Task 1

1.1 | General Task

Our objective was to identify the most suitable model for explaining the relationship between the 'x' and 'y' values within a dataset comprised of 15 data points. Given the limited dataset size, it's crucial to employ appropriate methods to find the best model and achieve the lowest **Sum of Squared Errors** (SSE) with the test dataset. To accomplish this, we have access to a range of techniques, including standard approaches like **lasso**, **ridge**, and **elastic net**, as well as more unique methods such as **nonlinear models**. Our aim is to select the most appropriate techniques to uncover a model that minimizes the SSE during training while also being confident that it will perform well in minimizing SSE when applied to the testing dataset. There were multiple optimization strategies discussed early on: the data provided was scarce, and the number of features was high. This meant that not a lot of trust should be put in the model's performance, considering it was trained with the entire data, as it could be **overfitting** said data. Trying to compute the model's **risk function** was deliberated, but given the nature of the task, no information about the probability distribution of the data was given, so it would require plenty of assumptions being made to employ said function, which would add volatility to the model's structure, so that idea was discarded. Due to very similar reasons, the use of **nonlinear regression** was also an idea that was discarded, for it would be responsible for greatly overfitting the developed model, a consequence that we very much mean to diminish as much as possible. To avoid this data fragility, there was a need to establish a **validation step** in the model's development, that would employ an accuracy scoring that would much more accurately represent the model's success in the test set's prediction. This method is also known as **cross-validation**, and it will also be a key factor in determining the best **hyperparameter** for a regularization adjustment of the model. Finally, a careful analysis of the data provided was made, to try to find any inconsistent or redundant factors present that would then in turn provide a great boost in accuracy were they taken care of.

1.2 | Cross-Validation and Hyper-Parameter Determination

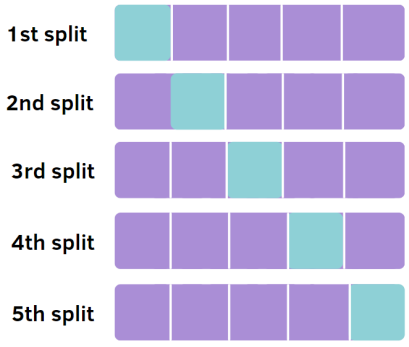

Figure 1 - Leave-One-Out Structure

To select the most suitable model for our data, we need to discuss how we evaluate a model's performance.
One common way is to use the Sum of Squared Errors (SSE) to gauge how closely our model aligns with the data points. However, relying solely on SSE can lead to problems like overfitting, where a model fits the current dataset perfectly but struggles with new data.
If we were to train and test the model using the same 15 data points, we wouldn't get meaningful results. Unfortunately, we lack additional data points for testing. To overcome this limitation, we employ a technique called **cross-validation**.
Cross-validation doesn't train the model on the entire dataset. Instead, it **divides** the data into multiple subsets. The model is trained on one subset and tested on the remaining portions. This process is repeated several times with different splits, and the performance metrics are averaged.

In essence, cross-validation provides a more comprehensive assessment of a model's potential performance on unseen data. This makes it a valuable tool, especially when fine-tuning hyper-parameters, a topic we'll explore later.
We primarily used 1-fold or 2-fold cross-validation in our testing. This choice was made because when you use a higher fold (such as 3-fold or more), you end up using less data to train the model. This reduction in training data can make any conclusions drawn from the model less reliable. Given the limited amount of data available to us, we opted for smaller folds to ensure that the conclusions we reach remain reliable and robust.

1.3 | Model Analysis

The following linear regression-based models were studied and applied throughout the development of this task, they all possess a generalization coefficient designed to prevent any kind of overfit:

→ Ridge - an extremely robust and **reliable** regression model that performs with little to no difference with untrained data. Any optimal hyper-parameter shift does not greatly affect its accuracy level, and thus it is the perfect choice for this problem's nature. The only downside is that, before feature removal, it gets beat by the models below in minimum SSE score.

→ Lasso - a model that replaces the coefficients of redundant features to 0, performing its own form of **feature removal**. However, this model is extremely fragile to any variation.

→ Elastic Net - A blend between both of the above models. Unfortunately, this problem displays the worst of both models, showing evident instability and sub-optimal minimum SSE values.

→ Others like **LassoLars**, and regression models with built-in cross-validation like **RidgeCV**, **LassoCV**, and **ElasticNetCV** were developed, but were deemed not apt for the nature of this introductory task (they were also not mentioned by the teaching staff).

