CAPSTONE DESIGN IN
COMPANY'S PROJECT (001)

PRESENTER: RAMAZON KOMILOV (201912128)

FALL DOWN DETECTION AI SOFTWARE

PROJECT BACKGROUND

In many countries, people live alone, and that could result in accidents where there is nobody to help.

According to WHO, falls are the second leading cause of unintentional injury deaths worldwide.

Our motivation is to develop an AI software application that can detect people who fall or faint suddenly and warn in case of serious injury.

PROJECT GOAL

01

Detect and track people as accurately as possible

02

Identify when they fall down along with some basic actions

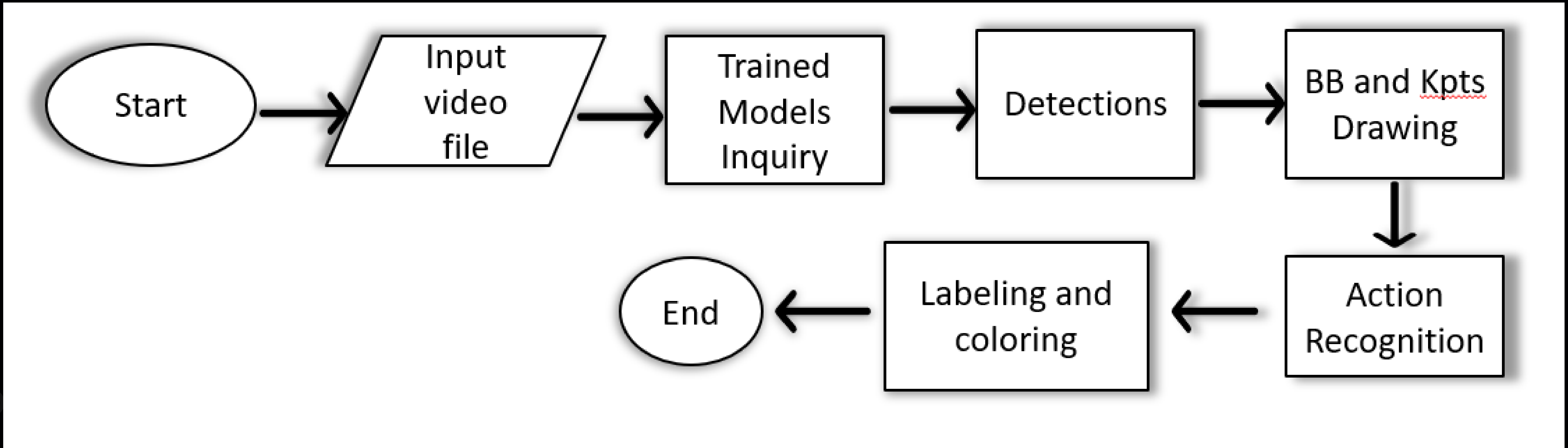
03

Help medical staff react faster in case of emergencies and serve as a fall prevention method

PLANNED SCHEDULE

세부내용	수행기간(월)																비고
	9				10				11				12				
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	
1. Research and study	O	O	O	O	O	O	O										
2. Training and testing								O	O								
3. Software implementation										O	O	O	O	O			

SOFTWARE OVERVIEW



SOFTWARE ARCHITECTURE

01

Input a video file

Input video file is transferred to the trained models inquiry block. Supported formats are mp4, mov, avi, gif, WMV. However, mp4 and avi are recommended to use.

02

Trained models inquiry

Tiny-YOLO one class, AlphaPose and ResNet50 are loaded to operate on the input file.

03

Detections

Person objects are detected by the tiny-YOLO one class. Unique IDs are given to each detected person in the given frames.

04

BB and Kpts Drawing

This is the module where the bounding boxes and keypoints are drawn by tiny-YOLO and AlphaPose. Our pose estimator draws all the keypoints from head to toes in the form of a skeleton.

05

Action Recognition

Here our Resnet50 model detects the actions of the detected person by tiny-YOLO. All the actions were labeled by hand in the phase of training.

