Fire Detection

Computer Vision &
Pattern Recognition Lab
kit 금오공과대학교
Kumoh National Institute of Technology

Procedure

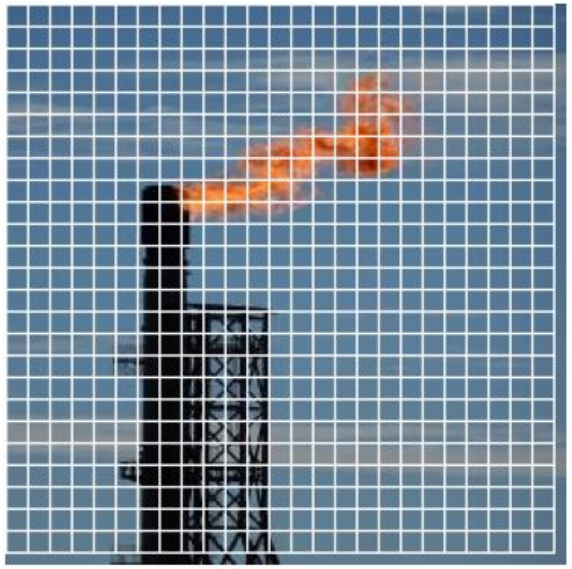
Algorithm 1 Processing Pipeline for Fire Detection

Input:
`video_pth` (path to video)
`output_pth` (path to video output)
`PATCH_WIDTH, PATCH_HEIGHT, STRIDE_X, STRIDE_Y` (patch parameters)
`model, THRESHOLD, device, inference_transform` (for fire detection)
Output: Video saved at output_pth with overlaid fire masks
frames ← GENFRAMES(video_pth)
masks ← empty list
for all each image `img` in `frames` **do**
    (*patches*, *patch_coords*) ← GEN_PATCHES(img, PATCH_WIDTH, PATCH_HEIGHT, STRIDE_X, STRIDE_Y)
    for all each index i and patch in `patches` **do**
        Convert `patch` from RGB to BGR
        patch ← GMM_FIRE_DETECTION(img = patch, model_path = "gmm/fire_gmm_lab.pkl", threshold = 1×10^{-6})
        (*patches*[i], _) ← CENTER_FIRE_PATCH(patch, patch)
    end for
    mask ← PROCESS_PATCHES(patches, model, THRESHOLD, device, inference_transform)
    Convert `mask` from BGR to RGB
    overlay ← CV2.ADDWEIGHTED(img, 0.5, mask, 0.7, 0)
    Append *overlay* to `masks`
end for
SAVE_VIDEO(masks, output_pth, fps = 24, codec = 'mp4v')

Procedure

Algorithm 2 gen_patches

```
function GEN_PATCHES(img, PATCH_WIDTH, PATCH_HEIGHT, STRIDE_X, STRIDE_Y)
    (img_h, img_w) ← dimensions of img
    patches ← empty list
    patch_coords ← empty list
    for top_y ← 0 to img_h − PATCH_HEIGHT step STRIDE_Y do
        for left_x ← 0 to img_w − PATCH_WIDTH step STRIDE_X do
            patch_ymin ← top_y
            patch_xmin ← left_x
            patch_ymax ← top_y + PATCH_HEIGHT
            patch_xmax ← left_x + PATCH_WIDTH
            patch ← copy of img[patch_ymin:patch_ymax, patch_xmin:patch_xmax]
            Append patch to patches
            Append (patch_ymin, patch_xmin) to patch_coords
        end for
    end for
    return patches, patch_coords
end function
```



Procedure

Algorithm 3 gmm_fire_detection

```
function GMM_FIRE_DETECTION(img_patch, model_path, threshold)
    Open file model_patch in read-binary mode as f
    gmm ← PICKLE.LOAD(f)
    Close file f
    lab ← CV2.CVTCOLOR(img_patch, cv2.COLOR_BGR2Lab)
    (h, w) ← dimensions of lab
    lab_flat ← reshape(lab, (−1, 3)) and convert to float64
    log_likelihood ← GMM.SCORE_SAMPLES(lab_flat)
    likelihood ← exp(log_likelihood)
    fire_mask_flat ← zero array of length (h × w), type uint8
    for each index i from 0 to length(lab_flat)-1 do
        if likelihood[i] > threshold then
            fire_mask_flat[i] ← 255
        end if
    end for
    fire_mask ← reshape(fire_mask_flat, (h, w))
    return fire_mask
end function
```