Models	SSE	SSE (feature removal)
Ridge	3.15431880440805	1.7426388792057521
Lasso	2.9310643553540268	1.8213664551860245
ElasticNet	3.0568127803023972	1.838665094903284
Linear Regression	7.864171987128417	1.8386650963553768

Table 1 - Model Performance Display

It is noted that due to the removal of redundant features in our data, for every model tested besides ridge regression, the optimal hyperparameter corresponded to near zero, whilst still displaying worse SSE values overall. With this, the optimal pick for ridge regression was evident, as it also ensures a high degree of **stability** in the developed model.

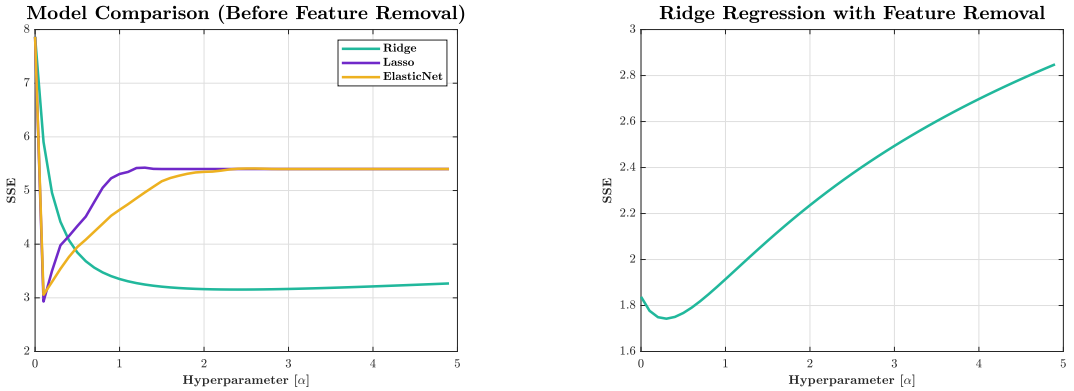

Figure 2 - Optimal Hyperparamter Analysis and Model Structure

Through Lasso observation, features 1, 5, 8, and 9 were revealed to be redundant. The additional removal of another two features, although not flagged by Lasso, effectively diminished SSE. However, due to this being only a slight change in SSE, and its removal being rather **risky**, those two features were opted to be kept in our data. It is worth noting that this implementation might have been what was missing for a number one spot on the leaderboard, but due to the importance of avoiding any kind of overfitting, especially due to the specific circumstances of the data provided, our choice still felt like the right one to make.

Task 2

2.1 | General Task

We were provided with a dataset comprising points sampled from two distinct linear regression models. Our challenge was to accurately classify each point according to the regression model it was derived from. Following the classification, our goal was to determine the coefficients of the regression lines in such a manner that, upon testing these models, the sum of squared errors (SSE) would be minimized.
To categorize the data points, our initial approach involved employing the **RANSAC** algorithm. This iterative technique is particularly effective for fitting models when outliers are present. In our case, we treated the second model as an outlier to obtain an approximate representation of the first model. However, a significant issue arose during intersections, as most of the data points were being assigned to a model, rather than being evenly distributed between both models. This problem was exacerbated by the fact that certain dimensions, such as the **1st** and **4th**, appeared to exhibit nearly parallel behavior, making their separation challenging, as can be seen in Figure 3.