06

Labeling and coloring

After the detections are made by ResNet50, the model labels actions based on 7 classes: standing, walking, sitting, lying down, stand up, sit down, fall down. "Fall down" action is the priority, so it is colored red.

ONLY PREREQUISITES

- Python > 3.6
- PyTorch > 1.3.1
- Optionally, CUDA-enabled

TRAINED MODELS

01 Tiny-YOLO One class

Tiny-YOLO model was trained using the COCO dataset to detect only people in the given frame. After the detection, the model draws bounding boxes and gives a unique ID number.

02 AlphaPose

Alpha Pose is an accurate multi-person pose estimator, which is the first open-source system that achieves 70+ mAP (72.3 mAP) on COCO dataset and 80+ mAP (82.1 mAP) on MPII dataset.

03 ResNet50 for Fall Detection

Using ResNet50, the model then was trained to identify and label when the detected people fall down along with some other basic actions such as standing, sitting, lying down.

DATASET INFORMATION

COCO Dataset

Our tiny-yolo model is wholly trained on only one class which is person class in COCO dataset

Fall Detection Dataset

This dataset contains raw RGB and Depth images of size 320x240 for the simulation purposes. Out of total datasets of 22636 images, we used 1011 images for training and 971 images for testing.

Custom Fall Dataset

This dataset contains 800 FHD video clips of professional actors simulating a fall and faints. 3062 images of important scenarios were cut and used for annotation.

LET'S DO SOME CODING!

1. Training phase
2. Pose loader
3. Detector loader
4. Action Loader

TRAINING PHASE

```python
save_folder = 'saved/TSSTG(pts+mot)-01(cf+hm-hm)'

device = 'cuda'
epochs = 30
batch_size = 32

# DATA FILES.
# Should be in format of
#  inputs: (N_samples, time_steps, graph_node, channels),
#  labels: (N_samples, num_class)
#   and do some of normalizations on it. Default data create from:
#       Data.create_dataset_(1-3).py
# where
#   time_steps: Number of frame input sequence, Default: 30
#   graph_node: Number of node in skeleton, Default: 14
#   channels: Inputs data (x, y and scores), Default: 3
#   num_class: Number of pose class to train, Default: 7

data_files = ['../Data/Coffee_room_new-set(labelXscrw).pkl',
              '../Data/Home_new-set(labelXscrw).pkl']
class_names = ['Standing', 'Walking', 'Sitting', 'Lying Down',
               'Stand up', 'Sit down', 'Fall Down']
num_class = len(class_names)
```

```python
def load_dataset(data_files, batch_size, split_size=0):
    """Load data files into torch DataLoader with/without spliting train-test.
    """
    features, labels = [], []
    for fil in data_files:
        with open(fil, 'rb') as f:
            fts, lbs = pickle.load(f)
            features.append(fts)
            labels.append(lbs)
        del fts, lbs
    features = np.concatenate(features, axis=0)
    labels = np.concatenate(labels, axis=0)

    if split_size > 0:
        x_train, x_valid, y_train, y_valid = train_test_split(features, labels, test_size=split_size,
                                                              random_state=9)
        train_set = data.TensorDataset(torch.tensor(x_train, dtype=torch.float32).permute(0, 3, 1, 2),
                                       torch.tensor(y_train, dtype=torch.float32))
        valid_set = data.TensorDataset(torch.tensor(x_valid, dtype=torch.float32).permute(0, 3, 1, 2),
                                       torch.tensor(y_valid, dtype=torch.float32))
        train_loader = data.DataLoader(train_set, batch_size, shuffle=True)
        valid_loader = data.DataLoader(valid_set, batch_size)
    else:
        train_set = data.TensorDataset(torch.tensor(features, dtype=torch.float32).permute(0, 3, 1, 2),
                                       torch.tensor(labels, dtype=torch.float32))
        train_loader = data.DataLoader(train_set, batch_size, shuffle=True)
        valid_loader = None
    return train_loader, valid_loader


def accuracy_batch(y_pred, y_true):
    return (y_pred.argmax(1) == y_true.argmax(1)).mean()


def set_training(model, mode=True):
    for p in model.parameters():
        p.requires_grad = mode
    model.train(mode)
    return model
```

