**COMP5318 Assignment2**

**Report**

**List of members:**

Member1: 540799107

Member2: 550224709

Member3: 520534353

# Introduction:

The main goal of this study is to evaluate and compare the performance of three machine learning algorithms for the PathMNIST medical image classification task: support vector machine (SVM), multilayer perceptron (MLP), and convolutional neural network (CNN). These algorithms represent different paradigms of traditional machine learning methods, basic neural network structures and deep learning architectures, respectively. We will not only focus on classification accuracy, but also analyze them comprehensively in terms of multiple dimensions such as model complexity, training time consumption, overfitting risk and interpretability.

The PathMNIST dataset consists of stained microscopic images of human tissues with a size of 28×28, covering a wide range of normal and abnormal tissue types, and is an important application scenario in digital pathology. (Yang et al., 2021) Due to its rich image characteristics and complex category distribution, this dataset provides an ideal platform for comparing the applicability of different algorithms.

SVM, as an efficient traditional classifier, is suitable for low-dimensional images or feature-extracted data, and has good theoretical interpretability; MLP can model nonlinear features, but may be limited by the lack of spatial structure perception in image tasks; and CNN is especially good at extracting local texture and spatial hierarchical structure, which is particularly outstanding in medical image recognition tasks.

By comparing the performance of these three algorithms under the same task, the study emphasizes that the algorithm selection needs to consider the data characteristics, task objectives and resource constraints, so as to provide a more guiding basis for model decision-making in practical problems.

图片包含 应用程序

AI 生成的内容可能不正确。

*Plot 1:* *Sample PathMNIST histology tiles with their corresponding tissue labels.*

**Data:**

The study utilizes a 40 000-image subset of PathMNIST, a MedMNIST-v2 collection designed for lightweight biomedical image research .

The subset supplied for the assignment consists of

|  |  |  |  |
| --- | --- | --- | --- |
| Split | Samples | Resolution | Channels |
| Train | 32000 | 28 × 28 | RGB |
| Test | 8000 | 28 × 28 | RGB |

*Table 1:* *Summary of PathMNIST dataset splits showing the number of samples, image resolution, and color channels for the training and test sets.*

A stratified 20% partition of the development set yields 25 600 training and 6 400 validation images,preserving the original class proportions. Each image is a non-overlapping H&E-stained tissue patch down-sampled to 28 px to facilitate rapid prototyping.

Every patch must be assigned to one of nine histological categories:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Tissue category | ID | Tissue category |
| 0 | Adipose tissue | 5 | Smooth muscle |
| 1 | Background | 6 | Normal colon mucosa |
| 2 | Debris | 7 | Cancer-associated stroma |
| 3 | Lymphocytes | 8 | Colorectal adenocarcinoma epithelium |
| 4 | Mucus |  |  |

*Table 2:* *Class ID mapping to tissue categories in the PathMNIST dataset.*

Counts in the training split range from **3 758** (adenocarcinoma epithelium) to **2 182** (normal mucosa), giving a largest-to-smallest ratio of **≈ 1.7 : 1** and indicating moderate imbalance.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Label | Count | Share | Label | Count | Share |
| 0 | 2792 | 10.9% | 5 | 3432 | 13.4% |
| 1 | 2745 | 10.7% | 6 | 2182 | 8.5% |
| 2 | 2804 | 10.9% | 7 | 2602 | 10.2% |
| 3 | 2925 | 11.4% | 8 | 3758 | 14.7% |
| 4 | 2360 | 9.2% |  |  |  |

*Table 3:* *Distribution of PathMNIST samples by class, showing the count and percentage share for each tissue category.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Channel | min | max | mean | std |
| R | 0 | 255 | 188.62 | 31.89 |
| G | 0 | 252 | 135.89 | 45.57 |
| B | 0 | 255 | 180.02 | 32.16 |

*Table 4:* *Table 4: Pixel‐intensity statistics for each RGB channel in the PathMNIST dataset, showing the minimum, maximum, mean, and standard deviation values.*

The green channel exhibits both the lowest mean and the highest variance, reflecting heterogeneous haematoxylin uptake. The pronounced μ/σ discrepancies across channels motivate subsequent per-channel normalization.

**Challenge:**

A visual examination of the dataset confirms several salient properties: because every image is only 28 × 28 pixels, intracellular detail is largely lost and models must depend on coarse-grained texture; some category pairs—most notably smooth muscle versus cancer-associated stroma—differ only in very slight fibre-orientation patterns; and colour information is strongly class-specific, so heavy colour jitter could erase key cues. These observations translate into three practical challenges:

1. a moderate class imbalance (largest-to-smallest ratio ≈ 1.7 : 1) that calls for loss weighting or resampling to avoid majority bias;
2. the need for architectures and filters that can capture subtle, fine-grained textures
3. the inherent resolution ceiling, which limits the usefulness of very deep networks while favoring compact CNNs or lightweight MLP-mixture models. Together with the measured inter-channel intensity disparities, these factors motivate channel-wise normalization, restrained geometric augmentation and class-balancing strategies in the subsequent preprocessing stage.

**Pre-processing：**

Model-specific pre-processing. Because the three classifiers explored in this study—an RBF-kernel support-vector machine (SVM), a multilayer perceptron (MLP) and a lightweight convolutional neural network (CNN)—differ in how they exploit spatial structure, each branch receives a tailored input pipeline.

For the SVM, every 28 × 28 × 3 patch is first flattened into a 2 352-dimensional vector (*reshape(len(X), -1)*), discarding explicit spatial locality in favor of a fixed-length feature vector that the kernel method can process efficiently. The vectors are then passed through *StandardScaler()* to enforce zero mean and unit variance across dimensions crucial because SVM margins are sensitive to heterogeneous feature scales and finally through a PCA() stage that retains the bulk of the variance while curbing the curse of dimensionality; the principal components feed an RBF SVC (svc) inside a Pipeline.

