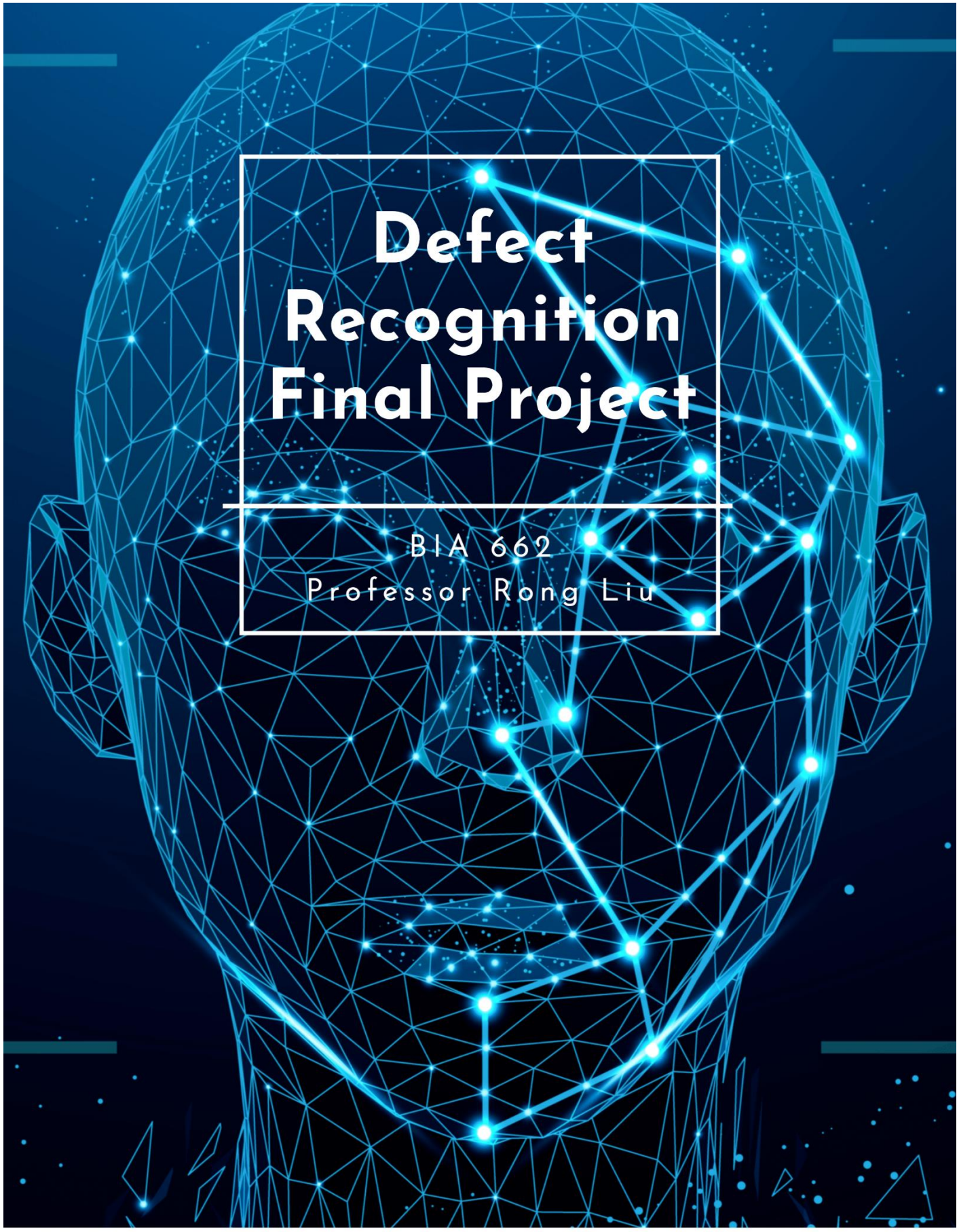

Defect Recognition Final Project

BIA 662

Professor Rong Liu

STEVENS INSTITUTE OF TECHNOLOGY

Kenneth Gan, Vikrant Gajare, Srivedh Chagantipati

May 13, 2022

Contents

Introduction

Problem Description

Quality control is integral in any manufacturing environment; however, it is especially relevant within a regulated production setting such as that of a medical device manufacturer. In addition to more stringent process validations that ensure the equipment, processes, and operators are capable of consistently outputting a product that meets pre-defined technical requirements, product verifications during in-process inspections, ahead of final packaging, help to ensure product quality. Nevertheless, manual inspections within a large-scale production process are both time-consuming and, more importantly, error prone [1]. Although a number of inspections involve non-destructive functional testing, visual inspections are also frequently employed. These visual inspections are performed by human operators and have been recognized as only being approximately 85% accurate; this necessitate 300% inspections (three, 100% inspections[1]) in order to boost that accuracy to a more acceptable 99.7% level [2]. As such, this has opened the door to the implementation of automated inspections that leverage artificial intelligence (AI), machine vision (vision systems), and deep learning [3]. The prevalence of deep learning and AI techniques within a manufacturer varies drastically, and it is heavily hinged on the production volumes and the ability to employ human labor for the inspections. For this reason, companies that have minimal exposure to this level of automation may be more reluctant to adopt advances made within the technological sector due to regulatory requirements that ensure validation to Good Automated Manufacturing Practice (GAMP) as opposed to the more traditional Good Manufacturing Practices (GMP) [3]. There are even pre-trained models (such as the VGG-16 model explored within this report) that can significantly reduce the costs and time required to implement a classification model, which makes the incorporation of these techniques even more realistic [4, 5]. As companies are exposed to the accuracy advantages that these systems offer, implementation of such techniques provides a robust means to supplement human inspections in a seamless manner. Ultimately, this ensures product quality, while offering support to an oftentimes depleted workforce in the wake of the COVID-19 pandemic.

Fully realizing the benefits of deep learning within the highly regulated medical device manufacturing industry is not as simple as constructing an image classification model. This is due in part to the need to validate all inspection systems along with the fact that these deep learning models are limited to binary “pass” or “fail” acceptance criteria without the ability to provide a detailed explanation for the resulting output [3]. Once these hurdles become better addressed, medical manufacturers certainly have utility for AI and Deep Learning applications to lean out manufacturing processes without compromising the high-degree of product quality that must be maintained for a variety of different product lines [3]. Aside from the mere challenges that impede the implementation of these techniques from a regulatory perspective, another central challenge with training these algorithms involves the availability of training data.

Business Question: Can deep learning techniques be leveraged to allow for image classification to accurately detect defects within a production setting, which can scaled for use in a high-volume manufacturing environment?

[1] Here, 100% inspections refer to an inspection that is performed on 100% of the units manufactured.

Challenges

Regulatory Requirements

The major challenge associated with this particular use case involves the regulatory implications involved with any type of manufacturing within the medical device industry. Not only are stringent validations required on the process itself to demonstrate consistency, but the in-process inspections must adhere to stringent requirements. In order to demonstrate acceptability, these inspections must meet pre-defined acceptance criteria involved with the amount of alpha and beta errors allowed during classifications[2]. In addition, any type of software that is incorporated into the inspection process, whether it is for processing or testing, require independent software validations that demonstrate robustness through a variety of use-cases. Although adhering to these requirements are certainly feasible, they require a significant investment in terms of initial labor and capital for development of the early framework.

Executive Acceptance

There are two main factors that are difficult to overcome in terms of AI involvement within a production environment. Executive leadership will be concerned with how well the system can perform in comparison to the human inspector alternative to ensure product quality improved (or at least maintained) as well as how readily the system can be implemented. The most important consideration during a decision making process within this industry involves overall product quality. If the system cannot achieve results that ensure bad product will not leave the door, it is not worth exploring. In addition, if there is not a readily implementable solution, it is unlikely that its value can be demonstrated in comparison with the traditional approach of replicate inspections. As such, the ultimate goal of this project is to demonstrate that Deep Learning models can be readily implemented and trained to score perfect results in routine inspection situations. As acceptance and interest heightens, this will allow for the more nuanced inspections to be explored to ideally redefine the typical inspection process within the manufacturing setting.

Dataset Description

Data used within the scope of this project was deliberately collected for use during this study, which involved using a 16MB Polaroid digital camera. The 910 images taken for this study were compressed during modeling to allow for more manageable computation times despite the use of colored images for the training and testing of the neural network models. A sample of this data has been included within Attachment 2, for reference. Because the intent was to utilize the images for classification of two different defects, the images involved the following classifications[3]:

1. Conforming/Acceptable Sample
2. Non-conforming Sample (Defect 1)
3. Non-conforming Sample (Defect 2)

---

[2] Alpha error corresponds to allowing a non-conforming part to proceed, which are the greater concern in comparison to Beta errors, which are scrapping or reworking a conforming product.

[3] As an extension to this project, the modeling should look to become more robust through use of multi-label classification, which will be elaborated further within a later section. For the intents of this final project analysis, however, the predicted and test labels fall into one of the four classifications.

Group 1, which was composed of the conforming samples, included 498 images, and the other non-conforming sample group contained 412 images. The images were all manually labeled and involved twenty different assemblies photographed in a variety of slightly different orientations. This helped to simulate the intended application of the full-scale model, capturing images of product as it moves along the assembly line ahead of final packaging. A major benefit to the intended application involves the fact that all product is nearly identical in nature as the robust manufacturing process has been designed such that there is minimal variation between products, especially when it comes to the assemblies included within the finished kits. As such, the most important aspect to consider when capturing images for the training and test sample sets involved getting a variety of slightly different tray orientations, shadow location as well as manually adjusting the positioning of the parts within the limits of the tray cavities. This helped ensure there is no unrecognized bias in the data to cause unintended classifications based upon unimportant factors such as lighting (shadows), tray positioning beneath the image capturing equipment, and orientation of the assemblies within confides of the tray.

Ahead of modeling, data was split into the following three sample groups:

1. Train (60% - 546 images)
2. Validation (20% - 182 images)
3. Test (20% - 182 images)

The distribution of the sample classes amongst the three groups is depicted within Figures 1+2.