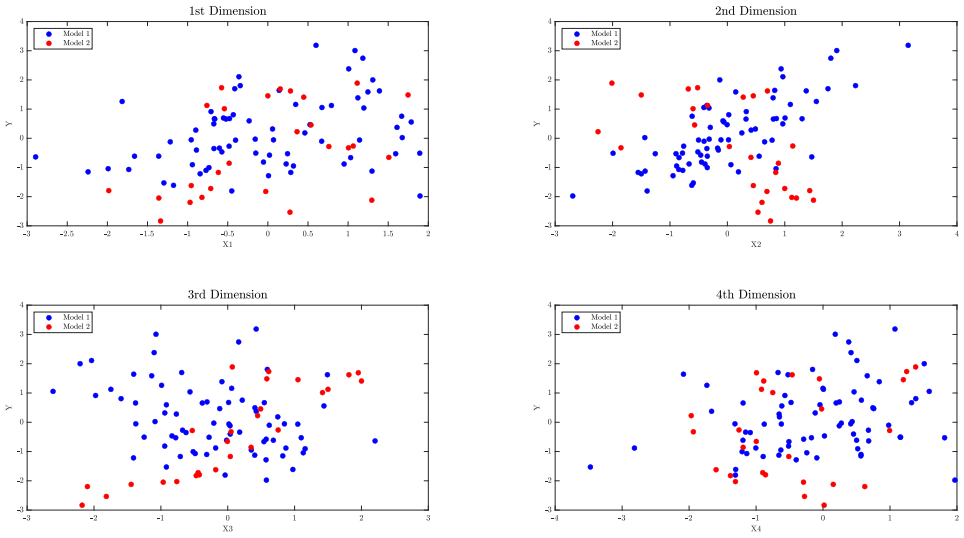

Figure 3 - Representation of the data points for each model using RANSAC

To address this challenge, we employed a **Gaussian Mixture** model. This statistical approach is employed to characterize the probability distribution within a dataset, particularly when the underlying data is suspected to arise from a combination of multiple Gaussian distributions. While this method successfully resolved the interception issue, it led to a less accurate approximation. This was due to the fact that, instead of optimizing for a linear fit, we were seeking a Gaussian distribution that was highly stretched, resulting in a less precise outcome. On the upper side, the lines in Figure 4 are clearly visible.

2.2 | Result Evaluation

After discussing the techniques employed to assess the regressions, we will delve into the testing procedures for both methods and the subsequent outcomes achieved. Having divided our data into two distinct sets through one of the methods, we used the techniques explained in Task 1 to determine which type of regression to use. Initially, we employed Lasso Regression to Model

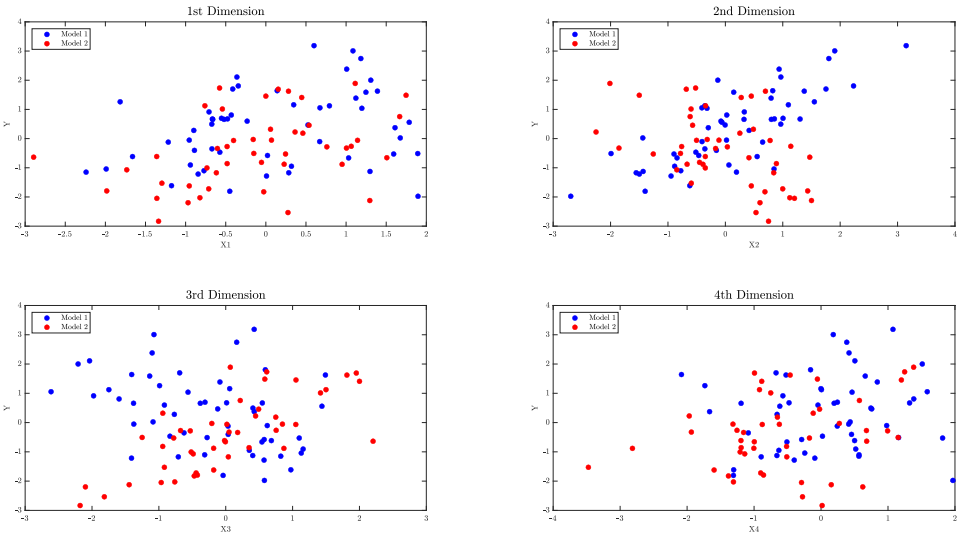

Figure 4 - Representation of the data points for each model using GM

1 to assess the presence of removable features. Upon confirming the absence of such features, we opted for **Ridge Regression** and utilized cross-validation leave-one-out to compute the optimal alpha value. In the case of Model 2, a similar fitting was performed as in Model 1. However, the alpha values approached zero, leading us to conclude that **Linear Regression** was the most suitable model. Subsequently, our focus shifted towards enhancing accuracy and minimizing errors. To achieve this goal, our initial step involved visualizing the residuals to identify misclassified data points.

