Reckless Road-User Eliminator

0816199陳清海 109550055李耕雨 109550135范恩宇

1. Introduction

The topic we named "Reckless Road-User Eliminator", is basically a system for detecting people that violate traffic rules through recorded videos. The technique refers to violation detection by combining object detection and object tracking, which are used for quite a lot of applications these days.

1.1 Motivation

The increase in vehicular traffic over the past few years has made it more challenging for law enforcement agencies to monitor and identify traffic violations effectively. Taiwan, in particular, has been considered to have the worst traffic among the world. Additionally, determining liability in car accidents can be difficult sometimes, and there have been instances where individuals were unjustly penalized for traffic violations they did not commit.Our project aims to address these issues by proposing a computer vision-based solution that can detect traffic rule violators through videos, which can assist authorities in enforcing traffic regulations efficiently and ensuring fair treatment of offenders.

1.2 Related Work

Traditional traffic violation detection relies on manual monitoring, which is time-consuming and error-prone. With advances in computer vision and deep learning, automated solutions have been proposed. Some use object detection to identify vehicles and pedestrians, while others focus on specific violations. However, many are limited in scope, so we aim to provide a comprehensive solution by leveraging object detection and tracking algorithms to detect multiple types of traffic violations, including overspeeding, not wearing helmets, and driving on the wrong side of the road, etc.

2. Methodology

2.1 Algorithms

Our project consists of two primary algorithms,

YOLOv3:

A real-time object detection system widely adopted in computer vision applications. In our project, it is used to detect helmets worn by motorcycle riders. The algorithm has been trained on a high-resolution helmet image dataset, helping it identify and localize helmets in input video frames.

Deep SORT:

A multiple object tracking algorithm combining information from object detectors and appearance descriptors to track objects across video frames. We use it to track vehicles and pedestrians, enabling the system to maintain consistent identities and trajectories for each object over time.

2.2 Models

Helmet Detection Model:

Trained with YOLOv3 on a dataset of high resolution helmet images. The trained model can detect and localize helmets worn by motorcycle riders in input video frames. Its performance and accuracy were evaluated during training, and the final weights were obtained after 2,400 iterations.

Vehicle and Pedestrian Detection Models:

Haar cascade classifiers were utilized for detecting cars (cars.xml) and motorcycles (motor-v4.xml) in the input video frames. These classifiers were trained on large datasets of vehicle and pedestrian images, helping the system locate then identify vehicles and pedestrians in the scene.

2.3 System

Having three main components: input video capturing real-time traffic information, processor analyzing the video using advanced computer vision techniques to identify overspeeding vehicles and helmetless motorcyclists, and output video labeling these traffic rule violators.

2.4 Code Implementation

The code has two parts, one checks if people wear helmets, the another checks if people overspeed and wear no helmets when riding a motorcycle.

Helmet Detection(helm.py):

drawPred -> Draw the predicted bounding box on the video.

```
# Draw the predicted bounding box
def drawPred(classId, conf, left, top, right, bottom, frame):
    global frame_count
    # Draw a bounding box.
    #cv.rectangle(frame, (left, top), (right, bottom), (255, 178, 50), 3)
    label = '%.2f' % conf
    # Get the label for the class name and its confidence
    if classes:
        assert(classId < len(classes))
        label = '%s:%s' % (classes[classId], label)

    #Display the label at the top of the bounding box
    labelSize, baseLine = cv.getTextSize(label, cv.FONT_HERSHEY_SIMPLEX, 0.5, 1)
    top = max(top, labelSize[1])

    label_name,label_conf = label.split(':') # spliting into class & confidance. will compare it with person.
    if label_name == 'Helmet':
        #cv.rectangle(frame, (left, top - round(1.5*labelSize[1])), (left + round(1.5*labelSize[0]), top + baseLine), (255, 255, 255), cv.FILLED)
        #cv.putText(frame, label, (left, top), cv.FONT_HERSHEY_SIMPLEX, 0.75, (0,0,0), 1)
        frame_count+=1
    #print(frame_count)
    if(frame_count> 0):
        return frame_count
```