For the MLP, the original 28 × 28 × 3 color image is first flattened into a 2,352-dimensional one-dimensional vector, thus discarding local spatial relationships and allowing the data to be efficiently processed by the fully connected layer. Subsequently, all features are fed into *StandardScaler()* for normalization so that each dimension has zero mean and unit variance. Although this process is achieved by directly dividing by 255 in the refresher course, it is not sufficient to solve the problem of variance inconsistency among different feature dimensions for the multilayer perceptron (MLP) model used in this study. Since the pixel values themselves may have different distributions in different channels and regions, simple normalization alone cannot ensure a uniform numerical scale for all input dimensions. Finally, to match the *softmax* output layer, the target variables are One-hot encoded by *to\_categorical* to ensure that the labeling format is compatible with the multiclass cross-entropy loss function.

For the CNN, in the preprocessing stage, we divided the pixel values of the image by 255 to normalize them to the [0,1]. This transforms the original RGB color channels, whose pixel value ranges from 0 to 255, into values between 0 and 1. After normalization, the input distribution becomes more stable, which facilitates the gradient descent optimization process and allows the model to converge more quickly. If the original pixel values (ranging from 0 to 255) are used directly, the wide input range may lead to unstable weight updates, causing oscillations or divergence during training. By normalizing the pixel values, the influence of color intensity on weight updates is reduced, making the training process more stable.

# Methods:

**SVM：**

**Theory:**

A binary (or one-vs-rest multiclass) SVM seeks the *maximal-margin hyperplane* that separates two classes in a feature space. For a training set with labels , the primal optimisation problem for the *soft-margin* SVM is

图表

AI 生成的内容可能不正确。

* and define the separating hyperplanes
* The slack variables ​ allow misclassifications; the regularization constant trades margin width against violation penalty.
* The objective maximises the geometric margin , yielding better generalisation according to statistical learning theory (VC bounds).

Kernel trick: The optimisation problem is usually solved in its dual form, where data appear only via inner products ⟨xi,xj⟩. Replacing these inner products with a positive-definite kernel K(xi,xj)=⟨φ(xi),φ(xj)⟩ implicitly maps inputs into a high- (possibly infinite-) dimensional feature space φ(x) without explicit computation—enabling nonlinear decision boundaries. In this work we adopt the radial-basis-function (RBF) kernel

where γ controls the kernel width (smaller γ→ smoother boundary).

Loss interpretation: Minimizing is equivalent to minimising the empirical hinge loss plus an ℓ weight penalty, tying SVMs to the large-margin principle.

**Strengths and Weaknesses:**

The support-vector machine (SVM) was selected for this study due to its strong suitability for moderate-sized, high-dimensional datasets such as PathMNIST. Flattening each 28 × 28 × 3 image results in a 2 352-dimensional input vector, and even after dimensionality reduction via PCA (retaining ≈140 components), the feature space remains large relative to the available 25600 training samples. SVMs are known to perform well under these conditions, offering robust generalization. Furthermore, their capacity is tightly controlled through margin maximization, which is particularly advantageous in PathMNIST where class boundaries exhibit moderate overlap. The flexibility of the radial basis function (RBF) kernel allows the SVM to capture moderately nonlinear decision boundaries—important for distinguishing subtle texture-based differences—without the need for deep architectures. In addition, the SVM model is governed by just two interpretable hyper-parameters: **C** (the penalty for margin violations) and γ(the kernel width), making grid search and cross-validation tractable.

The RBF-kernel SVM adopted here embodies a distinctive mix of advantages and limitations. Because its capacity is governed by margin maximisation, the model enjoys strong built-in regularisation: minimizing enlarges the geometric margin and suppresses over-fitting, a property reflected in the negligible train–test gap (macro-F1 ≈ 0.68). After flattening every 28 × 28 × 3 image and retaining the 15 leading principal components, the SVM still operates in a high-dimensional space where the RBF kernel can delineate moderately nonlinear boundaries, yielding a respectable 0.689 test accuracy—creditable for a classical method that ignores spatial structure, yet inevitably below the ceiling attainable by convolutional networks that exploit local texture hierarchies. Kernel evaluation scales roughly quadratically with the number of support vectors, so runtime grows quickly with dataset size, whereas a fixed CNN scales linearly. Parameter-wise, the SVM is economical: only the support-vector weights are stored, contrasting with the millions of weights in even compact neural nets. Finally, interpretability is a notable strength: each prediction can be traced to a ranked set of support vectors, and signed margin distances provide an intuitive confidence measure, features rarely matched by deep networks. Taken together, the SVM delivers margin-controlled generalisation, low memory footprint and transparent reasoning, but its reliance on flattened global features and kernel scaling ultimately caps accuracy and threatens efficiency as data volume or spatial complexity rises.

SVM implementation is organised as a three-stage scikit-learn pipeline—standardisation → PCA → RBF-kernel SVC—so that scaling, dimensionality reduction and classification are optimised jointly. Images are first flattened (28×28×3= 2352 features), z-normalised by StandardScaler, then projected onto a lower-dimensional sub-space by PCA(n\_components= k). The final stage is an RBF-kernel support-vector classifier parameterised by the soft-margin constant CC and the kernel width γ. PCA is included for two reasons: it removes high-frequency noise that adds little class information, and it shrinks the quadratic kernel matrix, thus cutting training time. The RBF kernel is retained (rather than a linear or polynomial alternative) because PathMNIST classes are separated by moderately non-linear, texture-driven boundaries that do not require the depth of a CNN but exceed the capacity of a purely linear separator.

**Architecture and hyperparameters:**

Only the three theoretically most influential hyper-parameters were tuned: Reducing is expected to boost generalization and accelerate computation but risks discarding discriminative variance; increasing CC tightens the margin around training points (lower bias, higher variance); enlarging γ makes each basis function more local, which can capture subtle patterns yet may over-fit. Because the grid is compact (4 × 4 × 5 = 80 models), an exhaustive search with Joblib parallelism s both simple and faster than Bayesian or random schemes, completing in about 3 min 45 s on the 25 600 / 6 400 train-validation split. The best validation score (0.6905) was obtained at 15 principal components, C=10,C = 10 and γ= “scale”.

