School of Computer Science
University of Birmingham

Improving Covid-19 Prediction Through Variational Bayes for Learned Class Weighting

Student: Jack Foster

Supervisor: Dr. Hyung Jin Chang

Additional Supervisors: Dr. Kwang Moo Yi, Dr. Jongwon Choi

Programme: MSc Robotics

September 3, 2020

Abstract

Current Covid-19 datasets are rife with class imbalance. This study identifies the consequences of such imbalance on Covid-19 diagnosis tasks, as well as presenting both existing and novel solutions to these problems. Firstly, class-based upsampling and loss function weighting are used. Combined with data augmentation, these strategies achieve a much greater performance on the Covid-19 minority class. However, this comes at the cost of lower accuracy in majority classes. As such, a novel adaptation of the Variational Bayes for Active Learning thesis is used to predict dynamic, per-sample classification difficulty scores for upsampling and loss function weighting. This method improves minority class prediction while improving overall performance, alleviating issues of the existing methods. In addition, the efficacy of supervised and unsupervised pretraining as a class imbalance strategy is explored. The findings show that pretraining may act as an orthogonal strategy that improves classification performance when combined with other strategies.

Keywords: Class Imbalance, Deep Learning, Image Classification, Variational Autoencoder, Pretraining

Acknowledgements

I'd like to thank my supervisors, Dr. Hyung Jin Chang, Dr. Kwang Moo Yi and Dr. Jongwon Choi, for their continued support and guidance throughout the project, especially given the difficulties presented by differences in timezones. I'd also like to thank George Sheppard, for his technical opinions that were offered throughout the development of this project.

Contents

1 Introduction

This project addresses the issue of class imbalance within current Covid-19 image datasets. Its primary contributions are a novel adaptation of Variational Bayes for Active Learning [1], to utilise the predicted probability of misclassification as per-sample loss function and upsampling weights. In addition, modern pretraining methods are proposed as a potential method for alleviating the impact of class imbalance.

Covid-19 testing methodologies are complex and require expert knowledge; some approaches, such as RT-PCR testing, are reportedly susceptible to false positives [2]. Analysing chest X-rays or CT scans is a viable approach, but this needs the attention of trained professionals who are in high demand and short supply during a pandemic. As such, there is a cross-disciplinary opportunity to apply modern machine learning approaches to this domain, automating the diagnosis process to alleviate some of the pressures placed upon medical staff.

Work has already been conducted compiling datasets of chest X-ray images [3], or CT scans [4]. However, the novelty of the disease has resulted in a scarcity of Covid-19 positive samples, leading to significant imbalance within available datasets. Class imbalance is problematic because machine learning architectures struggle to model the underlying characteristics of the minority class, as it simply does not observe a sufficient amount of samples [5]. Furthermore, class imbalance leads to skewed results, as the minority class represents such a small percentage of the validation set that high accuracy may be published, despite an inability to model entire classes within the data [6]. Despite this, current literature on classifying the novel Coronavirus fails to appropriately address the impacts of class imbalance or the potential performance increases that could occur if modern strategies were applied to mitigate the effects of this imbalance.

As such, this study seeks to answer three research questions:

1. Does class imbalance present a substantial issue for Covid-19 classification problems?
2. Can traditional strategies alleviate the impact of class imbalance?
3. Can modern techniques outperform traditional strategies and mitigate known issues with standard approaches?

The first research question is somewhat of a prerequisite of the second and third; exploring the efficacy of class imbalance approaches is only important if the class imbalance negatively impacts classification performance. The second research question explores the approaches that are deemed standard within the field, notably upsampling and loss-function weighting. Data augmentation is also used. The final research question is the primary focus of this work, and is also where novel contributions are made. This question aims to assess whether newer methods can contribute tangible improvements to image classification performance in this domain.

This report covers the necessary background knowledge, before exploring the methodology and implementation of the classifier networks, traditional class imbalance strategies, and modern approaches. Finally, this work goes on to present the findings and discusses the implications of these results, before identifying potential future work.

2 Background

2.1 Convolutional Neural Networks

Convolutional neural networks are well suited to image processing tasks [7], as they utilise spatial information within the input vector, unlike multi-layer perceptron neural networks. Furthermore, shared feature banks enable CNN architectures to be substantially deeper, without the exponential rise in connections [8], which facilitates the learning of more complex patterns. The invention of deep residual networks has lead to substantial improvements in image processing tasks [9]. Residual connections, or skip connections, allow a layer to take as input the output of much earlier layers, not just the prior output, through identity mapping connections. These connections help with gradient flow, and have been empirically shown to improve results on image classification tasks. As such, many current works utilise ResNet18 and ResNet50 architectures, which are simply deep residual networks of 18 and 50 layers, respectively.

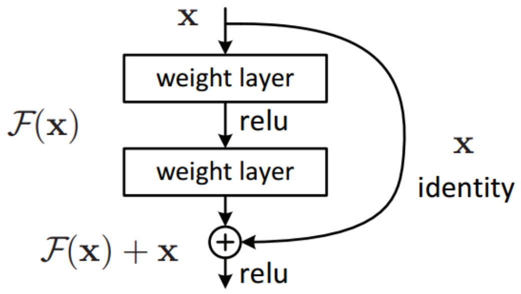

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 1: Residual Connection [9]. The input to the residual block is simply added to the output of the residual block, helping gradient flow to earlier layers of the network.

2.2 Covid-19 Prediction

Covid-19 prediction is formulated as an image classification task. There have been several works using deep residual CNNs to predict Covid-19 from medical images such as CT scans and x-rays. One such work produced COVnet, a deep convolutional neural network based on ResNet50 that takes a 3D input of a series of 512x512pixel CT scan slices of a patient's lung region [4]. The network utilises a softmax layer to output a probability for each category (COVID-19, standard pneumonia,or non-pneumonia). The interesting contribution from this paper is that the network was able to identify 2D and 3D global representative features which indicated the presence or absence of COVID-19. However, this comes at the cost of having a much larger input vector, slowing training and reducing the viability of such networks to run on lighter computer architectures. Furthermore, the input vector is very specific (i.e. multiple CT slices from the same patient for each input), and as such cannot utilise many of the larger, public datasets.

The similarly named, yet distinct, COVID-Net utilises residual connections like CovNet, however it also uses a series of ”projection-expansion-projection” layers, which are essentially a series of 1x1 convolutional layers, which project the input vector into lower, then higher dimensions repeatedly [3]. While this architecture performs well, it is bespoke and hand-crafted to best suit this specific problem and dataset, which is less-than-ideal as early stages of a pandemic are critical for containing and controlling the outbreak. As such, it is necessary to develop generic, ”plug-and-play” machine learning architectures that can be rapidly trained and operational, while also being reliable and accurate despite the bias in the available data. The COVID-net paper also introduces a public benchmarking dataset, COVIDx, to train and evaluate architectures. This dataset is heavily imbalanced.

2.3 Variational Autoencoders

Autoencoders are generative models [10]. They can be split into 2 components: an encoder and a decoder. The encoder portion compresses an input vector (x) into a lower dimensional representation, known as latent space (z). From there, the decoder portion attempts to reconstruct the original input from this latent space representation, leading the overall architecture to act as a form of lossy compression. The limitation of traditional autoencoders is that their latent space is sparse, and not a continuous distribution; in order to generate data other than simply the reconstruction, the latent space must be continuous. This is because if a sample is represented as a discrete point, then the vast majority of the search space is going to generate noise if sampled. If each input vector is represented as a distribution, then the the latent space may be sampled to achieve perturbations of the input. In other words, the vector clusters within the latent space of a standard autoencoder are sparsely distributed and thus vectors cannot be interpolated to generate an output somehow distinct from the input, as the decoder is unable to decode the vectors, likely producing noisy, useless outputs. Variational autoencoders remedy this by generating two latent variables from an input, a mean (μ) and a standard deviation (σ) for a given sample, representing a Gaussian distribution. The decoder can then sample from this distrubtion, creating variation in the output for a given input. As such, latent space representations of a given input become a continuous distribution, rather than discrete. As gradient cannot be backpropagated through a random sampling node, the latent variables are separated from the sampling through the reparameterization trick [10], allowing for the gradient to pass through the latent variables. By combining the traditional autoencoder loss function with the Kullback-Leibler divergence [11], the clusters can be trained to fall within constrained bounds, forming a continuous distribution. The new loss function is defined as:

$$loss = \sum_{i=1}^{n} ||x_i - \hat{x_i}||^2 + 0.5 \sum_{i=1}^{n} (\sigma_i^2 + \mu_i^2 - log(\sigma_i) - 1) \tag{1}$$

$$loss = BCE(x, \hat{x}) + KLD(N(\mu_x, \sigma_x), N(0,1)) \tag{2}$$

where x_i is an input vector and $\hat{x_i}$ is the reconstruction. Now that the latent space can be sampled, it is possible to generate new information from a given input. Finally, as the decoder is reconstructing x by sampling z, the reconstruction can be thought of as maximising $P(x|z)$.

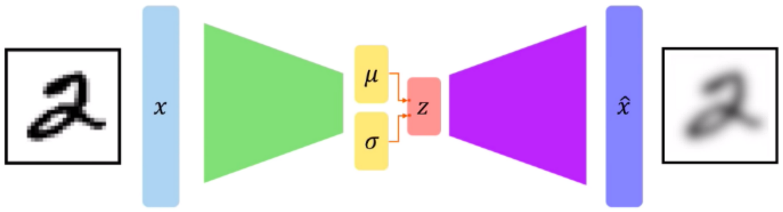


Figure 2: Variational autoencoder architecture [12]. Once trained, the network will always produce the same mean and standard deviation for a given input, however the reconstruction of this input will vary due to the random sampling.

2.4 Variational Bayes for Active Learning

Variational Bayes for Active Learning (VaB-AL) [1] is predicated upon a simple idea, if the difficulty of classifying each sample within a dataset is known, then the hard-to-classify samples may be presented to the network. One can estimate the difficulty of a sample through 3 terms, formulated as a Bayesian approach, given in the equation:

$$p(y \neq \hat{y}|x) \propto - \sum_{n=1}^{N_c} p(y_n|\hat{y}_n) p(x|\hat{y}_n) p(\hat{y}_n) \tag{3}$$

where y is the true label, $\hat{y}$ is the predicted label and x is the sample. To elaborate, the probability of the true label (y) not equalling the predicted label ($\hat{y}$) given a sample (x), is proportional to the negative sum of the probability of the network mislabelling a class, the likelihood of a sample given a class, and the probability of a class being predicted, summed across all classes. Unfortunately, the likelihood term is intractable and cannot be calculated directly. As such, it must be estimated using a variational autoencoder. This is achieved by applying a 0 mask to the latent space for each class, representing a class as the absent of parts. The predicted label is then calculated as the lowest value in the latent space which is calculated using:

$$w = [\sum_{j \in C_1} z_j^2, \sum_{j \in C_2} z_j^2, ..., \sum_{j \in C_{N_c}} z_j^2] \tag{4}$$

$$\hat{y} = \underset{n}{\operatorname{argmin}}\, w \tag{5}$$

where z_j is the latent space for the j^{th} class. This constraint is enforced by adding a new loss function to the standard variational autoencoder loss function.

$$L_{class} = H(softmax(-w), \hat{y}) \tag{6}$$

Where H is the cross entropy and w is the transposed vector of the latent spaces denoted in equation 4. This can then be combined with equation 7 and a tuning hyperparameter, λ, to give the final loss function.

$$loss = BCE(x, \hat{x}) + KLD(N(\mu_x, \sigma_x), N(0, 1)) + \lambda L_{class} \tag{7}$$

2.5 Pretraining

Pretraining can improve the performance of deep neural networks [13]. The intuition is that similar tasks are likely to require similar feature extractors, and as such similar network parameters. Pretraining involves training an architecture on an available dataset before it is trained on the current task, allowing it to learn the current task faster. Supervised pretraining is simple, whereby a network is first trained on an available labelled dataset.

Momentum contrast learning (MoCo) [14] is an unsupervised pretraining approach, the goal of which is to form low-dimensional representations of objects in order to distinguish them from one another. Formulated as a dictionary lookup problem, MoCo features 2 neural networks, as seen in figure 3.

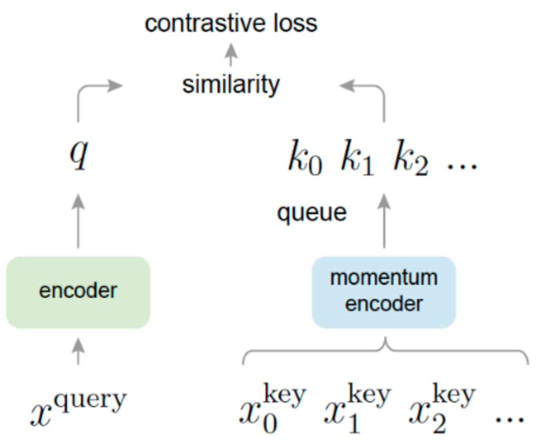


Figure 3: MoCo Architecture. The left network is what goes on to become the classifier for the Covid-19 prediction task. The right network is the dictionary network, used to facilitate the unsupervised learning methdology.