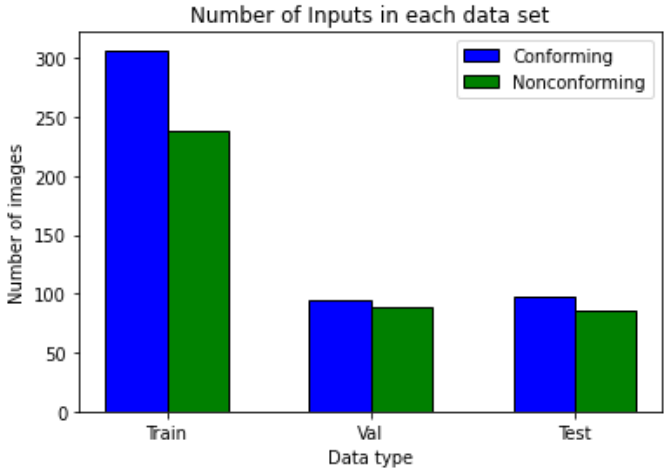

Figure 1 - Distribution of sample classes used for BASELINE amongst Train, Validation, and Test sample sets; this sample set was also utilized for the pre-trained VGG and Modified VGG models

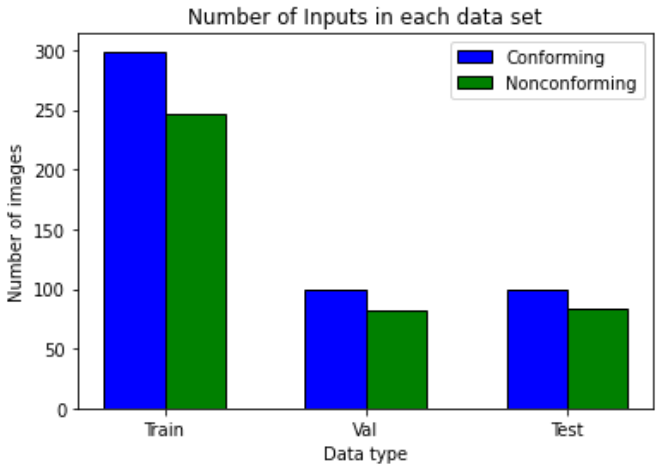

Figure 2 - Distribution of sample classes used for AUGMENTED CNN amongst Train, Validation, and Test sample sets

As the volume of each nonconforming class increases, the accuracy and predictive power of the model will certainly improve. This was a major limitation to this initial proof-of-concept; however, the techniques explored and evaluated demonstrate the utility that full-scale implementation has to offer. The resolution and size of sample groups was reduced both to challenge the models' abilities to learn as well as develop reasonably acceptable training times. As will be discussed during the analysis section of this report, the training times still oftentimes exceeded two (2) hours to fully run even with a Google Colaboratory PRO+ account.

Approach & State-of-the-Art Techniques

Models used

The business problem at hand was addressed by designed several different models to attempt to attain the highest test accuracy possible. In this case, the problem was rooted as a binary classification problem to assess whether a particular device was conforming (packed correctly). This will be touched upon in greater detail within the conclusion; however, an extension and potential means to improve the results would be to formulate a non-binary classification problem, or even multi-label classification. The binary approach serves as a sound method to demonstrate the model proof-of-concept, which can only be built upon moving forward through the collection of larger volumes and better quality of data. In order to challenge the modeling process, images were collected with a lower resolution digital camera using a variety of different angles. In a routine production setting this would be far more automated and robust. Although higher resolution images would certainly extend training times, this could be readily addressed through more precise, targeted modeling techniques.

Binary classification was first performed on the raw dataset through the use of a Convoluted Neural Network (CNN or ConvNet) as a baseline. This applied the Binary Cross Entropy (BCE) loss function as a means to train the model. After the baseline was established, more robust transformations and pre-trained networks could be applied for comparison.

CNN Model

Before delving into the specific as to the architecture of the CNN used for the purposes of the baseline to compare against, it is critical to understand how CNNs operate to determine the best

approaches for improving upon deficiencies that are leading to incorrect classifications. CNNs are a branch of Deep Learning algorithms that receive an input image[4], assign an importance to defining details that may be readily apparent or minute, and progressively learn to assist in the differentiation between classes [6, 7, 8]. An advantageous feature to CNNs is that they oftentimes require minimal pre-processing to effectively operate [6, 7]. Subsequent sections will detail ways in which data can be transformed and pre-processed ahead of interpretation to attempt to improve image recognition and classification; however, the baseline demonstrates that the only significant preprocessing required was labeling and compressing the image size[5]. The use of filters allows for tunable parameters to be implemented during development of the network; however, there are a number of hidden parameters involved that can be trained through cyclical training. Refer to Figure 3 for a graphic displaying the major aspects to these networks.

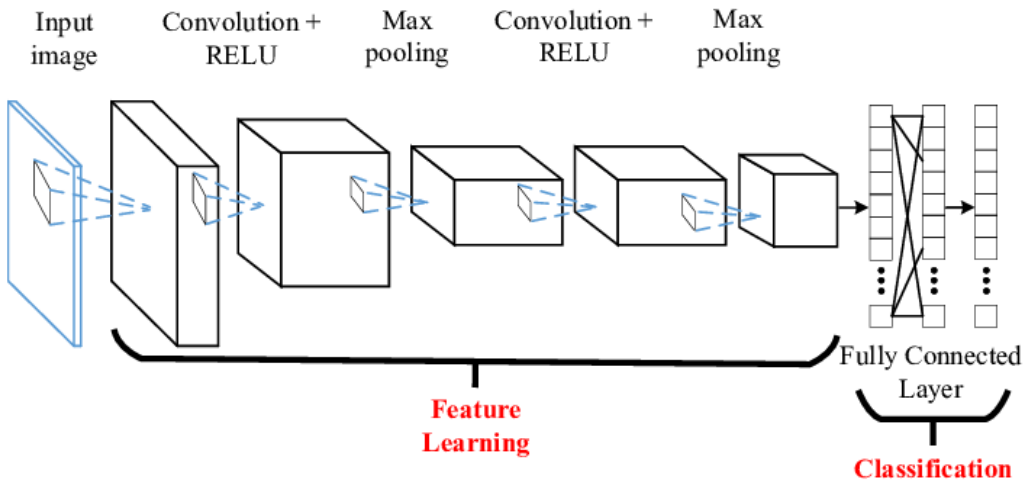

Figure 3 - Example architecture of CNN, which includes two (2) major regions; feature learning and classification. Image was derived from Kamencay et al. [8]

In addition to the fundamental input and hidden layers, there are several aspects to the architecture worth elaborating further upon. Namely, the convolutional layers consist of a set of learnable filters that have a small receptive field and extend through the full depth of input [7]. There is first a forward pass where the filter is convolved across both dimensions of the input volume, which computes a dot product between filter entries and the input to produce a two-dimensional (2D) activation map. Ultimately, this strategy allows the network to learn when filters should activate to determine specific features at spatial locations within the image [7]. This supports the extraction of high-level features such as edges. The use of multiple convolutional layers allows for different levels of extraction to occur. For example, while the first layer is capable of extracting the lower level features (edges, color, gradient orientation, etc.), the added layers allow for the high-level features to be discerned as well [8]. The end result is a network with a broad understanding of the images within the dataset in a comparable manner to that of a human.

[4] In this case, the business problem requires the use of a CNN for image classification, but they can be applied to various applications depending on the input data. Text processing for sentiment analysis is one such example.
[5] This assists in the processing speed of the model during training/testing.

After defining the convolutional layers, the remaining aspects that must be considered involve the activation function used, which defines the manner in which the "neurons"[6] are fired for the application of gradients, as well as the use of slightly more complex analyses such as Max Pooling. The ultimate goal of the pooling layers is to reduce the spatial size of outputs from the convolutional layers, which is why it is common to insert pooling layers between successive convolutional layers; this can facilitate the reduction of tunable parameters within subsequent layers, memory footprint, as well as the amount of overall computation within the network. As such, this is a sound method utilized to prevent overfitting of the model as the pooling returns the maximum value from the region covered by the filters.

Baseline CNN

The baseline CNN model was made up of four (4) convolutional layers with the ReLU activation function and max pooling applied. This portion of the feature learning section of the network yielded an output for interpretation through the fully connected, linear classification layer. The output from this fully-connected layer is then interpretable to make a binary classification. Refer to Figure 4 for the framework used within the scope of the baseline model using a batch size of 32; however, the batch size can be readily adjusted and does not impact the number of trainable parameters included within our model (3,453,121).

