APPLIED MACHINE LEARNING SYSTEM ELEC0134 22/23 REPORT

SN: 20121776

ABSTRACT

When it comes to biomedical image classification, machine learning algorithms are often helpful to automate and speed up the process. This project aims to train numerous machine learning models for pneumonia diagnosis and colorectal cancer survivability prediction, including Decision Tree (DT), Support Vector Machine (SVM) and Convolution Neural Network (CNN). The model performance is evaluated using accuracy, precision and recall, thus allows the optimal model to be selected. [1]

***Index Terms*—** DT, SVM, CNN, Image Classification

1. INTRODUCTION

The Modified National Institute of Standards and Technology database (MNIST) was created in the early 90s with a total of 70000 28x28 pixel images for image training purposes. The abbreviation of MNIST, then became the definition of this image format, and is widely used in machine learning.

This project uses two datasets (pneumoniaMNIST and pathMNIST) from the MedMNIST to perform two biomedical image classification tasks. The former consists of a total of 5856 X-ray chests images, while the latter consists of a total of 107180 histological images. Given the size of the dataset, manual classification is impratical and thus machine learning can be utilized.

To classify the binary grey scale images in task A, both SVM and DT are adopted. For better comparison, the SVM accepts a range of kernels, while the DT is trained using Adaptive Boosting (AdaBoost). The performance of the models are then evaluated using metric scores (accuracy, precision, recall) and the confusion matrix.

Yet, given that the dataset for task B has RGB images, the total number of features might be overwhelming for SVM or DT. A CNN is therefore constructed to handle the more complex data and improve the overall prediction accuracy. Similar to task A, metrics and plots will be provided for performance evaluation.

This report will therefore explain the theories of all 3 models, provide implementation details, as well as present quantitative performance measurements. Finally, suggestions will be made to further improve the models.

[1]The code can be accessed and downloaded from the GitHub project: AMLS_assignment23_24_SN20121776

2. LITERATURE SURVEY

In the field of image classification, a variety of supervised classical and deep learning machine learning approaches exist. Depending on the complexity of the dataset and the performance requirements, a single problem may have multiple potential solutions. For example, [1] states that SVM and CNN should be used for medium and large scale biomedical images dataset respectively to ensure satisfactory accuracy. A more in-depth research is carried out in [2] to compare the rationale and performance of a wide range of classical (SVM, Random Forest, Linear Regression) and deep learning (Artificial Neural Network, CNN, Recurrent Neural Network) machine learning techniques to biomedical images.

All the aforementioned approaches are potential solutions to this project, hence further literature review has been conducted on the selected methods to briefly explain their working principles, and provide application examples.

2.1. Decision Tree

Similar to an actual tree, a DT is constructed with nodes and branches. In between the parent (root) and the child (leaf) nodes, internal nodes are created by subsequent decisions. All 3 types of nodes are then connected by different branches (decision) to form a DT [3].

The simple structure and implementation of DTs made them reliable for data classification. For instance, [4] has developed a DT algorithm to classify a grey-scale x-ray image dataset into 57 classes. As the boosting strategy is proved to be advantageous to prediction accuracy, this project also aims to implement the Adaptive Boosting technique to enhance the performance of the DT model.

2.2. Support Vector Machine

The fundamental principle of a SVM is to construct hyperplanes to isolate data into different classes. In cases where the feature patterns are unrecognisable, a kernel function could be used to increase the dimensionality of the feature space, hence result in separable data [5].

An example of SVM implementation in the biomedical field is for classifying magnetic resonance (MR) images. [6] has successfully designed a SVM with radial basis function (rbf) kernel to classify MR brain images into normal or abnormal based on their axial and coronal symmetry. This lit-

erature, however, only utilized one kernel type and has sub-optimal accuracy. This project therefore aims to provide comparison between different kernels as well as comprehensive hyperparameter tuning to improve the model performance.

2.3. Convolution Neural Network

As a deep learning architecture, CNN is more complex compared to DT or SVM. It is composed of 3 types of layers, namely the convolution, pooling and fully-connected layers [7]. When the convolution layer is applied to a image, a feature map is generated using kernel filters. The resolution of the feature map is then lowered by the pooling layer to reduce image complexity. After stacking several convolution and pooling layers, the fully-connected layers are used to relate the data and generate the final output.