postprocess -> Remove the bounding boxes with low confidence and call drawPred to draw the predicted bounding box on the video.

```
# Remove the bounding boxes with low confidence using non-maxima suppression
def postprocess(frame, outs):
    frameHeight = frame.shape[0]
    frameWidth = frame.shape[1]
    frame_count_out=0
    classIds = []
    confidences = []
    boxes = []
    # Scan through all the bounding boxes output from the network and keep only the
    # ones with high confidence scores. Assign the box's class label as the class with the highest score.
    classIds = []               #have to fins which class have hieghest confidence.......=====>>><<<<=======
    confidences = []
    boxes = []
    for out in outs:
        for detection in out:
            scores = detection[5:]
            classId = np.argmax(scores)
            confidence = scores[classId]
            if confidence > confThreshold:
                center_x = int(detection[0] * frameWidth)
                center_y = int(detection[1] * frameHeight)
                width = int(detection[2] * frameWidth)
                height = int(detection[3] * frameHeight)
                left = int(center_x - width / 2)
                top = int(center_y - height / 2)
                classIds.append(classId)
                #print(classIds)
                confidences.append(float(confidence))
                boxes.append([left, top, width, height])
```

```
    # Perform non maximum suppression to eliminate redundant overlapping boxes with
    # lower confidences.
    indices = cv.dnn.NMSBoxes(boxes, confidences, confThreshold, nmsThreshold)
    count_person=0 # for counting the classes in this loop.
    for i in indices:
        box = boxes[i]
        left = box[0]
        top = box[1]
        width = box[2]
        height = box[3]
        frame_count_out = drawPred(classIds[i], confidences[i], left, top, left + width, top + height, frame)

        my_class='Helmet'
        unknown_class = classes[classId]

        if my_class == unknown_class:
            count_person += 1

    if count_person >= 1:
        path = 'test_out/'
        return 1
    else:
        return 0
```

detect -> Generate the predicted output from video frames and call the postprocess function.

```
# Process inputs
winName = 'Deep learning object detection in OpenCV'
cv.namedWindow(winName, cv.WINDOW_NORMAL)

def detect(frame):
    #frame = cv.imread(fn)

    # Create a 4D blob from a frame.
    blob = cv.dnn.blobFromImage(frame, 1/255, (inpWidth, inpHeight), [0,0,0], 1, crop=False)
    # Sets the input to the network
    net.setInput(blob)

    # Runs the forward pass to get output of the output layers
    outs = net.forward(getOutputsNames(net))

    # Remove the bounding boxes with low confidence
    # getPerfProfile() returns the overall time for inference(t) and timings for each of the layers(in layersTimes)
    t, _ = net.getPerfProfile()
    #print(t)
    #label = 'Inference time: %.2f ms' % (t * 1000.0 / cv.getTickFrequency())
    #print(label)
    #cv.putText(frame, label, (0, 15), cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255))
    #print(label)
    k=postprocess(frame, outs)
    if k:
        return 1
    else:
        return 0
```

Overspeeding Detection & Check if riders not wearing helmets(all_vehicles.py):

estimateSpeed -> Use the locations of two frames and the fps between the two frames as the input and calculate the speed of a vehicle.

```
def estimateSpeed(location1, location2,fps):
    d_pixels = math.sqrt(math.pow(location2[0] - location1[0], 2) + math.pow(location2[1] - location1[1], 2))
    # ppm = location2[2] / carWidht
    ppm = 8.8
    d_meters = d_pixels / ppm
    if fps == 0.0:
        fps = 18
    #fps=18
    #print("d_pixels=" + str(d_pixels), "d_meters=" + str(d_meters))
    #fps = 7
    speed = d_meters * fps * 3.6
    return speed
```

trackMultipleObjects -> Main function of the whole program.

```python
def trackMultipleObjects():
    rectangleColor = (0, 255, 0)
    frameCounter = 0
    currentCarID = 0
    currentBikeID=0
    fps = 0

    carTracker = {}
    bikeTracker = {}
    bikeNumbers = {}
    carNumbers = {}
    bikeLocation1 = {}
    carLocation1 = {}
    bikeLocation2 = {}
    carLocation2 = {}
    speed = [None] * 1000
    go =[False for i in range(1000)]
    identity = [0 for i in range(1000)]
    snaps = [False for i in range(1000)]
    types = ["cars" for i in range(1000)]
    Helmets = ["No Helmet Detected" for i in range(1000)]
    # Write output to video file
    out = cv2.VideoWriter('outpy.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 10, (WIDTH,HEIGHT))
```