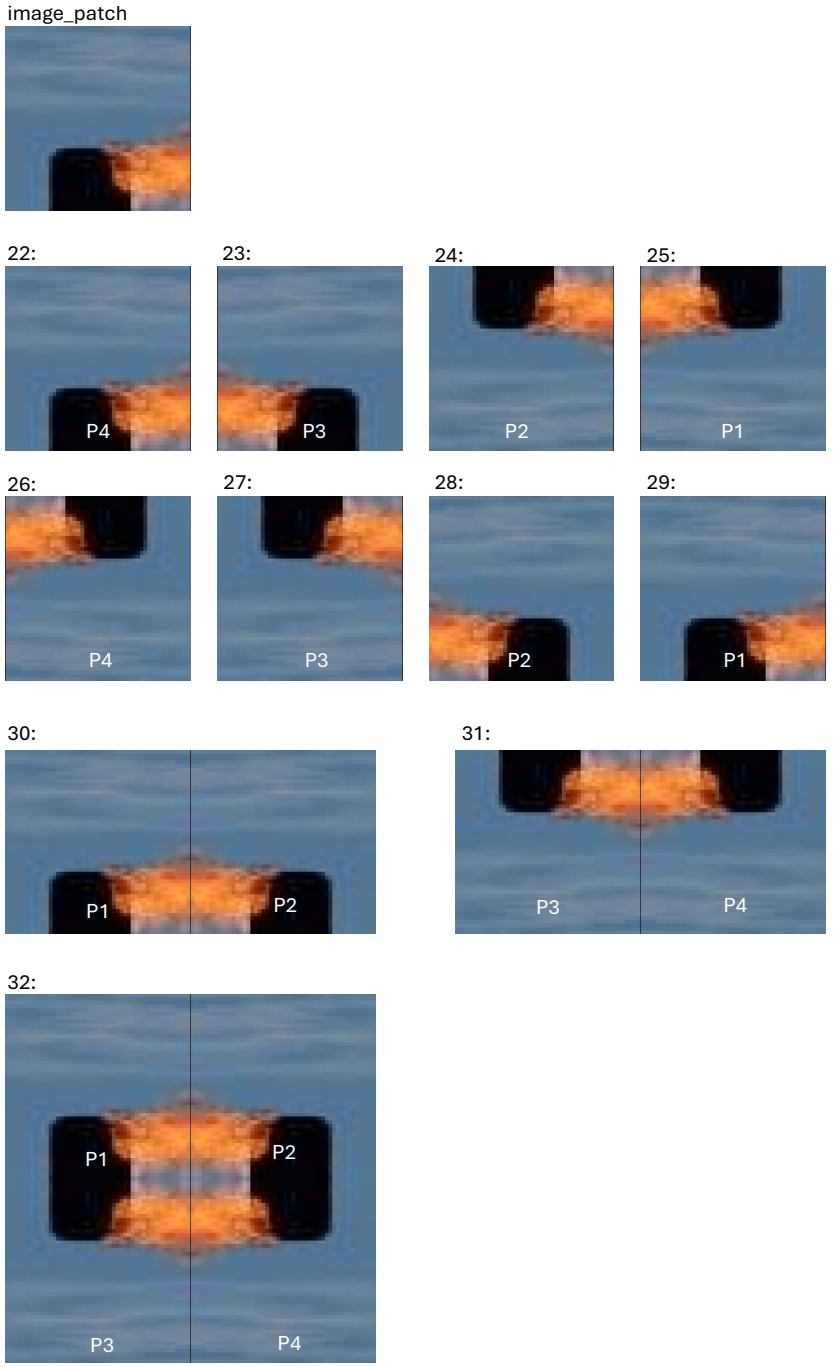
Image patch

fire_mask

Procedure

Algorithm 4 Center-Fire-Patch

```
function CENTER_FIRE_PATCH(image_patch, fire_mask)
    (H, W) ← dimensions of image_patch
    indices ← positions in fire_mask where value > 0
    if indices is empty then
        (cy, cx) ← (H // 2, W // 2)
    else
        (cy, cx) ← mean value of indices (average row and column)
    end if
    center_y ← H/2
    center_x ← W/2
    dx ← cx − center_x                    ▷ positive if fire is to the right
    dy ← cy − center_y                    ▷ positive if fire is below
    if dx < 0 ∧ dy ≥ 0 then
        base_image ← flip image horizontally
    else if dx ≥ 0 ∧ dy < 0 then
        base_image ← flip image vertically
    else if dx < 0 ∧ dy < 0 then
        base_image ← flip image horizontally then vertically
    else
        base_image ← copy of image
    end if
    p4 ← base_image
    p3 ← flip p4 horizontally
    p2 ← flip p4 vertically
    p1 ← flip p3 vertically
    p1 ← flip p1 vertically then horizontally
    p2 ← flip p2 vertically then horizontally
    p3 ← flip p3 vertically then horizontally
    p4 ← flip p4 vertically then horizontally
    top_row ← concatenate horizontally (p1, p2)
    bottom_row ← concatenate horizontally (p3, p4)
    centered_patch ← concatenate vertically (top_row, bottom_row)
    return centered_patch
end function
```