A novel implementation of CNN in biomedical image analysis is carried out in [8]. To reduce the time cost for manually labelling a large image dataset for machine learning, the researchers developed an active, incremental fine-tuning (AIFT) technique. This allows a pre-trained CNN to continuously annotate a portion of the dataset, while using the new annotated data to fine-tune the model. Although the training images for this project are already labelled, such approach could be a future improvement to extend the functionality of the model.

3. DESCRIPTION OF MODELS

3.1. Task A

Given that the size of the PneumoniaMNIST is relatively small, deep learning architectures that benefit from larger datasets may not necessarily outperform classical algorithms, both in terms of accuracy and computational intensity. Consequently, while CNNs are well-known for their prowess in image recognition, DT and SVM models are opted for this task.

3.1.1. Decision Tree

The proecess of constructing a DT starts with finding the feature that provides the most information gain (IG). This could be calculated by deducting the entropy of the class labels with the entropy of the class labels given a certain feature, as shown in the equations below.

$$H(Y) = -\sum_{x} p_x(y)\log_2(p_x(y))$$
$$H(Y|X) = -\sum_{x} p_x(y|x)\log_2(p_x(y|x))$$
$$IG(Y|X) = H(Y) - H(Y|X)$$

Where $H(\cdot)$ and $p_x(\cdot)$ represents entropy and probablity respectively. The feature with the highest IG is then selected as the root node, while the features with the next highest IG become its child nodes and so on. Depending on the stopping criteria, the resultant tree may be shallow or deep. Shallow DTs may have high bias, while deep DTs are prone to overfitting.

Therefore, adapative boosting from ensemble methods is chosen in this project to overcome the limitations of DTs. Instead of developing a strong, deep model, a weaker, shallower tree is constructed. At the beginning, each training data is given an equal weight. The values are then adjusted based on the prediction accuracy in each iteration. By increasing and decreasing the weights for wrongly and correctly classified data respectively, a new and relatively stronger model can be developed. This process repeats to strengthen the tree until the stopping criteria is met, which is usually the total number of iterations.

3.1.2. Support Vector Machine

As mentioned in Section 2, hyperplanes are the keys of SVM. They are optimised by maximising the margin (m) between the support vectors, which are the data points closest to the hyperplane. The distance between any data point $\boldsymbol{x}$ to the separator can be calculated as:

$$r = \frac{\boldsymbol{w}^T\boldsymbol{x}_i + b}{||\boldsymbol{w}||}$$

Where $\boldsymbol{w}$ is the weight vector that determines the orientation of the hyperplane, and b is the bias that offset the hyperplane from the origin.

Yet, when encountering non-linear problems, the training set may be inseparable by a 1-dimensional line. Kernel functions can therefore be used to map the features into a feature space with higher dimensionality. The most common Kernel functions include:

Table 1: Equations of common Kernels

linear	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T\boldsymbol{x}_j$
poly	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (1 + \boldsymbol{x}_i^T\boldsymbol{x}_j)^p$
rbf	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\frac{\|\|\boldsymbol{x}_i - \boldsymbol{x}_j\|\|^2}{2\sigma^2})$

When mapping samples to a higher-dimensional feature space, the transformation $\phi(\boldsymbol{x})$ of each data point is often complicated and computational intensive. The use of Kernel functions allows the transformation process to be implied without explicit calculations. This effectively lowers the computational cost and reduces the training time.

3.2. Task B

Although DT and SVM are capable of generating satisfactory results for Task A, their performance and training time

increases significantly with more features and larger datasets. Being a RGB 28x28 pixel image dataset, each sample of PathMNIST effectively has $3 \times 28 \times 28 = 2352$ features. This would require an exceptionally long processing time, especially if hyperparameter tuning and cross-validation are involved.

To maintain a reasonable training time, CNN is chosen for this task. The nature of CNN allows for the downscaling of image resolution using the combination of convolution and pooling layers, while keeping the relevant features. This increases the computational efficiency, and enables model training on large-scale datasets without comprimising performance.

