Harvard IQ +

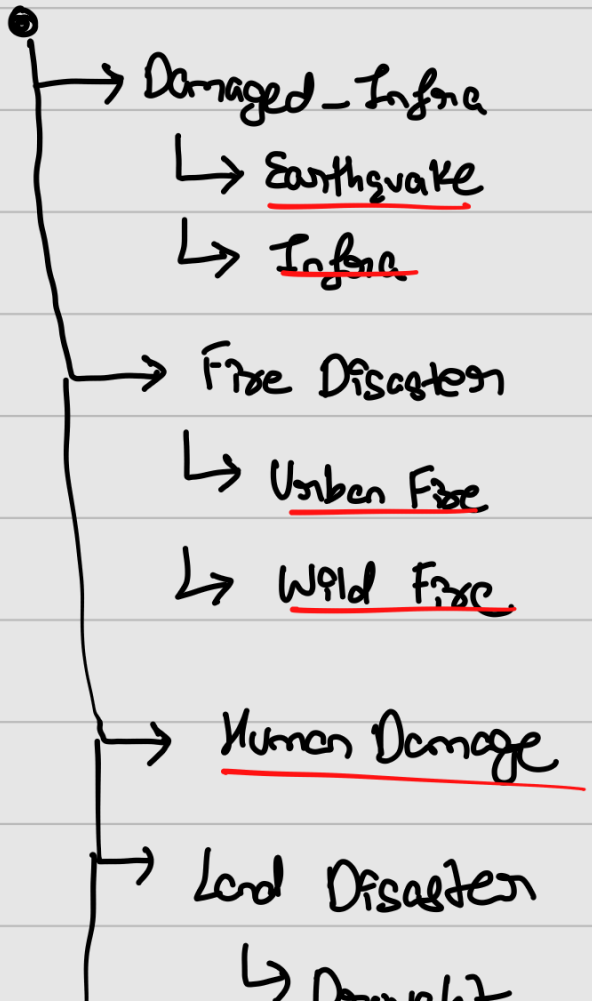
↳ Rajat Nandkumar Shedshyal

↳ after doing a lot of comparison I will be using EfficientNet - lite for fine tuning.

→ Has 5 models [lite0 to lite4]

→ I will go with lite 2 for balance of Accuracy & Speed. (Bad Trade off)

Dataset x Management



↳ Land Slide

↳ Non Damage

↳ Non Damaged Building

↳ Non Damaged Wildfire

↳ Human

↳ ~~Sea~~

↳ Water Disaster.

DataSet x Prep

↳ impt points only

os.walk(root_dir) → walks though every fold as subfold

↳ returns root, subdirs, files

- root → Current dir
- subdirs → List of Sub dirs
- files → List of files

os.path.relpath(root, root_dir)

↳ gets foldername relative to root_dir

root_dir / bla

↳ gets this

in "torchvision"

import datasets, then we get

```
[redacted] = datasets.[redacted].default_loader(path)
```

sample = datasets.folder.default_loader(path)

↳ to open images

or you could use

```
img = np.array(Image.open(img_path))
```

↳ But full way of handling errors

#intentionally wrong spelling 😆

(index+1) % len(self.samples) instead of (index+1)

↳ if 99 % 100 = 99

∴ Can go beyond last index

↓

can go beyond last index

Prob

→ torch.vision doesn't have efficientnet b2 so gotta use "timm".

Image x PreProcessing

→ resize to 224 x 224 [Bicubic]

→ random Horizontal & vertical flip

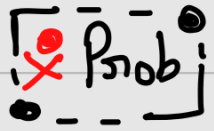
+ to tensor # makes range of pixels to [0,1]

So no Normalization as like ImageNet.

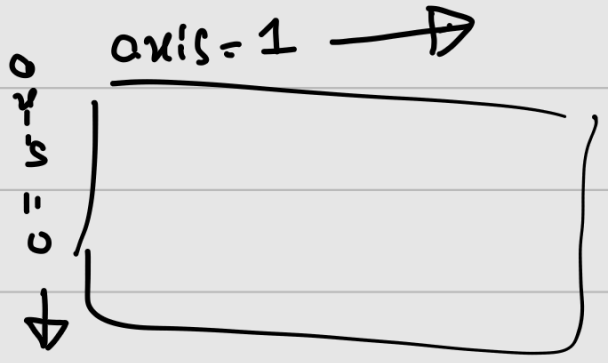
Training Part

axis 1

```
_, preds = torch.max(outputs, 1)
```



↳ when you split into batches na sometimes the last batch may not have same size. so how to handle error calculation?



