- **GitHub：https://github.com/dodo0517cc/VRDL_HW2**

- **Reference：**

Scaled-YOLOv4—https://github.com/WongKinYiu/ScaledYOLOv4
Mish-Cuda－https://github.com/thomasbrandon/mish-cuda

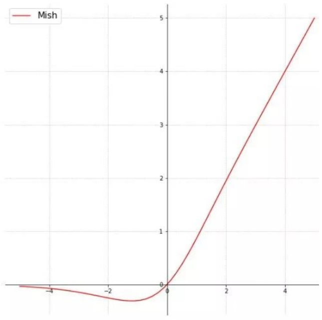


Figure 1. Mish Activation Function

We can see that positive values can reach any height to avoid saturation due to capping. The theoretically slight allowance for negative values allows for better gradient flow instead of hard zero boundaries like in ReLU. A smooth activation function allows better information to penetrate the neural network, resulting in better accuracy and generalization.

- **Brief introduction：**

First, we have to read the mat file and check the details of all the boxes and turn it to yolo format. Generate the files that yolo need. Then, put all the images, boxes and labels to ScaledYOLOv4 model. The most important thing is to tune the parameters. Finally train the images and test it with the best weight.

- **Methodology：**

✓ **Data pre-process：**

Read the .mat file：change it to boxes.csv, list the detail of boxes of each box.
Augmentation：
hsv_h（HSV-Hue augmentation）: 0.015
hsv_s（HSV-Saturation augmentation）: 0.7
hsv_v（HSV-Value augmentation）: 0.4
degrees（image rotation）: 20.0
scale（image scale）: 0.5

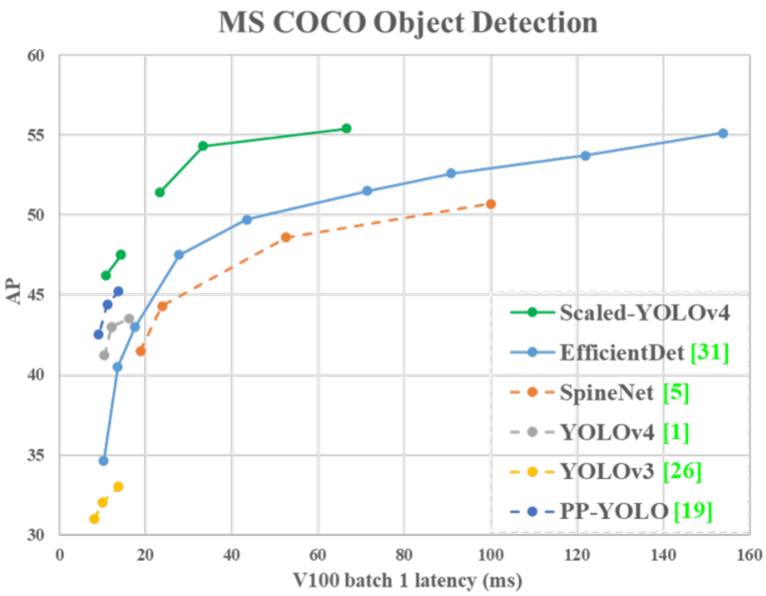
shear（image shear）: 10.0
perspective（image perspective）: 0.0008   range 0-0.001
mosaic: 1.0（probability）

✓ **Model architecture：**

Scaled YOLOv4—https://github.com/WongKinYiu/ScaledYOLOv4

Scaled-YOLOv4 was proposed on November 16, 2020 to improve YOLOv4.

- Designed a powerful model scaling method for small models, which systematically balances the computational cost and storage bandwidth of shallow CNN
- Design a simple and effective scaling strategy for large-scale target detectors
- Analyze the relationship between the scaling factors of each model, and scale the model based on the optimal group division
- Experiments confirmed that the FPN structure is essentially a one-off structure;
- Use the above methods to develop yolov4-tiny and yolo4v4-large



✓ **Hyperparameters：**

lr0: 0.01   # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.2   # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937   # SGD momentum/Adam beta1
weight_decay: 0.0005   # optimizer weight decay 5e-4
warmup_epochs: 3.0   # warmup epochs (fractions ok)
warmup_momentum: 0.8   # warmup initial momentum

warmup_bias_lr: 0.1 # warmup initial bias lr
box: 0.05 # box loss gain
cls: 0.3 # cls loss gain
cls_pw: 1.0 # cls BCELoss positive_weight
obj: 0.9 # obj loss gain (scale with pixels)
obj_pw: 0.9 # obj BCELoss positive_weight
iou_t: 0.20 # IoU training threshold
anchor_t: 4.0 # anchor-multiple threshold

- **Summary**

File in coco/annotations in mydrive:
instances_val2017.json

File in yolov4 in mydrive：
png_to_jpg.py
generate_txt.py
generate_train.py
generate_test.py
test.txt
boxes.csv
obj.zip
obj_test.zip
train_txt.zip
obj.names
weights－best.pt