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CNN_Classifier                           --                        --
├─Sequential: 1-1                        [64, 6272]                --
│    └─Conv2d: 2-1                       [64, 32, 148, 148]        896
│    └─ReLU: 2-2                         [64, 32, 148, 148]        --
│    └─MaxPool2d: 2-3                    [64, 32, 74, 74]          --
│    └─Conv2d: 2-4                       [64, 64, 72, 72]          18,496
│    └─ReLU: 2-5                         [64, 64, 72, 72]          --
│    └─MaxPool2d: 2-6                    [64, 64, 36, 36]          --
│    └─Conv2d: 2-7                       [64, 128, 34, 34]         73,856
│    └─ReLU: 2-8                         [64, 128, 34, 34]         --
│    └─MaxPool2d: 2-9                    [64, 128, 17, 17]         --
│    └─Conv2d: 2-10                      [64, 128, 15, 15]         147,584
│    └─ReLU: 2-11                        [64, 128, 15, 15]         --
│    └─MaxPool2d: 2-12                   [64, 128, 7, 7]           --
│    └─Flatten: 2-13                     [64, 6272]                --
├─Sequential: 1-2                        [64, 1]                   --
│    └─Linear: 2-14                      [64, 512]                 3,211,776
│    └─ReLU: 2-15                        [64, 512]                 --
│    └─Linear: 2-16                      [64, 1]                   513
==========================================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
Total mult-adds (G): 15.19
==========================================================================================
Input size (MB): 17.28
Forward/backward pass size (MB): 619.51
Params size (MB): 13.81
Estimated Total Size (MB): 650.60
==========================================================================================
```

Figure 4 - Breakdown of baseline CNN used for initial evaluation assessment

[6] The name "neuron" is applied as these networks operate in a comparable manner to the human nervous system. Instead of firing neurotransmitters to transmit a signal to the human brain, these networks utilize activation functions to dictate how information is transmitted between the layers of the network.

An important extension to the baseline CNN involves the transformation of the images to provide an augmented image for the model to interpret. This was the second type of model employed within this final project.

CNN with Data Augmentation

To improve the baseline performance (accuracy and consistency), data augmentation was utilized to artificially expand the dataset to further avoid overfitting challenges. This can include an array of manipulations, which were explored before ultimately incorporating into the CNN model. In the case of this dataset, the use of options for rotation could prove to be useful to account for the fact that most of the images are oriented quite similarly to one another. In addition, the shadows and lighting cause inherently minute differences within the data; however, we can look to emphasize particular regions of interest through transformations such as Grayscale and Color Jitter, which have several parameters that can be tuned to receive an optimal representation of the image for this use-case. These types of transformations can all be applied in an isolated or compounding fashion such that the overall test accuracies can be compared before determining which worked “best”.

As previously alluded to, the Color Jitter transformation was applied and tuned using several parameters that impact the lighting and color of the image being interpreted by the model. This includes the following, which were explored within the augmented CNN model implementation:

1. Brightness – Determines how much to jitter (manipulate) the brightness. The brightness factor is chosen as a uniformly applied value or a range between a minimum and maximum. These values should be non-negative values in the form of a float or tuple (min, max).
2. Contrast – Determines the contract factor that is chosen to be applied at either a uniform or range based manner (float or tuple with minimum and maximum values). Adjusting this parameter can help to exaggerate differences within particular sections of our image, and it should contain non-negative values.
3. Saturation – Determines the saturation factor to be applied. This follows the same requirements as brightness and contrast; however, it has different effects on the pixels included within the color images.
4. Hue – Determines the hue factor to be applied over the range of [-hue, hue] or between the provided minimum and maximum values (tuple). The requirements of this input are that the parameter should be non-negative to allow for conversion into the hue, saturation, value (HSV) space.
5. Grayscale – Determines the probability to randomly convert the image to grayscale. This can be set within a non-negative range from zero (0) to one (1).

Examples for a couple of the transformations are documented within Figures 5 + 6; however, the model was explored using several combinations before yielding the results described within subsequent sections of this report.

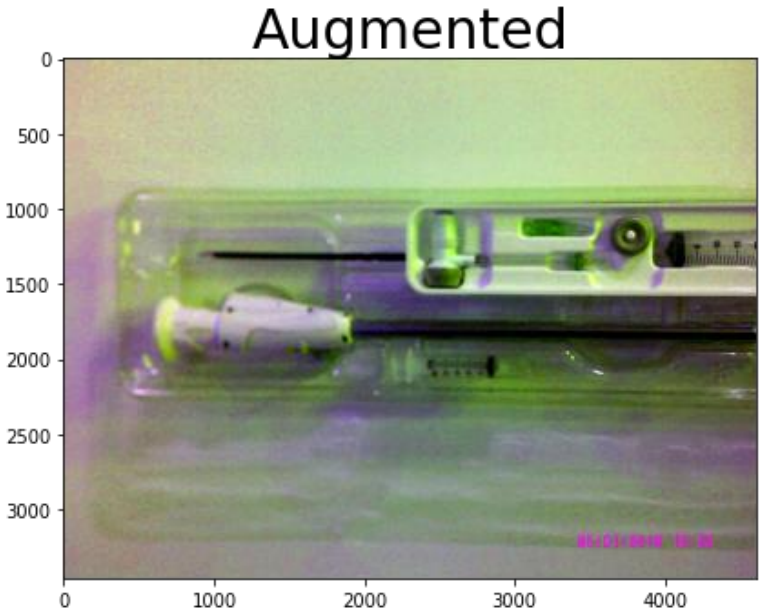

Figure 5 - Example of Color Jitter applied to a conforming sample. The parameters used for the finalized jitter transformation included a brightness of 0.2, contract of 0.9, saturation of 0.8, and a hue of 0.5.

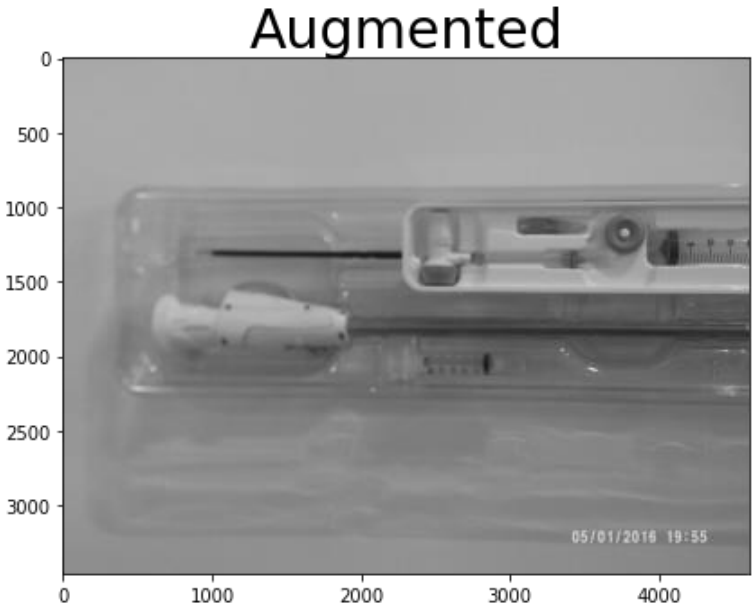

Figure 6 - Example of grayscale applied to a conforming sample.

After manipulating the various transformations at the model's disposal, the color jitter and grayscale were focused on due to their ability to capture regions of interest. The results section will touch upon the outputs obtained from combinations/manipulations of these transforms. Following each of the transformations, a compression transform step was executed to reduce the image size down to a more manageable state to improve run times and serve as a pseudo-normalization step.

VGG-16 Pre-Trained Model

The VGG-16 pre-trained model is a standard deep CNN constructed by Andrew Zisserman and Karen Simonyan. The name stands for Visual Geometry Group, and it refers to the department in which the creators conduct their research. The network contains a total of 16 layers with an input size of 224 x 224 x 3 [9]. The arrangements of the layers (convolutional and pooling) within the model are the fundamental architecture, and they have been systematically constructed to precisely train datasets for a wide array of image classification applications. Refer to Figure 7 for a description of each of the layers within the VGG-16 model. Both the architecture and the weights used within the VGG-16 model are readily imported using the Keras library, which is what this final project sought to employ. Although the weights are pre-established, we can selectively "unfreeze" these weights to compare the affect that adjusting some parameters can have on the dataset being classified. Throughout the duration of the report, the frozen version (VGG-16) shall be differentiated from the selectively unfrozen version (Modified VGG-16). In the case of this model, we elected to freeze the weights for all of the biases applied and all of the weights aside from Layer 24, Layer 16, and Layer 28. This strategy often supplements the application of user-defined classifiers. Defining the classifier allows the classification approach to be made more suitable for the input data, while the application of gradients helps to improve the learning process during the later layers of the CNN. This ensures the more generic aspects that the early layers capture can remain, while tailoring the ending to the more nuanced details. Refer to Figure 8 for a representation of layers and their state (frozen or unfrozen). A representation of this was update within the Modified VGG-16 was output ahead of modeling within this project and can be visualized in Figure 7, which was derived from Simonyan et al [9].

Overall, the VGG-16 results have been documented in literature to outperform previous iterations of the model, and was the classification task winner with performances that was superior to GoogLeNet by 0.9%. Refer to Figure 9 for a summary of the comparison with other, robust pre-trained models.

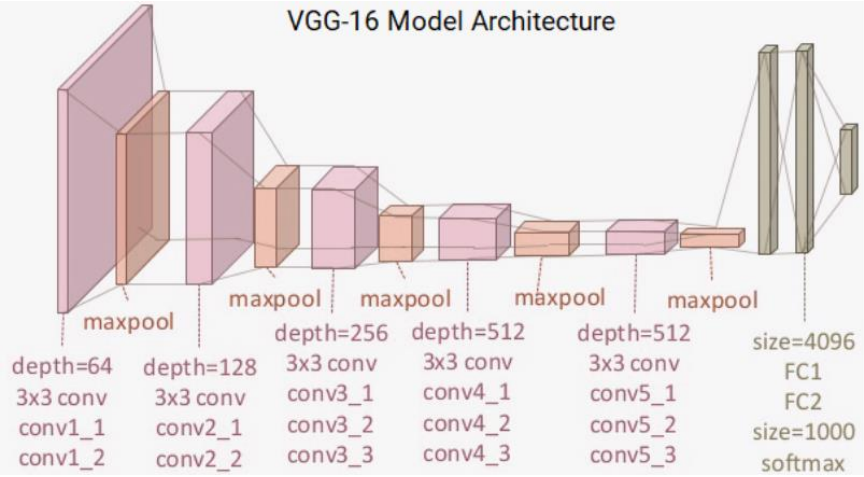

Figure 7 - Architecture of the VGG-16 pre-trained model. Image sourced from https://ebookreading.net/view/book/EB9781788831307_191.html

