Texture defect detection

Install Dependencies

```
# clone YOLOv5 repository
!git clone https://github.com/ultralytics/yolov5  # clone repo
%cd yolov5
!git reset --hard fbe67e465375231474a2ad80a4389efc77ecff99
```

```
Cloning into 'yolov5'...
remote: Enumerating objects: 12779, done.
remote: Counting objects: 100% (146/146), done.
remote: Compressing objects: 100% (105/105), done.
remote: Total 12779 (delta 83), reused 86 (delta 41), pack-reused 12633
Receiving objects: 100% (12779/12779), 12.96 MiB | 32.21 MiB/s, done.
Resolving deltas: 100% (8792/8792), done.
/content/yolov5
HEAD is now at fbe67e4 Fix `OMP_NUM_THREADS=1` for macOS (#8624)
```

```
# install dependencies as necessary
!pip install -qr requirements.txt  # install dependencies (ignore errors)
import torch

from IPython.display import Image, clear_output  # to display images
from utils.downloads import attempt_download  # to download models/datasets

# clear_output()
print('Setup complete. Using torch %s %s' % (torch.__version__, torch.cuda.get_device_prop
```

```
     |████████████████████████████████| 1.6 MB 38.1 MB/s
Setup complete. Using torch 1.12.1+cu113 _CudaDeviceProperties(name='Tesla T4', major
```

```
#follow the link below to get your download code from from Roboflow
!pip install -q roboflow
from roboflow import Roboflow
rf = Roboflow(model_format="yolov5", notebook="roboflow-yolov5")
```

```
     |████████████████████████████████| 145 kB 42.9 MB/s
     |████████████████████████████████| 178 kB 65.0 MB/s
     |████████████████████████████████| 1.1 MB 49.5 MB/s
     |████████████████████████████████| 67 kB 6.4 MB/s
     |████████████████████████████████| 54 kB 2.9 MB/s
     |████████████████████████████████| 138 kB 70.5 MB/s
     |████████████████████████████████| 62 kB 1.7 MB/s
  Building wheel for roboflow (setup.py) ... done
  Building wheel for wget (setup.py) ... done
upload and label your dataset, and get an API KEY here: https://app.roboflow.com/?mod
```

```
%cd /content/yolov5
#after following the link above, recieve python code with these fields filled in
########################################
#########################
```

```
from roboflow import Roboflow
rf = Roboflow(api_key="vmPqiZjOsUeOEeTUkYOX")
project = rf.workspace("uaf").project("texture-defect-detection")
dataset = project.version(2).download("yolov5")
```

```
/content/yolov5
loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in Texture-defect-detection-2 to yolov5pytorch: 100%
Extracting Dataset Version Zip to Texture-defect-detection-2 in yolov5pytorch:: 100%
```

```
# this is the YAML file Roboflow wrote for us that we're loading into this notebook with o
%cat {dataset.location}/data.yaml
```

```
names:
- Hole
- Stain
nc: 2
train: Texture-defect-detection-2/train/images
val: Texture-defect-detection-2/valid/images
```

Using GANs to generate images Dataset

```
from numpy import expand_dims
from numpy import zeros
from numpy import ones
from numpy import vstack
from numpy.random import randn
from numpy.random import randint
from keras.datasets.cifar10 import load_data
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Reshape
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import Conv2DTranspose
from keras.layers import LeakyReLU
from keras.layers import Dropout
from matplotlib import pyplot
```

```
def define_discriminator(in_shape=(32,32,3)):
    model = Sequential()
    model.add(Conv2D(64, (3,3), padding='same', input_shape=in_shape))
    model.add(LeakyReLU(alpha=0.2))
```

```
        model.add(LeakyReLU(alpha=0.2))
        model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Conv2D(256, (3,3), strides=(2,2), padding='same'))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Flatten())
        model.add(Dropout(0.4))
        model.add(Dense(1, activation='sigmoid'))
        opt = Adam(lr=0.0002, beta_1=0.5)
        model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
        return model

    def define_generator(latent_dim):
        model = Sequential()
        n_nodes = 256 * 4 * 4
        model.add(Dense(n_nodes, input_dim=latent_dim))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Reshape((4, 4, 256)))
        model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Conv2D(3, (3,3), activation='tanh', padding='same'))
        return model


    # Define GAN

    def define_gan(g_model, d_model):
        d_model.trainable = False
        model = Sequential()
        model.add(g_model)
        model.add(d_model)
        opt = Adam(lr=0.0002, beta_1=0.5)
        model.compile(loss='binary_crossentropy', optimizer=opt)
        return model

    def load_real_samples():
        (trainX, _), (_, _) = load_data()
        X = trainX.astype('float32')
        X = (X - 127.5) / 127.5
        return X

    def generate_real_samples(dataset, n_samples):
        ix = randint(0, dataset.shape[0], n_samples)
        X = dataset[ix]
        y = ones((n_samples, 1))
        return X, y

    def generate_latent_points(latent_dim, n_samples):
        x_input = randn(latent_dim * n_samples)
        x_input = x_input.reshape(n_samples, latent_dim)
```

```
    x_input = x_input.reshape(n_samples, latent_dim)
    return x_input

def generate_fake_samples(g_model, latent_dim, n_samples):
    x_input = generate_latent_points(latent_dim, n_samples)
    X = g_model.predict(x_input)
    y = zeros((n_samples, 1))
    return X, y


def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=200, n_batch=128):
    bat_per_epo = int(dataset.shape[0] / n_batch)
    half_batch = int(n_batch / 2)
    for i in range(n_epochs):
        for j in range(bat_per_epo):
            X_real, y_real = generate_real_samples(dataset, half_batch)
            d_loss1, _ = d_model.train_on_batch(X_real, y_real)
            X_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
            d_loss2, _ = d_model.train_on_batch(X_fake, y_fake)
            X_gan = generate_latent_points(latent_dim, n_batch)
            y_gan = ones((n_batch, 1))
            g_loss = gan_model.train_on_batch(X_gan, y_gan)
            print('>%d, %d/%d, d1=%.3f, d2=%.3f g=%.3f' %
                (i+1, j+1, bat_per_epo, d_loss1, d_loss2, g_loss))
        if (i+1) % 10 == 0:
            summarize_performance(i, g_model, d_model, dataset, latent_dim)

def summarize_performance(epoch, g_model, d_model, dataset, latent_dim, n_samples=150)
    X_real, y_real = generate_real_samples(dataset, n_samples)
    _, acc_real = d_model.evaluate(X_real, y_real, verbose=0)
    x_fake, y_fake = generate_fake_samples(g_model, latent_dim, n_samples)
    _, acc_fake = d_model.evaluate(x_fake, y_fake, verbose=0)
    print('>Accuracy real: %.0f%%, fake: %.0f%%' % (acc_real*100, acc_fake*100))
    save_plot(x_fake, epoch)
    filename = 'generator_model_%03d.h5' % (epoch+1)
    g_model.save(filename)

# Excecute
latent_dim = 100
d_model = define_discriminator()
g_model = define_generator(latent_dim)
gan_model = define_gan(g_model, d_model)
dataset = load_real_samples()
train(g_model, d_model, gan_model, dataset, latent_dim)
```

Define Model Configuration and Architecture

We will write a yaml script that defines the parameters for our model like the number of classes, anchors, and each layer.

```
# define number of classes based on YAML
import yaml
```

```
with open(dataset.location + "/data.yaml", 'r') as stream:
    num_classes = str(yaml.safe_load(stream)['nc'])



#this is the model configuration we will use for our tutorial
%cat /content/yolov5/models/yolov5s.yaml

    # YOLOv5 🚀 by Ultralytics, GPL-3.0 license

    # Parameters
    nc: 80  # number of classes
    depth_multiple: 0.33  # model depth multiple
    width_multiple: 0.50  # layer channel multiple
    anchors:
      - [10,13, 16,30, 33,23]  # P3/8
      - [30,61, 62,45, 59,119]  # P4/16
      - [116,90, 156,198, 373,326]  # P5/32

    # YOLOv5 v6.0 backbone
    backbone:
      # [from, number, module, args]
      [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
       [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
       [-1, 3, C3, [128]],
       [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
       [-1, 6, C3, [256]],
       [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
       [-1, 9, C3, [512]],
       [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
       [-1, 3, C3, [1024]],
       [-1, 1, SPPF, [1024, 5]],  # 9
      ]

    # YOLOv5 v6.0 head
    head:
      [[-1, 1, Conv, [512, 1, 1]],
       [-1, 1, nn.Upsample, [None, 2, 'nearest']],
       [[-1, 6], 1, Concat, [1]],  # cat backbone P4
       [-1, 3, C3, [512, False]],  # 13

       [-1, 1, Conv, [256, 1, 1]],
       [-1, 1, nn.Upsample, [None, 2, 'nearest']],
       [[-1, 4], 1, Concat, [1]],  # cat backbone P3
       [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

       [-1, 1, Conv, [256, 3, 2]],
       [[-1, 14], 1, Concat, [1]],  # cat head P4
       [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

       [-1, 1, Conv, [512, 3, 2]],
       [[-1, 10], 1, Concat, [1]],  # cat head P5
       [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

       [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
      ]


#customize iPython writefile so we can write variables
```

```
from IPython.core.magic import register_line_cell_magic

@register_line_cell_magic
def writetemplate(line, cell):
    with open(line, 'w') as f:
        f.write(cell.format(**globals()))


%%writetemplate /content/yolov5/models/custom_yolov5s.yaml

# parameters
nc: {num_classes}  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, BottleneckCSP, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, BottleneckCSP, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, BottleneckCSP, [1024, False]],  # 9
  ]

# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, BottleneckCSP, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, BottleneckCSP, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, BottleneckCSP, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, BottleneckCSP, [1024, False]],  # 23 (P5/32-large)
```

```
   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

Train Custom YOLOv5s Detector

