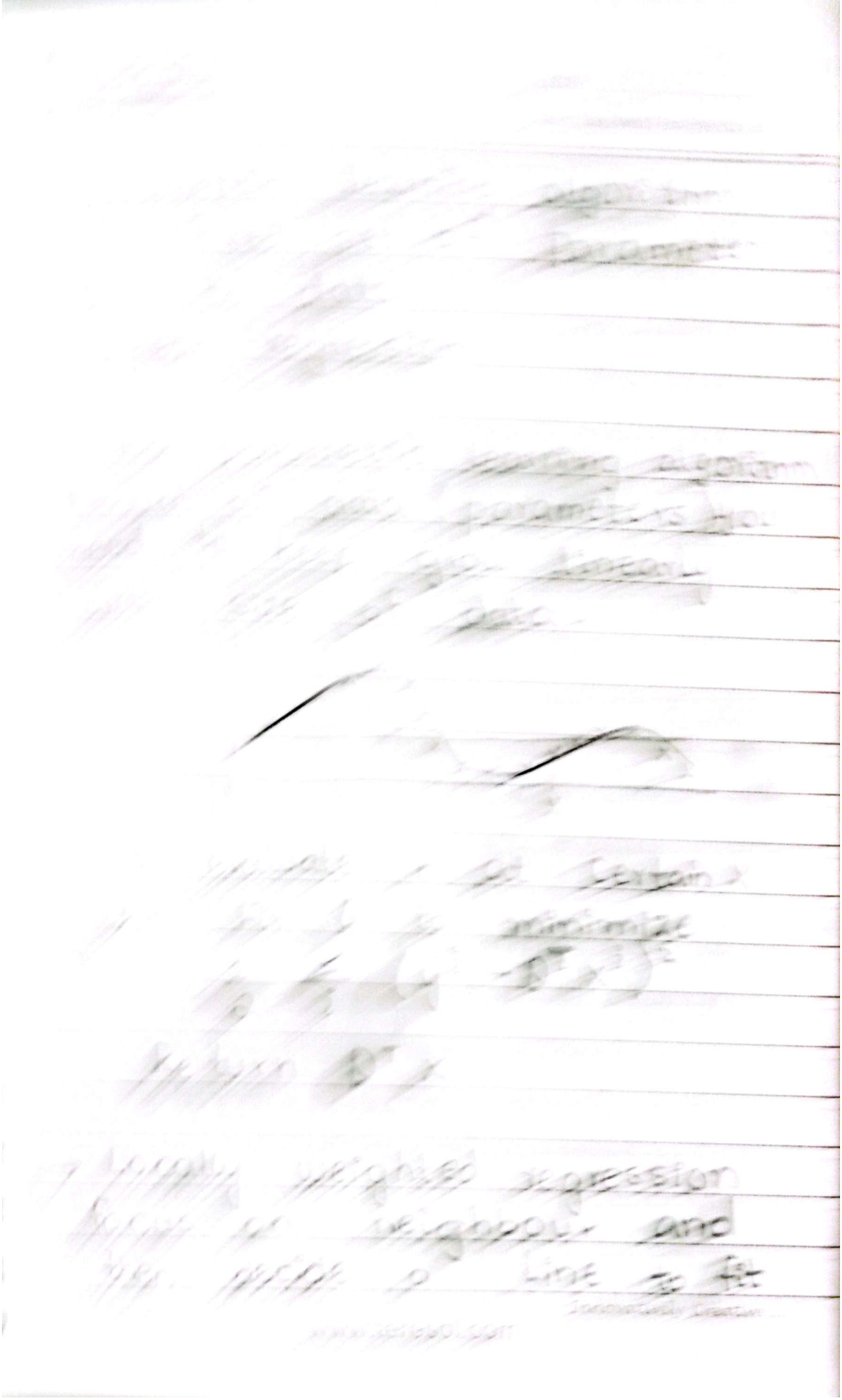
$\theta_0 + \theta_1 x + \theta_2 \sqrt{x}$

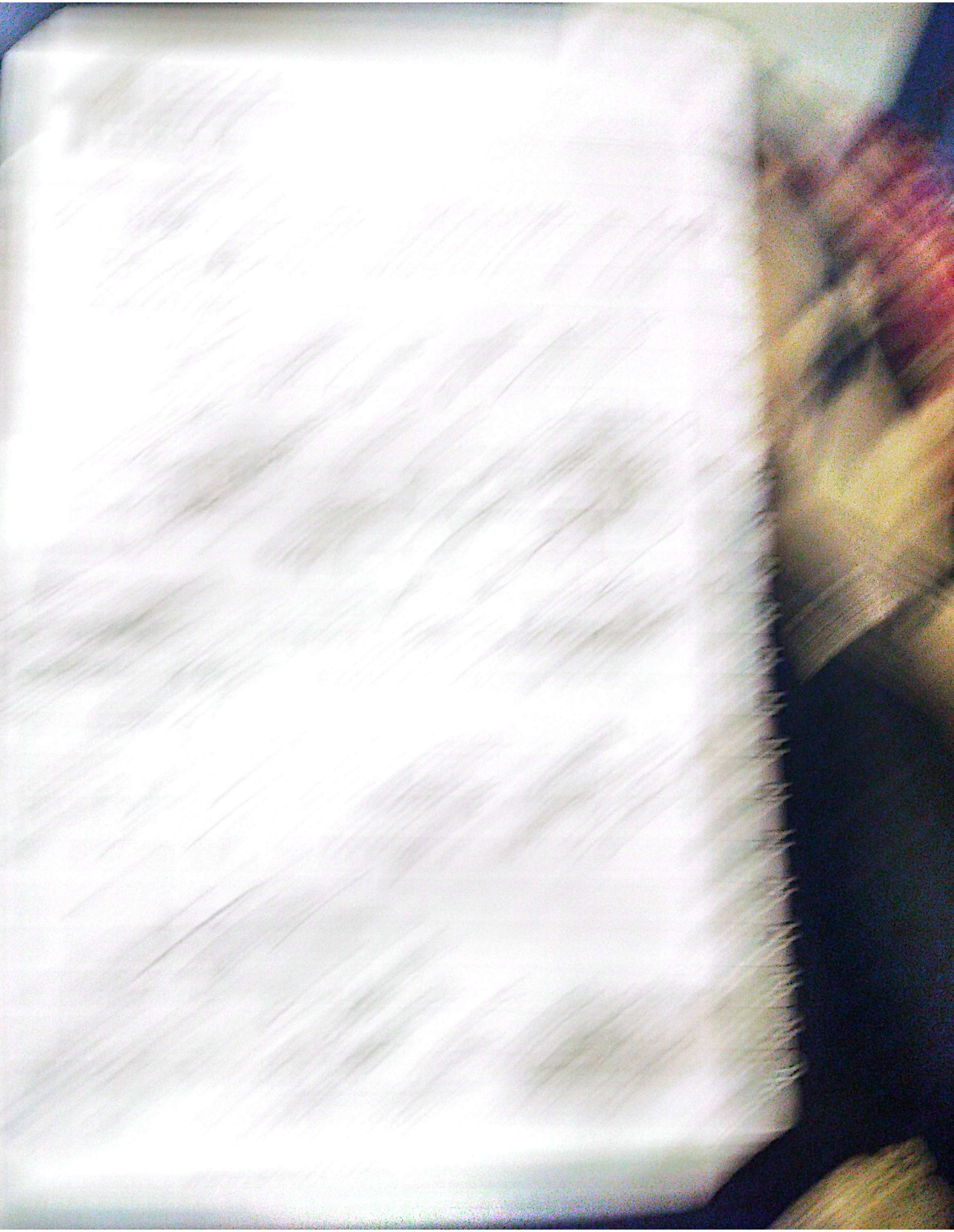
$\theta_0 + \theta_1 x + \theta_2 \sqrt{x} + \theta_3 \log(x)$

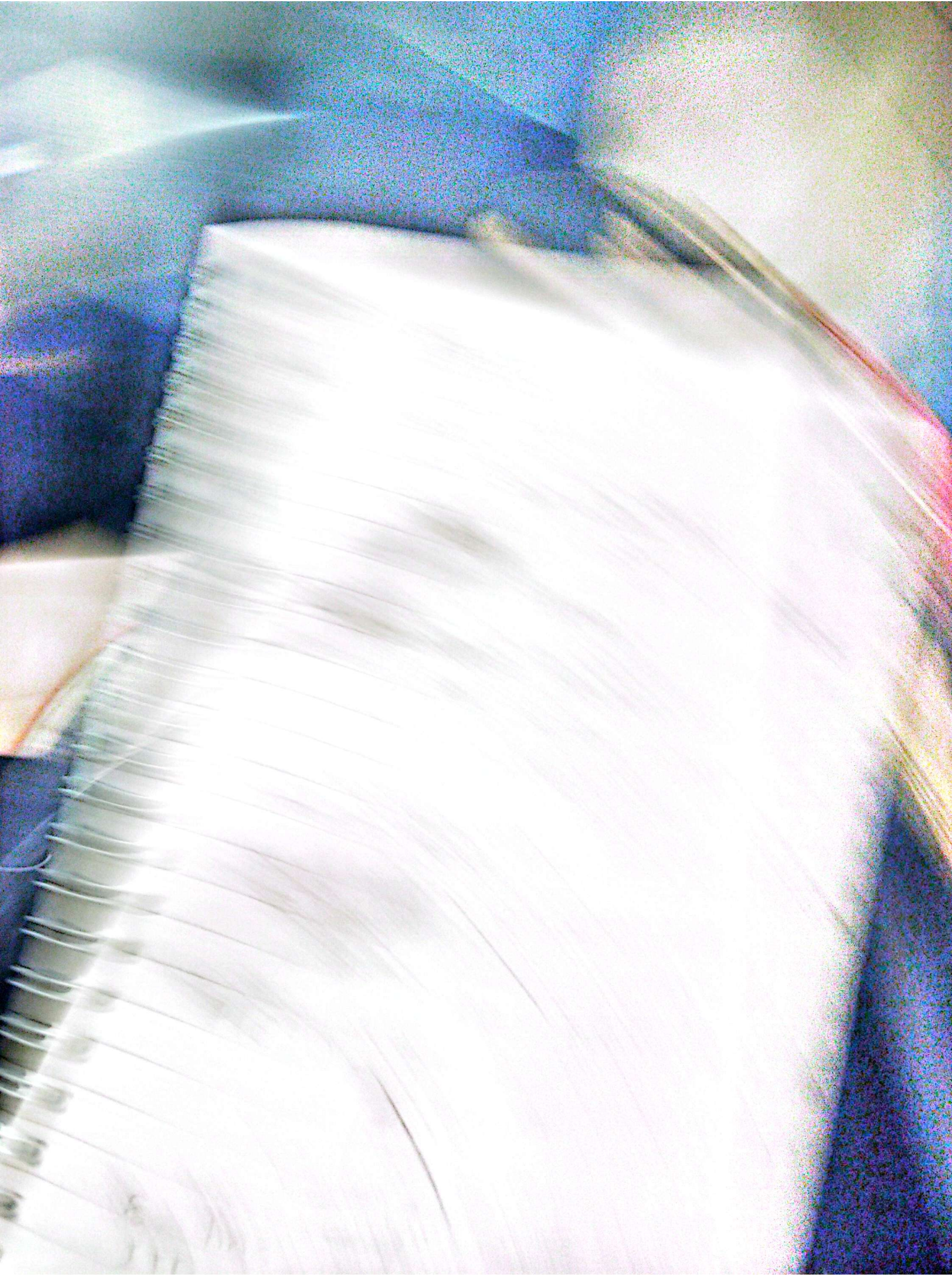
Logistic regression automatically select type of line required.

or locally weighted Linear regression









(binary classification)

Linear regression is not good for classification problem.

Logistic regression

$0 \le h_\theta(x) \le 1$

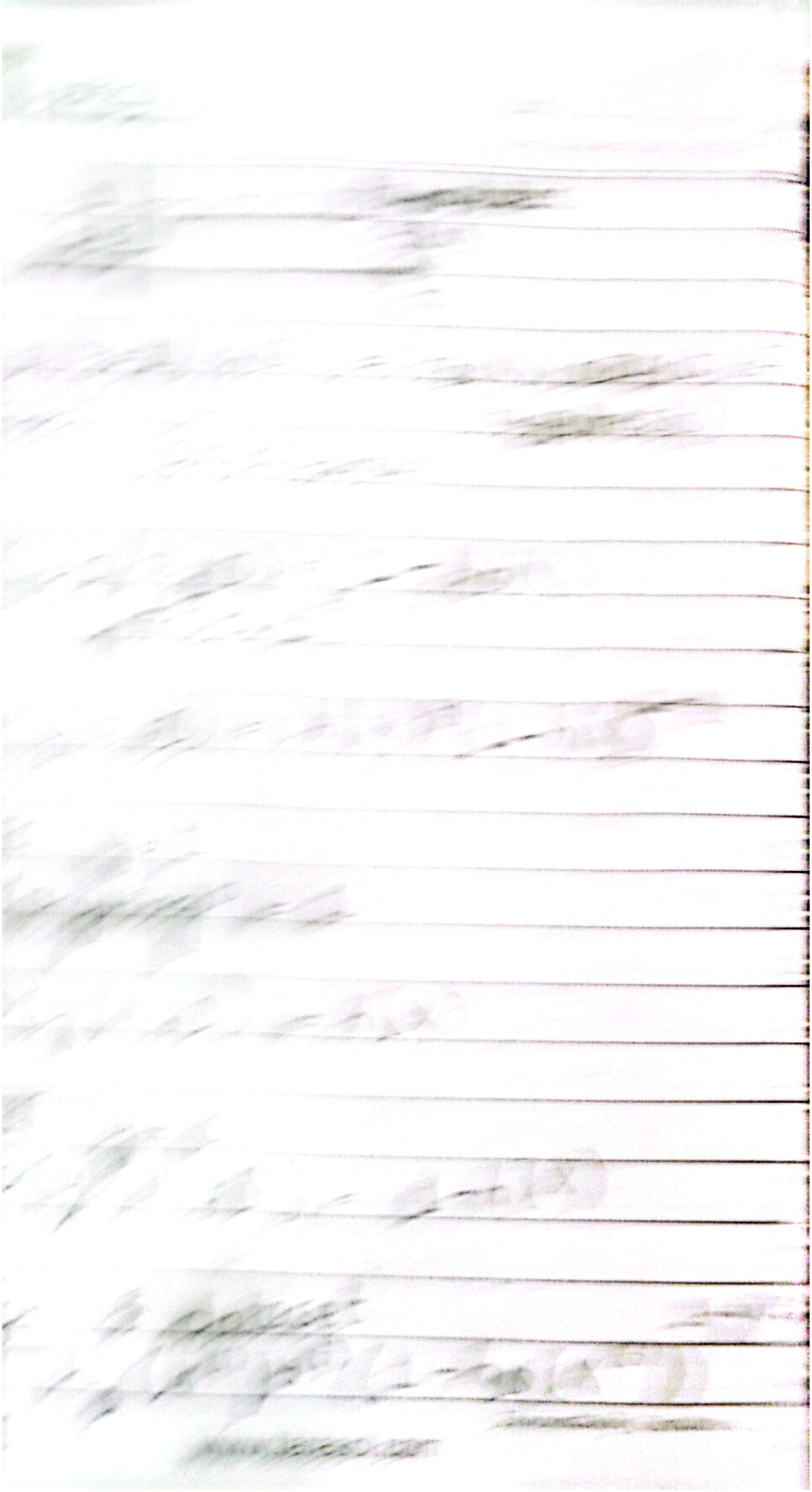
$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$

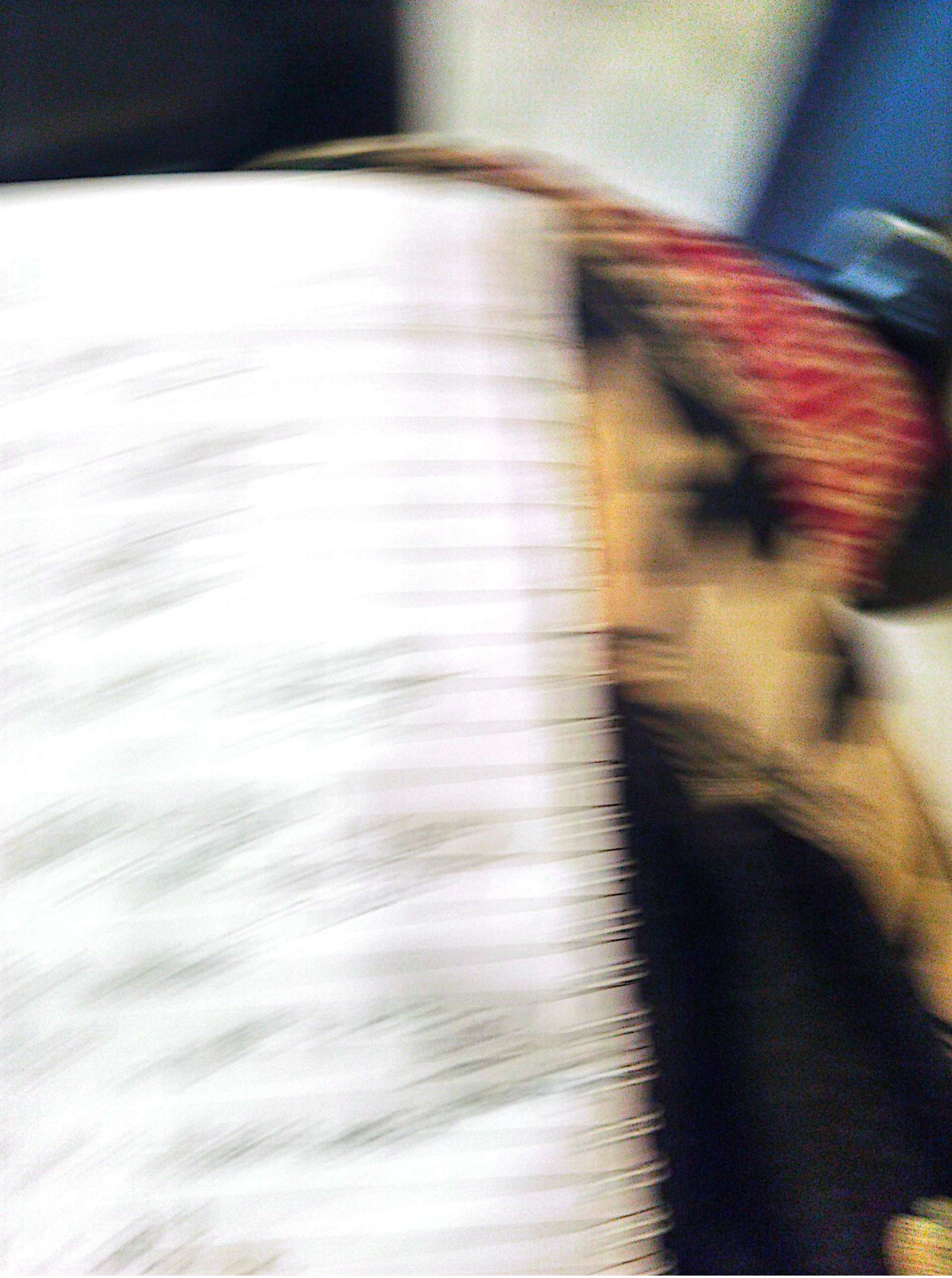
estimation

$g(z) = \frac{1}{1 + e^{-z}}$ out: $0 < g(z) < 1$

Sigmoid or logistic function

$h_\theta(x) = \theta^T x$





→ check for empty value
p.isnull().sum()

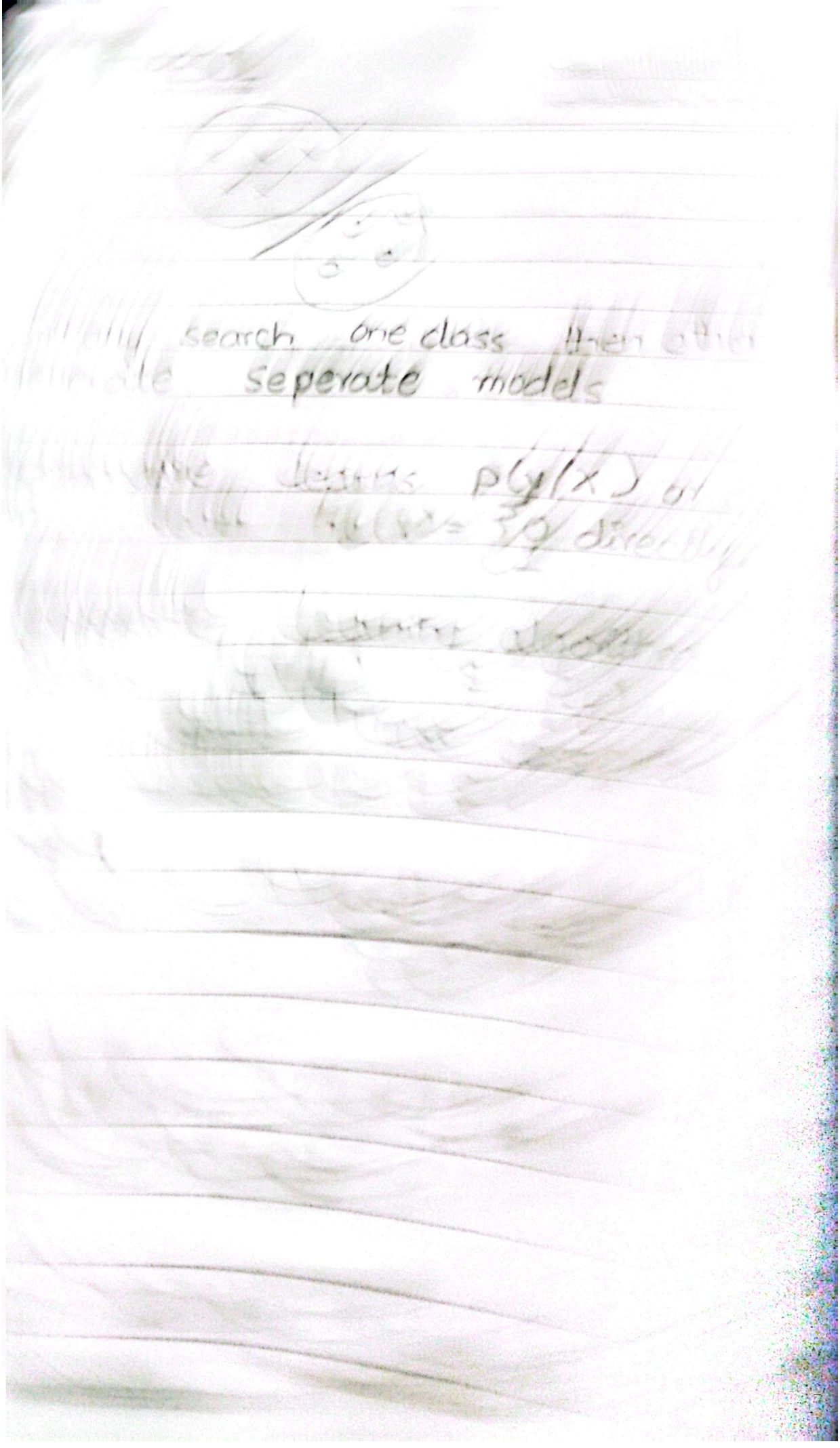
→ Linear Regression
Normal equation, $\theta = X^T y$
Error = prediction - actual
↓ Hypothesis $- y$
$h_\theta(x) - y$

