APPLIED MACHINE LEARNING SYSTEM ELEC0134 22/23 REPORT

SN: 23125632

ABSTRACT

We present a study on image classification techniques. The objective of this project is to evaluate and compare different approaches for image classification, from machine learning algorithms (Logistic Regression, Support Vector Machines, and Random Forest) to personalized state-of-the-art neural networks such as VGG16. We discuss the background of image classification. The training and testing are conducted on the datasets 'PneumoniaMNIST' and 'PathMNIST'. The results indicate that performance of machine learning algorithms on binary classification tasks is reasonably good, although they show some limitations. CNNs on the other hand are able to achieve great results with accuracy on multi-class classification algorithms at the cost of model complexity, however we consider this to be a reasonable trade-off given that the problem we are trying to solve requires good performance with low False Negative Rate. The link to access the code for the project is attached to this file.[1]

***Index Terms*—** Classification, Machine Learning, Neural Network, CNN, VGG16

1. INTRODUCTION

Classification is a fundamental task in computer vision that involves assigning a label to an input or separate into different classes based on its content and features. Over the years, researchers have developed numerous techniques to tackle this problem, ranging from classical methods to state-of-the-art deep learning approaches.

Medical imaging plays a key role in the study and diagnosis of diseases, which in turns results in large amounts of medical image data being generated every moment, which has to be manually analyzed and interpreted by medical professionals, which is subjective and time-consuming, and prone to human error. Recent advancements in machine learning and computer vision have allowed for the creation of models that can assist with the task of identifying the diseases present in a picture. Machine learning have revolutionized the medical imaging field by providing fast and mostly reliable analysis by using traditional machine learning algorithms or deep neural network approaches.

Traditional classification techniques use probability, statistics, and algorithms to identify patterns and relationships between the input and the corresponding class labels. They have been proven to be highly effective in performing classification tasks, although they may struggle with complex, high-dimensional data or when the underlying relationship between features and labels is nonlinear. Some of the most commonly used techniques for classification are Naïve Bayes, Logistic Regression, Support Vector Machines (SVMs), ensemble methods such as Random Forests among many others.

While traditional machine learning systems have been successful in many classification tasks, they often face limitations when handling high-dimensional data, such as images. This challenge led to the development of neural networks, particularly CNNs, which have revolutionized various domains including computer vision and natural language processing. Some state-of-the-art models that make use of CNNs are VGG16, AlexNet, ResNet…, which are able to classify images into thousands of classes with higher accuracy due to their ability of extracting relevant features.

On this project, we will implement different classification models and algorithms for two different tasks with different datasets each. We provide a literature review of the potential solutions for our problems and current approaches taken by researchers. In the next section we give a detailed description of the chosen algorithms and provide a justification for their choice. We describe our datasets and our implementation of the models as well as the training, validation, and testing process. Finally, we discuss the results of our project and give key takeaways.
A

2. LITERATURE SURVEY

Machine learning approaches are characterized by their simplicity yet powerful capabilities for image classification.

Naïve Bayes is a probabilistic classification method that is based on Bayes' theorem, which states that the conditional probability of an event A, based on the occurrence of another event B, is equal to the likelihood of the event B given A. It is expressed as in Eq. 1.

[1] The code is provided link-to-download-your-project.com and GitHub project: link-to-your-github-project

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

Equation 1. Bayes Theorem

Logistic Regression uses a logistic function called the sigmoid function to map the input variables to the desired output, which is a probability value between 0 and 1. The sigmoid function is can be expressed as Eq. 2. Logistic regression is limited to binary classification, but there exist techniques that allow it to perform multi-class classification such as OvR (One vs Rest), OvO (One vs One), in which a category is compared with all the other categories as a group and with all the categories. in which a category is compared with all the other categories as a group and with all the categories.

$$P = \frac{1}{(1 + e^{-z})}$$

Equation 2. Sigmoid function

Support Vector Machines (SVMs) focus on finding a hyperplane that separates datapoints from one class to the rest, maximizing the margin between each class. The margin is defined as the maximum distance between the datapoints of each class to the hyperplane, with no datapoints inside. The points that define the location of the hyperplane are known as support vectors.

Some of its problems are that the datapoints must be linearly separable and that in its default state, similar to logistic regression, as it is also limited to binary classification. These problems can be solved by applying OvR or OvO and the kernel trick. The kernel trick is an operation that transforms the space into a higher dimension in which is possible for the data to be linearly separable. One disadvantage of SVMs is that their training complexity can be very high depending on the input data.