epoch → Send all data
1 iter → 1 batch sent
1 iter complete

for batches, labels in loader:

loss.item() → gives you avg loss

" like if its MSE & your losses individually are 10, 20, 30 then loss.item() gives 20 i.e $\frac{10+20+30}{3} = 20$ "

So we do

running_loss += loss.item() * inputs.size(0)

"

like (10 + 20 + 30) / 3 = you set sum of losses

& once your all batches are done you divide it with len(dataset)

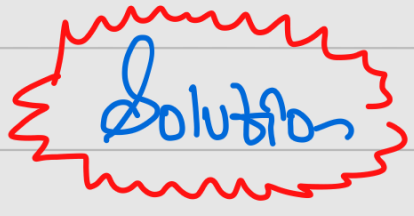
"

actual_epoch_loss = running_loss / len(dataset)

PyTorch → ONXX → Tf Lite

latest tensforflow says it doesn't have "keras.src.engine". Prblm is Onnx_tf calls smtg from tf & current version dont have it.

→ The version mismatch is a big problem



↓

Use virtual environment &

run the code. [Don't use kaggle]

model → exports to "__.onnx"

__.onnx → .tflite using the below code.

STEP 1>

```
torch.onnx.export ( model , dummy_input, "model.onnx",
                    input_names = ["input"],
                    output_names = ["output"],
                    opset_version = 11)
```

↳ gives you model.onnx

STEP 2>

download this model.onnx & then

main.py 2 | model.tflite

main.py > ...

```python
import os
import onnx
from onnx_tf.backend import prepare
import tensorflow as tf

# --- Step 1: Load the ONNX model ---
onnx_model = onnx.load("model.onnx")
print("✅ Loaded ONNX model.")

# --- Step 2: Convert ONNX to TensorFlow SavedModel ---
saved_model_dir = "converted_saved_model"
tf_rep = prepare(onnx_model)
tf_rep.export_graph(saved_model_dir)
print(f"✅ Converted to TensorFlow SavedModel at: {saved_model_dir}")

# --- Step 3: Convert SavedModel to TFLite ---
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
tflite_model = converter.convert()
print("✅ Converted to TFLite format.")

# --- Step 4: Save the .tflite model ---
tflite_path = "model.tflite"
with open(tflite_path, "wb") as f:
    f.write(tflite_model)
print(f"✅ Saved TFLite model to: {tflite_path}")

# --- Step 5: Print Input/Output Details ---
interpreter = tf.lite.Interpreter(model_path=tflite_path)
interpreter.allocate_tensors()

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

print("\n🔍 Model Input(s):")
for idx, i in enumerate(input_details):
    print(f"  [{idx}] Name: {i['name']}, Shape: {i['shape']}, Dtype: {i['dtype']}")

print("\n🔍 Model Output(s):")
for idx, o in enumerate(output_details):
    print(f"  [{idx}] Name: {o['name']}, Shape: {o['shape']}, Dtype: {o['dtype']}")

print("\n✅ All Done.")
```

this gives you .tflite

↳ install all dependencies first

Step 3) [OPTIONAL] You add in metadata to help App developers read your model.

→ You have to install "tflite_support" first

---

recalls =

num of classes →

for each epoch ↓

→ change the forward pass code in Class as

```
def forward (self, x):
    is_disaster = Transformer(—>)
```

yes {

Simple code

}

no {

[1,1,1,1,1,1,1,1,1,1]

}

→ EDA

↳ I dont have to do Manual Normalization as this from efficient Net gets the job done on its own

↳ but i dont (add it is i have to normalize the images

→ We see for 5 random images that pixel intensities are not uniform so lets do something & redistribute it