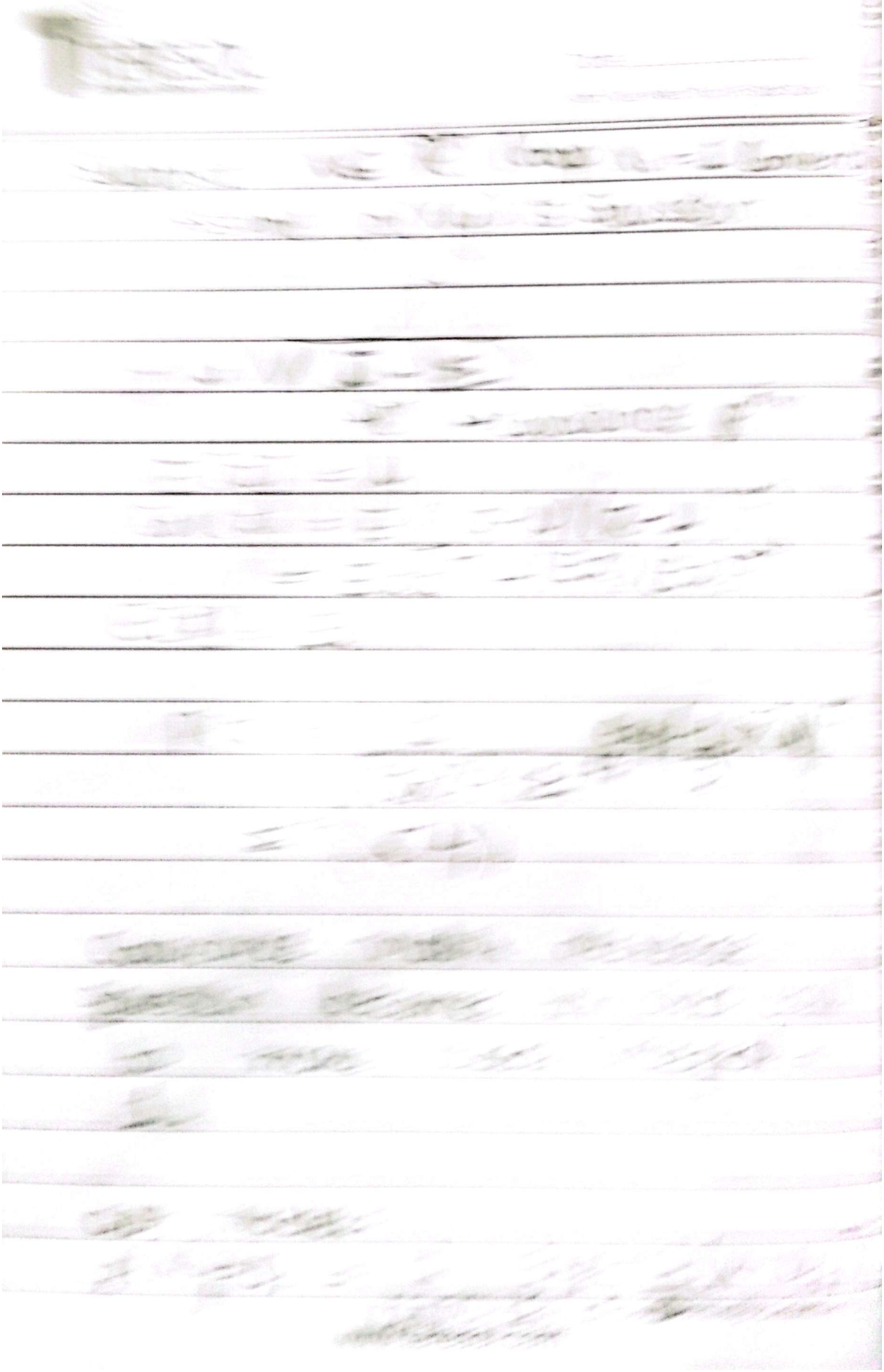
$$\text{cost} = \text{mse} = \frac{1}{2} m (h_\theta(x) - y)^2$$

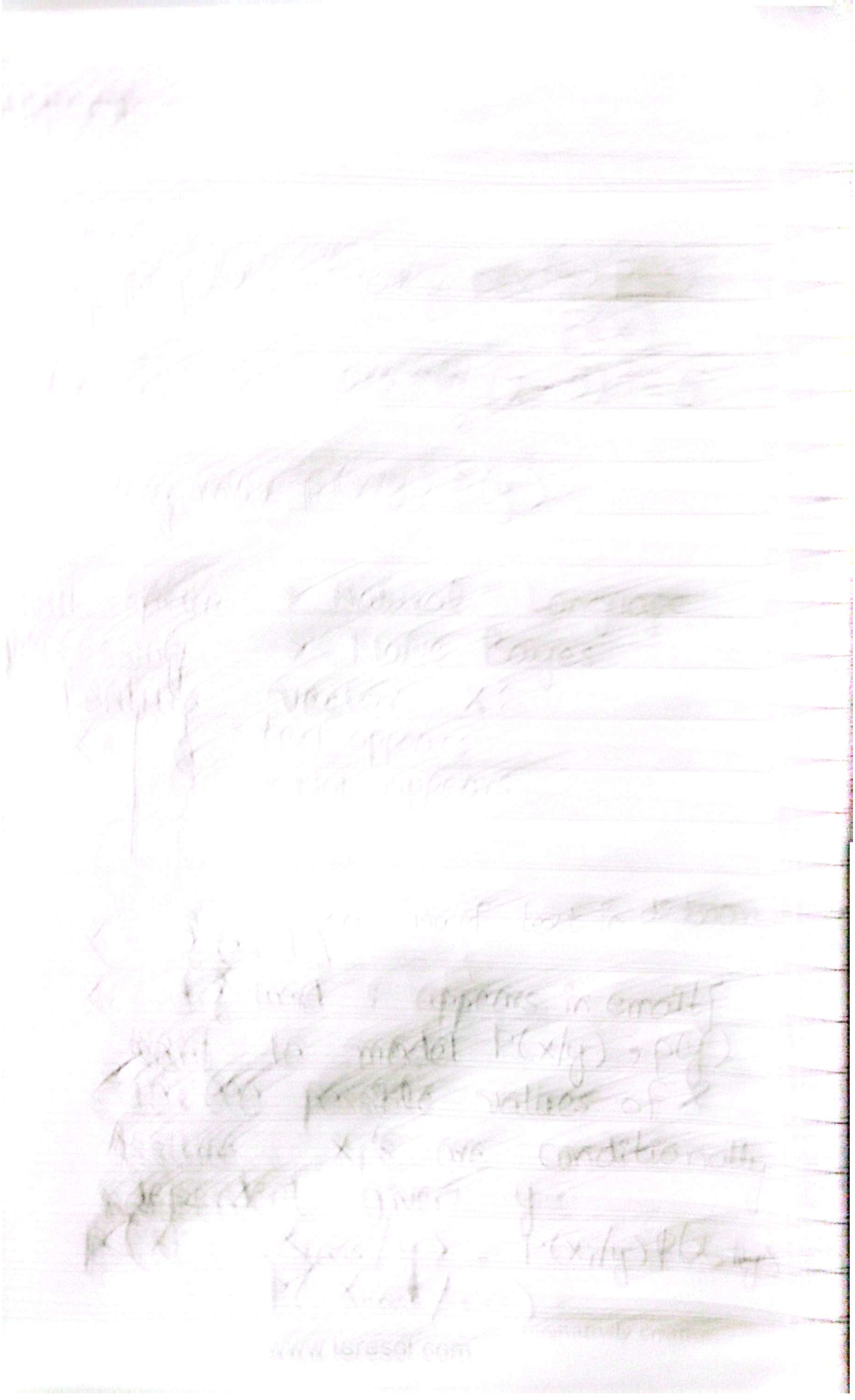
gradients = θ & cost
$\theta = \theta - (\text{learning rate} * \text{gradient})$
$$\text{gradient} = \frac{1}{m} * (x)(h_\theta(x) - y)$$

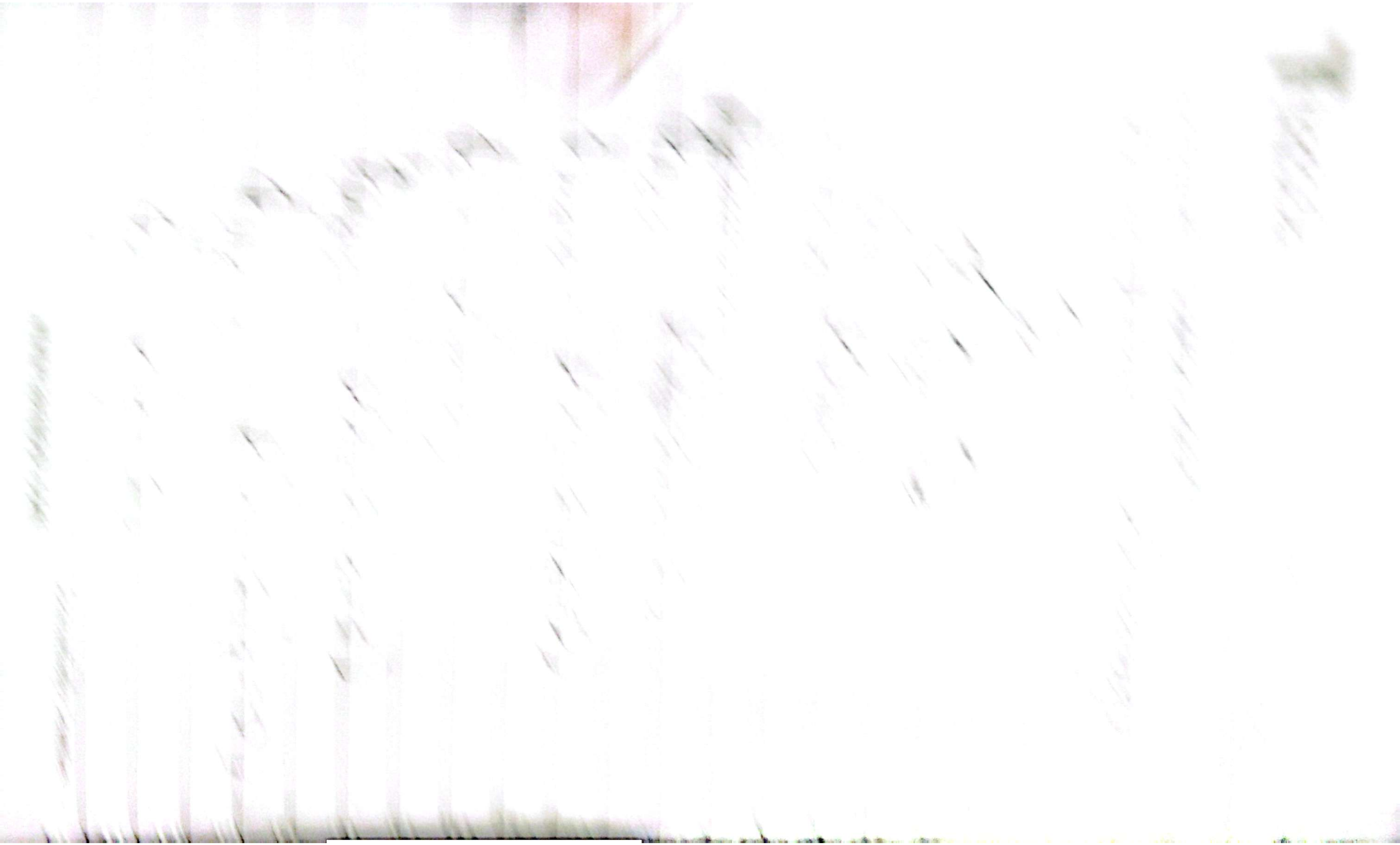
$h_\theta(x) = \theta^T x$
↓ Hypothesis

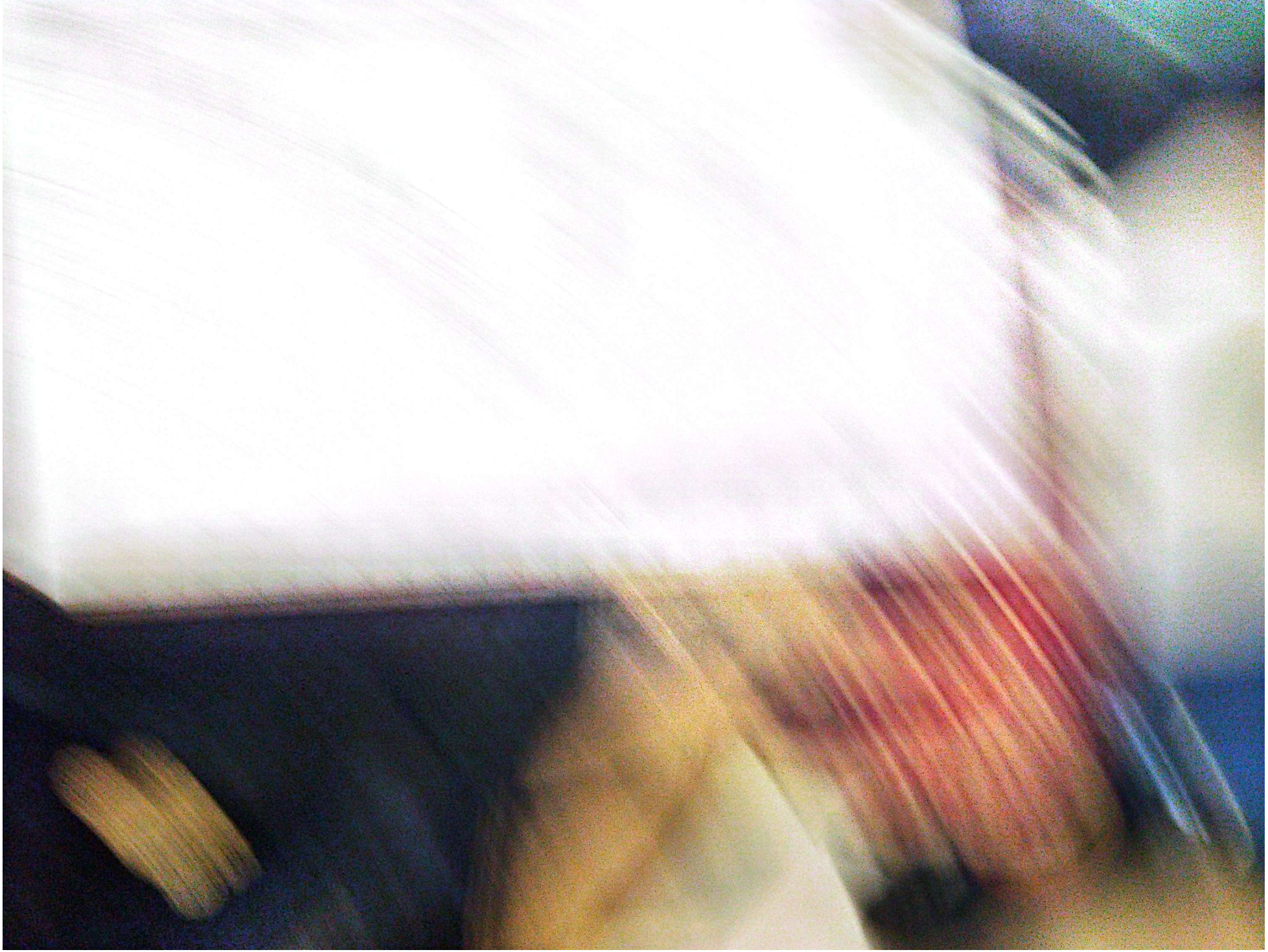
→ Scikit Learn module
from sklearn.learn_model import LinearRegression
Lin_reg = LinearRegression()











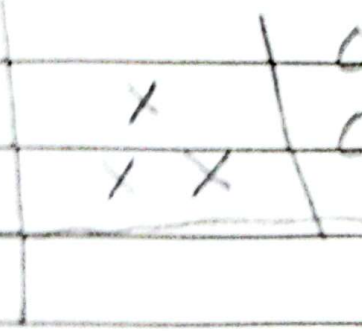
→ Support Vector Machines (SVM)
↓ non-linear decision boundary

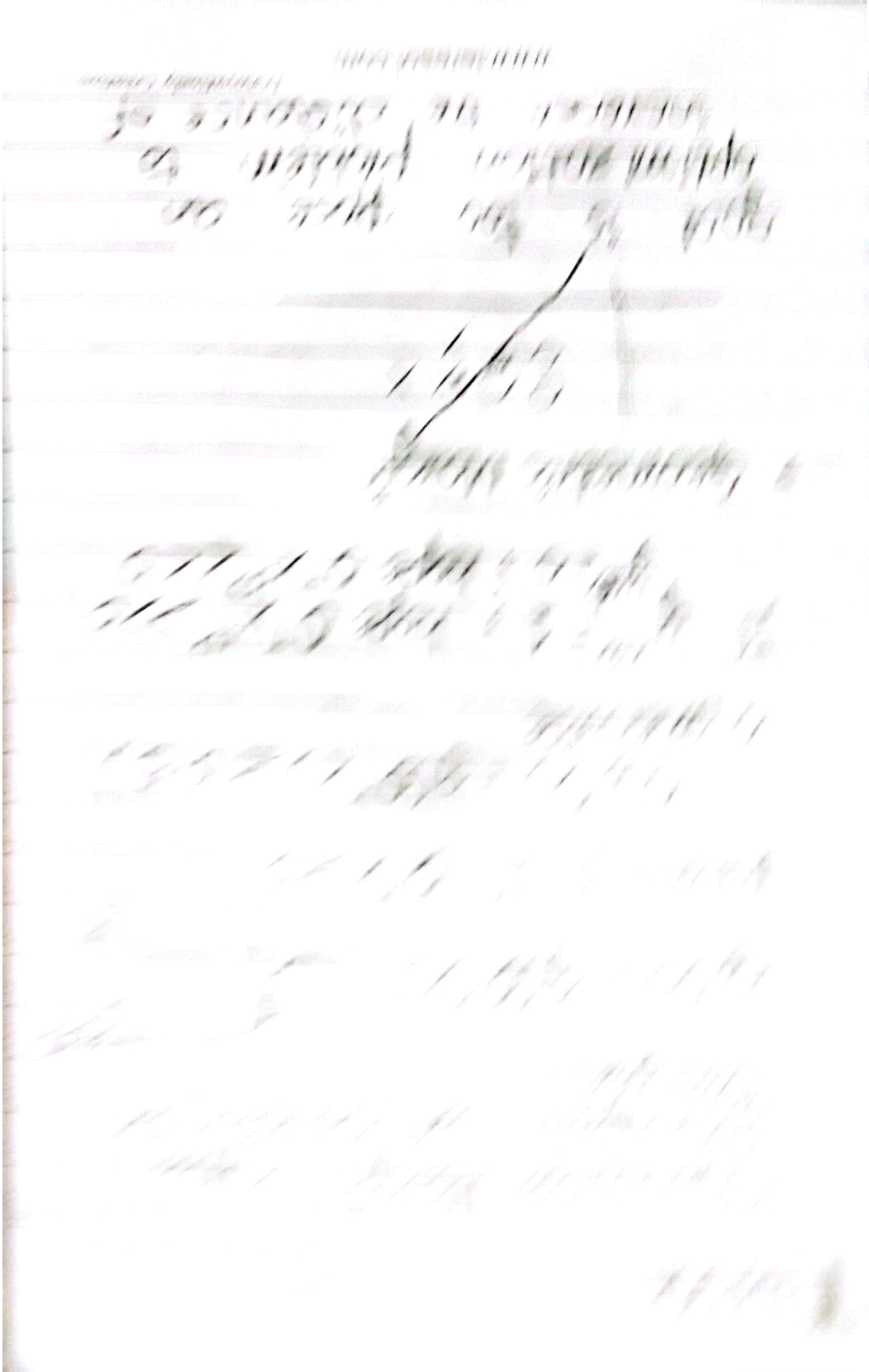
x_1 x_2 features
then high dimensional feature

$\phi(x) =$ | x_1 |
| x_2 |
| x_1^2 |
| x_2^2 |
| $x_1 x_2$ |

Less parameterized ↓
No learning rate & Simple

→ optimal margin classifier
(seperable case)





If $y^{(i)} = -1$, $w^T x^{(i)} + b < 0$

If $\hat{\gamma}^{(i)} > 0$,
that means $h(x^{(i)}) = y^{(i)}$

Functional margin w.r.t training set.

$\hat{\gamma} = \min_i \hat{\gamma}^{(i)}$ → worst example in our training set

$i = 1, \ldots, m$

$w \leftarrow 2w$

$b \leftarrow 2b$

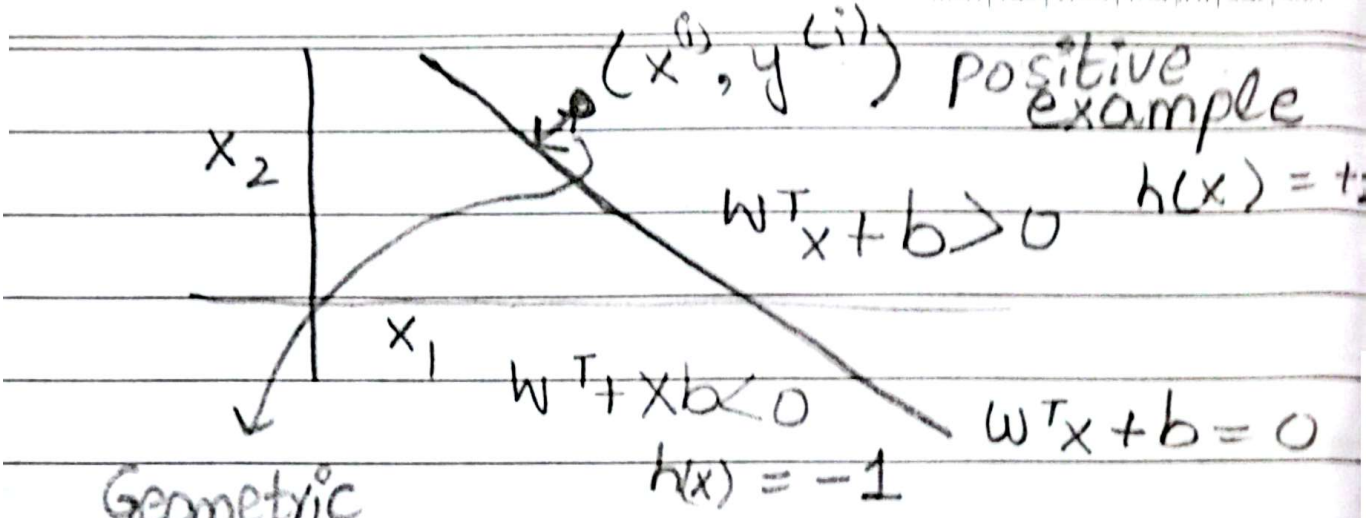
functional margin increases

Normal
$\|w\| = 1$

$(w, b) = \left(\frac{w}{\|w\|}, \frac{b}{\|w\|}\right)$

↳ Euclidean length

Geometric margin



Geometric Margin :

Geometric margin of hyperplane (w, b) wrt $(x^{(i)}, y^{(i)})$

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b)$$

$$\gamma^{(i)} = \frac{\hat{\gamma}^{(i)}}{\|w\|}$$

$$\gamma^{(i)} = \frac{\hat{\gamma}^{(i)}}{\|w\|}$$

Geometric margin w.r.t training set

$$\gamma = \min_i \gamma^{(i)}$$

$\hat{\gamma}$ = functional margin

γ = geometric Margin

→ Optimal margin classifier

Choose w, b to maximize γ

$$\max_{\gamma, w, b} \gamma$$

$$\text{Such that: } \frac{y^{(i)}(w^T x^{(i)} + b)}{\|w\|} \geq \gamma$$

$$i = 1, \ldots, m$$

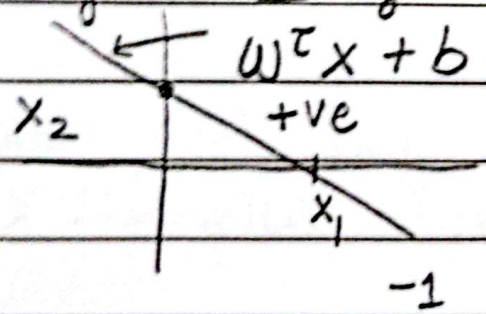
Reformulate,

$$\left.\begin{array}{l} \min_{w,b} \|w\|^2 \\ \text{Such that } y^{(i)}(w^T x^{(i)} + b) \geq 1 \end{array}\right\}$$

→ Optimization problem:

$\min_i j^{(i)} >>$ To make it big

$j^{(i)} \geq \gamma$



Date

Mon | Tue | Wed | Thu | Fri | Sat | Sun

$$\min \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y^{(i)} y^{(j)} \underbrace{x^{(i)T} x^{(j)}}_{\langle x^{(i)}, x^{(j)} \rangle}$$

$\langle x, z \rangle = x^T z$ is the inner product between values.

