COMS 4771 - Final Project Report

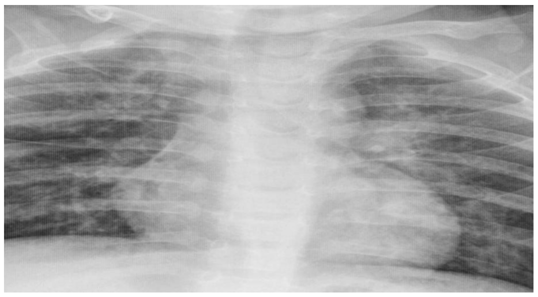
Ahmed Sheta - as5452

May 8th, 2020

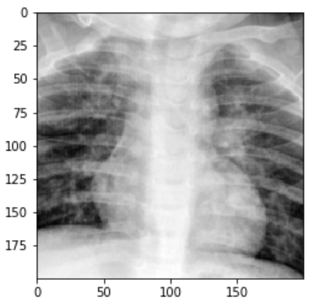
1 Reading Data

The first step of the project was figuring out how to embed the images as data points that could be used for classification and training.

The first problem was that the images had different sizes/resolutions, and so they needed to be resized uniformly. The uniform size of 200 x 200 pixels for standardization was chosen based on visual assessment, with the goals of preserving the natural aspect ratio of the X-Rays, and minimizing the size as much as possible in order to reduce the dimensionality of the problem while preserving the resolution needed to make proper classification. First, by trying different aspect ratios, we observed that square proportions were ideal, not only for preserving aspect ratio of good-resolution images, but also fixing initially distorted images. For instance, in Fig 1, the original image is shown to the left, and the resized image is shown to the left. We observe not only the more natural aspect ratio of the resized image, but the crisper, stronger resolution. Very roughly, rescaling to more natural proportions accentuates defining features of the X-Ray - as we can see in the right image of Fig 1, the internals of the lung image become more defined and visually more apparent.



(a) Original image, of size 498 x 918



(b) Resized image, of size 200 x 200

Figure 1: Effects of resizing

By adjusting around and fine tuning, 200 x 200 was found to be roughly optimal. Lower qualities started to reduce the resolution of important parts of the image such as the details of the lung, while higher resolutions unnecessarily blow up the dimensionality of the problem without significant enhancement of resolution.

Each image is then read as an array of its pixel values on a grayscale. This is a very rudimentary way of reading images which still yielded acceptable accuracies (~67%) - improvements will be discussed later below. We note that even at 200 x 200 pixels, the initial dimensionality of the problem is very high, at 40,000 features per data point.

2 Initial Attempts

A quick run through some of the initial attempts, and using them to explore the structure of the data.

2.1 Perceptron

I started by coding a very simple linear perceptron, based on the algorithm V0 from HW2. Training it was very fast, and its accuracies typically never went below 50% after a reasonable training phase. For a very crude model based on a crude way of reading data, that wasn't so terrible. This simple model, however, provided helpful insight about the structure of the data, that was helpful for improving other models.

The problem was its high dependence on number of iterations used to train the model. For instance, for a 70-30 split of the training set, the accuracies on the 30% validation set can be seen in Fig 2 below.

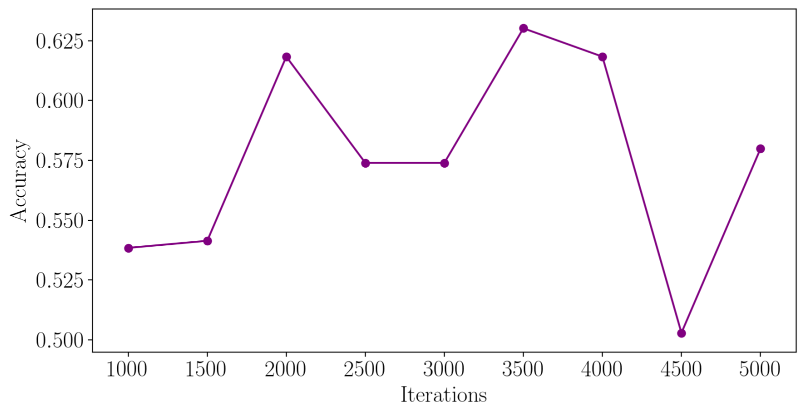
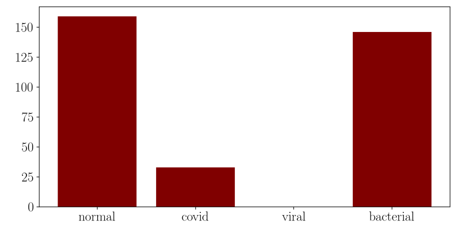