↓

Use histogram equalization

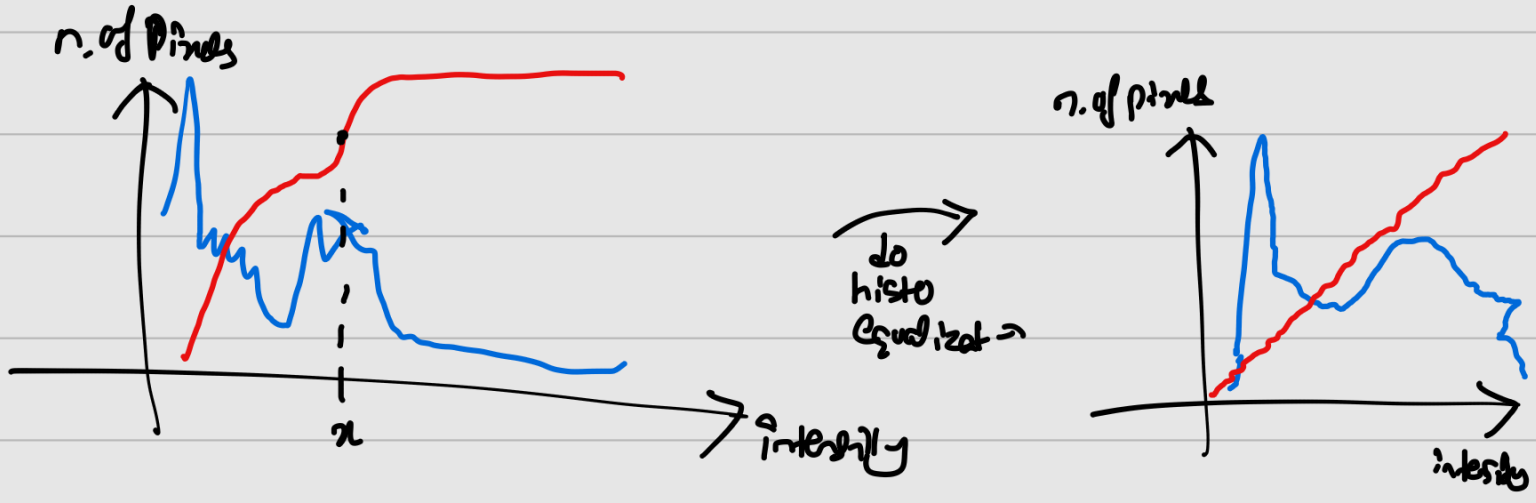
↳ a method to redistribute pixel intensities so that all intensities are equally distributed

CDF → Cummulative Distribution Function

↳ Cummulative sum of intensity values

Histo → counts for each intensity how many pixels

CDF → tells as you go on increasing how many pixels have the same intensity or less



means most of the pixels have intensity $< x$ so the image is darker.

CLAHE → contrast limiting adaptive equalization

↳ to enhance local contrast

1. Image is divided into small tiles / blocks
2. a histo for each tile
3. If any histo bin is above clip limit then those pixels are clipped
4. The clipped excess is redistributed uniformly to other bins before hist normalization
5. Hist normalization to each tile
6. Final Image is obtained by interpolating the tiles.

$$P_{normalized} = \frac{P - P_{min}}{P_{max} - P_{min}} \times (new_{max} - new_{min}) + new_{min}$$

Solving Imbalanced data

Problems → Biasness

→ Metrics like accuracy are not reliable

Even a dumb model will predict good accuracy

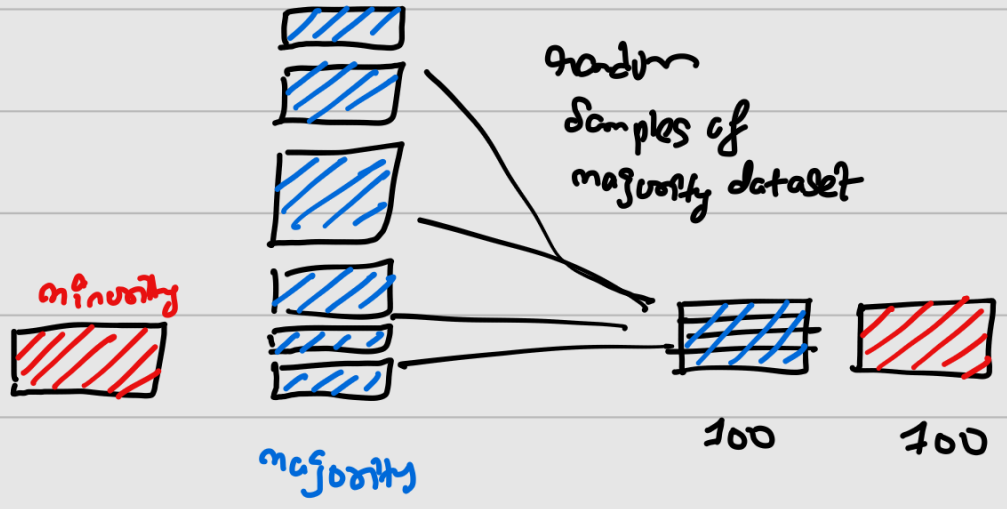
Why IMPT?