$$y^{(i)} \left(\sum_j \alpha_j y^{(j)} x^{(j)} \right)^T x^{(i)} + b \geq 1$$

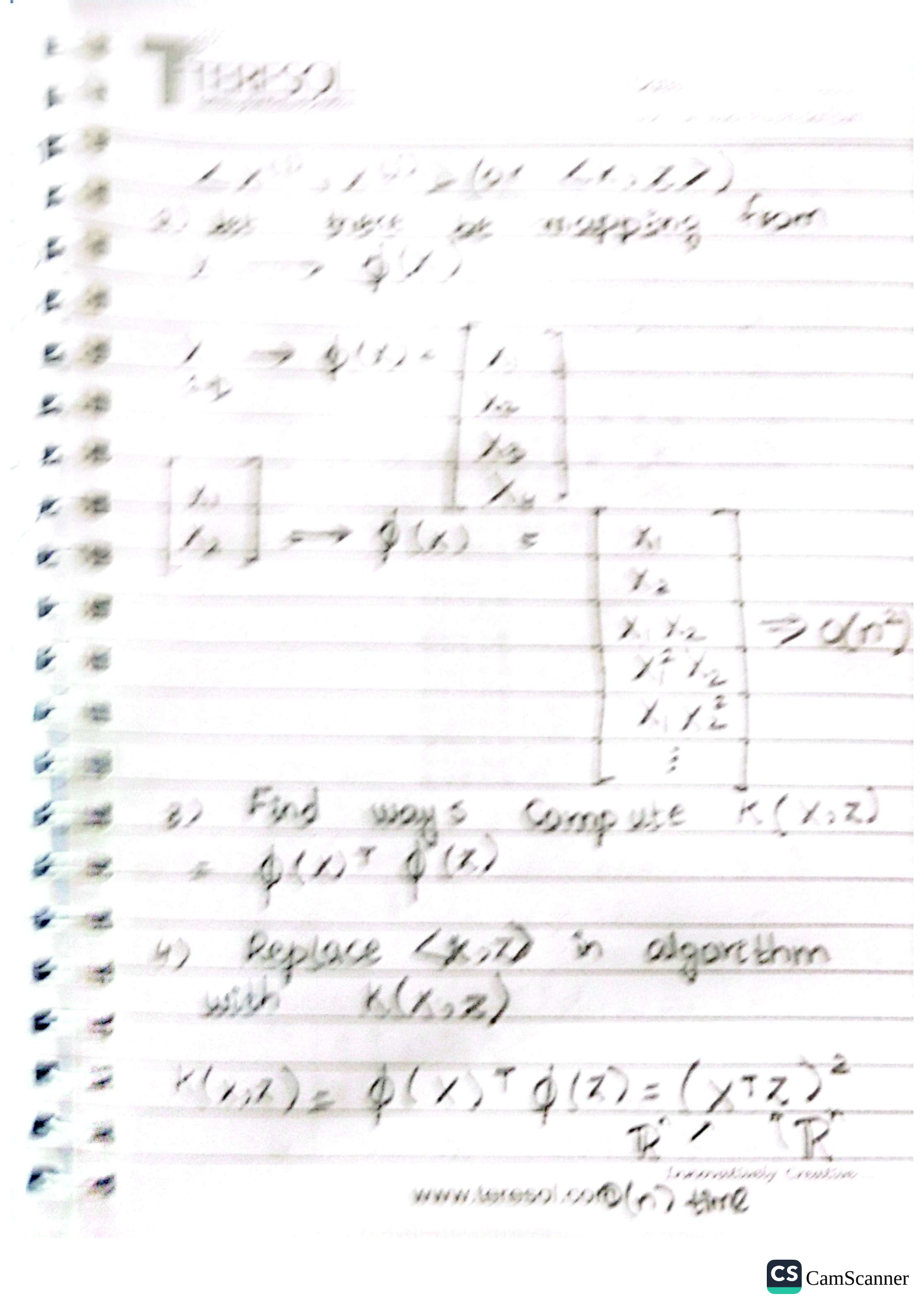
$$y^{(i)} \left(\sum_j \alpha_j y^{(j)} \langle x^{(j)}, x^{(i)} \rangle + b \right) \geq 1$$

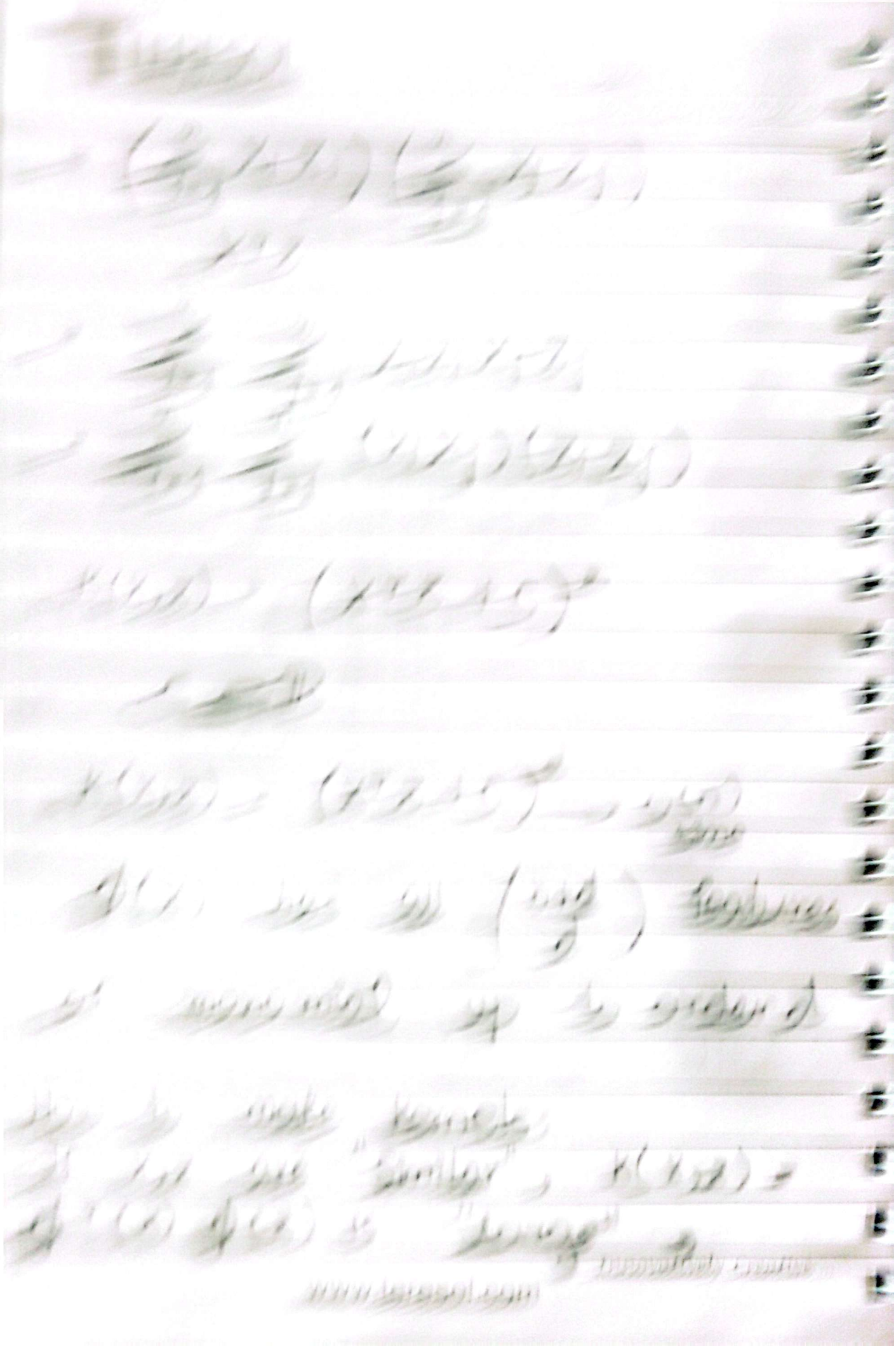
$$\max \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$$

$$h_{w,b}(x) = g(w^T x + b)$$

$$= g\left(\left(\sum_i \alpha_i y^{(i)} x^{(i)} \right)^T x + b \right)$$

$$= g\left(\sum_i \alpha_i \langle x^{(i)}, x \rangle + b \right)$$





Date

re dissimilar, $K(x, z)$ is

$$, z) = \exp\left(\frac{-\|x - z\|^2}{2\sigma^2}\right)$$

Kernel: $K(x, z) = x^T z$

$$\phi(x) = x$$

$$\phi(x) \in R^N$$

$$\|w\|^2 + C \sum_{i=1}^{m} \xi_i$$

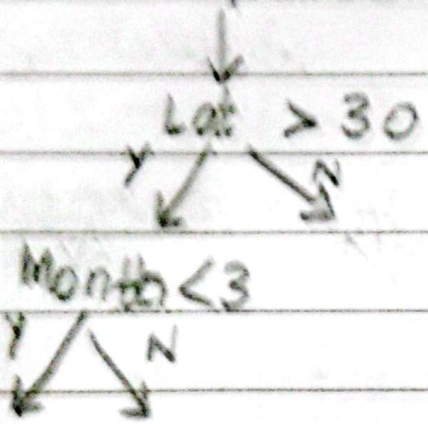
$$y^{(i)}(w^T x^{(i)} + b) \geq 1$$

$$i = 1, \ldots, m$$

$$\geq 0$$

trees:

Also known as Greedy, Top down, Recursive partioning



Region R_p, Looking for a split

$S_p(j, t) = (\{x / x_j < t, x \in R_p\}, \{x_j \geq t, x \in R_p\})$

R_1 R_2

How to choose splits

Define $L(R) \rightarrow$ Loss on R

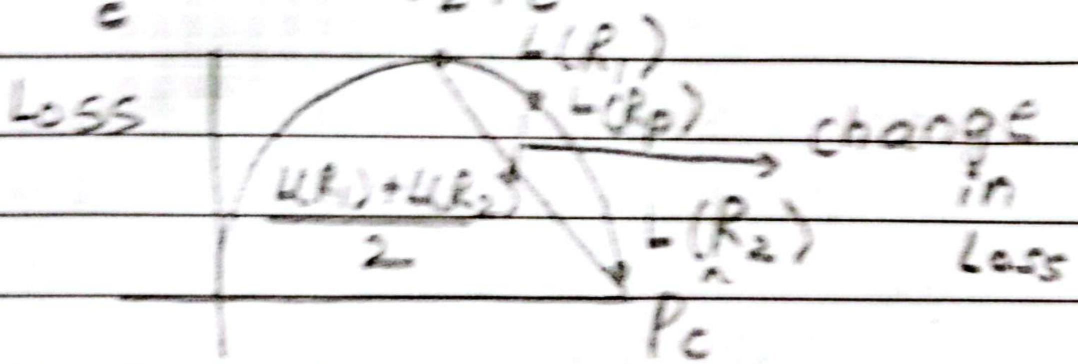
Given c classes, define $\hat{p}_c$ be the proportion of examples in R that are of class c.

$$L_{misclass} = 1 - \max_c \hat{p}_c$$

$$\min \; \underbrace{L(R_p)}_{\text{Parent Loss}} - \underbrace{(L(R_1) + L(R_2))}_{\text{Children Loss}}$$

Instead define Cross entropy loss

$$\text{Loss}_{\text{Cross}} = -\sum_{c} \hat{p}_c \log_2 \hat{p}_c$$



Predict $$\hat{Y}_m = \frac{\sum_{i \in R_m} Y_i}{|R_m|}$$

$$L_{\text{squared}} = \frac{\sum_{i \in R} (y_i - \hat{Y}_m)^2}{|R_m|}$$

→ Regularization of decision trees

1) Min leaf size
2) Max depth
3) Max number of nodes
4) Min decrease in Loss
5) pruning (Misclassification with val set)

Before split : $L(R_p)$
After split : $L(R_1) + L(R_2)$

→ Runtime
n examples
f features
d depth

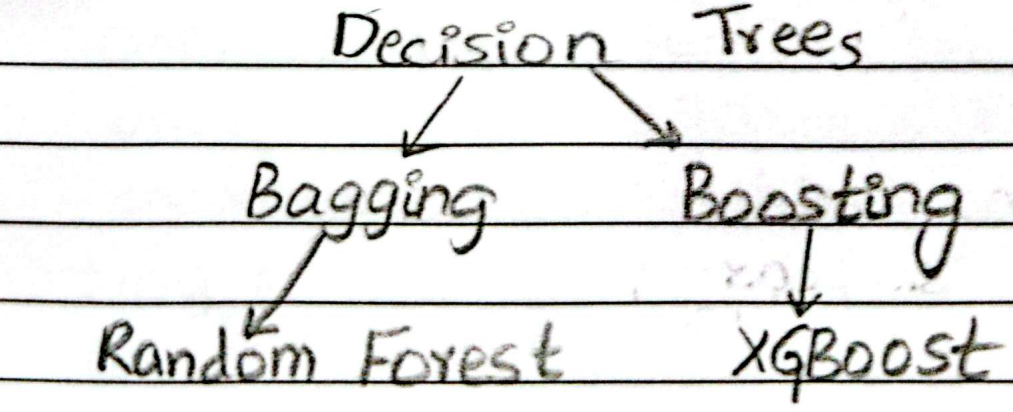
Test:
O(d)
$d < \log_2 n$

Train:
Each point is part of O(d) nodes.

Cost of point at each node is $O(f)$

Total cost is $O(nfd)$
Data matrix is of size nf.

- ⊕ Easy to explain
- ⊕ Interpretable
- ⊕ Categorical var
- ⊕ Fast
- ⊖ High Variance
- ⊖ Bad at additive
- ⊖ Low predictive



→ Decision trees

```
from sklearn import tree
from sklearn.model_selection import
 train_test_split
from sklearn.metrics import accuracy_
score, classification_report
 dt = tree.DecisionTreeClassifier()
  tree.plot_tree(dt) → After training
```

→ Random Forest

```
from sklearn.ensemble import
 RandomForestClassifier
 rf = RandomForestClassifier()
  tree.plot_tree(rf.estimators_[i],
   filled = True)
```

→ Filling NaN values

```
from sk.learn.impute import
 simpleImputer

imputer = SimpleImputer(missing_
values = np.Nan, strategy = 'mean')
```

```
Columns = ['bill_depth_mm', 'bill_
  length_mm', 'flipper_length_mm',
  'body_mass_g']
  imputer = impute.fit(df[Columns])
df[Columns] = impute.transform
      (df[Columns])
```

→ .unique() returns names on which 0,1 mapping cannot be done.

→ When numerical data cannot be assigned as 0, 1.
Then required 0, 1, 2, then .map() cannot be used.
One-Hot encoding is used

```
from sklearn.preprocessing import
OneHotEncoder
one_hot = OneHotEncoder()
encoded = one_hot.fit_transform([
    df['island']]) or (df[['island']]
df[one_hot.categories_[0]] =
    encoded.to_array()
```

→ Displaying random Forest

```
plot_tree(forest.estimators_[0],
    feature_names = X.columns,
    class_names = df['species'].
                  unique(),
    filled = True, rounded = True)
```

→ K-NN algorithm:
- → Feature Similarity
- → K-NN classifier

includes number of nearest neigbours in voting process

→ parameter tuning for obtaining K value.

step → ① K = Sqrt (n = total value)

step → ② Always make K = odd

→ use

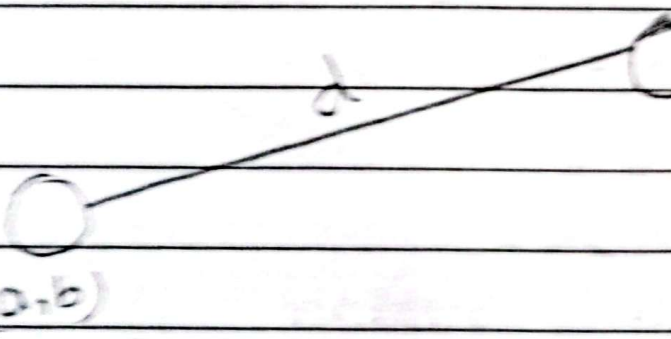
① Data Labelled
② Dataset is small
③ Noise-free data
④ It is lazy learner.

Date .....

Mon Tue Wed Thu Fri Sat Sun

To find nearest neighbour, we obtain euclidean distance

euclidean distance $= \sqrt{(x-a)^2 + (y-b)^2}$



now obtain K and take K nearest neighbours

```
import pandas as pd
import numpy as np
from sklearn.model_selection import
train_test_split
from sklearn.preprocessing import
StandardScaler → from removing biased
from sklearn.neighbors import
KNeighborsClassifier
```

→ Replace zero by NaN

```
zero_not = ['col1', 'col2' ... ]
```

```
for column in zero_not:
    data[column]= data[column].replace
        (0, np.nan)
    mean = int (data[column].mean
    (skipna = True))
          ↳ skips NaN
```

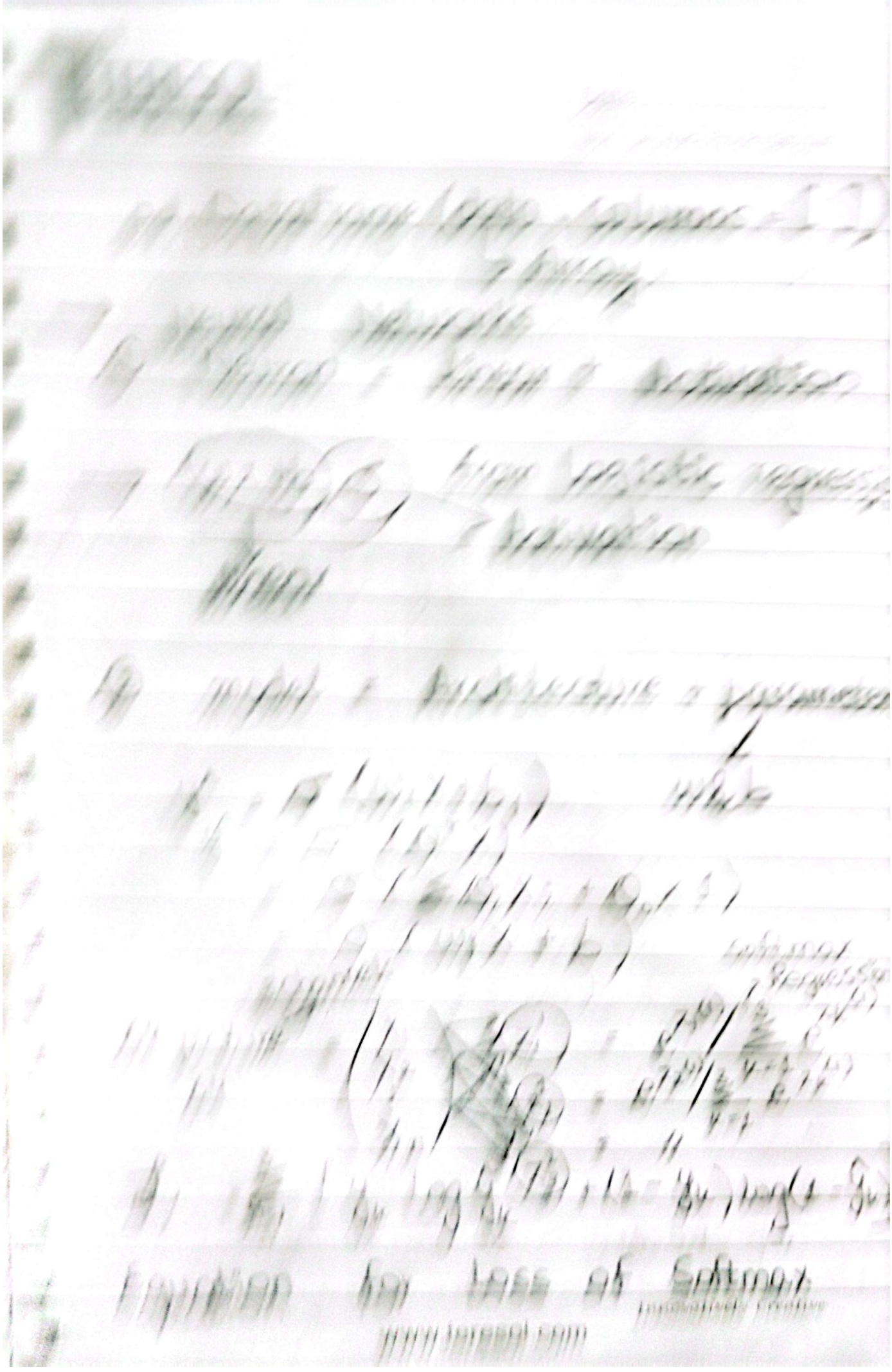
→ Standard Scaling

```
sc_X = StandardScaler()
X_train = sc_X.fit_transform
    (X_train)
X_test= sc_X.fit_transform
    (X_test)
```

```
kn = KNeighborsClassifier(n_neighbors
    K , p = ? , metric = 'euclidean')
          ↓ No of outcomes
    ↳ √len(y_test)
```

→ K-clustering Algorithm :

```
from sk.learn.cluster import KMeans
```



equations:

Softmax / Cross entropy Loss

$$-\sum_{k=1} y_k \log \hat{y}_k$$

Rayleigh function

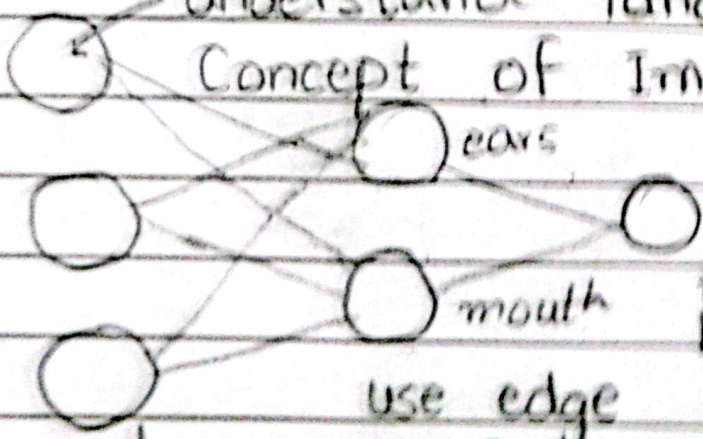
$-\infty \longrightarrow 0 = 0$

$0 \longrightarrow \infty$ = value other than sigmoid function.

For regression

II Neural Networks

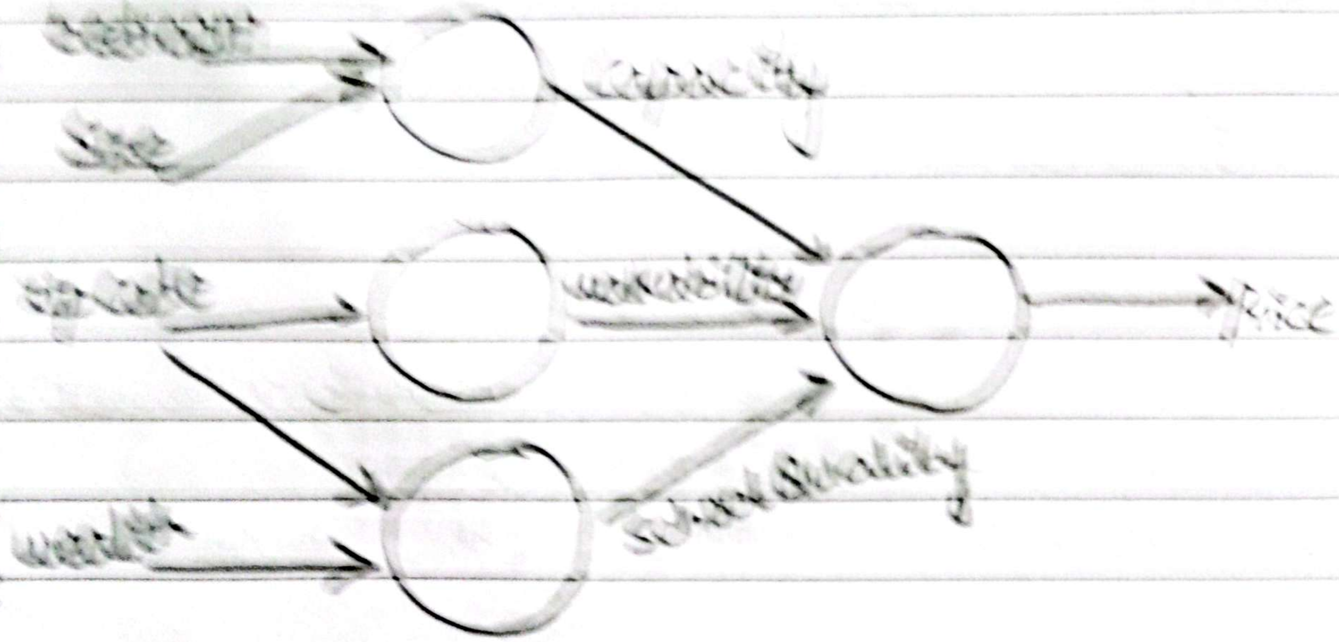
understand fundamental Concept of Image



construct the face based on information

use edge to figure object

House price prediction



deep learning / end to end learning / blackbox model

→ Propagation equation

(3,1) $z^{[1]} = W^{[1]}_{(3,4)} x_{(4,1)} + b^{[1]}$

(3,1) $a^{[1]} = \sigma[z^{[1]}]$

(2,1) $z^{[2]} = W^{[2]}_{(2,3)} a^{[1]}_{(3,1)} + b^{[2]}_{(2,1)}$

(2,1) $a^{[2]} = \sigma[z^{[2]}]$

(1,1) $z^{[3]} = W^{[3]}_{(1,2)} a^{[2]}_{(2,1)} + b^{[3]}_{(1,1)}$

(1,1) $a^{[3]} = \sigma(z^{[3]})$

adjust weight = error . input . ϕ'(sigmoid)

$$\phi(x) = \frac{1}{1+e^{-x}} \quad \& \quad \phi'(x) = x(1-x)$$

→ Introduction to pyCaret

1) classification

a) class pyCaret.classification.ClassificationExperiment (Determines a multiclass)

→ use:

from pyCaret.classification import *(all)

Setup (data = "input" , target = "out")

parameters in Setup:

data = "input" with n-samples and n features.

data_func = inplace of data when data set is large and require parallel operations

target : int , str or sequence: (default: -1) .

sdect output col

train_size : float , default = 0.7
test_data : If train_size is ignored then test data used.

ignore_features = list of str
preprocess : bool , default = True
imputation_type : str or None , default = 'simple' or iterative
numeric_imputation : int , float or str , default = 'mean'
normalize : bool
categorical_imputation = 'str'
str = drop , mode , str
use_gpu = bool
session_id : int

2) Model selection
compare_models()

3) Model selection
best_model = automL(optimize = 'Accuracy')

4) Tune

tune = tune_model (best_model)

or tune_model (best_model, optimiz

or dt = create_model ('dt') 'Auc')

lr, knn, nb, dt, svm, rbfsvm, gpc, mlp, ridge, rf, qda, ada, gbc, lda, et, xgboost

→ ensemble_model (dt, method = 'Bagging')

~ Boosting

5) predict

pred = predict_model (tune, data = test_data)

6) plot_model

plot_model (tune, plot = 'auc', 'pr', 'feature', 'confusion_matrix')

used by evaluate_model()

distribute and exploit all available Cpus available in local Computer.