This part is for deleting the car which is not showing on the video from the carTracker array.

```python
    while True:
        start_time = time.time()
        rc, image = video.read()
        if type(image) == type(None):
            break

        image = cv2.resize(image, (WIDTH, HEIGHT))
        resultImage = image.copy()
        frameCounter = frameCounter + 1
        carIDtoDelete = []
        for carID in carTracker.keys():
            trackingQuality = carTracker[carID].update(image)

            if trackingQuality < 7:
                carIDtoDelete.append(carID)

        for carID in carIDtoDelete:
            print ('Removing carID ' + str(carID) + ' from list of trackers.')
            print ('Removing carID ' + str(carID) + ' previous location.')
            print ('Removing carID ' + str(carID) + ' current location.')
            carTracker.pop(carID, None)
            carLocation1.pop(carID, None)
            carLocation2.pop(carID, None)
```

This part is used to add new cars and bikes into the tracking list.

```python
    if not (frameCounter % 10):
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        cars = carCascade.detectMultiScale(gray, 1.1, 13, 18, (24, 24))
        bikes = bikeCascade.detectMultiScale(gray, 1.1 , 13, 18, (24,24))
        for (_x, _y, _w, _h) in cars:
            x = int(_x)
            y = int(_y)
            w = int(_w)
            h = int(_h)

            x_bar = x + 0.5 * w
            y_bar = y + 0.5 * h

            matchCarID = None

            for carID in carTracker.keys():
                trackedPosition = carTracker[carID].get_position()

                t_x = int(trackedPosition.left())
                t_y = int(trackedPosition.top())
                t_w = int(trackedPosition.width())
                t_h = int(trackedPosition.height())

                t_x_bar = t_x + 0.5 * t_w
                t_y_bar = t_y + 0.5 * t_h

                if ((t_x <= x_bar <= (t_x + t_w)) and (t_y <= y_bar <= (t_y + t_h)) and (x <= t_x_bar <= (x + w)) and (y <= t_y_bar <= (y + h))):
                    matchCarID = carID

            if matchCarID is None:
                print ('Creating new tracker ' + str(currentCarID))

                tracker = dlib.correlation_tracker()
                tracker.start_track(image, dlib.rectangle(x, y, x + w, y + h))

                carTracker[currentCarID] = tracker
                carLocation1[currentCarID] = [x, y, w, h]

                currentCarID = currentCarID + 1
```

```
        for (_x, _y, _w, _h) in bikes:
            x = int(_x)
            y = int(_y)
            w = int(_w)
            h = int(_h)

            x_bar = x + 0.5 * w
            y_bar = y + 0.5 * h

            matchCarID = None

            for carID in carTracker.keys():
                trackedPosition = carTracker[carID].get_position()

                t_x = int(trackedPosition.left())
                t_y = int(trackedPosition.top())
                t_w = int(trackedPosition.width())
                t_h = int(trackedPosition.height())

                t_x_bar = t_x + 0.5 * t_w
                t_y_bar = t_y + 0.5 * t_h

                if ((t_x <= x_bar <= (t_x + t_w)) and (t_y <= y_bar <= (t_y + t_h)) and (x <= t_x_bar <= (x + w)) and (y <= t_y_bar <= (y + h))):
                    matchCarID = carID

            if matchCarID is None:
                print ('Creating new tracker ' + str(currentCarID))

                tracker = dlib.correlation_tracker()
                tracker.start_track(image, dlib.rectangle(x, y, x + w, y + h))

                carTracker[currentCarID] = tracker
                carLocation1[currentCarID] = [x, y, w, h]
                types[currentCarID]= "bikes"
                currentCarID = currentCarID + 1
```

This part is for getting new locations of cars to do speed estimation.

```
for carID in carTracker.keys():
    trackedPosition = carTracker[carID].get_position()

    t_x = int(trackedPosition.left())
    t_y = int(trackedPosition.top())
    t_w = int(trackedPosition.width())
    t_h = int(trackedPosition.height())
    #cv2.rectangle(resultImage, (t_x, t_y), (t_x + t_w, t_y + t_h), rectangleColor, 4)

    # speed estimation
    carLocation2[carID] = [t_x, t_y, t_w, t_h]
```