K-Nearest Neighbors (KNN) is a very simple machine learning algorithm that makes decisions based on the proximity to datapoints from certain classes to make classifications. It makes the assumption that points of the same class are close together in the feature space. There are many ways of measuring how close a datapoint is, some examples include Euclidean distance, Manhattan distance, Minkowski distance… K is the number of neighbours it measures its distance with. It is a hyperparameter and its choice influences its bias and variance. The main disadvantages of KNN are that is not efficient with large datasets as it does not scale well, and it does not perform well with high-dimensional data, suffering from the curse of dimensionality which can also cause overfitting. Solutions exists such as feature extraction and dimensionality reduction techniques such as Principal Component Analysis.

Random Forest is an ensemble learning technique that combines multiple decision trees and outputs the mode of the classes. Decision trees make observations from the independent input variables and based on those observations, it makes decisions about which class they belong, denoted by the leaf nodes. Random Forest aggregates the classifications made by such decision trees and outputs the most popular result. These decision trees have low correlation between each other as the algorithm creates a subset of features for each tree to train, as well as different subsets of the dataset. The main advantages over normal decision trees are that Random Forest improve overfitting problem and determines better the features that give the best split. In consequence, they are also more complex as they have to compute the result for multiple trees and combine the results.

Some other state-of-the-art techniques predominantly use neural networks. Neural networks are machine learning models inspired by the human brain, in which nodes that are interconnected to each other in a layered structure. These nodes perform linear combinations in parallel between a weight each node has with the input data and generate output values which are fed-forward to the next layer of interconnected nodes, and the final layer will produce a prediction or classification. Neural networks with many layers are called Deep Neural Networks (DNN). DNNs are capable of extracting low-level and high-level features from the data in each layer, that give information about which category they belong. Convolutional Neural Networks are a type of DNN that work very effectively specially well with structured data such as images.

Convolutional Neural Networks are a type or Deep Neural Network that works well with data such as images. CNNs are built with convolutional layers, pooling layers, and fully connected layers. The convolutional layers consist of filters, that slide over the images while performing linear combinations between the weights of the filter and the pixels of the image. The filters or kernels, extract features from the image capturing spatial patterns and structures useful for prediction tasks. The pooling layers reduce the dimensionality of the extracted features, making the network more efficient and the fully connected layers make predictions based on the learned patterns.

3. DESCRIPTION OF MODELS

We have decided to take different approaches for each task, since the problems are slightly different and have separate datasets.

In this section, you should briefly describe the model you are using for each task, along with the rationale. You may opt to use a single learning algorithm to solve the problem or multiple ones, but bear in mind there are page limitations and that you should explain your rationale behind your choices. That is, the algorithmic description must detail your reasons for selecting a particular model.

3.1. Task A: PneumoniaMNIST

For Task A, we have decided to use Logistic Regression, as well as Support Vector Machines and Random Forest. Because of the nature of the dataset, it is highly likely that the images classes might not be linearly separable, however we decide to implement Logistic Regression first to test what performance could it achieve. Since it is a rather simple binary classification problem, it might be possible that traditional machine learning algorithms yield good results, without the need of implementing more complex algorithms such as neural networks.

We choose to implement a simple Logistic Regression algorithm as it computationally efficient and has low memory requirements. Logistic Regression in its basic form does binary classification, and does not need any modifications for this task, besides hyperparameter tuning. It is simple and interpretable and assumes that the data is linearly separable.

The results, discussed in the next section, show us that the data is linearly separable. For this reason, we also try to implement a SVM with a linear kernel. Although SVMs are more computationally expensive, they are also more robust against overfitting and may provide better generalization than Logistic Regression. The linear kernel is defined as Eq. 3.

$$k(X_i, X) = X_i \times X$$

Equation 3. Linear kernel

For the sake of comparison, we will also implement an ensemble method such as random forest, which is capable of obtaining comparable results to SVM with lower training complexity. The training complexity of SVM can be roughly O(n^2m) to O(n^3m) whereas for Random forest O(k*n* log(m)), where n is the number of examples, m is the number of features and k the number of trees.

3.2. Task B: PathMNIST

Because of the more complex dataset, traditional machine learning algorithms will not be able to capture all the complex relationships between the pixels and labels, we therefore decide to instead implement a more complex model capable of extracting relevant features of the images that are useful for classification purposes state of the art models such as VGG16 (Fig. 2), ResNet, AlexNet, use convolutional neural networks or CNNs, for multi-class classification tasks. They show impressive performance with very high accuracy.

CNNs have some special characteristics that make them very suitable for this dataset. CNNs are specifically designed to address the spatial structure of images as they leverage the property that neighboring pixels in an image are usually closely related. By using convolutional layers, CNNs exploit the local connectivity of pixels, capturing useful local patterns and features. This also makes them invariant to translation movements, scale rotation or deformation, meaning that even if the discriminative features are in different locations or positions, the CNN will be able to detect them. in the case of medical images, these characteristics make CNNs an excellent choice for classification tasks.