→ Load data faster affect Cpu load, increase or decrease RAM Usage (speeds up training if data is loaded more into RAM if Gpu is waiting for data. Bottleneck occurs.

→ Variance refers to model dependance on training data

→ Model tends to fit up noise as well as data → overfitting

→ underfit → high training & testing error

→ overfit → Low training & high test error

→ Epoch: Entire dataset passed forward to back

→ Batch size: total Number of training examples present in single batch.

→ Batch: Divide dataset into patches

→ Iterations: Number of batches required to complete one epoch

Underfitting → Optimal → Overfitting
low more most

Steps per epoch = $\frac{\text{No of training Example}}{\text{Batch size}}$

Total steps = Steps per Epoch X Number of epoch

Data loader → Batch size and dataset.

len (Data loader) = $\frac{\text{Total No of Examp}}{\text{Batch Size}}$

→ Random state:
splitting testing and training data, controls shuffling applied to data before shuffling it
It ensures reproducibility.

1) If None, it produce different results in every execution.

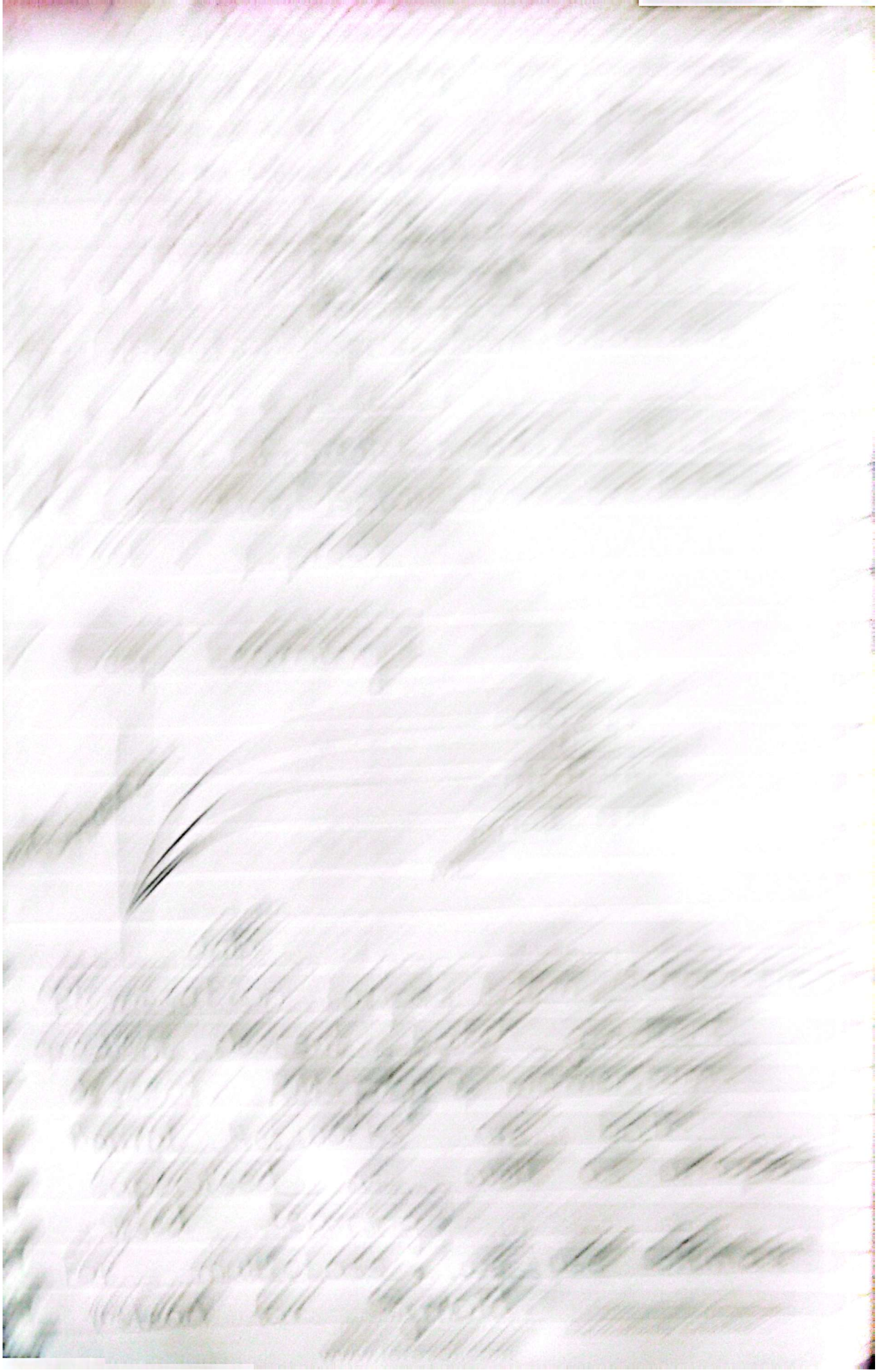
2) If integer ... it will produce the same result for an integer value.
If integer value is changed, then only the result will be different.

For dataset:

training data   Testing data
Training size!   Testing size!

Total Combination = Total data set! / (Training)! * (Total - Train)!

→ Scikit-learn polynomials
from sklearn.preprocessing import PolynomialFeatures
→ Calibration:
from sklearn.calibration import calibration_curve, CalibrationDisplay
prob_true, prob_pred = calibration_curve(y_true, y_pred, n_bins=3)
disp = CalibrationDisplay(prob_true, prob_pred, y_prob)
disp.plot()



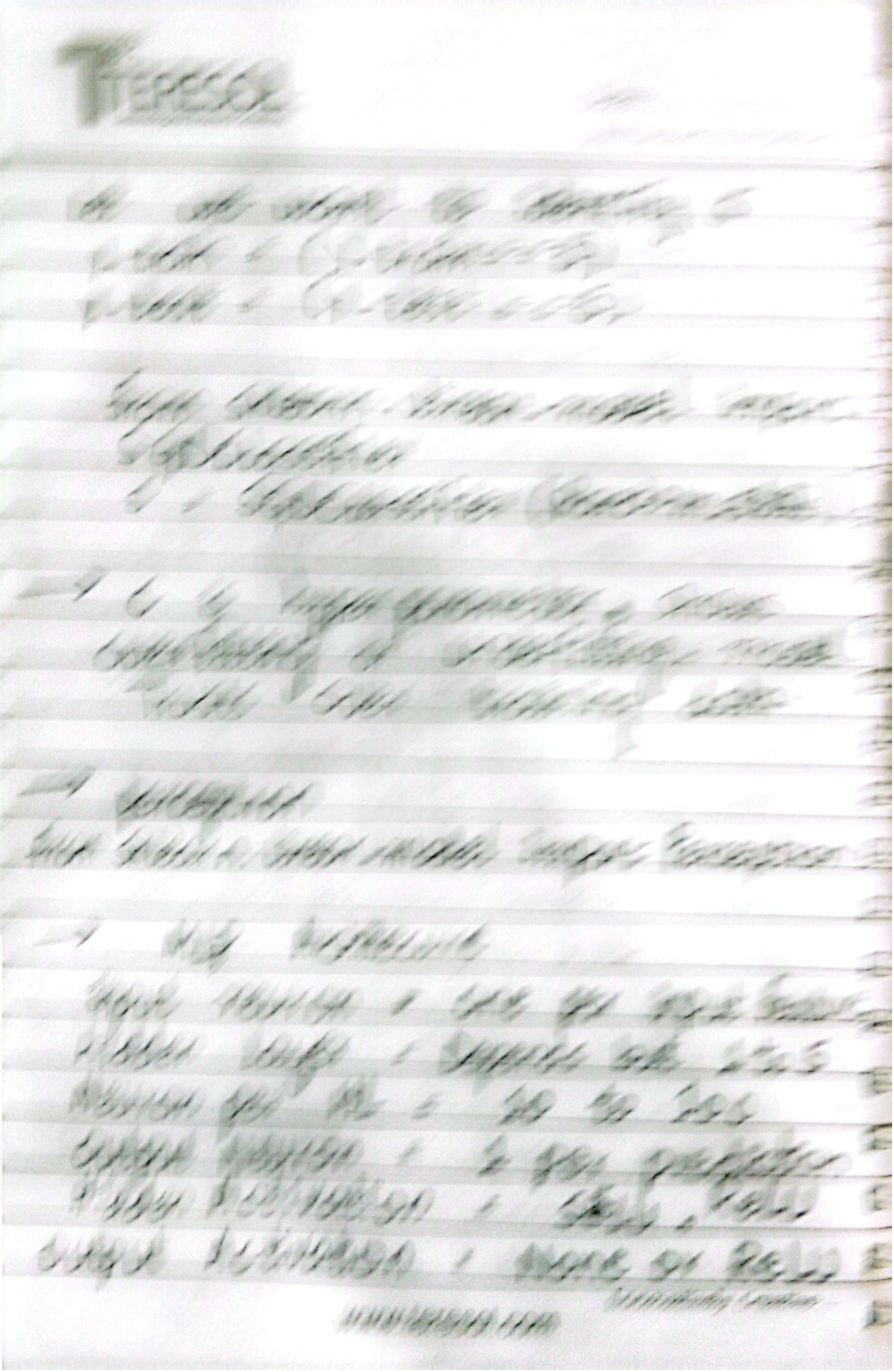
→ Reinforcement learning → Hit & trial & error using feedback from its own actions & experience learning using rewards and punishments as signal for positive & negative behaviour.

→ Cross validation → Break dataset into multiple folds, one fold serves as validation set, while trains data.

```
from sklearn.model_selection import KFold
kf = KFold(n_splits = 2)
kf.get_n_splits(X)
for train, test in kf.split(X):
    print(train, test)
cross_val_score(clf, X, y, cv = 2nt)
```