```
Layer 0: features.0.weight freezed
Layer 0: features.0.bias freezed
Layer 2: features.2.weight freezed
Layer 2: features.2.bias freezed
Layer 5: features.5.weight freezed
Layer 5: features.5.bias freezed
Layer 7: features.7.weight freezed
Layer 7: features.7.bias freezed
Layer 10: features.10.weight freezed
Layer 10: features.10.bias freezed
Layer 12: features.12.weight freezed
Layer 12: features.12.bias freezed
Layer 14: features.14.weight freezed
Layer 14: features.14.bias freezed
Layer 17: features.17.weight freezed
Layer 17: features.17.bias freezed
Layer 19: features.19.weight freezed
Layer 19: features.19.bias freezed
Layer 21: features.21.weight freezed
Layer 21: features.21.bias freezed
Layer 24: features.24.weight not freezed
Layer 24: features.24.bias not freezed
Layer 26: features.26.weight not freezed
Layer 26: features.26.bias not freezed
Layer 28: features.28.weight not freezed
Layer 28: features.28.bias not freezed
```

Figure 8 - Architecture of the Modified VGG-16 pre-trained model.

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	**23.7**	**6.8**	**6.8**
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-	7.9	
GoogLeNet (Szegedy et al., 2014) (7 nets)	-	**6.7**	
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

Figure 10 - Performance of VGG-16 model; image sourced from Simonyan et al. [10]

Baseline Model Results

The baseline model was implemented using a multi-layer CNN structure as depicted within Figure 4. Each of the convolutional layers contained filters (kernels) with a size of three (3), which indicates the number of pixels used within the filtering process of the layer. Their input and output values shapes were adjusted during the initial training process, and it was determined that the model employed delivered a suitable point to being more complicated extensions to the model. An important consideration when reviewing the results of model implementation involves the fact that each subsequent run will yield slightly modified results. As such, the outputs provided within this report may not align precisely with the Notebook provided as an attachment (Attachment 1).

Using a learning rate of 0.005 and batch size of 32 yielded a promising starting point to build upon in subsequent modeling techniques. The number of epochs was set to 50 after several longer runs were employed. Although the loss values were still somewhat declining at this point, they were quite erratic, which can be visualized within Figures 10 + 11. Another noticeable attribute to this model involved the extended number of epochs required to appropriately train the model to improve validation scores.

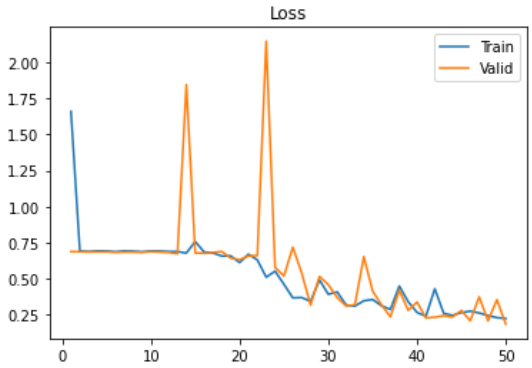

Figure 10 - Loss curve for the training and validation data; loss was relatively stagnant up until epoch 20 with several instances of erratically large losses

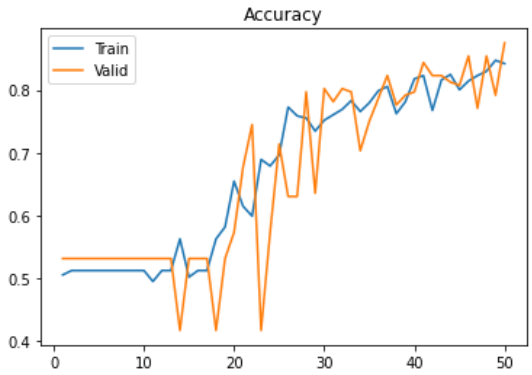

Figure 11 - Accuracy curve for the training and validation data; accuracy was relatively stagnant up until epoch 20 with several instances of erratically low accuracies; following this point, the accuracy was increasing, but remained quite erratic

In this case, it was not until epoch 20 / 50 that validation accuracy began to improve, which caused a lengthy training period for this model. More details comparing the results with subsequent models is included; however, these were the main takeaways from the Loss and Accuracy curves displayed. In addition, this model was successful in accurately assigning labels for 90.7% of the testing data, which exceeded any individual validation accuracy for an epoch. This can be visually observed in Figure 12, and it provides a sound starting point in supplementing the human visual inspections with a noted accuracy of approximately 85%. The total runtime for this training approach lasted just over two and a half hours.

These curves also highlight the previously alluded to deficiency in this type of initial proof-of-concept modeling approach. Because there is limited data to not only train, but also test to, there can be erratic trials where the loss increases and accuracy plummets. With greater volume of data, which would certainly be available in a production setting, the training times would increase with the added benefit of a much more robust output that is reliable for all portions of the dataset (train/validation/test).

```
Epoch:12 / 50, train loss:0.6881 train_acc:0.5122, valid loss:0.6820 valid acc:0.5312
Epoch:13 / 50, train loss:0.6880 train_acc:0.5122, valid loss:0.6729 valid acc:0.5312
Epoch:14 / 50, train loss:0.6777 train_acc:0.5625, valid loss:1.8434 valid acc:0.4167
Epoch:15 / 50, train loss:0.7573 train_acc:0.5017, valid loss:0.6783 valid acc:0.5312
Epoch:16 / 50, train loss:0.6854 train_acc:0.5122, valid loss:0.6767 valid acc:0.5312
Epoch:17 / 50, train loss:0.6787 train_acc:0.5122, valid loss:0.6821 valid acc:0.5312
Epoch:18 / 50, train loss:0.6569 train_acc:0.5625, valid loss:0.6880 valid acc:0.4167
Epoch:19 / 50, train loss:0.6584 train_acc:0.5816, valid loss:0.6425 valid acc:0.5312
Epoch:20 / 50, train loss:0.6116 train_acc:0.6545, valid loss:0.6317 valid acc:0.5729
Epoch:21 / 50, train loss:0.6694 train_acc:0.6146, valid loss:0.6579 valid acc:0.6771
Epoch:22 / 50, train loss:0.6303 train_acc:0.5990, valid loss:0.6605 valid acc:0.7448
Epoch:23 / 50, train loss:0.5107 train_acc:0.6892, valid loss:2.1451 valid acc:0.4167
Epoch:24 / 50, train loss:0.5522 train_acc:0.6788, valid loss:0.5778 valid acc:0.5729
Epoch:25 / 50, train loss:0.4615 train_acc:0.6962, valid loss:0.5188 valid acc:0.7135
Epoch:26 / 50, train loss:0.3674 train_acc:0.7726, valid loss:0.7181 valid acc:0.6302
Epoch:27 / 50, train loss:0.3696 train_acc:0.7587, valid loss:0.5400 valid acc:0.6302
Epoch:28 / 50, train loss:0.3431 train_acc:0.7552, valid loss:0.3173 valid acc:0.7969
Epoch:29 / 50, train loss:0.4918 train_acc:0.7344, valid loss:0.5156 valid acc:0.6354
Epoch:30 / 50, train loss:0.3922 train_acc:0.7517, valid loss:0.4589 valid acc:0.8021
Epoch:31 / 50, train loss:0.4085 train_acc:0.7604, valid loss:0.3696 valid acc:0.7812
Epoch:32 / 50, train loss:0.3186 train_acc:0.7691, valid loss:0.3116 valid acc:0.8021
Epoch:33 / 50, train loss:0.3107 train_acc:0.7830, valid loss:0.3207 valid acc:0.7969
Epoch:34 / 50, train loss:0.3472 train_acc:0.7656, valid loss:0.6532 valid acc:0.7031
Epoch:35 / 50, train loss:0.3562 train_acc:0.7795, valid loss:0.4136 valid acc:0.7500
Epoch:36 / 50, train loss:0.3124 train_acc:0.7986, valid loss:0.3187 valid acc:0.7865
Epoch:37 / 50, train loss:0.2873 train_acc:0.8056, valid loss:0.2359 valid acc:0.8229
Epoch:38 / 50, train loss:0.4492 train_acc:0.7622, valid loss:0.4202 valid acc:0.7760
Epoch:39 / 50, train loss:0.3408 train_acc:0.7812, valid loss:0.2814 valid acc:0.7917
Epoch:40 / 50, train loss:0.2662 train_acc:0.8177, valid loss:0.3379 valid acc:0.7969
Epoch:41 / 50, train loss:0.2402 train_acc:0.8229, valid loss:0.2284 valid acc:0.8438
Epoch:42 / 50, train loss:0.4303 train_acc:0.7674, valid loss:0.2336 valid acc:0.8229
Epoch:43 / 50, train loss:0.2603 train_acc:0.8160, valid loss:0.2442 valid acc:0.8229
Epoch:44 / 50, train loss:0.2461 train_acc:0.8247, valid loss:0.2351 valid acc:0.8125
Epoch:45 / 50, train loss:0.2659 train_acc:0.8003, valid loss:0.2800 valid acc:0.8073
Epoch:46 / 50, train loss:0.2755 train_acc:0.8142, valid loss:0.2090 valid acc:0.8542
Epoch:47 / 50, train loss:0.2630 train_acc:0.8229, valid loss:0.3745 valid acc:0.7708
Epoch:48 / 50, train loss:0.2444 train_acc:0.8299, valid loss:0.2077 valid acc:0.8542
Epoch:49 / 50, train loss:0.2304 train_acc:0.8472, valid loss:0.3546 valid acc:0.7917
Epoch:50 / 50, train loss:0.2244 train_acc:0.8420, valid loss:0.1851 valid acc:0.8750
test acc:0.907
```

Figure 12 - Baseline test output; stagnant changes observed initially before achieving much higher accuracies throughout later epochs

Using the early stopping approach[7], the best model in terms of validation accuracy could be utilized for testing; however, this can still be predisposed to a more unbalanced test set that does not perform quite as well. After establishing an effective baseline, Data Augmentation was the first approach implemented to assess whether this baseline could be improved upon as alluded to in preceding sections.

Experimentation and Results Analysis

Data Augmentation Modeling Results

Despite incorporating several augmentation techniques to reduce the effect of overfitting to the training data, there were two transformations that stood out in terms of ability to consistently perform well, RandomGrayscale and ColorJitter. When applied separately, they were quite powerful in overall test accuracy; however, they were far less effective when incorporated together in a combined approach. For reference, a snippet of the training process for the combined approach has been included below to help in visualizing the stagnant loss and accuracy values. Because this model was not effective in reducing the loss, it was foregone, and the independent implementations became the focus of this Data Augmentation portion. Refer to Figure 13.

[7] In this example, early stopping refers to utilizing the best available model. Because the accuracies included highly stagnant periods, the model should continue without regarding the patience values. Nevertheless, the "best" model in terms of validation accuracy at least provides an expectation that similar success would be achieved on the test set.