For these reasons, we find appropriate to adopt CNNs for this classification task.

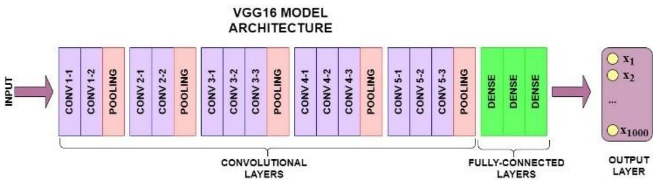


Fig. 1 Architecture of VGG-16

4. IMPLEMENTATION

This section must provide the detailed implementation of your models. In particular, you must provide the name and use of external libraries, explain hyper-parameter selection, training pipeline (if any) and key modules/classes/functions/algorithms.

4.1. Task A: PneumoniaMNIST

4.1.1 Dataset

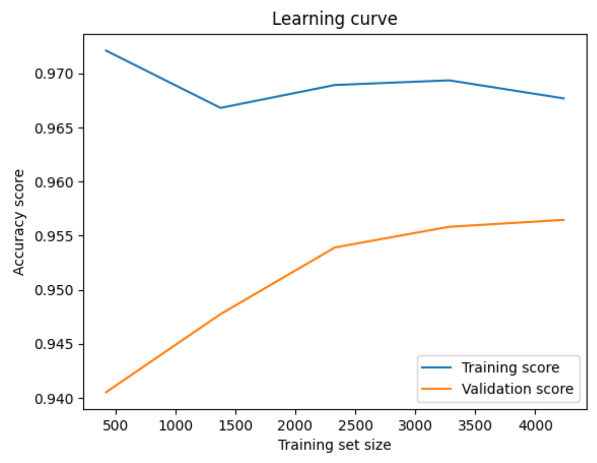
The dataset consists of 5856 images, divided in three subsets for training, validation and testing, each having 4708, 524 and 624 images respectively. Every image has a dimension of 28x28 pixels, with each pixel value ranging between 0 and 255. We find that the dataset is quite imbalanced since almost 75% of the images belong to the same class, this will cause some problems in our models as they will be biased to towards the majority class. to lessen this problem, we try to adjust the weights associated with each class inversely proportional to the frequencies of the input data. The dataset is in the format of a *numpy* array.

For preprocessing, we rescale the images from values between 0-255 to float values between 0-1, this helps with numerical stability. Since we are working with linear models, we will also have to transform the 2D image into a 1D array. This is done by flattening the image.

For the implementation of our models in Task A, we make use of the functions and methods implemented in the *scikit-learn* library. We use the training dataset to train the model, and use five-fold cross validation to evaluate the performance of the model.

4.1.2 Logistic Regression

For Logistic Regression set the maximum number of iterations that the algorithm can execute to 500 epochs to ensure that it converges. For hyperparameter tuning, since the algorithm is not expensive computationally, we train it several times with different hyperparameters. The values that give the best results with better generalization can be seen in the implementation in Fig. 3. As we can see from the graph in Fig. 3, the model converges before the validation accuracy gets close to the training accuracy, this could be normal since the training accuracy is always higher than validation accuracy. Because the validation score is not much lower than training, it is unlikely that there has been any overfitting. The penalty function used for regularization is L2.



```
model = LogisticRegression(penalty='l2', class_weight='balanced', max_iter=500)
```

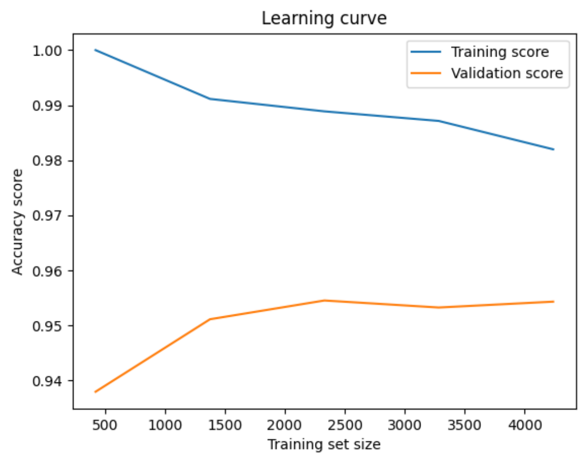
Fig 3. Logistic Regression learning and validation curve, and code implementation, respectively.

The stopping criteria that has been applied is defined by the tol parameter (tolerance of stopping criteria), which in our case is 0.0001. That means that if the learning steps are smaller than 0.0001, the algorithm will automatically stop. This ensures that there is no overfitting.

