**DOCUMENTATION**

*FAULT DETECTION SYSTEM DEVELOPMENT*

**Augmentation Pipeline**

dataset\_augment =

augmenters.Sequential([

augmenters.HorizontalFlip(p=0.25),

augmenters.VerticalFlip(p=0.25),

augmenters.SomeOf(5, [

augmenters.blur.GaussianBlur(sigma=(0, 2), seed=augment\_seed),

augmenters.blur.MedianBlur(k=(1, 3), seed=augment\_seed),

augmenters.size.Crop(percent=(0, 0.10), seed=augment\_seed),

augmenters.geometric.Affine(rotate=(-45, 45), scale=(0.9, 1.1), translate\_percent=(-0.05, 0.05), seed=augment\_seed, cval=100, mode="symmetric"),

augmenters.geometric.Rot90(k=(1, 3), seed=augment\_seed),

augmenters.arithmetic.Dropout(p=(0, 0.075), seed=augment\_seed),

augmenters.arithmetic.SaltAndPepper(p=(0, 0.075), seed=augment\_seed),

augmenters.color.MultiplyBrightness(mul=(0.5, 1.5)),

augmenters.color.MultiplySaturation(mul=(0, 5), seed=augment\_seed),

augmenters.iaa\_convolutional.Sharpen(alpha=(0.75, 1), lightness=(0.75, 1.25), seed=augment\_seed),

augmenters.iaa\_convolutional.Emboss(alpha=(0.75, 1), strength=(0.75, 1.25), seed=augment\_seed),

augmenters.contrast.CLAHE(seed=augment\_seed),

augmenters.contrast.GammaContrast(gamma=(0.2, 5), seed=augment\_seed),

])

])

* The above code shows the augmentation pipeline used; it is developed using the *imgaug* python package.
* Out of all the images in the dataset, 25 % of them will be horizontally flipped and 25 % of them will be vertically flipped.
* Out of all the subsequent augments shown, anywhere between 0 up until a maximum of 5 augments will be applied; the number of augments to be applied is chosen at random.
* Setting a seed for each augment and a global imgaug seed will allow for completely reproducible behaviour.

**General Notes**

* In order to save space and time, no images are written to a storage device. All the images used are only stored in the RAM, first as numpy arrays, which are then converted to torch tensors to be used with the pytorch models.
* Since images are stored in RAM, care should be taken to not exceed the storage capacity. Since almost all the models discussed here use (uint8) 3 channel images of size 224x224, the size of each image in RAM is now 3x224x224 = 0.14355 MB

**1. VGGesque Classifiers**

* Model used a very simple VGGesque Architecture.
* Tested on very few and large number of filters per layer.
* Tested on both grayscale and color images of varying sizes : 48x48, 96x96, 144x144 and 192x192
* Model Information can be found [here](https://colab.research.google.com/drive/1dr2AJQR1hL6Sr7mwOLL_zPtf0rjAhNlb?authuser=1#scrollTo=qs7yKqEbrXUD&line=4&uniqifier=1).
* Assumed that the end user is able to provide both, a faulty sample and one that is not.

Notes:

* If few filters were used, model only worked if presented with objects of the same class, irrespective if they were faulty or not.
* If large number of filters were used, models’ realtime performance improved as it was also able to reject other unrelated objects, but the system was not sensitive enough to classify extremely minute faults or extremely similar but distinct objects.

Conclusions:

* This idea was dropped since training a classifier that could at the very least reliably distinguish between distinct objects would require a lot of data and take a lot of time to train. Also, upon further testing performed at a later time, it was also found out that even if provided with the data, the system CANNOT detect minute faults within the object.

Conv2d (3x3)

BatchNorm

ReLU

Maxpool (2x2)

x4

Linear Classifier

Flatten

**x 4**

**2. Usage of pretrained models to extract features and use that data in an ANN**

* In the previous case, the models used generated the feature vectors from scratch.
* This idea made used of the plethora of pretrained computer vision models available to extract highly robust features and it was these features that were used in the classification task.
* Feature Extractor Models were constructed by taking the pretrained model architecture and slicing off the final classification layer such that the output of the network after being passed an image is now a vector. The vector length was chosen to be kept at 2048.
* On testing the various models, it was found that the VGG16 with Batch Normalization and ResNet-50 Architectures provided the best results. (All analysis was performed on these two architectures only)
* Before passing the feature vectors through an ANN, they are normalized to values between 0 and 1.
* Model was tested on 2 scenarios:

1. User can provide positive and negative samples (More Ideal)

2. User can provide only a positive sample (More Info can be found [here](https://colab.research.google.com/drive/1dr2AJQR1hL6Sr7mwOLL_zPtf0rjAhNlb?authuser=1#scrollTo=6pkSpotb0XQH&line=12&uniqifier=1))

* Model Information can be found [here](https://colab.research.google.com/drive/1dr2AJQR1hL6Sr7mwOLL_zPtf0rjAhNlb?authuser=1#scrollTo=gp5S-J7at5I0&line=22&uniqifier=1).