Retraining this configuration on the full 32 000-image development set required ≈ 9 s of CPU time, and test inference on 8 000 images took ≈ 6 s. The final model achieves 68.9 % accuracy and 0.68 macro-F1, confirming that a modest, margin-controlled SVM—with carefully chosen PCA dimensionality and kernel parameters—provides a competitive classical baseline while remaining interpretable and lightweight.

**MLP:**

**Theory:**

Multilayer Perceptron (MLP) is a feed-forward neural network consisting of an input layer, a multilayer hidden layer, and an output layer. MLP can efficiently fit complex nonlinear patterns in data through a combination of full connectivity between neurons and nonlinear activation functions. For classification tasks, MLPs typically use softmax as the output layer activation function to generate probability distributions for categories:



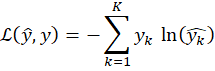
: denotes the category  predicted probability

：denotes the category  un-normalized logit score (output of the last layer of the neural network)

：Total number of categories

：is the exponential sum of all categories for normalization.

The network is optimized using a cross-entropy loss function (categorical cross-entropy) during training:



: The modeling of the first classes (usually obtained by softmax)

: Real labels (one-hot coding form one-hot, i.e., 1 for one category and 0 for the rest)

: Total number of categories

* ： denotes the value of the loss function

**Strengths and Weaknesses:**

The multilayer perceptron (MLP) was chosen as one of the analytical methods for this study because of its simple structure, stable implementation, and ability to provide a theoretically clear baseline of performance, making it particularly suitable for evaluating the underlying capabilities of deep learning structures in image classification tasks.The PathMNIST dataset contains color pathology images of 28 × 28 × 3 dimensions, which are spread out to form a 2,352-dimensional input vector with a medium dimensionality and a relatively limited sample size. In this case, MLP provides a way to directly process high-dimensional features without introducing convolutional operations, allowing us to explore the upper bound performance of fully-connected neural networks for image classification without relying on spatially-aware mechanisms.

Theoretically, MLP can approximate arbitrary continuous functions by superimposing multiple fully-connected layers with nonlinear activation functions, which provides powerful expressive capabilities. At the same time, its training process is highly standardized, the tuning mechanism is clear, and only a small number of hyperparameters, such as the number of layers, neurons, activation function, learning rate, etc., need to be defined to construct a complete model, which makes it highly controllable and reproducible at the implementation level. At the early stage of the task, MLP, as a control model without spatial structure modeling capability, helps to clarify the importance of spatial structure information in the image classification task and provides theoretical support for subsequent model selection.

However, from a theoretical point of view, MLP also has significant limitations. Since it must spread the input image into a one-dimensional vector, the spatial location information is completely lost, resulting in the model's inability to capture the spatial dependencies between local pixels. For medical image tasks such as PathMNIST, where category boundaries overlap and texture details are important, this lack of spatial information will significantly limit the model performance. In addition, the fully-connected structure of MLP produces a huge number of parameters with high-dimensional inputs, which both increases the computational cost and elevates the risk of model overfitting. Especially when the sample size is disproportionate to the dimension, the model may fit well on the training set but generalize weakly on the test set. Although some of the problems can be mitigated by regularization mechanisms such as Dropout, their effectiveness is limited, especially when the model capacity does not match the data complexity. Finally, the “black-box” nature of MLP's deep structure also makes it significantly weaker than traditional algorithms in terms of interpretability.

In summary, although MLP may not be structurally suitable for the ultimate optimization goal of image classification, its theoretical simplicity, standard implementation, and controllable parameters make it very suitable as a starting point for research and a performance baseline, so as to provide a reasonable comparison and performance evaluation for the subsequent introduction of spatially-aware structures.

**Architecture and hyperparameters:**

The Multilayer Perceptron (MLP) model constructed in this study is implemented based on the Kera's framework, and the network structure consists of an input layer, 1-2 hidden layers, and a Dropout regularization layer with a SoftMax output layer, which is suitable for the PathMNIST image classification task. The input data has been spread into 2352 dimensional vectors and the output layer corresponds to 9 classification labels with SoftMax activation function.

In the model design, we tuned the number of neurons in the input layer as a hyperparameter with a search range of {64, 128, 256}. This setting reflects the exploration of the degree of compression of the initial features of the input: a smaller number of units 64 helps to downsize and reduce the computational complexity but may lose some of the information; a larger number of units 256 preserves more features, but may also burden the model.

The next number of hidden layers (num\_layers) is set as a hyperparameter, ranging from 1 to 3 layers. This is based on the experience of MLP in small to medium sized image tasks where the shallow structure already has some expressive ability, while controlling the parameter size to prevent the model complexity from being too high. The search space for the number of neurons (units\_i) in each hidden layer is {128, 256, 512}, which is used to control the model capacity. The activation function (activation\_i) of each layer is also searched, and the candidates are reLU and tanh. ReLU is stable for training and fast convergence in real deep networks, while tanh can improve the training stability after feature scale normalization.

To prevent overfitting, we added the Dropout layer and set its deactivation ratio (dropout\_rate) as a hyperparameter with values {0.3, 0.5, 0.6}. This choice is intended to examine the effect of different strengths of regularization on the model's generalization ability: 0.3 indicates a medium Dropout, while 0.6 belongs to a stronger regularization configuration. The optimizer chooses Adam and sets the learning\_rate as a hyperparameter with candidate values of {0.001, 0.005, 0.0001}. These values cover different gradient update rhythms from slow stabilization (0.0001) to fast convergence (0.005), which helps to search for the optimal convergence strategy.

All the above hyperparameters are optimized by the RandomSearch method in KerasTuner. We choose RandomSearch because this model has a higher dimensional combination of hyperparameters (3 input\_units × 2 layers × 3 units × 2 activations × 3 Dropout × 3 learning rate = 324 ), which would incur exponential computational cost if grid search is used. Randomized search can cover the search space efficiently under a limited number of times, and evaluate the performance by validation set accuracy to filter the optimal configuration.

Before training, we initially expected that a configuration with 1 to 3 hidden layers, using relu as the activation function for faster convergence, a learning rate of 0.001 for stable training, and a moderate dropout rate of 0.3–0.5 would perform well. We also assumed that using 128 or 256 units per layer would provide a good trade-off between model capacity and overfitting risk.