4.1.3 Support Vector Machine

The implementation and learning curves for SVM is as indicated by Fig 4. We can see that SVM starts to plateau in the last iterations of the training process, meaning that it has successfully found a hyperplane that can separate the two classes. More iterations are unlikely to make any significant changes. The convergence is determined by tol parameter which is 0.0001. We can see from the graph that validation score at the final iterations stops increasing, therefore further training contributes little to nothing to the model.

For hyperparameter tuning, we also run the model several times with different parameters. We use again the L2 penalty for regularization, and the loss function is squared hinge.



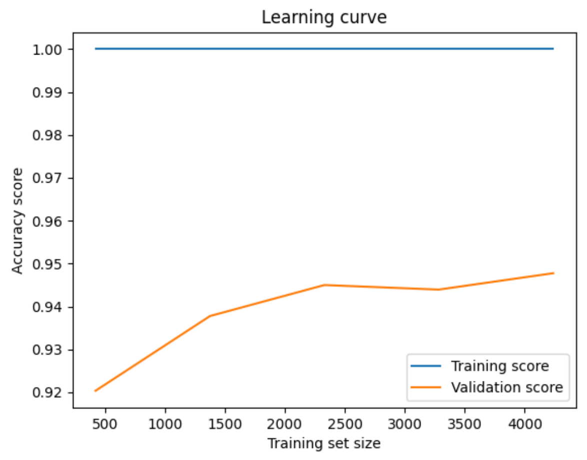
```
model = LinearSVC(penalty='l2', loss='squared_hinge', class_weight='balanced', dual=False)
```

Fig 4. SVM curves and code implementation.

4.1.3 Random Forest

In Fig 5. We can see the implementation of Random Forest and its learning curves. We can see that the validation accuracy gap is slightly larger than the other two algorithms. This suggests that Random Forest do not generalize as well as the other algorithms, and that its complexity hinders the performance of the model.

In the tuning process, even though the model is more complex, because training can be parallelized, the time it takes is still very short therefore we tune in the same fashion as before. The best hyperparameters we get are using 100 trees, using Information Gain to measure the quality of split of the trees.



```
model = RandomForestClassifier(criterion="entropy", class_weight='balanced', n_jobs=10)
```

Fig 5. Random Forest curves and code implementation

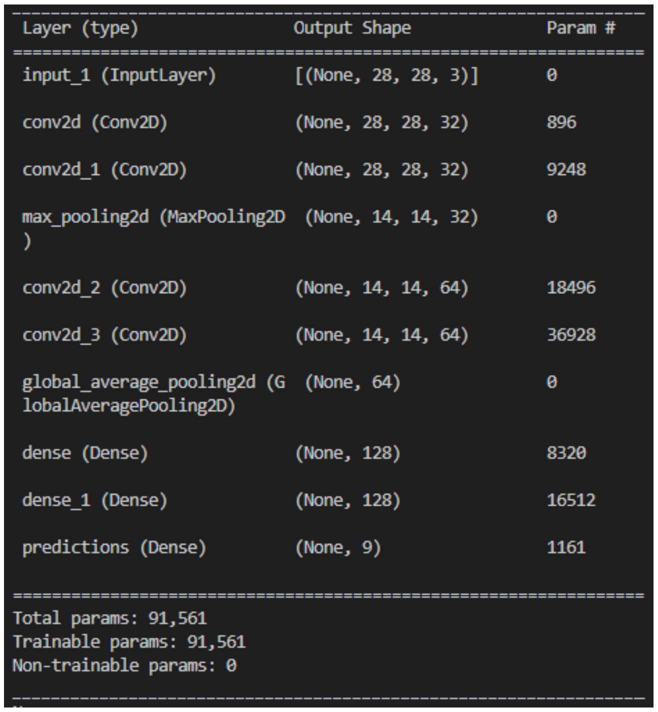
4.2. Task B: PathMNIST

4.2.1 Dataset

The dataset consists of 107180 images, of which 89996 are for training, 10004 for validation and 7180 for testing. the images also have a dimension of 28x28 pixels but this time they have three color channels. The value of each pixel is again between 0 and 255. The dataset is more balanced in this case, so we will be able to train a model to make good predictions without any bias introduced by the dataset. The format is a *numpy* array.

For preprocessing, we only need to scale the data to float values between 0 and 1, as CNNs are capable of handling 2D data like images.