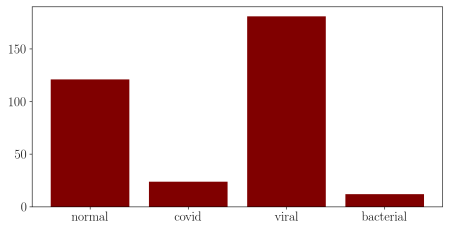
Figure 2: Accuracy of Linear Perceptron as the number of training iterations is varied

As seen above, the perceptron apparently doesn't converge. For very simple chages in the number of iterations used for training, the accuracy could drop as high as 10%. This is reasonable within the context of this problem. The perceptron algorithm is guaranteed to converge only if the data is linearly separable. And since this 40,000-dimensional embedding of the data is completely arbitrary and not in any way tuned to the problem at hand, we expect clusters of the same label to be highly non-linear within the space. The perceptron algorithm is then almost guaranteed to make more mistakes whenever we allow for more iterations, and so any additional iterations result in a very different model, which can be seen in Fig 2.

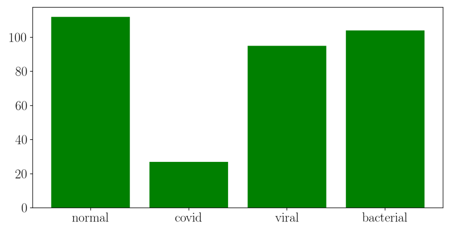
Another problem was the huge skew in labelling bacterial/viral. For specific iterations, the model almost predicted all the bacterial as viral, while for others the model almost predicted all the viral as bacterial. One such example is seen in Fig 3. The true distribution of labels was around 80 viral and 90 bacterial. A 3500-iteration trained perceptron classified ∼140 data points as bacterial and 0 points as viral, while the 5000-iteration trained perceptron classified ∼170 data poitns as viral and almost none as bacterial.



(a) Classified Labels after 3500 iterations



(b) Classified Labels after 5000 iterations



(c) True Labels

Figure 3: Label Distribution

Since the number of data points classified as normal or covid remains relatively stable, this skew suggests a strong resistance to linear separability especially between bacterial and viral clusters in the current embedding. Then, as the perceptron keeps iterating more, it accumulates a very large weighting vector for either the bacterial or viral cluster depending on the specific training set used.

The linear perceptron is then a very unstable model that would be unpractical to use, althought its linearity and very fast runtime allows some nice exploration of the data. Its unstability are especially problematic in a clinical setting during a pandemic, where thousands of new X-Rays are continuously being added to the database. Instead of benefitting from an increasing training dataset, the linear perceptron performs worse in unpredictable ways as we iterate more.

An attempt was made at kernelizing the perceptron, which showed the same behavior, but with an overall improvement in the range of accuracies. A special case, however, was using a 10^{th}-degree polynomial kernel. It showed a spike in performance, reaching 68% accuracy, and was relatively stable under changing iteration number - its accuracies changing within a range $< 3\%$. This suggests that within the current embedding of the data, the most natural space where the clusters are relatively linearly separable is a 10-dim polynomial of the current 40,000-dim embedding.