→ MNIST dataset

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784',
```



(softplus (if +ve) or Logistic / Tanh
(if bounded output)
Loss function → MSE or MAE / Huber
(if outliers)

# Deep learning model
import tensorflow as tf
from tensorflow import keras

# Working with Keras
fash = keras.datasets.fashion_mnist
or
from keras.datasets import fashion_mnist
fash = fashion_mnist
(x_train, y_train), (x_test, y_test) =
fash.load_data()

Relu: Rectified linear function
Activation function inside neural
network decides depending upon
the information recieved from
previous layer activates the neuron

It is a mathematical function that introduces non-linearity to the neural network, enabling model to learn complex patterns and help to make accurate predictions.

① Help understand complex patterns through Non-linearity.

② Activation function derivative define amount by which each weight needs to be updated during backward propagation.

③ It assign different level of importance to different inputs:

$$z = aw_1 + bw_2 + \text{bias}$$

ⓐ Sigmoid:

$$\frac{1}{1 + e^{-z}}$$

→ for binary classification

→ Transformation between 0 & 1.

→ Derivative of sigmoid is very small even for large function

→ 0.5 being midpoint, It tends to produce output biased towards 0.5 make learning rate slower.

→ (b) Tanh:

→ b/w -1 to 1

→ symmetric about zero

→ $\text{Tanh} = \frac{e^{2z} - 1}{e^{2z} + 1}$

→ used when data is symmetric.

→ for even large change, $\frac{\partial}{\partial z} \tanh(z)$ is very small, learning rate is slow.

→ for very small or very large values, tanh saturates, get close to -1 or 1, gradient becomes very close to zero.

→ (c) ReLU → Rectified Linear Unit

→ input value if positive else zero

ReLU = $\max(0, z)$

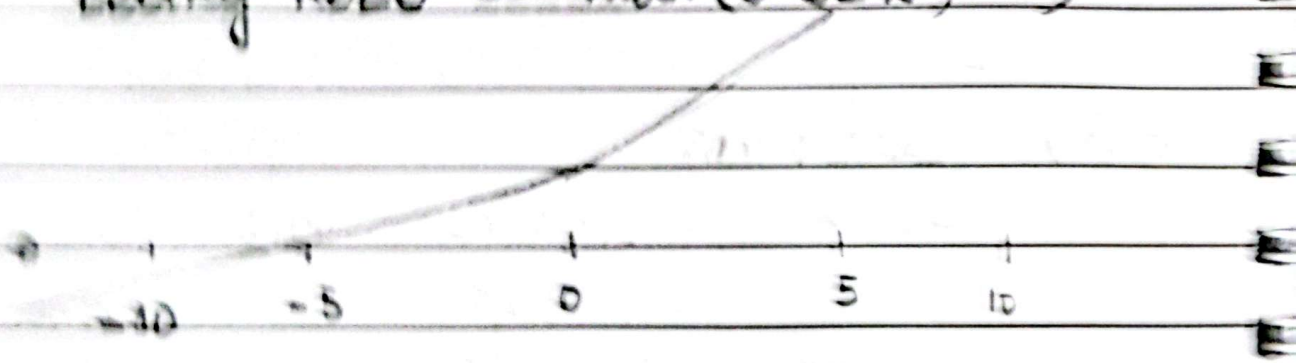
- fast & reduce gradient
- Reduced model capacity as negative neuron doesnot contribute to the learning process. (Dying ReLU).

(d) Leaky ReLU.

→ Address dying neuron issue.

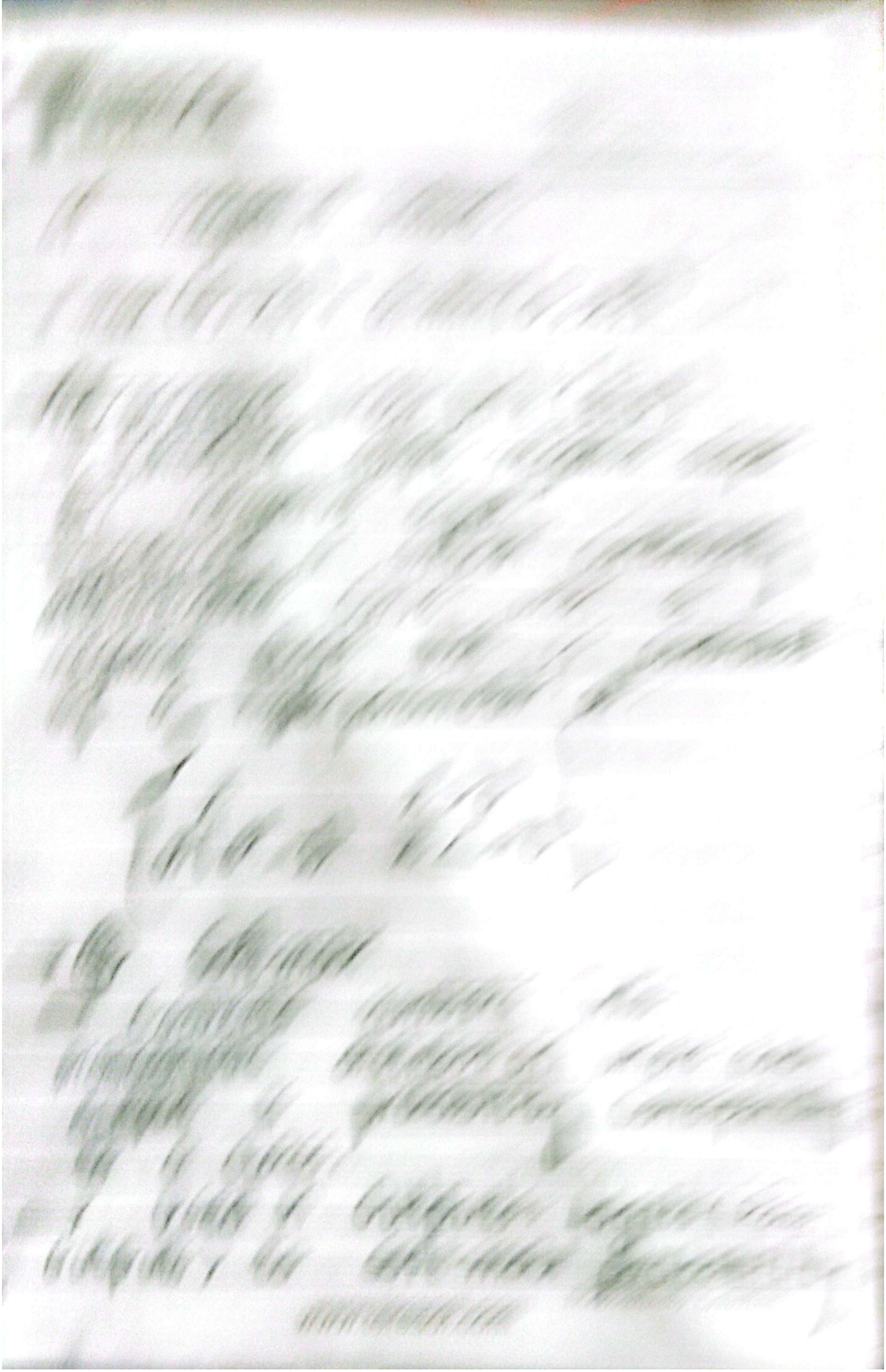
→ makes provision for non-zero output even for negative values by introducing small slope.

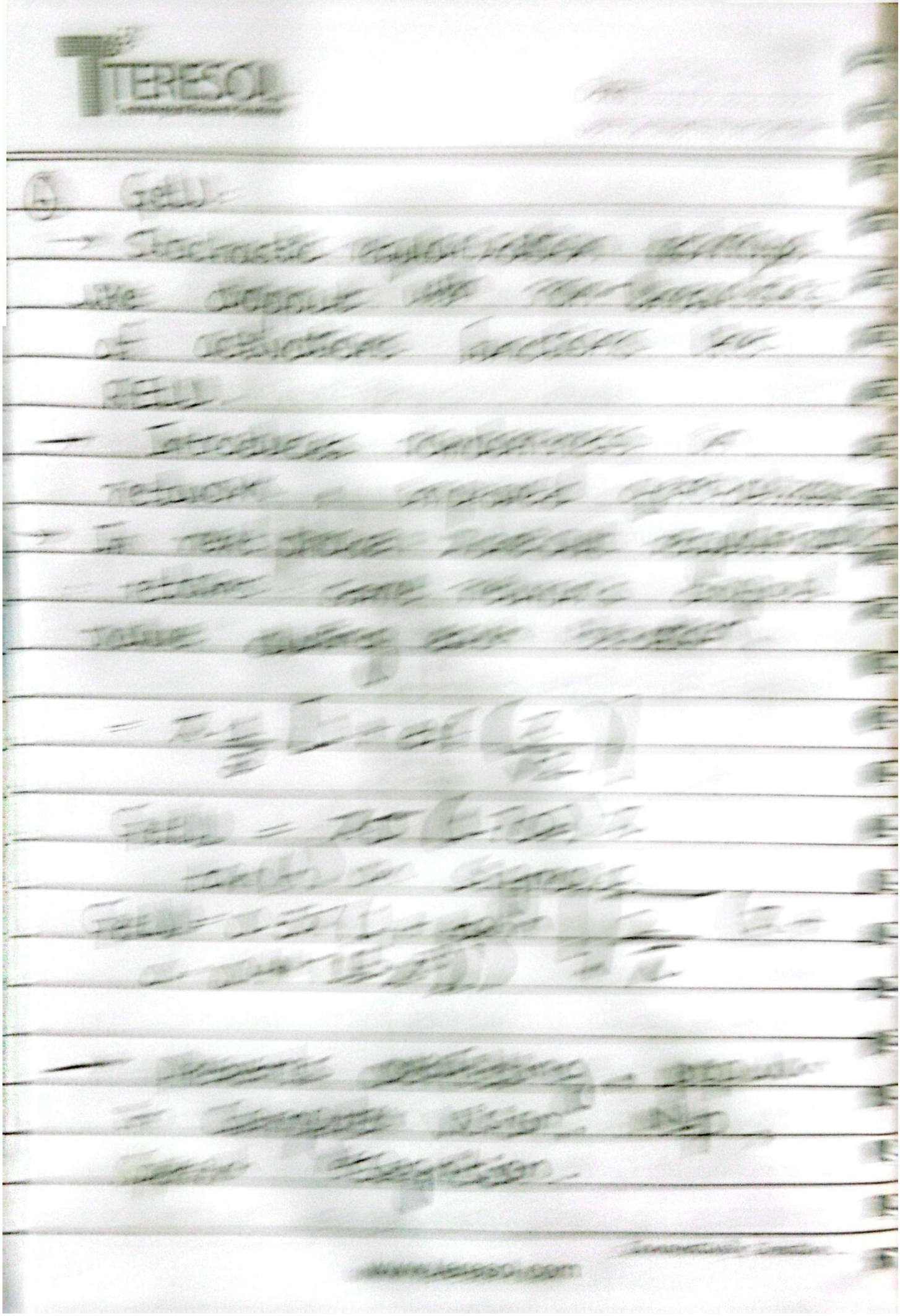
Leaky ReLU = max(0.01z, z)



(e) Parametric ReLU (pReLU)

→ incorporates a learnable parameter to control slope





pd.DataFrame(history.history).plot()

→ model.evaluate(X_test, y_test)

→ y_proba = model.predict(X_test[:3])

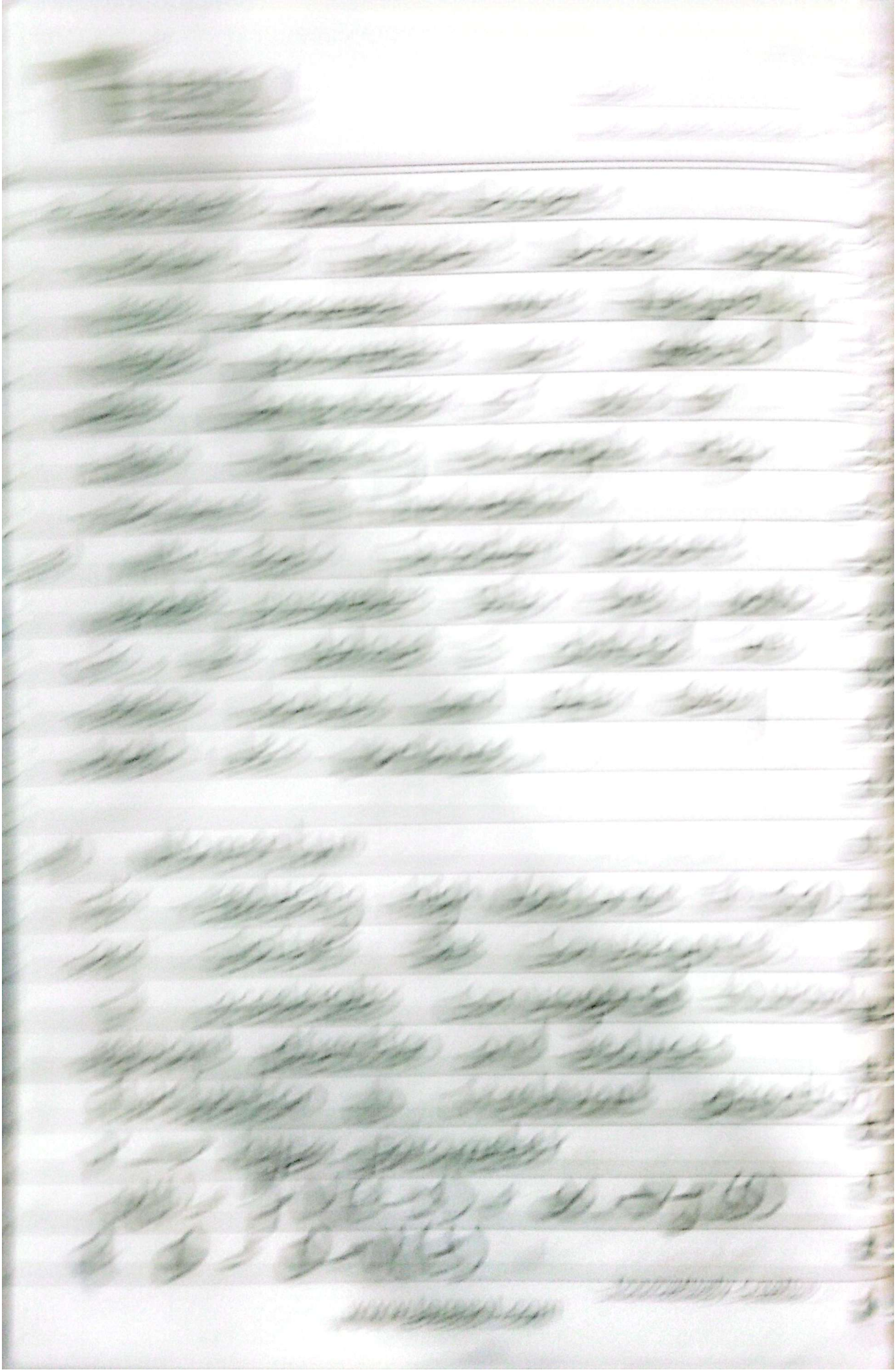
→ y_pred = np.argmax(y_proba, axis=1)
(X_test[:5]) ↓
returns respective Column with max.

→ optimizers
Algorithms used to change attributes of your neural networks such as weights and Learning rate inorder to reduce losses, obtain results faster

) Gradient descent:
First order optimization algorithm.

$\theta = \theta - \alpha \Delta J(\theta)$ (Loss function)

weights are changed after calculating gradient on whole dataset.
Requires large memory to calculate gradient on whole dataset.

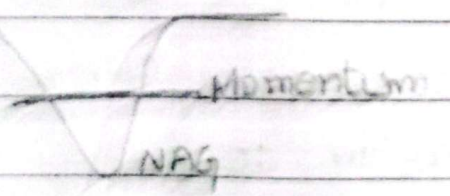


Date

usually $\gamma = 0.9$

5) Nesterov Accelerated Gradient:
Momentum may not be good if momentum is ↑ the algorithm may miss the local minima and continue to rise up.

$$V(t) = \gamma V(t-1) + \alpha \nabla J(\theta - \gamma V(t-1))$$



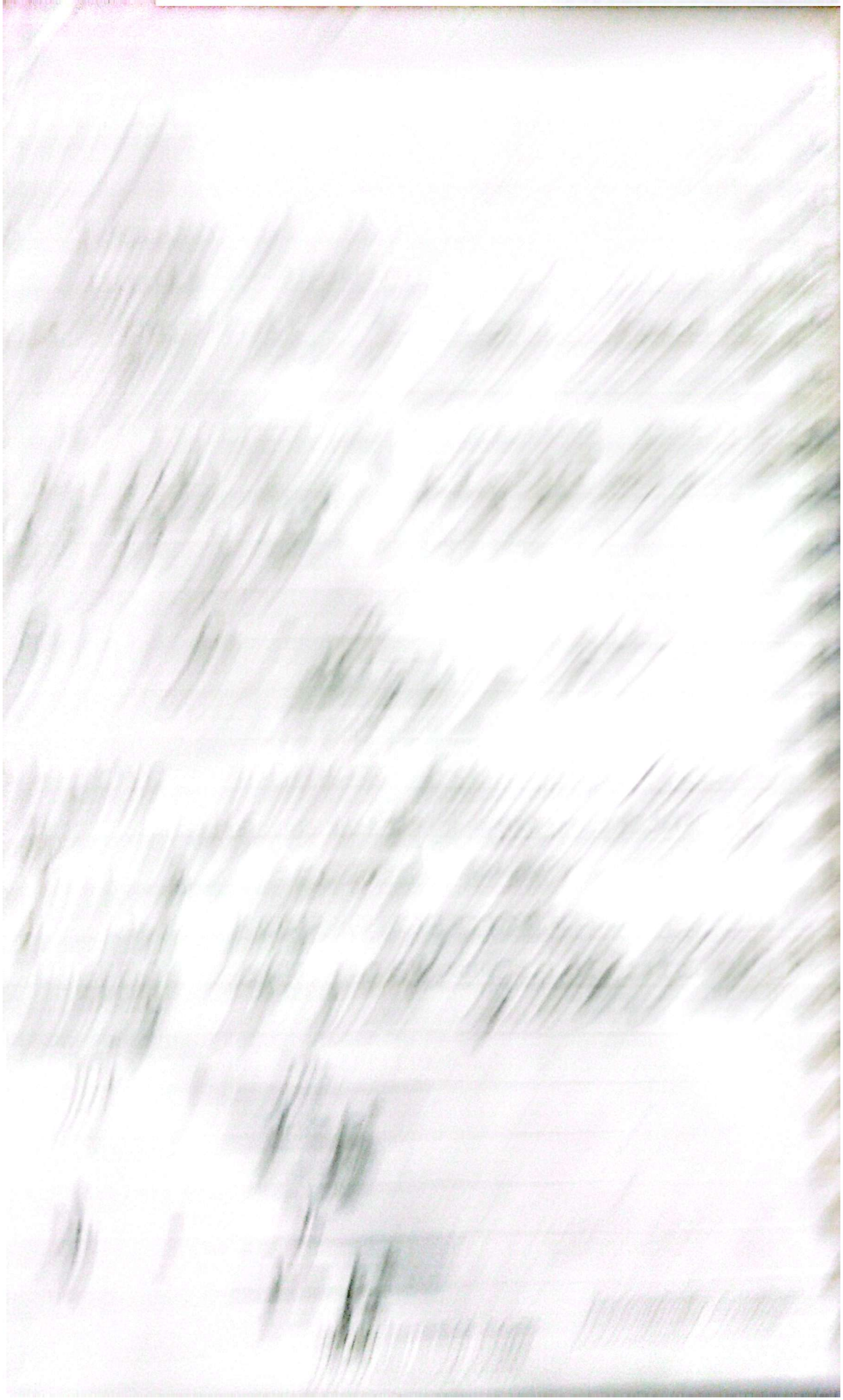
6) Adagrad
changes learning rate at every step(t). second order algorithm

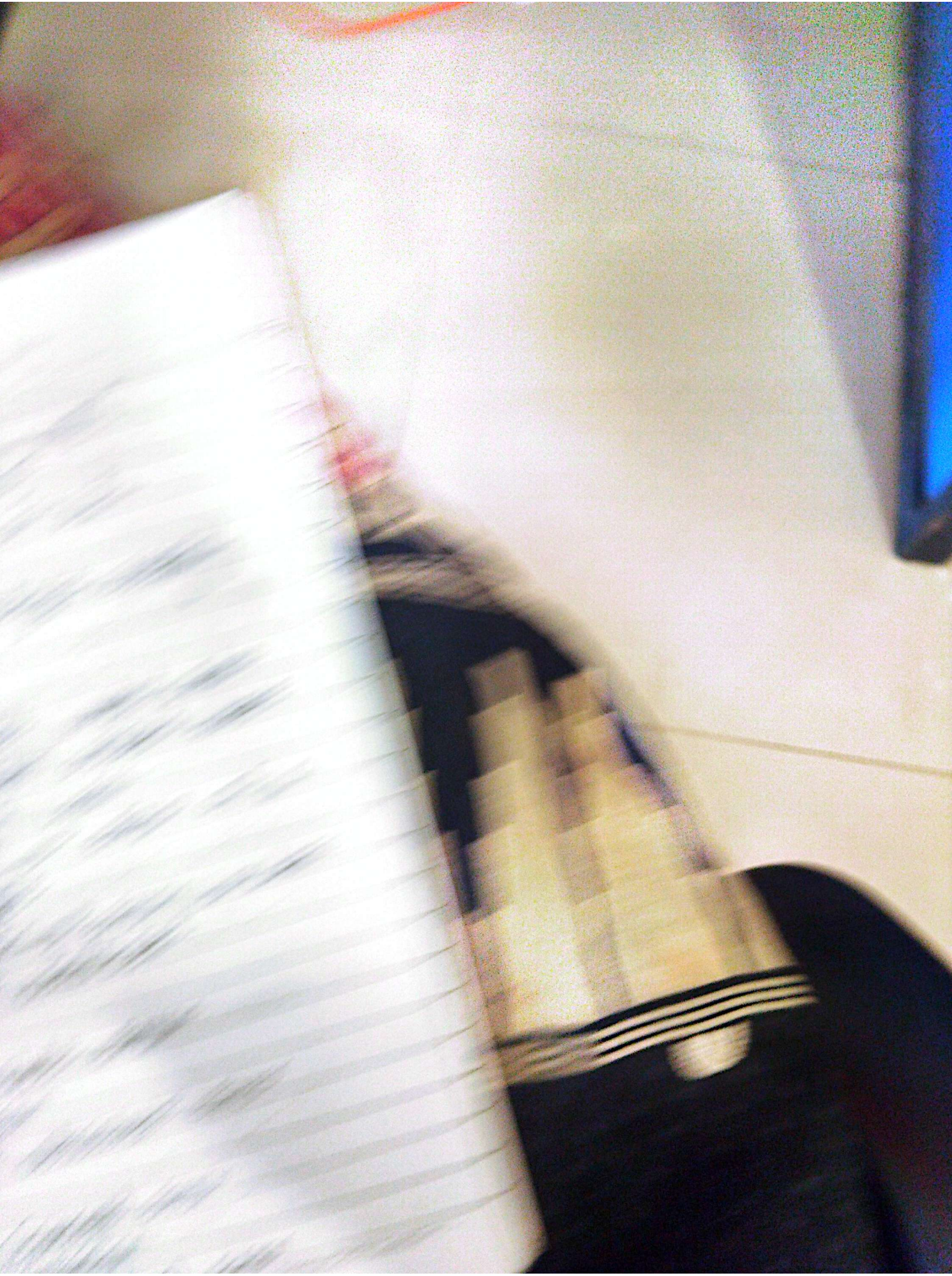
$$g_{t,i} = \nabla_\theta J(\theta_t, i)$$

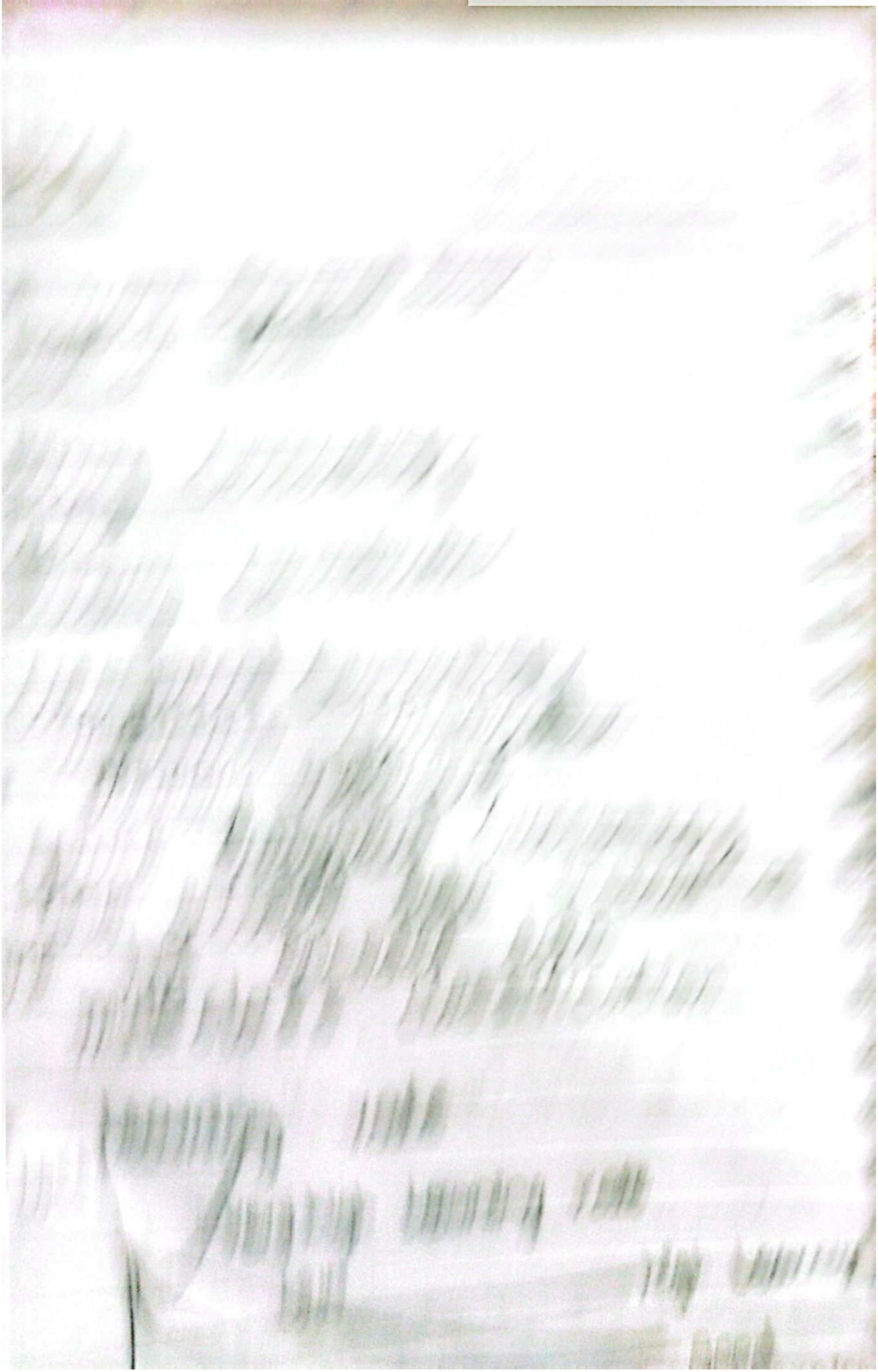
$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

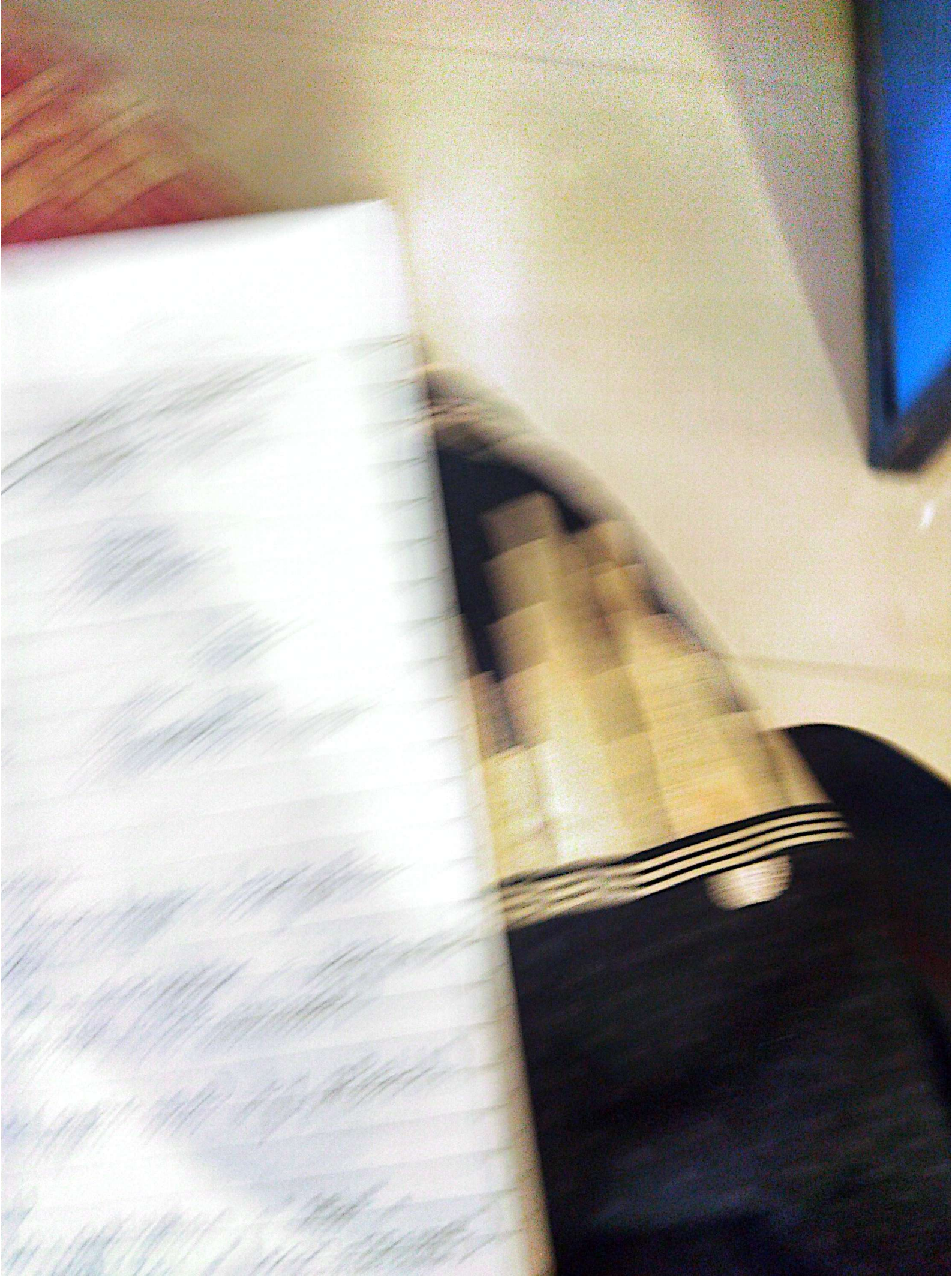
sum of gradient squares

7) AdaDelta
Remove the decaying Learning









→ Batch Normalization:

keras.layers.BatchNormalization()

x = tf.range(value)

dataset = tf.data.Dataset.from_tensor_slices(x)

↳ from slices from values

dta = dataset.repeat(3).batch(7)

dataset = dataset.map(lambda x:
x*2)

data set

dataset = dataset.apply(tf.data.
experimental.unbatch())

↳ unbatch

dataset = dataset.filter(lambda x:
x < 10)

↳ filter

for item in dataset.take(3):
↳ Take first 3

→ Computer vision in deep learning flatten depends upon number of features.

Basic sharpening

0	-1	0
-1	5	-1
0	-1	0

Edge:

0	1	0
1	-4	1
0	1	0

Strongest Edge:

-1	-2	-1
0	0	0
1	2	1

More sharpening

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 10 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

→ Cropping

```
plt.imshow(image[100:150, 25:75, cmap
    = ' ')
```

→ skimage import data, io

3 channel have [x, y, 3]

else [x, y]

data.coins()

data.coffee()

→ from scipy.signal import convolve2d

