Problem Statement

Data Description:
You will be predicting the location and type of defects found in steel sheets. Images are named with a unique ImageId. You must segment and classify the defects in the test set. Each image may have no defects, a defect of a single class, or defects of multiple classes. For each image you must segment defects of each class (ClassId = [1, 2, 3, 4]).

The segment for each defect class will be encoded into a single row, even if there are several non-contiguous defect locations on an image. Find more about the encoding standard in the Evaluation section.

Dataset link:

https://drive.google.com/file/d/1D-mPo5AOFrS3q38UuKdiBfJbPSh3UJJ3/view?usp=sharing

Files:

- train_images/ - folder of training images
- test_images/ - folder of test images (you are segmenting and classifying these images)
- train.csv - training annotations which provide segments for defects (ClassId = [1, 2, 3, 4])
- sample_submission.csv - a sample submission file in the correct format; note, each ImageId has 4 rows, one for each of the 4 defect classes

Evaluation:
Performance of a machine learning solution is evaluated on the mean Dice coefficient. The Dice coefficient can be used to compare the pixel-wise agreement between a predicted segmentation and its corresponding ground truth. The formula is given by:

{2*|X intersection Y|} / {|X| union |Y|}

where X is the predicted set of pixels and Y is the ground truth. The Dice coefficient is defined to be 1 when both X and Y are empty.

EncodedPixels:

Prediction uses run-length encoding on the pixel values. Instead of submitting an exhaustive list of indices for your segmentation, you will submit pairs of values that contain a start position and a run length. E.g. '1 3' implies starting at pixel 1 and running a total of 3 pixels (1,2,3).

The submission should be a space delimited list of pairs. For example, '1 3 10 5' implies pixels 1,2,3,10,11,12,13,14 are to be included in the mask. The metric checks that the pairs are sorted, positive, and the decoded pixel values are not duplicated. The pixels are numbered from top to bottom, then left to right: 1 is pixel (1,1), 2 is pixel (2,1), etc.

File Format for Output of Machine Learning method:

Initially the output should be stored in csv format, with a header and columns names: ImageId_ClassId, EncodedPixels. Each row in csv file represents a single predicted defect segmentation for the given ImageId, and predicted ClassId, and you should have the same number of rows as num_images * num_defect_classes. The segment for each defect class will be encoded into a single row, even if there are several non-contiguous defect locations on an image.

The GUI and its integration with the End-to-end Machine Learning solution:
Come Up with a pictorial representation of the GUI, with which we will integrate the inference module of the proposed end-to-end machine learning solution and deploy the inference solution in the form of a standalone application, locally hosted solution or web hosted solution.

Discussions:

Dividing the problem into 4 parts:

1. Binary Classification (defect or no defect)(depends on the question, if there are any non defective images included)
2. Segmentation Model (type of defect, location of defect)=>generate masks for the image, involves conversion of RLE (Run Length Encoder) to masks (also called pixel-level classification. In other words, it involves partitioning images (or video frames) into multiple segments or objects.)
3. Convert masks to encoded pixels
4. Make a gui to display the results

About the data set:

1. Each example in train data has image id, class id and encoded pixels
2. Each image has dimensions 1600*256 pixels

Binary Classification Models

1. Sequential VGG model using SGD optimizer and categorical_crossentropy loss-Failed due to exploding gradients, led to 'nan' values for loss
2. Sequential VGG model using Adams optimizer and reducing the layers in the model categorical_crossentropy loss-baseline, working model

Segmentation Models

1. Loss: Dice Loss-Used when data is imbalanced, Crossentropy loss, Combo Loss, Dice+BCE(BCEDice), Lovasz, Weighted dice+BCE, focal loss, focal+BCE loss,Jaccard, BCEDice
2. Metrics: Dice-coef to compare 2 images
3. Options for Data generator:
   X=image, y=4 masks

15-01-2022

1. https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/114254

    - Doubts- Pseudo labels(to be used when segmentation and classification are making same decisions, may not be useful for our model)

2. https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/114297

    - 4-class classification->making mask only for the predicted class->rle
        - Problem: there is small number of images with defect.
        - Solution 1: train segmentation model only on images with defect.
        - It may work only if your classifier is so awesome it can predict everything.
        - Solution 2: train on all images with defect plus some empty images.
        - Which images to choose? The ones with high probability from classifier.
        - When high means >0.01
        - (train on mistakes of classifier)
    - Training data was split into 4 for the classes to handle imbalance
    - Metric-Not to use the default metric of taking ll masks together, flattening, and then calculating- it is inaccurate in the cases where there are no defects
    - Loss-lovasz_hinge and lovasz_softmax.(BCE-> choose best model according to metrics->lovasz_hinge)
    - Model - efficientnet, seg-b0 &b1 & b2(??)
3. https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/114410
    - Encoder - efficientnet-b5
    - Decoder - unet, nested unet
    - Trained on 5 models and averaged scores, (don't use rms to get a weighted average here)
    - Only segmentation models with increased pixel threshold for non-defective images
    - Doubts- apex mixed precision, pseudo labels, overlapping defect predictions in sigmoid activation