2.2 SVM

To bypass the problem of fine-tuning the number of training iterations, we used Python's library implementation of SVM, which resulted in roughly the same accuracy on validation set ($\sim 62\%$ for the linear SVM), but was guarnteed to return the most stable linear boundary (and thus one that is more generalizable to generic testing sets), independent of the number of iterations/order of training set. Although the holdout accuracy of the SVM is very similar to the accuracy of the Perceptron when the iteration number is fine tuned to produce best results, it performed much stronger on the Kaggle submission. The sensitivity of the perceptron to training iterations implies that fine tuning iteration number to perform well on a specific set couldn't generalize to a generic testing set. Thus, a perceptron model that was trained for accuracy 60% on the validation set performed around 50% on the Kaggle submission. On the other hand, the SVM model that got 62% on validation set produced a 64% on the Kaggle submission, which matches the expected generalizability of the SVM.

Finally, polynomial kernelization of the SVM was tried, which produced slightly better results. The accuracies of the kernelized SVM on a holdout validation set are shown in Fig 4 below.

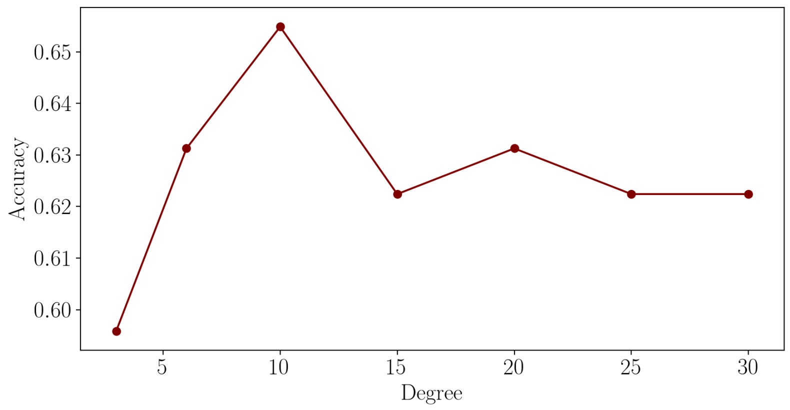


Figure 4: Accuracy of Kernelized SVM as polynomial degree is varied

Polynomial kernel seems to improve the accuracy overall, with an interesting spike at $p = 10$. This suggests that the current embedding of the data most naturally lives in a 10^{th}-degree polynomial of that space (i.e. in that space, the clusters are most linearly separable). To confirm the hypothesis that the data is most linearly seprabale by a 10^{th}-degree polynomial boundary, a kernelized version of the Perceptron was trained by the data, and the results are shown in Fig 5.

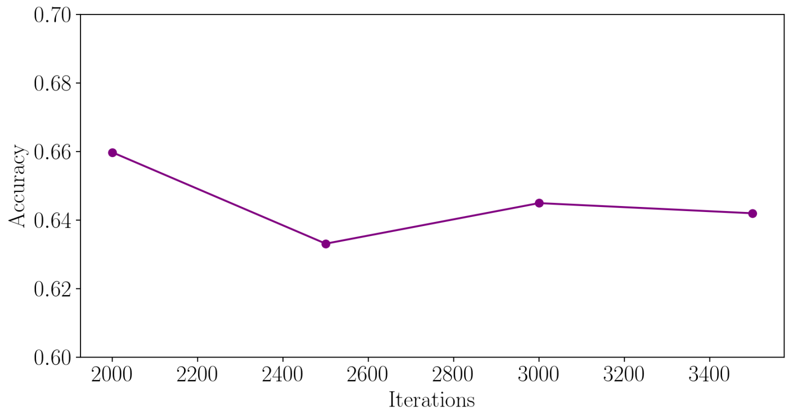


Figure 5: Accuracy of kernelized perceptron of 10^{th}-degree as number of iterations is varied.

As we can see, the kernelized perceptron becomes highly robust against number of iterations, which is what we expect if the data is approximately linearly separable.