The momentum encoder network takes a stack of images as input, before outputting lower dimensional representations of them to form the keys to a dictionary. One of these input images is also given to the query network, but is heavily augmented through random crops, rotations, and similar transformations. The encoder network then outputs a lower dimensional representation of the image, with the aim of maximising its similarity to the matching output from the momentum encoder, while maximising the difference between it and the false-keys. In doing so, the encoder network is able to learn a set of feature extractors without ever using labelled data. This is beneficial as larger, web-scraped datasets such as the Instagram 1 billion image dataset may be used to train [14].

The contrastive loss function is imperative in achieving this goal [15]. Contrastive loss measures the similarity of sample pairs in representation space [14]. The primary difference between contrastive loss and traditional loss functions is that instead of calculating the difference between an output and a fixed target, the targets can vary during training and can be defined in terms of data representation as computed by a neural network. As discussed above, this task may be though of as a dictionary look up task. The keys are the low-dimensional representations $\{k_0, k_1, ..., k_n\}$. This dictionary is then queried with the low-dimensional q, which should match one of the keys (k_p), (i.e. they are low dimensional representations of the same image, however the image is randomly transformed before being presented to the query network). The contrastive loss function, known as InfoNCE [16], is used:

$$L_q = -\log \frac{\exp(q \cdot k_p / \tau)}{\sum_{i=0}^{K} \exp(q \cdot k_i / \tau)} \tag{8}$$

where τ is a temperature hyper-parameter. Intuitively, this loss function is the log loss of a softmax-based classifier that tries to classify q as k_p [14]. The most important aspect of this loss function is that neither the keys nor the query are fixed labels that are provided by a human-in-the-loop, instead both are generated during training enabling the entire system to train in an unsupervised manner.

While momentum contrast learning is not the first work to use contrastive loss, a primary contribution of the paper is training the dictionary network through momentum contrast. An alternative method could be to use 2 entirely separate networks, passing gradient back through both using back-propagation [15], however, larger dictionary sizes lead to more robust performances, and so the ability to back-propagate both networks is limited by GPU memory. As such, other works used a memory bank of low-dimensional representations produced by the query network from previous epochs [16]. However, there are clear issues with this approach as these representations are only updated when the query network sees the sample, and so the network may be trying to match the query to outdated representations. As such, momentum contrast is used whereby the parameters of the dictionary network, θ_k, are updated using a function of the query network:

$$\theta_k \leftarrow m\theta_k + (1 - m)\theta_q \tag{9}$$

where $m \in [0, 1)$ is a momentum coefficient, which allows the slow training and evolution of the dictionary network. As only θ_q is updated via back-propagation, larger input queues may be used.

3 Analysis and Specification

3.1 Problem Identification

The research questions defined in the introduction are chronological, and the validity of questions (2) and (3) are predicated on the premise that network performance is negatively impacted by class imbalance. If not, then there is little reason to address class imbalance in this domain.

Once the impacts are identified, implementing existing standard approaches provides useful insight into the problem, while also functioning as a benchmark for novel techniques to be measured against. Then, finally, the more substantial contributions may be made through exploring the efficacy of modern approaches and formulating novel adaptations to said approaches.

3.2 COVIDx Dataset

Chest X-ray images are suitable for several reasons. Firstly they facilitate rapid triaging and can be conducted in parallel with viral testing, such as RT-PCR testing [17]. Furthermore, chest X-ray imaging technology is widespread and accessible, and given the portability of many x-ray systems [17], there are a growing number of samples.

The COVIDx dataset is an aggregation of several other datasets:

- IEEE8023 Covid chest X-ray dataset [18]

- Non-COVID19 pneumonia patient cases and COVID-19 patient cases from the COVID-19 Image Data Collection [19]

- COVID-19 patient cases from the Figure 1 COVID-19 Chest X-ray Dataset Initiative [20]

- COVID-19 patient cases from the ActualMed COVID-19 Chest X-ray Dataset Initiative [21]

- Healthy and non-COVID-19 pneumonia patient cases from RSNA Pneumonia Detection Challenge dataset [22]

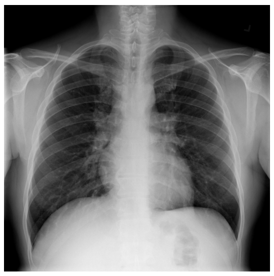


Figure 4: COVIDx sample image, an example of an input vector that will be classified.

The dataset is comprised of approximately 14000 chest X-ray images. These images belong to one of three classes: Covid-19 positive, healthy, or community pneumonia where the patient is ill but not with Covid-19. As seen in figure 6, there is considerable imbalance between the classes, with 8851 healthy samples, 6052 pneumonia samples and only 568 Covid-19 samples.

The dataset was compiled using an adapted script from the original authors [3]. In line with their methodology, the dataset was split into a training and test (validation) set. The training set consisted of 7966 normal samples, 5458 pneumonia samples, and 468 Covid-19 samples. The remaining samples were used as the unseen validation set, which is used to assess the network performance, as the classifier may overfit to the training set. Ideally, a third set would have been created, shown to the network only at the very end of training. This would demonstrate that the results are not biased towards the validation set. However, the dataset is too small to conduct another split and as the classifier is never trained against the validation set, it is still an effective measure of the network's ability to generalise to unseen data.

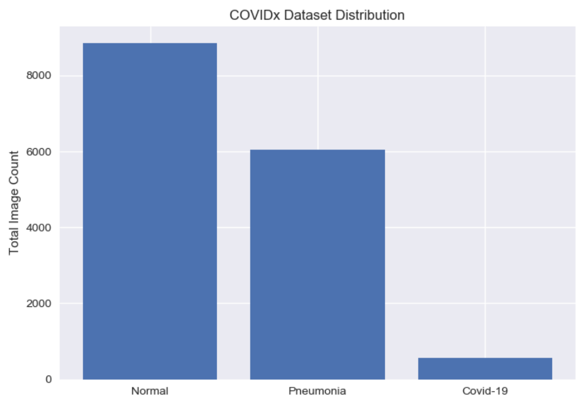


Figure 5: COVIDx dataset distribution. The dataset contains approximately $\sim 3.7\%$ Covid-19 samples.

As mentioned previously, class imbalance is often problematic as it may inhibit the network's ability to model the characteristics of the minority class. This is especially important in this task, as the nature of the domain dictates that identifying Covid-19 patients is the most important goal, as false negatives may lead to unwell patients not receiving treatment or self-isolating. Furthermore, as the dataset is only $\sim 3.7\%$ Covid-19 samples, networks could be published with 96% accuracy, without ever successfully classifying a Covid-19 sample. It is therefore necessary to define performance metrics that can accommodate these issues.

3.3 Performance Metrics

Validation accuracy is ubiquitous across deep learning literature. This is used to measure how well a network is doing across all classes, and it is calculated as the percentage of samples from the unseen dataset that are correctly classified. As mentioned above, this does not necessarily indicate strong performance on the Covid-19 samples.

$$Acc = \frac{\text{Correct Predictions}}{\text{Total Predictions}} \tag{10}$$

Covid-19 sensitivity is the performance on only the Covid-19 samples from the unseen validation set. This ignores overall accuracy, and measures how sensitive the network is to samples that contain Covid-19 allowing for a better measure of how well the network can overcome class imbalance and predict the minority class.

$$Sens = \frac{\text{Correct Covid-19 Predictions}}{\text{Total Covid Samples}} \tag{11}$$

Covid-19 positive predictive value (PPV) is used to address an issue in the sensitivity metric. If the network were to only predict Covid-19, the sensitivity would be 100%, but the network would not have learned to model the class features. Positive predictive value is the percentage of Covid-19 predictions that actually correspond to a Covid-19 sample.

$$PPV = \frac{\text{Correct Covid-19 Predictions}}{\text{Total Covid-19 Predictions}} \tag{12}$$

4 Design and Methodology

4.1 Discriminator Network

The discriminator network is the primary network in the study, responsible for classifying the COVIDx dataset. As covered in background (section 2), variations of the residual network are used in many Covid-19 classification tasks [3, 4]. ResNet18 and ResNet50 were selected as the network architectures because they featured heavily in current literature. While ResNet50 is the preferred network, ResNet18 was necessary due to the technical limitations of VaB-AL; as a generative model, it is much harder to reconstruct larger input vectors and so a smaller discriminator network is necessary. This is discussed further in section 4.4. Both networks were trained without imbalance strategies, as well as with standard approaches. Only the ResNet18 was trained with the VaB-AL methodology, and only the ResNet50 was trained with pretraining.

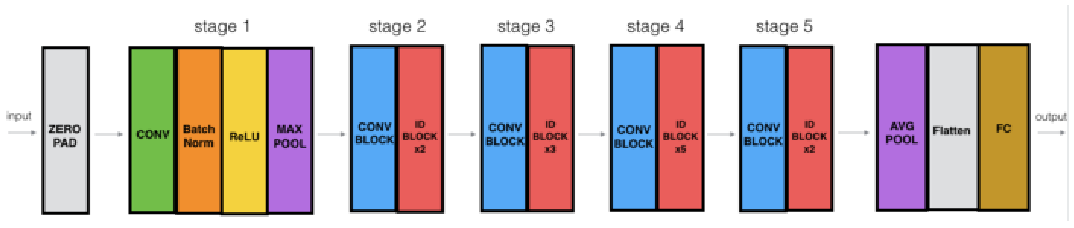


Figure 6: ResNet50 architecture [23]. Convolutional blocks feature a series of convolutional layers for feature extraction, while ID blocks feature skip connections to facilitate gradient flow.

The network was trained for 40 epochs for every experiment. A learning rate of 0.0002 was used with an Adam optimiser [24]. The Adam optimiser, adaptive moment estimation, is an extension of stochastic gradient descent algorithm (SGD). While SGD maintains a single learning rate for all weight updates, Adam computes individual adaptive learning rates for each network parameter from estimates of the first and second moments of the gradient. Adam has become increasingly popular in deep learning problems as empirical results indicate it outperforms other optimisation algorithms on a variety of tasks [25].

Cross Entropy was used as the criterion, and is simply the measure of difference between two probability distributions [8]. For the probability distributions q and p, It is defined as:

$$H(p, q) = -\sum_{x \in X} p(x) \log q(x) \tag{13}$$

where p and q are the probability distributions, and x is an input sample.

During training, a batch size of 8 was used. Input images were resized to 224x224 pixels for the ResNet50, and 32x32 for the ResNet18. The input images were also normalised, with values recommended by existing literature [9]. ResNet50 inputs were normalised with mean $(0.485, 0.456, 0.406)$ and standard deviation $(0.229, 0.224, 0.225)$, in the R, G and B channels. ResNet18 inputs were normalised with mean $(0.4914, 0.4822, 0.4465)$ and standard deviation $(0.2023, 0.1994, 0.2010)$.

4.2 Addressing Research Question 1

In order to evaluate both the impact of class imbalance and the efficacy of solutions to the imbalance, it was necessary to first complete control runs with the discriminator network to establish a baseline network performance. The network was presented the training set, split into batches, at each epoch. After each epoch the network weights were frozen, to prevent training, before it attempted to classify the validation set samples. Once the training completed the performance was recorded, and specific attention was paid to the Covid-19 sensitivity compared to the validation accuracy. This is because if the average sensitivity was substantially below the overall accuracy, then it is indicative that the minority class was underperforming and thus the imbalance was a detriment to the classification capability of the discriminator. While the results are discussed at length in section 7, it should be noted that the findings did indicate that class imbalance was negatively impacting the network, thus validating the decision to pursue research questions 2 and 3.

4.3 Standard Approaches

The first techniques that were addressed are commonly employed in a wide range of deep learning problems. The techniques are upsampling, loss function weighting, and data augmentation.

Upsampling, also known as oversampling, is a technique whereby samples that belong to a minority class have a greater chance to be selected and given to the network than majority class samples, helping even the class distribution that the classifier trains on. This is achieved by applying weights to a sampler that then randomly selects samples from the data distribution. Upsampling is very effective at increasing the presence of the minority class, however it does not increase the data pool and as such the variance is not increased. Therefore, upsampling may lead to overfitting on the minority class, actually reducing the performance of the network when classifying unseen data samples.

Following the findings of section 4.2, upsampling was the first method attempted. The weightings were static, applied class-wide before training began and selected such that each class was equally represented. The weights were $[\frac{1}{468}, \frac{1}{7966}, \frac{1}{5458}]$ for Covid-19, healthy, and pneumonia respectively.

Loss Function Weighting is somewhat similar to upsampling, in that class-wide weights are applied to ensure the minority class is appropriately represented. Instead of increasing the sample-rate of the minority class, however, this approach instead applies a greater penalty to the network should it misclassify a minority class sample. This is achieved by applying the same weights to the cross entropy loss function.