4.2.1 CNN



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 3)]       0

 conv2d (Conv2D)             (None, 28, 28, 32)        896

 conv2d_1 (Conv2D)           (None, 28, 28, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 14, 14, 64)        18496

 conv2d_3 (Conv2D)           (None, 14, 14, 64)        36928

 global_average_pooling2d (G  (None, 64)               0
 lobalAveragePooling2D)

 dense (Dense)               (None, 128)               8320

 dense_1 (Dense)             (None, 128)               16512

 predictions (Dense)         (None, 9)                 1161

=================================================================
Total params: 91,561
Trainable params: 91,561
Non-trainable params: 0
_________________________________________________________________
```

Fig 6. Our CNN architecture

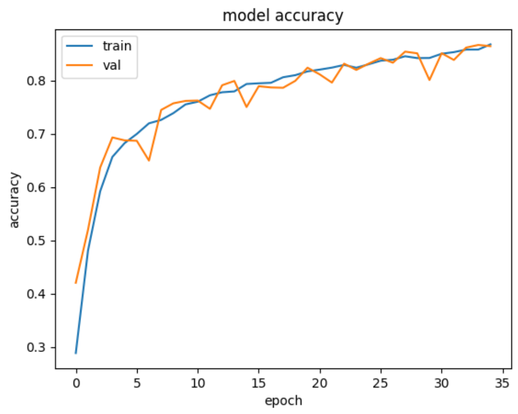
The architecture of our CNN model is given by Fig 6. and is inspired by the VGG16 model. For the implementation of our CNN, we would use the TensorFlow and keras libraries.

Because the problem in our hands is significantly easier to solve, we find unnecessary to use add as many layers as the proposed implementation of VGG16 in the original paper. We therefore make some modifications to simplify the model. We take the first two blocks of convolutions as well as the last three fully connected layers to produce our predictions.

The model takes as input a batch of 28x28x3 images and applies 2 convolutional layers with a ReLU activation function after each convolution, then a maxpooling layer, then another two convolutions and another maxpooling. After this, a average pooling layer is applied before finalizing with three fully connected layers that aggregates all the nodes in the final layer, in which the last one produces an output of nine probabilities of belonging to each class.

The final activation function to produce the output is SoftMax.

We initially set the batch size to 1024 and the training epochs to 35. The first run was able to reach reasonably good results, but however we did not train the model with sufficient epochs and the batch size was very large, leading to poor generalization. as we can see from the learning curve in Fig 7., the model rapidly learns to differentiate between epochs is about x epochs, after that, the improvement rapidly decreases. Although we cannot see the plateau, it is likely that the model would have stopped learning a few epochs after. The loss curve gives a similar impression that the model is still learning, but at a much slower rate. The results, discussed in the next section, could indicate that the model is not trained enough or is not generalizing well. We therefore train a second time with different hyperparameters.



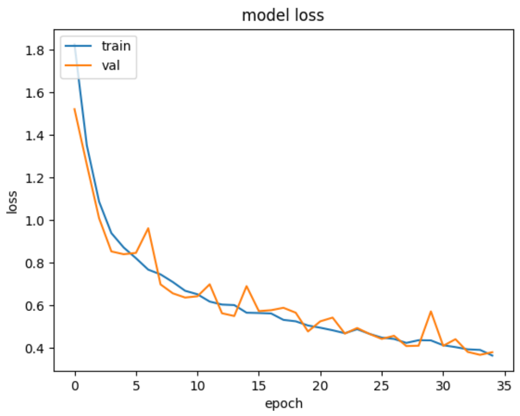


Fig 8. Learning and loss curve of 1st run.