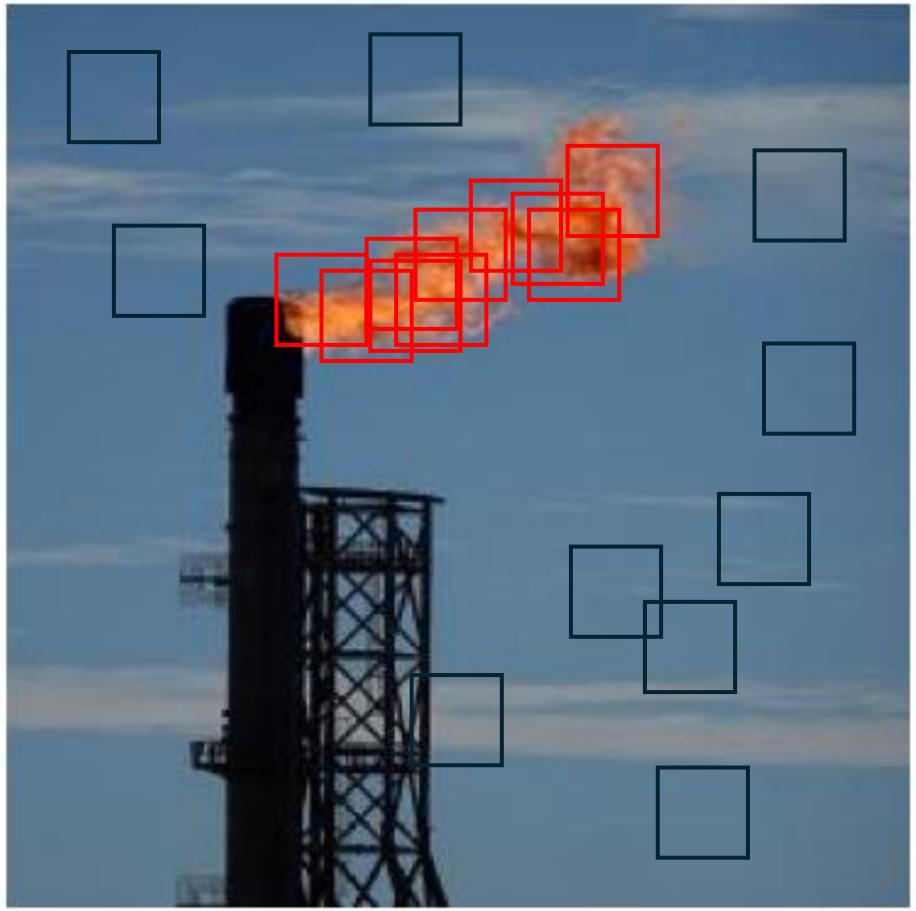


Procedure

Algorithm 5 process_patches

function PROCESS_PATCHES(patches, model, thres, transform, device)
    $(img\_h, img\_w) \leftarrow$ dimensions of *img*
    mask $\leftarrow$ black image of size $(img\_h, img\_w, 3)$
    patch_masks $\leftarrow$ empty list
    MODEL.EVAL ▷ Set model to evaluation mode
    for all each patch (*patch_img*) in *patches* **do**
        if transform is provided **then**
            patch_tensor $\leftarrow$ TRANSFORM(patch_img)
        end if
        logits $\leftarrow$ MODEL(patch_tensor) ▷ fire classification model
        probs $\leftarrow$ softmax(logits)
        fire_prob $\leftarrow$ fire probability from *probs*
        if fire_prob $\geq$ thres **then**
            $\text{normalized\_conf} \leftarrow 0.5 \times \frac{(fire\_prob - thres)}{(1.0 - thres)}$
            normalized_conf $\leftarrow$ clamp(normalized_conf, 0.0, 1.0)
            intensity_value $\leftarrow \lfloor 255 \times normalized\_conf \rfloor$
            patch_mask $\leftarrow (0,\ 0,\ intensity\_value)$
            Append patch_mask to patch_masks
        end if
    end for
    return patch_masks
end function