Finally, we note that despite the improvement, the accuracies of kernelized SVM in Fig 4 are still low. We expect that any data set could be linearly separated in high enough dimensions, and so the plateu of accuracies at ∼64% is considered relatively low. However, for high degrees, there are plenty of parameters to fit, and so we need a lot more data to fit an accurate generalizable model. In this case, 700 training data points is simply not enough to train a high-degree polynomial boundary in a 40,000-dim space.

In a Kaggle submission, the 10^{th}-degree polynomial kernelized SVM performed slightly better than the linear SVM, with an accuracy of 66%.

Within the current embedding of data, elementary classifiers seem to plateu at 66%. In order to achieve higher accuracies, new methods of embedding the data are explored.

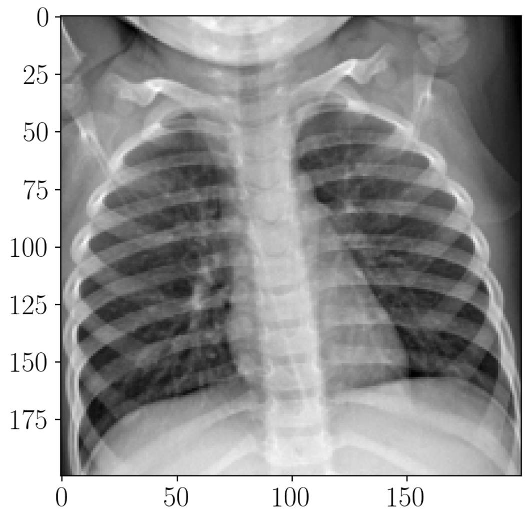
3 Alternative Embeddings

3.1 Dimensionality Reduction - PCA

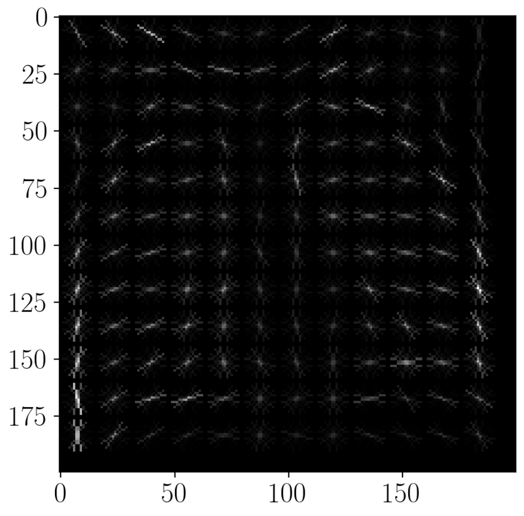
I tried using PCA on the raw data, which reduced dimensionality of the input space from 40,000 to ∼1000. The observed performance declined uniformly across validation set and Kaggle Submissions to ∼50%. This makes sense given our previous discussion about the high non-linearity of the data. Within a non-linear model, finding linear projections in directions of highest variance could actually be capturing noise/irrelevant information - the features relevant to classification could be distributed non-linearly in a way such that their observed Euclidean variance is minimal. Therefore, applying linear projections on highly non-linear data could be downgrading the discriminative quality of the input space, which is observed in the significantly reduced accuracies.

3.2 Histogram of Oriented Gradients

An alternative way of embedding images is the Histogram of Oriented Gradients (HOG) method. It divides the image into cells of uniform chosen size, and it calculates the gradients of all pixels inside a cell and embeds them into a histogram according to the range of gradients. It just concatenates the histograms of all cells as a flat vector. This methods has 2 main advantages beneficial in our situation. First, rather than absolute values of pixels on a grayscale, it relies on the gradients of the image, which extract useful information about changes happening within the image. For instance, in Fig 6, the original 200x200 image is shown to the left and its HOG representation is shown to the right.