Data augmentation is considerably different to the other methods listed. Weights are not used, and instead the notion is that through random perturbations of the input vector, variance is increased. If variance in the dataset is increased, then theoretically performance should be preserved when generalised to unseen data. Data augmentation is not strictly a class imbalance strategy, but it has been shown to improve deep learning performance on image classification tasks [26]. There are many different types of data augmentation, this work focuses on affine transformations, random crops, random horizontal flips, and brightness and colour jitter.

Rotations, scaling and translations are all forms of affine transformations. A rotation between $\pm 10°$ is used, alongside a translation between $\pm 10\%$ in either axis, and finally a scale factor is randomly selected in the range $[0.85, 1.15]$. These variations help alleviate the noise that will be present in unseen data as a result of different patients, different X-ray machines and other real-world variance.

While affine transforms do increase randomness in the dataset, they preserve all lines and parallelism within the input vector. As such, the network may be able to overcome the variance by modelling the structure of the specific data samples presented, rather than identifying the true characteristics of the wider class. As such, random crops may be applied to help inhibit this underlying structure. Random crops simply take a random subsection of the input image, adding a different kind of variance to the dataset, as unlike affine transforms, crops may actually lead to data being lost from the initial input and thus identifying the true underlying characteristics becomes much more important. Instead of resizing the input vectors to 32x32 or 224x224, they are instead resized to 36x36 and 255x55. Following this, random crops of 32x32 and 224x224 may then be taken and used as input.

Random horizontal flips are very simple, and they do not change the structure of the data particularly. They are simply a trivial way of increasing the accessible data pool. They are achieved by applying the flip with a probability of 0.5 before presenting the input to the network.

Finally, colour and brightness jitter perturbs the pixel intensity in each of the input channels (red, green, blue). This doesn't change structural information in the images, but helps add noise that mimics variation in machinery and other real-world variance. The intensity is multiplied by a scale factor randomly selected from the range $[0.9, 1.1]$.

As data augmentation is not a class imbalance strategy, but rather a strategy to improve generalisation and increase variance across all classes, it was used in all subsequent experiments (VaB-AL, pretraining) to augment the learning.

Combining these 3 techniques was the final step in investigating the standard approaches. The hypothesis was that upsampling and loss function weighting would improve the minority class classification, while allowing for lower weightings to be used for each because weights were to be applied at multiple points in the training. As upsampling was likely to overfit, using data augmentation would be instrumental in ensuring the classifier could generalise to unseen data.

4.4 Variational Bayes for Active Learning

The first modern approach investigated was VaB-AL. The objective was to predict how hard each image was to classify for the discriminator. This was achieved using the equation presented in section 2. However, in order to achieve this goal the likelihood of the data, $p(x|\hat{y})$, must be estimated. As this is intractable, a variational autoencoder was used to estimate the likelihood of the data, as well as to predict the labels $\hat{y}$.

The input vector for the autoencoder is the normalised global average of each convolutional layer from the ResNet18. To elaborate, the output of each convolutional layer is taken, and the global average of that layer's output calculated, yielding a scalar output for each layer. These values are then concatenated into a single input vector, before being batch normalised. There are 2 primary advantages to doing this; firstly, the variational autoencoder can implicitly use the classifier's feature extractor, allowing for label prediction to be much more accurate as the VAE can better interpret the sample. This is important as the VAE has no convolutional layers of its own. Secondly, generative models struggle to reconstruct high-dimensional input vectors; by using the lower-dimensional feature representations of the sample, it is easier to achieve accurate reconstruction. This is also the reason that ResNet18 must be used with VaB-AL, using ResNet50 increases the input dimensionality substantially.

In order to obtain the scores, the VAE receives input and first averages and normalises the input, before feeding the input through the fully connected network. The predicted label, $\hat{y}$, is taken as the minimum class-latent space, as the class is represented through the absence of parts. The sample likelihood, $p(x|\hat{y})$, is estimated using the binary cross entropy of the masked reconstruction, as well as the masked Kullback-Liebler divergence. The prior probability, $p(\hat{y})$, is updated with the new prediction. These 3 terms are calculated 100 times for each sample in the dataset, because the process is probabilistic and the values are samples from the continuous distribution (z), it is necessary to sample multiple times to obtain the most consistent value. With the 3 terms calculated for each sample, they can then be multiplied together and summed across each class. Note here that the terms are actually a prediction of how easy the sample is to classify, and thus the scores are subtracted from 1, giving the difficulty of classification.

The first methodology evaluated was very similar to that used in the original publication [1]. The intuition is that by training on a subset of the data that is hard to classify, the network should improve quickly. The first subset is randomly selected and both the classifier and variational autoencoder train against it. After a set of epochs, the VAE predicts the scores of the remaining data samples and selects a new subset from that data. This process is repeated until the end of training. This method worked in the original paper as the subset was selected to be labelled from an unlabelled set and thus it was a necessary process to iteratively grow the labelled data pool.

However, this methodology proved ineffective for the COVID-19 classification task, disengaging entirely from the learning process. This is likely because the network overfits to the subset of the data. The total dataset is fairly small, so by taking subsets there isn't enough variance to facilitate generalisation to unseen data. As such, several adaptations to the methodology were made. The primary change was to present the entire dataset to both networks at each epoch. The scores were then calculated for the entire dataset. Instead of using them to select a subset, the scores were used for loss function and oversampling weights. As such, the methodology is essentially achieving the same goals as the standard approaches, but is now using per-sample, dynamic weights that change with the performance of the network instead of static, per-class weights that are determined a priori. This process is the primary novel contribution of the study.

4.5 Supervised Pretraining

Pretraining was the final approach explored. The classifier was pretrained against the ImageNet database [7], with a fully connected output layer of 10 nodes, corresponding to the 10 ImageNet classes. Once the network had trained on ImageNet, the output layer was replaced with a new fully connected layer, with 3 output nodes and randomly initialised weights. The parameters were frozen for every other layer, and so only the output layer was trained on the COVIDx dataset. The justification for pretraining is that the convolutional layers should form a generic feature extractor, while the output layer is trained to interpret these features and classify the input vector accordingly. Furthermore, pretraining is orthogonal to the other methods explored in this study and thus may be used in parallel. That is, pretraining the network does not prevent loss function and upsampling weights being applied during the training of the output layer on the Covid-19 dataset.

4.6 MoCo

As per the literature [14], training a network with momentum contrast learning required up to 7 days of permanent training across 8 GPUs, making it infeasible to conduct the pretraining within the constraints of this study. Fortunately, the original authors provide a pretrained MoCo network for this reason, it is trained with the method outlined in section 2. The network is a ResNet50 architecture, and in the paper stochastic gradient descent with weight decay was used as an optimiser. This optimiser was replaced with Adam, to ensure consistency across methodologies employed here.

The network's weights were then frozen for all layers bar the output layer, which was replaced with a randomly initialised output layer with 3 outputs, corresponding to Covid-19, healthy, and pneumonia predictions. The network was trained for 40 epochs, and used the cross entropy loss function.

5 Implementation

5.1 Technologies

The project was developed in Python3, using the PyTorch machine learning library [27]. PyTorch provides implementations for data loaders, data augmentation, loss functions and optimisers. Furthermore, PyTorch allows neural networks to be implemented using an object oriented approach, as the developer may extend the *NN.Module* class to create unique networks that are still able to abstract away the internal complexity of fully connected and convolutional layers, forward passes and back propagation.

All networks were trained on GPU architecture, utilising CUDA cores for faster, parallel training [28]. Both an Nvidia 1080ti and Nvidia RTX 2070 was used for training.

5.2 Data Loader

A PyTorch dataloader was used to handle the processing of the dataset as there is inbuilt multi-threaded support. One data loader was used for the training set, and another for the test set. The data was loaded using an ImageFolder, which automatically assigned labels to the data samples based on file structure (fig 7). The numeric labels generated ($[0, 1, 2]$) are saved in a dictionary, corresponding to their human readable names (Covid-19, normal, pneumonia) to allow easy tracking of performance. By using separate ImageFolders for the training and test sets, data augmentation could be applied only to the training set. It is important to keep the validation set free from perturbations in order to accurately compare the performance of different methods.

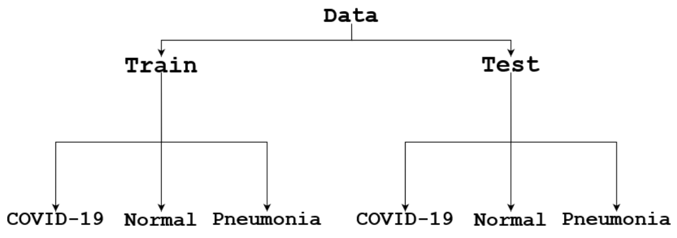


Figure 7: Dataset File Structure

The images are resized to 255x255 for ResNet50 tasks, or 36x36 for ResNet18 tasks. They are then randomly cropped to 224x224 and 32x32 for the training tasks, and they are centre cropped for validation tasks. Data is loaded in batches of 8 during training.

For the traditional upsampling, PyTorch's weighted sampler object can be instantiated and directly inserted into the dataloader, which applies the class-wide weights to the dataset. However, due to the increased complexity of applying dynamic VaB-AL weights to each sample, a custom data class was created for the VaB-AL experiments; it extends the PyTorch dataset class and thus seamlessly integrates with the training methodology employed for other experiments. The class contains an image folder that applies the data augmentation and loads these images into memory. The dataset is modelled as a dictionary, containing the keys [image, target, weight]. The image and label values are acquired by parsing a meta data file, generated during the original dataset compilation; these values are static throughout training. The weights, initially instantiated to 1, may then be updated during training by calling the *apply_weights* function, or altering the *weights* variable.

5.3 ResNet18

The ResNet18 architecture implementation from the original VaB-AL thesis was used. The network is comprised of a convolutional layer, followed by batch normalisation in 2 dimensions, feeding into 4 residual blocks shown in figure 8.

The residual block is a standard series of convolutional layers and batch normalisation. The primary difference is the residual shortcut that adds the input, x, to the output of the final layer. After each batch normalisation layer, the output is fed through a non-linear activation unit, a ReLU. The last ReLU takes both $f(x)$ and (x) as input.

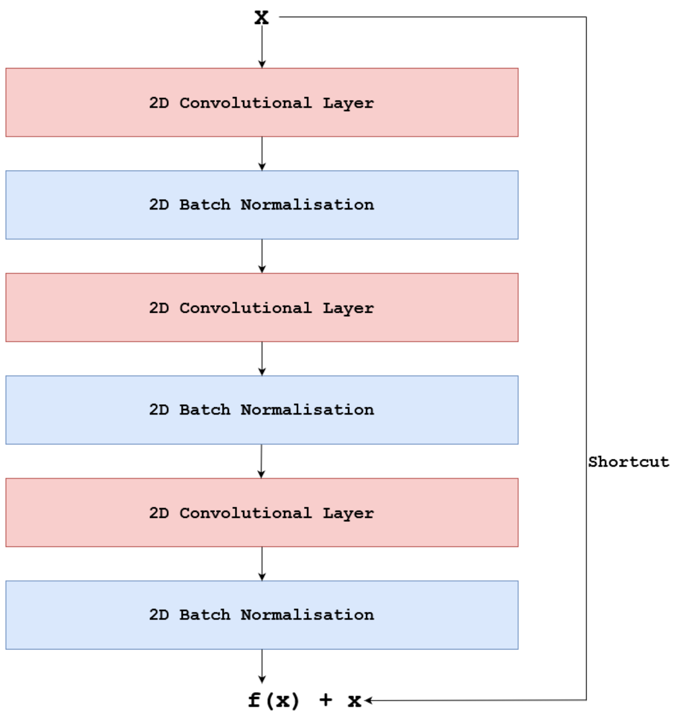
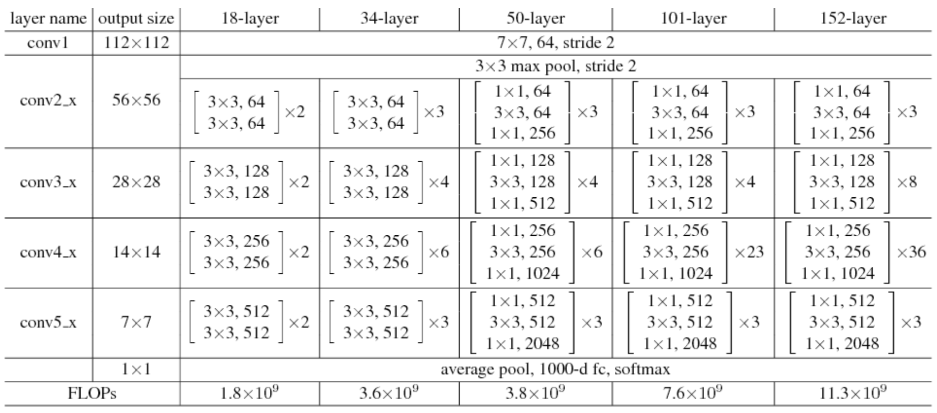


Figure 8: ResNet18 Residual Block

While the PyTorch ResNet50 model is used, this version of ResNet18 is preferable to PyTorch's as it allows for a custom function to be created within the network's class, which takes the output of each residual block and returns this to the variational autoencoder during the VaB-AL experiments, to be used as VAE input.