3.2.1. Convolution Neural Network

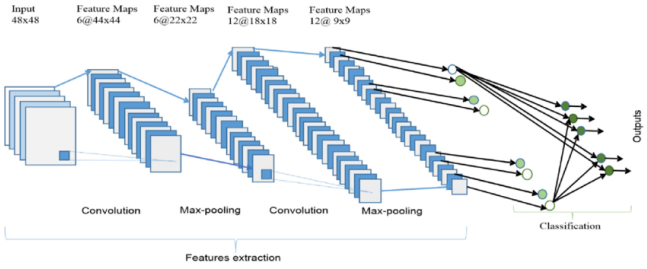


Fig. 1: A simple CNN structure. [9]

As shown in Fig. 1, CNN consists of multiple convolution, pooling and fully-connected layers. When an image is input to the network, a kernel filter will sweep through all pixels to generate a new feature map. In general, if padding is included, the new feature map would maintain its original dimension, as shown in the right example of Fig. 2. Kernel filters are essentially weights of pixels and are initialised randomly, but will automatically improve (from detecting corners and edges to classifying images) using the backward propagation technique, which will be explained later in this section.

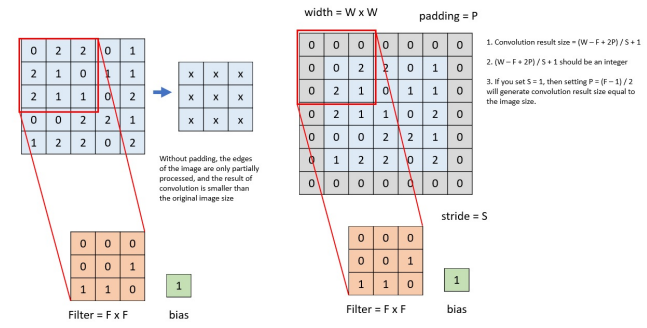


Fig. 2: Convolution operation with and without padding. [10]

The new feature map generated by the convolution layer will then go through a pooling layer to keep only the most important information. This layer splits the image into non-overlapping pooling regions (common sizes are 2x2 and 3x3) for max pooling or average pooling to be applied. The former discards all values except the highest, while the latter averages all values within the region. Either method down-samples the image and effectively removes the unnecessary parameters, which simplifies further calcuations.

After alternating multiple convolution and pooling layers, all nodes from the final feature map will be linked a fully-connected layer. This is where weightings and bias are applied so that the information captured in previous layers can be used for classification. It is worth noticing that in between each layer, activation functions (typically ReLU functions) are used to process the feature maps. Their non-linearity and non-zero gradients avoid the CNN from being a linear model, and allows for gradient descent based backward propagation respectively.

With all the steps above, the first epoch (iteration) of a CNN is completed. The classification loss are then used to calculate the partial derivatives with respect to weights and bias in preceding layers. Their values are then backward propagated and updated using the equation below.

$$\Phi_{new} = \Phi_{old} - l\frac{\partial J(\Phi)}{\partial \Phi}$$

Where Φ is the value of weight or bias, l is the pre-defined learning rate and $J(\Phi)$ is the loss function. By introducing more epochs, the loss of the network will decrease and eventually reach an optimal value.

4. IMPLEMENTATION

4.1. Task A

4.1.1. Data Pre-processing

Scikit-learn and *Imbalance-learn* are the two major libraries used in Task A. As their built-in functions for DT and SVM require the same input syntax, the two classes involved (Svm and Tree) share similar methods structure but different parameters and models.

The code for both classes starts with initialising the instance by loading the data from the PneumoniaMNIST.npz file. Since this dataset contains pre-splitted 28x28 gray scale images for the training, validation and testing set (4708/ 524/ 624 images respectively), no further separation is required. The occurance of each unique class label is then recorded to check if dataset is severely unbalanced. It appears that Class 1 (3494 images) has almost triple the number of samples in Class 0 (1214 images). This could result in biased and underperforming models. A synthetic Minority Over-sampling technique (SMOTE) is therefore used genereate more synthetic data points for Class 0.

The augmented dataset is then used in the hyperparameter tuning, cross-validation and prediction functions.

4.1.2. Decision Tree Hyperparameters tuning

The DT model for this task consists of 2 tune-able hyperparameters as shown in Table 2. n is responsible for controlling the total number of weaker classifiers, while lr is the learning rate that adjust the weights for datapoints in each iteration. Note that the default base estimator parameter of the *AdaBoostClassifier* function is configured as *DecisionTreeClassifier* with maximum depth of 1. This is the fundamental definition of an Adaptive Boosted DT and hence no modification is made.