4. https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/104851
    - Make training, inference kernel
    - Doubts-What is random seed kfold, holdout
      What is normal, snapshot ensemble
5. https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/114215
    - Doubts-What is Noam scheduler (AdamW with the Noam scheduler),
      mixup and label smoothing,
      random sampler

6. https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/110188
    - Removing false positives is imp
    - Can be done with or without classification
7. https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/114716

- Training two-headed NN for segmentation and classification.
- Combine heads at inference time as with soft gating (mask.sigmoid() * classifier.sigmoid())
- Loss=Focal loss / BCE + Focal loss
- Training with grayscale instead of gray-RGB
- Doubts-FP16 with usage of Catalyst and Apex

8. https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/117208
    - To handle imbalance-lovasz hinge, pseudo labelling,
    - Image augmentation can be used to deal with data imbalance
9. https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/114225
    - 5 class classification->4 class seg
    - BCE loss
    - Training resnet, efficientnet works
10. https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/114309
    - mean ensemble of the multilabel segmentation nets(??)
    - Useful:
        - [Keras] https://github.com/qubvel/segmentation_models
        - [Keras] https://github.com/qubvel/efficientnet
        - [Keras] https://github.com/qubvel/tta_wrapper
        - [Everywhere] https://github.com/albu/albumentations
    - Used 4 binary segmentation models
    - Save weights as check points and then try different combinations

11. https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/114465
    - 4 separate binary segmentation
    - No prediction for class2
    - Trained on entire image, so no overfitting, no tuning required
    - Doubts-What is grid search for augmentation
    - 0.75 * BCE + 0.25 * LovaszHinge (lovasz was better than dice)
12. https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/114269
    Ensemble of these algorithms :
    - unet_resnet34,
    - unet_se_resnext50_32x4d,
    - unet_mobilenet2,
    - fpnb0,
    - fpnb1,
    - fpnb2,
    - fpn_se_resnext50_32x4d,
    - fpn_resnet34_v2
    - (heng's resnet34 classifier)

13. https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/114332
    - Ensemble of many models,classification
    - Adam, steplr scheduler

- base: 1e-3 -> 1e-4
- linear: 1e-4 -> 1e-5
- Segmentation
- (??)encoder, decoder->
- Adam, steplr scheduler
- encoder: 1e-4 -> 5e-5 -> 1e-5
- decoder: 1e-4 -> 5e-5 -> 1e-5

14. https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/114367
    Doubt-Data augmentation-albumentation, balanced sampling(written that it is done for non-defective images)
15. https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/114217

****

Training data for segmentation model:

1. X=image, y=4 masks ,1 per class
2. New csv file is made for the updated dataframe, called all_encoded_pixels-train.csv

****

Mahek's Approach

1. https://github.com/qubvel/segmentation_models/blob/master/examples/multiclass%20segmentation%20(camvid).ipynb
   A 2 class classification followed by 4 class segmentation similar to the above link

2. A 2 class classification followed by 4 class segmentation using smp, pytorch similar to module 9 on this list
3. 4 separate binary segmentation models, similar to 11
4. Encoder-Decoder architecture, like 3

22-01-22
Analysis of 9

1. Configuration file to store all the variables in a systematic manner & parsing arguments
    - Load it using string.**format()**
    - Parsing arguments using argparse library-> The program defines what arguments it requires, and **argparse** will figure out how to parse those out of sys.argv. The argparse module also automatically generates help and usage messages and issues errors when users give the program invalid arguments.

2. Split_folds- data preprocessing, cleaning
    - Loading config:Load the configuration files with a default configuration, using **easydict** for this, which allows the user to access dictionary values as attributes. The config file read here is base_config.yml , merging this and the default config(passed as dictionary, converted to attributes) is what is returned by load_config
    - Make the working directory(specified in base_config), here kaggle/working
    - Data splitting:used **stratifiedKfold** from sklearn, which distributes all classes defects equally across the splits made by it(here 5) to handle data imbalance
    - Using pandas df.**pivot()** to organize the table by columns->classId i.e. creating 4 new columns per class, each column has value encoded pixels, blank if encoded pixels are absent
    - Add fold(the ones we generated by stratifiedkfold) column using imageid as the common column to merge dfs
      Saving this df as a csv file for later
3. Transforms available
    - **Ablumentatations(module) https://arxiv.org/abs/1809.06839**
        - HorizontalFlip
        - VerticalFlip
        - RandomCrop
        - Resize
        - Cutout
        - Normalize
        - Compose
        - GaussNoise
        - IAAAdditiveGaussianNoise
        - RandomContrast
        - RandomGamma
        - RandomBrightness
        - OneOf