```
kernel = np.array([[-1, -1, -1, -1], [1
        , 1, 1, 1], [1, 1,
    1, 1], [-1, -1, -1,
    -1]])
```

first & last row -1 for horizont

convolve2d(image, kernel, mode = 'valid')

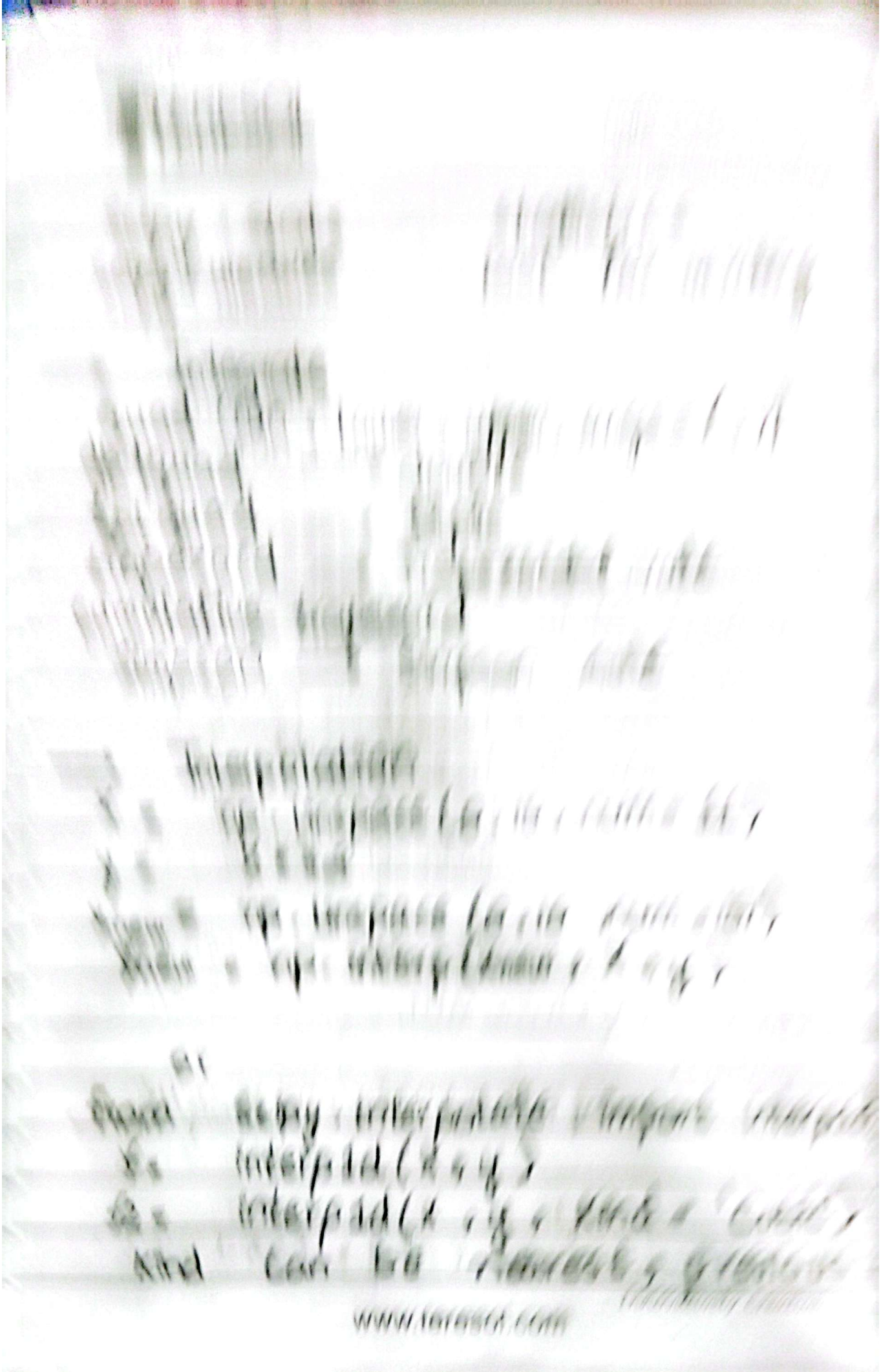
→ Convolution layer

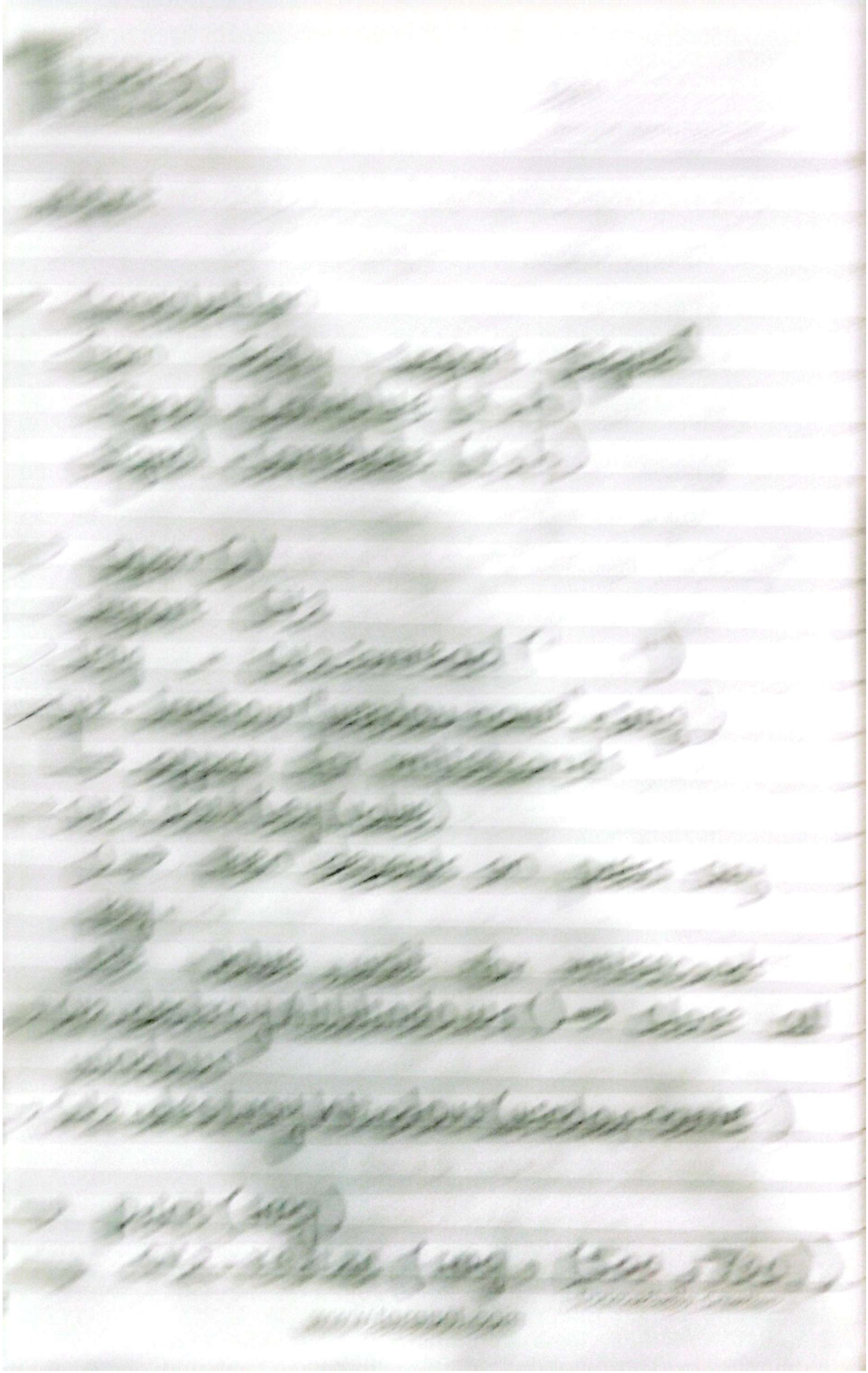
→ output width & height

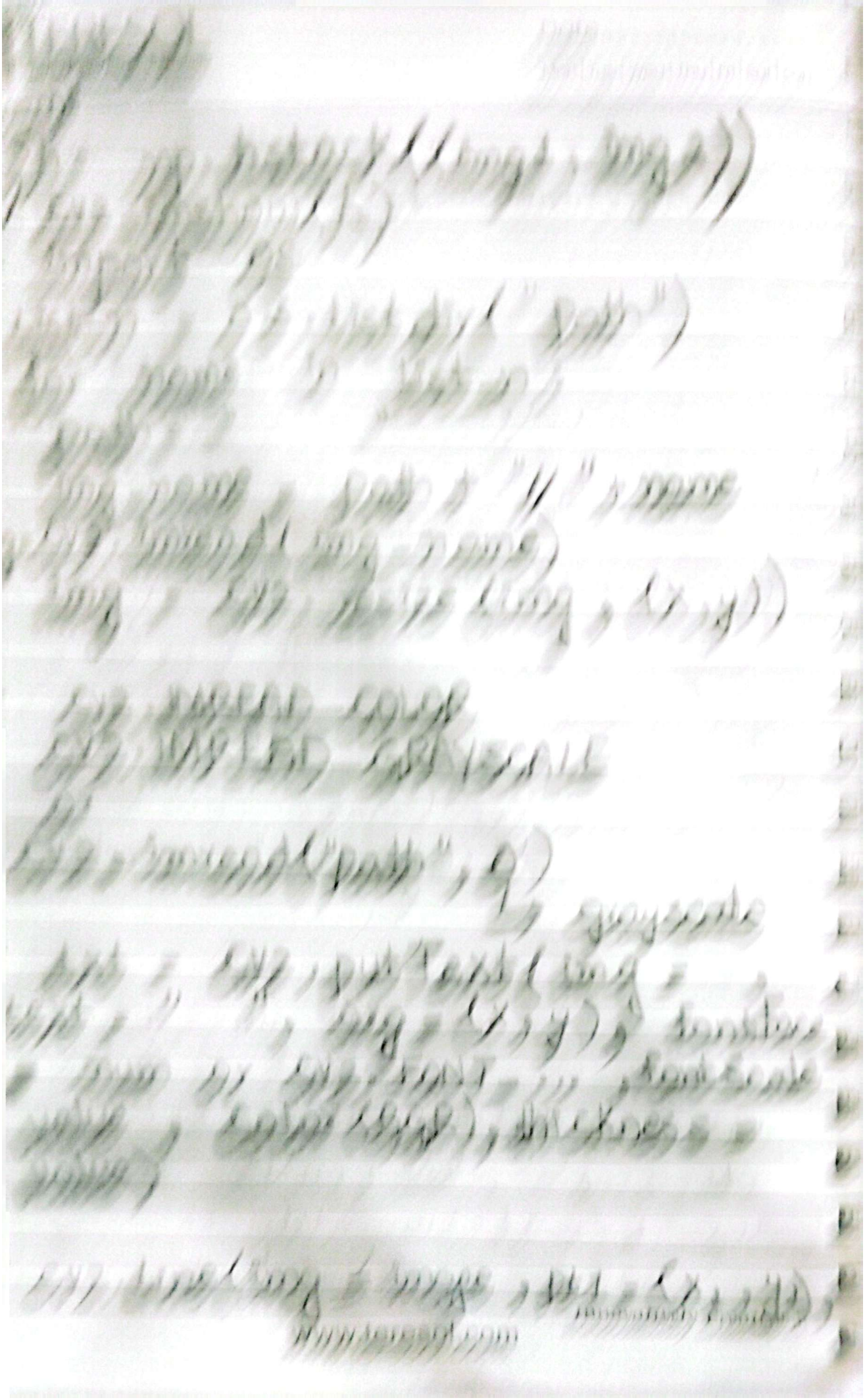
$$O.W = \frac{W - F_W + 2P}{S_W} + 1$$

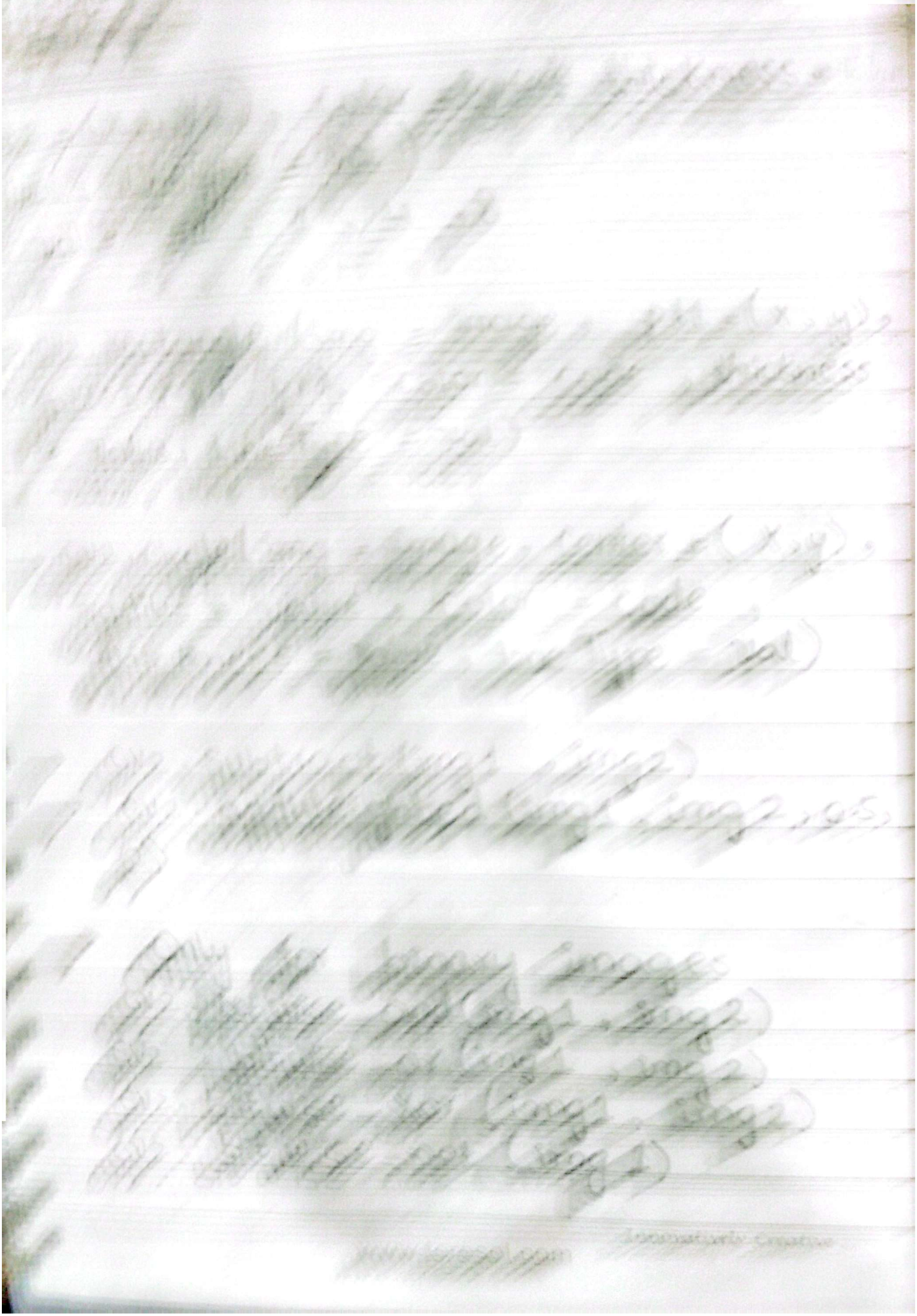
→ scipy subpackage

scipy.cluster	vector quantization / k-means
scipy.constants	Physical & Maths Constant
scipy.fftpack	fourier transform
scipy.integrate	Integration
scipy.linalg	Linear algebra
scipy.io	Input & output
scipy.ndimage	n-dim image
scipy.interpolation	interpolation
scipy.odr	orthogonal distance re
scipy.optimize	optimization
scipy.signal	Signal processing
scipy.sparse	sparse matrices
scipy.spatial	Spatial dsA
scipy.special	special function









- Edge detection in OpenCV
  cv2.Canny(image, threshold1, threshold2)
  ↓ Increase to reduce details.

- LeNet-5

Layer	Type	Maps	Size	Kernel	Stride	Activ.
7	FC	-	10	-	-	RBF
6	FC	-	84	-	-	tanh
5	Conv	120	1×1	5×5	1	"
4	Avg pool	16	5×5	2×2	2	"
3	Conv	16	10×10	5×5	1	"
2	Avg pool	6	14×14	2×2	2	"
1	Conv	6	28×28	5×5	1	"
In	Inp	1	32×32	-	-	-

→ Create filters

```
filters = np.zeros(shape = (7,7, channels, 2), dtype = np.float32)
filters[:, 3, :, 0] = 1  Vertical
filters[3, :, :, 1] = 1  Horizontal
```

out = tf.nn.conv2d(images, filters, strides = 1, padding = "SAME")

or

conv = keras.layers.Conv2D(filters = 32, kernel_size = 3, strides = 1, padding = "SAME", activation = "relu")

→ Max-pooling

maxpool = keras.layers.MaxPool2D(pool_size = 2)

or

out = tf.nn.max_pool(images, ksize = (1, 1, 1, 3), strides = (1, 1, 1, 3), padding = 'valid")

Layer	Type	map	size	Kernel	stride	Pad	Act
out	FC	—	1000	—	—	—	SM
9	FC	—	4096	—	—	—	ReLU
8	FC	—	4096	—	—	—	Relu
7	Conv	256	13x13	3x3	1	same	Relu
6	Conv	384	13x13	3x3	1	//	ReLU
5	Conv	384	13x13	3x3	1	//	ReLU
1	Max pool	256	13x13	3x3	2	Valid	—

3	Conv	256	27x27	5x5	1	Same Pad
2	Max pool	96	27x27	3x3	2	Valid
1	Conv	96	55x55	11x11	4	Valid Pad
0	Input	3 (RGB)	227x227	-	-	-

→ Open CV

→ rotating an image

```
w,h = img.shape[0], img.shape[1]
m = cv2.getRotationMatrix2D((w/2, h/2), angle, scale)
                                          ↓ value
new_img = cv2.warpAffine(img, m, (h, w))
```

→ Image blurring

1) Gaussian blurring
   ↳ use a kernel matrix through Gaussian function
   Soft blurring

```
g = cv2.GaussianBlur(img, kernel size, Activation function)
                     (3,3)
```

Date

Mon | Tue | Wed | Thu | Fri | Sat | Sun

Median Blur

cv2.medianBlur(img, kernel size)

↳ only 3 or 5

Bilateral Blur

cv2.bilateralFilter(img, matrix, smooth factor, smooth img)

cv2.bilateralFilter(img, 9, 75, 75)

Saving into folder

cv2.imwrite("path // name.jpg", img)

play a video

cap = cv2.VideoCapture(r"path")

check while video is open

while cap.isOpened():

ret, frame = cap.read()

if ret == True:

cv2.imshow("name", frame)

if cv2.waitKey(1) & 0xFF ==

ord('q'):

break

→ Image pyramid
↳ shrink $\frac{1}{2}$ for original pic

```
new = cv2.pyrDown(img)

new1 = cv2.pyrUp(new)
```

↳ increase by 2 every provided pic

→ Image translation
↳ Hide a part of image
↳ Crop
↳ Shift

Generate mask

1) Cropping an image

```
m = np.float32([[1, 0, 100],
[0, 1, 50]])