```
# train yolov5s on custom data for 100 epochs
# time its performance
%%time
%cd /content/yolov5/
!python train.py --img 416 --batch 32  --epochs 100 --data {dataset.location}/data.yaml --
```

```
      Epoch   gpu_mem       box       obj       cls    labels  img_size
      55/99     3.25G   0.07769    0.0197  0.009603        25       416: 100% 5/5 [(
                Class     Images     Labels          P          R     mAP@.5 mAP@.5
                  all         17         27      0.116      0.456      0.128     0.(

      Epoch   gpu_mem       box       obj       cls    labels  img_size
      56/99     3.25G   0.07929   0.01928  0.008197        26       416: 100% 5/5 [(
                Class     Images     Labels          P          R     mAP@.5 mAP@.5
                  all         17         27      0.105      0.338       0.16     0.(

      Epoch   gpu_mem       box       obj       cls    labels  img_size
      57/99     3.25G   0.07968   0.01901  0.008257        23       416: 100% 5/5 [(
                Class     Images     Labels          P          R     mAP@.5 mAP@.5
                  all         17         27     0.0629      0.376     0.0459     0.(

      Epoch   gpu_mem       box       obj       cls    labels  img_size
      58/99     3.25G   0.07672   0.02141  0.007242        26       416: 100% 5/5 [(
                Class     Images     Labels          P          R     mAP@.5 mAP@.5
                  all         17         27      0.607        0.1      0.064     0.(

      Epoch   gpu_mem       box       obj       cls    labels  img_size
      59/99     3.25G   0.07706   0.02099  0.006543        35       416: 100% 5/5 [(
                Class     Images     Labels          P          R     mAP@.5 mAP@.5
                  all         17         27      0.605       0.15     0.0798      0

      Epoch   gpu_mem       box       obj       cls    labels  img_size
      60/99     3.25G   0.07908   0.02042  0.005591        42       416: 100% 5/5 [(
                Class     Images     Labels          P          R     mAP@.5 mAP@.5
                  all         17         27      0.818       0.05     0.0682     0.(

      Epoch   gpu_mem       box       obj       cls    labels  img_size
      61/99     3.25G   0.07435   0.02021  0.005686        32       416: 100% 5/5 [(
                Class     Images     Labels          P          R     mAP@.5 mAP@.5
                  all         17         27      0.147      0.468      0.153     0.(

      Epoch   gpu_mem       box       obj       cls    labels  img_size
      62/99     3.25G   0.07641   0.01896   0.00654        36       416: 100% 5/5 [(
                Class     Images     Labels          P          R     mAP@.5 mAP@.5
                  all         17         27      0.632       0.15     0.0859     0.(

      Epoch   gpu_mem       box       obj       cls    labels  img_size
      63/99     3.25G   0.07493    0.0181  0.005113        27       416: 100% 5/5 [(
                Class     Images     Labels          P          R     mAP@.5 mAP@.5
```

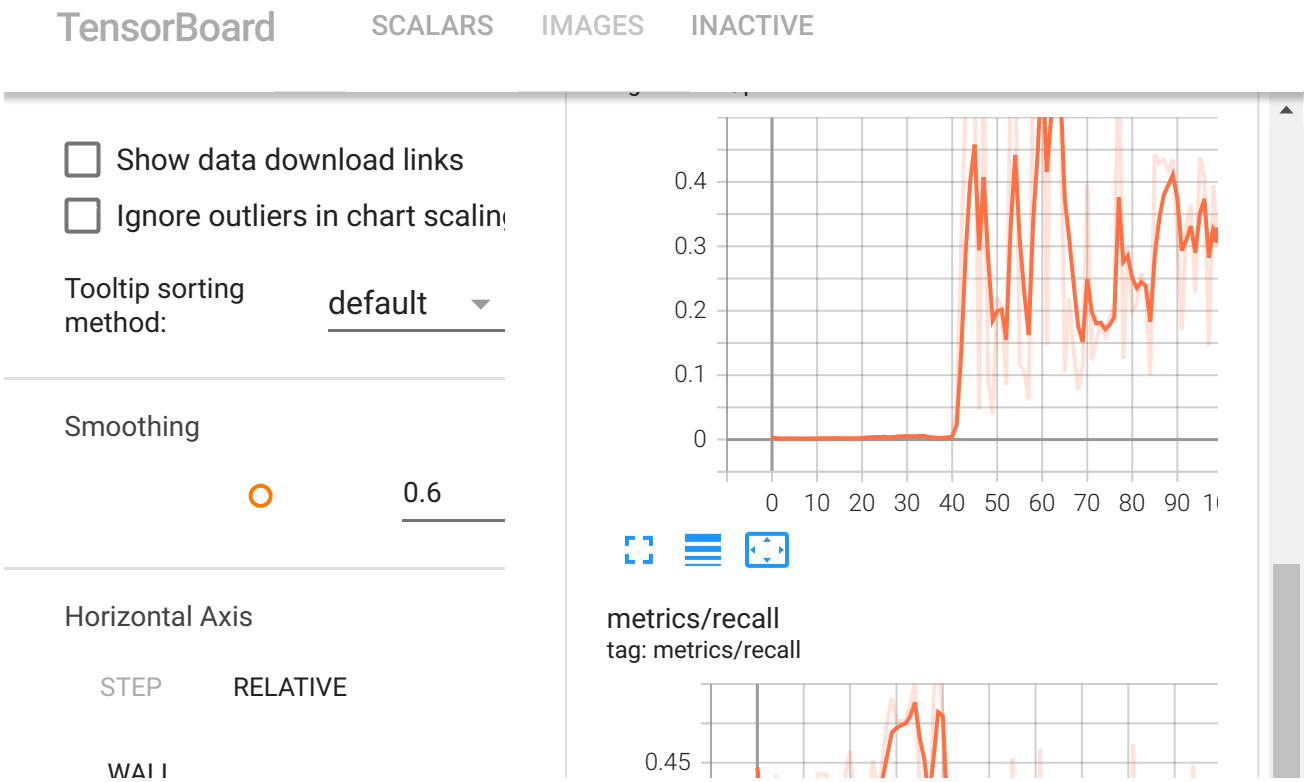
```
               all         17         27      0.587       0.15     0.0415     0.0

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     64/99     3.25G   0.07549   0.01941  0.005421        36       416: 100% 5/5 [0
               Class     Images     Labels          P          R     mAP@.5 mAP@.5
                 all         17         27      0.579       0.05     0.0567      0

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     65/99     3.25G   0.07351   0.01751  0.004409        26       416: 100% 5/5 [0
               Class     Images     Labels          P          R     mAP@.5 mAP@.5
                 all         17         27      0.105        0.1      0.069     0.0