TRAINING PHASE

```python
# TRAINING.
loss_list = {'train': [], 'valid': []}
accu_list = {'train': [], 'valid': []}
for e in range(epochs):
    print('Epoch {}/{}'.format(e, epochs - 1))
    for phase in ['train', 'valid']:
        if phase == 'train':
            model = set_training(model, True)
        else:
            model = set_training(model, False)

        run_loss = 0.0
        run_accu = 0.0
        with tqdm(dataloader[phase], desc=phase) as iterator:
            for pts, lbs in iterator:
                # Create motion input by distance of points (x, y) of the same node
                # in two frames.
                mot = pts[:, :2, 1:, :] - pts[:, :2, :-1, :]

                mot = mot.to(device)
                pts = pts.to(device)
                lbs = lbs.to(device)
```

```python
            # Forward.
            out = model((pts, mot))
            loss = losser(out, lbs)

            if phase == 'train':
                # Backward.
                model.zero_grad()
                loss.backward()
                optimizer.step()

            run_loss += loss.item()
            accu = accuracy_batch(out.detach().cpu().numpy(),
                                  lbs.detach().cpu().numpy())
            run_accu += accu

            iterator.set_postfix_str(' loss: {:.4f}, accu: {:.4f}'.format(
                loss.item(), accu))
            iterator.update()
            #break
    loss_list[phase].append(run_loss / len(iterator))
    accu_list[phase].append(run_accu / len(iterator))
    #break

print('Summary epoch:\n - Train loss: {:.4f}, accu: {:.4f}\n - Valid loss:'
      ' {:.4f}, accu: {:.4f}'.format(loss_list['train'][-1], accu_list['train'][-1],
                                     loss_list['valid'][-1], accu_list['valid'][-1]))

# SAVE.
torch.save(model.state_dict(), os.path.join(save_folder, 'tsstg-model.pth'))
```

POSE LOADER

```
class SPPE_FastPose(object):

    def __init__(self,
                 backbone,
                 input_height=320,
                 input_width=256,
                 device='cuda'):
        assert backbone in ['resnet50', 'resnet101'], '{} backbone is not support yet!'.format(backbone)

        self.inp_h = input_height
        self.inp_w = input_width
        self.device = device

        if backbone == 'resnet101':
            self.model = InferenNet_fast().to(device)
        else:
            self.model = InferenNet_fastRes50().to(device)
        self.model.eval()

    def predict(self, image, bboxs, bboxs_scores):
        inps, pt1, pt2 = crop_dets(image, bboxs, self.inp_h, self.inp_w)
        pose_hm = self.model(inps.to(self.device)).cpu().data

        # Cut eyes and ears.
        pose_hm = torch.cat([pose_hm[:, :1, ...], pose_hm[:, 5:, ...]], dim=1)

        xy_hm, xy_img, scores = getPrediction(pose_hm, pt1, pt2, self.inp_h, self.inp_w,
                                              pose_hm.shape[-2], pose_hm.shape[-1])
        result = pose_nms(bboxs, bboxs_scores, xy_img, scores)
        return result
```