```
model, history_augmented = train_model(cnn_model, train_data_trans, valid_data_trans, device, batch_size=16, epochs=50, lr=0.0005)

Training Start
Epoch:1 / 50, train loss:1.0749 train_acc:0.5054, valid loss:0.6891 valid acc:0.5104
model saved!
Epoch:2 / 50, train loss:0.6909 train_acc:0.5357, valid loss:0.6891 valid acc:0.5104
Epoch:3 / 50, train loss:0.6895 train_acc:0.5357, valid loss:0.6911 valid acc:0.5104
Epoch:4 / 50, train loss:0.6879 train_acc:0.5429, valid loss:0.8162 valid acc:0.5104
Epoch:5 / 50, train loss:0.6956 train_acc:0.5250, valid loss:0.6893 valid acc:0.5104
Epoch:6 / 50, train loss:0.6894 train_acc:0.5357, valid loss:0.6889 valid acc:0.5104
Epoch:7 / 50, train loss:0.6896 train_acc:0.5357, valid loss:0.6880 valid acc:0.5104
Epoch:8 / 50, train loss:0.6911 train_acc:0.5357, valid loss:0.6931 valid acc:0.5104
Epoch:9 / 50, train loss:0.6914 train_acc:0.5357, valid loss:0.6910 valid acc:0.5104
Epoch:10 / 50, train loss:0.6905 train_acc:0.5357, valid loss:0.6889 valid acc:0.5104
Epoch:11 / 50, train loss:0.6923 train_acc:0.5357, valid loss:0.6902 valid acc:0.5104
Epoch:12 / 50, train loss:0.6900 train_acc:0.5357, valid loss:0.6890 valid acc:0.5104
Epoch:13 / 50, train loss:0.6871 train_acc:0.5357, valid loss:0.6988 valid acc:0.5104
Epoch:14 / 50, train loss:0.6900 train_acc:0.5250, valid loss:0.7812 valid acc:0.5104
Epoch:15 / 50, train loss:0.6914 train_acc:0.5357, valid loss:0.6891 valid acc:0.5104
Epoch:16 / 50, train loss:0.6850 train_acc:0.5304, valid loss:0.7520 valid acc:0.5104
Epoch:17 / 50, train loss:0.6976 train_acc:0.5589, valid loss:0.6784 valid acc:0.5104
```

Figure 13 - Foregone, combined approach to use Color Jitter and Grayscale in a pipeline

Visualizing the images after these transformations (Figure 6 and Figure 7), it makes sense how these types of changes can offer a means to better capture the regions of interest within the images as the loss and accuracy still contained sharp peaks, but it was far less exacerbated than the baseline model to provide a more robust, consistent test accuracy.

Conversely, when run independently, each of these modeling approaches was highly effective in reducing the effects of overfitting and yielding highly accurate models. This was likely due to the fact that defects were rather apparent in the sense that a non-conforming unit was missing an entire component necessary for use. As a result, augmentation techniques that exaggerate these regions can be robust as standalone approaches. The Color Jitter approach was explored first, and it yielded sudden changes in the loss and accuracy; however, it was much more controlled within the later epochs. This provided high confidence that it would be suitable for use on the test dataset. This can be visualized within Figures 14 - 16, which depict the accuracy and loss curves as well as the final epochs used ahead of testing. Once again, the early stopping approach allowed a previously run model to be stored if it yielded the best accuracy on the validation set, which corresponded to a highly accurate test accuracy of greater 99.5%. Similar phenomena were observed using the grayscale, which was applied to 100% of the images to highlight the regions of interest. In this case, the results were equally as accurate, which is depicted within Figures 17 - 18. Compared with the baseline, data augmentation models took only three seconds longer to run, which is negligible.

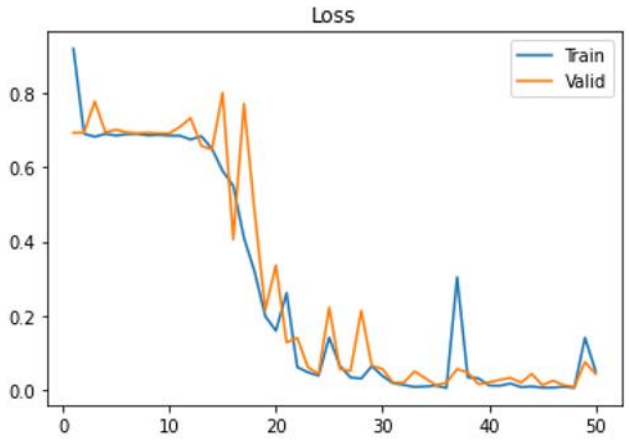

Figure 14 - Loss curve for the training and validation data of the Color Jitter augmented datasets; loss was relatively stagnant up until epoch 12 with several instances of sudden peaks in the graph

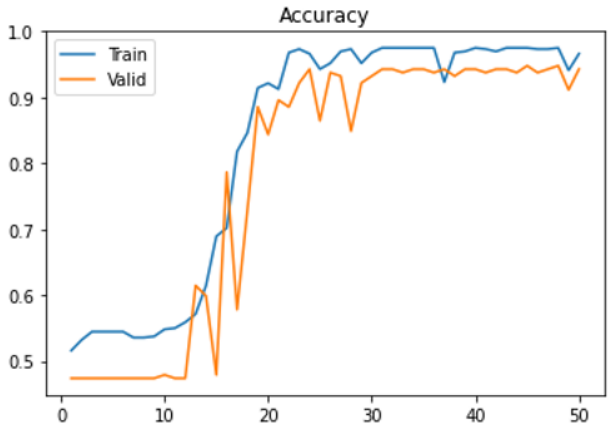

Figure 15 - Accuracy curve for the training and validation data; accuracy was relatively stagnant up until epoch 12 with several instances of sudden peaks in the graph

```
Epoch:6 / 50, train loss:0.6898 train_acc:0.5446, valid loss:0.6947 valid acc:0.4740
Epoch:7 / 50, train loss:0.6908 train_acc:0.5357, valid loss:0.6926 valid acc:0.4740
Epoch:8 / 50, train loss:0.6866 train_acc:0.5357, valid loss:0.6940 valid acc:0.4740
Epoch:9 / 50, train loss:0.6879 train_acc:0.5375, valid loss:0.6923 valid acc:0.4740
Epoch:10 / 50, train loss:0.6860 train_acc:0.5482, valid loss:0.6921 valid acc:0.4792
Epoch:11 / 50, train loss:0.6855 train_acc:0.5500, valid loss:0.7096 valid acc:0.4740
Epoch:12 / 50, train loss:0.6755 train_acc:0.5589, valid loss:0.7333 valid acc:0.4740
Epoch:13 / 50, train loss:0.6845 train_acc:0.5714, valid loss:0.6585 valid acc:0.6146
Epoch:14 / 50, train loss:0.6505 train_acc:0.6143, valid loss:0.6482 valid acc:0.5990
Epoch:15 / 50, train loss:0.5906 train_acc:0.6893, valid loss:0.8016 valid acc:0.4792
Epoch:16 / 50, train loss:0.5503 train_acc:0.7018, valid loss:0.4058 valid acc:0.7865
Epoch:17 / 50, train loss:0.4101 train_acc:0.8179, valid loss:0.7718 valid acc:0.5781
Epoch:18 / 50, train loss:0.3184 train_acc:0.8464, valid loss:0.4807 valid acc:0.7292
Epoch:19 / 50, train loss:0.1984 train_acc:0.9143, valid loss:0.2155 valid acc:0.8854
Epoch:20 / 50, train loss:0.1589 train_acc:0.9214, valid loss:0.3357 valid acc:0.8438
Epoch:21 / 50, train loss:0.2609 train_acc:0.9125, valid loss:0.1276 valid acc:0.8958
Epoch:22 / 50, train loss:0.0609 train_acc:0.9679, valid loss:0.1399 valid acc:0.8854
Epoch:23 / 50, train loss:0.0467 train_acc:0.9732, valid loss:0.0607 valid acc:0.9219
Epoch:24 / 50, train loss:0.0374 train_acc:0.9661, valid loss:0.0421 valid acc:0.9427
Epoch:25 / 50, train loss:0.1407 train_acc:0.9429, valid loss:0.2218 valid acc:0.8646
Epoch:26 / 50, train loss:0.0641 train_acc:0.9518, valid loss:0.0560 valid acc:0.9375
Epoch:27 / 50, train loss:0.0327 train_acc:0.9696, valid loss:0.0503 valid acc:0.9323
Epoch:28 / 50, train loss:0.0298 train_acc:0.9732, valid loss:0.2130 valid acc:0.8490
Epoch:29 / 50, train loss:0.0633 train_acc:0.9518, valid loss:0.0645 valid acc:0.9219
Epoch:30 / 50, train loss:0.0370 train_acc:0.9679, valid loss:0.0556 valid acc:0.9323
Epoch:31 / 50, train loss:0.0170 train_acc:0.9750, valid loss:0.0186 valid acc:0.9427
Epoch:32 / 50, train loss:0.0125 train_acc:0.9750, valid loss:0.0187 valid acc:0.9427
Epoch:33 / 50, train loss:0.0072 train_acc:0.9750, valid loss:0.0492 valid acc:0.9375
Epoch:34 / 50, train loss:0.0085 train_acc:0.9750, valid loss:0.0312 valid acc:0.9427
Epoch:35 / 50, train loss:0.0107 train_acc:0.9750, valid loss:0.0122 valid acc:0.9427
Epoch:36 / 50, train loss:0.0045 train_acc:0.9750, valid loss:0.0186 valid acc:0.9375
Epoch:37 / 50, train loss:0.3035 train_acc:0.9232, valid loss:0.0552 valid acc:0.9427
Epoch:38 / 50, train loss:0.0329 train_acc:0.9679, valid loss:0.0465 valid acc:0.9323
Epoch:39 / 50, train loss:0.0306 train_acc:0.9696, valid loss:0.0145 valid acc:0.9427
Epoch:40 / 50, train loss:0.0111 train_acc:0.9750, valid loss:0.0195 valid acc:0.9427
Epoch:41 / 50, train loss:0.0103 train_acc:0.9732, valid loss:0.0263 valid acc:0.9375
Epoch:42 / 50, train loss:0.0165 train_acc:0.9696, valid loss:0.0317 valid acc:0.9427
Epoch:43 / 50, train loss:0.0067 train_acc:0.9750, valid loss:0.0188 valid acc:0.9427
Epoch:44 / 50, train loss:0.0085 train_acc:0.9750, valid loss:0.0423 valid acc:0.9375
Epoch:45 / 50, train loss:0.0051 train_acc:0.9750, valid loss:0.0122 valid acc:0.9479
Epoch:46 / 50, train loss:0.0053 train_acc:0.9732, valid loss:0.0240 valid acc:0.9375
Epoch:47 / 50, train loss:0.0079 train_acc:0.9732, valid loss:0.0122 valid acc:0.9427
Epoch:48 / 50, train loss:0.0047 train_acc:0.9750, valid loss:0.0071 valid acc:0.9479
Epoch:49 / 50, train loss:0.1399 train_acc:0.9411, valid loss:0.0741 valid acc:0.9115
Epoch:50 / 50, train loss:0.0498 train_acc:0.9661, valid loss:0.0424 valid acc:0.9427
test acc:0.995
```

Figure 16 - Output from training function with stagnant values up until epoch 12

One noticeable distinction between the two augmentation styles stemmed from the consistency demonstrated by the grayscale transformation. This is observable in the accuracy and loss curves, which plateaued and remained consistent for a majority of the later epochs in the training/validation runs (Refer to Figures 17 - 18). Based upon the ease with which this transformation visualized the input images, it would make sense to apply a randomly assigned grayscale rate, which would stress the modeling process further and potentially make the predictions more robust.

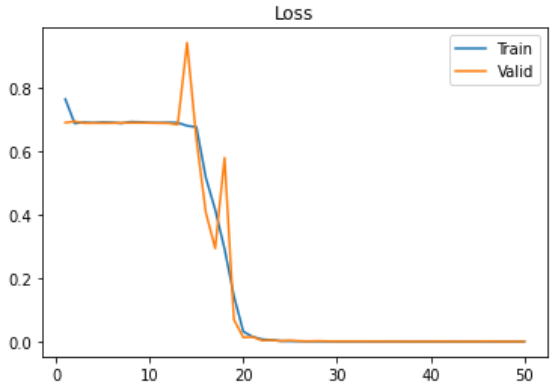

Figure 17 - Loss curve for grayscale transformation; although it took slightly longer to improve compared to Color Jitter, this loss was far more consistent and plateaued without any cause for concern (overfitting)

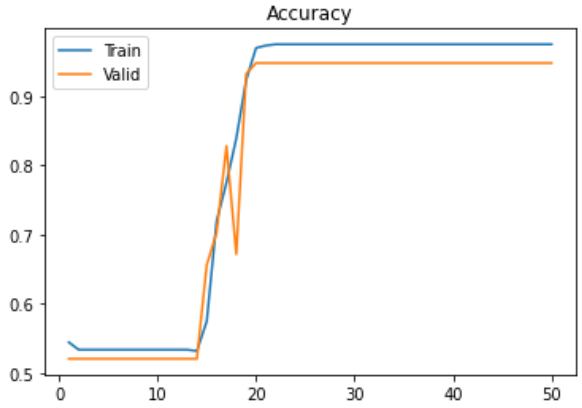

Figure 18 - Loss curve for grayscale transformation; although it took slightly longer to improve compared to Color Jitter, this loss was far more consistent and plateaued without any cause for concern (overfitting)

VGG-16 Modeling Results

After assessing the utility of the data augmentation transformation in classifying the test images, the final approach explored was the VGG-16 pre-trained network model. The implementation of this model was straightforward and required little code manipulation such that this robust model is effectively a “plug-and-play” model; it can readily be applied to image classification problems without much coding needed for consistently predictability. This proved to be the case within our model execution as the loss and accuracy improved much more rapidly than they had within the previously explored network models. In this case, “rapid” refers to the number of epochs required to reach accuracies exceeding the performance of the baseline. The deep nature of this model causes this model to take a longer period of time to train in most applications; however, the model employed within the scope of this final project, it only took XXXXX, which was far quicker than the CNNs that were not pre-trained. These aspects can be visualized within Figures 19 and 20, which relate to the loss and accuracy curves, respectively, for the pre-trained model. One interesting aspect to the model stemmed from the speed with which it could approach the higher accuracies. After applying a reduced learning rate of 0.000001, the model was consistent; however, it was unable to train within the selected number of epochs as evidenced by the consistently reduced loss values. This is likely why the model achieved only a 92.9% accuracy on the test dataset as evidenced by Figure 21.

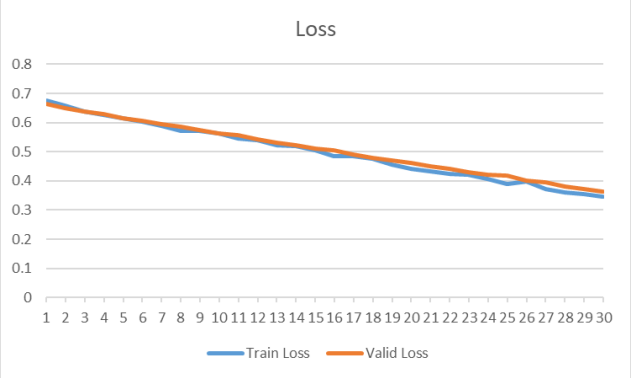

Figure 19 - Loss for VGG-16 with a learning rate of 0.000001 applied; cannot fully train evidenced by continued decrease in Loss (Note: Plot developed based upon tabular output from Notebook ahead of final implementation)

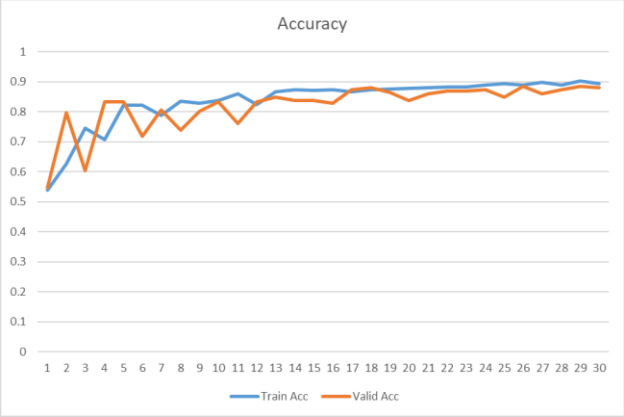

Figure 20 - Loss for VGG-16 with a learning rate of 0.000001 applied; cannot fully train evidenced by continued decrease in Loss (Note: Plot developed based upon tabular output from Notebook ahead of final implementation)