Notes:

* Feature Vectors of the various objects were compared. Almost all of them were very similar to each other. (According to the cosine similarity metric)
* Feature Vectors of an object and the same object that contains a minute fault are nearly identical, both, by correlation and cosine similarity standards.
* When trained on only the positive samples, the system almost always failed, either by not being able to identify a fault in the current object or classifying an object as not faulty even though it is a completely different object. An explanation for this behaviour is that the feature vectors of the two images (frames) are much too similar to be able to reliably differentiate.

Conclusions:

* The idea was dropped because even after a lot of tests (under the assumption that Scenario 2 is possible) and preprocessing, the system could not classify minute faults on the same object. In rare cases, it would also fail when presented with another object with an extremely similar feature vector

ANN Classifier Training

1. Capture Image

2. Synthetically create dataset via Augmentation

Feature Extractor

Normalization

Feature Vector Dataset

Fault Detector

**3. Extract RoI from the image and extract features from the RoI, then use data with an ANN**

* Uses an object detector to extract the ROI from the frame.
* Similar to the previous case, features are extracted from the ROI Image using a pretrained model.
* Data is then passed into an ANN which is trained to try and classify between faulty or not.
* Assumed that the end user is able to provide both, a faulty sample and one that is not.

Notes:

* The previous 2 models considered the background of the image/frame in their analysis. Using a detector to only extract the ROI eliminated the background from influencing the decision of the network.
* The bounding box coordinates do not give the bounding box of minimum area. So if the object was not placed at an integer multiple of 90 degrees, the background would again creep into influencing the decision of the network.
* The system was not able to reliably differentiate between faulty and not faulty samples of the same object, unless the faults were obvious.

Conclusions:

* This idea was dropped because

1. Too computationally intensive

2. Object Detectors do not give the minimum area bounding box, so using it did not provide any benefits as it highly restricted how the objects can be placed.

3. Unreliable

Feature Extractor

1. RoI Image

2. Synthetically create dataset via Augmentation

Feature Vector Dataset

ROI Extractor

Capture Image

Normalization

ANN Classifier Training

Fault Detector

**4. Two Stages (Train ANN, then train Feature Extractor + ANN)**

* Feature Extractor Models were constructed by taking the pretrained model architecture and slicing off the final classification layer such that the output of the network after being passed an image is now a vector. The vector length was chosen to be kept at 2048.
* Before passing the feature vectors through an ANN, they are normalized to values between 0 and 1.
* Train the ANN
* Now finetune the entire network by training the pretrained feature extractor and the ANN trained in the previous step; this step is highly computationally intensive.
* Model was tested on 2 scenarios:

1. User can provide positive and negative samples (More Ideal)

2. User can provide only a positive sample (More Info can be found [here](https://colab.research.google.com/drive/1dr2AJQR1hL6Sr7mwOLL_zPtf0rjAhNlb?authuser=1#scrollTo=6pkSpotb0XQH&line=12&uniqifier=1))

Notes:

* When trained on only the positive samples, the system almost always failed, either by not being able to identify a fault in the current object or classifying an object as not faulty even though it is a completely different object. An explanation for this behaviour is that the feature vectors of the two images (frames) are much too similar to be able to reliably differentiate.