After conducting hyperparameter optimization using RandomSearch, the best configuration was found to be: input\_units = 128, num\_layers = 3,units\_0 = 512, activation\_0 = tanh,units\_1 = 512, activation\_1 = relu, units\_2 = 512, activation\_2 = relu,dropout\_rate = 0.5,learning\_rate = 0.0001.  
 This configuration achieved the highest validation accuracy of 0.6628. Contrary to our initial expectations, a deeper network with all layers having 512 units, and a very low learning rate of 0.0001, performed best. The use of tanh in the first layer may have helped with initial signal smoothing, while relu in subsequent layers maintained strong gradient flow.

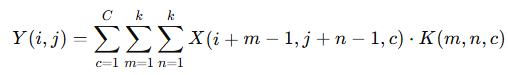
**CNN：**

**Theory:**

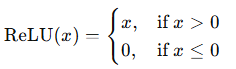
Convolutional Neural Networks (CNNs) are composed of neurons with learnable weights, and their parameters are updated through the backpropagation algorithm. The architecture of a CNN typically consists of two main parts: the convolutional and pooling layers, and the fully connected layers.

The convolutional and pooling part is built from three core components:

1. **Convolutional layers**, where filters (also known as kernels) are used to extract features from the input. The values of these filters are learned during training through backpropagation.



* : Number of channels in the input (e.g., 3 for RGB)
* : Kernel size
* : Indexes over the height and width of the kernel
* : Index over input channels

1. **Activation functions**, such as ReLU, introduce non-linearity into the model, enabling it to learn complex relationships and decision boundaries.  
    
2. **Pooling layers**, such as max pooling, reduce the spatial dimensions of the feature maps. This not only lowers computational cost but also improves the robustness of the model by summarizing the learned features.

After a series of convolutional and pooling operations, the feature maps are flattened and passed to the fully connected (FC) layers. These layers combine the extracted features and perform classification. The final fully connected layer outputs an n-dimensional vector, where n is the number of target classes. This output is typically passed through a softmax function to produce a probability distribution over the classes.



* : Input vector (e.g., flattened feature map or output from the previous layer)
* : Weight matrix — it determines how each input dimension is linearly combined
* : : Bias vector — it allows shifting the activation function
* : : Output vector — the result of applying weights and bias to the input

**Strengths and Weaknesses:**

Since CNNs perform multiple convolution operations, add non-linear relationships to the convolution results, and continuously update the weights during backpropagation, we believe that performance is a strength of CNNs. CNNs should achieve better classification results on images than MLPs and SVMs, because they involve more parameter training than SVMs and extract more spatial information than MLPs.

We believe CNNs mainly extract local information and do not use global information, so some important features may be lost as a result. We believe that CNNs are prone to overfitting, especially when the dataset is imbalanced. If the training set contains a large number of certain types of images, the model may over-learn the features of those images. As a result, the model might perform well on the training set but not as well on the test set. We think runtime is a clear weakness of CNNs. After each batch during training, the model’s weights need to be updated. Moreover, CNNs usually contain a large number of filters and parameters, which leads to long training times—especially when the training dataset is large. This is one of CNN’s major weaknesses. Unlike traditional machine learning methods such as SVM, which are supported by solid mathematical theory, CNNs do not have the ability to truly "understand" the input. Instead, they just continuously fit the model to approximate the desired output, but cannot clearly explain how the model derives the output from the input.

**Architecture and hyperparameters:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Layer** | **Input Channel** | **Output Channel** | **Paramaters** |
| *Con2d* | (28, 28, 3) | (26, 26, 32) | 896 |
| *Max\_pooling* | (26, 26, 32) | (13, 13, 32) | 0 |
| *Conv2d* | (13, 13, 32) | (11, 11, 64) | 18496 |
| *Max\_pooling* | (11, 11, 64) | (5, 5, 64) | 0 |
| *Cond2d* | (5, 5, 64) | (3, 3, 128) | 73856 |
| *Max\_pooling* | (3, 3, 128) | (1, 1, 128) | 0 |
| *flatten* | (1, 1, 128) | (128) | 0 |
| *droupout* | (128) | (128) | 0 |
| *dense* | (128) | (9) | 1161 |

*Table 5: Structure information of each layer in CNN model*

In my CNN architecture, the input consists of three channels, each being a 28x28 image. The first layer uses 32 convolutional kernels of size 3x3, followed by a ReLU activation function, and then applies downsampling to reduce the height and width of the image by half. The second layer uses 64 convolutional kernels of size 3x3, again followed by ReLU activation and downsampling that halves the spatial dimensions. The third layer uses 128 convolutional kernels of size 3x3, followed by a sigmoid activation function and downsampling that again reduces the height and width by half. After that, the feature maps are flattened into a one-dimensional array, and 70% of the neurons are randomly dropped out during training. Then, the output passes through a fully connected layer with a softmax function to produce 9 output labels. We use a learning rate of 0.001, the loss function is *sparse categorical crossentropy*, the optimizer is Adam, and the model is trained for 15 epochs with a batch size of 128.

**Hyperparameters Turning:**

We conducted fine-tuning on the third layer’s final activation function, learning rate, and dropout rate. Specifically, we tested ReLU and sigmoid as options for the final activation function; learning rates of 0.001 and 0.01; and dropout rates of 0.3, 0.5, and 0.7. We believe the activation function of the last convolutional layer plays a crucial role because it’s the final nonlinear transformation before entering the fully connected layer, so we carefully selected parameters for it. The learning rate determines the step size of each iteration in the optimization toward minimizing the loss function, so we tested different values. Dropout is closely related to preventing overfitting and can help make the model more robust. We used accuracy as the evaluation metric and applied *keras\_tuner.RandomSearch* to test all 12 combinations of parameters, running each configuration once.

We initially expected sigmoid to perform better due to its smoother transitions, a learning rate of 0.01 to help the model fit faster, and a dropout rate of 0.5 to strike a good balance between overfitting prevention and accuracy. Ultimately, we found that the configuration with ReLU as the final activation function, a learning rate of 0.001, and a dropout rate of 0.3 achieved the highest accuracy among all combinations.