This part is to use the functions we previously mentioned, to check whether the car is overspeed and the bike is helmeted and draw the result into the video.

```
fps=0.0
#cv2.putText(resultImage, 'FPS: ' + str(int(fps)), (620, 30),cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 2)
for i in carLocation1.keys():
    if frameCounter % 1 == 0:
        [x1, y1, w1, h1] = carLocation1[i]
        [x2, y2, w2, h2] = carLocation2[i]
        #print 'previous location: ' + str(carLocation1[i]) + ', current location: ' + str(carLocation2[i])
        carLocation1[i] = [x2, y2, w2, h2]

        # print 'new previous location: ' + str(carLocation1[i])
        if [x1, y1, w1, h1] != [x2, y2, w2, h2]:
            result = False
            roi = resultImage[y1:y1+h1,x1:x1+w1]
            if types[i]=="bikes" and Helmets[i] == "No Helmet Detected" and identity[i]< OPTIMISE:
                result = helm.detect(roi)
            if result==True:
                Helmets[i]= "Helmet Detected"
        #   if y1 >= 275 and y1 <= 285:
            if 7==7:
                if not (end_time == start_time):
                    fps = 1.0/(end_time - start_time)
                speed[i] = estimateSpeed([x1, y1, w1, h1], [x2, y2, w2, h2],fps)
                #print(str(speed[i]))
            #if y1 > 275 and y1 < 285:
            if int(speed[i])>40:
                speed[i]= speed[i]%40
            if go[i] == True and int(speed[i])<10:
                speed[i]=speed[i]+15
            if int(speed[i])==0:
                continue
            if identity[i]%LAG==0:
                if int(speed[i])>30:
                    go[i]=True
                #if we want to find overspeeding speed, print speed[i]
                    cv2.putText(resultImage, "OverSpeeding ALERT", (int(x1 + w1/2), int(y1-5)),cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 2)
                elif speed[i] != None and y1 >= 180 and speed[i]!=0:
                    ans= str(int(speed[i])) + " km/hr "
                    if types[i]=="bikes":
                        ans= ans+ Helmets[i]
                    cv2.putText(resultImage, ans, (int(x1 + w1/2), int(y1-5)),cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 255, 0), 2)
            identity[i]+=1
```

3. Experiments

In the evaluation process, various parameters and video inputs were examined to assess the system's performance and robustness under different conditions.

3.1 Speed Threshold Setting

The system's speed estimation component was set to detect overspeeding violations based on a predefined speed threshold. In the experiments, this

threshold was set to 40 km/h, which is a common speed limit in urban areas. By setting this threshold, the system could accurately identify vehicles exceeding the designated speed limit, mimicking real-world traffic enforcement scenarios.

3.2 Video Input Selection:

The performance of the system was evaluated using several video inputs, with a particular emphasis on high-resolution videos. The choice of high-resolution ones was important for two reasons:

A. Helmet Detection Accuracy:

The helmet detection model performs a lot better in input video with higher resolution. Higher resolution videos provide more detailed information, enabling the model to identify and localize helmets worn by motorcycle riders more accurately. In addition, our model is trained with high resolution images, that could also be a reason.

For example, in this low resolution video, our program incorrectly says this person doesn't wear a helmet.

Figure 1. Man wears a helmet but the program says he doesn't

B. Object Tracking Robustness:

High-resolution videos also contribute to robustness of the object tracking component. With more visual information available, the Deep SORT algorithm can maintain consistent object identities and trajectories, even in challenging scenarios where objects may undergo occlusions or significant changes in appearance.

3.3 Parameter Tuning:

To optimize the system's performance, several parameters were fine-tuned during the experiments:

A. Non-Maximum Suppression (NMS) Threshold:

A critical parameter in object detection process, which determines the level of overlap allowed between detected bounding boxes for them to be considered separate objects. Adjusting this threshold can impact the system's ability to accurately detect and distinguish between closely spaced objects.

B. Tracking Confidence Thresholds:

Relied by Deep SORT algorithm to initiate and maintain object tracks. These thresholds were tuned to strike a balance between track stability and ability to detect new objects entering the scene.