     Epoch   gpu_mem       box       obj       cls    labels  img_size
```

Evaluate Custom YOLOv5 Detector Performance

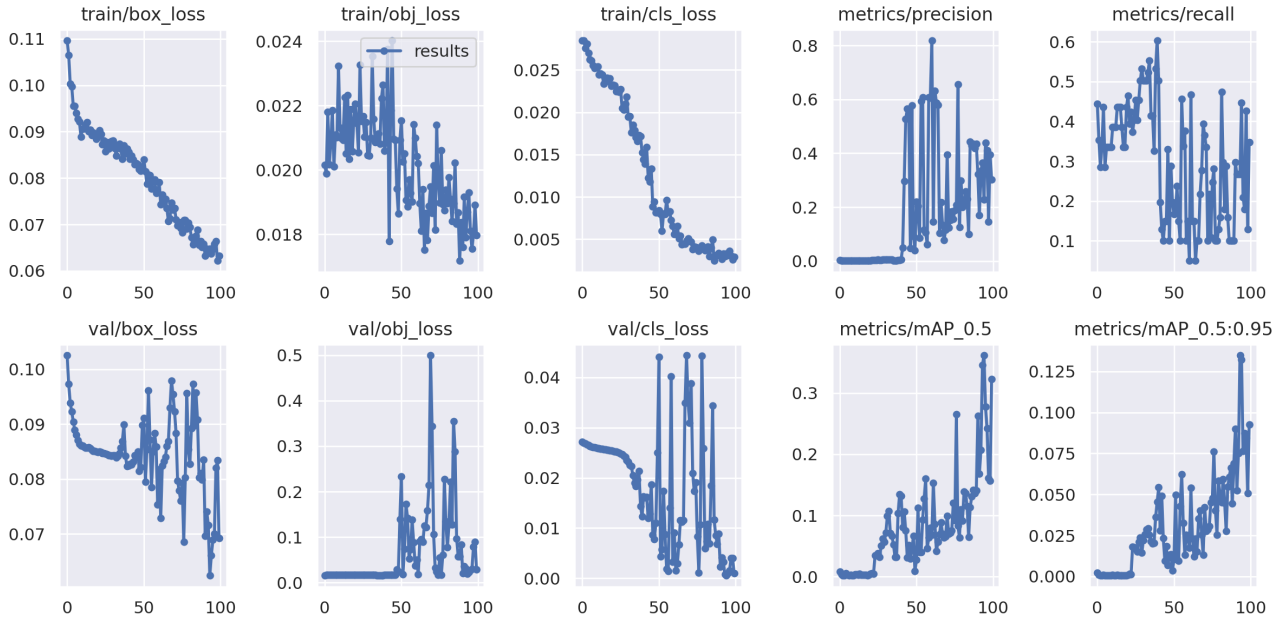
Training losses and performance metrics are saved to Tensorboard and also to a logfile defined above with the **--name** flag when we train. In our case, we named this `yolov5s_results`. (If given no name, it defaults to `results.txt`.) The results file is plotted as a png after training completes.

Note from Glenn: Partially completed `results.txt` files can be plotted with `from utils.utils import plot_results; plot_results()`.

```
# Start tensorboard for visulization
# logs save in the folder "runs"
%load_ext tensorboard
%tensorboard --logdir runs
```

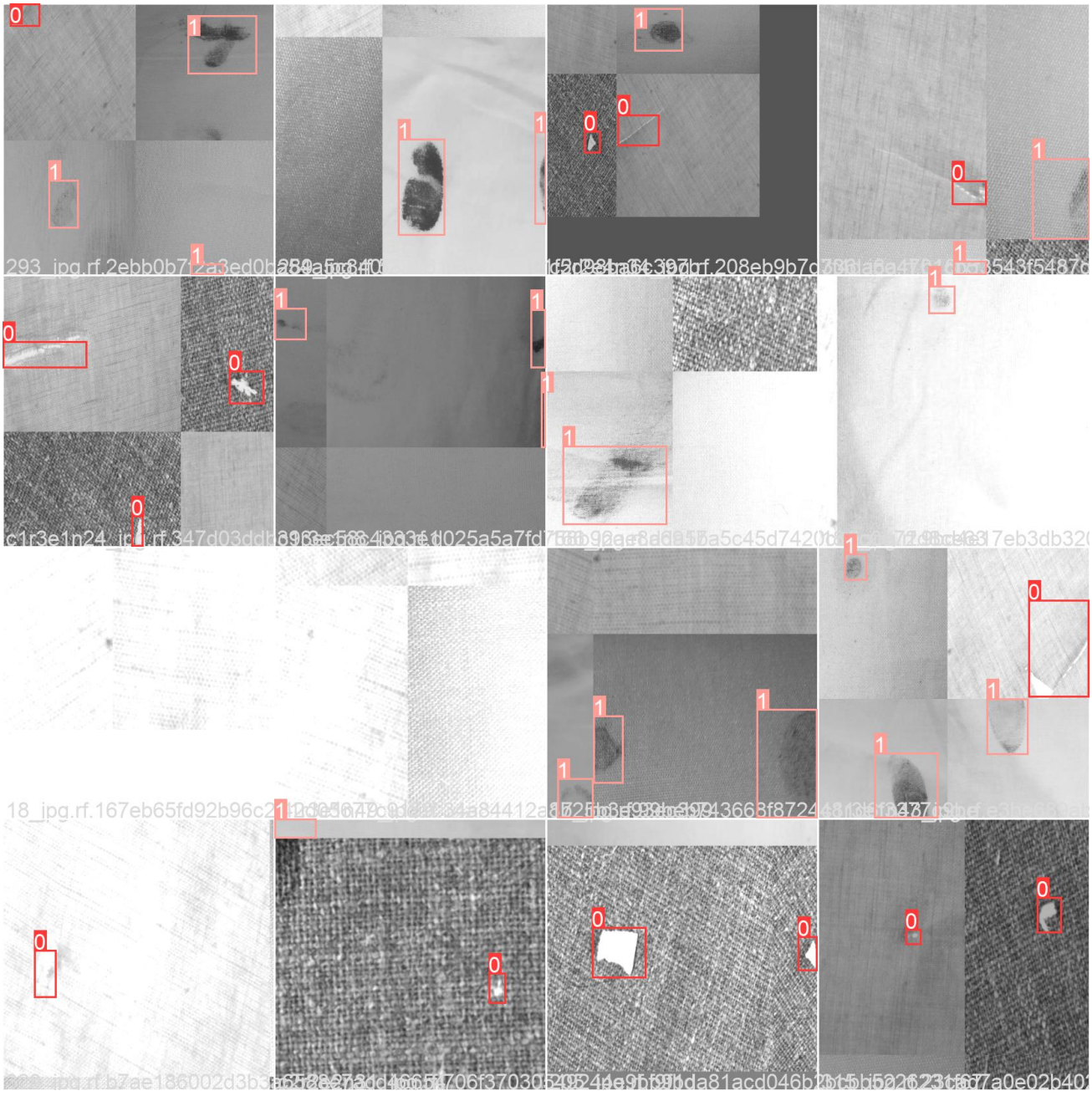



```
# we can also output some older school graphs if the tensor board isn't working for whatev
from utils.plots import plot_results  # plot results.txt as results.png
Image(filename='/content/yolov5/runs/train/yolov5s_results/results.png', width=1000)  # vi
```



```
# print out an augmented training example
print("GROUND TRUTH AUGMENTED TRAINING DATA:")
Image(filename='/content/yolov5/runs/train/yolov5s_results/train_batch0.jpg', width=900)
```

```
GROUND TRUTH AUGMENTED TRAINING DATA:
```



Run Inference With Trained Weights

Run inference with a pretrained checkpoint on contents of `test/images` folder downloaded from Roboflow.