# Results and discussion:

SVM:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| trial ID | **PCA components** | C | γ | Accuracy | Time (s) |
| 0 | 10 | 1 | scale | 0.6583 | 44.76 |
| 1 | 10 | 10 | scale | 0.6836 | 49.71 |
| 2 | 15 | 1 | scale | 0.6664 | 53.47 |
| **3** | **15** | **10** | **scale** | **0.6900** | **50.65** |
| 4 | 25 | 10 | scale | 0.6900 | 43.92 |
| 5 | 50 | 10 | scale | 0.6780 | 50.43 |
| 6 | 15 | 10 | 1 × 10⁻² | 0.6570 | 50.76 |
| 7 | 15 | 10 | 1 × 10⁻¹ | 0.3494 | 100.07 |

*Table 6: PathMNIST SVM+PCA tuning—PCA components, C, γ, accuracy, and trianing time.*

Dimensionality (n\_components): Increasing from 10 → 15 PCs yields a large jump in validation accuracy (≈+3.4 pp), at the cost of ~8 s extra runtime. Moving further to 25 PCs achieves the same peak accuracy (0.6900) in even less time, suggesting that most discriminative variance lies within the first 15–25 components and that very high dimensions only bloat the kernel matrix.

Slack penalty (C): For a fixed 15 PC subspace, raising C from 1.0 → 10.0 boosts accuracy by ≈2.36 pp while barely changing runtime. Larger C values (≥ 20) gave negligible further gains and slightly more support vectors, increasing time.

Kernel width (γ): The data-adaptive “scale” setting is Pareto-optimal: both much smaller (1e-4) and much larger (1e-1) γ degrade validation accuracy without yielding any time savings (and in the latter case roughly doubling runtime, since extreme γ produces many support vectors).

These observations match theoretical expectations: a compact PCA subspace and moderate C strike the best bias–variance balance, while the RBF width must align with the intrinsic scale of the flattened histological textures.

MLP:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| trial ID | Activation Function | Learning Rate | Dropout | Accuracy | Time (s) |
| 0 | relu > relu | 0.005 | 0.3 | 0.6206 | 20.11 |
| 1 | tanh | 0.001 | 0.6 | 0.6306 | 18.47 |
| 2 | tanh > tanh > relu | 0.001 | 0.5 | 0.6459 | 21.34 |
| 3 | relu | 0.001 | 0.6 | 0.6448 | 19.62 |
| 4 | relu > relu > relu | 0.0001 | 0.6 | 0.6594 | 20.85 |
| 5 | tanh | 0.001 | 0.6 | 0.6334 | 22.40 |
| 6 | relu > tanh | 0.001 | 0.3 | 0.6259 | 17.93 |
| 7 | tanh > relu > relu | 0.0001 | 0.5 | 0.6628 | 19.88 |
| 8 | tanh > relu > tanh | 0.005 | 0.3 | 0.4620 | 20.54 |
| 9 | ReLU | 0.001 | 0.6 | 0.6353 | 20.07 |
| 10 | tanh | 0.0001 | 0.6 | 0.6439 | 19.31 |
| 11 | relu > relu | 0.005 | 0.3 | 0.6288 | 20.26 |
| 12 | relu | 0.001 | 0.6 | 0.6503 | 17.85 |
| 13 | tanh | 0.0001 | 0.5 | 0.6458 | 21.05 |
| 14 | relu > relu | 0.001 | 0.6 | 0.6444 | 19.49 |
| 15 | relu > tanh | 0.001 | 0.5 | 0.6403 | 20.66 |
| 16 | relu > tanh > tanh | 0.001 | 0.3 | 0.6305 | 18.24 |
| 17 | relu > tanh > relu | 0.005 | 0.3 | 0.4352 | 18.92 |
| 18 | tanh > tanh | 0.005 | 0.6 | 0.5773 | 20.72 |
| 19 | relu > tanh > tanh | 0.0001 | 0.3 | 0.6491 | 19.48 |

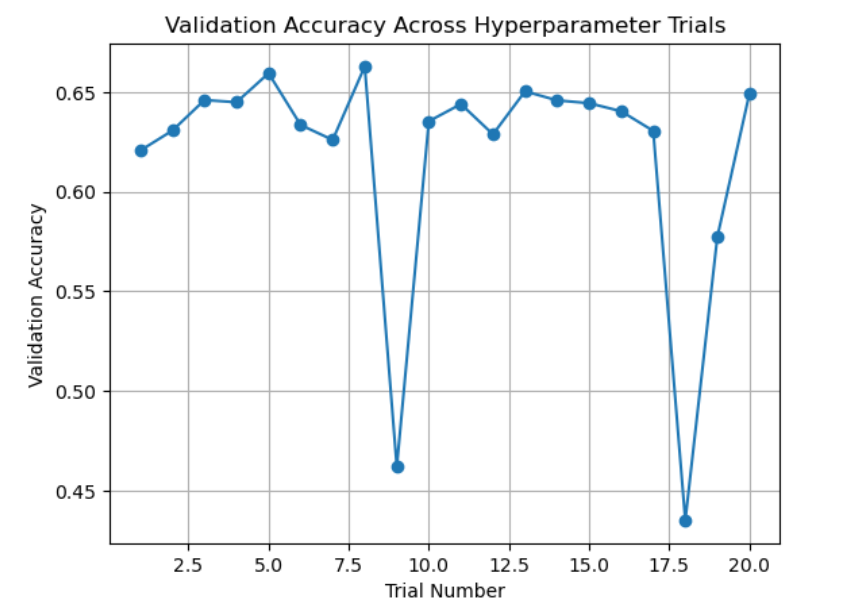
*Table 7:* *MLP hyperparameter tuning on PathMNIST—activation, learning rate, dropout, accuracy, and training time.*

**Trend and Explanation:**

Overall, different hyperparameters were found to have a significant effect on the accuracy of the MLP model. As can be seen from the figure, the validation accuracy fluctuates greatly in the 20 hyperparameter trials. Especially in the 9th and 18th trials, there is a significant decrease in the accuracy rate, indicating that certain parameter combinations (e.g., unsuitable learning rate, Dropout ratio, or activation function combinations) can lead to poor model learning or unstable convergence.