C. Frame Sampling Rate:

The system's processing speed and computational requirements are influenced by the frame sampling rate. During the experiments, different frame sampling rates were evaluated to find an optimal trade-off between real-time performance and accurate violation detection.

4. Results and Analysis

The result is like the following images, it effectively identifies multiple infractions, including speeding and the absence of helmets among motorcyclists.

The output video showcases real-time detection capabilities, with violators distinctly flagged—vehicles exceeding speed limits are marked with an "OverSpeeding ALERT" notification, while helmetless riders are identified with a "No Helmet Detected" message. Furthermore, the system provides velocity estimates for all detected vehicles, offering a thorough analysis of traffic conduct.

As you can see in the screenshot of this output video processed with this system(Figure 2), the man in the middle apparently rides the motorcycle with a suitable speed in an urban area.

You can check it in the original video,
link: https://www.youtube.com/watch?v=rob0av_XNes.

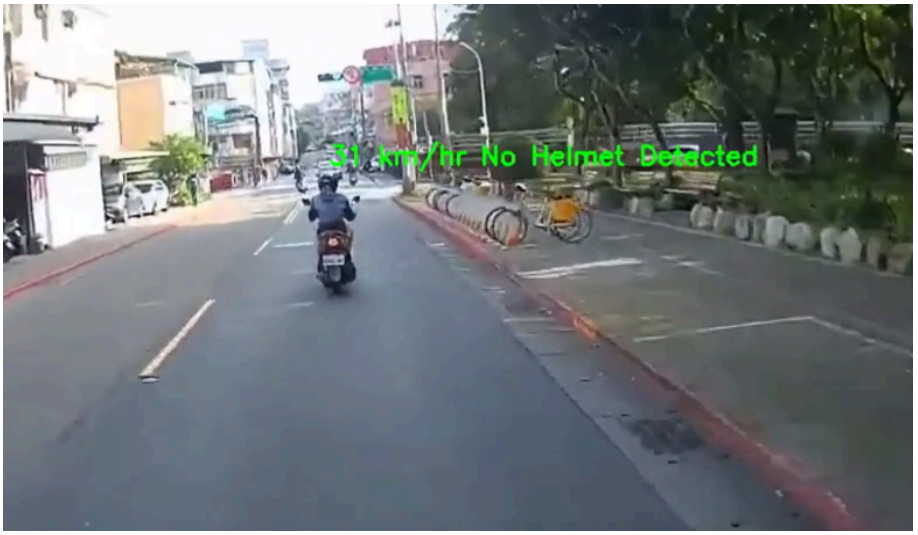

Figure 2. Man with suitable speed

The system shows his current velocity, which can help if there's any accident that happens to those following the traffic rules. Since it does identify the speed of moving objects well, those overspeeding will also be detected. Once people are detected driving or riding with more than speed of 40km/h, they will be marked with an overspeeding alert message(Figure 3).