Table 2: Range of Hyperparameters for the Decision Tree Model

Parameter	Symbol	Range
No. Weak Classifiers	n	50 - 200
Learning Rate	lr	0.1 - 10

4.1.3. Support Vector Machine Hyperparameters Tuning

The main difference between the Tree and the Svm class is that the latter takes an additional parameter (kernel) in initialisation. To allow for thorough comparison, all 3 kernels stated in Table 1 can be used. This parameter configures the tune-able hyperparameters as shown in Table 3, and adjust the kernel function used in the SVM model.

Table 3: Range of Hyperparameters for the Support Vector Machine Model

Parameter	Symbol	Range
Linear Kernel		
Regularisation Constant	C	1e-7 - 1e1
Polynomial Kernel		
Regularisation Constant	C	1e-7 - 1e1
Degree	d	5 - 7
Constant	r	1e-3 - 1e3
Rbf Kernel		
Regularisation Constant	C	1e-7 - 1e1
Gamma	γ	1e-7 - 1e-5

More specifically, C is the regularisation constant that determine whether the SVM has a strict decision boundary. Small C is prone to training errors, while large C is prone to testing errors due to overfitting. d are r are exclusive to polynomial kernel SVMs. The former controls the total feature space dimensions used by the model, while the latter adjusts the importance of higher-degree polynomials. The final parameter γ is the only available to the rbf kernel. It modifies the width of the gaussian function, which affects the flexibility of the decision boundary.

4.1.4. Model Training and Prediction

The *GridSearchCV* function from the *scikit-learn* library is used to perform a 3-fold cross-validation on the validation set over all possible combinations in Table 2 and Table 3. The optimal parameters and their validation accuracies are presented as follows.

Table 4: Optimal Hyperparameter Values and Validation Accuracy

Model	lr	n	C	d	r	γ	%
Tree	1	200					92.3
Linear SVM			1e-6				88.6
Poly SVM			1e-6	6	10		91.3
Rbf SVM			10			1e-6	96.0

After obtaining the best parameters, 5-fold cross validations are carried out on the training set using the *KFold* and *cross_val_score* functions, as shown in Table 5. If the results are satisfactory, the model will be used for predictions.

Table 5: 5-fold Cross Validation Accuracy (%)

Model	1	2	3	4	5	Avg
Tree	95.6	95.4	94.5	95.3	95.3	95.2
Linear SVM	88.9	88.3	89.1	88.5	87.8	88.5
Poly SVM	94.1	94.8	93.5	95.2	93.5	94.2
Rbf SVM	98.1	98.5	98.1	97.9	98.1	98.1

4.1.5. Model Evaluation

In order to simplify results analysis, related parameters and metrics, including the best hyperparameter set and its validation accuracy, the 5-fold cross-validation accuracy, prediction accuracy, precision, recall and f1 score can be accessed directly as class attributes. If necessary, the data can also be exported to individual .CSV files for futher processing.

4.2. Task B

4.2.1. Data Pre-processing

In Task B, the *Pytorch* library is used for constructing the CNN model, and the *Scikit-learn* library is used for calculating the performance metrics. To start with, the PathMNIST.npz file consists of pre-splitted 28x28 RGB scale images for the training, validation and testing set (89996 / 10004 / 7180 images respectively). Similar to Section 4.1, datasplitting is not required. Further inspection shows that the database is fairly balanced, with around 8000 - 12000 samples

per class in the training set, and around 900 - 1500 samples per class in the validation set. Thus data augmentation may not have significant impact to the overall performance of the CNN model.

Due to the unique data format (tensor) required by *Pytorch*, a *CustomImageDataset* class that inherits the *Dataset* class from *Pytorch* is used to load and reformat the data from the .npz file.

4.2.2. CNN Architecture Configuration

The construction of a CNN model begins with defining the number of convolution layers. This model settles with 3 layers as trial and error shows that any increase beyond 3 gives diminishing performance boost. After that, the detailed parameters for each layer are determined as Table 6. Where C_{in}, C_{out}, K, S and P represent the input channels, output channels, kernel size, stride and padding respectively. Since the images are in RGB format, the input channels to the first layer are bound to be 3 (for R, G and B). The number of channels for other layers are arbitrary, but should gradually increase for more details in the images to be captured. In this model, the number of channels are selected from 2^n, which is a typical configuration of CNN.