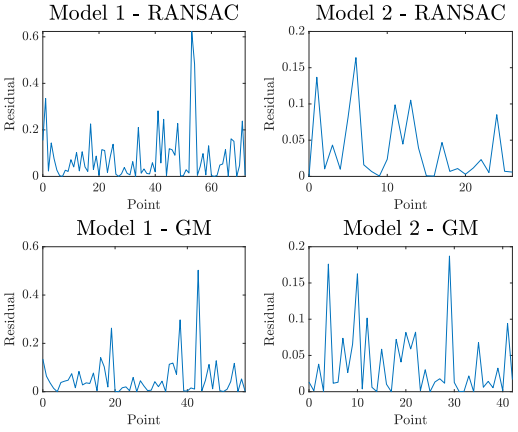

Figure 5 - SSE per point

In our effort to enhance each method, we found that the misclassified points were not high residual points but rather points mixed in with noise (interception). This made it challenging to identify the right model for each point.
After careful examination, we also concluded that the points with high Sum of Squared Errors (SSE) were not misclassified, but rather potentially noisy data. Consequently, we made the decision to remove these high SSE points. However, it's worth noting that this choice carried some **risk**, as there was a possibility that these points may not have been as noisy as initially assumed. The final results were:

Method	Models	SSE	SSE (point removal)
RANSAC (Ridge)	1	0.08138965626665125	—
RANSAC (Linear)	2	0.03661800871987473	—
ML (Ridge)	1	0.0563490119433999	0.0361434407340808
ML (Linear)	2	0.037996251521764335	—

Table 2 - Methods Performance Display

Task 3

3.1 | General Task

In this task, we got a set of 28x28 pictures in RGB with values ranging from 0 to 255, and our job was to determine whether each picture displayed the skin disease nevus or melanoma. The main challenge here is that both diseases possess extremely similar features in their images, being very hard to differentiate, and the data we do have is unevenly split. We can't obtain additional images, and the dataset consists of only 14% melanoma cases. In the following section, we will discuss how we handled the data and then explain the techniques we used to categorize the pictures, which consisted of multiple diverse forms of **neural networks** and **SVM image classification**.

3.2 | Data Treatment

Before discussing different data treatments, it's important to state that the training and validation data were separated first. This means that the augmentation steps to be described were applied to training data **only**, otherwise, there would be tampering with the validation.
Our initial expectations of the data provided suspected the existence of outliers in its midst, and due to their extreme similarity, this factor remained somewhat unclear. However, trust was imposed that no image was labeled incorrectly, for it would induce significant problems. With this, the underlying issue with the provided data is its **imbalance**, thus it was crucial that this data was correctly augmented. We explored various solutions to tackle this problem, including:

→ Oversampling: Involved collecting additional melanoma images until there were an equal number of images for each category, offering great benefit in increasing its total size.

a) **SMOTE**: We tried using this method, but the generated images didn't meet our expectations, so we eventually stopped using it. SMOTE is an extremely useful tool for regular data augmentation, but it's very impractical for image classification (e.g. mixes images, blurs them, and causes all kinds of unnecessary changes).

b) **Manual Image Transformations**: This technique involved several factors, with a key focus on rotating the dataset using fixed or random angles until it was balanced. We primarily relied on this approach for the task and noted that a more "human-assisted" augmentation beats unsupervised functions like SMOTE, especially when dealing with images.

c) **External Images**: We attempted to resize the HAM10000 dataset to 28x28 by reducing resolution. We observed that most pictures were similar to the dermaMNIST dataset. However, the use of external images was prohibited.

→ Class Weights: We tried changing the learning rates for melanoma and nevus, but the outcomes were inconsistent.

→ Undersampling: This approach was employed with a custom SVM model to cope with longer processing times. It involved deleting nevu images until melanomas were of equal size. However this was considered the most dangerous method of the three, due to an active removal of provided data, and it accurately resulted in poorer results.

Some techniques we also used were to stratify and to shuffle the data. This ensures that the validation set mirrors the data arrangement in the original training data. Additionally, the newly created images at the dataset's end are randomized, improving the model training performance. When applying transfer learning, it was crucial to ensure that the data met the model's requirements. We experimented with different models: some required 32x32 images, so we added black padding to make 28x28 images fit, while others models needed larger images, so we up-scaled the image accordingly.