new = cv2.warpAffine(img, m,
(500, 500))
```

↳ size of new image

COLOR_BGR2HSV)

→ Crop images:

crop = img[x1:x2, y1:y2]

→ Create blank images

new_img = np.ones((500, 500, 3), np.uint8) * 255
cv2.imshow("name", new_img)

→ Region of interest

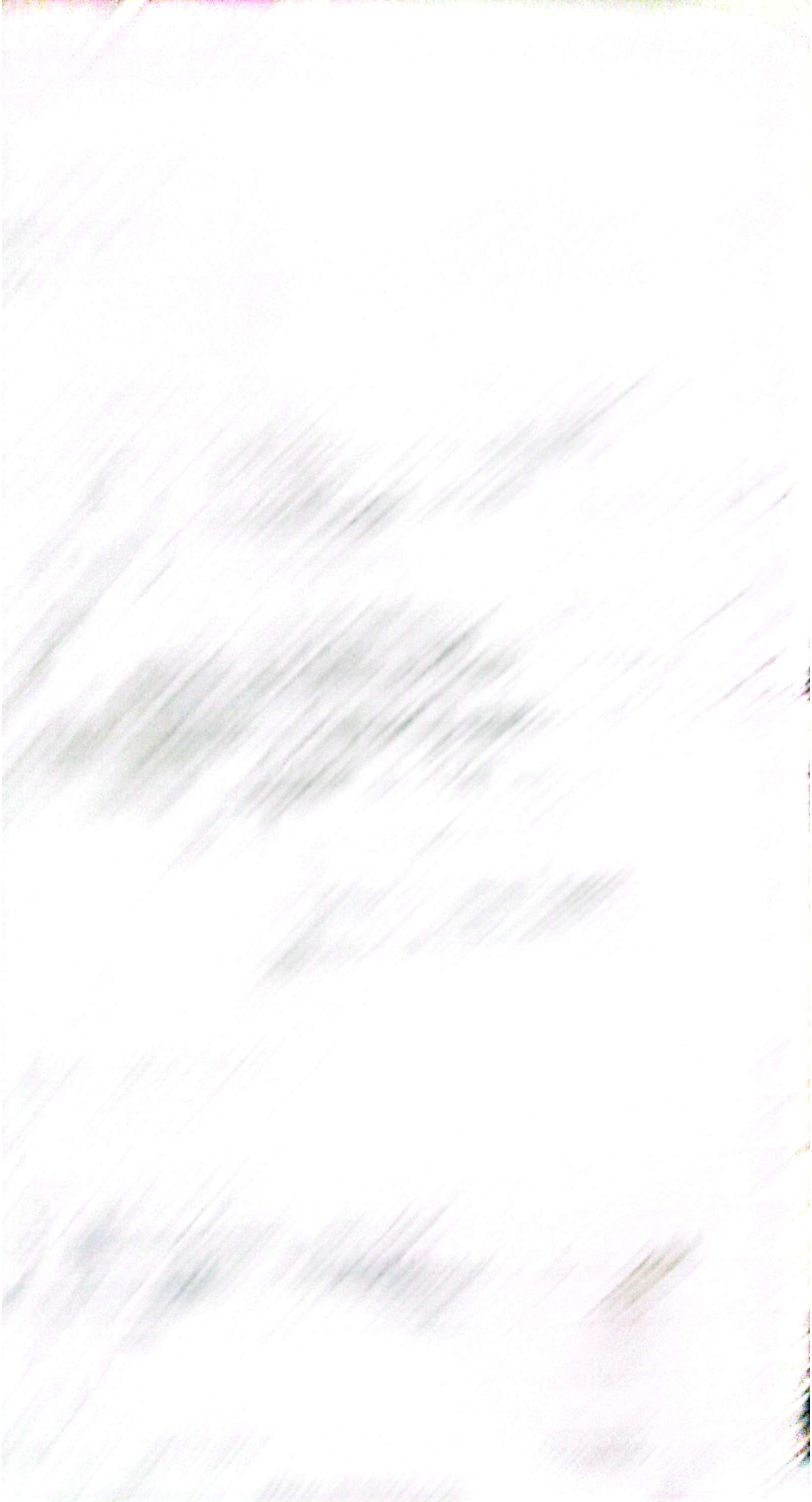
image[x1:x2, y1:y2] = crop

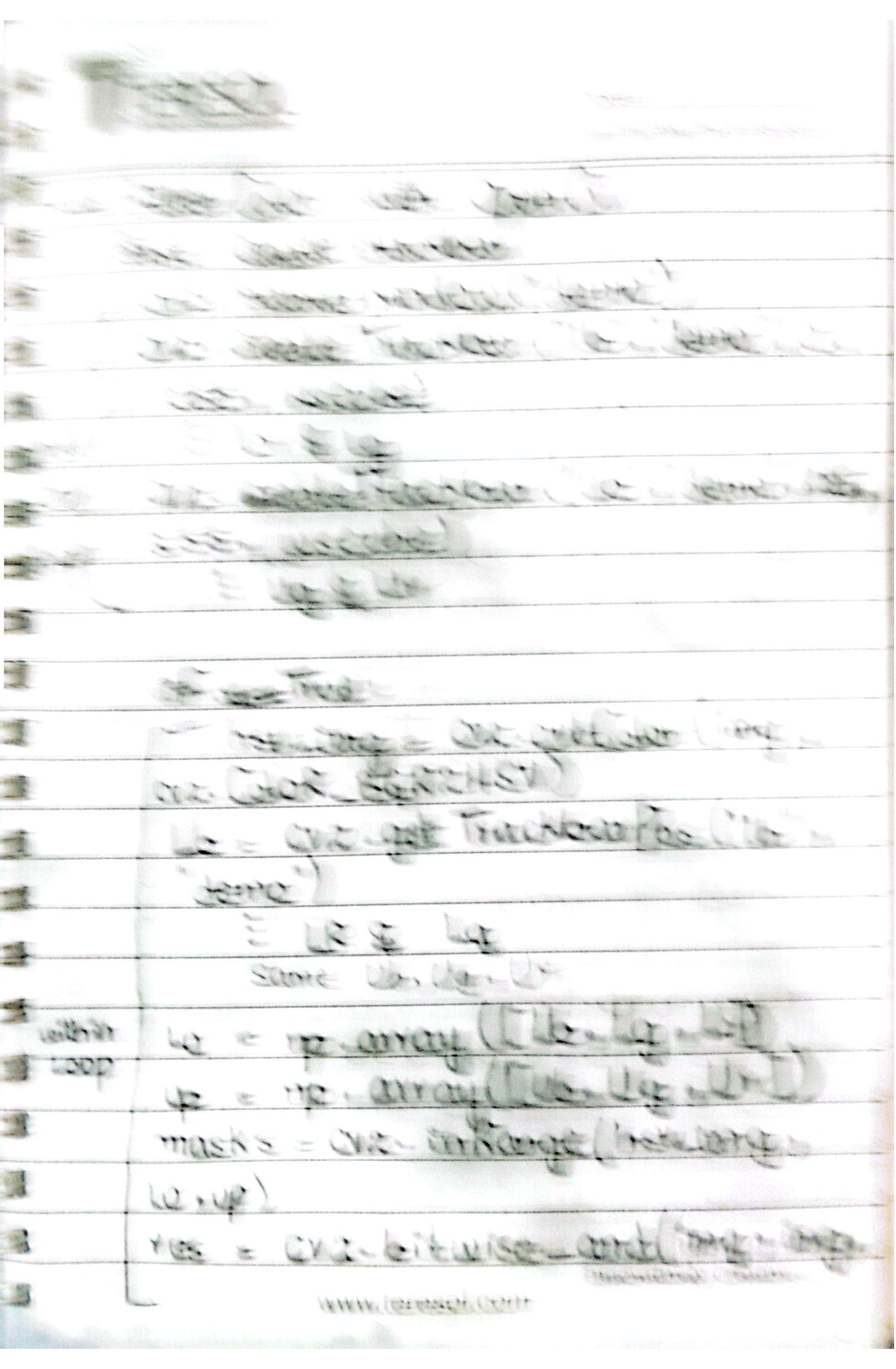
→ flip

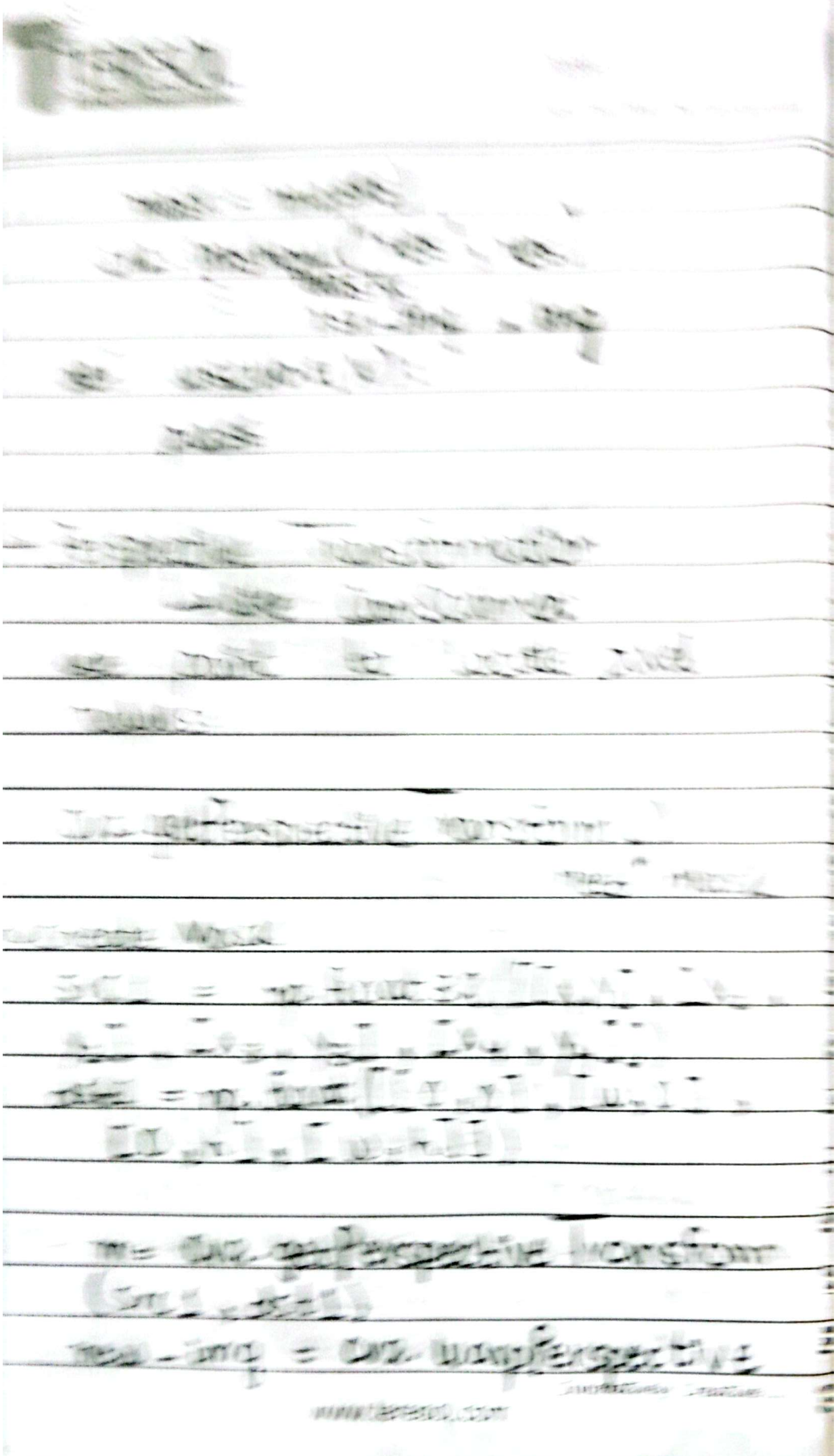
image = cv2.flip(img, 0)
↳ x-axis
1 for y-axis
-1 for x & y both axis

→ rotate

img = cv2.rotate(img, cv2.ROTATE_







Date

(img, m, (w, h))
↳ img w & h

h, w, = img.shape

→ Threshold → object removal

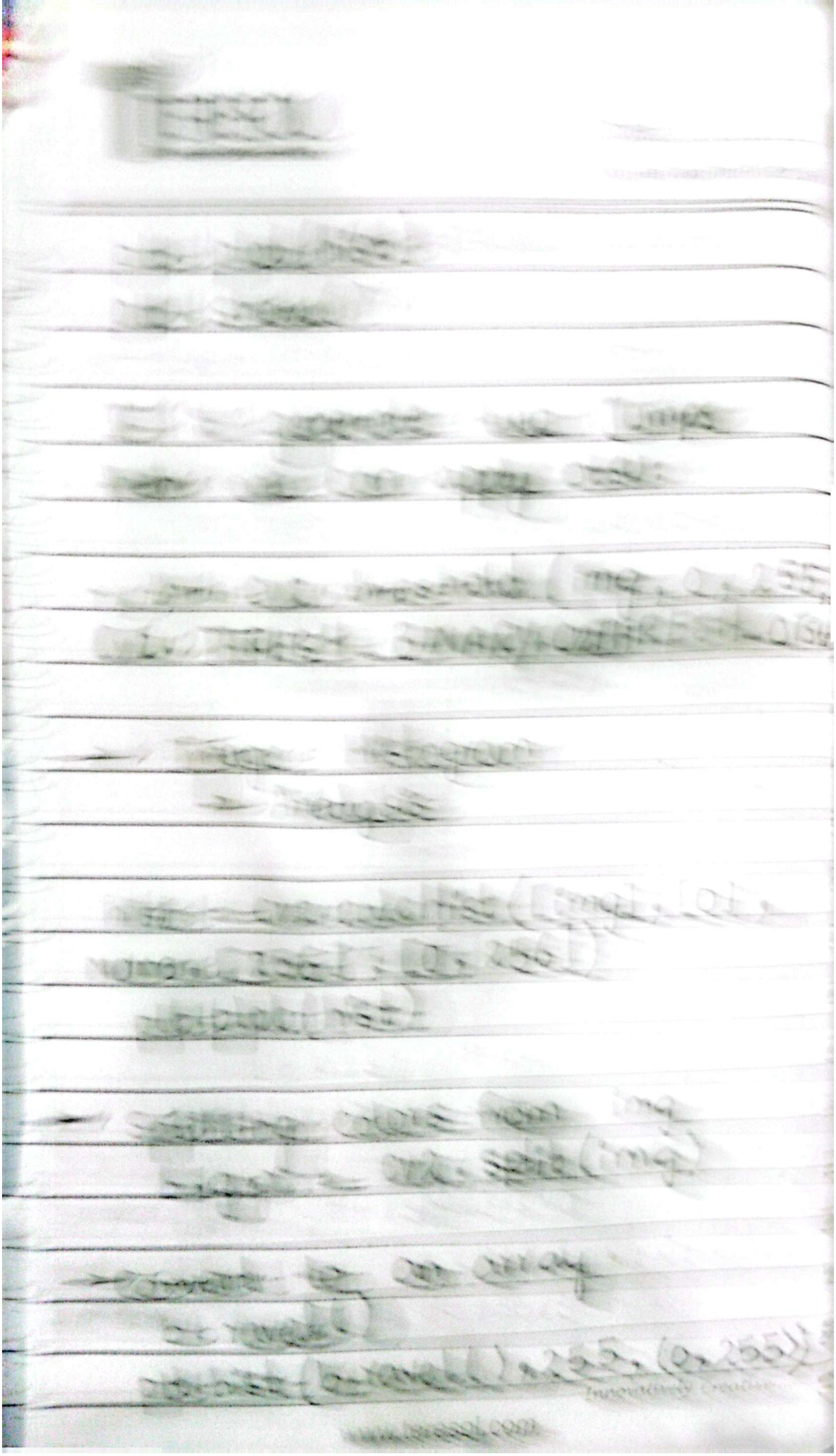
r, th = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY)
r → binary; th → Thresh; 0 → increase

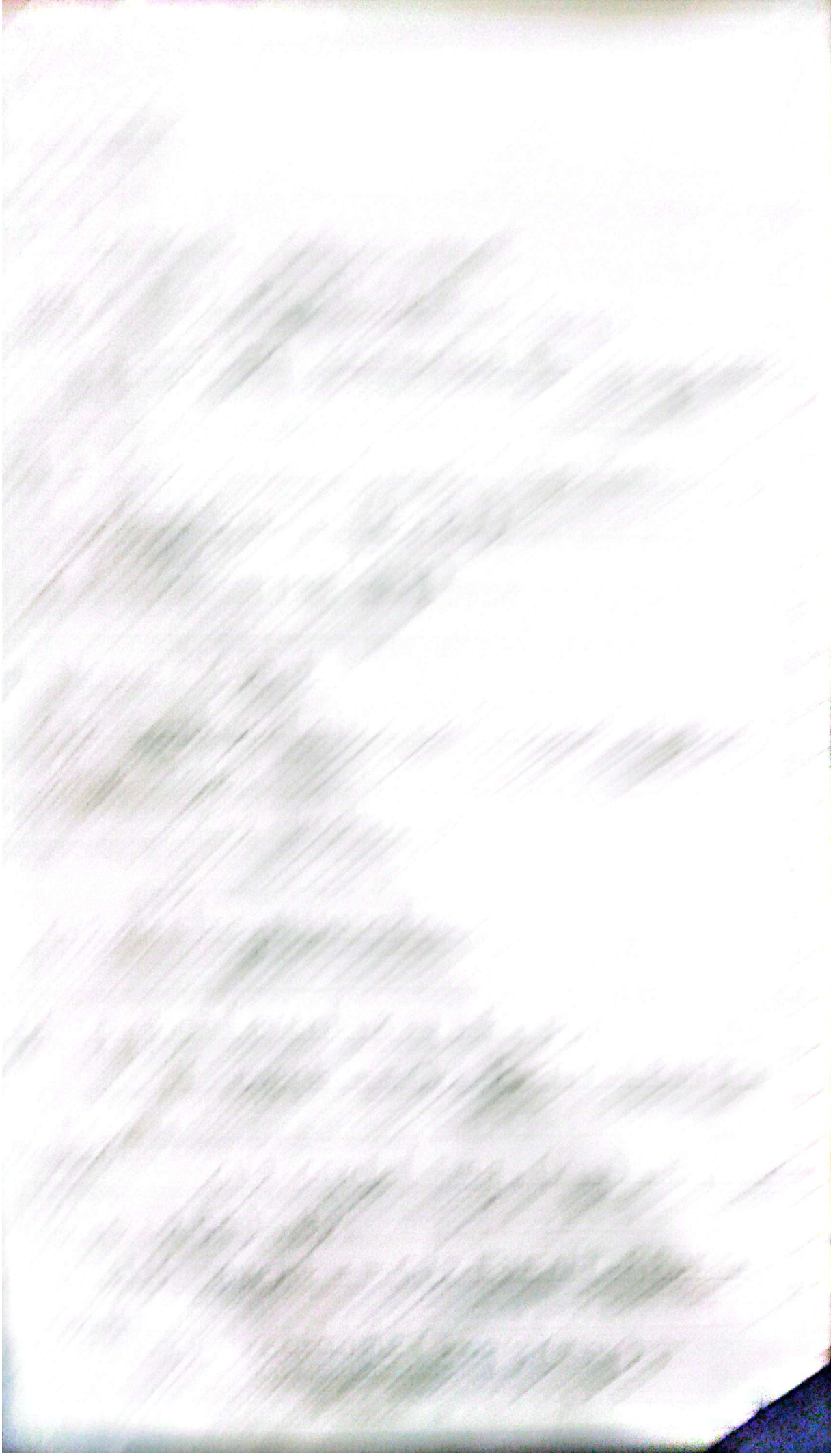
THRESH_BINARY_INV
THRESH_MASK
THRESH_TRIANGLE
THRESH_TOZERO
THRESH_TOZERO_INV

→ OTSU Threshold

img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
color Channel

hist = cv2.calcHist([img], [0], None, [255], [0, 255])
None → Masking
[255] → Size of histogram





(x + 20, y + 20), (255, 0, 0), 4)

→ Image Contours
first perform thresh as Contour can only be performed on Gray

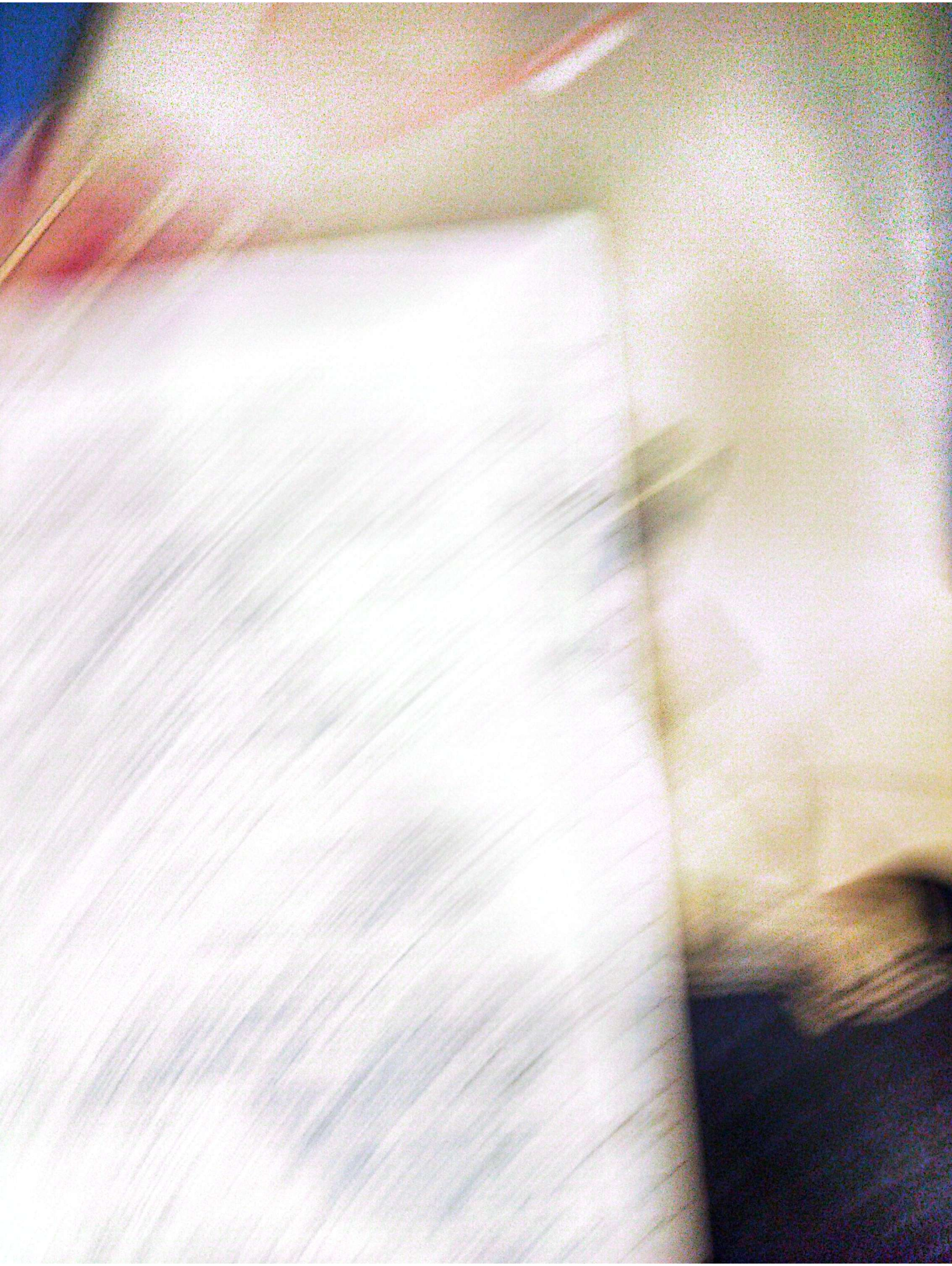
```
c, h = cv2.findContours(thr, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
new-img = cv2.drawContours(img, c, -1, (255, 0, 0), 2)
```

-1 → contour index, -1 for all
(255, 0, 0) → BGR color
2 → Thickness

→ Area & points of Contour

```
ar = []
for Ci in C:
    m = cv2.moments(Ci)
    x = int(m["m10"]/ m["m00"])
    y = int(m["m01"]/ m["m00"])
    cv2.Circle(img, (x, y), 2, (255, 0, 0), -1)
    a = cv2.contourArea(ci)
```

Centre of Contour (x, y)



```
_, thr = cv2.threshold(mask,
  200, 255, cv2.THRESH_BINARY)
mask = cv2.merge((mask, mask,
  mask))
yes = cv2.bitwise_or(img, mask)
```

→ Line eq: $r\cos\theta + r\sin\theta = y$

→ Hough Transformation Lines

Take Gray pic

```
edg = cv2.Canny(gry, 20, 250)
```

↳ edge detection

```
Lines = cv2.HoughLines(edg, 1, np.pi/180,
  200)
```

→ Thresholding

↳ Design Lines

Design θ:

```
np.pi

for r, th in Lines[0]:
    a = np.cos(th)
    b = np.sin(th)
    x0 = a*r
    y0 = b*r
    x1 = int(x0 + 1000*(-b))
```

```
_, thr = cv2.threshold(mask,
  200, 255, cv2.THRESH_BINARY)
mask = cv2.merge((mask, mask,
  mask))
yes = cv2.bitwise_or(img, mask)
```

→ Line eq: $r\cos\theta + r\sin\theta = y$

→ Hough Transformation Lines

Take Gray pic

```
edg = cv2.Canny(gry, 20, 250)
```

↳ edge detection

```
Lines = cv2.HoughLines(edg, 1, np.pi/180,
  200)
```

↳ Design Lines → Thresholding

Design θ:

```
np.pi

for r, th in Lines[0]:
  a = np.cos(th)
  b = np.sin(th)
  x0 = a*r
  y0 = b*r
  x1 = int(x0 + 1000*(-b))
```

```
y1 = int(y0 + 1000*(a))
x2 = int(x0 - 1000*(-b))
y2 = int(y0 - 1000*(a))
   cv2.line(img,(x1,y1),(x2,y2),
   (0,255,0),2)
```

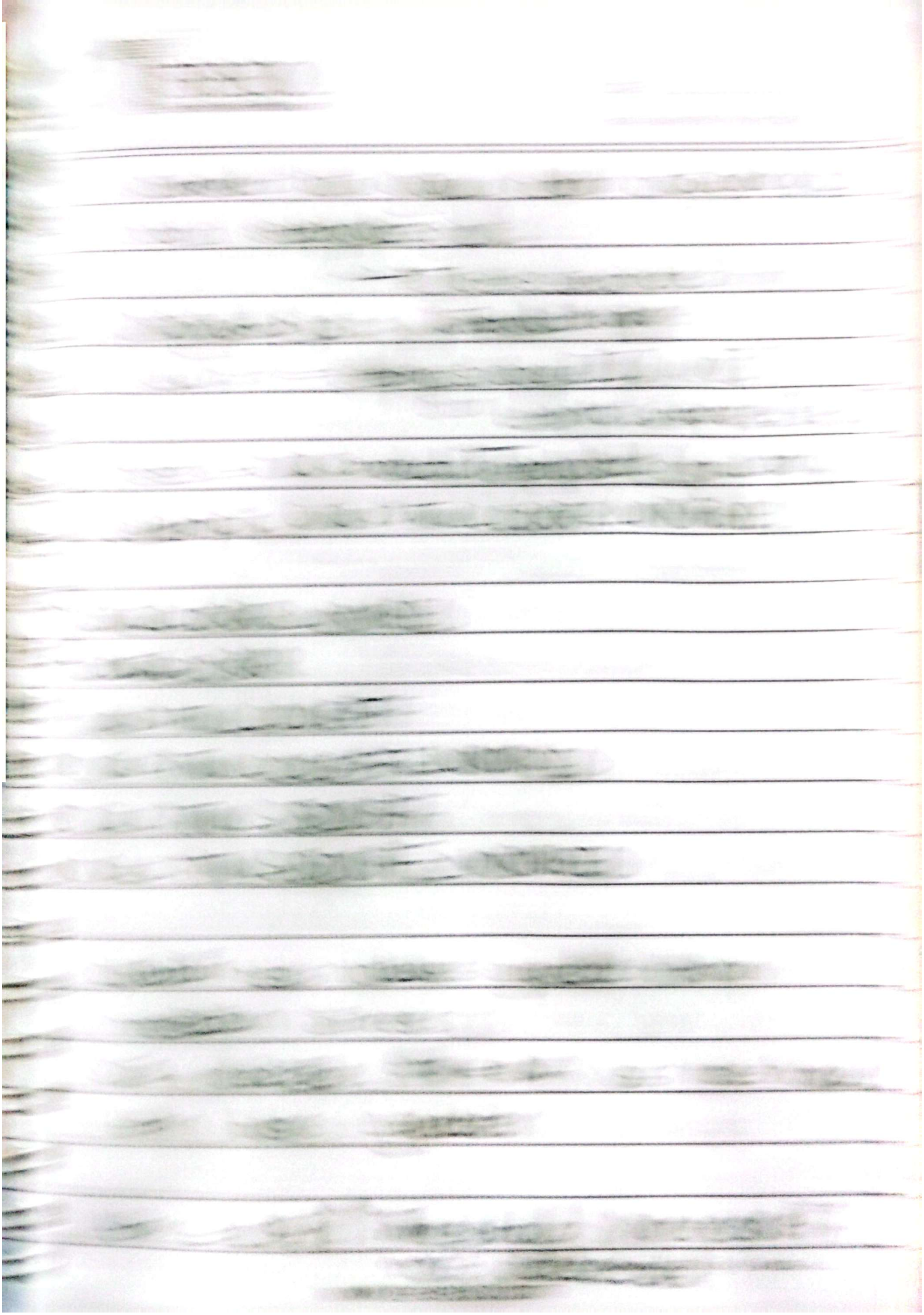
↓ Longest procedure
  ↳ change threshold continuously

→ HoughLinesP

```
L = cv2.HoughLinesP(edg, 1,
np.pi/180, 200, minLineLength =
120, maxLineGap = 100)

for l in L:
    x1,y1,x2,y2 = l[0]
    cv2.line(img,(x1,y1),
(x2,y2),(0,255,0),2)
```

→ Template matching



l = np.where (res >= thr)
→ two separate objects so we zip
for i in zip(*l[::-1])
cv2.rectangle(img, i, (i[0]+w, i[1]+h), (0,0,255), 0)
~ loop end ~
img = cv2.resize(img, (600, 500))

→ Hough Circle detection
↳ Convert image to grayscale
gr = cv2.medianBlur(img, 7)
↳ If Circle have distortion and are unable to be detected
c = cv2.HoughCircles(gr, cv2.HOUGH_GRADIENT, 1, 20, Param1 = 50,
Min ↓ Max gradient point
param2 = 30, minRadius = 0,
maxRadius = 0)
data = np.unit16(np.around(c))
↳ round off
for (x, y, r) in data[0,:]:
cv2.Circle (img, (x, y), r, (0,

0, 255), 4)

→ Grabcut Algorithm for background Image

efficient background removal

Two: Background mask
foreground mask
Overall mask

```
mask = np.zeros(img.shape[:2],
np.uint16)
bgmask = np.zeros((1,65), np.float
    64)*255
fgmask = np.zeros((1,65), np.float
64)*255
r = [x1, y1, x2, y2]  → rect over object from Paint
cv2.grabCut(img, mask, r, bgmask,
fgmask, iterCount = 10, cv2.GC_INIT_
WITH_RECT)

mask2 = np.where((mask==2)|(
 mask==0), 0, 1).astype("uint8")
img = img * mask2[:,:,np.newaxis]
```

→ Video Background Removal:
removal require two algorithm

```
algo1 = cv2.createBackgroundSubtractorKNN(detectShadows = True)
algo2 = cv2.createBackgroundSubtractorMOG(detectShadows = True)

In if y = True: Type:
y1 = algo1.apply(frame)
y2 = algo2.apply(frame)
```

→ Object Tracking & Detection

```
y, f = cap.read()
x, y, w, h = from point ~
t = (x, y, w, h)
roi = f[y:y+h, x:x+w]
hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_roi, np.array((0.0, 60., 32.)), np.array((180., 255., 255.))
```

```
roi_hist = cv2.calcHist([hsv_roi], [0],
mask, [180], [0, 180])
cv2.normalize(roi_hist, roi_hist, 0,
255, cv2.NORM_MINMAX)
tr = (cv2.TERM_CRITERIA_EPS | cv2.
TERM_CRITERIA_COUNT, 10, 1)
```

~ Inside Loop `if r == True:` frame

```
hsv_f = cv2.cvtColor(f, cv2..)
d = cv2.calcBackProject(
[hsv_f], [0], roi_hst, [0, 180], 1)
r, tp = cv2.meanShift(d, t, tr)
x, y, w, h = tp
final = cv2.rectangle(frame,
(x, y), (x+w, y+h), (0, 0, 255),
4)
```

use camShift

→ Corner detection (object + Human)

Convert image to gray

& Harris can only be applied to float ... convert img to float