Health Care → rare diseases

Fraud → fraud, risk assessment

Consumer → churn prediction

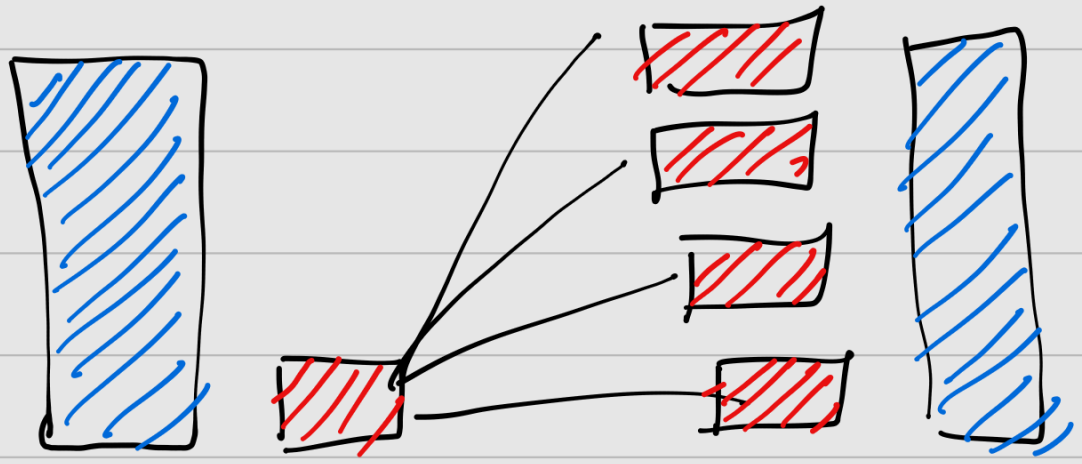
1. Undersampling



Original Dataset

Advantages	Disadvantages
1. Red= in bias	1. Info loss to underfitting
2. Fast training	2. Sampling Bias

2. Over sampling



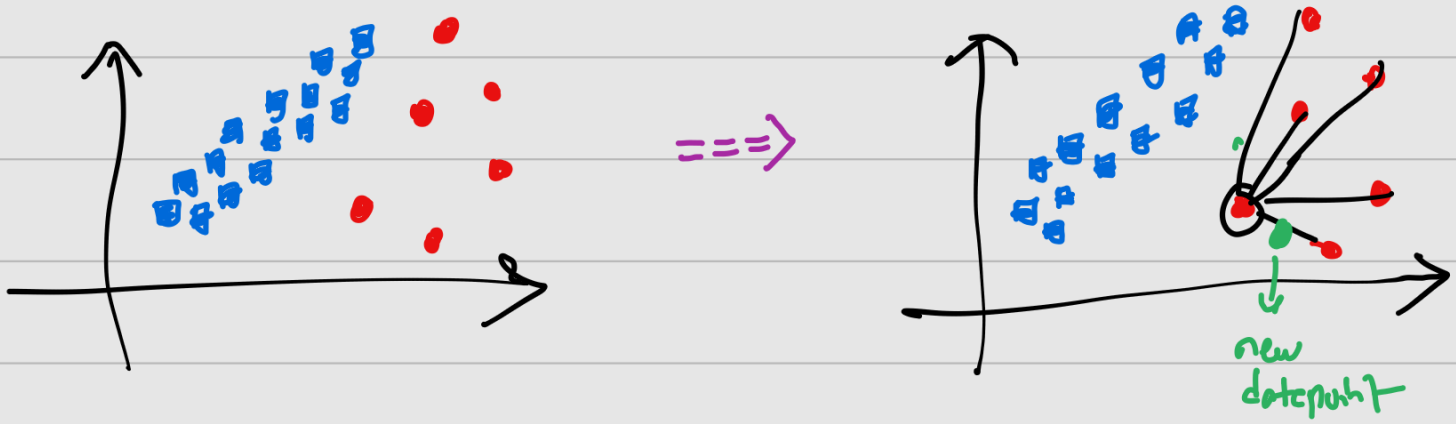
→ Copies but random Sampling as before

Advantages	Disadvantages
1. Red= in Bias	1. Increased size
	2. Dupl= of data, so* may overfit