K controls the size of the filter applied to the image. Conventional filter sizes are 3x3, 5x5 and 7x7, but the larger the filter is, the longer the training time will be. Given that the images used in this task are relatively small (28x28), 3x3 filters are more than sufficient. S adjusts how far the filters are applied from each other. Again, with small images, it is more preferred to have smaller strides so that more information can be obtained. P determines the size of the paddings around the input images. As mentioned in Section 2, the dimension of the image would decreases for each convolution layer it goes throgh without paddings. This is sub-optimal as they are supposed to be down-sampled by the pooling layers, not the convolution layers. Size 1 paddings are therefore involved in all 3 convolution layers of this model.

Table 6: Parameters for Each Convolution Layer

Layer	C_{in}	C_{out}	K	S	P
1	3	16	3	1	1
2	16	32	3	1	1
3	32	64	3	1	1

The next step is to define the activation function that provides non-linearity. The Retified Linear Unit (ReLU) function is chosen for this model, as it is one of the most common options for CNN.

$$ReLU(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

To down-sample the feature spaces, the max-pooling layers are defined to have 2x2 kernel size and stride of 2. This effectively reduces the length and width of the original space by 2, so that the useful information can be concentrated and further processed. The CNN ends with 2 fully-connected layers, which flatten the feature space from the 3rd pooling layer and construct two vectors with length of 128 and 9 respectively.

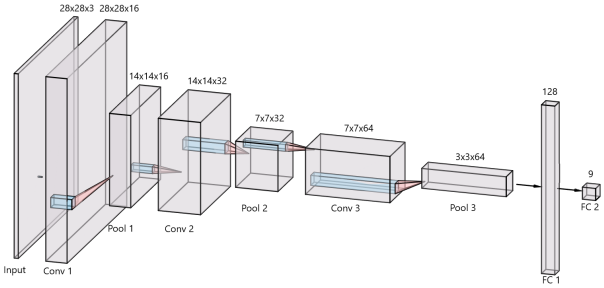


Fig. 3: The detailed architecture of the CNN model

The 9 outputs in the 2nd pooling layer then represent the 9 possible classes of the dataset.

4.2.3. CNN Model Parameters

After initialising the *Cnn* class, the *DataLoader* function from *Pytorch* is used to sample the training and validation images in batches of 32. Together with the total number of classes, the multi-class cross entropy loss function can be written as:

$$Loss(y, \hat{y}) = -\frac{1}{32}\sum_{i}^{32}\sum_{j}^{9} y_{ij} \log\left(\hat{y_{ij}}\right)$$

Where y_{ij} and $\hat{y_{ij}}$ are the true and predicted labels respectively. By defining learning rate as 0.001, momentum as 0.9 and decay factor as 0.9, the stochastic gradient descent (SGD) can be more dynamic and accelerate the process of loss minimisation.

4.2.4. Model training and Prediction

With all the aforementioned architectures and parameters, the CNN model can now be trained with the provided dataset. To speed up the process, it is recommended that the code should be run with Compute Unified Device Architecture (CUDA) compatible computers to enable GPU computation.

In each iteration (epoch), batches of training data will be fitted in the model. The losses for each batch are calcuated with the cross entopy function. Using the built-in *loss.backward* function, the losses can be propagated backward to fine tune the kernel filters. The model is then validated using the validation set, and the average training and validation loss are recorded to demonstrate the model's improvement over each epoch.

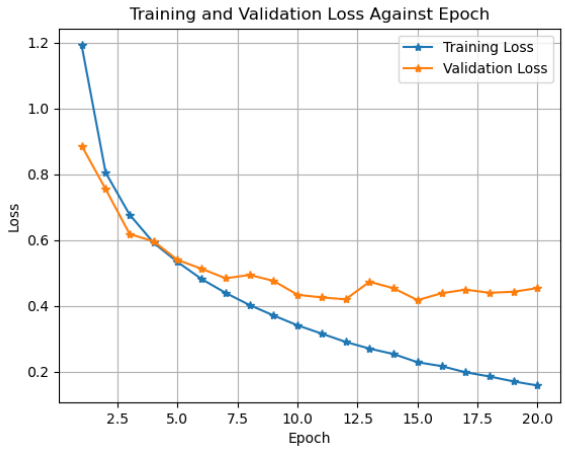


Fig. 4: The visualised performance metrics

As more epochs are carried out, the validation accuracy will begin to stagnate or even deteriorate due to overfitting, as shown in Figure 4. The stopping criteria is therefore defined such that the best model state is the last epoch with improvement in both accuracy and loss compared to the previous epoch. This technique shows that the best number of epoch for this model is 15, with validation loss and accuracy of 0.417 and 86.0 % respecively.