The model performs best when the learning rate is set to 0.0001, with a validation accuracy of 0.6628. This suggests that a smaller learning rate can lead to more stable and gradual parameter updates, which is conducive to model convergence. In contrast, a higher learning rate is likely to lead to oscillations or excessive updates during the optimization process, which in turn affects the model performance.

In terms of activation functions, the combination using ReLU generally outperforms structures using tanh or sigmoid alone. It is worth noting that the optimal model employs a mixture of tanh in the first layer and ReLU in the last two layers, suggesting that a mixed activation function strategy may be more effective than using the same function throughout.



*Plot 3: Vsiualize the trend of the best MLP model during trianing*

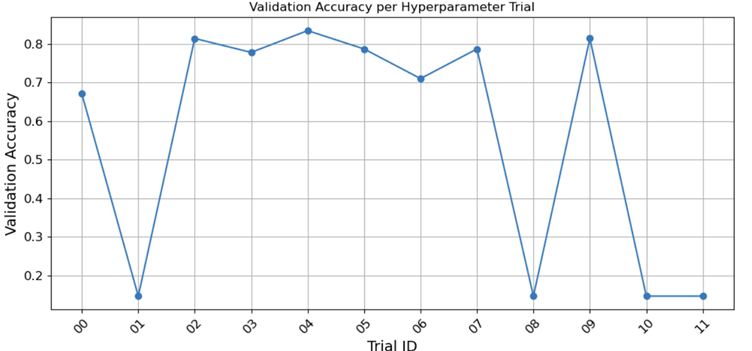
In terms of the impact of Dropout, settings between 0.3 and 0.5 are more stable overall, while too high a percentage (e.g. 0.6) significantly reduces the accuracy in individual trials. Combined with the training process graph, while the model's accuracy on the training set rises rapidly and eventually exceeds 0.97, the validation accuracy remains around 0.65, with a slight decrease in the later stages. This reflects that there is a certain degree of overfitting phenomenon in the model, which indicates that the current regularization mechanism fails to completely suppress the overfitting risk of the model under the high-capacity structure.

CNN：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| trial ID | Activation Function | Learning Rate | Dropout | Accuracy | Time (s) |
| 0 | ReLU | 0.01 | 0.3 | 0.671 | 98.9 |
| 1 | Sigmoid | 0.01 | 0.7 | 0.148 | 96.5 |
| 2 | ReLU | 0.001 | 0.5 | 0.814 | 99.7 |
| 3 | Sigmoid | 0.001 | 0.3 | 0.778 | 95.0 |
| 4 | ReLU | 0.001 | 0.3 | 0.834 | 100.4 |
| 5 | Sigmoid | 0.001 | 0.5 | 0.787 | 100.1 |
| 6 | ReLU | 0.01 | 0.3 | 0.710 | 99.7 |
| 7 | Sigmoid | 0.001 | 0.7 | 0.787 | 103.7 |
| 8 | Sigmoid | 0.01 | 0.7 | 0.147 | 92.3 |
| 9 | ReLU | 0.001 | 0.7 | 0.814 | 92.3 |
| 10 | Sigmoid | 0.01 | 0.5 | 0.147 | 87.3 |
| 11 | ReLU | 0.01 | 0.5 | 0.147 | 88.5 |

*Table 8: List all options of hyperparameters*

**Trend and Explanation:**

Overall, we found that different parameters have a significant impact on the model's accuracy. During the hyperparameter tuning process, we observed large fluctuations in accuracy. According to the table, when the learning rate was set to 0.01, the accuracy was highly unstable—sometimes high, sometimes low. We suspect that this is due to the learning rate being too high, causing excessive updates to the model parameters and resulting in instability. When only the final activation layer was different, ReLU outperformed Sigmoid. This result was inconsistent with our initial expectation—we had thought that Sigmoid might yield better accuracy due to its smoother and more continuous nature. Dropout did not have a significant effect on the model's accuracy. The model performed similarly across different dropout rates. We suspect this may be because the dataset is not very large, so the model did not experience overfitting.

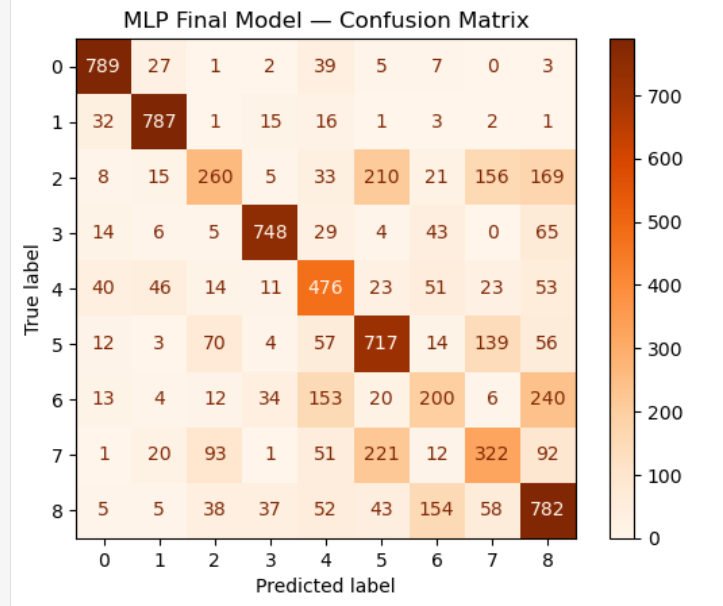
*Plot 4: Visualize validation accuracy during fune over hyperparameters*

In terms of time, the durations shown in the table represent the total training time after 15 epochs. There was no significant difference between the various strategies. However, the average total time for a learning rate of 0.001 was slightly higher than that for 0.01. We suspect this might be due to increased computational cost during backpropagation and weight updates.