Model Training



For each image make patches

if fire area > threshold

add to fire_patches list

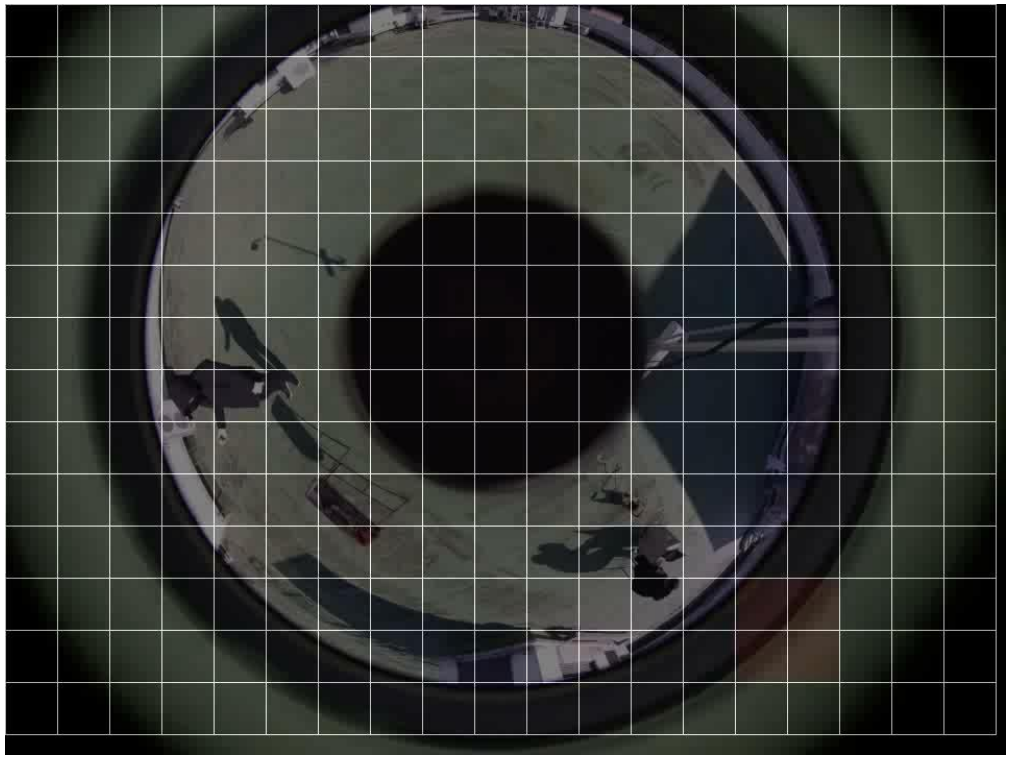
else

add to non-fire_patches list

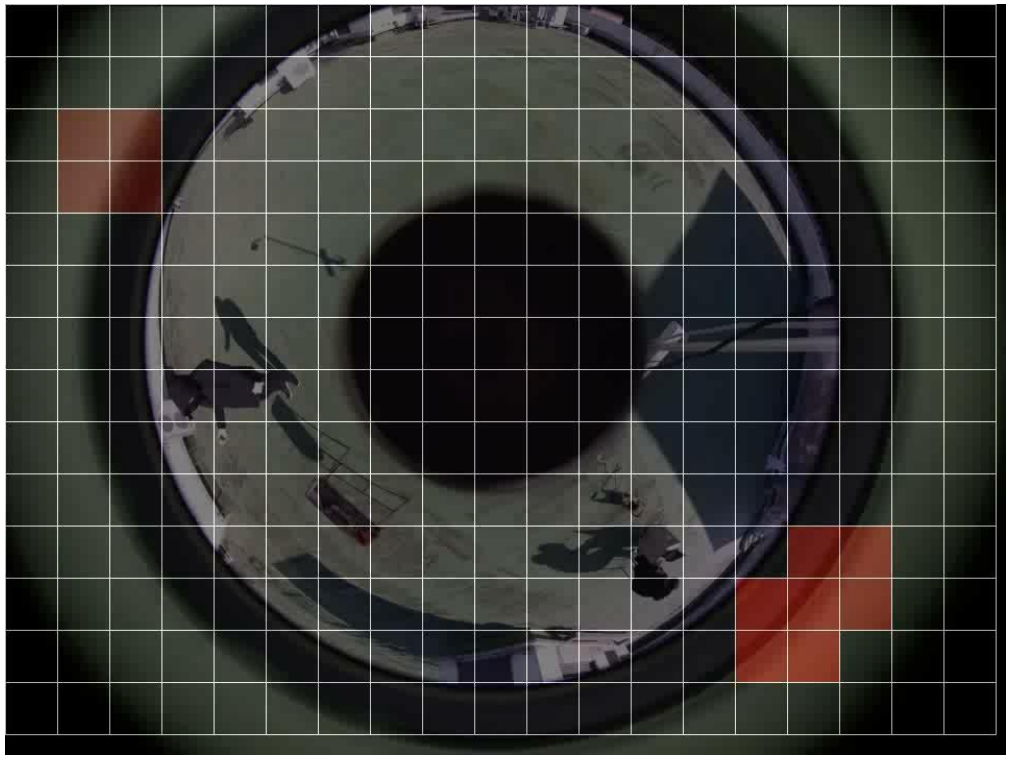
Randomly pick and save 10 fire patches and 10 non-fire images

Train classification model on fire and non-fire patches

Results



With centering fire patches



Without centering fire patches