4.2.5. Model Evaluation

Similar to Task A, the class instance of the CNN model allows for direct access of the performance metrics via class attributes. A multi-class confusion matrix is also generated using the *Scikit-learn* library for better visualisation.

5. EXPERIMENTAL RESULTS AND ANALYSIS

To perform more comprehensive analyses, several metrics are used. Accuracy measures the percentage of correctly predicted samples, while precision and recall measures the percentages of true positives in predicted and actual positives respectively. Note that the F1 score is the harmonic mean of precision and recall, hence by maximising this value, both parameters are improved simutaneously. The metrics and their corresponind equations can be viewed as follows.

$$Accuracy = \frac{TP + TN}{n}$$
$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN}$$
$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

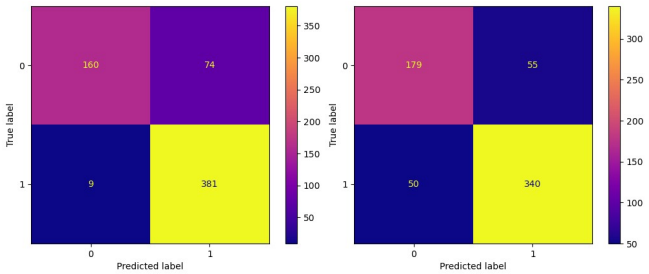
Where TP, TN, FP, FN and n represent true positive, true negative, false positive, false negative and total number of test samples respectively. These values can be obtained from the confusion matrices for each model.

In biomedical images classification problems, it is less desirable to not identify a positive case, than identifying a negative case as positive. This is because positively flagged patients tend to repeat the imaging process to verify the diagnosis, while negatively flagged patients tend to halt further medical interventions. This may leave patients oblivious to their conditions and result in delayed treatment. However, if false alarms (false positive) are raised too often, patients may lose confident in the model, and render the entire system obsolete. It is therefore essential to balance the trade-off between false positives and false negatives to maximise the performance of the model.

5.1. Task A

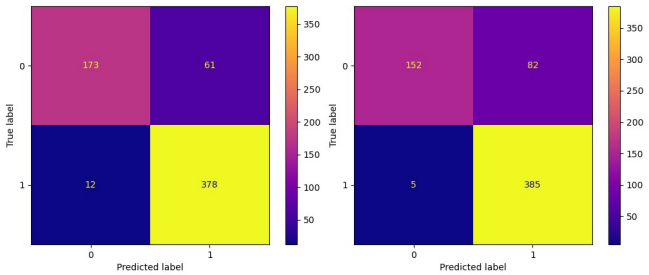
Table 7: Performance Metrics (%) for Models in Task A

Model	Accuracy	Precision	Recall	F1
Tree	86.7	83.7	97.7	90.2
Linear SVM	83.2	86.1	87.2	86.6
Poly SVM	88.3	86.1	96.9	91.2
Rbf SVM	86.1	82.4	98.7	89.8



(a) Decision Tree Model

(b) Linear SVM Model



(c) Poly SVM Model

(d) RBF SVM Model

Fig. 5: Confusion Matrices for the Models. Top left: TN; Top Right: FP; Bottom Left: FN; Bottom Right: TP

According to the results in Table 7 and Figure 5, all four

models in Task A have decent performance, but have weaknesses in different aspects.

Table 7 highlights that the linear SVM has the worst accuracy and recall of 83.2 % and 87.2 % respectively. Out of the 390 positive images, 50 are misclassified as negative. This reveals the model's likelihood to false negative predictions. The reason for this failure is the non-linearity of the input features. With linear kernel, the model attempts to separate the samples using a straight line, which is hardly effective due to the complexity of 28x28 images. The linear SVM, therefore, is undesirable for biomedical images classification tasks.

On the other hand, although the rbf SVM has demonstrated an outstanding performance in recall of 98.7 %, its overall accuracy, precision and f1 score are subpar compared to the poly SVM and DT models. This means that this model is prone to false positive results, with 82 misclassification in 470 positive predictions. Since Table 4 and 5 shows that this model has the highest validation accuracy among all models, it is justifiable to say that the rbf SVM model is overfitted to the validation and training set. Consequently, the rbf SVM is also sub-optimal for this task.