3.3 | Methods Used

Starting with our worst performing model, the **Support Vector Classification** (SVC). We explored various kernels and adjustable settings, but the highest balanced accuracy we achieved was around 70%. Since the available kernels didn't suit our image classification needs, we developed a custom kernel. This new approach utilized pixel histograms and color differences to classify images. Although training this model took hours (which led us to use undersampling), we managed to achieve nearly **75% balanced accuracy** on the validation set.
The use of SVM still naturally was expected to underperform compared to a CNN, especially with image classification (SVM's do not take into account multiple useful image attributes and features, that a CNN, for example, would employ).
Convolutional Neural Networks are extremely optimized for image classification, especially because it is a model that is much more "tunable" than every other one tested before. This practicality allowed the manual assembling of a neural network architecture that is built and specialized around our specific problem, and thus it was from this principle that our very best model originated. Through rigorous testing, a significant benefit was definitely seen in the existence of a **convolutional section** of a high degree of complexity (network structures whose convolutional impact was very small, i.e. a simple network, underfits with the data provided and reflects poorly with a balanced accuracy of around 50-60%). Of course, an extremely complex feature detection section also may put an extremely heavy strain on the network's computation (would also likely overfit the network), so a middle-ground is the goal to achieve. The best convolutional section tested consisted of three straight convolutional layers with a relatively high number of filters (32), that act as the **main feature detector**. Understandably this heavily increases our feature size, and thus the existence of max pooling followed by dropout layers are implemented after subsequent convolutional layers to end up with only the most notable features in the provided dataset. This feature learning process is key to the model's optimal performance, especially considering the nature of our problem (the two classes are extremely similar, so the network needs a complex feature detection section).
The **MLP section** of the CNN needed a compact 64-size dense layer followed by a batch normalization (due to our high batch size) and an extremely high dropout layer (0.7). This dropout aided in generalization and preventing overfitting, despite 0.7 being a high dropout value, it still delivered strong results. It is important to note that this decision process of architecture was heavily aided by hyperparameter tuning with **Hyperas**, which allowed for continuous evaluation of multiple variations (e.g. filter size, dropout values, dense layer outputs) in the developed CNN. A high batch size ensured needed stability in the training process and a high number of epochs with a **callback method** played important parts in obtaining the optimal results. Throughout the development of this CNN, a wide array of balanced accuracy scores were obtained, ranging from **70%** in our early versions to our highest metric of **81%** after our mindful optimization. Due to it being fine-tuned to specifically tackle our problem, it is no surprise that this model ranked the highest in the ones developed.
The method we got our second to best result was while using **Transfer Learning**. As our selection of images is somewhat limited, the idea was to obtain a network that was already trained to get specific features from the image and then we would make our own classification layers in order to obtain the desired outputs. If the images had a ~1080p image resolution this method would be ideal because it would be a lot more efficient to obtain the most important feature from a transfer learning model than from a CNN done and trained with only 8000 images. And if the resolution was actually that big we would need an even bigger training set to train the convolution layers. However, because the images were 28 per 28, the feature detection models struggled to obtain good enough information for our classification layers. Also, this low resolution made training the network much easier.
Initially, we employed **VGG16** as our transfer learning model. It operated on images as small as 32x32 pixels. Due to its simplicity at this scale, it achieved a balanced accuracy of **72%** with just 2 classification layers, sized 256 and 2. Later, we switched to **Inception V3**, which required larger 150x150 pixel images. Despite the low resolution of the upscaled images, this model outperformed others. For this model, we utilized a 1024 dense layer, a 0.2 dropout layer,

and 2 dense layers for the final softmax classification. In the best run, we obtained a balanced accuracy of **76%**.
This transfer learning analysis served to show that the size of the images provided was not adequate for any transfer learning model, at least in regards to obtaining the best possible result. The impact of feature detection displayed due to image resizing is simply too significant to ignore.

3.4 | Results Discussing

The optimal model described earlier was submitted to place in **20th** place, with an **81%** score (as noted in our testing), and whose results are displayed below:

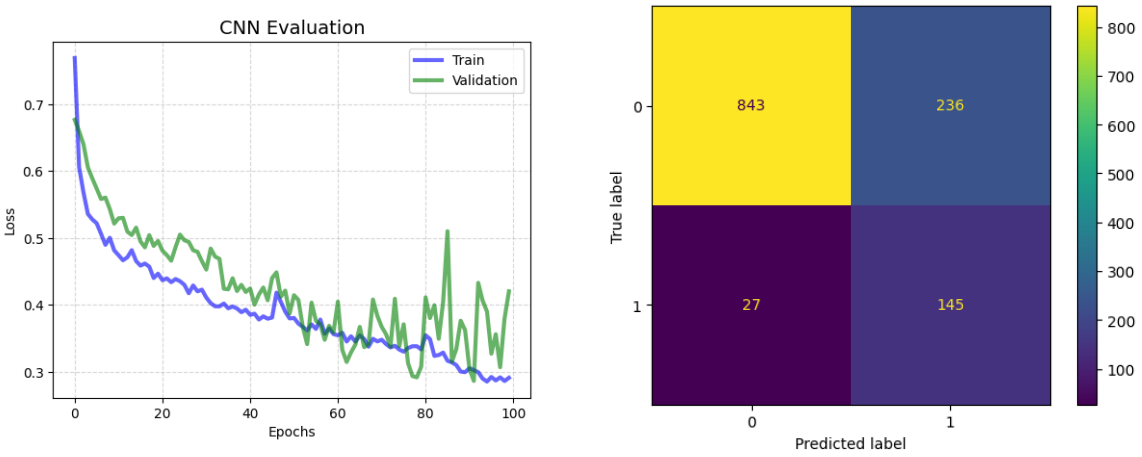

Figure 6 - CNN Training Evaluation and Confusion Matrix

Where a number of notable aspects can be pointed out. The **confusion matrix** provides a very good balance in class accuracy, with 78% accuracy for label 0 and 84% accuracy for label 1 (joint **81%** balanced accuracy). This is an extreme improvement over earlier simple models that prioritized class 0 predictions and ended up guessing most of class 1's images (the weaker class in terms of sample numbers) wrong.
The **CNN Evaluation graph** rightly displays the challenge in training image classification with similar-looking classes, as evidenced by the very unstable validation loss values, even when at a very high batch size (in the next task, which adds classes that are much more distinct between each other, will the real difference in loss fluctuation be understood). However, no plateau is observable, even at these low validation loss values, which shows the capable model's performance, even in this tough environment.
Finally, it is worth noting that the developed model performed reasonably well when classifying data from a different hospital (beating most of the groups above us in ranking), which ensured us that the developed model was a great classifier in general, and not simply a product of overfitting.
In summary, we examined several unique approaches to achieve optimal classification outcomes. This helped us realize that techniques such as SVM can yield good results when compared to well-known methods like neural networks. We also reinforced the significance of proper data handling and its impact when training a model. These data constraints encouraged us to explore new techniques, such as varying learning rates for each layer and applying regularization, which deepened our understanding of this field.

Task 4

4.1 | General Task

This final task involves an image classification problem quite similar to the previous one, however this time we are in the presence of **six different classes**, where half pertains to a different dataset. This last bit of information is important because we are given direct intel that encourages splitting the dataset for specificity in optimization.
The two classes of the last task are also present here, and this will be key information that will have a significant impact on the model development because many of the lessons learned in the earlier task are fit to also be applied in this one. Namely, the diligently developed architecture to detect the features of these demonstrably similar classes.

4.2 | Provided Data Analysis

It is very important to mention what was discovered about the provided dataset. It is composed of two subsets: one of three **skin diseases**, and the other of three **blood diseases**. It is apparent how easy it is to differentiate between these two subsets, which will in turn become an important factor in one of the models we developed. Another key attribute of this dataset consists in the relative ease of classifying the classes belonging to the blood disease subset, which can be noted in the **Confusion Matrix** of the six class classifications utilizing a **single CNN**, of similar architecture to the one on the previous task:

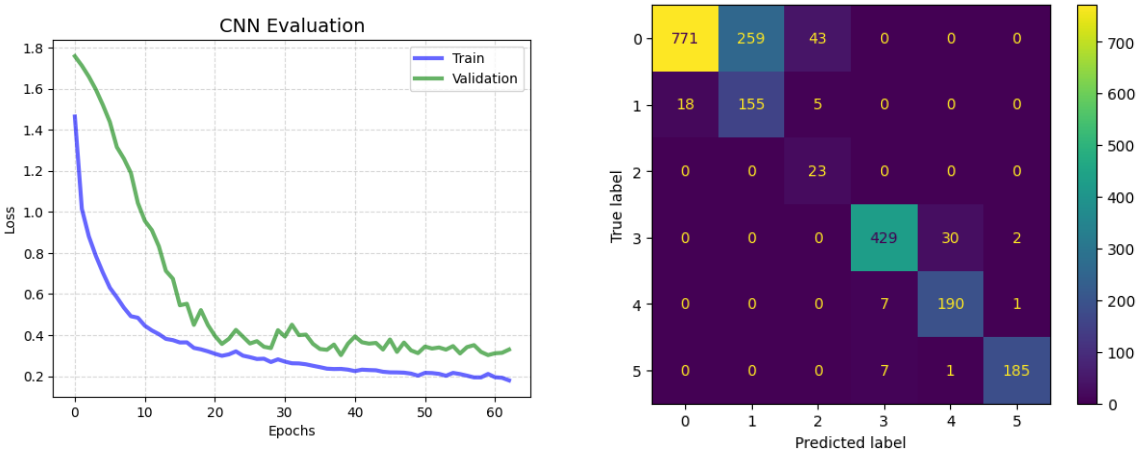

Figure 7 - CNN Training Evaluation and Confusion Matrix

Where a lot of interesting conclusions can be made of the dataset:

→ Classes **0** and **1** (the same classes as the previous task) show the most classification difficulty. This an **extremely** important detail, because it dictates that these two classes will be the biggest contributing force for the model's overall optimal performance, making it very useful the fact that a deliberate fine-tuning of a convolutional section prioritized to detect features for both of these classes was already implemented in the previous task and is seen in full effect here.

→ There is no **cross-error**, i.e. there is absolutely no misclassification of one subset to the other, which is another factor encouraging the dataset separation to be discussed later.

→ The blood disease subset presents near-perfect results, which confirm our theory about ease of classification for those classes, and further imply the sheer **importance** of making sure that the first two classes have the least amount of misclassification error possible.

4.3 | Employed Methods

All of the data augmentation methods tested in the previous task were also experimented with this new dataset, and to no surprise the very same method as before (manually causing **non-image damaging alterations** like random rotations and mirroring) proved to significantly yield the best results. This is mainly because the principles of this new dataset are still extremely similar to the one in the earlier task. The high batch size combined with high epoch training along with an early callback also proved extremely beneficial in this new dataset, seeing as how it's even bigger in sample size after data augmentation than the previous task.
As stated before, there is a great benefit in dividing the dataset into two subsets. Thus we developed a **Dual Branch CNN** (as seen on the left-hand side in **Figure 8**), that is composed of three CNNs: one that performs a classwise split, dividing the dataset into the two subsets, and two CNN's whose weights and structure are specialized to the specific assigned subset. This method for a dataset that was plagued with cross-errors would provide an extremely significant boost in classification accuracy, however, due to the nature of our dataset, it provided almost a similar performance to the admittedly already extremely optimized single CNN.