```
history = train_model(VGG_model, train_data, valid_data, test_data, device, batch_size=32, epochs=30, lr=0.000001)

Training Start
Epoch:1 / 30, train loss:0.6746 train_acc:0.5382, valid loss:0.6640 valid acc:0.5469
Epoch:2 / 30, train loss:0.6575 train_acc:0.6267, valid loss:0.6503 valid acc:0.7969
Epoch:3 / 30, train loss:0.6372 train_acc:0.7448, valid loss:0.6393 valid acc:0.6042
Epoch:4 / 30, train loss:0.6274 train_acc:0.7083, valid loss:0.6278 valid acc:0.8333
Epoch:5 / 30, train loss:0.6149 train_acc:0.8229, valid loss:0.6162 valid acc:0.8333
Epoch:6 / 30, train loss:0.6018 train_acc:0.8229, valid loss:0.6070 valid acc:0.7188
Epoch:7 / 30, train loss:0.5883 train_acc:0.7882, valid loss:0.5950 valid acc:0.8073
Epoch:8 / 30, train loss:0.5726 train_acc:0.8351, valid loss:0.5867 valid acc:0.7396
Epoch:9 / 30, train loss:0.5727 train_acc:0.8299, valid loss:0.5746 valid acc:0.8021
Epoch:10 / 30, train loss:0.5637 train_acc:0.8385, valid loss:0.5635 valid acc:0.8333
Epoch:11 / 30, train loss:0.5455 train_acc:0.8594, valid loss:0.5574 valid acc:0.7604
Epoch:12 / 30, train loss:0.5401 train_acc:0.8247, valid loss:0.5425 valid acc:0.8333
Epoch:13 / 30, train loss:0.5214 train_acc:0.8681, valid loss:0.5317 valid acc:0.8490
Epoch:14 / 30, train loss:0.5202 train_acc:0.8750, valid loss:0.5225 valid acc:0.8385
Epoch:15 / 30, train loss:0.5037 train_acc:0.8715, valid loss:0.5121 valid acc:0.8385
Epoch:16 / 30, train loss:0.4847 train_acc:0.8733, valid loss:0.5045 valid acc:0.8281
Epoch:17 / 30, train loss:0.4835 train_acc:0.8663, valid loss:0.4897 valid acc:0.8750
Epoch:18 / 30, train loss:0.4774 train_acc:0.8750, valid loss:0.4800 valid acc:0.8802
Epoch:19 / 30, train loss:0.4565 train_acc:0.8767, valid loss:0.4694 valid acc:0.8646
Epoch:20 / 30, train loss:0.4416 train_acc:0.8785, valid loss:0.4629 valid acc:0.8385
Epoch:21 / 30, train loss:0.4327 train_acc:0.8802, valid loss:0.4509 valid acc:0.8594
Epoch:22 / 30, train loss:0.4245 train_acc:0.8837, valid loss:0.4404 valid acc:0.8698
Epoch:23 / 30, train loss:0.4224 train_acc:0.8837, valid loss:0.4305 valid acc:0.8698
Epoch:24 / 30, train loss:0.4066 train_acc:0.8906, valid loss:0.4208 valid acc:0.8750
Epoch:25 / 30, train loss:0.3883 train_acc:0.8941, valid loss:0.4169 valid acc:0.8490
Epoch:26 / 30, train loss:0.3975 train_acc:0.8906, valid loss:0.4011 valid acc:0.8854
Epoch:27 / 30, train loss:0.3731 train_acc:0.8993, valid loss:0.3965 valid acc:0.8594
Epoch:28 / 30, train loss:0.3613 train_acc:0.8906, valid loss:0.3817 valid acc:0.8750
Epoch:29 / 30, train loss:0.3552 train_acc:0.9028, valid loss:0.3713 valid acc:0.8854
Epoch:30 / 30, train loss:0.3451 train_acc:0.8941, valid loss:0.3637 valid acc:0.8802
test acc:0.929
```

Figure 21 - Training output for VGG-16 pre-trained model with a learning rate of 0.000001 applied; cannot fully train evidenced by continued decrease in Loss and increased accuracy

Once the baseline VGG-16 results were tabulated using a learning rate of 0.000001, it was observed that the model did not have sufficient training time. As such, the rate was increased by a factor of ten (10), which allowed faster training and an even more robust model that correctly predicted 100% of the testing samples. The steady increase in accuracy and decrease in loss can be visualized within Figures 22 + 23, and the overall output is depicted in Figure 24.

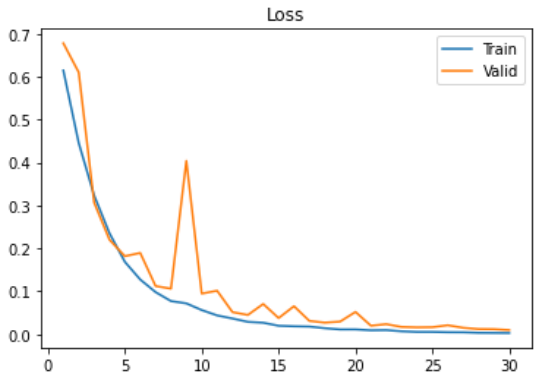

Figure 22 - Loss curve for VGG-16; rapid, consistent decrease in the loss of the model for both train and validation data

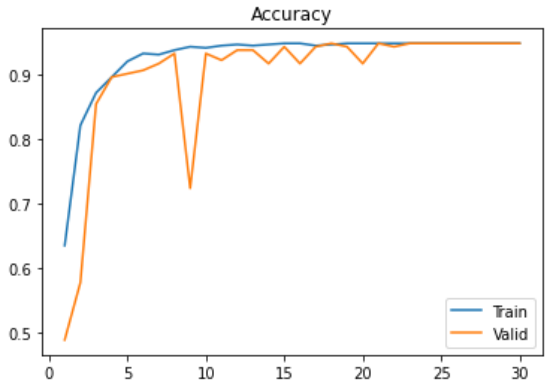

Figure 23 - Loss curve for VGG-16; rapid, consistent decrease in the loss of the model for both train and validation data