5.4 ResNet50

The ResNet50 architecture is imported from the prebuilt models section of PyTorch. This is firstly because there is no reason to reimplement the architecture, but it is also to ensure consistency in implementation between the momentum contrast runs and the ResNet50 experiments. The momentum contrast network must use the PyTorch implementation for reasons covered in section 5.7.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 9: PyTorch ResNet50 implementation [9, 27]. Image shows the architecture of all available residual networks within PyTorch.

The precise architecture is outlined in fig 9, under the 50-layer column. Layers 2-5 are residual blocks which feature the skip connections illustrated in figure 8.

5.5 Training

While Va-BAL and momentum contrast learning require specific training paradigms, all training followed the same fundamental loop. Once the data had been loaded, the network trained for 40 epochs. The data was loaded in batches of 8 samples, and the network first predicted the labels for these samples before the cross entropy loss function calculated the loss, which was then back-propagated though the classifier.

Once the network had trained against all train-set samples the weights were frozen on the network and training was paused, so that the validation set could be presented to the network to track network performance on unseen data. These scores were then recorded for later evaluation. A high-level pseudocode algorithm of the training loop can be seen below:

Algorithm 1: Classifier Training Loop

```
for epoch in range(num_epochs))
 do
    for data in training_dataset
     do
        inputs, targets = data.images, data.labels
        predictions = classifier(inputs)
        loss = cross_entropy(predictions, targets)
        back-propagate loss
        optimise weights
    end
    for data in validation_dataset
     do
        freeze classifier weights
        inputs, targets = data.images, data.labels
        predictions = classifier(inputs)
        accuracy = number of correct predictions
        save(accuracy)
    end
end
save(classifier)
```

5.6 VaB-AL

The VaB-AL implementation is more complicated than other approaches, due to the presence of a second network. The training loop is largely identical to the algorithm in section 5.5, it is simply extended to facilitate the training of the VAE as well as sample weight assignment.

The variational autoencoder uses fully connected layers instead of the convolutional layers. Upon receiving input, the network first preprocesses the input vector by taking the global average for each input feature, and then applying batch normalisation to the processed vector. Finally, these values are then concatenated into a single fully connected layer, forming the first layer of the network.

The encoder portion of the network consists of three fully connected layers, with a rectified linear unit placed between each layer. The final layer of the network is actually two fully connected layers in parallel, one corresponding to the mean, and the other to the log variance of the Gaussian distribution that is modelled in the latent space. The log of the variance (logvar) is predicted instead of the variance directly, as the encoder output could be negative, which is invalid for the variance; it is easier to calculate log of the variance and then raise e to the power of logvar to acquire the variance value. The latent space vector is 10 dimensions for both the mean and logvar, per class, resulting in a latent space size of 30 for each latent variable and a latent space of size 60.

The latent space is generated through the reparametrization trick outlined in equation 14, using an ϵ parameter to represent the random sampling of the continuous distribution, allowing the mean and logvar nodes to be trained through back propagation.

$$z = \mu + \epsilon \cdot \sigma \tag{14}$$

From this, the decoder is able to reconstruct the input vector. The decoder architecture is simply an inversion of the encoder, with an identical structure simply reversed in order to expand the dimensionality of the latent space to estimate the input vector (from z to $p(x|z)$).

While the classifier is trained in line with the other methods, each epoch also features the training of the VAE. The first step of training the VAE is to freeze the classifier weights and pass the sample through it, thus obtaining the low-dimensional feature representations of the sample. This is then passed through the variational autoencoder which returns the reconstructed vector, the mean and logvar values which are then used to calculate the three loss terms for the VAE:

- Binary Cross Entropy between the input vector and the reconstruction.
- Kullback-Liebler Divergence to ensure the Gaussian distributions are constrained.
- The class loss function to ensure the latent space is accurately modelling the predicted class.

The VAE training loop is as follows:

Algorithm 2: VAE Training Loop

```
for epoch in range(num_epochs))
 do
    for data in dataset do
        Sample, label = data.image, data.label
        features = classifier.extract_features(sample)
        mean, logvar, z, reconstruction = vae(features)
        VAE loss = BCE + KLD + Loss$_{class}$
        back-propagate loss
        Update VAE parameters
    end
end
```

The classifier network and VAE are trained against the training set 4 times and 5 times per epoch, respectively. After this epoch the VAE updates the scores for each training sample, before the classifier predicts the validation set labels to track the classifier performance. At the start of the next epoch of training, the sample weights are replaced with the newly calculated values, and the cycle repeats for 10 epochs, ensuring the classifier trains for 40 cycles of the training set - in line with other methods. A psuedo-code algorithm can be seen below:

Algorithm 3: VaB-AL Training Loop

```
for i_round in num_rounds do
    for epoch in range(num_epochs))
     do
        Classifier Training Loop
        VAE Training Loop
    end
    Estimate likelihood of samples
    Calculate prior probability of predicted labels
    Calculate labelling error probability
    Update sample weights
    Predict validation set
    Save validation scores
end
save(classifier)
```

5.7 Pretraining

The ResNet50 architecture that is imported from PyTorch is also available in a pretrained version, where the network was first trained against the ImageNet database. Once imported, the network's output layer is replaced with a new 3 node fully-connected layer, and then the weights are frozen for all other layers. The network is then trained in accordance with the training regime in section 5.5.

As mentioned previously, training the MoCo network takes a very long time, and thus it was infeasible to complete this training within the scope of this study. As such, the MoCo network was also acquired already pretrained. The pretrained network was stored as a state dictionary, which is a byte file that stores the parameter settings of the classifier network. As such, it is necessary to use the exact class in the source code from the original paper, or else the weights will not map to the correct parameters. As is the case with supervised learning, once pretrained, the classifier output layer can be replaced, and trained against the Covid-19 dataset for 40 epochs using the training loop from section 5.5.

6 Project Management

This project was of substantial size and complexity. Before the project began discussions were held to determine its general direction, before an initial project proposal was developed. This proposal featured a Gantt chart which helped keep the project on track. In practice, however, the project development only loosely followed the Gantt chart and infact the investigation of pretrained networks wasn't planned for before the project was already underway.

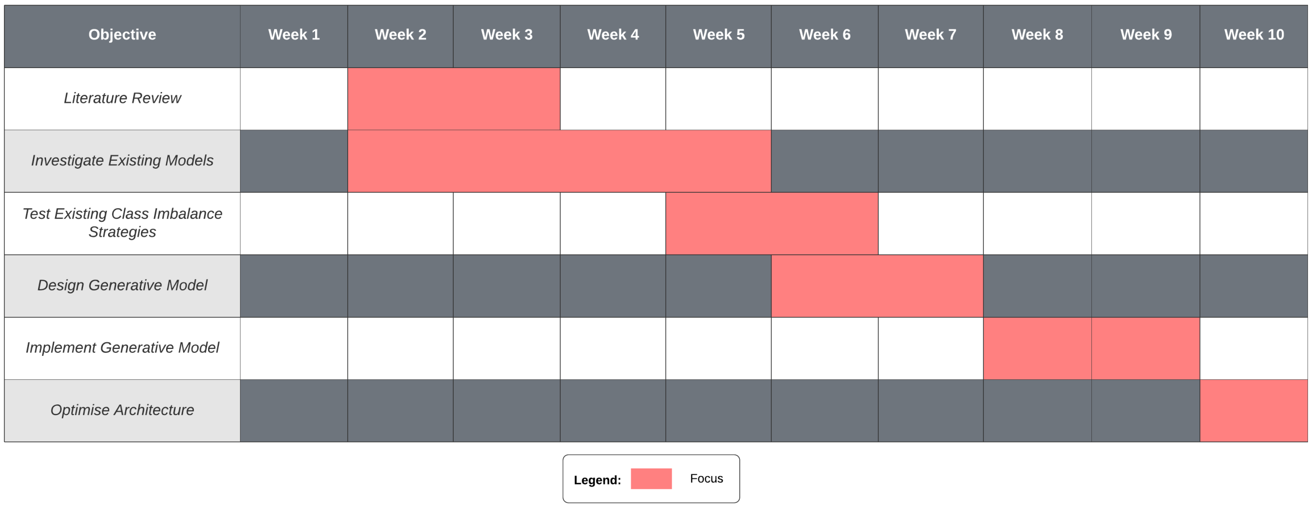


Figure 10: Gantt chart visualising the anticipated project progression

Many of the topics studied (such as variation autoencoders and momentum contrast learning) were beyond the scope of my previous experience, and so weekly meetings with supervisors were invaluable. Due to the weekly nature of the project, an agile Scrum methodology was adopted, ensuring that each week a set of small objectives were achieved so that the meetings were constructive and any lingering issues could be addressed.

Risk was managed throughout the project with a range of strategies. Firstly, using Git for version control helped mitigate the risk of data loss, which is always a threat due to the nature of digital storage. Furthermore, the version control features of Git enabled code rollback to occur, should any catastrophic errors be pushed to the main branch.

Another risk was that innate variance in the training paradigms could lead to unreliable results. This was managed through repeating each experiment three times, and using the mean score as the result. Error bars were also calculated using the standard deviation, in order to indicate the severity of the variance. However, 3 runs per experiment is not optimal, and variance will still play a large role in the results. A more effective and robust method would be to conduct statistical testing such as Wilcoxon signed-rank testing, however this requires 21 runs to be conducted per experiment which is far beyond the time constraints of this project.

To improve the methodology the vision and direction for the project should have been clearer before development steps began. While it is the nature of agile methodologies that adaptations should be made week-to-week, the lack of overarching direction with the project led to the occasional week being stagnant in terms of progress.

7 Results

Experiments ran for 40 epochs and were repeated 3 times each in order to mitigate randomness and variance in data augmentation, batch selection and parameter initialisation. While more runs would be preferable, time constraints dictated that 3 runs was the highest feasible value.

7.1 Research Question 1

The first experiments were conducted to determine whether class imbalance did cause a reduction in network performance. This was accomplished through analysing the network's performance on the validation set at each epoch, comparing the performance across all classes against the performance specifically for the Covid-19 samples.

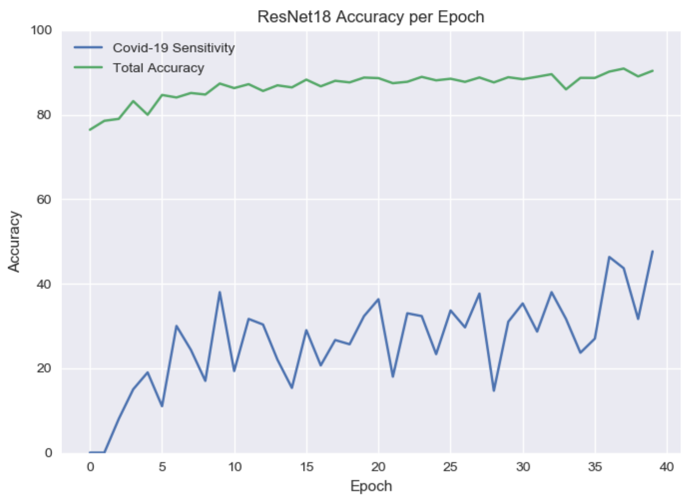


Figure 11: Control ResNet18 performance, no data augmentation or class imbalance strategies.

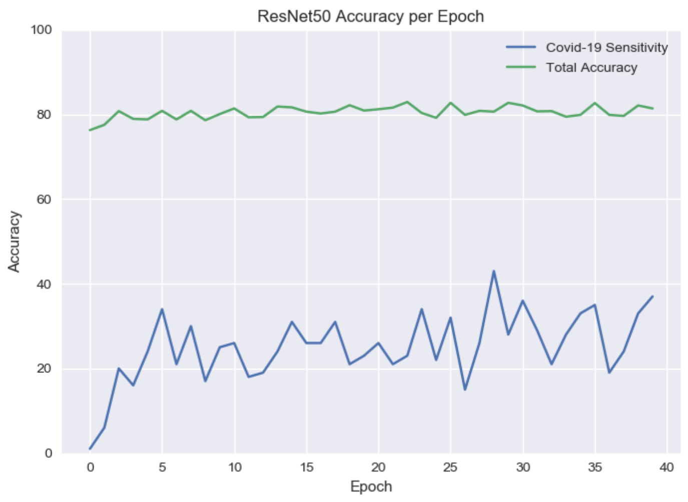


Figure 12: Control ResNet50 performance, no data augmentation or class imbalance strategies.