```
# trained weights are saved by default in our weights folder
%ls runs/
```

```
train/
```

```
%ls runs/train/yolov5s_results3/weights
```

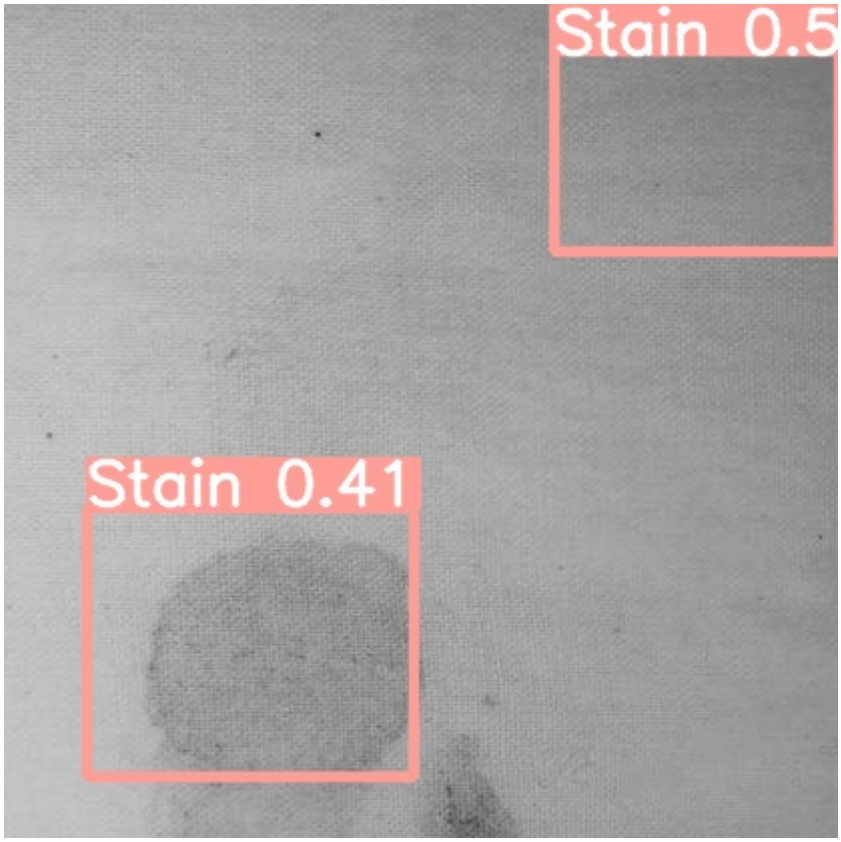
```
ls: cannot access 'runs/train/yolov5s_results3/weights': No such file or directory
```

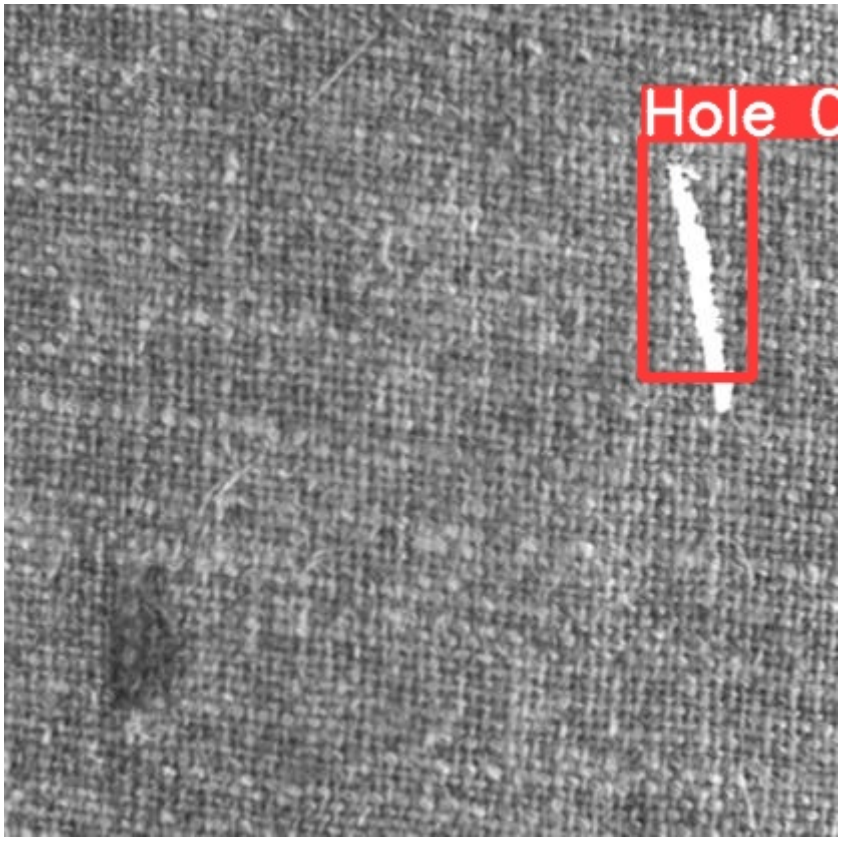
```
# use the best weights!
%cd /content/yolov5/
!python detect.py --weights runs/train/yolov5s_results/weights/best.pt --img 416 --conf 0.
```

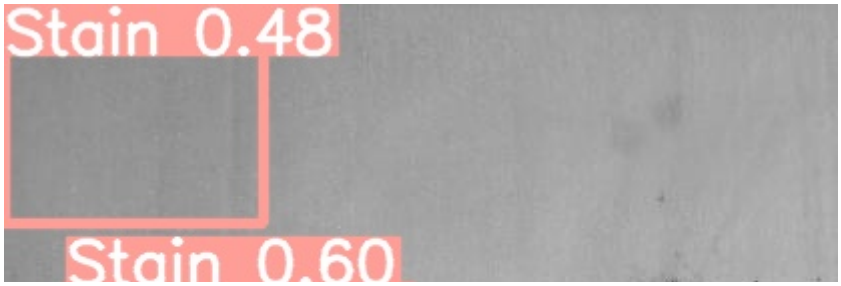
```
/content/yolov5
detect: weights=['runs/train/yolov5s_results/weights/best.pt'], source=/content/yolov
YOLOv5 🚀 v6.1-306-gfbe67e4 Python-3.7.14 torch-1.12.1+cu113 CUDA:0 (Tesla T4, 15110

Fusing layers...
custom_YOLOv5s summary: 232 layers, 7249215 parameters, 0 gradients, 16.7 GFLOPs
image 1/22 /content/yolov5/Texture-defect-detection-2/test/images/160_jpg.rf.8c694ed0
image 2/22 /content/yolov5/Texture-defect-detection-2/test/images/169_jpg.rf.77e1ac97
image 3/22 /content/yolov5/Texture-defect-detection-2/test/images/194_jpg.rf.81f9f346
image 4/22 /content/yolov5/Texture-defect-detection-2/test/images/199_jpg.rf.22d1c5c9
image 5/22 /content/yolov5/Texture-defect-detection-2/test/images/217_jpg.rf.c1f1e7dc
image 6/22 /content/yolov5/Texture-defect-detection-2/test/images/218_jpg.rf.1ba514de
image 7/22 /content/yolov5/Texture-defect-detection-2/test/images/221_jpg.rf.491a4999
image 8/22 /content/yolov5/Texture-defect-detection-2/test/images/286_jpg.rf.3e460922
image 9/22 /content/yolov5/Texture-defect-detection-2/test/images/287_jpg.rf.246cf1d6
image 10/22 /content/yolov5/Texture-defect-detection-2/test/images/290_jpg.rf.e3a1962
image 11/22 /content/yolov5/Texture-defect-detection-2/test/images/2_jpg.rf.0da4597db
image 12/22 /content/yolov5/Texture-defect-detection-2/test/images/301_jpg.rf.5be115a
image 13/22 /content/yolov5/Texture-defect-detection-2/test/images/98_jpg.rf.977ba878
image 14/22 /content/yolov5/Texture-defect-detection-2/test/images/c1r3e1n16_jpg.rf.3
image 15/22 /content/yolov5/Texture-defect-detection-2/test/images/c1r3e1n2_jpg.rf.94
image 16/22 /content/yolov5/Texture-defect-detection-2/test/images/c1r3e1n31_jpg.rf.a
image 17/22 /content/yolov5/Texture-defect-detection-2/test/images/c1r3e1n4_jpg.rf.92
image 18/22 /content/yolov5/Texture-defect-detection-2/test/images/c1r3e1n7_jpg.rf.41
image 19/22 /content/yolov5/Texture-defect-detection-2/test/images/c2r2e2n24_jpg.rf.1
image 20/22 /content/yolov5/Texture-defect-detection-2/test/images/c2r2e2n38_jpg.rf.2
image 21/22 /content/yolov5/Texture-defect-detection-2/test/images/c2r2e2n41_jpg.rf.2
image 22/22 /content/yolov5/Texture-defect-detection-2/test/images/c2r2e4n21_jpg.rf.e
Speed: 0.3ms pre-process, 9.3ms inference, 0.6ms NMS per image at shape (1, 3, 416, 4
Results saved to runs/detect/exp2
```

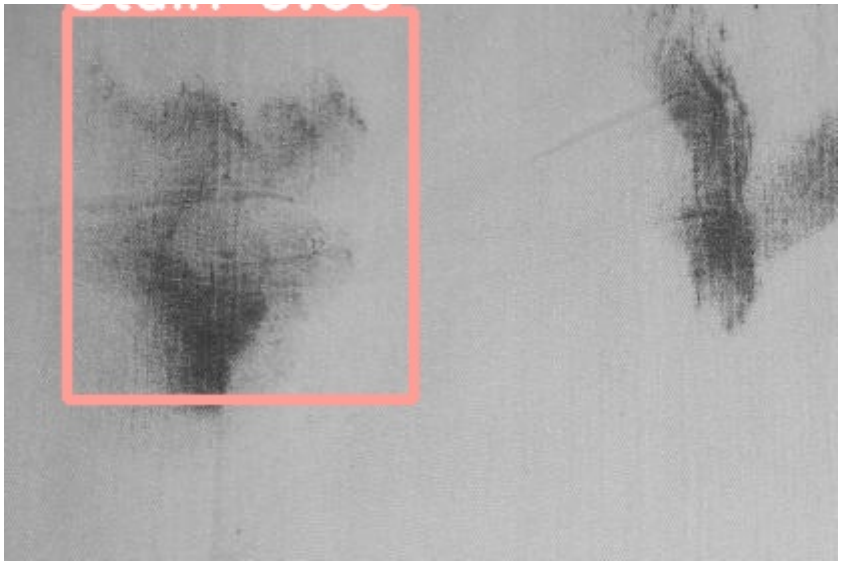
```
#display inference on ALL test images
#this looks much better with longer training above
```

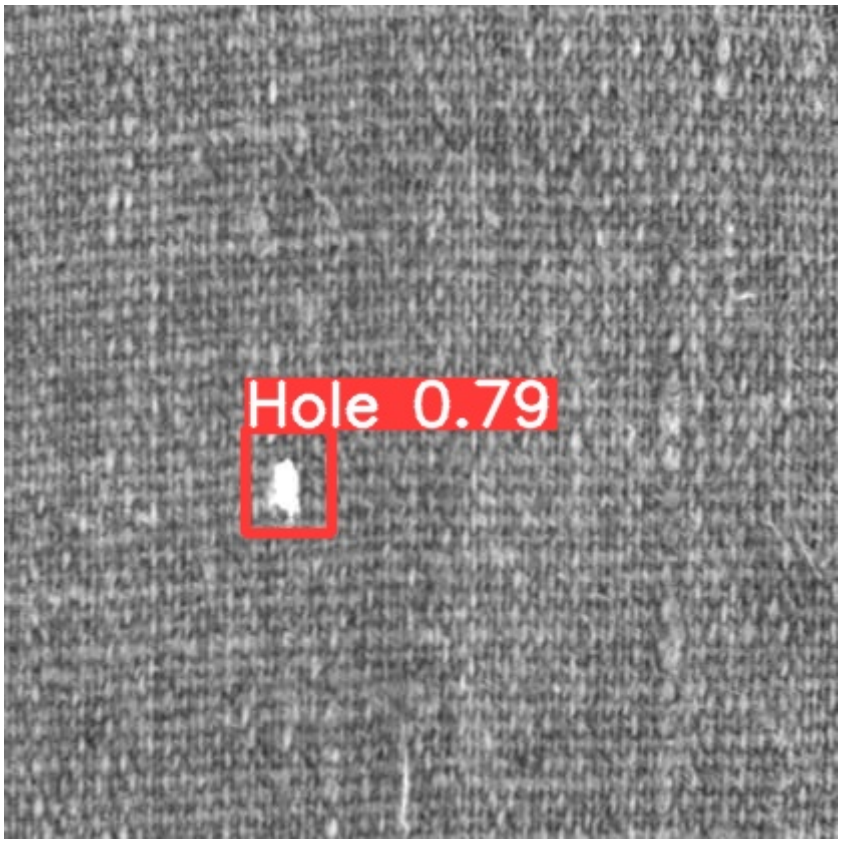
```
import glob
from IPython.display import Image, display