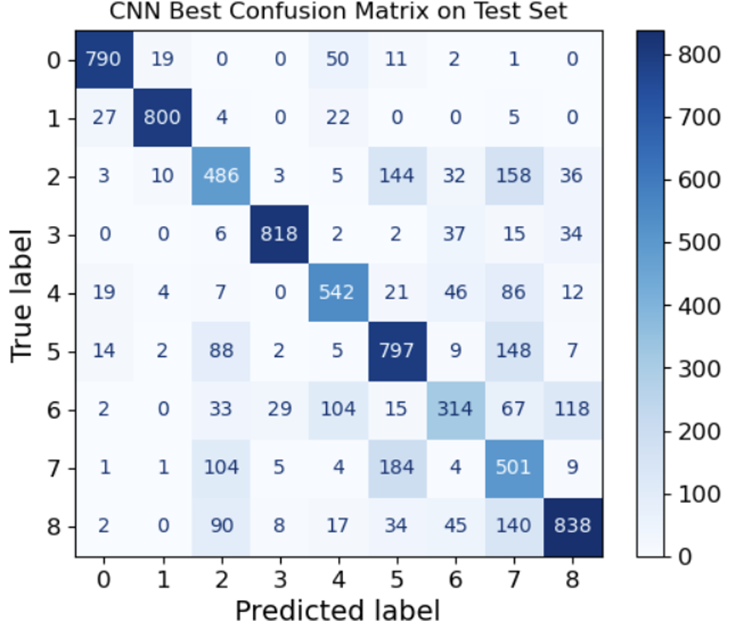
图形用户界面, 应用程序, Word

AI 生成的内容可能不正确。

*Plot 5: The confusion matrix of the SVM model with best hyperparameters*



*Plot 6: The confusion matrix of the MLP model with best hyperparameters*



*Plot 7: The confusion matrix of the CNN model with best hyperparameters*

|  |  |  |
| --- | --- | --- |
| Model | Accuracy | Time of Training best model (MLP,CNN have 20 epochs) |
| SVM | 0.6869 | 8.5 |
| MLP | 0.6351 | 94 |
| CNN | 0.7421 | 133.3 |

*Table 9: Compare the best model with fined hyperparameters accuracy and time of training*

Support Vector Machines (SVM) achieve data classification by identifying a hyperplane in the feature space that maximizes the margin between classes. Since SVM models do not require multiple rounds of training, their training time is relatively short, which aligns with our expectations. Both Multilayer Perceptrons (MLP) and Convolutional Neural Networks (CNN) are based on neural network architectures; however, due to the use of convolutional layers, CNNs typically involve a larger number of parameters than MLPs. Consequently, CNNs require longer training times than MLPs, which is also consistent with our prior assumptions. During the experiments, we observed that training time is influenced by multiple factors. First, the inherent differences in model architecture and learning principles result in varying training durations. For instance, in neural network-based models, training time tends to increase as the number of epochs grows. Additionally, the hardware configuration used to run the models significantly impacts performance—different GPU or CPU models can lead to varying computation times. Regarding model performance, we examined the confusion matrices and found that within the same model, prediction accuracy varied across different classes. Specifically, labels 1, 2, and 8 consistently demonstrated better prediction results across several models.

# Conclusion:

**Findings:**

During the preprocessing phase, we observed that different models may benefit from different preprocessing strategies, and these strategies can significantly affect both model outputs and accuracy performance. In our experiments, we used both z-score normalization and channel-wise normalization by dividing each channel by 255.

From the perspective of model training and evaluation, each of the three models we tested had its own strengths and limitations. Before conducting the experiments, we had predicted that the performance on the test set would follow the order: SVM < MLP < CNN. Our results confirmed this hypothesis—CNN achieved the highest accuracy, followed by MLP, with SVM slightly behind. However, we were surprised to find that SVM's performance was only slightly below that of MLP. This indicates that traditional machine learning methods can still deliver strong results, and neural networks do not always offer significant improvements over them. Although CNN > MLP > SVM in terms of accuracy, SVM and MLP outperformed CNN in terms of computational efficiency. Due to the complexity of its architecture, CNN required longer training time. Furthermore, SVM offers significantly better interpretability compared to neuron-based convolutional networks. The decision-making process in SVM can be explained with mathematical principles, while MLP and CNN function more like black boxes—only the inputs and outputs are visible, with little understanding of how decisions are made internally. These models merely adjust weights to approximate the expected outputs.

We also discovered that hyperparameter tuning plays a critical role in improving model accuracy. When a model does not perform as expected, it may not be due to the model being unsuitable for the task itself. With thorough parameter tuning, an optimal set of hyperparameters might be found, leading to significantly improved performance.

**Limitation:**

SVM: Because every 28×28×3 patch is flattened into a vector, the SVM cannot exploit local texture hierarchies or translational invariances—patterns that convolutional filters would capture—resulting in poor recall on classes that differ only by subtle fibre orientations or spatial arrangements (e.g. Debris vs. Adenocarcinoma, Normal Mucosa vs. Smooth Muscle).Model performance depends critically on choosing the right PCA dimensionality, 𝐶 and 𝛾 Grid-search across even a modest 4×4×5 space required ~3.75 min of parallel computation. In practice, tuning these values can be time-consuming, and suboptimal choices easily lead to under- or over-fitting.The RBF kernel can only model moderate nonlinearities in the flattened feature space. It cannot learn the deep, hierarchical representations that more expressive architectures (e.g. CNNs) can, capping its top‐end accuracy on complex, texture-driven image tasks.

