Three Experiments on CNNs for Object Detection in Chest X-rays

Terry Griffin
COMP.5300 Deep Learning
Spring 2020
terence_griffin@student.uml.

1 Introduction

This paper presents three experiments related to object segmentation of manifestations of Tuberculosis in chest X-rays (CXRs) using Convolutional Neural Networks (CNNs). This work is a continuation of research into improving the speed and quality of TB diagnosis through the use of deep learning and mobile computing technologies [1]. The results presented here are part of an effort to improve the performance of deep learning networks for object segmentation.

The first experiment deals with transfer learning. The use of models using pre-trained weights from the ImageNet [2] dataset is common in the deep learning literature. Here we ask the question of whether we can modify a model and pre-trained weights from a three channel dataset (i.e. color images) for a task involving single channel images, such as CXRs. We show that while it is possible to modify a model and the pre-trained weights to accommodate input images of one channel, the benefits of doing so are negligible.

The second experiment compares the performance of a Region-based CNN (R-CNN) [3] based on Faster R-CNN [4] against Mask R-CNN [5]. An R-CNN model extracts bounding boxes for object locations, whereas a Mask R-CNN model extracts polygonal masks. As R-CNN has the intuitively easier task we should expect some performance gain, and the collected data supports this.

We also compare Cascade R-CNN [6] versions of both architectures to determine if the additional processing of the Cascade architecture provides any performance gains. For our task, using our dataset, no significant difference in performance was observed.

The third experiment involves using the location of a manifestation as a means to lower the false positive rate. We show that for manifestations that only occur in certain locations we can effectively use this information to filter out some false positives as a post-processing step at test time.

All of the experiments described here were implemented in Python using the PyTorch[1] deep learning framework and the Detectron2 [8] library for object detection and segmentation.

2 Grayscale Transfer Learning

Transfer learning is used to provide the initial weights for a model for a given task. Rather than starting with random values, the model weights are initialized from the weights learned when training the model on a related task [7]. For many image processing tasks, CNN-based models are initialized using weights from training on the ImageNet dataset [2]. This dataset was created for the image classification task, and contains 1.2 million color images annotated with 1,000 classes.

Modern deep learning frameworks such as PyTorch and Tensorflow[2] provide collections of predefined models, commonly called a "model zoo", which include implementations of standard, well-known network architectures such as AlexNet [9], VGG [10], ResNet [11], GoogLeNet [12], and DenseNet [13]. These frameworks also provide pre-trained weights for these models created using the ImageNet training set.

Many projects involved in image classification or object detection using chest X-rays (CXRs) use models pre-trained on the ImageNet dataset. Because the ImageNet dataset is composed of three channel color images, the input to these models is also a three channel image. However, CXRs are inherently single channel data. To use a single channel grayscale image with a model designed for a color image, the grayscale image is first expanded to three channels, typically by copying the data from one channel to three. The question this section seeks to answer is if we can modify the model and the pre-trained weights to accept a single channel image, and if there is a significant advantage in doing so.

2.1 Approach

The first layer of a CNN takes the image as input and creates a set of feature maps. For a color image the kernels learned by this layer with have a depth of 3, corresponding to the number of image channels. When applying pre-trained weights to a model it must have the same shape as the pre-trained model. So the model's first convolutional layer must have an input depth of three when the pre-trained weights are set.

The greyscale image is expanded to three channels by simply repeating the pixel values for each channel. The output of each convolution is a weighted sum of the image area and the

1 https://pytorch.org

2 https://www.tensorflow.org

kernel. Because the pixel values are identical at each depth channel, we can replace the three dimensional kernel with a two dimensional kernel and achieve the same result.

From [7], the result of convolving a three dimensional kernel K_3 of size $MxNxD$ at a pixel in the three channel image I_3 is:

$$S(i,j)=\sum_M \sum_N \sum_D I_3(i+m, j+n, d) K_3(m,n,d)$$

The three dimensional image was constructed by copying the single channel image I_1 so we have

$$\sum_D I_3(i,j,d)=3\, I_1(i,j)$$

Since I_1 does not have a depth dimension, we can replace the three dimensional kernel with a two dimensional kernel. The constant multiplier of 3 here accounts for the duplication of the image from one to three channels, as above.

$$K_2(m,n)=3\sum_D K_3(m,n,d)$$

The output of the convolution can now be rewritten as:

$$S(i,j)=\sum_M \sum_N I_1(i+m, j+n) K_2(m,n)$$

The two dimensional kernel is constructed by summing the depth dimension of the three dimensional kernel and multiplying by 3. Applying the convolution of this two dimensional kernel over the single channel image gives the same result as applying the convolution of the three dimensional kernel with the three dimensional image.