As seen in figures 11 and 12, the average accuracy of the networks lingered around 90% and 80% accuracy for the ResNet18 and ResNet50, respectively. However, the Covid-19 sensitivty for the validation set is substantially worse for both networks, averaging around 40% performance, never surpassing 47% sensitivity.

Such a huge deficit in performance strongly suggests that there are substantial negative consequences to dataset class imbalance, which justified the exploration of potential strategies to alleviate these issues.

7.2 Research Question 2

The performance of each independent classic strategy can be seen in figure 13. Oversampling is the most successful method for achieving high Covid-19 sensitivity, reaching 84% compared to a sensitivity of $\sim 38\%$ for the control ResNet50. As Upsampling outperformed loss function weighting by such a large margin, it appears that it is more important to view the samples regularly than to have a high reward in order to achieve accurate modelling of a class. It was also to be expected that upsampling would outperform data augmentation, as the latter has no intrinsic property that would lead to large improvement minority class classification, it is instead present to aid generalisation.

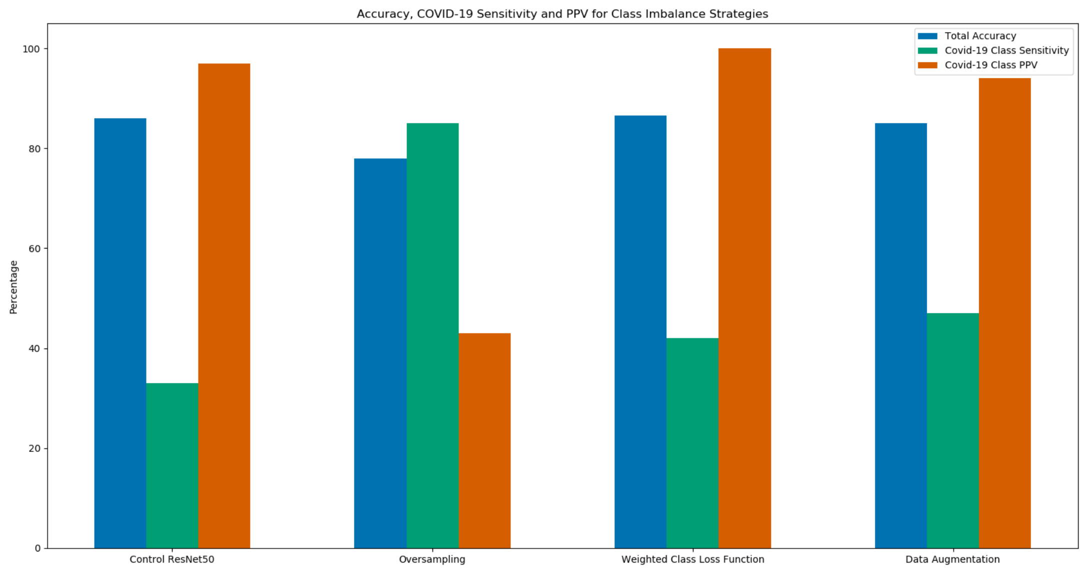


Figure 13: ResNet50 Classic Imbalance Strategy Comparison

While upsampling performs very well against the sensitivity metric, it should be noted that the total accuracy and Covid-19 positive predictive value both indicate that the strategy is fundamentally flawed. This is because despite the increase in minority class performance, the total accuracy actually decreases for the overall validation set. The positive predictive value offers some insight as to why this might be, as the metric drops from $\sim 95\%$ to approximately 45%. This means that the network is predicting Covid-19 very often, but is wrong more than half of the time, which causes a substantial reduction in the majority-class performance.

In contrast to upsampling, loss function weighting realises a much smaller sensitivity growth of $\sim 8\%$. While this is a much less relevant increase, it does come without the cost attached as the positive predictive value and total accuracy actually increases when the weighted loss function is used. This may be because the increased penalty encourages the network to actually learn the underlying characteristics of the minority class, and so not only is it able to better predict Covid-19 samples, the classifier is likely able to discriminate between important similarities between samples, and arbitrary, irrelevant ones. To elaborate, the greater penalty may be encouraging the classifier to learn the meaningful features that correspond to Covid-19, reducing false positives as it learns to ignore arbitrary similarities that occur due to noise and variance; similarities that do not actually correspond to samples belonging to the same class.

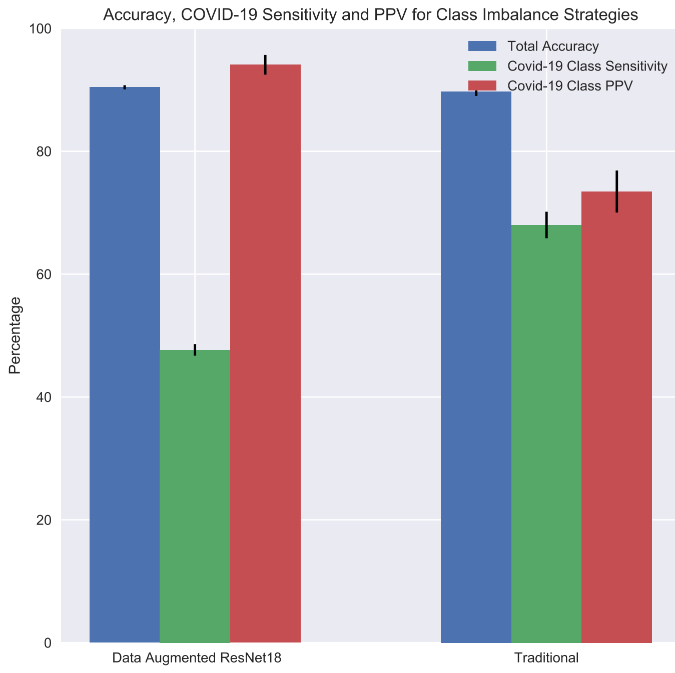


Figure 14: ResNet18 Combined Classic Strategies Comparison

The combination of all three approaches led to a more effective strategy than any one individual method. This is possibly because, through combining the loss function weighting with the upsampling, it is possible to take lower values for each weight - helping reduce the negative consequences of oversampling. This is further supported by the fact that the sensitivity is slightly reduced compared to using only upsampling. Furthermore, as upsampling is likely causing the network to overfit to the minority class samples, the use of data augmentation plays a vital role in increasing the data variance, allowing for more robust models to be learnt.

Figures 14 and 15 show the network's performance with only data augmentation against a network using all 3 methods. The error bars illustrate the standard deviation across the 3 runs, they suggest that introducing upsampling and loss function weighting greatly increases the variance in network performance, which is an observation that has no obvious cause; further experimentation would be needed to determine the reason for the added variance.

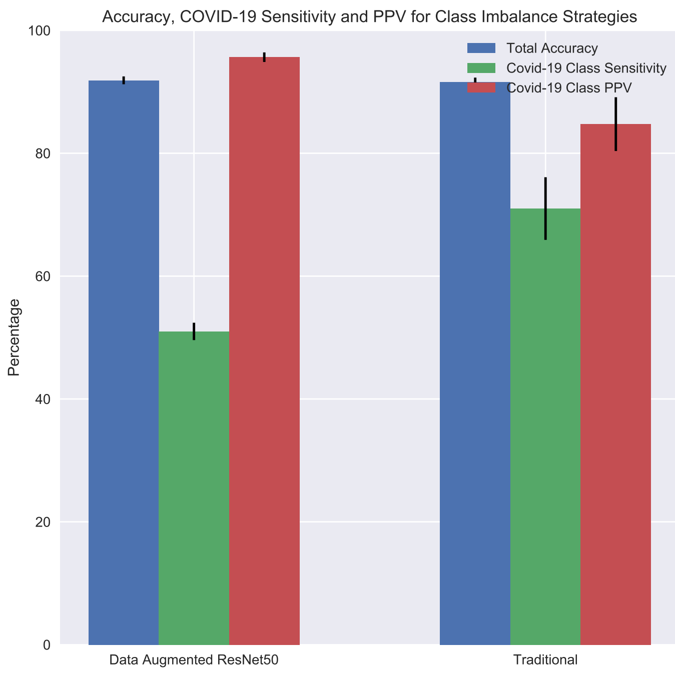


Figure 15: ResNet50 Combined Classic Strategies Comparison

Data augmentation is not strictly a class imbalance strategy, but rather a general good practice for machine learning tasks. As such it was used in all future experiments. When other methods (such as VaB-AL and MoCo) are compared to the control results, all networks used data augmentation.

7.3 Research Question 3 - VaB-AL

The first novel application of VaB-AL was for loss function weighting. Loss function weighting was the worst performing variation of VaB-AL, with an average finish of 50% Covid-19 sensitivity, realising an increase of about 5%. Figure 16 illustrates the change in all metrics per epoch. As discussed earlier, a Va-BAL epoch is equivalent to 4 epochs in other experiments. The figure indicates that the total accuracy is at its peak after only 5 epochs, and the sensitivity is at its second highest performance during this epoch, also. After the fifth epoch, total accuracy declines continuously. This suggests that the network has overfit after only half of the available training time. If the network was stopped preemptively, the network could have achieved average performance of up to 55% performance, with $\sim 91\%$ total accuracy. As discussed for the second research question, the poor performance compared to upsampling is likely because viewing samples regularly is pivotal to modelling the underlying class characteristics.

The loss function weighting has very stable ppv and total accuracy values, with both hovering at 90% for almost all of the training, the primary change per epoch is found in the sensitivity metric. This is to be expected, as the VaB-AL thesis should in general prioritise minority class samples. This is a finding that extends to all VaB-AL strategies explored in this study.

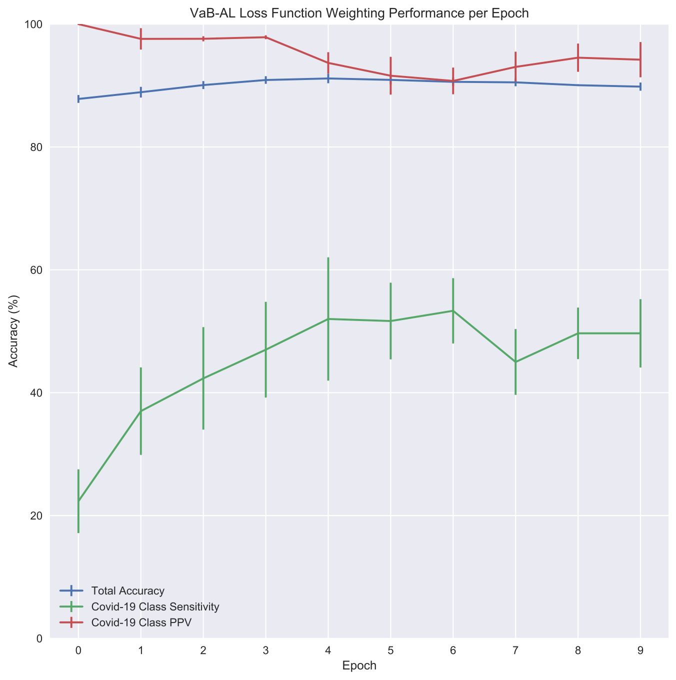


Figure 16: VaB-AL Loss Function Weighting Performance

Using the weights for upsampling performs better than loss function weighting on sensitivity, but worse for total accuracy. The final results for upsampling were a total accuracy of 89%, with a sensitivity of 52% and positive predictive value of 88% (fig 17). The first observation with the upsampling approach is that either the network has potentially overfit, as the 9th epoch boasts a total accuracy of 92% and a Covid-19 sensitivity of $\sim 61\%$, or the network has not yet finished training and the training was stopped preemptively. This is because the final epoch sees a significant drop of 9% sensitivity and 2% total accuracy. While this may indicate the beginning of overfitting, the network also saw similar performance decreases earlier in training before recovering to find better solutions within the parameter space. It is especially hard to assess the efficacy of upsampling, because the variance in performance is very high. Across the 3 runs conducted, a single standard deviation of the final epoch could see the network output a sensitivity of 41%, or 64%. Furthermore, taking a single standard deviation of the best performing epoch reaches performances matching that of traditional methods, peaking at 70%. Similar to the standard methods tried, upsampling saw a larger increase in sensitivity, in exchange for a decrease in total accuracy compared to loss function weighting.