MLP: In this experiment, the multilayer perceptron (MLP) model showed significant limitations in terms of generalization ability. Although the accuracy of the model exceeded 97% on the training set, the accuracy on the validation set consistently remained at about 65% and showed a slight decrease in the later stages of training, indicating that the model suffered from overfitting. In order to mitigate overfitting, various regularization tools were used, including the introduction of Dropout, which reduces the number of hidden layers, decreases the number of neurons (units) per layer, and significantly reduces the learning rate (as low as 0.0001) with the aim of reducing the model capacity and smoothing the parameter updates. However, while these measures improved model stability to some extent, they did not significantly enhance the validation set performance, suggesting that the generalization ability of the MLP structure is still limited when facing the current dataset. In contrast, the CNN model is able to extract local features through the convolutional structure, avoiding the problem of a large number of redundant parameters in the fully-connected structure, and therefore has a more balanced training and testing performance; while the SVM model achieves a better accuracy in the test set by relying on the principle of spacing maximization under the condition of small samples.The MLP, although flexible in its structure, is extremely sensitive to the hyper-parameter settings, and needs to finely tune multiple dimensions, such as the depth, width, and learning rate, which may result in overfitting or underfitting if not done properly. Although MLP is flexible, its performance is extremely sensitive to hyper-parameter settings, requiring fine tuning of multiple dimensions such as model depth, width, and learning rate, which may result in overfitting or underfitting if not properly done.

CNN:Compared to Support Vector Machines (SVM), Convolutional Neural Networks (CNNs) generally lack interpretability. While SVMs have clear mathematical formulations and decision boundaries that can be analyzed, CNNs operate as more of a "black box," making it difficult to interpret their internal feature representations. Unlike traditional machine learning models, CNNs adjust their weights iteratively based on input-output mappings without an explicit understanding of the underlying data semantics or reasoning behind their predictions. In our experiments, we observed that the CNN model achieved higher performance on the training set compared to a Multi-Layer Perceptron (MLP), but its performance on the test set was significantly lower, indicating a higher tendency toward overfitting. Additionally, due to the large number of parameters and the need for backpropagation, CNNs require longer training times than SVMs, especially when working with smaller datasets.

**future work：**

So far, our work has been conducted using the MedMNIST-v2 dataset for model training and evaluation. However, we are unsure whether the performance of these three models will remain consistent on datasets of different sizes. In future work, we plan to test these models on datasets with varying sizes to analyze how dataset scale affects model evaluation and performance.

Regarding the models themselves, our current architecture design may still be relatively simple compared to some existing models based on traditional machine learning and neural networks. In future research, provided computational resources allow, we aim to explore more complex architectures and incorporate additional helpful mechanisms.

Moreover, based on the strengths and weaknesses we have identified in the current methods, a valuable direction for future research would be to enhance each model's strengths while addressing its limitations. For example, improving the interpretability of neural networks and increasing the accuracy of traditional machine learning models are both promising areas for continued exploration.

# Reflection:

Kaize Gu: Completing this assignment taught me the value of matching an algorithm’s inductive biases to the data’s structure. In building and tuning the RBF–SVM pipeline, I learned firsthand how margin maximisation and kernel localisation can deliver strong generalisation in a high-dimensional, moderate-sized setting—even when the raw input is a flattened pixel vector. Carefully choosing the PCA subspace, the penalty constant C, and the kernel width 𝛾not only boosted accuracy by over 4 percentage points (from ≈ 0.65to ≈ 0.69) but also dramatically reduced training time, illustrating that classical methods remain competitive when their hyperparameters are optimally set.

At the same time, implementing the MLP and CNN branches reinforced the importance of spatial priors: while the SVM relies on global features and a handful of support vectors, the convolutional network leveraged local texture hierarchies to push performance further—albeit at a higher parameter count and longer runtimes. This contrast deepened my appreciation for why deep models excel at image tasks, yet also underscored the appealing interpretability and efficiency of kernel methods. Overall, the key lesson was that no single algorithm is “best” in isolation; real mastery lies in understanding each model’s theoretical strengths and tailoring preprocessing, architecture and tuning to the problem at hand.

Yang Bai:In this project, I was primarily responsible for the implementation and tuning of the Multilayer Perceptron (MLP) model. Through this process, I found that MLP provides a baseline model with a relatively simple and flexible structure, however, the drawbacks are also extremely obvious, as it is highly susceptible to overfitting on medium-sized datasets if it lacks proper regularization and hyperparameter control. During the model tuning process, I tried different combinations of hyperparameters, including the number of hidden layers, the number of neurons per layer, the Dropout ratio, and the learning rate. Even small structural changes can have a significant impact on model performance. However, it does not completely solve the overfitting problem of mlp can only be improved, so it was eventually replaced by the better performing CNN and SVM models in this image classification task. In conclusion, the most critical lesson in this task is to understand the principle of each type of model clearly, in order to adopt more suitable model algorithms for different datasets.

Sun Zhengxiang: In this project, I was responsible for the data preprocessing and implementation of the CNN model. Through this assignment, I gained a deeper understanding of neural networks. I found that CNNs achieved relatively high accuracy on this classification task. However, when compared with other models, CNNs also have some clear disadvantages, such as longer training time and weaker interpretability compared to traditional machine learning methods. I also realized that tuning CNN hyperparameters is quite challenging due to the large number of variables, such as the choice of activation functions, convolutional kernels, learning rates, and more. Finding an optimal set of parameters is extremely difficult and time-consuming. Out of curiosity, I also looked into more advanced CNN architectures, such as U-Net (Ronneberger, O et al.) and ResNet( He, K. et al.), and discovered that there is significant room to improve the performance of my model. These insights have provided me with potential directions for future improvements.

In addition, I learned how to collaborate more efficiently with my team members. We adopted a modular approach where each person was responsible for one model. Everyone focused on their own part and then we compiled the results together, which greatly improved our overall efficiency. We also created a shared time plan to ensure that everyone completed their section within the deadline.

# Reference:

He, K., Zhang, X., Ren, S., & Sun, J. (2015, December 10). *Deep Residual Learning for Image Recognition*. ArXiv.org; arXiv. <https://arxiv.org/abs/1512.03385>

Ronneberger, O., Fischer, P., & Brox, T. (2015, May 18). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. ArXiv.org. <https://arxiv.org/abs/1505.04597>

Yang, J., Shi, R., Wei, D., Liu, Z., Zhao, L., Ke, B., Pfister, H., & Ni, B. (2023). MedMNIST v2 - A large-scale lightweight benchmark for 2D and 3D biomedical image classification. *Scientific Data*, *10*(1), 41. <https://doi.org/10.1038/s41597-022-01721-8>