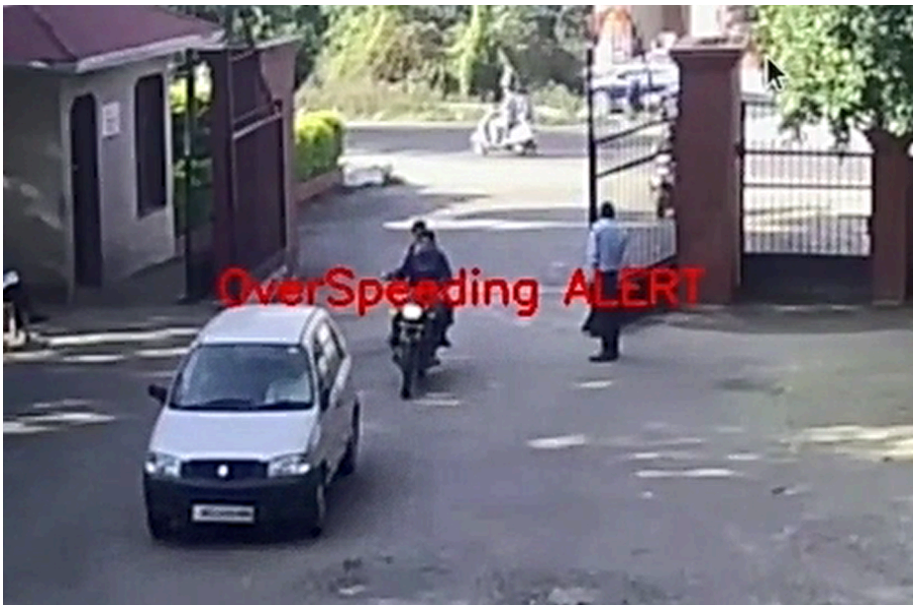

Figure 3 Overspeeding alert triggered

However for helmet detection, there are problems sometimes. In the first output video, it correctly points out the man at the pillion seat wearing no helmet and marks him with the corresponding messages(Figure 4). But in some other lower resolution recordings we got, it doesn't work well. We infer that it's because we use high-resolution images to train the mode.

Besides, as Figure 5 shows, the man wearing black helmet is misidentified as a wearing-no-helmet guy. Maybe it's because his helmet color is similar to the color of people's hair in our images for training the model.

Figure 4 Man at the pillow seat doesn't wear a helmet

Figure 5 The man wearing black helmet but identified as wearing no helmet

5. Conclusions

5.1 Current Achievements

The system can now successfully integrate computer vision techniques and deep learning algorithms to automatically identify vehicles exceeding speed limits and riders not wearing helmets. The system's key achievements include:

Real-Time Object Detection and Tracking:

The system leverages Haar cascades and Deep SORT to detect and track multiple vehicles and pedestrians simultaneously, across video frames.

Speed Estimation:

By analyzing the displacement of tracked objects between consecutive frames, the system can estimate their speeds, enabling the detection of overspeeding violations.

Helmet Violation Detection:

The integration of a custom YOLOv3-based helmet detection model allows the system to identify whether motorcycle riders are wearing helmets or not, facilitating the detection of helmet violations.

Violation Visualization:

The system provides visual feedback by displaying messages about detected violations near the detected target.

5.2 Limitations

While the current implementation showcases the system's potential, it is subject to certain limitations:

Limited Violation Types:

The system is currently restricted to detecting overspeeding and helmet violations, limiting its applicability to other types of traffic violations.

Performance Constraints:

The system's performance may degrade when handling high traffic densities or complex scenes, as the number of objects to track increases.

Helmet Detection Accuracy:

The helmet detection model's accuracy is dependent on the resolution of the

input video, with lower resolutions potentially leading to inaccurate results. We think it's because we use high-resolution images to train the model initially.

5.3 Future Work

To address the limitations and further enhance the capabilities of the traffic violation detection system, the following future work can be considered:

Support for Additional Violation Types:

Extending the system to detect and handle other types of traffic violations, such as running red lights, illegal lane changes, or driving on the wrong side of the road.

Improved Helmet Detection:

Enhance helmet detection in low-resolution videos by testing new architectures, larger datasets, or more features. Optimize real-time performance with parallelization to handle denser traffic without losing accuracy.

Comprehensive Violation Logging and Reporting:

Develop user-friendly interfaces and create APIs to integrate with existing traffic systems, enabling good data exchange.

6. What we have Learned

Through this final project, we gained more experience with computer vision and machine learning applications. It's quite fun to try finding a solution to the long-standing problem of traffic in Taiwan. By implementing and fine-tuning algorithms, we developed a system capable of detecting overspeeding and helmetless riding, despite the fact that the latter still has some problems.

This project enhanced our theoretical knowledge and provided practical skills in building and optimizing real-time detection systems, demonstrating the potential for real-world applications in improving road safety and traffic regulation. Hope to improve the existing functions in this system and make it able to detect more kinds of traffic-violation.

7. Reference

https://github.com/ben-haim/Motobike-Detect
https://github.com/meetvora1308/cars-detection-using-opencv
https://github.com/xiaochus/YOLOv3
https://github.com/eric612/Vehicle-Detection
https://github.com/theAIGuysCode/yolov3_deepsort
https://github.com/GeekAlexis/FastMOT

8. Contribution of each group members

0816199	陳清海	33.3%
109550055	李耕雨	33.3%
109550135	范恩宇	33.3%