for imageName in glob.glob('/content/yolov5/runs/detect/exp2/*.jpg'): #assuming JPG
    display(Image(filename=imageName))
    print("\n")
```

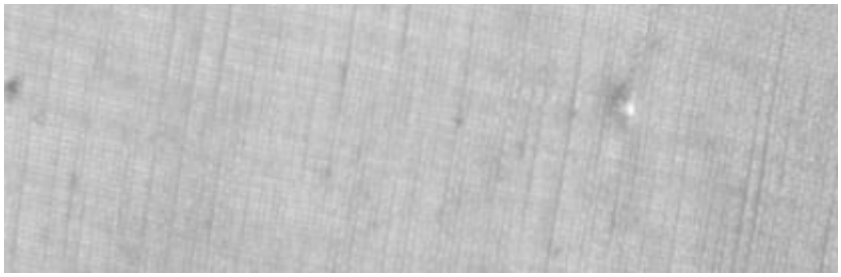

Stain 0.5
Stain 0.41

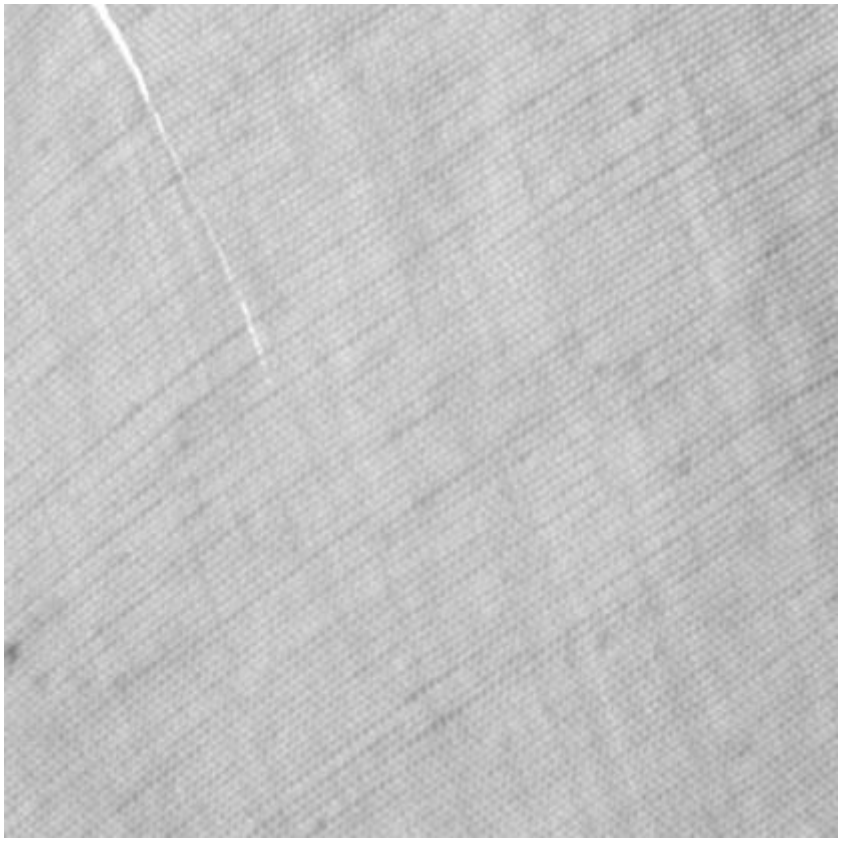

Hole 0

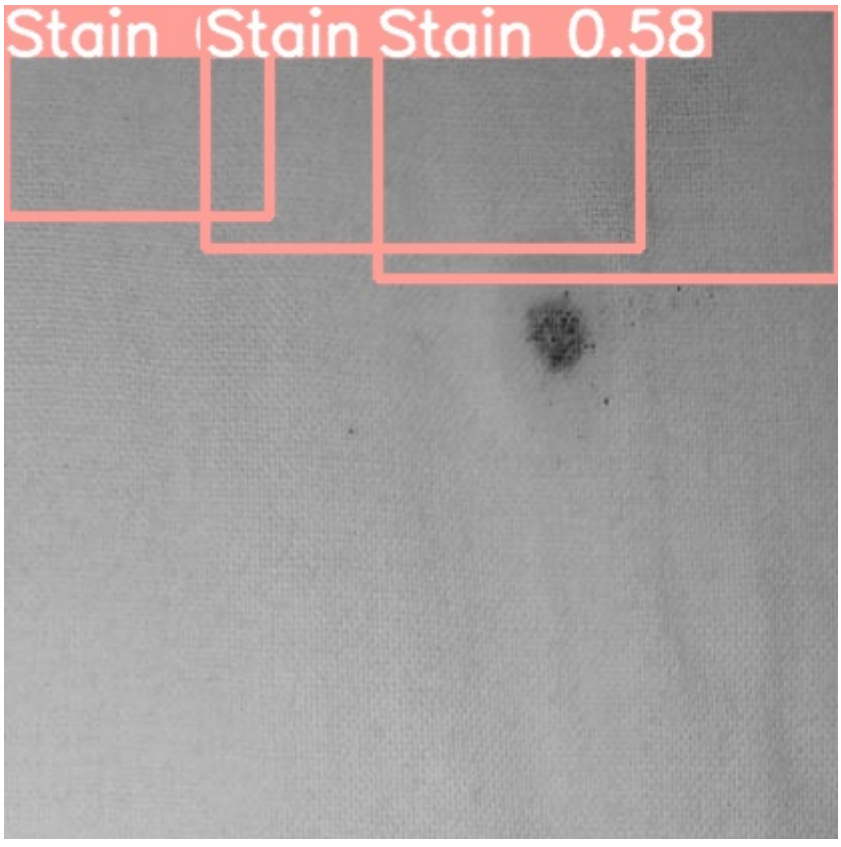