```
Training Start
Epoch:1 / 30, train loss:0.6145 train_acc:0.6354, valid loss:0.6777 valid acc:0.4896
model saved!
Epoch:2 / 30, train loss:0.4454 train_acc:0.8212, valid loss:0.6100 valid acc:0.5781
model saved!
Epoch:3 / 30, train loss:0.3234 train_acc:0.8715, valid loss:0.3069 valid acc:0.8542
model saved!
Epoch:4 / 30, train loss:0.2352 train_acc:0.8958, valid loss:0.2200 valid acc:0.8958
model saved!
Epoch:5 / 30, train loss:0.1686 train_acc:0.9201, valid loss:0.1816 valid acc:0.9010
model saved!
Epoch:6 / 30, train loss:0.1273 train_acc:0.9323, valid loss:0.1893 valid acc:0.9062
model saved!
Epoch:7 / 30, train loss:0.0979 train_acc:0.9306, valid loss:0.1120 valid acc:0.9167
model saved!
Epoch:8 / 30, train loss:0.0771 train_acc:0.9375, valid loss:0.1061 valid acc:0.9323
model saved!
Epoch:9 / 30, train loss:0.0717 train_acc:0.9427, valid loss:0.4039 valid acc:0.7240
Epoch:10 / 30, train loss:0.0561 train_acc:0.9410, valid loss:0.0946 valid acc:0.9323
Epoch:11 / 30, train loss:0.0437 train_acc:0.9444, valid loss:0.1012 valid acc:0.9219
Epoch:12 / 30, train loss:0.0365 train_acc:0.9462, valid loss:0.0514 valid acc:0.9375
model saved!
Epoch:13 / 30, train loss:0.0287 train_acc:0.9444, valid loss:0.0447 valid acc:0.9375
Epoch:14 / 30, train loss:0.0266 train_acc:0.9462, valid loss:0.0704 valid acc:0.9167
Epoch:15 / 30, train loss:0.0195 train_acc:0.9479, valid loss:0.0375 valid acc:0.9427
model saved!
Epoch:16 / 30, train loss:0.0183 train_acc:0.9479, valid loss:0.0652 valid acc:0.9167
Epoch:17 / 30, train loss:0.0177 train_acc:0.9444, valid loss:0.0309 valid acc:0.9427
Epoch:18 / 30, train loss:0.0139 train_acc:0.9462, valid loss:0.0269 valid acc:0.9479
model saved!
Epoch:19 / 30, train loss:0.0111 train_acc:0.9479, valid loss:0.0293 valid acc:0.9427
Epoch:20 / 30, train loss:0.0111 train_acc:0.9479, valid loss:0.0519 valid acc:0.9167
Epoch:21 / 30, train loss:0.0090 train_acc:0.9479, valid loss:0.0197 valid acc:0.9479
Epoch:22 / 30, train loss:0.0094 train_acc:0.9479, valid loss:0.0233 valid acc:0.9427
Epoch:23 / 30, train loss:0.0067 train_acc:0.9479, valid loss:0.0168 valid acc:0.9479
Epoch:24 / 30, train loss:0.0053 train_acc:0.9479, valid loss:0.0159 valid acc:0.9479
Epoch:25 / 30, train loss:0.0052 train_acc:0.9479, valid loss:0.0163 valid acc:0.9479
Epoch:26 / 30, train loss:0.0045 train_acc:0.9479, valid loss:0.0208 valid acc:0.9479
Epoch:27 / 30, train loss:0.0044 train_acc:0.9479, valid loss:0.0149 valid acc:0.9479
Epoch:28 / 30, train loss:0.0032 train_acc:0.9479, valid loss:0.0119 valid acc:0.9479
Epoch:29 / 30, train loss:0.0029 train_acc:0.9479, valid loss:0.0116 valid acc:0.9479
Epoch:30 / 30, train loss:0.0026 train_acc:0.9479, valid loss:0.0096 valid acc:0.9479
--- 5558.103278160095 seconds ---
test acc:1.000
```

Figure 24 - Training results for VGG-16 model with decreased learning rate

Once the VGG-16 model was applied in its frozen state, we can observe the fact that the model is fully suited for its intended, application. As discussed within the introduction, this particular defect selected for non-conformances involved a straight forward difference between conforming and non-conforming units. Moving forward, after depicting the utility of the model for the more simplified inspection, the model can be readily expanded to contain much more complicated situations. As such, the project also wanted to consider the VGG-16 pre-trained model beyond its "frozen" state (weights and biases were retained from their pre-defined values). In this case, outputs of interest involved with overall performance becomes speed and consistency. To assess whether this could be achieved through selectively unfreezing layer weights, a slight modification was made to allow for three (3) layer weights to be updated during the training process. As discussed within the description of this model, this serves utility in the sense that the model can train to the sample set provided as opposed to using the more generalized form. Refer to outputs of both the loss and accuracy curves provided in Figures 25 + 26, respectively, which display the consistent, robust nature of this pre-trained model. Another interesting aspect to the curves was the manner in which the model experience a slight decrease in performance (less than 10%) early during the training process before quickly rectifying itself with an optimal output for the remaining epochs. The same effect was noted within the VGG-16 with frozen gradients, which lends to the belief that this robust model attempted to improve before recognizing the spike in loss output. Refer to Figure 27 for a complete output from the run.

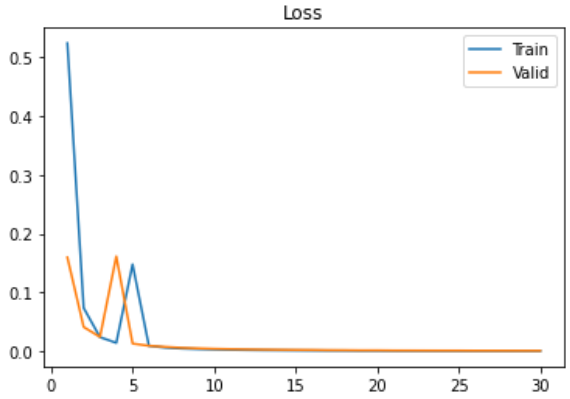

Figure 25 - Loss curve for Modified VGG-16 model; exponentially decreased during first few epochs with minor spike

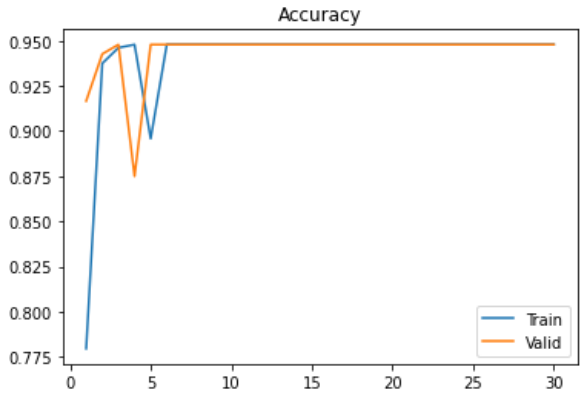

Figure 26 - Accuracy curve for Modified VGG-16 model; plateaued within early portion of training