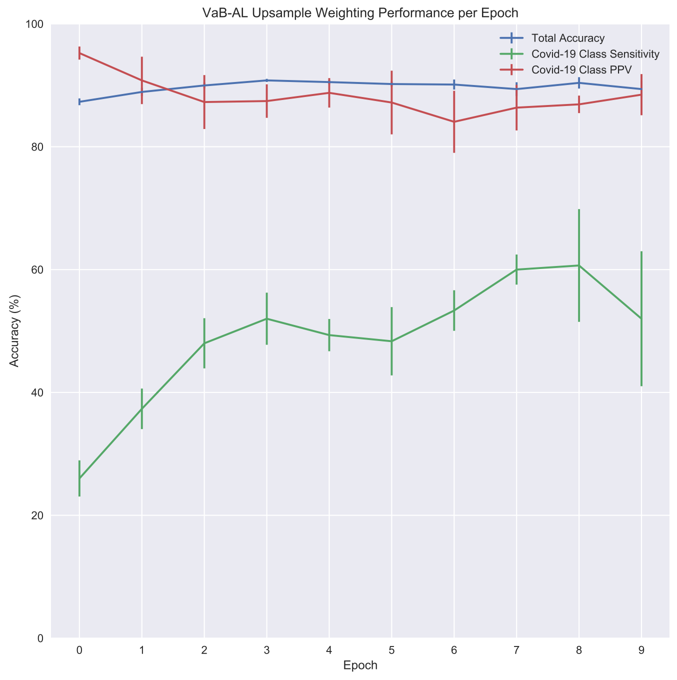


Figure 17: VaB-AL Upsample Weighting Performance

The final method combined both upsampling and loss function weighting with VaB-AL. Like traditional methods, combining these strategies led to an improvement in performance, but unlike traditional methods both the accuracy and sensitivity improved. The classifier achieved a total accuracy of 91%, alongside a sensitivity of 55%. However, once again, evidence strongly suggests that overfitting has occurred. Figure 18 shows a clear peak-performance at the seventh epoch for both the overall accuracy and the sensitivity. If early stopping is employed, the accuracy and sensitivity become 90% and 60%, respectively. While all three methods outperformed the original data augmented ResNet18, using both weights was the most effective VaB-AL strategy, and so this was the method selected to represent VaB-AL when comparing performance to other strategies.

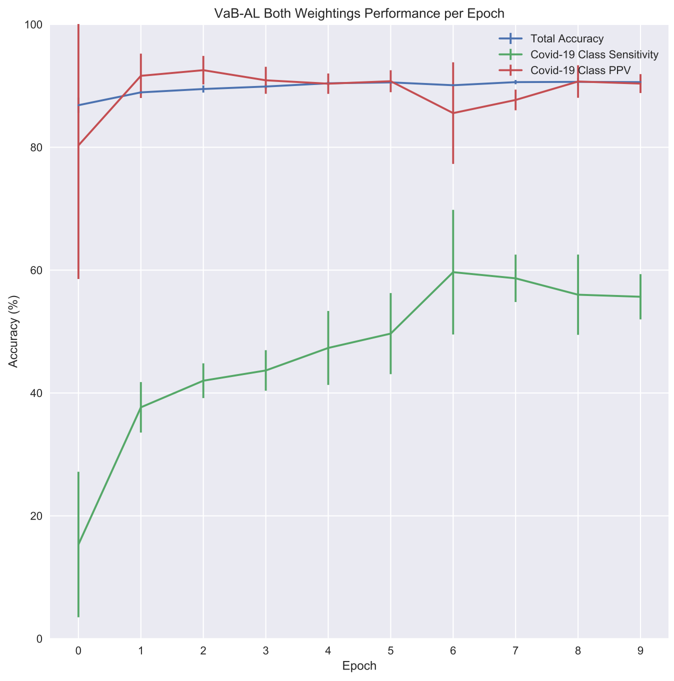


Figure 18: VaB-AL Both Weights Performance

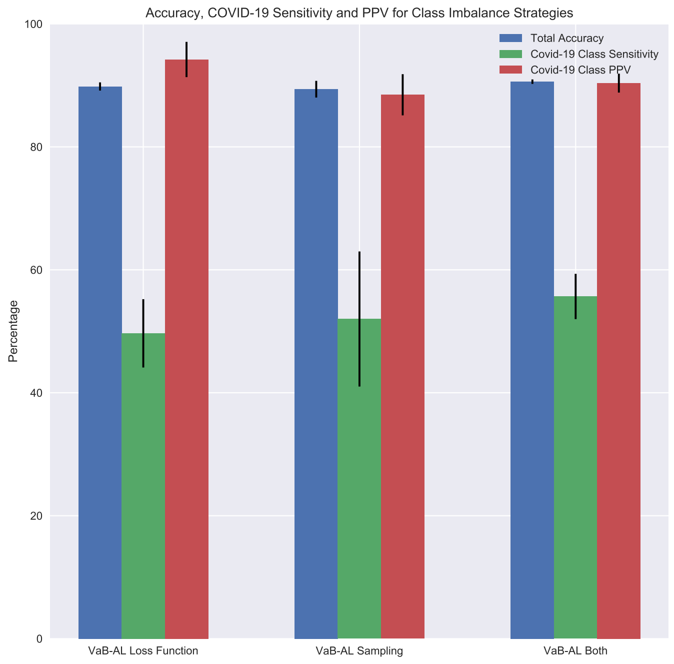


Figure 19: VaB-AL Methods

Figure 20 shows the performance of the original network (with data augmentation), the traditional strategies, and VaB-AL. While traditional strategies do outperform with regard to Covid-19 sensitivity, it is clear that this has a rather large cost attached. In order to obtain the 68% sensitivity, the method sees a reduction in positive predictive value by approximately 20%, and a reduction in overall performance by around 1.5%. In contrast, the smaller sensitivity increase produced by VaB-AL has no such cost attached, as while there is a slight reduction in ppv, overall accuracy actually increases by 1%. This suggests that the VaB-AL thesis is also helping improve the ability of the discriminator to classify the majority classes, too. This is logical, as while traditional methods use static class-wide weights, VaB-AL is not forced to ignore the majority classes, and it may learn to identify the harder areas of the majority class sample space, thus improving the classifier's ability to model the underlying characteristics of the majority classes.

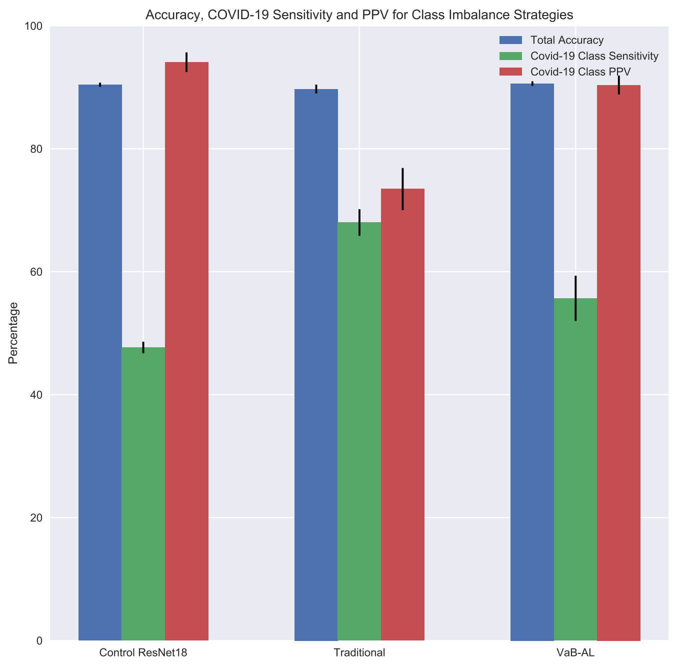


Figure 20: VaB-AL Comparison

The overfitting problem is shared between all VaB-AL implementations, however the overfitting occurs at different times during training. The final figure in this section, figure 21, compares the performance of the VaB-AL network with early stopping and both weighting functions. As discussed before, the sensitivity is notably better and actually rivals that of the traditional methods; one of the runs actually achieved performance of 73%, which is very clearly matching the traditional strategies' performance. While elaborated upon in section 8 (discussion), it should be emphasised that the VaB-AL performances varied greatly, and that with a grid-search to identify optimal hyper-parameters and extended training times to confirm overfitting has occurred, it is likely that the VaB-AL strategies would outperform traditional methods. Furthermore, statistical testing such as Wilcoxon signed-rank testing would be needed to confirm that any of these methods actually lead to statistically significant performance increases, however obtaining the necessary number of runs to perform such tests was out of the scope of this study, due to time and hardware constraints.

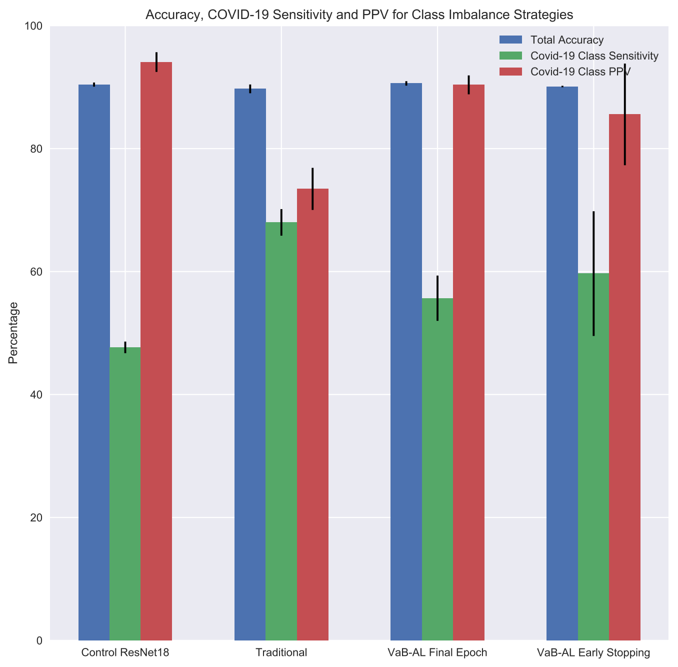


Figure 21: VaB-AL Early Stopping Comparison

Method	Accuracy (%)	Sensitivity (%)	PPV (%)
Control	90	47	**95**
Data Augmentation	90	48	94
Upsampling + LF weight	89	**67**	73
VaB-AL Both Weights	91	56	90
VaB-AL Both Weights with early stopping	91	60	88
VaB-AL LF weight with early stopping	91	55	**95**
VaB-AL Upsampling with early stopping	**92**	61	87

Table 1: ResNet18 Methods Results Comparison

7.4 Research Question 3 - Pretraining

Upon initially comparing the results from the different strategies tried (figure 22), it appears as though pretraining has an adverse effect on performance. While this is true for the final epoch shown in the figure, the pretraining methods exhibit a much higher variance in their performance and had a higher accuracy and sensitivity for much of the training.

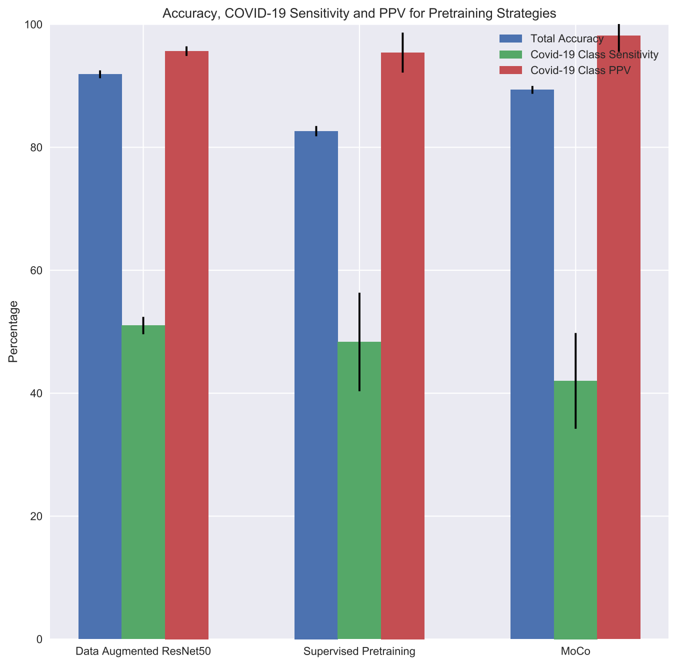


Figure 22: Pretraining Performance, No Balancing Strategy

Momentum contrast is very erratic in its training, and thus it is hard to draw conclusions from its performance. Figure 23 shows that MoCo, without any balancing strategies applied, was actually able to reach Covid-19 sensitivity values of 68%, while also regularly dropping to performances below 30% sensitivity, even after 15-20 epochs of training. This notion that the final epoch is an unfair representation of momentum contrast's potential is reinforced by the findings in figure 24. MoCo can be seen to consistently outperform the other strategies through all of training, only to see a substantial drop at the end of performance. This could be due to overfitting, however the erratic and unpredictable nature of its training from the start of training would suggest that it was rather just chance that lead to a worse performance in the final epoch.