DETECTOR LOADER

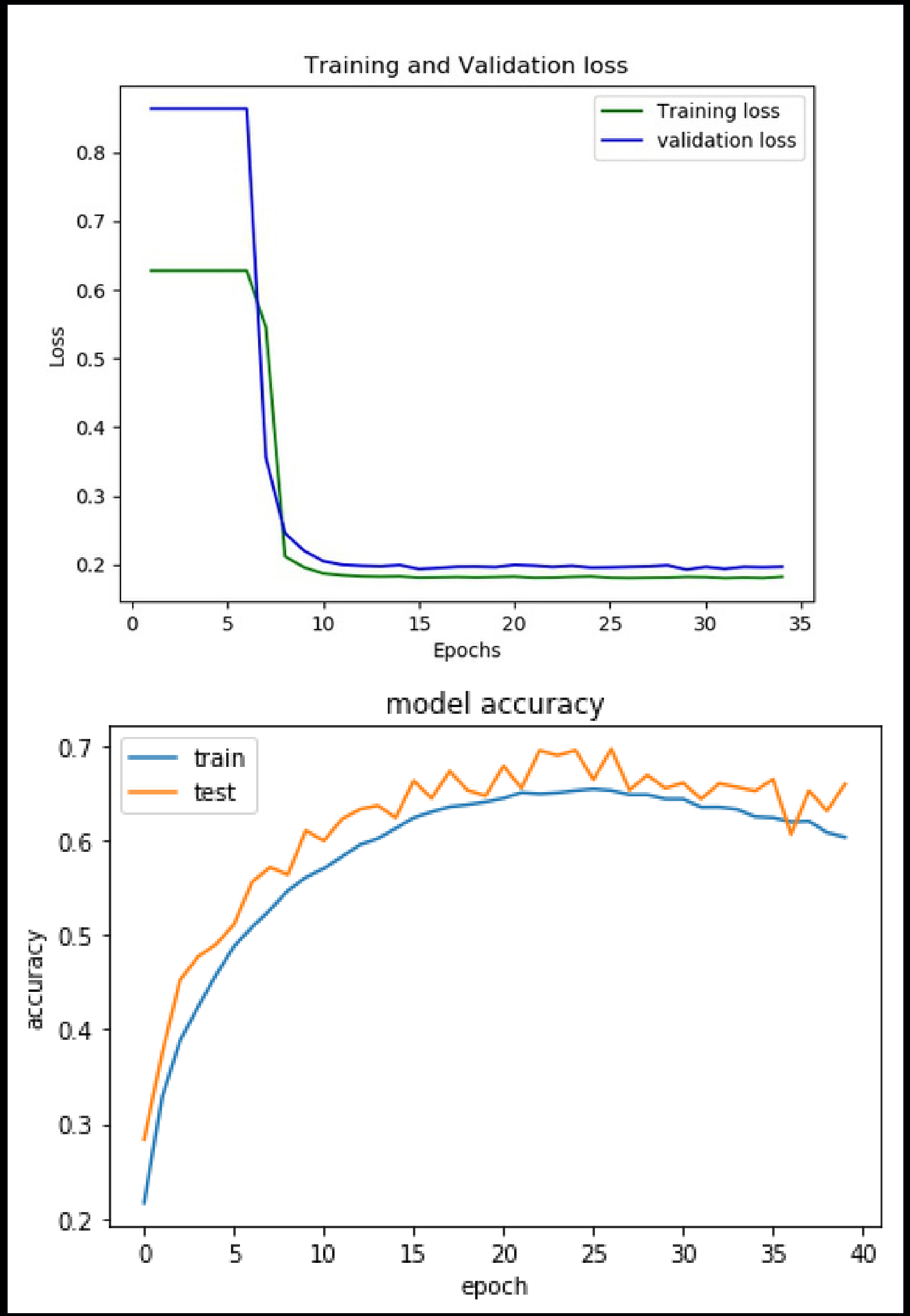
```python
class TinyYOLOv3_onecls(object):
    """Load trained Tiny-YOLOv3 one class (person) detection model.
    Args:
        input_size: (int) Size of input image must be divisible by 32. Default: 416,
        config_file: (str) Path to Yolo model structure config file.,
        weight_file: (str) Path to trained weights file.,
        nms: (float) Non-Maximum Suppression overlap threshold.,
        conf_thres: (float) Minimum Confidence threshold of predicted bboxs to cut off.,
        device: (str) Device to load the model on 'cpu' or 'cuda'.
    """
    def __init__(self,
                 input_size=416,
                 config_file='Models/yolo-tiny-onecls/yolov3-tiny-onecls.cfg',
                 weight_file='Models/yolo-tiny-onecls/best-model.pth',
                 nms=0.2,
                 conf_thres=0.45,
                 device='cuda'):
        self.input_size = input_size
        self.model = Darknet(config_file).to(device)
        self.model.load_state_dict(torch.load(weight_file))
        self.model.eval()
        self.device = device

        self.nms = nms
        self.conf_thres = conf_thres

        self.resize_fn = ResizePadding(input_size, input_size)
        self.transf_fn = transforms.ToTensor()
```

```python
    def detect(self, image, need_resize=True, expand_bb=5):
        """Feed forward to the model.
        Args:
            image: (numpy array) Single RGB image to detect.,
            need_resize: (bool) Resize to input_size before feed and will return bboxs
                with scale to image original size.,
            expand_bb: (int) Expand boundary of the boxs.
        Returns:
            (torch.float32) Of each detected object contain a
                [top, left, bottom, right, bbox_score, class_score, class]
            return `None` if no detected.
        """
        image_size = (self.input_size, self.input_size)
        if need_resize:
            image_size = image.shape[:2]
            image = self.resize_fn(image)

        image = self.transf_fn(image)[None, ...]
        scf = torch.min(self.input_size / torch.FloatTensor([image_size]), 1)[0]

        detected = self.model(image.to(self.device))
        detected = non_max_suppression(detected, self.conf_thres, self.nms)[0]
        if detected is not None:
            detected[:, [0, 2]] -= (self.input_size - scf * image_size[1]) / 2
            detected[:, [1, 3]] -= (self.input_size - scf * image_size[0]) / 2
            detected[:, 0:4] /= scf

            detected[:, 0:2] = np.maximum(0, detected[:, 0:2] - expand_bb)
            detected[:, 2:4] = np.minimum(image_size[::-1], detected[:, 2:4] + expand_bb)

        return detected
```