Training:
Step1: Use GPU. Set up the environment.
Step2: Git clone the project：https://github.com/WongKinYiu/ScaledYOLOv4
Step3: Install torch==1.6.0+cu101, torchvision==0.7.0+cu101
Step4: Git clone https://github.com/thomasbrandon/mish-cuda, then install.
Step5: Update YAML
Step6（Done by png_to_jpg.py）: Turn the images from png file to jpg file and upload the zip file（obj.zip, obj_test.zip） of the images. Unzip it to the data folder that is in ScaledYOLOv4-yolov4-csp file.
Step7 (Done by generate_txt.py): Change labels to yolo format and save it to .txt file. The yolo format is standardized to 0~1 class, x_center, y_center, width, height. Upload the zip file（train_txt.zip） of all of the

txt flies. Unzip it to the data folder that is in ScaledYOLOv4-yolov4-csp file.
Step8: Copy gerenate_train.py to ScaledYOLOv4-yolov4-csp file. Then, run it. It will generate two files, train.txt and valid.txt, respectively.
Step9: Create a digits.yaml in the data folder, which stores the training set, validation set and test set paths, the number of categories and the category names
Step10: Modify cfg file. Copy a original cfg file and change the image width and height to 576, filters to 45(filters=(classes + 5)*3, and classes to 10.
Step11: Train

Inference：
Step1: Use GPU. Set up the environment.
!sudo apt update
!sudo apt install libgl1-mesa-glx -y
Step2: Git clone the project：https://github.com/WongKinYiu/ScaledYOLOv4
Step3: Install torch==1.6.0+cu101, torchvision==0.7.0+cu101
Step4: Git clone https://github.com/thomasbrandon/mish-cuda, then install.
Step5: Update YAML
Step6: Create a digits.yaml in the data folder, which stores the training set, validation set and test set paths, the number of categories and the category names
Step7: Modify cfg file. Copy a original cfg file and change the image width and height to 576, filters to 45(filters=(classes + 5)*3, and classes to 10.
Step8: Upload the zip file（obj_test.zip） of all of the test images. Unzip it to the data folder that is in ScaledYOLOv4-yolov4-csp file. Copy gerenate_test.py to ScaledYOLOv4-yolov4-csp file. Then, run it. It will generate test.txt.
Step9: Copy weights(best.pt) to ScaledYOLOv4-yolov4-csp file and obj.names to the data folder that is in ScaledYOLOv4-yolov4-csp file
Step10: Test
Colab link :
https://colab.research.google.com/drive/1ZydftPlARDwjBYslWqYspbx88jjIczGL?usp=sharing

```python
data_listdir = os.listdir("./data/test_jpg")
# Test your inference time
TEST_IMAGE_NUMBER = 100 # This number is fixed.
test_img_list = []

# Read image (Be careful with the image order)
data_listdir.sort(key = lambda x: int(x[:-4]))

with open("./data/test.txt", "w") as outfile:
  for img_name in data_listdir[:TEST_IMAGE_NUMBER]:
    img_path = os.path.join("/content/gdrive/MyDrive/ScaledYOLOv4-yolov4-csp/data/test_jpg", img_name)
    test_img_list.append(img_path)
float: start_time _path)
1637740581.7288082 ')

start_time = time.time()
# for img in tqdm(test_img_list):
    # your model prediction
!python test.py --img 576 --conf 0.001 --batch 8 --device 0 --data data/digits.yaml --names data/obj.names --cfg models/yolov4-csp_416.cfg --weights best.pt --task test
end_time  = time.time()
print("\nInference time per image: ", (end_time - start_time) / len(test_img_list))
```

```
Namespace(augment=False, batch_size=8, cfg='models/yolov4-csp_416.cfg', conf_thres=0.001, data='data/digits.yaml', device='0', exist_ok=False, img_size=576, iou_thres=0.65, name
Using torch 1.6.0+cu101 CUDA:0 (Tesla K80, 11441MB)

Model Summary: 516 layers, 52544487 parameters, 52544487 gradients
Scanning images: 100% 100/100 [00:00<00:00, 516.17it/s]
Scanning labels /content/gdrive/MyDrive/ScaledYOLOv4-yolov4-csp/data/test_txt.cache3 (0 found, 0 missing, 100 empty, 0 duplicate, for 100 images): 100it [00:00, 419850.25it/s]
WARNING: No labels found in /content/gdrive/MyDrive/ScaledYOLOv4-yolov4-csp/data/test_txt/. See https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 13/13 [00:06<00:00,  2.13it/s]
                 all         100           0           0           0           0           0
Speed: 44.1/2.0/46.0 ms inference/NMS/total per 576x576 image at batch-size 8
Results saved to runs/test/exp7

Inference time per image:  0.13941767454147339
```