(a) Original Image



(b) HOG representation

Figure 6: HOG representation of images

As we can see, all of the gory details of the X-Ray like the rib cage, the black background of the lung, the skeletal structure, etc. are discarded, while the useful details of the lung such as the internal variations in the lung structure are preserved in the form of vectors pointing along these variations, with magnitude proportional to intensity of change. Second, by adjusting the cell size, we adjust the "resolution" of the resulting transformed HOG representation. The goal is to find cell size on the scale of the size of variations of interest, such that the gradients represent variations we actually care about. By experimenting with the effects of accuracy, it was found that cells of size 16 x 16 pixels produce optimal results. A big cell size as this is also very robust to noise and irrelevant fluctuations in the images. A very useful side effect is a huge dimensionality reduction, such that each transformed HOG image is a data point of dimension=1152, which is 1/40 of the original dimension. In a way, HOG both takes the gradients of the images, accentuating relevant information and eliminating noise, and performs dimensionality reduction to capture the most relevant features of the image at the same time.

4 Final Classifier - Kernelized SVM using HOG

4.1 General Discussion

Finally, we ran the kernelized SVM on the HOG embedding of the data. A validation set was constructed using a 70-30 split of the data, and the kernelized SVM was trained using 70% of the training data - its accuracies on the validation set are shown in Fig 7.

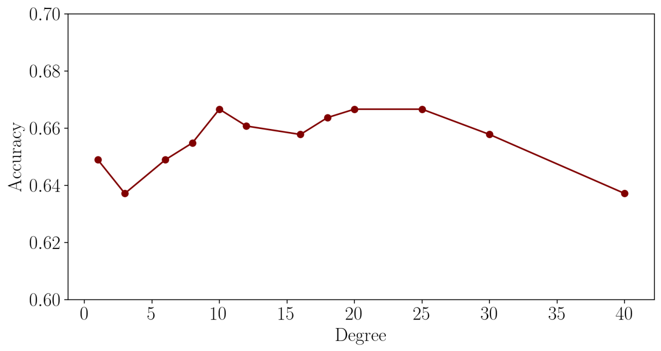


Figure 7: Accuracy of Kernelized SVM trained using 90%, as the degree of polynomial is varied.

Interestingly, the same spike at $p = 10$ is observed as before. This suggests that regardless of the embedding of the data, its boundaries are most naturally 10^{th}-degree polynomials. Moreover, as the polynomial degree is increased, we witness an initial increase in accuracy followed by a decline. The decline is due to overfitting - the higher the degree of the polynomial, the more parameters there are to fit, and thus the model becomes prone to overfitting to the training set used. As discussed previously, we simply don't have enough data to accurately train a very high-degree polynomial boundary without overfitting to the training set used. However, this suggests that in a clinical setting, as more data is accumulated in the database, the classifier could be trained better.

To confirm this, we tried training the kernelized SVM with different splits of training sets, and the results are shown in Fig 8 below.

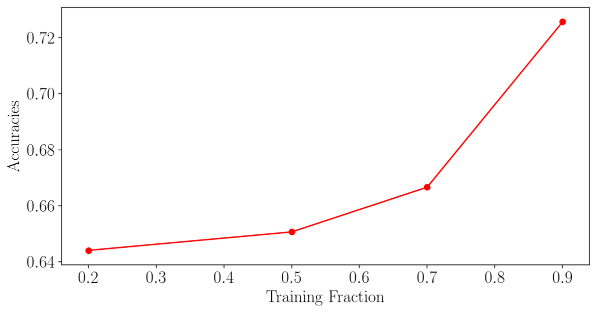


Figure 8: Accuracy of Kernelized SVM of degree 10, using different split sizes.

Even with minimal data (trained using 20% of the training set, around ∼200 images), it performs reasonably well with 64% accuracy. This model is very robust and its performance improves as it's trained with more data - which would be very helpful in a clinic that's accumulating more X-Rays every day.

The model is also more easily interpretable as well, thanks to HOG embedding. By embedding the images using HOG initially and visually inspecting the results, such as in Fig 6, we see that automatically the most prominent features are within the lung. And guided by the significant improvement in accuracy by using this technique, it's evident that these features correlate to successful classification of Covid/Pneumonia/normal. The clinician could then use these embeddings rather than the original X-Rays to diagnose the patient, which would be a more natural way to examine X-Rays for purposes of disease classification.