A second training of the model reduces the batch size to 128 and an increase of the number of training epochs to 50. As we can see from Fig. 9, the model starts with a higher accuracy compared to the previous run. the smaller batch size allows the model to learn more examples in every epoch, this is similar to how mini-batch gradient descent sometimes work better than batch gradient descent. the model is able to reach better performance with higher accuracy sooner, suggesting that 50 epochs was more than enough and even too much for optimal solution, as it would start overfitting. The loss curve during training and validation is also smaller compared to the first train. This suggests that the main problem was that batch size. Reducing it has led to impressive performance improvements.

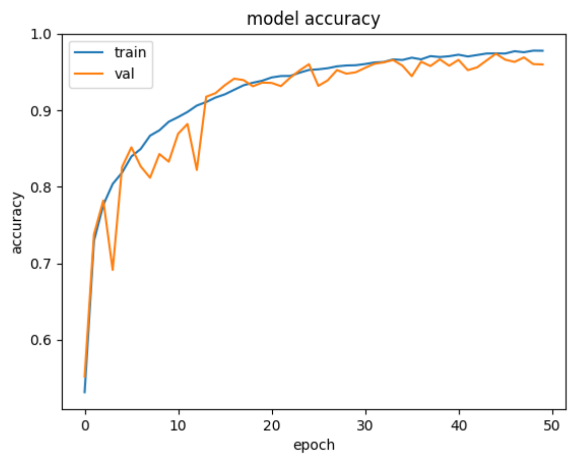


Fig 9. Learning curve of 2nd run.

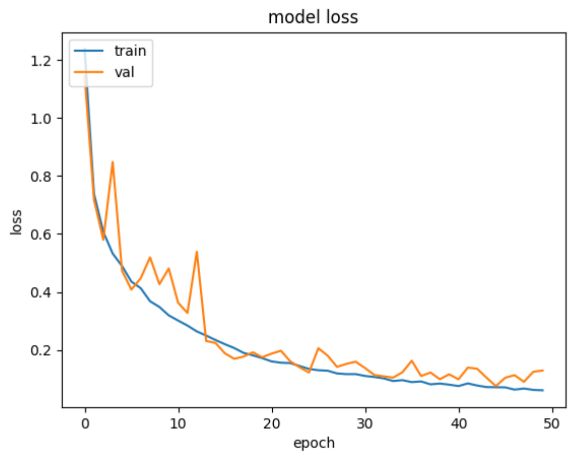


Fig 10. Loss curve of 2nd run.

Because of the fast learning, accuracy stops increasing around epoch 50, similarly, loss also stops decreasing around the same time. We therefore do not increase the training epochs more since later iterations would only cause the model to overfit without improving the performance of the model. The weights for the model in the last epoch are provided in the repository with the code.

5. EXPERIMENTAL RESULTS AND ANALYSIS

5.1.1. Task A

All three models achieve reasonable performance, although the metrics scores for the test set are significantly lower than the validation set. This is mainly because the hyperparameter tuning process was done with the validation set. nonetheless, they are still able to make reasonably good predictions. The metrics obtained by the models can be seen in Table 1.

Table 1. Task A metrics

	Acc	Prec	Rec	F1
LogReg	0.87	0.88	0.87	0.87
SVM	0.87	0.88	0.87	0.87
RanFo	0.85	0.86	0.85	0.84

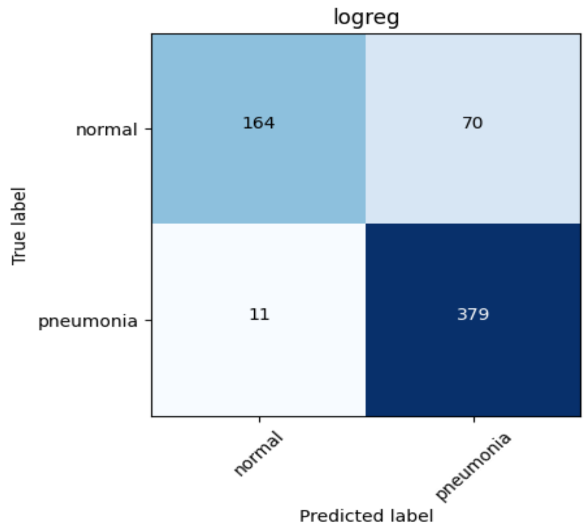
We can see that the metrics for Logistic Regression and Support Vector Machine and the same, this is expected as they both work in similar ways. Since the kernel for SVM is linear, the hyperplane that linearly separates the points. Random Forest, on the other hand, shows slightly worse performance than the other two, despite being able to capture more complex relationships and the decision boundaries between the data and the labels. This could happen if the data has a lot of irrelevant features. In our case, it is possible that not every pixel in the image contains relevant features that can help to classify the images.

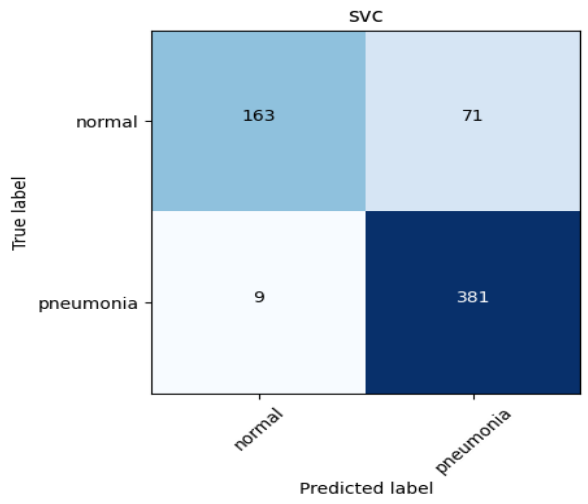
We also compute the confusion matrices for the three models. We can see that all three models have a significant false positive rate, this suggests that the problem is slightly more complex. although linear classifiers are able to achieve good results, applying some dimensionality reduction, may be able to get a better performance. Random Forests, despite naturally being able to learn nonlinear decision boundaries has worse predictions, suggests that maybe our hyperparameters need further tuning.