Stain 0.48
Stain 0.60

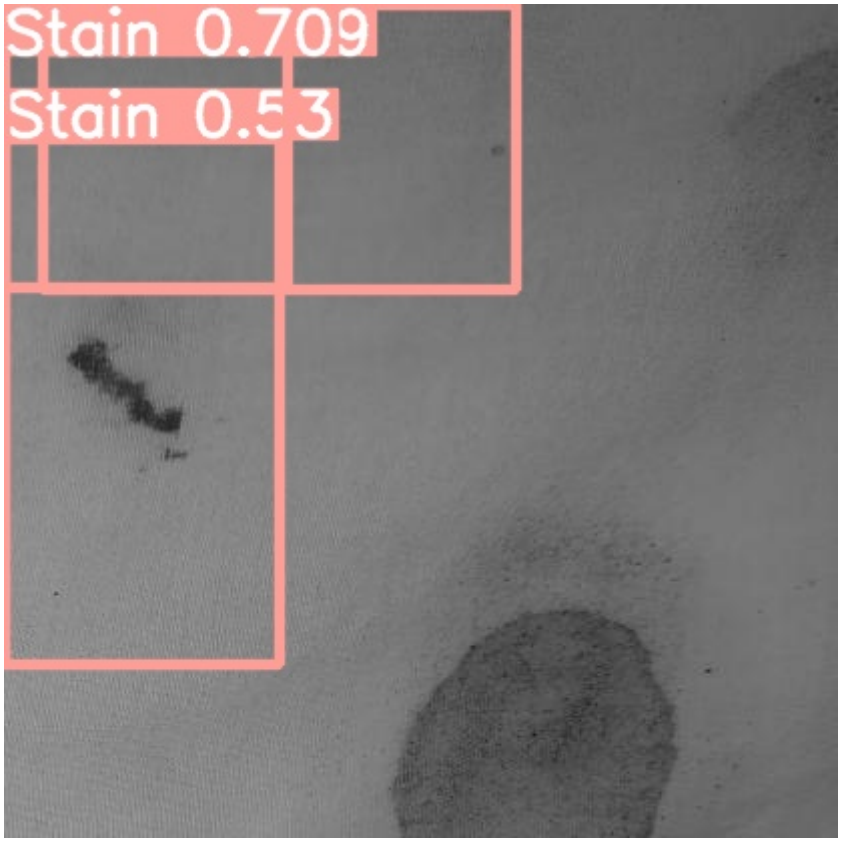


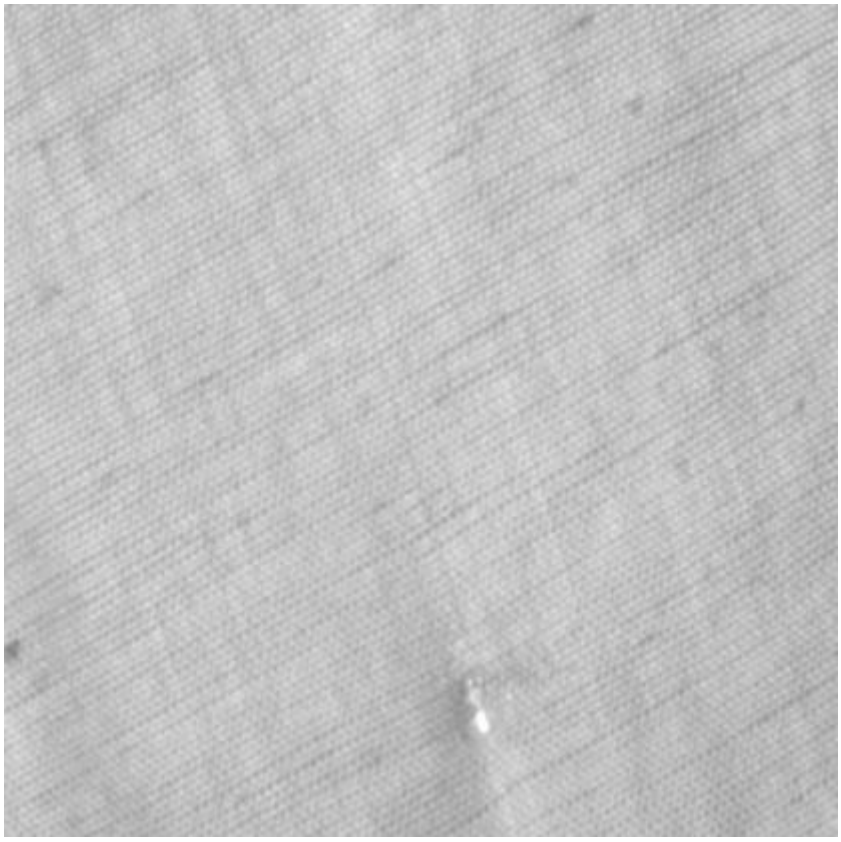

Hole 0.79

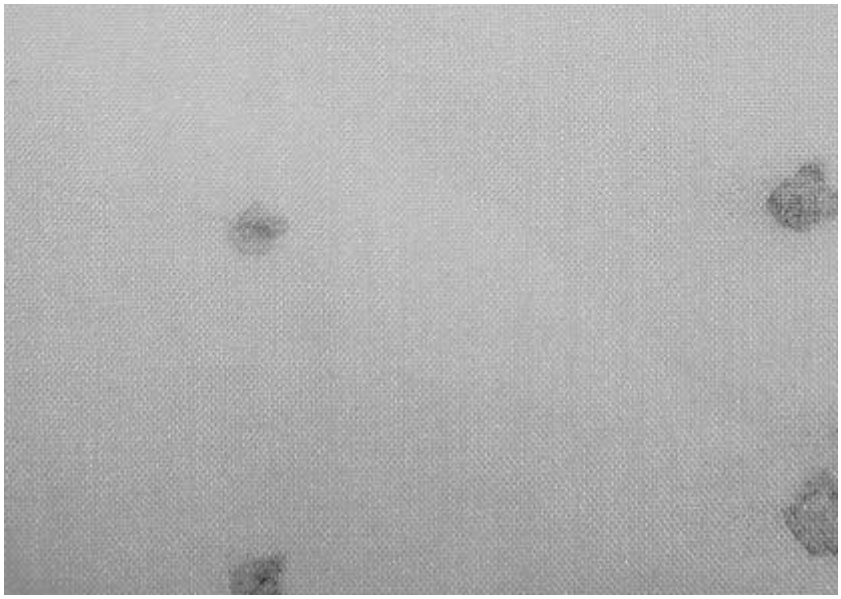


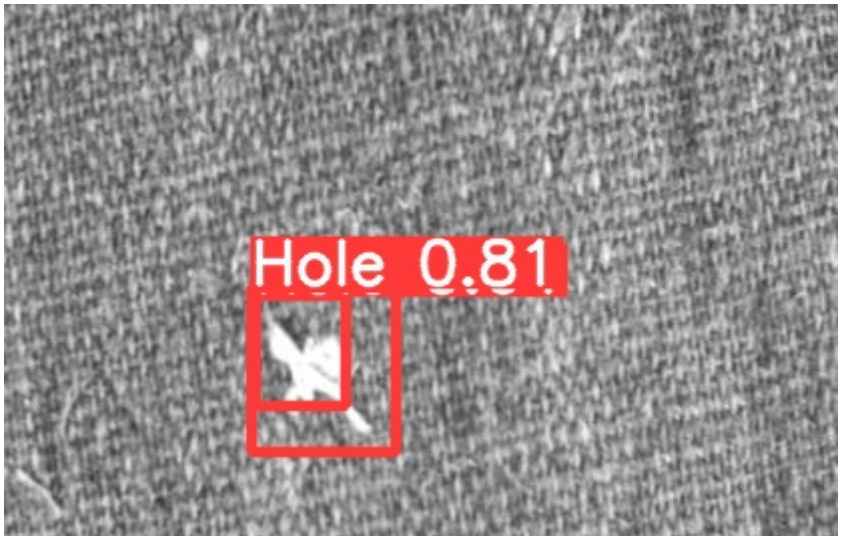


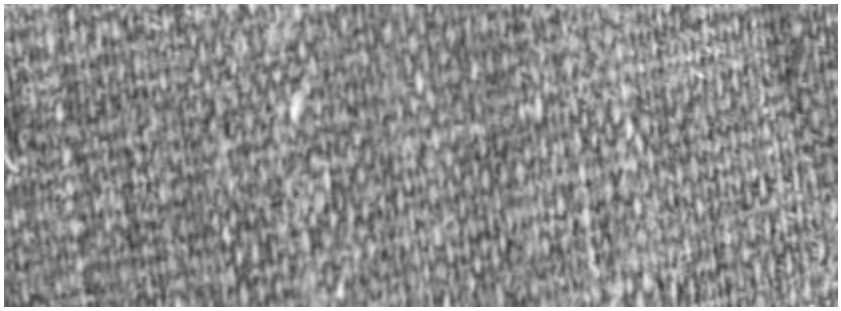

Stain
Stain Stain 0.58