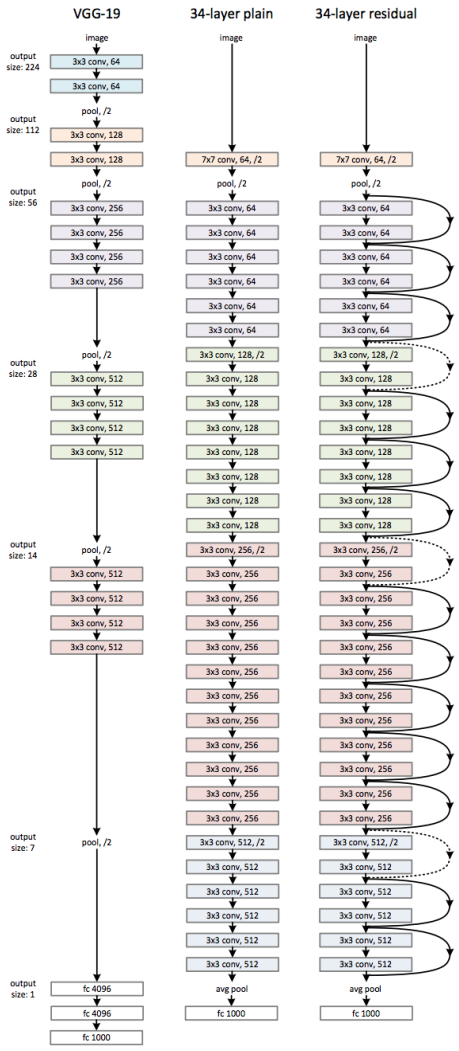
- List is created and whichever transformations are to be used according to the config file are appended
- Composed list is returned

4. Datasets
   Make_loader returns a Dataloader, decodes images using jpeg4py.JPEG(path).decode() and apply transformations from transforms
   X is images and y is the labels/classId
5. Training
   - Making a new directory for the model, saving its configuration file(obtained from models' config merged with default config) in this directory
   - In train file the transforms are stored in a dictionary against the keys 'train' and 'valid' respectively fortran and test
   - Making data loaders for train and validation sets
   - **About torchvision models**
     The models subpackage contains definitions of models for addressing different tasks, including: image classification, pixelwise semantic segmentation, object detection, instance segmentation, person keypoint detection and video classification.
     Resnet:
     https://www.mygreatlearning.com/blog/resnet/#sh1

     https://arxiv.org/abs/1512.03385
     - Residual Networks, or ResNets, learn residual functions with reference to the layer inputs, instead of learning unreferenced functions. Instead of hoping each few stacked layers directly fit a desired underlying mapping, residual nets let these layers fit a residual mapping. They stack residual blocks on top of each other to form a network: e.g. a ResNet-50 has fifty layers using these blocks.
     - It has been found that there is a maximum threshold for depth with the traditional Convolutional neural network model. Issue ->Vanishing gradients problem
     - Residual blocks have a skip connection,
       1. Without them
          H(x)=f(wx + b)
          or H(x)=f(x)
       2. With the introduction of skip connection, the output is changed to
          H(x)=f(x)+x
       3. The projection method is used to match the dimension which is done by adding 1×1 convolutional layers to input. In such a case, the output is:
          H(x)=f(x)+w1.x
       4. The skip connections in ResNet solve the problem of vanishing gradient in deep neural networks by allowing this alternate shortcut path for the gradient to flow through

5. It also allows the model to learn the identity functions which ensures that the higher layer will perform at least as good as the lower layer, and not worse (when f(x)=0, H is identity)

- Architecture of vgg vs plain vs resnet
  https://arxiv.org/pdf/1409.1556.pdf



- Using torchvision models are imported, a parameter is passed to give a boolean option to choose if model should be pre-trained
  Last linear layer is added so that the output layer shape is as desired.