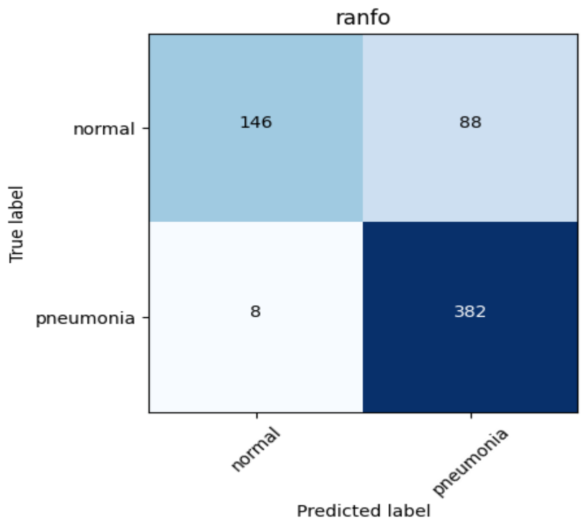
However, considering the simplicity of the model, and given that for this medical dataset ideally there would be a low false negative rate (FNR) these models yield good results (we consider a false negative an incorrect prediction of normal). The results show similar performance in the three models. both LogReg and SVM maintain a low FNR with similar true positive rate (TPR), Random Forest however, has higher TPR and not so different FNR. Considering the complexity of the models, these results clearly show that among the three models evaluated, LogReg is surprisingly the best performing model as it is a much simpler, faster to train, computationally less expensive and more explainable than the other two. SVM and Random Forest fail to perform better despite an increased learning capacity.

They clearly show some limitations, and might not be suitable for real life applications, however they prove that traditional machine learning algorithms are also capable of yielding good results without the need of complex architectures, and that they might be good choices for other kind of problems.

For example, LogReg and SVM assume linear separability of the data, which is not always the case, they are also unable to capture spatial relationships between the data and are very sensitive to outliers. Random Forest lack interpretability which is very important in medical imaging.







5.1.2. Task B

We can see in Table 2., a weighted average of each score for each run. It is clear that the change of batch size greatly impacted the accuracy in our model. The first iteration had a significantly lower score in all metrics. v2 has a very high accuracy. The lower precision in class 2 and 7 means that they have high number of false positives. However, a less acceptable score is the recall for class 5 and 7, which indicates that there are high numbers of false negatives. The three worst performing classes are 2, 5, and 7, with them having the lowest F1 score.

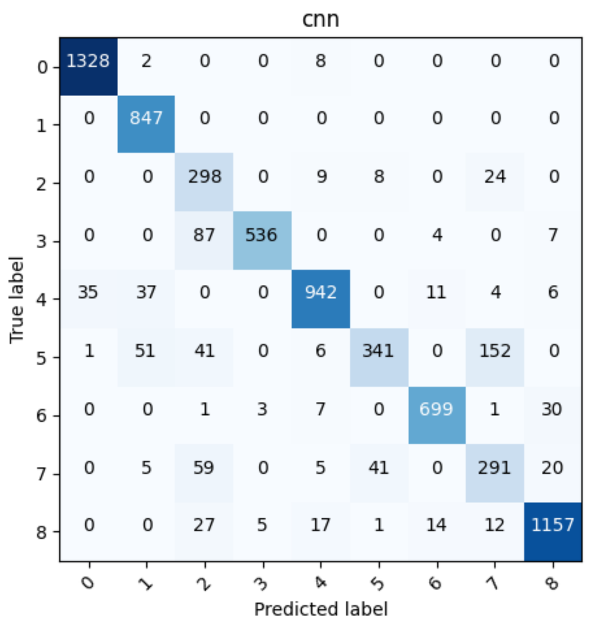
It is unclear why the model has not been able to correctly classify theses classes, a possible explanation is unbalance in the dataset however this is not the case. It is possible that the images for these classes do not have good quality or that the model is unable to capture relevant information from them. The most likely explanation is that the model has overfit/underfit for those classes in particular during training, as the training accuracy is very high.

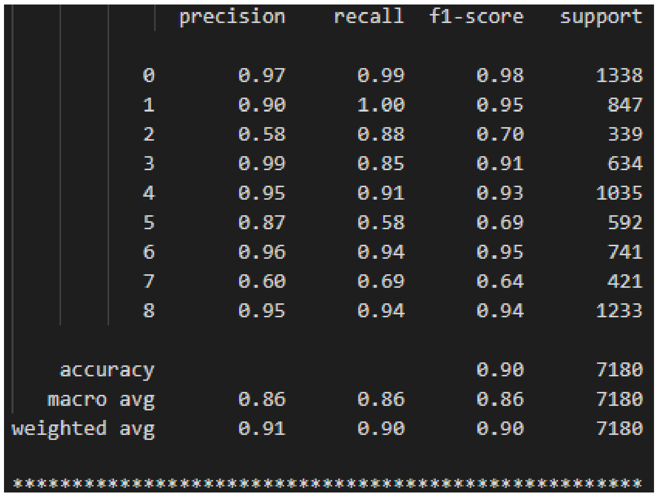
Overall, the model seems to be able to extract most relevant features and classify the images with high accuracy.

Table 2. contains the metrics obtained from the test dataset.

Table 2. Metrics from Task B

	Acc	Prec	Rec	F1
v1	0.78	0.81	0.78	0.78
v2	0.90	0.91	0.90	0.90