```
Training Start
Epoch:1 / 30, train loss:0.5244 train_acc:0.7795, valid loss:0.1595 valid acc:0.9167
model saved!
Epoch:2 / 30, train loss:0.0733 train_acc:0.9375, valid loss:0.0408 valid acc:0.9427
model saved!
Epoch:3 / 30, train loss:0.0236 train_acc:0.9462, valid loss:0.0238 valid acc:0.9479
model saved!
Epoch:4 / 30, train loss:0.0138 train_acc:0.9479, valid loss:0.1612 valid acc:0.8750
Epoch:5 / 30, train loss:0.1477 train_acc:0.8958, valid loss:0.0126 valid acc:0.9479
Epoch:6 / 30, train loss:0.0088 train_acc:0.9479, valid loss:0.0089 valid acc:0.9479
Epoch:7 / 30, train loss:0.0057 train_acc:0.9479, valid loss:0.0070 valid acc:0.9479
Epoch:8 / 30, train loss:0.0042 train_acc:0.9479, valid loss:0.0054 valid acc:0.9479
Epoch:9 / 30, train loss:0.0032 train_acc:0.9479, valid loss:0.0045 valid acc:0.9479
Epoch:10 / 30, train loss:0.0026 train_acc:0.9479, valid loss:0.0037 valid acc:0.9479
Epoch:11 / 30, train loss:0.0020 train_acc:0.9479, valid loss:0.0031 valid acc:0.9479
Epoch:12 / 30, train loss:0.0016 train_acc:0.9479, valid loss:0.0027 valid acc:0.9479
Epoch:13 / 30, train loss:0.0013 train_acc:0.9479, valid loss:0.0024 valid acc:0.9479
Epoch:14 / 30, train loss:0.0011 train_acc:0.9479, valid loss:0.0021 valid acc:0.9479
Epoch:15 / 30, train loss:0.0009 train_acc:0.9479, valid loss:0.0018 valid acc:0.9479
Epoch:16 / 30, train loss:0.0007 train_acc:0.9479, valid loss:0.0017 valid acc:0.9479
Epoch:17 / 30, train loss:0.0006 train_acc:0.9479, valid loss:0.0014 valid acc:0.9479
Epoch:18 / 30, train loss:0.0006 train_acc:0.9479, valid loss:0.0013 valid acc:0.9479
Epoch:19 / 30, train loss:0.0004 train_acc:0.9479, valid loss:0.0010 valid acc:0.9479
Epoch:20 / 30, train loss:0.0004 train_acc:0.9479, valid loss:0.0010 valid acc:0.9479
Epoch:21 / 30, train loss:0.0003 train_acc:0.9479, valid loss:0.0008 valid acc:0.9479
Epoch:22 / 30, train loss:0.0003 train_acc:0.9479, valid loss:0.0008 valid acc:0.9479
Epoch:23 / 30, train loss:0.0002 train_acc:0.9479, valid loss:0.0007 valid acc:0.9479
Epoch:24 / 30, train loss:0.0002 train_acc:0.9479, valid loss:0.0006 valid acc:0.9479
Epoch:25 / 30, train loss:0.0002 train_acc:0.9479, valid loss:0.0006 valid acc:0.9479
Epoch:26 / 30, train loss:0.0001 train_acc:0.9479, valid loss:0.0005 valid acc:0.9479
Epoch:27 / 30, train loss:0.0001 train_acc:0.9479, valid loss:0.0005 valid acc:0.9479
Epoch:28 / 30, train loss:0.0001 train_acc:0.9479, valid loss:0.0004 valid acc:0.9479
Epoch:29 / 30, train loss:0.0001 train_acc:0.9479, valid loss:0.0004 valid acc:0.9479
Epoch:30 / 30, train loss:0.0001 train_acc:0.9479, valid loss:0.0004 valid acc:0.9479
--- 5230.65821647644 seconds ---
test acc:1.000
```

Figure 27 - Training log for Modified VGG-16 model with perfect accuracy achieved after three epochs

Results Analysis & Comparison

The observed results from this project were fairly expected, yet extremely powerful in the message that they deliver. Deep Learning techniques can offer a means to classify trivial manufacturing outputs with superior confidence. Although the more basic models utilized were capable of achieving results that would be defined as superior to a normal human inspection during a more complicated inspection, they were not acceptable for the task at hand. In the case of the baseline CNN, the training process was relatively stagnant for the first 20% of the run. This then turned to a sporadic improvement in terms of validation accuracy; however, there were instances in which the accuracy plummeted due to overfitting to the training data. With an accuracy of 90%, this served as a suitable means to build upon to attempt to reduce the variance of the accuracy and improve the overall training times.

The second set of models involved the data augmentation trials. As previously alluded to, several options were explored before settling on the Grayscale and Color Jitter transforms. Both of these models attained robust results comparable to that of a 300% inspection with test accuracies of 99.5%. The main difference between these models and the baseline involved the more steady improvement process. As opposed to more drastic peaks and valleys, these curves were more linear in their improvement before reaching peak performance at the end of training. Ultimately, the grayscale transformation yielded accuracy and loss curves that were much smoother in comparison to the baseline, which meant more reliable testing outputs could be anticipated.

After assessing the models that could be constructed using techniques derived from the Deep Learning & Business Applications course, the VGG-16 pre-trained model was implemented to demonstrate how a robust pre-trained model could further improve results. This notion was confirmed when the VGG-16 model was provided sufficient training time. Although the lower learning rate of 0.000001 was able to yield a controlled approach towards perfect accuracy, it was determined that the loss values were still decreasing when the model was stopped. As such, the learning rate was increased, which demonstrated that this modeling approach provided a means to perfectly classify the target test data with an accuracy of 100%. Nevertheless, despite taking far less time that the previously implemented CNNs, the learning process was made more adaptive for the provided dataset by unfreezing the layer weights applied. This anticipated effect was realized during implementation as the peak accuracy was achieved after three epochs compared to the 18 required for the traditional VGG-16 pre-trained model. Not only was the model able to achieve a 100% accuracy much more quickly that the frozen form, but it also ran at much higher speeds (greater than 5% faster). As more data is added to these training situations, it will become increasingly important to ensure that they can be run in a reasonable amount of time. Otherwise, they will serve no purpose in the fast-paced manufacturing environment.

With this aspect in mind, it is useful to also compare the run times for each of the employed techniques. Both of these VGG-16 pre-trained models ran approximately one hour faster than the CNNs explored earlier (1.5 hours compared to 2.5 hours), and they were able to attain more accurate results in doing so. Nevertheless, the augmented approach was nearly perfect in terms of its accuracy, which should not go unacknowledged. Using these transforms were sufficient in improving the 90% accuracy up to 99.5%.

Conclusion

Business Impact

Pivoting back to the introduction where the problem at-hand was described, this project sought to provide a robust model that could be capable of supplementing human inspections to provide assurance that only conforming product is released as sellable material. In this particular application, the non-conformance being investigated was whether or not all of the assembly components were included within the kit. This appears to be trivial by design; however, it is a reasonable defect that can occur during the packaging portion of the production line. The current controls to prevent this type of error from occurring stems from the presence of multiple inspection steps to verify the package contains all of its required parts. With the sheer volume of product being inspected, the amount of time this inspection adds can quickly build up. The modeling techniques leveraged within this project provided a means to relieve some of this labor burden by providing a confident, robust inspection process.

Using images recorded with the desired defect to classify, these models demonstrate that an image capturing system can foreseeably be used to relay sample images as they are moving along the production line. These images could then be fed into a similarly crafted classification algorithm to ensure that no defective product can pass through final inspection. Based upon the perfect results attained with the pre-trained model and nearly perfect results of the augmented CNN, it is reasonable to conclude that this type of approach holds great utility in inspection situations during manufacturing. As more support and interest in this space develops, the defects can become more complicated, and the systems can become more adaptable to a

range of different classifications. Especially due to the fact that research has acknowledged the ability for visual inspections to lead to misclassifications does exist, a means to implement automated inspections becomes of greater interest.

Extensions for Future Work

The most exciting aspect of this project stems from the manner in which it can be leveraged and extended upon to become even more relevant for production implementation. There are inherent challenges to operating within the medical device space; however, as more robust models are developed, their utility becomes far more convincing.

One of the main goals for this type of system will be to conduct the inspections that are more challenging for routine production operators to confirm or question their results. In these situations, this type of inspection application would require a wealth of training data; however, with upper-management support, this is not an insurmountable task. In addition, higher resolution, focused images would even further the application possibilities. The successes demonstrated within this project, it should serve as a stepping stone to investigate these more nuanced non-conformances to improve overall product quality. Whether it is through incorporation of readily available pre-trained models, which performed relatively quickly for the task at hand, or transformed models specific to an intended inspection, this project has demonstrated that product inspections can be feasibly automated. In a similar sense, the model can also become more robust through the application of a multi-label model, or a non-binary classification. In relation to the models discussed within this report, this would have meant that the different types of non-conformances were separated into unique classes. Instead of outputting a “1” for non-conforming, “1” would have indicated a particular defect (“Defect 1”, for example), and “2” would have indicated a distinctly different defect (“Defect 2”). This would cause for a much more challenging model, which could have led to a need for more training data. Once again, as acceptance for this type of inspection option expands, this would become increasing more feasible.

In addition to the inspection difficulty (both defect type and multi-label capabilities), a major area to be integrated into the model involves recognizing a model’s deficiencies. This relates to the ability of the model to output images of the samples that were misclassified such that the model parameters can be tweaked, more training data can be collected, or the existing data can be better transformed. Ultimately, this type of a feature would greatly improve the effectiveness of model implementation, and it could also provide visibility as to the “type” of error (alpha or beta). This is situation is comparable to that of fraud detection; the users would be far more concerned with the number of alpha errors as this would mean that non-conforming product is mistaken as conforming. Adding this level of visibility would enable thresholds to be employed to ensure overall accuracy can be sacrificed to preserve recall.

A final extension to this modeling project involves the potential applications of the techniques employed. Within the business problem and motivation, the scope of this function was limited to a production inspection environment. An even more promising means to employ these technologies involves the incoming inspection of raw materials that often involve tolerances at the thousandth of an inch scale, which can become difficult for even a human to classify. Recent technological advances have allowed dimensions to be measured using laser rendering of products; however, these do not account for material defects such as particulate or surface imperfections. Using robust training data and modeling comparable to that of this project, a real

business opportunity exists if the entirety of component inspections could be orchestrated through an automated system.

Attachments

Attachment 1 - Google Colab Notebook

Attachment 2 - Sample Data

References

[1] Philip, R. (2021). How manufacturing industries can detect defects faster with artificial intelligence and deep learning. *ThinkPalm*. Retrieved from https://thinkpalm.com/blogs/how-manufacturing-industries-can-detect-defects-faster-with-artificial-intelligence-and-deep-learning/

[2] Torbeck, L. D. (2010). Statistical solutions: Visual inspections goes viral. *Pharmaceutical Technology, 34*(9). Retrieved from https://www.pharmtech.com/view/fda-limits-use-of-janssen-covid-19-vaccine

[3] Yardley, E. (2020). Deep learning (AI) - enhancing automated inspection of medical devices? *Med-Tech Innovation*. Retrieved from https://www.med-technews.com/medtech-insights/deep-learning-ai-enhancing-automated-inspection-of-medical-d/

[4] Krasnokutsky, E. (2021). AI visual inspection for defect detection. *MobiDev.* Retrieved from https://mobidev.biz/blog/ai-visual-inspection-deep-learning-computer-vision-defect-detection

[5] Yang, J., Li, S., Wang, Z., Dong, H., Wang, J., & Tang, S. (2020). Using deep learning to detect defects in manufacturing: A comprehensive survey and current challenges. *Materials*, *13*(24), 5755. https://doi.org/10.3390/ma13245755

[6] Deka, P., & Mittal, R. (2018). Quality inspection in manufacturing using deep learning based computer vision. *Towards Data Science*. Retrieved from https://towardsdatascience.com/quality-inspection-in-manufacturing-using-deep-learning-based-computer-vision-daa3f8f74f45

[7] Saha, S. (2018). A comprehensive guide to convolutional neural networks - the EL15 way. *Towards Data Science*. Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[8] Kamencay, P., Benco, M., Mizdos, T., & Radil, R. (2017). A new method for face recognition using convolutional neural network. *Advances in Electrical and Electronic Engineering, 15*(4). https://doi.org/10.15598/aeee.v15i4.2389

[9] Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *ICLR 2015.* Retrieved from https://arxiv.org/pdf/1409.1556.pdf