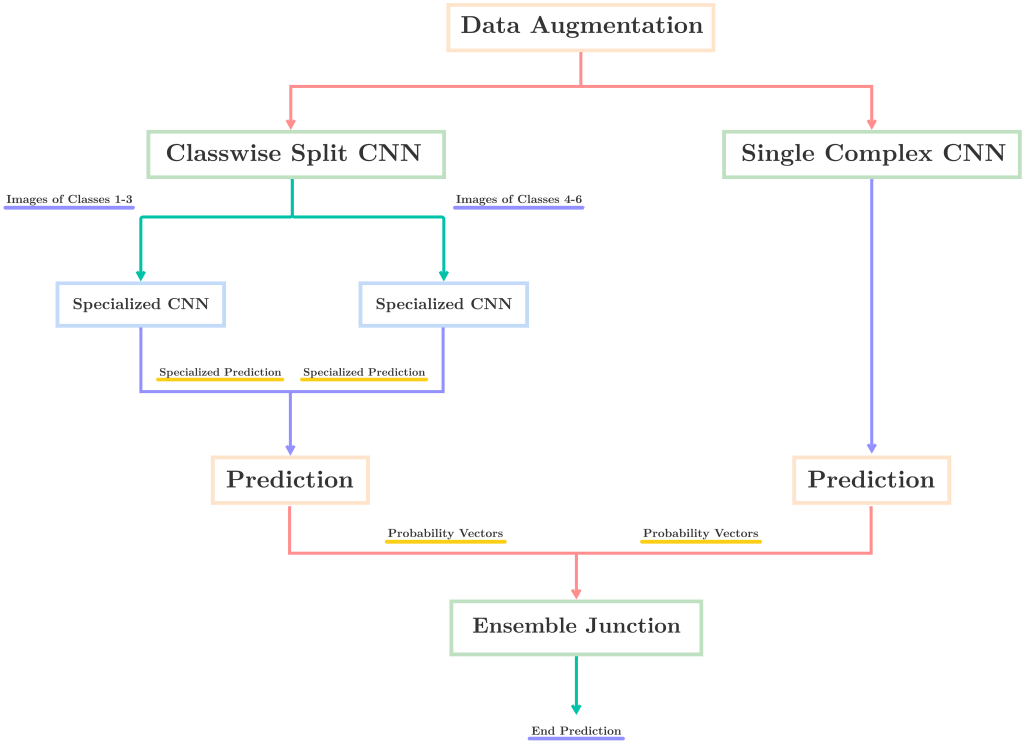

Figure 8 - Ensemble Structure Diagram

However this situation creates a really useful opportunity in having two different extremely robust and precise models that predict the same dataset, and have similar balanced accuracy, which is in diligently combing through the predicted results of both models, through its **probability vector** (an array that indicates per image the certainty for each class), and cross-analyze each certainty pertaining to each of the two models per image, and thus providing a step-up in the validity of our prediction. This is a high-end method known as an **Ensemble** (as described in **Figure 8**), and is essential to many complex classification tasks, and it just so happens to also probably be the best possible method for this problem. However, it is important to note that for final dense layers that use **softmax** activation functions, such as our case due to the multi-class classification property of this task, the probability vector doesn't exactly correlate certainty as the sigmoid does. But while it doesn't completely correlate, it is still related, this simply means that it is possible for instances where an assigned certainty in a probability may not correspond

exactly to a real one, but this is a compromise that still very well offers a tremendous boost in prediction secureness and precision.
When it came to the best optimal performance for this task, these three methods naturally ranked the highest. As explained earlier, due to the nature of the provided dataset (no cross-errors, and only two classes that were difficult to classify), both the single CNN and the dual branch one provided near-exact results (the single CNN squeaked out with just a few tenths in highest balanced accuracy). Evidently, the ensemble of these two models would surely improve performance, but unfortunately also not to a **significant** extent, again, due to the nature of this problem and both of the developed models, while equally excelling in classification accuracy, and offering distinct confusion matrixes, they still didn't provide unique enough differences to really cause an impact. Another reason is simply that the performances already achieved for this classification task are very near to reaching their **plateau**, so an ensemble of this form could very well provide a significant boost, were these two models instead giving 70-80 % balanced accuracy each. At the level of accuracy that each model currently lies on, even a high-end method like this ensemble can't really provide significant improvements, which is understandable.
There was however a diverse set of models also implemented and tested, which naturally ranked lower than the main ones described before, but certainly provided insight into each model's specific strengths and weaknesses. Both of the first models were previously implemented in the previous task, this time seeing fit the implementation of some form of **decision tree**, as it was a model suggested to us by the teaching staff.

→ Transfer Learning: unfortunately, much like in the earlier task, and seeing as though the more prevalent classes in this problem are the ones that were transferred from the previous, it is no surprise that transfer learning achieved almost the same results as last time. Its main weakness was still the fact that the images had to be resized to fit the minimum requirements, which provoked a significant **loss of feature clarity**. However, because of the quality of the convolutional section that the Inception V3 has for feature detection, it was still able to rank third in our model results round-up.

→ Support Vector Machine (SVM): this model suffers from a similar bane that the random forest has, which is they are **not specifically designed for image classification** (e.g. the images have to be turned into vectors for both of these models inputs), which in turn makes it so these models can't make use of crucial image information the higher ranking models do. However, through the use of a **custom kernel**, as explained in the previous section, the SVM still edges out the random forest with the highest balanced accuracy.

→ Random Forest (Decision Tree): this model consists of an **ensemble** of numerous decision trees, where in this case, decision trees, in theory, were believed to be reasonably apt for this task, due to their ease of interpretation and with dealing with unbalanced data. However, as evidenced by the results, this turned out to not be the case, mostly due to this being an image classification task, rather than a regular classification, which meant that a lot of the benefits of a decision tree were wasted. Specifically with its ease of interpretation, that falls short when all of the components of each tree are image features that are incomprehensible to any person. Sadly this problem simply isn't one that allows us to really delve into the best attributes of decision trees, because they are certainly an integral part of general machine learning problems.

Models	**Balanced Accuracy** [%]
Neural Network (CNN)	90.63 %
Dual Branch Neural Network	90.45 %
Transfer Learning	77.1 %
Support Vector Machine (SVM)	66.9 %
Random Forest	59.2 %

Table 3 - Model Performance Display