2.2 Implementation

Using models from the PyTorch framework, it is fairly straight forward to make this change. After instantiating the a model and applying the pre-trained weights, the first convolution layer will have a depth of 3. This layer is updated to have a depth of 1 through a simple parameter change. The weights for the layer are updated by summing across the depth dimension and multiplying by 3.

Listing 1 shows the PyTorch code required to convert a model from a three channel input to an equivalent model with a single channel input, for ResNet and DenseNet based models from the PyTorch model zoo.

Listing 1

```
def resnet_grayscale_model(model):
    conv = model.conv1
    if conv.in_channels > 1:
        with torch.no_grad():
            conv.in_channels = 1
            conv.weight =
nn.Parameter(conv.weight.sum(dim=1, keepdim=True) *
3)

def densenet_grayscale_model(model):
    conv = model.features.conv0
    if conv.in_channels > 1:
        with torch.no_grad():
            conv.in_channels = 1
            conv.weight =
nn.Parameter(conv.weight.sum(dim=1, keepdim=True)*
3)
```

The code is very simple but has a fairly serious drawback. The conversion is dependent on the implementation details of the particular network and would need to change if the underlying implementation where to change. Notice that because the implementation of each model uses a different naming convention, separate methods are needed for each architecture. Routines of a similar size can be written for other network architectures, each slightly different depending on the naming convention used for the first layer.

2.3 Analysis

The single channel model constructed by first instantiating the three channel model with the ImageNet pre-trained weights and then applying the resnet_grayscale_model or densenet_grayscale_model routine has equivalent performance to the original models, as would be expected since they are mathematically equivalent. This was verified empirically using models built for image classification using the Fashion MNIST dataset.

The advantage of the single channel model is that there are fewer parameters. Since the first convolutional layer now has a depth of one instead of a depth of three, the number of parameters is reduced by a factor of three. However, it turns out that this reduction is insignificant compared to the total number of parameters in a reasonably deep network. Table 1 compares the number of parameters for three channel and one channel networks, and the difference as a percent of the number of parameters of the larger network, for different sizes of ResNet and DenseNet models. Even in the best case the reduction in the number of parameters is less than one-tenth of one percent. The results for other network architectures (e.g. Inception) should be similar.

Given that that gain from this process is very small, and the cost is an implementation dependent modification of the model, we can conclude that transforming the model does not have a significant benefit over transforming the input by expanding the image from one to three channels.

Table 1

Parameter Counts for Three and One Channel Input

Model	Number of Parameters for Three Channel Input	Number of Parameters for One Channel Input	Difference
ResNet18	11,689,512	11,683,240	0.054%
ResNet34	21,797,672	21,791,400	0.029%
ResNet50	25,557,032	25,550,760	0.025%
ResNet101	44,549,160	44,542,888	0.014%
ResNet152	60,192,808	60,186,536	0.010%
DenseNet121	7,978,856	7,972,584	0.079%
DenseNet161	28,681,000	28,671,592	0.033%
DenseNet169	14,149,480	1,4143,208	0.044%
DenseNet201	20,013,928	20,007,656	0.031%

In the accompanying code, the routines to covert a three channel model to a one channel model are in to_grayscale_model.py. Programs to calculate the number of model parameters are in resnet_grayscale_stats.py and densenet_grayscale_stats.py. Test programs for the modified models are in resnet_reduce_fashion_mnist.py and densenet_fashion_mnist.py.

3 Comparison Of Mask R-CNN, R-CNN, Cascade Mask R-CNN, and Cascade R-CNN Performance

As part of previous research a Mask R-CNN model was developed for detecting manifestations of Tuberculosis (TB) in CXRs. An open question is whether a different network architecture might be better suited to the task and dataset. In this section we compare the performance of the Mask R-CNN model with models based on Faster R-CNN, Cascade Mask R-CNN and Cascade R-CNN.