ACTION LOADER

```python
def __init__(self,
             weight_file='./Models/TSSTG/tsstg-model.pth',
             device='cuda'):
    self.graph_args = {'strategy': 'spatial'}
    self.class_names = ['Standing', 'Walking', 'Sitting', 'Lying Down',
                        'Stand up', 'Sit down', 'Fall Down']
    self.num_class = len(self.class_names)
    self.device = device

    self.model = TwoStreamSpatialTemporalGraph(self.graph_args, self.num_class).to(self.device)
    self.model.load_state_dict(torch.load(weight_file))
    self.model.eval()

def predict(self, pts, image_size):
    """Predict actions from single person skeleton points and score in time sequence.
    Args:
        pts: (numpy array) points and score in shape `(t, v, c)` where
            t : inputs sequence (time steps).,
            v : number of graph node (body parts).,
            c : channel (x, y, score).,
        image_size: (tuple of int) width, height of image frame.
    Returns:
        (numpy array) Probability of each class actions.
    """
    pts[:, :, :2] = normalize_points_with_size(pts[:, :, :2], image_size[0], image_size[1])
    pts[:, :, :2] = scale_pose(pts[:, :, :2])
    pts = np.concatenate((pts, np.expand_dims((pts[:, 1, :] + pts[:, 2, :]) / 2, 1)), axis=1)

    pts = torch.tensor(pts, dtype=torch.float32)
    pts = pts.permute(2, 0, 1)[None, :]

    mot = pts[:, :2, 1:, :] - pts[:, :2, :-1, :]
    mot = mot.to(self.device)
    pts = pts.to(self.device)

    out = self.model((pts, mot))

    return out.detach().cpu().numpy()
```