```
img = np.float32(img)
```

Increase Corner points (→ 2)

res = cv2.cornerHarris(img, 2, 3, 0.04)

(2) ↓ kernel size to be multiplied by image

res = cv2.dilate(res, None)

img[res > 0.01 * res.max()] = [0,0,255]

(img ↓) color-img

→ Shi-Tomasi Corner detection

↓ convert to gray

cr = cv2.goodFeaturesToTrack(gray, 10, 0.01, 20)

10 ↓ Max Corner points

0.01 ↓ quality level

20 ⟶ Min distance between points

It need int data

so,

cr = np.int64(cr)

for i in cr:

x,y = i.ravel() ⟶ Two → one dim data

cv2.circle(img, (x,y), 2, (0, 0, 255), -1)

→ Face detection

Convert to gray

f = cv2.CascadeClassifier("haar cascade file ...")

↳ python > Lib > site-packages > cv2 > data

standard ←

d = f.detectMultiScale(gray img, 1.3, 2)

↳ gap

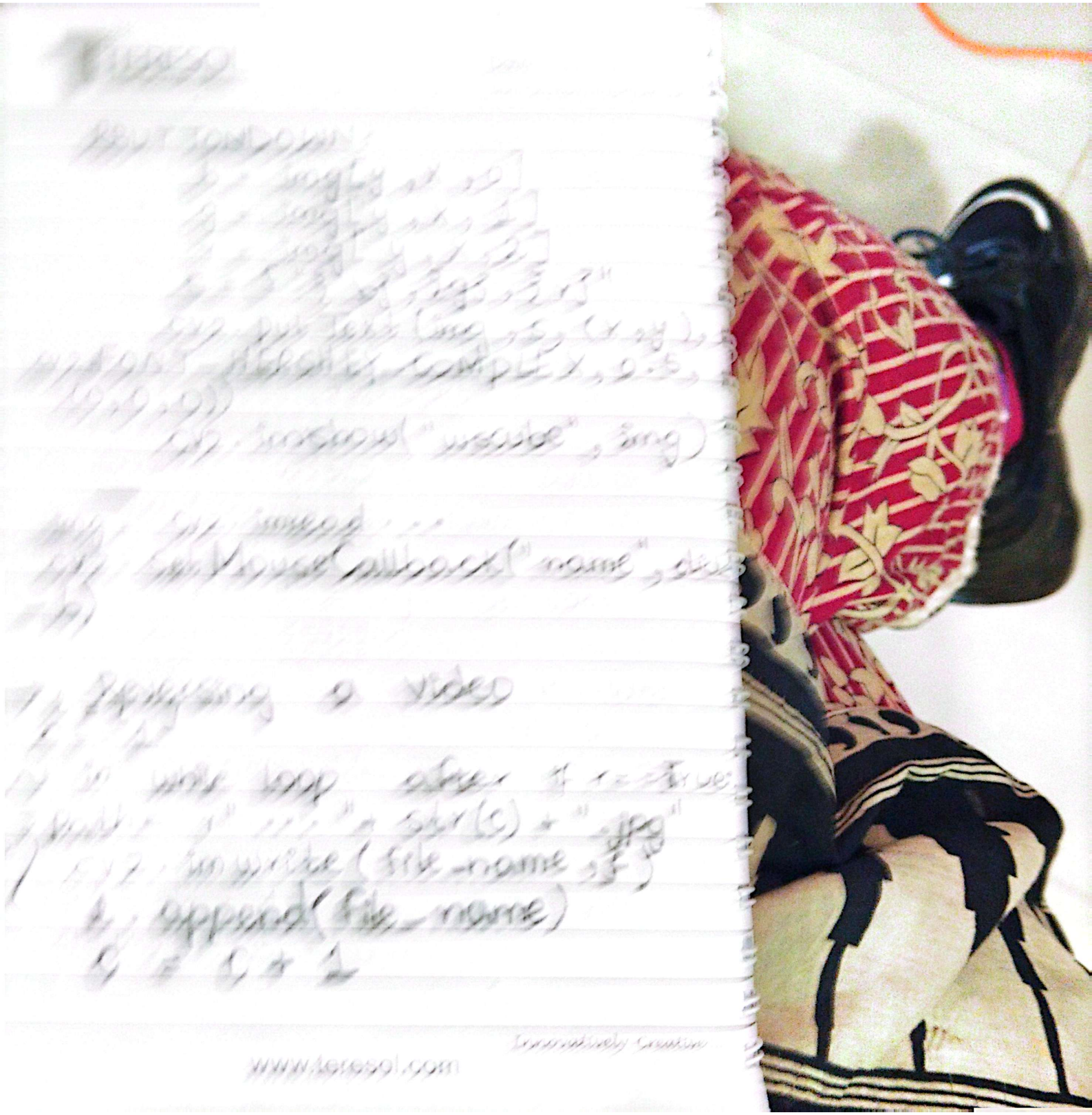
↓ increase but depend on face

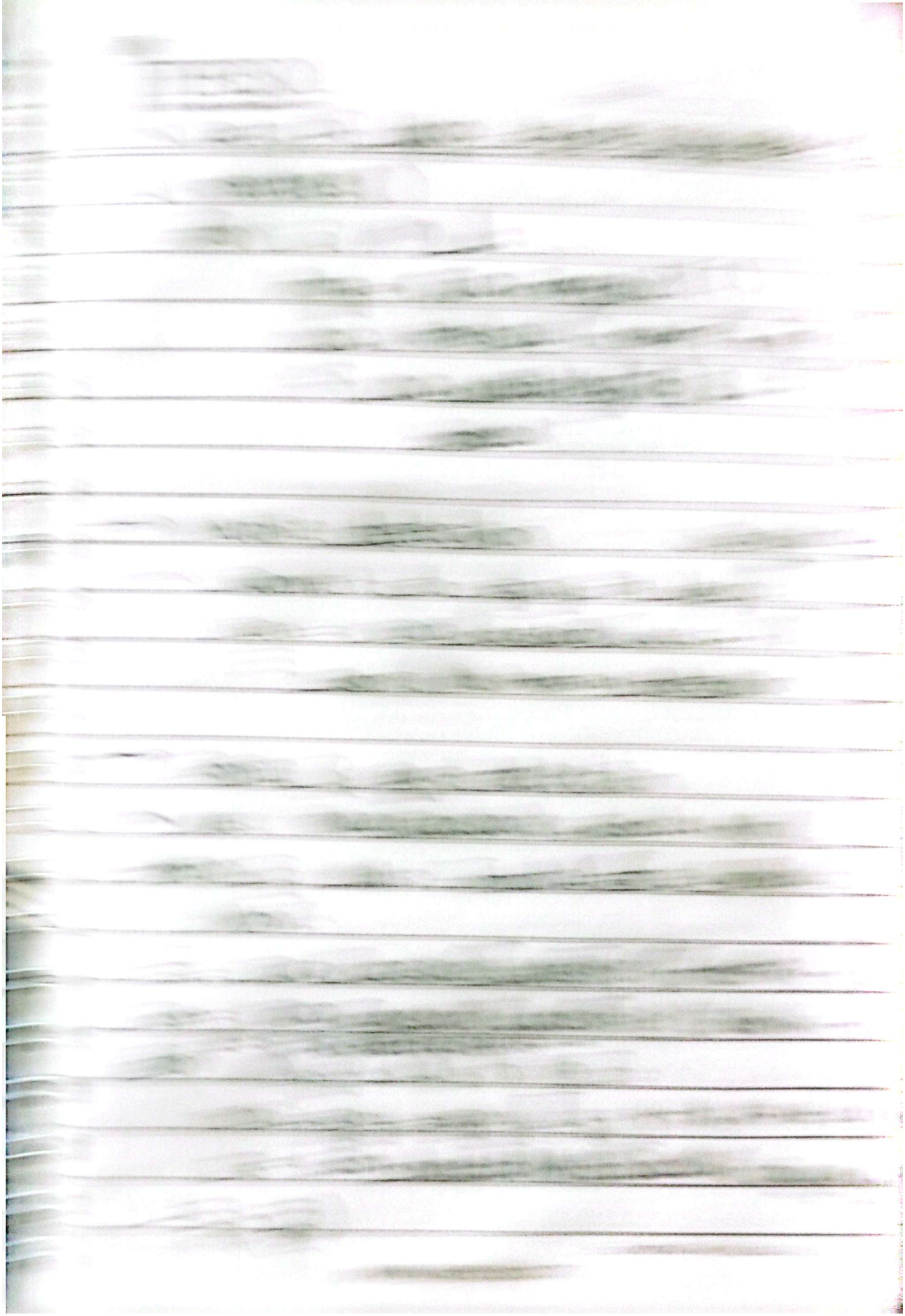
for (x, y, w, h) in d:

cv2.rectangle(img, (x, y), (x+w), (y+h), (0, 0, 255), 3, 3)

→ On-mouse click display color

```
def click_b(event, x, y, f, p):
    if event == cv2.EVENT_LBUTTONDOWN:
        s = f"{x}, {y}"
        cv2.putText(img, s, (x, y), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 0))
        cv2.imshow("img", img)
    elif event == cv2.EVENT_
```





built in for loop

```
for xs, ys, ws, hs in s:
    cv2.rectangle(f, (xs, ys), (xs+ws, ys+hs), (0, 255, 0), 2)
```

* Media pipe
  - Different body parts
  - Documentation

```
import mediapipe as mp
mp_face = mp.solutions.face_detection
mp_draw = mp.solutions.drawing_utils
face_det = mp_face.FaceDetection(min_detection_confidence=0.8, model_selection=0)
```

* with in loop

```
f = cv2.cvtColor(f, cv2.COLOR_BGR2RGB)
res = face_det.process(f)
```

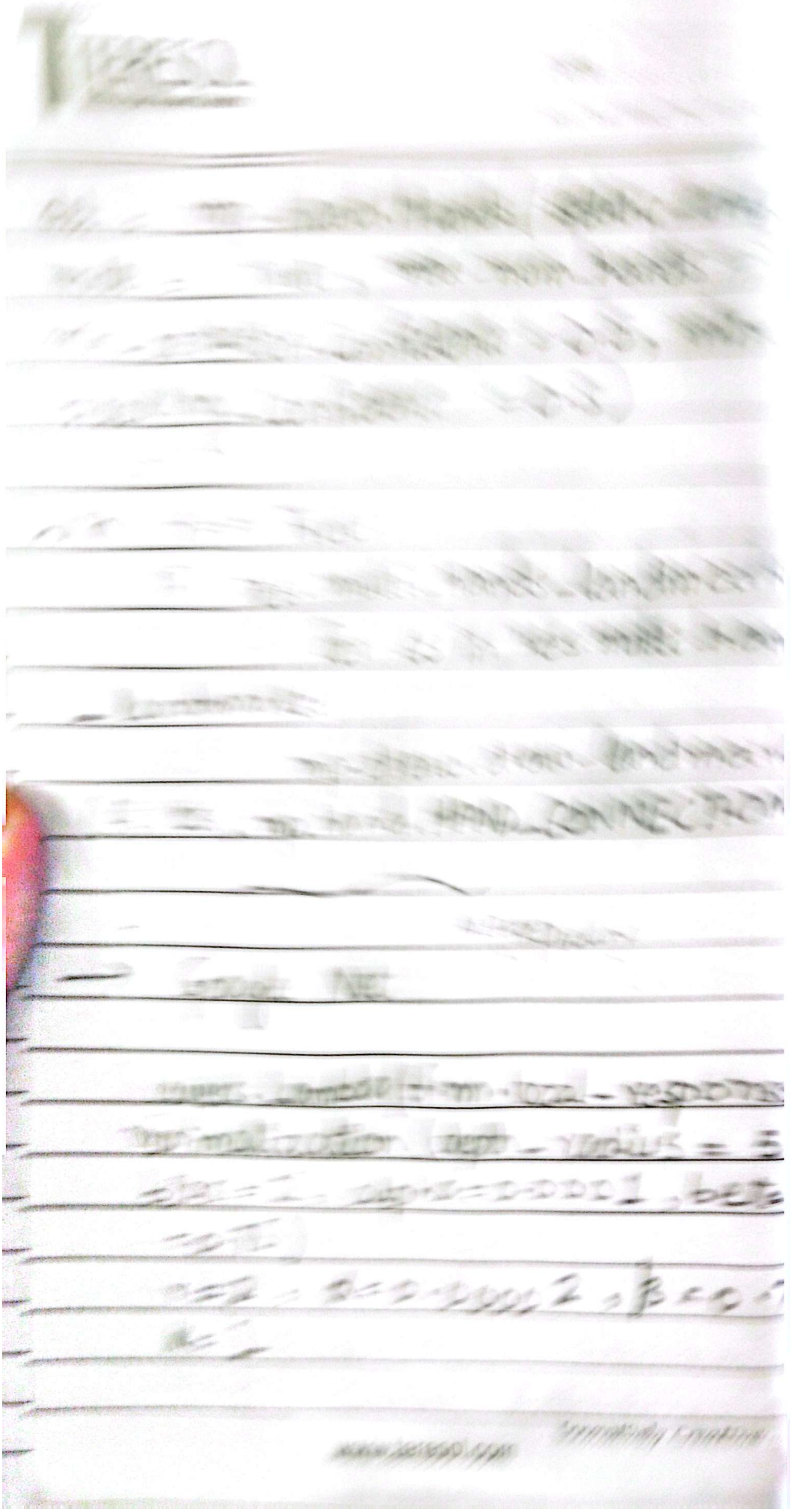
f = cv2.cvtColor(f, cv2.COLOR_RGB2BGR)

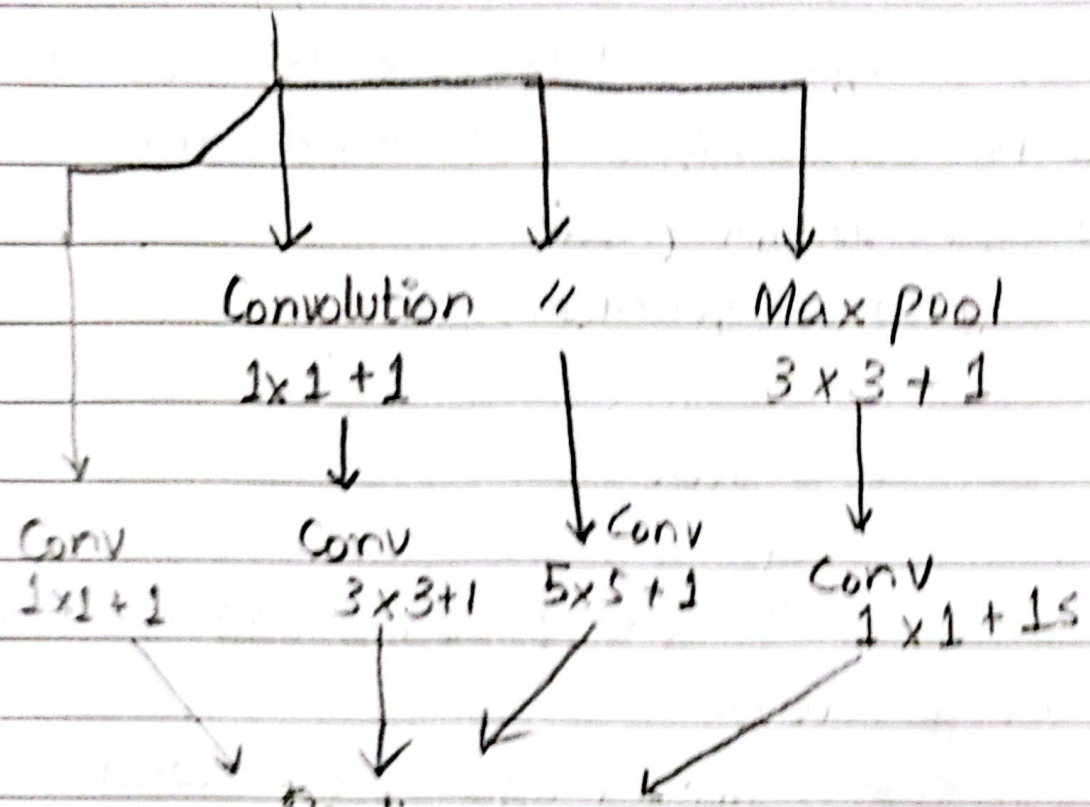
~ if res.detections:
    for cr in res.detections:
        mp_draw.draw_detection(f, cr)

→ Hand Tracking

0. WRIST	11. " - DIP
1. THUMB - CMC	12. " - TIP
2. THUMB - MCP	13. RING " - MCP
3. THUMB - IP	14. " - PIP
4. THUMB - TIP	15. " - DIP
5. INDEX - FINGER - MCP	16. " - TIP
6. " - PIP	17. PINKY - PMCP
7. " - DIP	18. " - PIP
8. " - TIP	19. " - DIP
9. MIDDLE - " - MCP	20. " - TIP
10. " - PIP	

mp_hand = mp.solutions.hands
mp_draw = ...





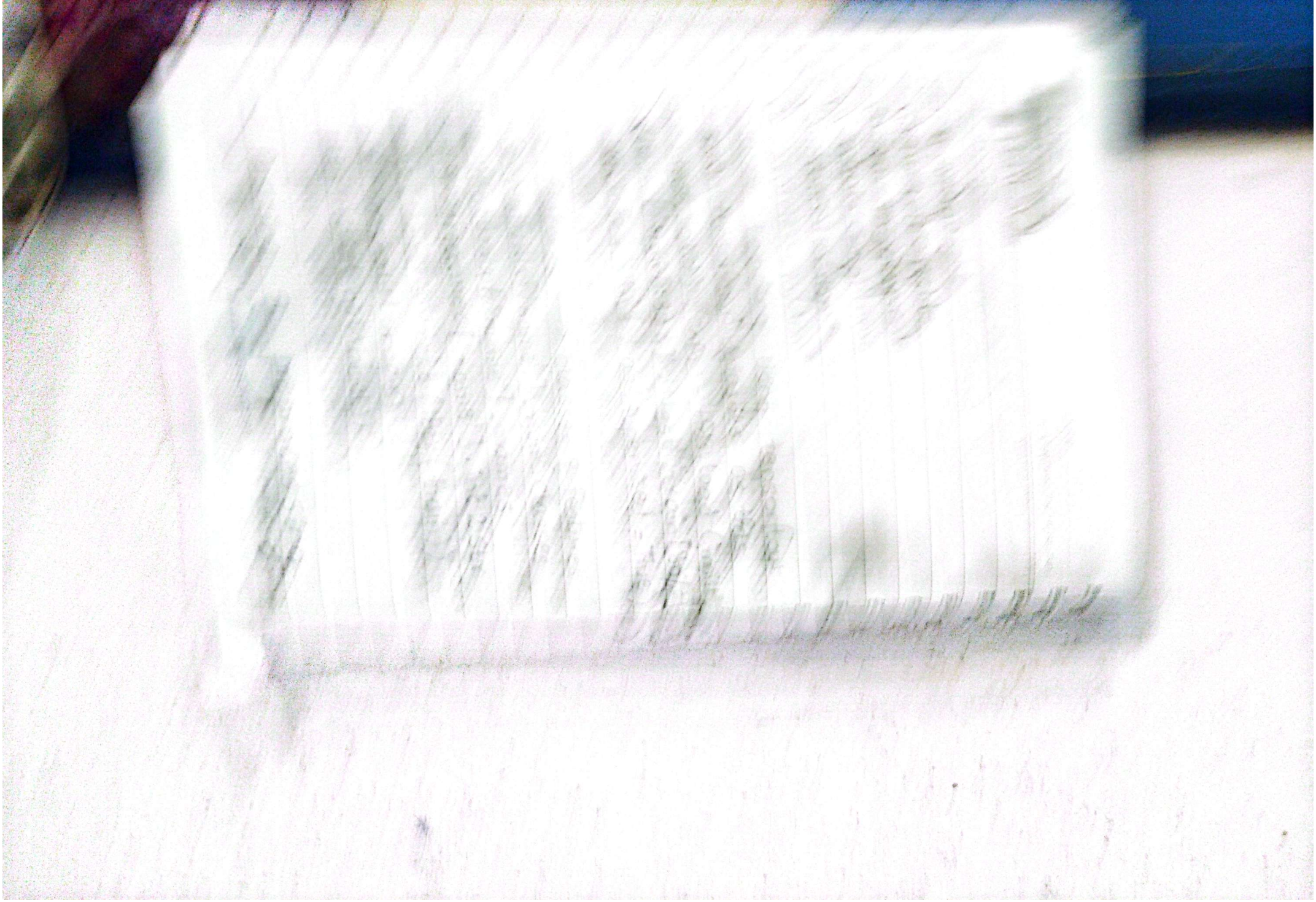
from keras.layers import
Concatenate

out = concatenate([path1...], a
= -1)

→ Data Augmentation:

~ Image processing in Tensorflow

① Resizing
② Rescaling
③ RandomFlip
④ RandomRotation



~ Manipulate pixel size

m = Sequential()
layers - Resizing (size, size)

~ Manipulate pixel values

layers - Rescaling (1./255)
If needed then -1 & 1
(1./127.5, offset = -1)

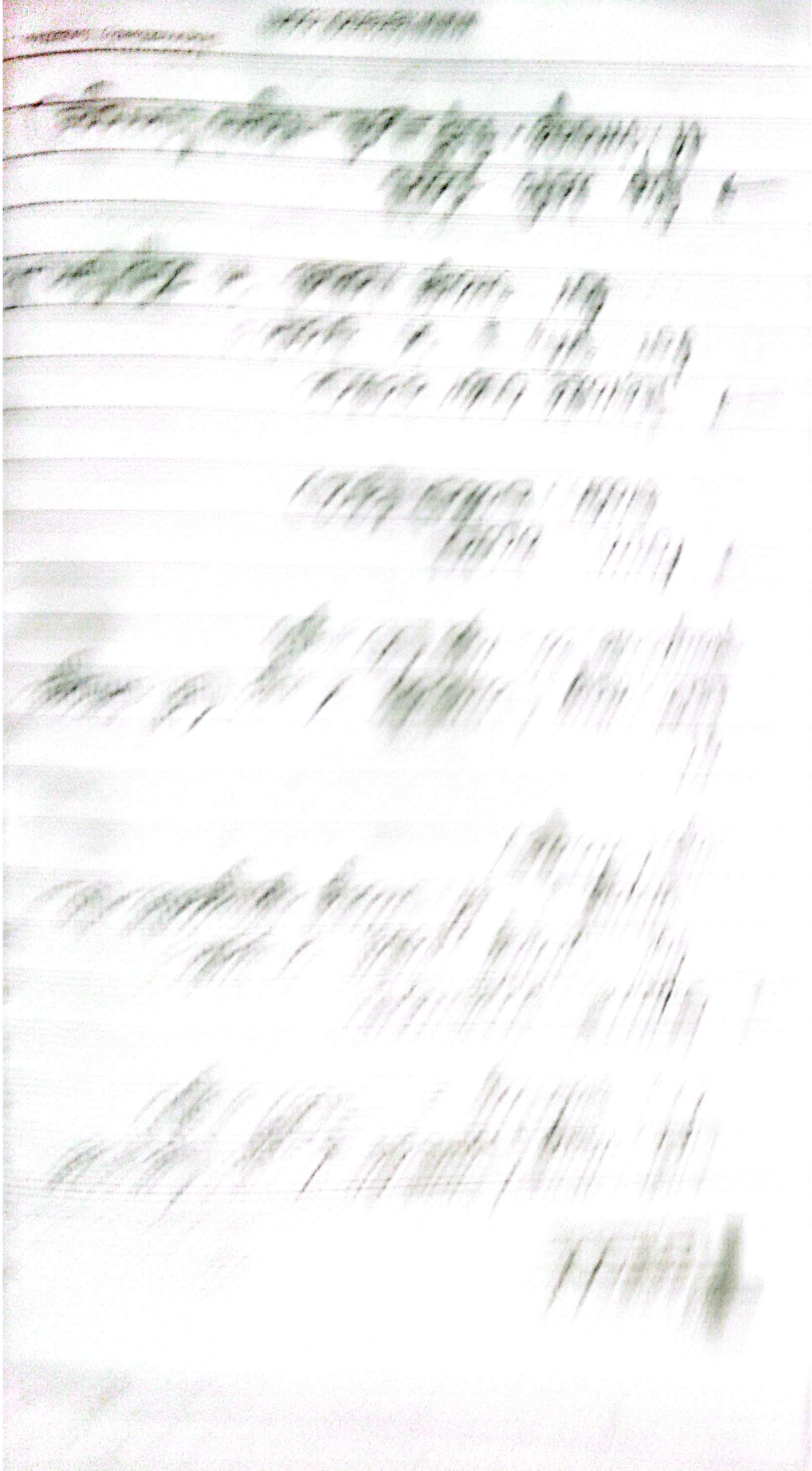
x_train, y_train = next (iter (train))

~ Rotation & flip

L = Sequential()
L.add (RandomFlip ("horizontal"))
L.add (RandomRotation (0.2))

~ Expand dim

tf.expand_dims (image, 0) → axis
tf.cast (g, tf.float32) ~ convert to required shape
tf.squeeze (g) → reduce dim



- Denoising Encoders
  - No regularizer
  - No node fixing Conv2D layers
- contractive
  - same as above, generate same input
- Variational
  - Instead of discrete encoding using Bernoulli / Gaussian probabilities.

→ load a pretrained model

tf.keras.applications.MobileNetV2(include_top = True, weights = 'imagenet

pretrained_model.trainable = False

decode_predictions = tf.keras.applications.mobilenet_v2.decode_predictions