3.1 Dataset

The dataset consists of 1,036 chest X-ray images annotated by an expert pulmonologist. The annotations contain the locations of regions where manifestations consistent with TB have been identified. Of the manifestations labeled, four have a sufficient number of occurrences to attempt detection: Airspace Consolidation, Cavitation, Lymphadenopathy, and Pleural Effusion. The distribution of each manifestation across in the dataset is shown in Table 2.

3.2 Network Architectures

An R-CNN network predicts bounding boxes and classes for objects in an image. In our case the objects being detected are the manifestations identified by the pulmonologist. The implementation is based on Faster R-CNN, using the implementation from the PyTorch model zoo and the Detectron2 library.

Table 2

Distribution of Manifestations

Manifestation	Images	Instances
Airspace Consolidation	917	1274
Cavitation	305	399
Lymphadenopathy	306	342
Pleural Effusion	119	122

Faster R-CNN is a two stage network containing a region proposal network and a classification network. The region proposal network learns to identify potential objects in an image, and the classification network learns the bounding boxes and classification of the objects. During training the loss computed by the network is a combination of the loss for the bounding box location and the class.

Mask R-CNN expands on R-CNN by adding an additional branch to identify a mask (polygonal shape) for the object. This provides more detailed location information and therfore should be of more use. This additional branch adds a loss term for the location of the mask, which is summed with the losses defined by the base R-CNN network during training.

The PyTorch implementation of Mask R-CNN contains outputs for both the bounding box branch and mask branch of the network. This allows us to compare the performance of the simpler bounding box task to the mask task for the same network. However, because the loss values from the mask branch are used during training, the performance of the network on the bounding box task may be different than the performance of a similar R-CNN network. An R-CNN network may outperform a Mask R-CNN network on the bounding box task because it is focused solely on that task, whereas the Mask R-CNN network is influenced by both tasks.

One problem identified with R-CNN based models (including Mask R-CNN) is that the model is trained using a single intersection over union (IoU) threshold value to discriminate positive and negative samples. A small threshold leads to noisy detections and a high threshold can degrade performance [12].

The Cascade R-CNN architecture seeks to reduce this issue by using a sequence of detectors trained with increasing IoU thresholds. In the PyTorch implementation used for this work these detection heads are trained using IoU thresholds of 0.5, 0.6 and 0.7. The Cascade architecture can be adapted to both R-CNN and Mask R-CNN networks.

3.3 Approach

To compare the performance of the four network architectures, models where created for each of the four manifestations in our dataset. The models were constructed using ResNet-101 and ResNet-50 backbone networks.

The performance of each model is measured using the COCO performance metrics for mean average precision (mAP) and average precision at an IoU threshold of 0.5 (AP50). The

COCO mAP is an average of the precision over 10 IoU threshold values equally spaced between 0.05 and 0.95.

The hyperparameters for each network architecture were tuned individually. The four networks for each architecture (one per manifestation) share the same set of hyperparameters.

The dataset for each manifestation was split into three groups: 80% for training, 10% for validation to tune hyperparameters, and 10% for testing.

3.4 Results

The performance measures for each network on the four manifestations are shown in Tables 3-6. The performance between the four manifestations varies widely. This is due to the discrepancies in the number of samples of each and the differences in the average size. Airspace consolidation has both the largest number of instances and large instances. Pleural Effusion has the fewest number of instances and each instance tends to be relatively small.

Within each manifestation the comparison between different network architectures is similar. R-CNN has a slightly better performance than Mask R-CNN. This makes some sense due to the slightly simpler task. The Cascade variety of networks did not outperform the base networks in these tests. One possible reason that we don't see a similar performance gain to that presented in [6] is the limited number of samples in our dataset. The advantage of multiple detectors using different IoU thresholds may not be apparent without a larger or more varied dataset.

One additional item to note from this data is that the smaller ResNet-50 backbone outperformed the larger ResNet-101 backbone for the Pleural Effusion manifestation. This result is likely due to the small number of instances of this manifestation.