DEMONSTRATION



DEMONSTRATION

NOFALL

ONGOING AND FUTURE IMPROVEMENTS

53%

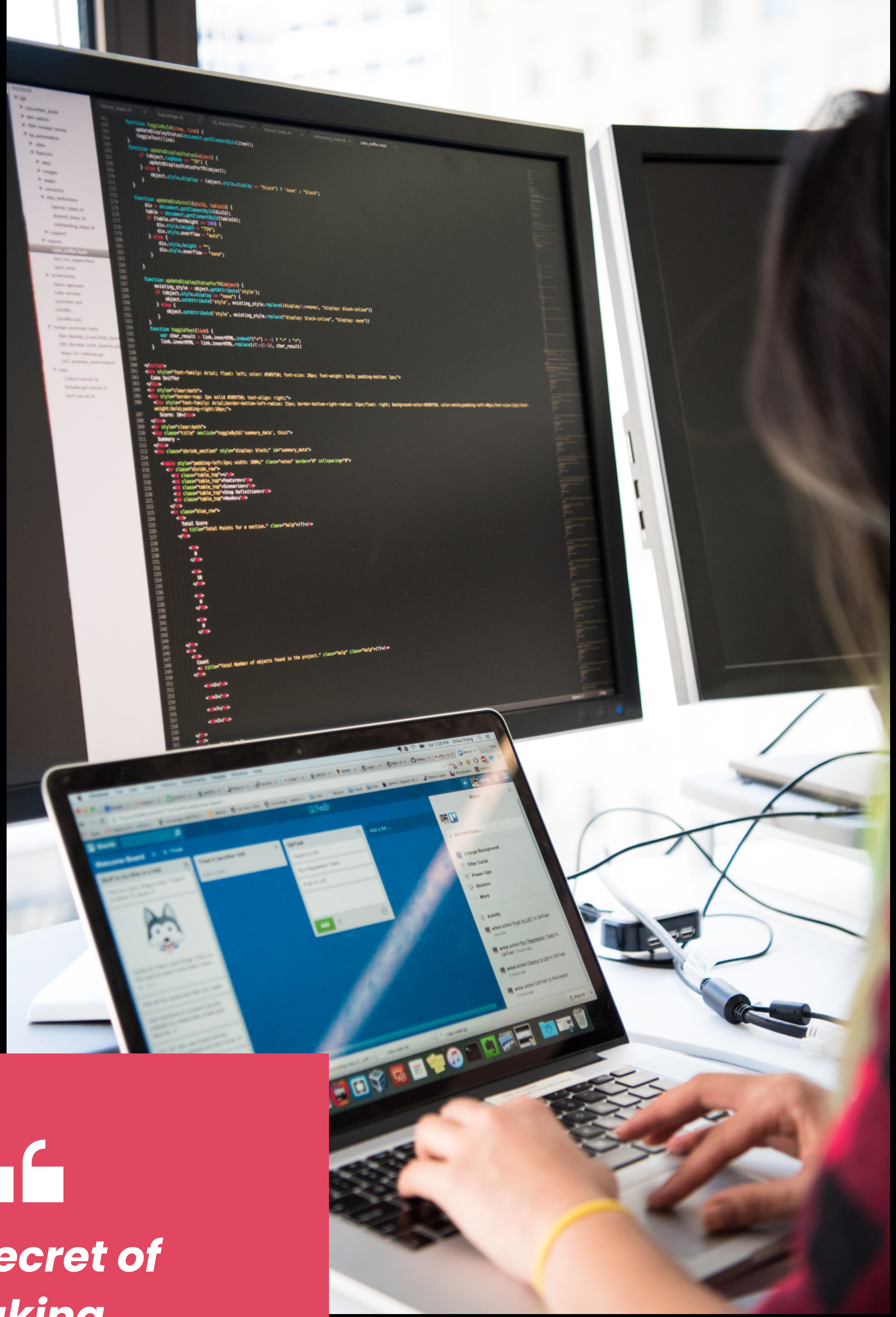
Real-time Monitoring

Real-time monitoring through any camera for any falls or other emergencies

21%

Graphical Interface

A GUI-based software that lets users interact with the application and store all the footage in the computer



"The secret of making progress is to get started"

CAPSTONE DESIGN IN COMPANY'S PROJECT (001)

PRESENTER: RAMAZON KOMILOV (201912128)

THANK YOU