The Kaggle submission using this classifier had 75.8% accuracy.

4.2 Error Analysis and Applications

For error analysis, the 10^{th}-degree kernelized SVM is trained using 70% of the training set and tested on a 30% validation set. The accuracy, precision, recall, and f1 scores are shown below

Accuracy	66.6%
Precision	69.3%
Recall	64.6%
F1	66.3%

The confusion matrix is plotted below

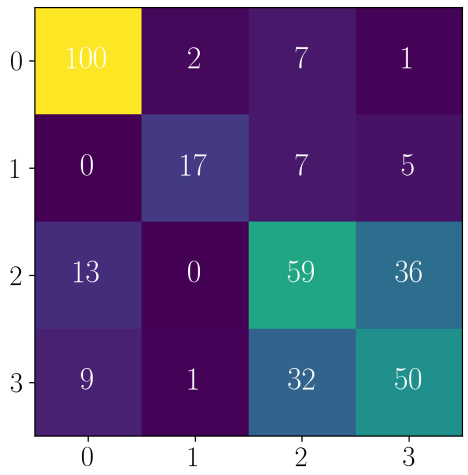


Figure 9: Confusion Matrix.

where the labels used are {0: Normal, 1:Covid, 2:Viral, 3:Bacterial}.

First, normal images are classified very well, with 100/110 of the normal images classified correctly as normal. More importantly, the Covid classification is relatively good, with 17/29 classified correctly as Covid. However, even among the Covid cases misclassified, none of them as classified as normal, which is very good - if someone has Covid, they will be immediately diagnosed with a disease that requires subsequent looking into. Moreover, among all of the other clusters, only 3 total non-Covid cases are classified as Covid, which is very useful. If a patient's X-Ray gets classified as Covid, it's extremely likely to actually have Covid.

The worst performance of the classifier is in differentiating between viral and bacterial pneumonia. This matches what we observed before with the linear perceptron, where the bacterial/viral distinction was very unstable. This might suggest that we need higher resolution to differentiate between the two - but this then risks mixing other classes such as normal and Covid. Thus, within the context of a Covid pandemic, the viral/bacterial distinction isn't a big issue - any patient diagnosed with some form of pneumonia can be further looked into at their own pace.

Despite the relatively unreliable total accuracy of 75%, this classifier then could be very useful in a clinical setting. If we lessen the expectations into 3-label classification: Normal-Covid-Pneumonia, this classifier has 86% accuracy, which is reasonably high. It could be used as a filtering pre-processing step within a clinic: if it classifies the patient as normal, they could be acquitted immediately (modulu some pneuomnia cases, but that's okay in a Covid pandemic). And if it classifies a patient as Covid, then it's extremely likely that they actually are Covid patients. For pneumonia classification, a further classifier could be constructed that differentiates bacterial from viral.

5 Conclusion

One bug that I faced while working was a reasonably high validation accuracy, but a very low Kaggle accuracy ∼20%. After a lot of investigation and asking around, the problem was that the test labels were being shuffled, thus producing a random set of labels that match 25% accuracy on a 4-label classification problem. The key insight was that the 25% accuracy must correspond to a completely random classifier, and so the issue was not in the model, however bad it is, but in ordering labels.

Finally, in implementing classifiers, I tried to stick to the simplest, most interpretable classifiers - but the highest accuracy I could get is 75.8%. Tweaking the standardization size and the HOG parameters could potentially have brought this up by 5%, but I think that's it. As we saw in HW2, the kernelized perceptron was able to achieve almost 96% accuracy on digit classification, and so kernelized linear boundary is definitely a very strong classifier if utilized on the right problem. Maybe an elementary discriminative classifier (linear/kernelized) isn't simply sophisticated enough to tackle this problem, and potentially we need more perceptron layers - i.e. a neural network.