Table 3

Performance on Airspace Consolidation

Network	Backbone	mAP	AP50
R-CNN	ResNet-50	0.3625	0.7184
Mask R-CNN	ResNet-50	0.3313	0.6705
Cascade R-CNN	ResNet-50	0.3718	0.7094
Cascade Mask R-CNN	ResNet-50	0.3299	0.6785
R-CNN	ResNet-101	0.4002	0.7990
Mask R-CNN	ResNet-101	0.3583	0.7871
Cascade R-CNN	ResNet-101	0.3992	0.7882
Cascade Mask R-CNN	ResNet-101	0.3602	0.7891

Table 4

Performance on Cavitation

Network	Backbone	mAP	AP50
R-CNN	ResNet-50	0.1120	0.265
Mask R-CNN	ResNet-50	0.0997	0.237
Cascade R-CNN	ResNet-50	0.1203	0.261
Cascade Mask R-CNN	ResNet-50	0.1001	0.241
R-CNN	ResNet-101	0.1147	0.2805
Mask R-CNN	ResNet-101	0.0988	0.2896
Cascade R-CNN	ResNet-101	0.1199	0.2902
Cascade Mask R-CNN	ResNet-101	0.1021	0.2843

Table 5

Performance on Lymphadenopathy

Network	Backbone	mAP	AP50
R-CNN	ResNet-50	0.992	0.2932
Mask R-CNN	ResNet-50	0.0961	0.2688
Cascade R-CNN	ResNet-50	0.987	0.2902
Cascade Mask R-CNN	ResNet-50	0.0952	0.2672
R-CNN	ResNet-101	0.0920	0.3385
Mask R-CNN	ResNet-101	0.0786	0.3084
Cascade R-CNN	ResNet-101	0.0941	0.3407
Cascade Mask R-CNN	ResNet-101	0.795	0.3102

3.5 Future Work

Mask R-CNN is a reasonable starting point for the task of identifying the location of abnormalities in CXRs. Additional work needs to be done to determine if this is the best choice for our task and our dataset. In particular, experimentation with some newer object detection and segmentation strategies or backbone architectures may reveal a better choice.

Table 6

Performance on Pleural Effusion

Network	Backbone	mAP	AP50
R-CNN	ResNet-50	0.1027	0.3584
Mask R-CNN	ResNet-50	0.0801	0.3036
Cascade R-CNN	ResNet-50	0.1004	0.3492
Cascade Mask R-CNN	ResNet-50	0.0792	0.2998
R-CNN	ResNet-101	0.0623	0.2725
Mask R-CNN	ResNet-101	0.0415	0.2487
Cascade R-CNN	ResNet-101	0.0615	0.2720
Cascade Mask R-CNN	ResNet-101	0.0409	0.2412

4 Reducing False Positives Using Location Data

One property of CNNs is that they are translation invariant, meaning they are able to find an object regardless of the location in the image. While this is a useful property in general, for the task of abnormality detection in chest X-ray, it can lead to unnecessary false positives.

Some manifestations occur in only certain locations with the lungs. For example, Pleural Effusion only appears at the bottom of the lungs, and Lymphadenopathy only appears near the lymph nodes, to the left and right of the spine. In this section we explore using this information to filter out some false positives based on the location of the detected mask.

4.1 Approach

To encode the likeliness of a manifestation occurring at a particular location we created heatmaps based on the training and validation datasets for each manifestation.

A 1,024 x 1,204 pixel heatmap was constructed from the annotation data by counting the number of instances whose mask includes each pixel. The values where then normalized to a range of 0-1. This process is implemented in the file create_cat_heatmap.py in the accompanying code.

The resulting heatmaps are shown in Figures 1-4. The images are reasonable given the nature of each manifestation. Airspace consolidation and cavitation can appear throughout the lungs. The heatmap for Airspace Consolidation looks smoother, due to the large number of instances included in the dataset. The heatmap for cavitation shows a hot spot on the left lung and does not fill the lung completely. It is unclear if this represents true properties of the manifestation or are due to the limited size of the dataset.