3 Smote

3. SMOTE

↳ Synthetic minority oversampling technique



Train a KNN on minority class obs - find 5 nearest neighbours for each

→ select one at random (for the interpolation)

→ Extract a random number b/w 0 & 1 (factor)

→ Calculate the new example as

$$= \underset{\text{Sample}}{\text{Org}} - \text{factor} * (\underset{\text{Sample}}{\text{Original}} - \text{neighbour})$$

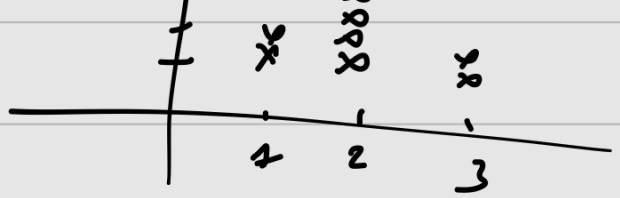
→ Repeat until you get minority class as equal as majority class

Advantages

Bias ↓

Disadvantages

1) Doesn't handle categorical data



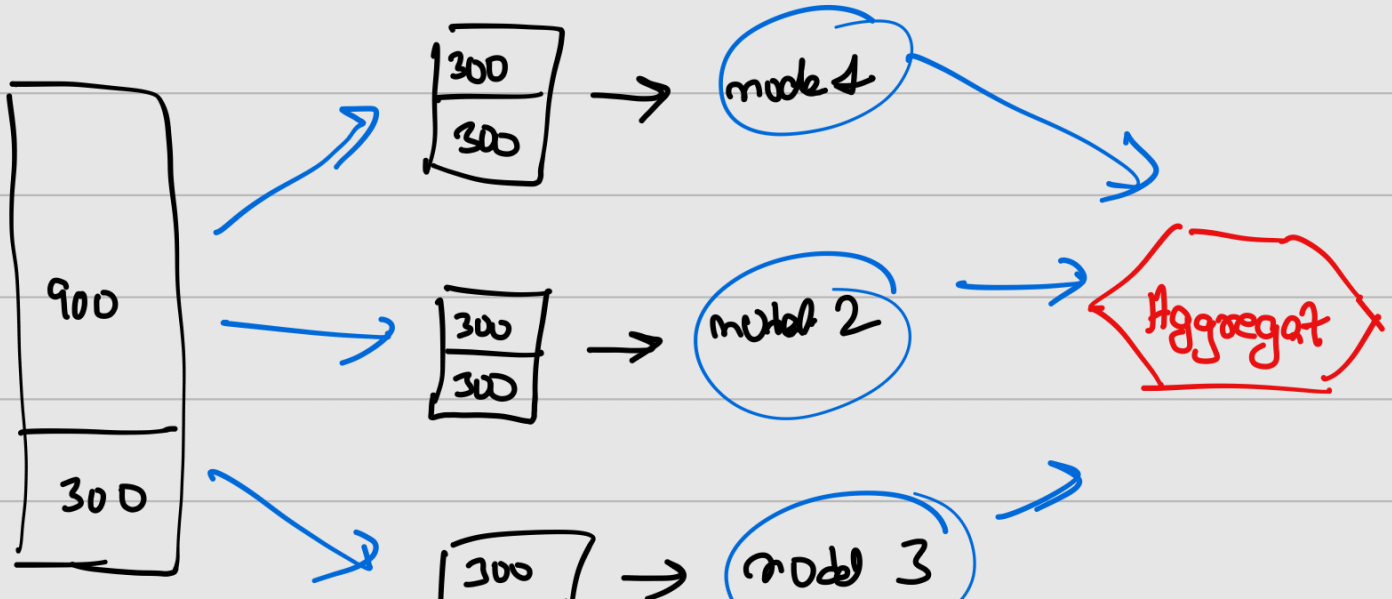
2) Computational Complexity

3) depends on K

4) Sensitive to outliers

5) Balance achieved may not reflect true nature.