Suprisingly, the DT and Poly SVM are the two best-performing models. They have similar accuracy, with DT excelling in recall and Poly SVM standing out in precision. Both models achieves over 90% in f1 score, indicating that they both manage to achieve satisfactory balances for avoiding false positives and false negatives, which are critical to biomedical images classification tasks. The detailed metrics in Figure 5a and 5c reveals that the DT has 74 false positives and 9 false negatives, whereas the poly SVM has 61 false positives and 12 false negatives. As the the latter has more balanced results and relatively low false negative rate, it can be concluded that the poly SVM model is the best option for diagonsing pneumonia.

5.2. Task B

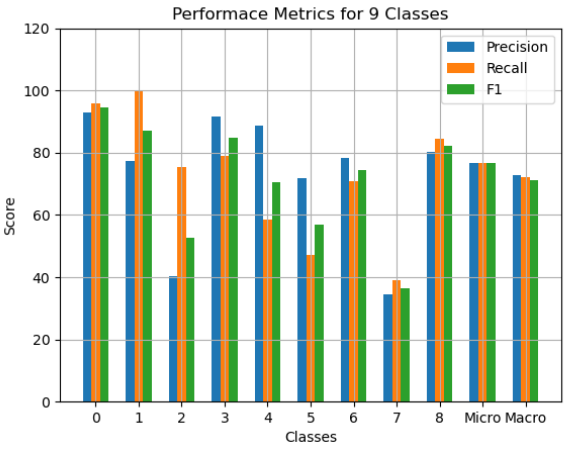


Fig. 6: The visualised performance metrics

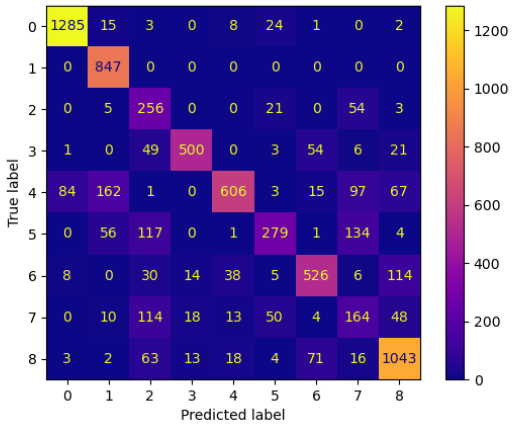


Fig. 7: Confusion matrix for the CNN model. Left diagonal: TP; Sum of values horizontal to TP: FN; Sum of values vertical to TP: FP; Others: FN

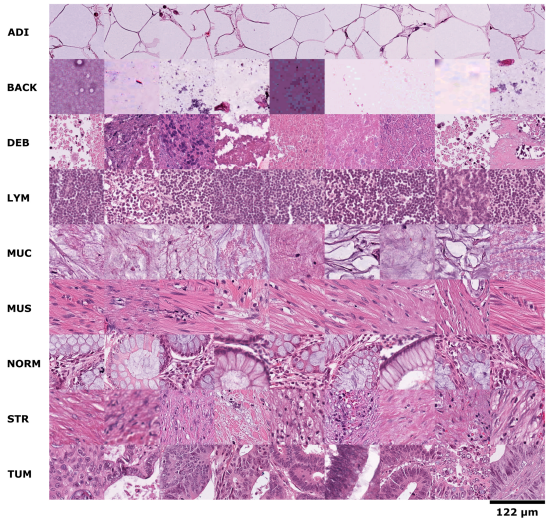


Fig. 8: Class samples of the PathMNIST dataset [11].

As Task B is a multi-class classification problem, metrics are class-dependent. The micro-metrics are calculated using the sum of all true positives, false positives and false negatives in Figure 7, whereas the macro-metrics are the unweighted mean of metrics from each class. For better presentation, a bar graph is plotted as Figure 6.

Given that the average overall prediction accuracy of the CNN model is 76 %, Figure 6 shows that certain classes have significantly inferior performance compared to others. For instance, while the majority of classes have metrics above 60 %, class 2 has precision of 40.4 %, class 5 has recall of 58.6 %, and class 7 has both metics below 40 %. These outliers not only lower the class-corresonding f1 scores, but also confine both micro and macro metrics to around 70 %. Although the overall accuracy is satisfactory, the error spikes in specific

classes may make the model unreliable for biomedical applications, especially when the false negative rate (1 - recall) for class 7 is as large as 61 %.