Figure 1: Heatmap for Airspace Consoliation

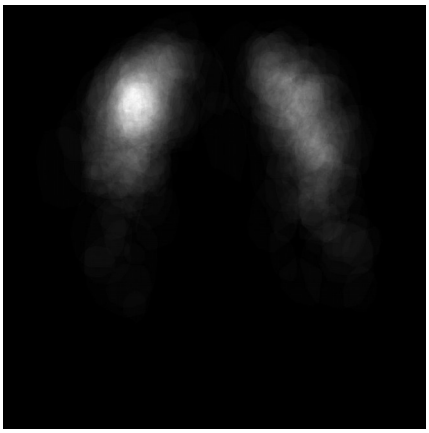

Figure 2: Heatmap for Cavitation

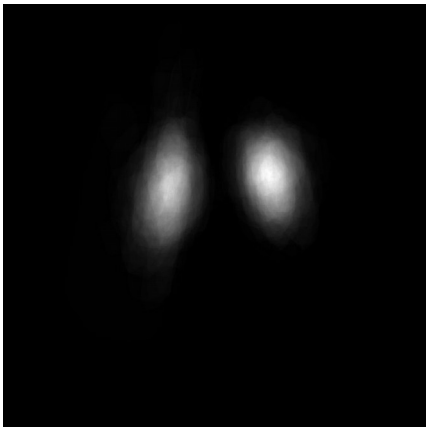

Figure 3: Heatmap for Lymphadenopathy

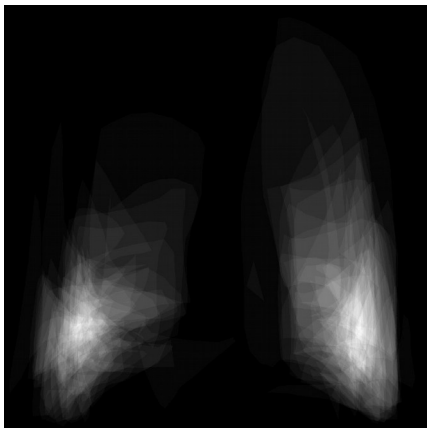

Figure 4: Heatmap for Pleural Effusion

The heatmaps for Lymphadenopathy and Pleural Effusion correctly show the particular locations where those manifestations are found. The very limited number of samples for Pleural Effusion leads to the roughness of the heatmap, where distinct polygons can be seen.

The per manifestation heatmaps are used during the evaluation process for each network, as a post-processing step after running the model on the input. For each instance detected by the model, the bounding box is compared with the heatmap for the identified manifestation. If the area of the heatmap covered by the bounding box does not include any value above a certain minimum threshold then the instance is deemed a false positive and removed from the set of detected objects. The threshold value of 0.2 was arrived at empirically through testing against the validation sets.

This evaluation is performed against the test set after training and on the validation set during training. The evaluation on the validation set is used to tune the hyperparameters of each model.

4.2 Results

The results of this experiment are encouraging. The inclusion of the heatmap did not have any affect on the performance of either the Airspace Consolidation or Cavitation manifestations, and did not result in any additional false negatives for the Lymphanopathy and Pleural Effusion manifestations. Thus, there were no negative affects to adding this processing.

Of the 40 false positives identified by the model for Lymphadenopathy, 8 were filtered out using the heatmap. For Pleural Effusion 3 out of 8 false positives were filtered out. This accounts for a significant number of false positives identified for these two manifestations.

4.3 Future Work

The current implementation uses the location heatmap as a post-processing step. Including this step as part of the training process by incorporating it either as part of the mask loss or as a regularization step may lead to better performance.

The PyTorch framework is very flexible and modification of the existing mask loss or optimization routines should be possible, although it is not immediately obvious how to do this with the current code base.

REFERENCES

[1] Y. Cao, C. Liu, B. Liu, M. J. Brunette, N. Zhang, T. Sun, P. Zhang, J. Peinado, E. S. Garavito, L. L. Garcia, and W. H. Curioso, “Improving tuberculosis diagnostics using deep learning and mobile health technologies among resource-poor and marginalized communities,” in *2016 IEEE First International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, June 2016, pp. 274–281

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: large-scale hierarchical image database,” in *IEEE Computer Vision and Pattern Recognition*, 2009, pp. 248,255.

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[4] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” *TPAMI*, 2017.

[5] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, “Mask R-CNN,” CoRR, vol. abs/1703.06870, 2017. [Online]. Available: http://arxiv.org/abs/1703.06870

[6] Z. Cai and N. Vasconcelos: Cascade r-cnn: Delving into high quality object detection. [Online]. Available: http://arxiv.org/abs/1712.00726 (2017)

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[8] Y. Wu, A. Kirillov, M Francisco, W. Lo, and R. Girshick: Detectron2, 2019, [Online] Available: https://github.com/facebookresearch/detectron2

[9] A. Kirihevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional networks. In *Advances in neural information processing systems*, 2012.

[10] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[12] C. Szegedy, et. al., Going deeper with convolutions, 2018 [Online] Available: https://arxiv.org/abs/1409.4842

[13] . Huang, Z. Liu, K. Q. Weinberger, and L. Maaten. Densely connected convolutional networks. In *CVPR*, 2017