4. Ensemble Methods



these models are Trees

5. Cost Sensitive

learning process → imbalan data

1) Class weights

[1, 0]

↓

weight [1, 10]

↳ in model only you get this

2) Custom loss function

MSE

Log loss etc

Ex:- FIL IT

+

Stratified K-fold

Class Imbalance is solved

Custom model {

Vision Transformer 17k 20kft

↓

efficient net

}

I have added one extra class which has no train data just so that ViT such as Roma return

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

↑ Fake Disaster Class

↓

Test later

→ We will use m. Cross Entropy loss with weights

$W_c = \frac{N}{C \cdot n_c}$ → Total samps

C → n. of classes

n_c → n. of samples for c class

weight for c class

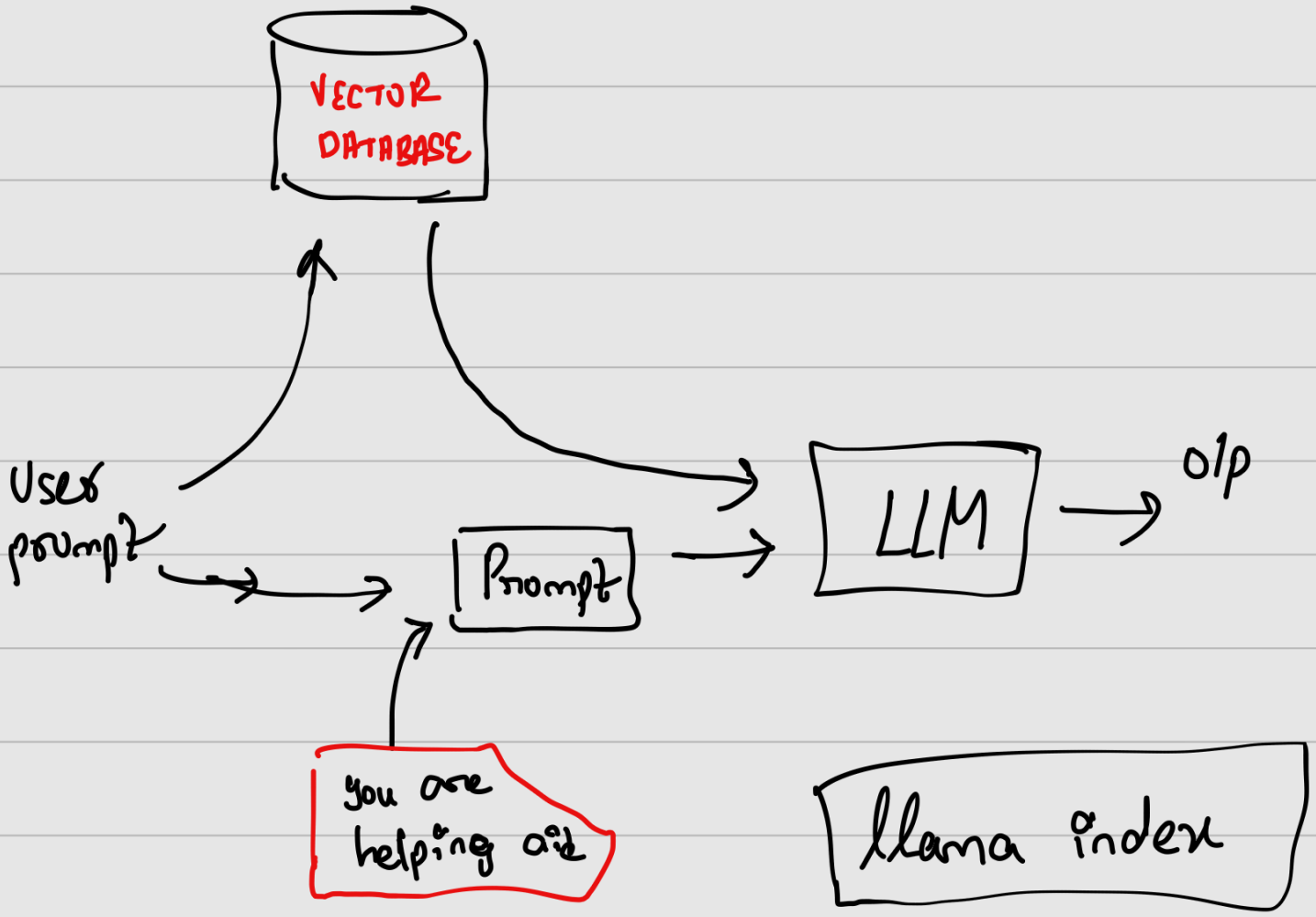
∴ m. Cross Entropy loss

$$L = -\sum_{c=1}^{C} w_c \cdot y_c \log(p_c)$$

Cross Entropy loss + Class weights

→ Usually you are supposed to use diff

models for Stratified K-Fold, but we just wanna solve



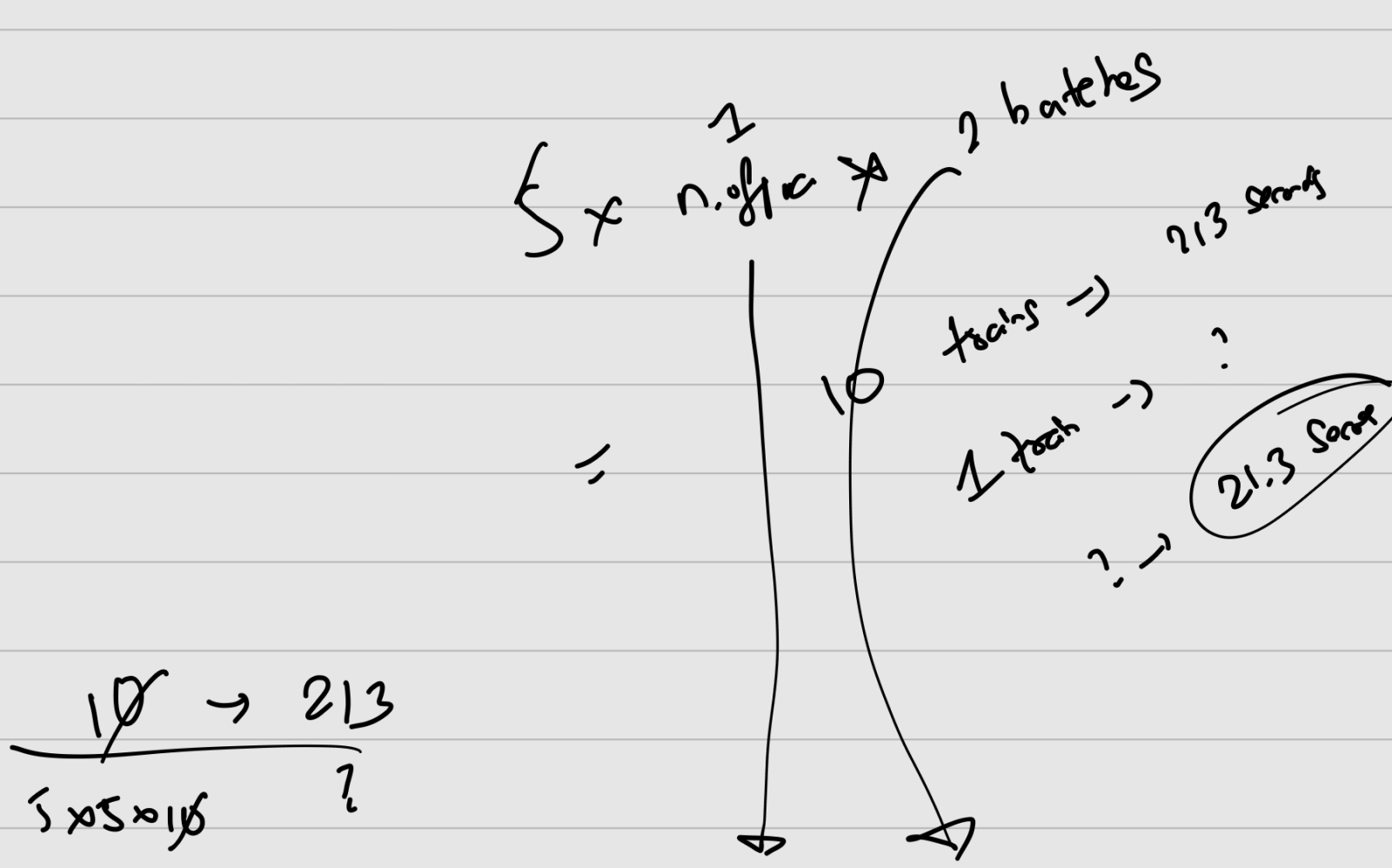
↳ use API

↳ hosted on HITHUB

5 fold

1 epoch

↳ 2 batch



$$\frac{10 \rightarrow 213}{5 \times 5 \times 10 \quad ?}$$

$? \rightarrow 213 \times 5 \times 32$ $\{x \leq x \times 10$