Further inspection on the PathMNIST dataset in Figure 8 shows that the 3 poorly performing classes (2: DEB , 5: MUS, 7: STR) have somewhat similar patterns (pink background with tiny white strips or dots), making them less distinctable. Hence with the current simple CNN model, these classes are more prone to classification errors compared to the others.

6. CONCLUSION

In summary, the suggested models have successfully solved the classification problems in both tasks. The best performing model in task A is the Polynomial Kernel SVM, which has predicion accuracy of 86 % and has demonstrated the best balance between false positives and false negatives. For task B, although the CNN model struggles to correctly classify certain classes, it still manages to maintain the overall prediction accuracy of 76 % and macro-metrics of around 70 %.

Despite the satisfactory results, the models certainly have a large room for improvement. For instance, given the limited size of PneumoniaMNIST, data augmentation can be carried out for a more comprehensive dataset. It is expected that with more available samples, more accurate hyperparameters can be found to enhance the model performance. Also, Section 5 shows that the 2nd well-performing model (DT) uses 200 weak classifiers, which is the upper limit of the defined range. If the maximum n is higher, the DT may out-perform other SVM models at the cost of computational efficiency.

As for task B, knowing that the majority of error comes from classes with similar patterns, more internal channels and convolution-pooling layer set can be included in the CNN model architecture. This effectively increases the depth and complexity of the model, hence allows for more image details to be captured. It is anticipated that the model performance can be improved by reducing the confusion between similar patterns.

In conclusion, accuracy is exceptionally critical in the biomedical field. While the models have achieved satisfactory results, further improvements and thorough testings are necessary for them to be reliable and trust-worthy, especially if they are to be integrated into disease diagnosing processes.

7. REFERENCES

[1] Christian Tchito Tchapga, Thomas Attia Mih, Aurelle Tchagna Kouanou, Theophile Fozin Fonzin, Platini Kuetche Fogang, Brice Anicet Mezatio, and Daniel Tchiotsop, "Biomedical image classification in a big data architecture using machine learning algorithms," *Journal of Healthcare Engineering*, vol. 2021, 2021.

[2] Hyunseok Seo, Masoud Badiei Khuzani, Varun Vasudevan, Charles Huang, Hongyi Ren, Ruoxiu Xiao, Xiao Jia, and Lei Xing, "Machine learning techniques for biomedical image segmentation: An overview of technical aspects and introduction to state-of-art applications," *Medical Physics*, vol. 47, no. 5, 2020.

[3] Yan yan SONG and Ying LU, "Decision tree methods: applications for classification and prediction," *Shanghai archives of psychiatry*, vol. 27, no. 2, 2015.

[4] R. Marée, P. Geurts, J. Piater, and L. Wehenkel, "Biomedical image classification with random subwindows and decision trees," in *Computer Vision for Biomedical Image Applications*, Y. Liu, T. Jiang, and C. Zhang, Eds., 2005.

[5] William S Noble, "What is a support vector machine?," *Nature Biotechnology*, vol. 24, 2006.

[6] Noramalina Abdullah, Umi Kalthum Ngah, and Shalihatun Azlin Aziz, "Image classification of brain mri using support vector machine," in *2011 IEEE International Conference on Imaging Systems and Techniques*, 2011.

[7] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen, "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, 2018.

[8] Zongwei Zhou, Jae Shin, Lei Zhang, Suryakanth Gurudu, Michael Gotway, and Jianming Liang, "Fine-tuning convolutional neural networks for biomedical image analysis: Actively and incrementally," in *2017 IEEE International Conference on Imaging Systems and Techniques*, 2017.

[9] Md. Zahangir Alom, Tarek Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Nasrin, Mahmudul Hasan, Brian Van Essen, Abdul Awwal, and Vijayan Asari, "A state-of-the-art survey on deep learning theory and architectures," *Electronics*, vol. 8, 2019.

[10] James D. McCaffrey, "Convolution image size, filter size, padding and stride," https://jamesmccaffrey.wordpress.com/2018/05/30/convolution-image-size-filter-size-padding-and-stride/, 2018.

[11] Kather JN, Krisam J, Charoentong P, Luedde T, Herpel E, and et al., "Predicting survival from colorectal cancer histology slides using deep learning: A retrospective multicenter study," *PLOS Medicine*, vol. 16, no. 1, 2019.