```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1338
           1       0.90      1.00      0.95       847
           2       0.58      0.88      0.70       339
           3       0.99      0.85      0.91       634
           4       0.95      0.91      0.93      1035
           5       0.87      0.58      0.69       592
           6       0.96      0.94      0.95       741
           7       0.60      0.69      0.64       421
           8       0.95      0.94      0.94      1233

    accuracy                           0.90      7180
   macro avg       0.86      0.86      0.86      7180
weighted avg       0.91      0.90      0.90      7180

*******************************************************
```

Figure 11. Confusion Matrix and per-class metrics.

6. CONCLUSION

This project has analyzed and implemented different algorithms for different classification tasks. We demonstrated that traditional machine learning algorithms are capable of achieving good performance in binary classification tasks while being simple and explainable. For more complex datasets, it is more desirable to develop deep neural networks as they have greater capabilities, as they are able to learn the patterns in the images.

7. REFERENCES

[1] Karen Simonyan, & Andrew Zisserman. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition.

[2] Díaz-Uriarte R, Alvarez de Andrés S. Gene selection and classification of microarray data using random forest. BMC Bioinformatics. 2006 Jan 6;7:3. doi: 10.1186/1471-2105-7-3. PMID: 16398926; PMCID: PMC1363357.

[3] Kirasich, Kaitlin; Smith, Trace; and Sadler, Bivin (2018) "Random Forest vs Logistic Regression: Binary Classification for Heterogeneous Datasets," SMU Data Science Review: Vol. 1: No. 3, Article 9.
Available at: https://scholar.smu.edu/datasciencereview/vol1/iss3/9

[4] Boyang, Li., Qiangwei, Wang., Jinglu, Hu. (2013). Fast SVM training using edge detection on very large datasets. Ieej Transactions on Electrical and Electronic Engineering, 8(3):229-237. doi: 10.1002/TEE.21844

[5] Bobby Azad, Reza Azad, Sania Eskandari, Afshin Bozorgpour, Amirhossein Kazerouni, Islem Rekik, & Dorit Merhof. (2023). Foundational Models in Medical Imaging: A Comprehensive Survey and Future Vision.

[6] Ian Goodfellow, Yoshua Bengio, & Aaron Courville (2016). Deep Learning. MIT Press.

[7] https://ozbunae.medium.com/class-imbalance-and-hyperparameters-in-svm-ce8b0f8f9e97

[8]https://stats.stackexchange.com/questions/242833/is-random-forest-a-good-option-for-unbalanced-data-classification

[9] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[10] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC

[11] https://stackoverflow.com/questions/58275113/proper-use-of-class-weight-parameter-in-random-forest-classifier

[12] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[13] https://towardsdatascience.com/evaluating-linear-relationships-1d239f51297b

[14] https://stats.stackexchange.com/questions/182329/how-to-know-whether-the-data-is-linearly-separable

[15] https://marcovirgolin.github.io/extras/details_time_complexity_machine_learning_algorithms/

[16] https://stackoverflow.com/questions/52670012/convergencewarning-liblinear-failed-to-converge-increase-the-number-of-iterati

[17] https://stackoverflow.com/questions/30972029/how-does-the-class-weight-parameter-in-scikit-learn-work

[18] https://keras.io/examples/vision/mnist_convnet/

[19] https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/

[20] https://stackoverflow.com/questions/73422130/what-are-all-the-valid-strings-i-can-use-with-keras-model-compile?noredirect=1&lq=1

[21] https://www.analyticsvidhya.com/blog/2021/05/tuning-the-hyperparameters-and-layers-of-neural-network-deep-learning/

[22]https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit

[23] https://www.linkedin.com/advice/1/how-do-you-balance-trade-off-between-regularization#:~:text=Regularization%20in%20CNNs%20works%20by,a%20trade%2Doff%20between%20them.