Conclusions:

* This idea was dropped because

1. Too computationally intensive

2. Finetuned Feature Vectors cannot detect minute faults

3. Unreliable

**STAGE I (Classifier Training)**

Capture Image and synthetically create dataset via Augmentation

Feature Extractor

Feature Vector Dataset

Normalization

ANN Classifier Training

**STAGE II (Feature Extractor Finetuning + Classifier Training)**

ANN Classifier Training

Feature Vector Dataset

Feature Extractor Finetuning

Image Dataset Creation

Fault Detector

**5. Siamese Networks**

* Feature Extractor Models were constructed by taking the pretrained model architecture and slicing off the final classification layer such that the output of the network after being passed an image is now a vector. The vector length was chosen to be kept at 2048.
* Before passing the feature vectors through an ANN, they are normalized to values between 0 and 1.
* Instead of using a simple ANN, as Siamese ANN is used instead. The architecture during the training and validation phases are shown below:

**Training Phase**

Positive Sample

Input Layer (2048)

Embedding (2048)

Output Layer (1) [Similarity Score]

Negative Sample

Input Layer (2048)

Embedding (2048)

**Validation Phase**

Sample

Input Layer (2048)

Embedding (2048)

Output Layer (1) [Similarity Score]

* All Models in this [repository](https://github.com/pchandrasekaran1595/Fault-Detection) use a Siamese Network Architecture
* The core concept behind the usage of this architecture is to be able to separate apart extremely similar vectors. This is achieved by learning new embeddings in accordance with binary labels assigned based on the class of the image.
* This implementation requires 2 images during the training phase; an anchor image and an image from the dataset (can be either a positive or negative image). Each image in the positive directory is considered to be an anchor image. For each of the anchor images, the system generates *num\_samples* (anchor, positive) and *num\_samples* (anchor, negative) image pairs for a total of *2\*num\_samples* image pairs per anchor. Hence, the total number of image pairs generated would now be, *(num\_of\_anchor\_images \* (2 \* num\_samples))*.
* Model Information can be found [here](https://colab.research.google.com/drive/1dr2AJQR1hL6Sr7mwOLL_zPtf0rjAhNlb?authuser=1#scrollTo=juenSN8kA4wt&line=8&uniqifier=1)
* Model was tested on 2 scenarios:

1. User can provide positive and negative samples (More Ideal)

2. User can provide only a positive sample (More Info can be found [here](https://colab.research.google.com/drive/1dr2AJQR1hL6Sr7mwOLL_zPtf0rjAhNlb?authuser=1#scrollTo=6pkSpotb0XQH&line=12&uniqifier=1))

Notes:

* When tested with a model trained on only positive samples (negative samples artificially created), the system would fail to identify defects in the object, because the negative sample data has been artificially created and it does not represent real world data. However, subsequent runs and additions of real world False Negatives and Positives highly improves the performance of the system
* When tested with a model trained on both positive and negative samples, system performance is very good with only few instances of misclassifications.
* Primary reason for misclassifications are due to the fact that the lightning employed in the current system is extremely sub-optimal; synthetic augmentation and real world augmentation results in the system looking at 2 completely different objects)

Conclusions:

* This is the network architecture chosen for the final model.
* Rigorous testing has proven that the system will be able to reliably classify components into faulty or not; provided the system has been trained on the data.

Future Recommendations:

* The final system allows for continuous learning, which at present is not automated and must be manually performed. A recommendation would be to retrain the model after every 5 images have been added to either the Positive or Negative Dataset.
* The system at present takes into consideration the background of the frame/image as well. Once the lighting issues have been fixed, the frame can be preprocessed using contour approximation to generate a mask from which only the object can be extracted. A similar concept can be used if depth map information is present.
* Future systems that could potentially use depth map information can also make use of this architecture to reliably differentiate between samples.

**6. Triplet Networks**

* Feature Extractor Models were constructed by taking the pretrained model architecture and slicing off the final classification layer such that the output of the network after being passed an image is now a vector. The vector length was chosen to be kept at 2048.
* Before passing the feature vectors through an ANN, they are normalized to values between 0 and 1.
* New embeddings are learnt in accordance to an anchor, positive and negative sample and these are the embeddings that are fed into the classifier whose output is the fault detection system.
* The embedding network is trained using TripletMarginLoss and the classifier is trained using BinaryCrossEntropy Loss.
* Model information can be found [here](https://colab.research.google.com/drive/1dr2AJQR1hL6Sr7mwOLL_zPtf0rjAhNlb?authuser=1#scrollTo=K-C5RCOlOS_D&line=41&uniqifier=1)

Notes:

* The only reason this idea was dropped was because this was not adequately tested to form a strong conclusion.
* Further testing and experimentation is possible to check and see if a more complex triplet embedding learner network could provide better results

Anchor Embedding (2048)

Anchor Feature Vector

Positive Feature Vector

Positive Embedding (2048)

Input Layer (2048)

Negative Feature Vector

Negative Embedding (2048)