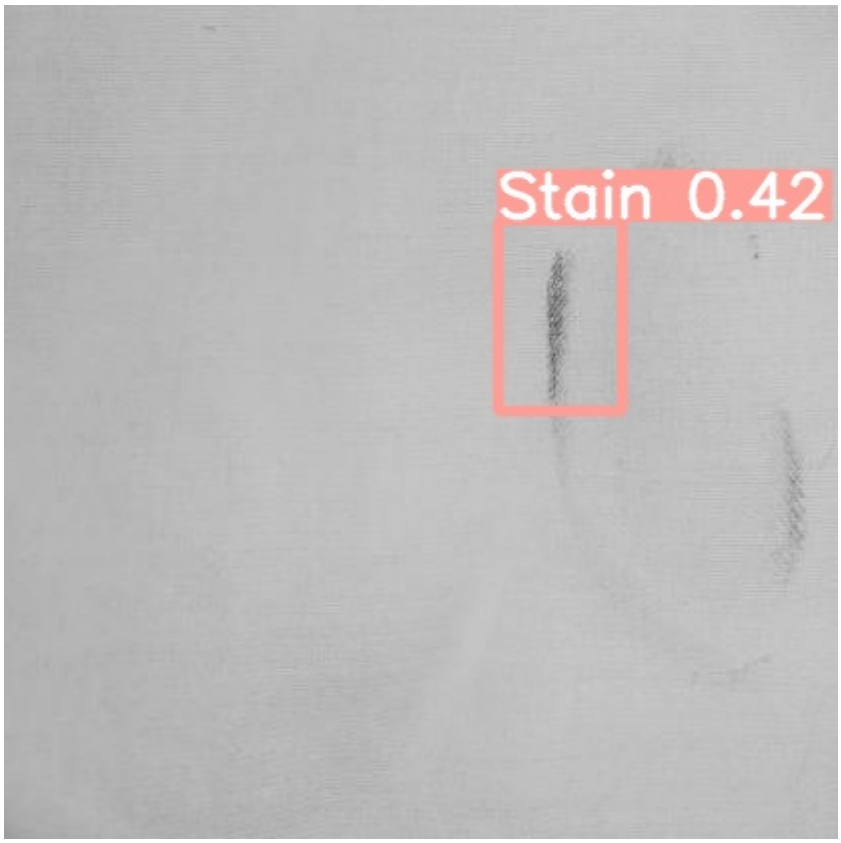

Stain 0.709
Stain 0.53

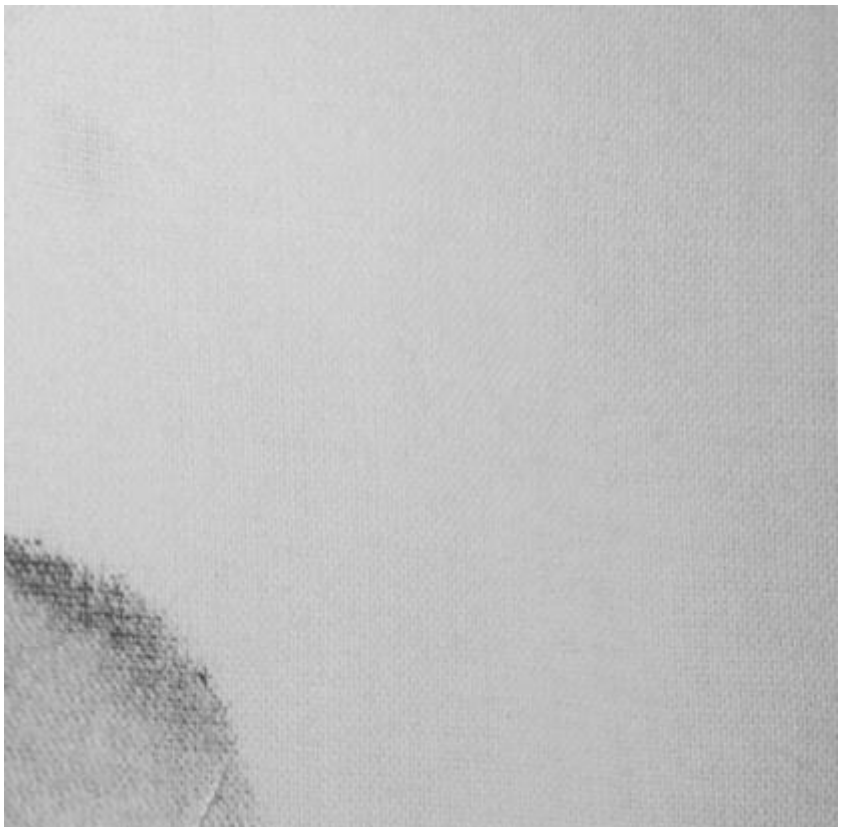


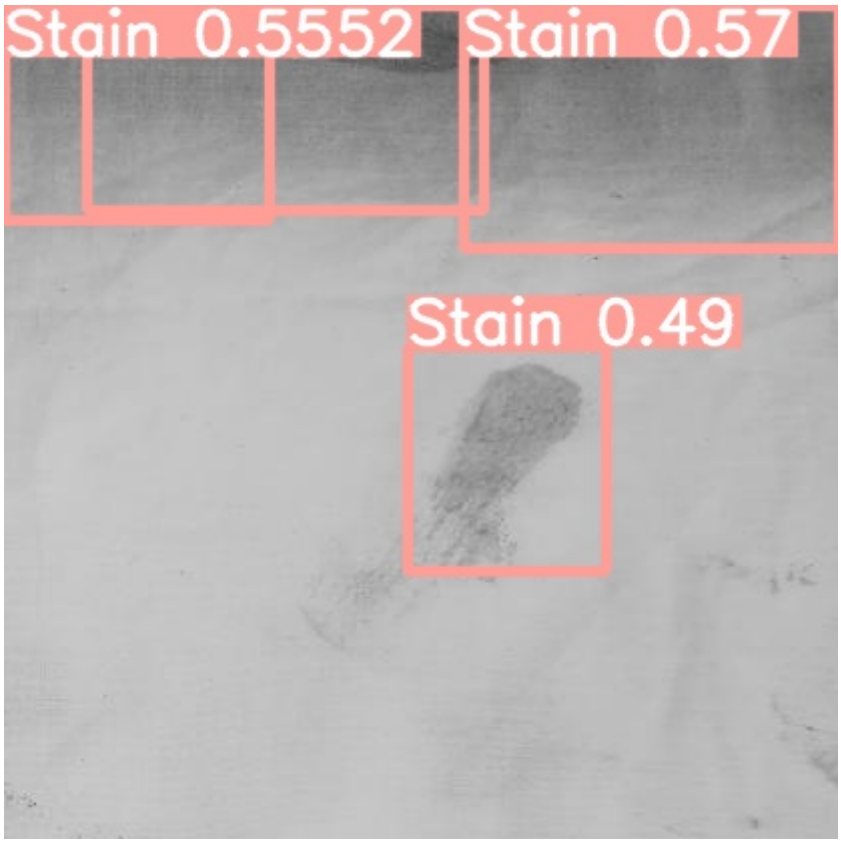


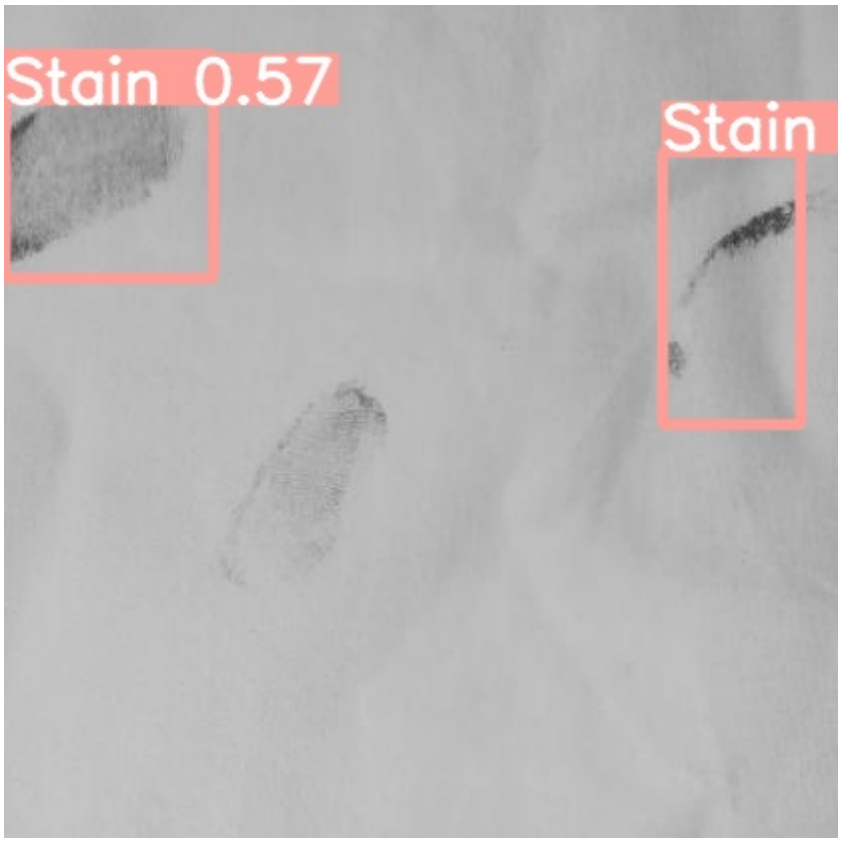

Hole 0.81

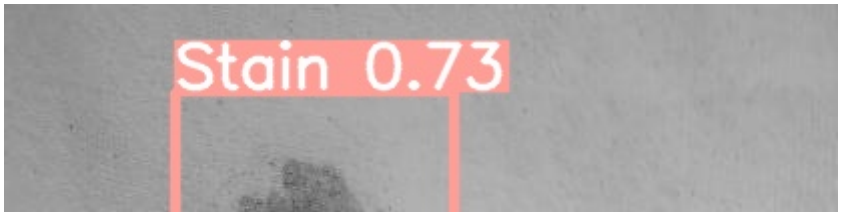


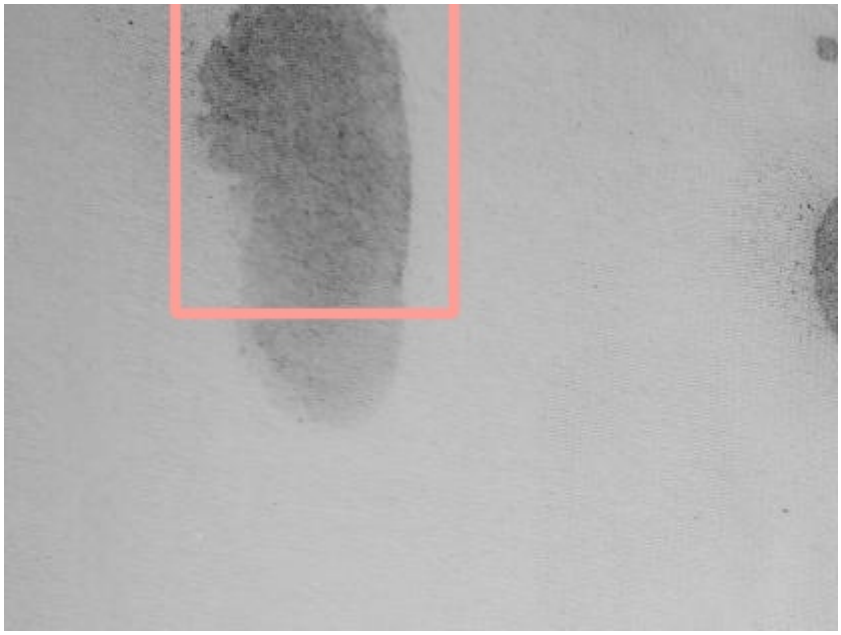

Stain 0.42