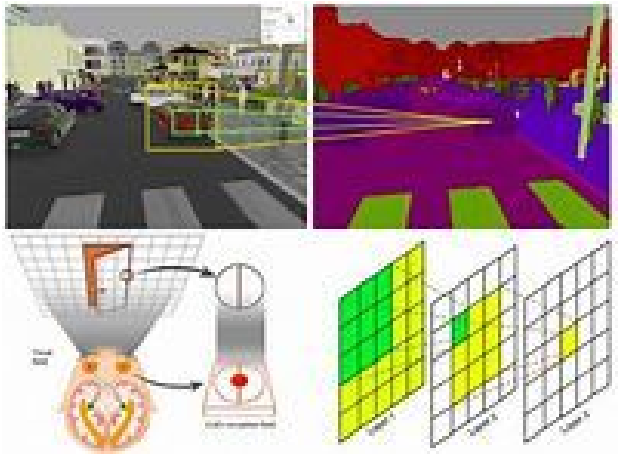
- Optimizers :
  - RAdam

- Losses
    - Binary cross entropy=>
        1. Binary cross entropy compares each of the predicted probabilities to actual class output which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close or far from the actual value.
        2. It is the negative average of the log of corrected predicted probabilities.
- schedulers
- Callbacks
    - Multi class accuracy => total correct outputs/total inputs
    - F1 score=>Harmonic mean of precision and recall (where (precision= fraction of true positive examples among the examples that the model classified as positive. In other words, the number of true positives divided by the number of false positives plus true positives.); (recall=Recall, also known as sensitivity, is the fraction of examples classified as positive, among the total number of positive examples. In other words, the number of true positives divided by the number of true positives plus false negatives.))
- fp16

25-01-22

Doubts of previous part(answers)

1. Data representation:
    - data is stored and then processed and later transmitted through graphical representation like in bar graph, line graph, frequency table etc.
2. Receptive field
    - Receptive Field (RF) is defined as the size of the region in the input that produces the feature.
    - A measure of association of an output feature (of any layer) to the input region (patch)



3. Stride
    - It allows performing several operations with an image in a very fast manner (in constant time) by simple modification of image metadata
    - 8 bits are used to store the color value of a single channel=> the total bit depth(amount of bits per pixel) of one pixel is 24
    - Easier to process 32 and 16-bit chunks of data on typical 32 and 64-bit processors.=> 8 extra bits are stored per pixel, this is pixel padding
    - total number of bytes occupied by a pixel in memory is called pixel stride
    - (Similarly line padding and line stride also exist)

4. Relu vs sigmoid
    - ReLU avoids vanishing gradients problem
    - Relu has a much lower run time. (max (0,a) runs much faster than any sigmoid function (logistic function for example = 1/ (1+e^ (-a)) which uses an exponent)
    - Sigmoid activation is an ‘S-Shaped’ curve that maps the input values in the range of 0 and 1.
    - The value of the sigmoid function asymptotically approaches 0 and 1. This is in line with the way probabilities work.

5. Response normalization(local response normalization=lrn)
    - a normalization layer that implements the idea of lateral inhibition. Lateral inhibition(neurobiology) refers to the phenomenon of an excited neuron inhibiting its neighbors: this leads to a peak in the form of a local maximum, creating contrast in that area and increasing sensory perception.
    - In practice, we can either normalize within the same channel or normalize across channels when we apply LRN to convolutional neural networks.

6. ILSVRC challenge(ImageNet Large Scale Visual Recognition Challenge)
    - evaluates algorithms for object detection and image classification at large scale

7. Kernel-size-stride-padding-maxpooling layer and fully connected layer relation

Segmentation

1. Load configuration file
2. Make working directory
3. Transforms used: from ablumentations
    a. Train-
        i. HorizontalFlip
        ii. VerticalFlip
        iii. Cutout:(the process of darkening square regions of an image)
            1. num_holes: 10
            2. hole_size: 25
        iv. RandomCropScale
        v.
    b. Test-
        i. HorizontalFlip
        ii. VerticalFlip
        iii. Cutout:
            1. num_holes: 10
            2. hole_size: log (to the base 10) 256
4. Datasets
   Make_loader returns a Dataloader, decodes images using jpeg4py.JPEG(path).decode() and apply transformations from transforms
   X is images and y is the labels/classId

5. Model
    a. arch: 'Unet'
        i. relies on the strong use of data augmentation to use the available annotated samples more efficiently. Such a network can be trained end-to-end from very few images and outperforms the prior best method (a sliding-window convolutional network)
    b. encoder: 'resnet18'
    c. Pretrained on : 'imagenet'

6. Loss: BCEDice
    a. Bce+dice loss
    b. BCEWithLogitsLoss from pytorch=This loss combines a Sigmoid layer and the BCELoss in one single class. This version is more numerically stable than using a plain Sigmoid followed by a BCELoss as, by combining the operations into one layer, we take advantage of the log-sum-exp trick for numerical stability.
    c. $\ell(x,y)=L=\{l_1,\dots,l_N\}^\top, l_n=-w_n[y_n\cdot\log\sigma(x_n)+(1-y_n)\cdot\log(1-\sigma(x_n))],$

7. Optimizer: RAdam-Rectified Adam, or RAdam, is a variant of the Adam stochastic optimizer that introduces a term to rectify the variance of the adaptive learning rate. It seeks to tackle the bad convergence problem suffered by Adam(Adam is shown not being able to converge to the optimal solution in certain cases.)
8. Scheduler
9. mixup,cutmix
10. Callbacks
    a. DiceCallback,
    b. IouCallback,
    c. CheckpointCallback,
    d. MixupCallback