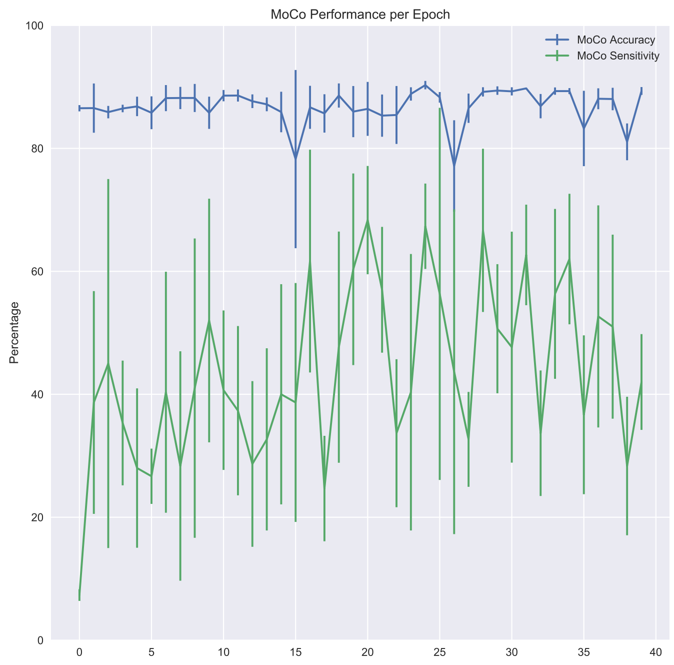


Figure 23: MoCo Accuracy and Sensitivity per Epoch

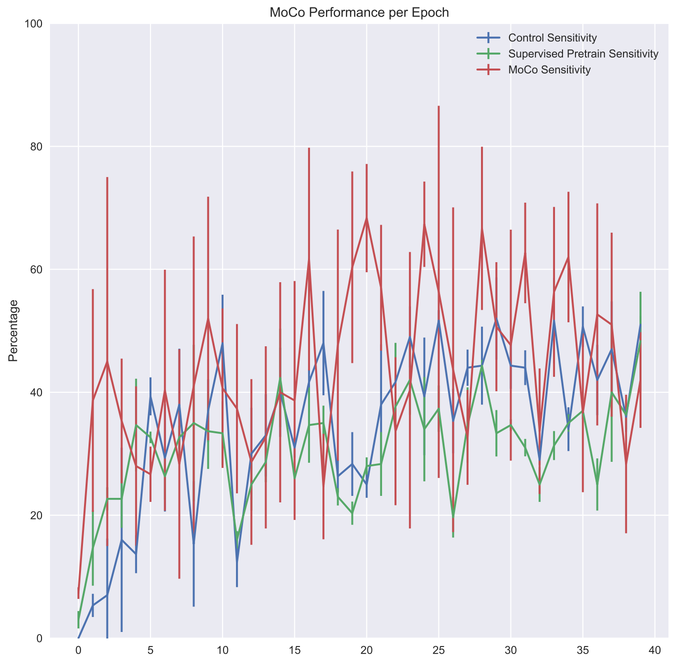


Figure 24: Pretraining Sensitivity Per Epoch, No Balancing

It is uncertain why MoCo suffers from such large perturbations in performance epoch to epoch; when the learning rate was reduced there was no change to this behaviour and thus the true cause remains unclear.

Unlike MoCo, supervised pretraining does appear to train as expected, however it has a lower sensitivity than the original network. Furthermore, both pretraining algorithms have a lower total accuracy, which is not surprising as they are unable to fine tune their feature extractors to identify important features that are unique to this domain, as only the output layer is trained against the Covid-19 dataset. While MoCo's total accuracy is also erratic, it is consistently worse than that of traditional learning (fig 25).

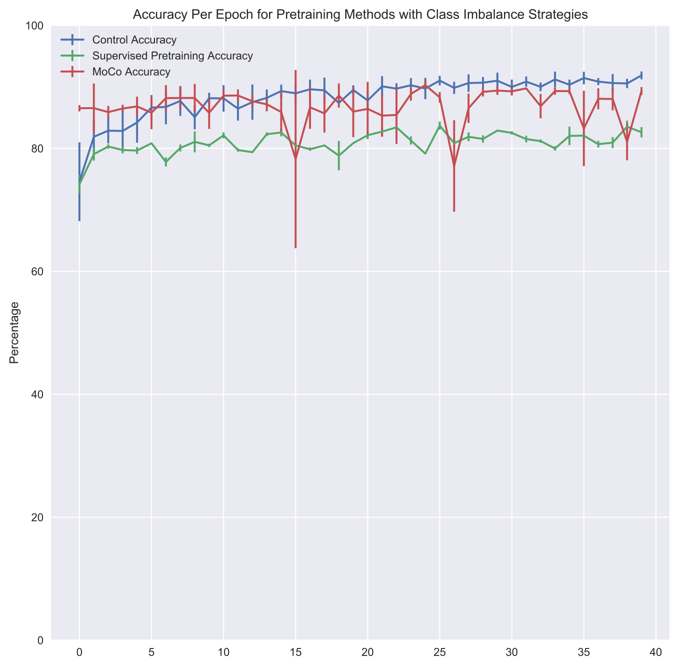


Figure 25: Pretraining Accuracy Per Epoch, No Balancing

Following these experiments, the traditional class imbalance strategies were applied to the pretraining architectures. Interestingly, despite worse performance without the strategies, supervised pretraining surpasses the control network in Covid-19 sensitivity, ending on a 78% sensitivity, compared to the 70% of the control network. This does, however, come at the expected cost of a severely reduced positive predictive value, dropping from $\sim 85\%$ to $\sim 55\%$. Furthermore, both pretraining models suffered from a significantly reduced overall accuracy.

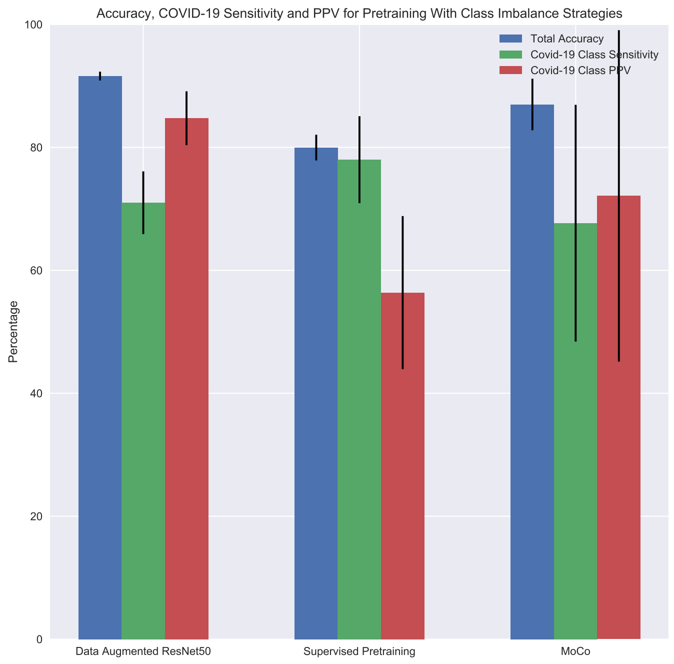


Figure 26: Pretraining Performance, Classic Balancing Strategy

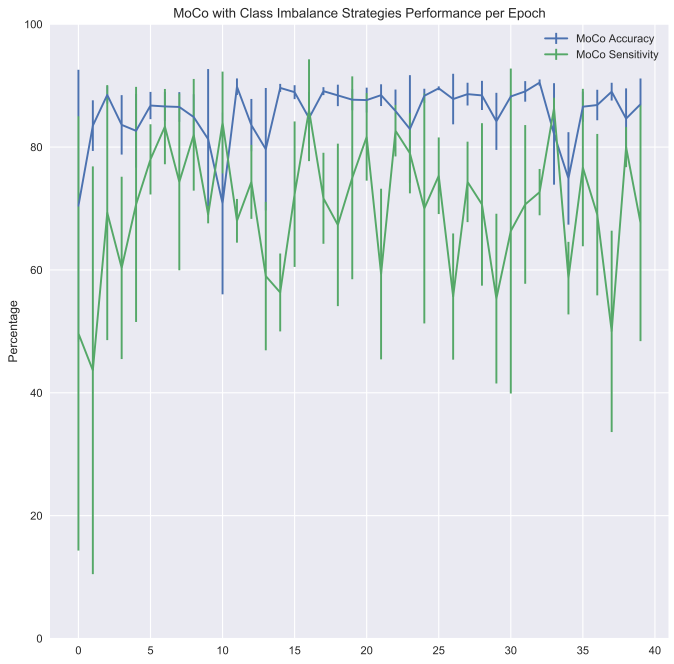


Figure 27: MoCo Accuracy and Sensitivity per Epoch, With Imbalance Strategies

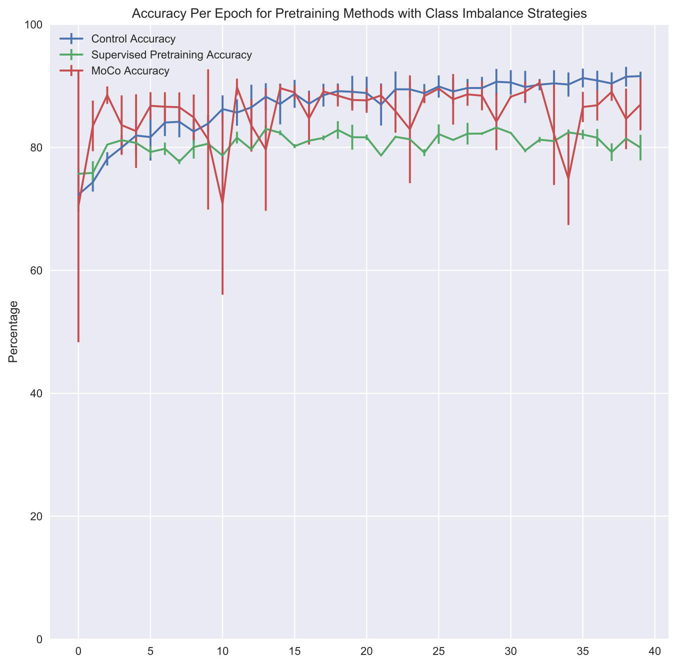


Figure 28: Pretraining Accuracy Per Epoch, Classic Balancing

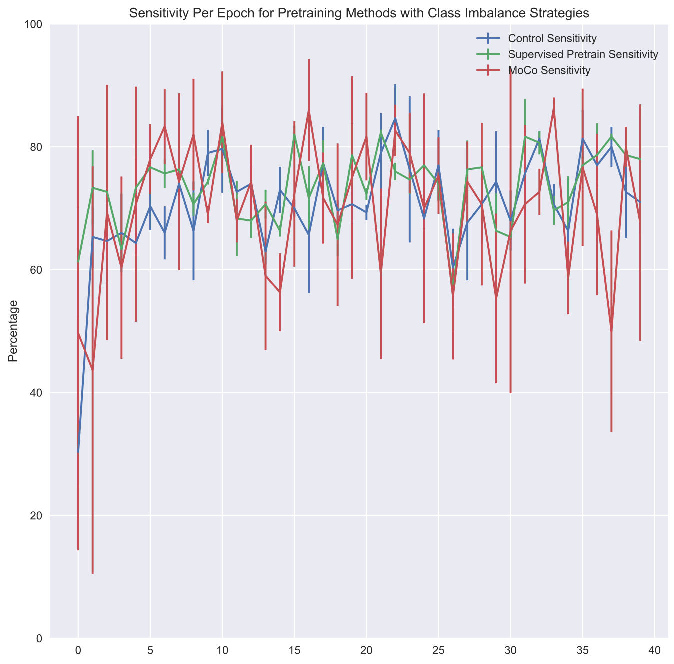


Figure 29: Pretraining Sensitivity Per Epoch, Classic Balancing

Once again, a major challenge is deciding how to evaluate the performance of the MoCo architecture, given the variance. The network was able to perform with average results ranging from $\sim 45\%$ up to $\sim 88\%$. MoCo presents with a sensitivity standard deviation of almost 20%, with a ppv standard devation of 30%. With such huge deviations in performance, it continues to bring into question the validity of any conclusions that may be drawn from these findings.

Method	Accuracy (%)	Sensitivity (%)	PPV (%)
Control	88	38	**97**
Data Augmentation	**92**	51	96
Upsampling + LF weight	91	71	84
Supervised Pretraining with Upsampling + LF weight	80	**78**	55
MoCo with Upsampling + LF weight	88	68	72

Table 2: ResNet50 Methods Results Comparison

8 Discussion

For many tasks, maximising the total validation accuracy will be of upmost importance. For these tasks, it is reasonable to claim that VaB-AL is the most effective method for improving minority class performance. This is due to the method's ability to increase the accuracy of all classes, while primarily impacting the minority class sensitivity. However Covid-19 prediction, and other medical diagnosis tasks, are somewhat unique in that it is very important to have low false negative rates, and thus a high sensitivity. This is because it is not too impactful if a healthy person has to undergo further testing because the classifier flagged their X-ray as Covid-19 positive. However, the consequences of a false negative classification could lead to an infected individual not quarantining and further spreading the disease, as well as putting themselves at risk by not being monitored closely during the development of the illness. As such, it could be argued that the reduction in overall accuracy is a worthwhile trade-off for maximising the sensitivity in this specific domain. In this scenario, the traditional methods would be more effective, especially when used on top of a pretrained network.