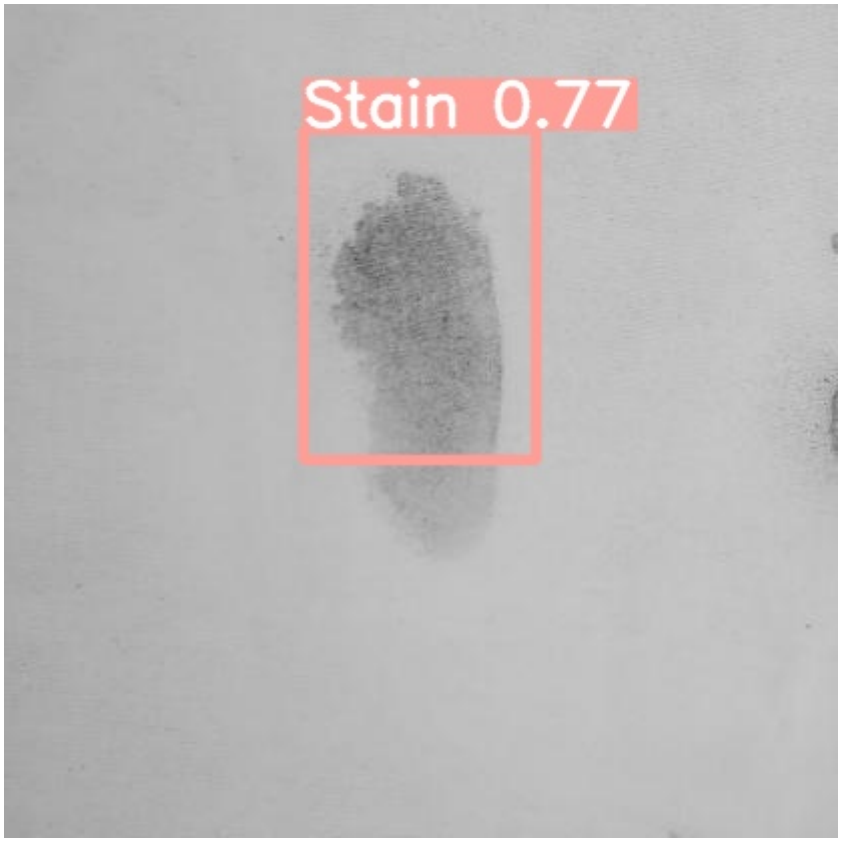


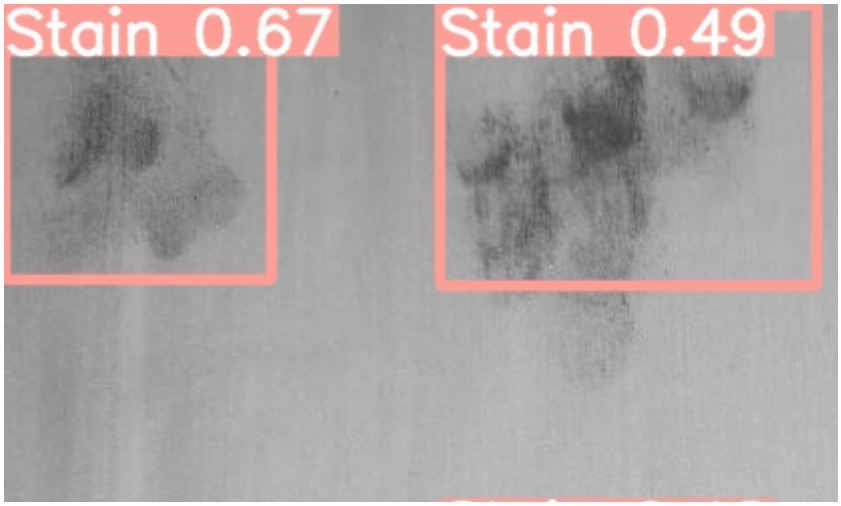

Stain 0.5552
Stain 0.57
Stain 0.49

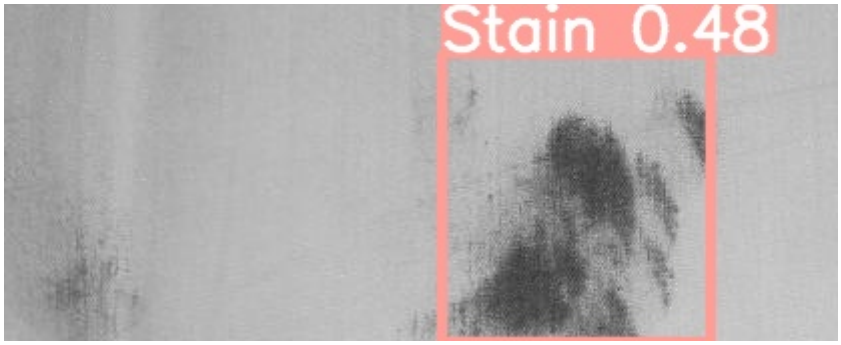

Stain 0.57
Stain

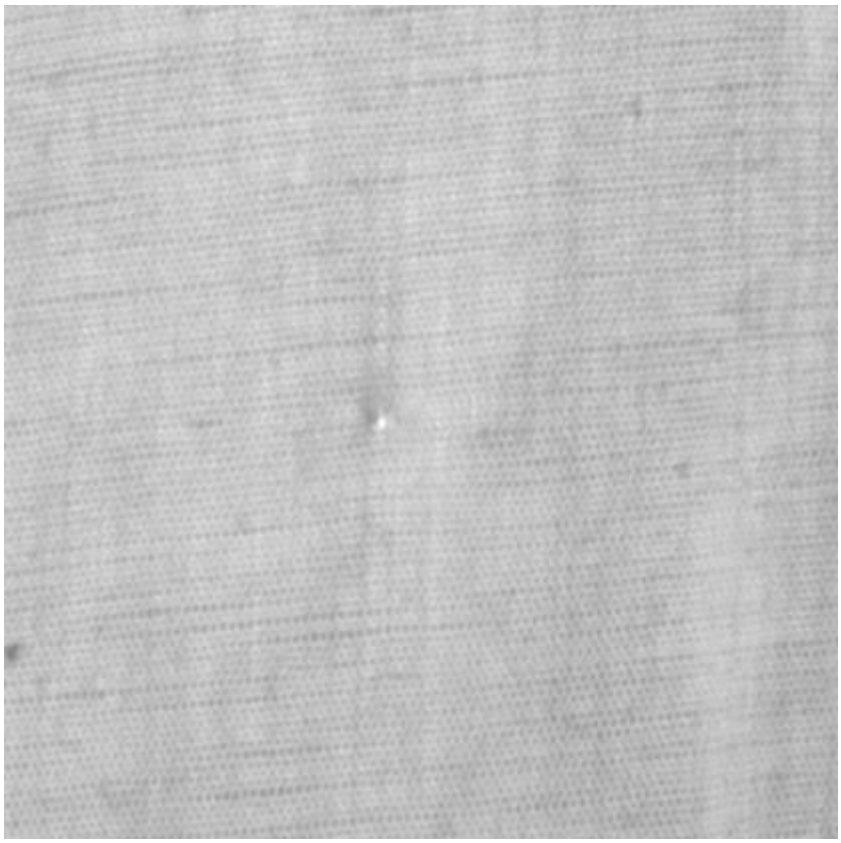

Stain 0.73




Stain 0.77


Stain 0.67
Stain 0.49





Completed!