Another significant benefit to using traditional methods is that they may be implemented trivially using modern libraries, which is vital if there are tight time constraints on a project. However, while VaB-AL may take much longer to train and implement, the benefit is that due to the dynamic and learned-nature of the weights, there is no need to conduct a grid search to optimise the weight values, whereas they must be found empirically for traditional methods.

VaB-AL shares a similar issue in that it also requires empirical optimisation; the variance in the results poses a significant barrier to forming conclusive statements as to the efficacy of each approach. For example, VaB-AL was able to achieve Covid-19 sensitivity scores of up to 73%, however the average was almost 20% below this figure. If there were no time constraints on the study, obtaining optimal hyper-parameter values and training times would likely lead to a significant improvement in average performance. These changes would also help indicate whether the increase in overall accuracy was due to variance, or whether the thesis actually does help majority class performance too. Despite these limitations, it is clear that VaB-AL does provide a means to autonomously assign dynamic, per-sample weights to an entire dataset, and as a result yield a significant increase in minority class performance.

One potential improvement could be scaling the scores once they are predicted. By raising the weights to a power, the distance between them would grow. As such, the minority class could be sampled more by applying an exponent greater than 1 to the weights, causing the larger weights to increase much more than the smaller weights. Similarly, if the network is overfitting to a subset of the data due to the weights, an exponent less than 1 would help the weights to converge.

As discussed before, VaB-AL struggles to function with larger input vectors. It is necessary to develop better generative models that can handle larger reconstruction tasks in order for VaB-AL to find consistent use with the strategies presented in this paper. This is because performance will drop when the data samples and classifiers grow in size and depth. The constraint to only use small classifiers and data sample dimensions prevents the strategy from being implemented with many new techniques, including the momentum contrast learning work explored in this paper, as the classifier for that is a ResNet50 with 224x224 input.

Supervised pretraining was shown to improve minority class classification when combined with upsampling and loss function weighting. In contrast, momentum contrast learning actually finished training worse than the original network. However, it is very likely that the momentum contrast network is actually much better, as for the majority of training it outperformed the other networks, and its final result is uncharacteristically poor. It could be argued that an earlier reading for MoCo should be used in evaluation, as the network gets significantly worse at the final epoch. However, for the sake of academic integrity, it was decided that transparency about these issues was a better approach, while still emphasising the strength of MoCo during much of the training.

One possible reason for the large variance in results for certain methods (specifically MoCo) could be that training was not conducted for enough epochs, and so the network never arrived at its final, stable fitness. This could also explain the large variance between runs.

An interesting observation about the metrics was that sensitivity and positive predictive value appeared to have an antagonistic relationship. That is, as the Covid-19 sensitivity increased, the positive predictive value would in general be diminished. This makes sense, as sensitivity requires the network to predict Covid-19 more commonly, leading to increased risk of misclassifying a sample as Covid-19 - reducing the positive predictive value. A potential area for further study would be exploring how these values could be optimised in tandem, as a multi-objective problem, instead of the single-objective function that is used in this study.

To summarize, VaB-AL is a very promising technique that improves the minority class performance, while minimising reduction in positive predictive value and actually improving the overall accuracy. Pretraining also improves network performance, however further work is necessary to understand the limits of the momentum contrast learning paradigm, and to stabilise its learning process.

There are several avenues for future work. The most obvious next step would be to integrate momentum contrast learning and VaB-AL, to explore whether the combination of strategies can further improve performance. Furthermore, the VaB-AL weights may help alleviate the erratic nature of the MoCo training, by converging on the hard-to-classify samples. This is subject to solving the issues with large input vectors discussed above.

Another potential avenue is data synthesis. As generative models are capable of creating data, it would be interesting to explore whether they could generate novel data samples from the minority class, expanding the available data pool and mitigating class imbalance. While methods exist that are similar, they are only effective for very low dimensional problems and thus are not viable for image processing tasks such as this.

9 Conclusion

Class imbalance causes significant performance reduction in Covid-19 prediction tasks. This study utilised a residual convolutional neural network to classify chest X-ray images. The work illustrated that there are a variety of techniques, some known and some novel, that can alleviate the impact of this imbalance. While upsampling and loss function weighting are effective tools for improving network performance on minority class samples, they lead to a reduction in overall network performance. As such, Variational Bayes' for Active Learning is the more promising technique in the long term, as it realises minority-class improvements while also increasing the overall accuracy. Applying the VaB-AL framework as a means for generating dynamic, per-sample upsampling and loss function weights is the primary novel contribution of this paper. Furthermore, while the significance of pretraining is less clear than that of VaB-AL, there are strong indications that using a pretrained network will also improve minority class performance. Further work is needed to identify the limits of these approaches, however this study has demonstrated their potential to improve the classification capability of deep residual networks in Covid-19 prediction tasks on a heavily imbalanced dataset.

10 Submitted Code Appendix

10.1 Code Structure and Execution

Code for this project can be found here: https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2019/jxf908

Firstly, the data must be downloaded from the github (https://github.com/lindawangg/COVID-Net). Inside this GitHub, there are instructions on how to compile the data, as well as a script that automates the scraping of said data from its various sources. The script will also generate two text files, containing the metadata of each sample (such as its label, file name, etc). Once this data is acquired, it must be split into the hierarchy discussed in the Analysis section of this report. This is to facilitate the labelling functionality of PyTorch, and it can be achieved by running the sort.py function that I created, found within dataset_building_functions. Sort.py relies upon the two text files that the original script generates. All methods employed assume this file structure, and assume that the root of the data is /data/.

The code for VaB-AL runs can be found inside the VaBAL subdirectory. To run the code, simply run the main.py script, with –debug for verbose mode. There are many command line arguments that can be used to alter hyperparameters. The full list can be viewed at the top of the main.py script.

The CustomDataLoader.py, dataset_load.py files are used to parse and load the dataset. CustomDataLoader.py requires the presence of the two text files generated by the dataset compilation script, in order to apply the weights dynamically. The models, resnet and vae, are stored inside the models subdirectory of VaB-AL, and the scripts save logs to the logs subdirectory, as well as saving the networks to saved_nets.

The subdirectories VaBAL_Extended and VaBAL_Subset_Methodology are no longer used, the former is an implementation that is more heavily based off pre-existing code, but thus is less streamlined for the task at hand. It is preserved only as a sanity check, should the newer code encounter issues. The Subset folder implements the unsuccessful subset methodology discussed in this report, it is not currently used.

The Early Investigation Work folder features logs, presentations and notes - alongside obsolete networks - from early in the project. None of this is used in the final implementation, but they are kept for posterity and for reference, should they be needed later.

Results simply contains all of the results, saved networks, graphs, and graph making scripts that were used for the report.

Finally, the Code directory contains code to execute the upsampling and loss function weighting network, as well as MoCo. To run the former, simply navigate to the Upsampling and Ordinary ResNet file, and run the CovidResNet.py file. There are variables within the code that can be changed to alter the specific weightings. This file also runs the supervised pretraining, this is accomplished by changing the ”pretrain” variable to true, and uncommenting the ResNet50 code, while commenting out the ResNet18 code.

MoCo can be run from the MoCo directory. It requires the pretrained network to be downloaded, and it is run with the following command:
py ./main_lincls.py -a resnet50 –lr 30 –batch-size 8 –pretrained ./moco_v2_800ep_pretrain.pth.tar –dist-url ’tcp://localhost:10001’ –world-size 1 –rank 0 –epoch 40 ../../../data/

Much of this line is self explanatory, or should not be changed. Specific details of each command-line parameter can be found here: https://github.com/facebookresearch/moco . The first term is the file to run, the variable after –pretrained is the location of the pretrained network, and the file variable is the path to the data root.

10.2 Code Attribution

Firstly, the ResNet18 architecture used was taken from the original VaB-AL code produced for paper [1]. This was taken as there was little point reinventing the wheel, and ResNet implementations are already standardised.

The dataset building functions are adapted by me, but originally written by the dataset authors and can be found here: https://github.com/lindawangg/COVID-Net . The exception is the affine_transform_data_generation.ipynb and the sort.py files.

The CovidResNet.py file, for the control run, supervised pretraining, and upsampling/data augmentation/ loss function weighting is written by myself.

Due to the nature of state dictionaries, most of the MoCo code is taken https://github.com/facebookresearch/moco . It is adapted from there to suit the needs of this project, but it is primarily provided by the authors at Facebook.

The Code found within VaBAL is mixed. While I used the code provided by my supervisors for their paper [1] as a reference point and for guidance, everything was reimplemented from scratch, using parts of their code where it made sense, but it is streamlined and restructured to accommodate the many changes made by myself. The exception is the code within VaBAL_Extended which, as mentioned, is not currently used but does feature substantially more code from the original authors.

References

[1] J. Choi, K. M. Yi, J. Kim, J. Choo, B. Kim, J.-Y. Chang, Y. Gwon, and H. J. Chang, "Vab-al: Incorporating class imbalance and difficulty with variational bayes for active learning," 2020.

[2] C. P. West, V. M. Montori, and P. Sampathkumar, "Covid-19 testing: the threat of false-negative results," in *Mayo Clinic Proceedings*, Elsevier, 2020.

[3] L. Wang and A. Wong, "Covid-net: A tailored deep convolutional neural network design for detection of covid-19 cases from chest x-ray images," *arXiv preprint arXiv:2003.09871*, 2020.

[4] L. Li, L. Qin, Z. Xu, Y. Yin, and B. K. J. B. e. a. Wang, Xin, "Artificial intelligence distinguishes covid-19 from community acquired pneumonia on chest ct.," *Radiology*, vol. 200905, 2020.

[5] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *SIGKDD Explor. Newsl.*, vol. 6, p. 20–29, June 2004.

[6] A. Luque, A. Carrasco, A. Martín, and A. de las Heras, "The impact of class imbalance in classification performance metrics based on the binary confusion matrix," *Pattern Recognition*, vol. 91, pp. 216 – 231, 2019.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[10] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013.

[11] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[12] A. Amini and A. Soleimany, "Deep generative modeling — mit 6.s191: Introduction to deep learning."

[13] D. Erhan, A. Courville, Y. Bengio, and P. Vincent, "Why does unsupervised pre-training help deep learning?," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 201–208, 2010.

[14] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," 2019.

[15] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, pp. 1735–1742, IEEE, 2006.

[16] A. van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," 2018.

[17] G. D. Rubin, C. J. Ryerson, L. B. Haramati, N. Sverzellati, J. P. Kanne, S. Raoof, N. W. Schluger, A. Volpi, J.-J. Yim, I. B. K. Martin, D. J. Anderson, C. Kong, T. Altes, A. Bush, S. R. Desai, o. Goldin, J. M. Goo, M. Humbert, Y. Inoue, H.-U. Kauczor, F. Luo, P. J. Mazzone, M. Prokop, M. Remy-Jardin, L. Richeldi, C. M. Schaefer-Prokop, N. Tomiyama, A. U. Wells, and A. N. Leung, "The role of chest imaging in patient management during the covid-19 pandemic: A multinational consensus statement from the fleischner society," *Radiology*, vol. 296, no. 1, pp. 172–180, 2020. PMID: 32255413.

[18] J. P. Cohen, P. Morrison, L. Dao, K. Roth, T. Q. Duong, and M. Ghassemi, "Covid-19 image data collection: Prospective predictions are the future," *arXiv 2006.11988*, 2020.

[19] M. Chowdhury, T. Rahman, A. Khandakar, M. K. R. Mazhar, Z. Mahbub, K. Islam, M. Khan, A. Iqbal, N. Al-Emadi, M. Reaz, and M. T. Islam, "Can ai help in screening viral and covid-19 pneumonia?," *IEEE Access*, vol. 8, pp. 132665–132676.

[20] L. Wang, A. Wong, Z. Q. Lin, P. McInnis, A. Chung, H. Gunraj, J. Lee, M. Ross, B. VanBerlo, A. Ebadi, K.-A. Git, and A. Al-haimi, "Figure 1 covid-19 chest x-ray dataset initiative."

[21] L. Wang, A. Wong, Z. Q. Lin, P. McInnis, A. Chung, H. Gunraj, J. Lee, M. Ross, B. VanBerlo, A. Ebadi, K.-A. Git, and A. Al-haimi, "Covid-19 patient cases from the actualmed covid-19 chest x-ray dataset initiative."

[22] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. Summers, "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," in *2017 IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*, pp. 3462–3471, 2017.

[23] P. Dwivedi, "Understanding and coding a resnet in keras." towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[25] S. Ruder, "An overview of gradient descent optimization algorithms," 2016.

[26] A. Mikołajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," in *2018 international interdisciplinary PhD workshop (IIPhDW)*, pp. 117–122, IEEE, 2018.

[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[28] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, p. 40–53, Mar. 2008.