COMP4318
Assignment 2 Report

Group 169

1. Introduction....................4
1.1. Aim of the Study.................... 4
1.2. Importance of the Study....................4
1.2.1. Importance of the Dataset....................4
1.2.2. Importance of Algorithm Comparison & Suitability....................4
2. Data....................5
2.1. Description and Exploration....................5
2.1.1. Important Characteristics....................5
2.1.2. Exploration and Challenges/Difficulties....................5
2.2. Pre-Processing.................... 7
2.2.1. Techniques Used.................... 7
2.2.2. Techniques Considered But Not Unnecessary.................... 8
3. Methods....................10
3.1. Theoretical Description of Algorithms....................10
3.1.1. k-Nearest Neighbours (KNN)....................10
3.1.2. Multilayer Perceptron Neural Network (MLP).................... 10
3.1.3. Convolutional Neural Network (CNN).................... 10
3.2. Relative Strengths & Weaknesses of Algorithms....................11
3.2.1. Performance on Image Data....................11
3.2.2. Overfitting and Generalisation....................11
3.2.3. Runtime....................11
3.2.4. Number of Parameters....................12
3.2.5. Interpretability.................... 12
3.3. Architecture and Hyperparameters....................13
3.3.1. Search Method....................13
3.3.2. k-Nearest Neighbours (KNN)....................13
3.3.3. Multilayer Perceptron (MLP).................... 13
3.3.4. CNN Hyperparameters (CNN).................... 14
4. Results and Discussion.................... 16
4.1. Hyperparameter Tuning....................16
4.1.1. KNN Results.................... 16
4.1.2. KNN Discussion....................18
Runtime Trends....................20
Performance vs Runtime Trade-off....................21
4.1.3. MLP Results....................22
4.1.4. MLP Discussion.................... 31
Performance vs Runtime Trade-off.................... 33
Alignment with Expectations.................... 34
4.1.5. CNN Results.................... 34
4.1.6. CNN Discussion....................42

Runtime Trends......44
Performance vs Runtime Trade-off......45
Alignment with Expectations......46
4.2. Models......47
4.2.1. Result......47
4.2.2. Model Comparison and Discussion......48
Performance Comparison......48
Runtime Analysis......49
Types of Errors and Class-wise Performance......50
Alignment with Theoretical Expectations......52
Additional Trends and Interesting Observations......53
5. Conclusion......54
5.1. Main Findings......54
Performance and Accuracy......54
Runtime and Practicality......54
Interpretability......54
Limitations......54
5.2. Future Work Suggestions......55
Enhanced Hyperparameter Optimisation......55
Improved Feature Extraction and Dimensionality Reduction for KNN......55
Interpretability Techniques for CNN......55
Expanded and Diversified Dataset Evaluation......55
Real-time Deployment and Practical Testing......56
6. Reflection......57
6.1. Reflection for 520463341......57
6.2. Reflection for 520494068......57
7. References......58

1. Introduction

1.1. Aim of the Study

The aim of this study is to evaluate and compare the performance of different machine learning algorithms on a medical image classification task. Specifically, we implement three supervised classification models: k-Nearest Neighbours (KNN), a Multilayer Perceptron (MLP), and a Convolutional Neural Network (CNN), using a subset of the PathMNIST dataset. Through systematic design, tuning, and evaluation of each algorithm, we aim to highlight their respective strengths, limitations and overall suitability when applied to RGB biomedical images.

1.2. Importance of the Study

1.2.1. Importance of the Dataset

The importance of the PathMNIST dataset lies in both its biomedical significance and its practical utility for machine learning. Originating from histopathological images of human tissues, this dataset supports clinically relevant objectives such as distinguishing between normal and abnormal tissue types, which is an essential step in early diagnosis, treatment planning, and pathology research. As such, it offers a real-world context for evaluating the applicability of machine learning in healthcare.

In addition to its medical relevance, PathMNIST has several technical properties that make it ideal for algorithmic benchmarking. Its low-resolution RGB images (28x28) reduce computational demands, allowing for faster experimentation while still preserving meaningful texture and colour features. The dataset includes clearly labeled samples across nine tissue classes, and exhibits relatively low missing data, enabling fair and controlled model comparison. These characteristics support its role as a compact yet challenging benchmark for exploring model generalisation, overfitting, and optimisation techniques.

1.2.2. Importance of Algorithm Comparison & Suitability

Comparing different types of models/algorithms allows us to highlight the trade-offs between interpretability, scalability, and accuracy. Understanding these differences is essential for choosing suitable models in real-world biomedical applications, especially in settings constrained by time, data availability, or computational resources.

2. Data

2.1. Description and Exploration

2.1.1. Important Characteristics

The dataset used in this study is a subset of the PathMNIST dataset from the MedMNIST v2 collection [1], which contains images of histological slides sampled from nine classes of colorectal tissue. Each image is labeled with a class ID from 0 to 8, corresponding to specific tissue types. Although class names are not included in the assignment-provided version, here are the tissue types that each of the class ID's correspond to:

- 0: adipose
- 1: background
- 2: debris
- 3: lymphocytes
- 4: mucus
- 5: smooth muscle
- 6: normal colon mucosa
- 7: cancer-associated stroma
- 8: colorectal adenocarcinoma epithelium [2]

The dataset comprises:

- Training set: 32,000 images (X_train: (32000, 28, 28, 3), y_train: (32000,))
- Test set: 8,000 images (X_test: (8000, 28, 28, 3), y_test: (8000,))

Each image is 28x28 pixels has 3 color channels (RGB) and is stored as an 8-bit integer array (uint8), with pixel intensities ranging from 0 to 255.

2.1.2. Exploration and Challenges/Difficulties

We conducted several analyses to understand the structure and challenges of the dataset:

- **Class Distribution:** As shown in Figure 1, while the dataset is not heavily imbalanced, some class counts vary significantly (e.g., class 6 has ~2700 samples, class 8 has ~4700). This suggests that models should be evaluated using class-sensitive metrics such as F1-score.
- **Pixel Intensity Histograms:** Figure 2 shows pixel value distributions across RGB channels. Red and blue intensities tend to be higher, consistent with the H&E staining used in pathology slides. The distributions are skewed, but show no major anomalies such as extreme outliers or zero-dominant artifacts.
- **Sample Images:** Figure 3 presents a grid of randomly selected images from each class. The images show clear inter-class variability in color texture and structure but also highlight intra-class variance and similarity across classes (e.g., class 3 vs. class 4), which may introduce classification difficulty.

These insights informed the subsequent preprocessing decisions, ensuring both theoretical grounding and practical justification.

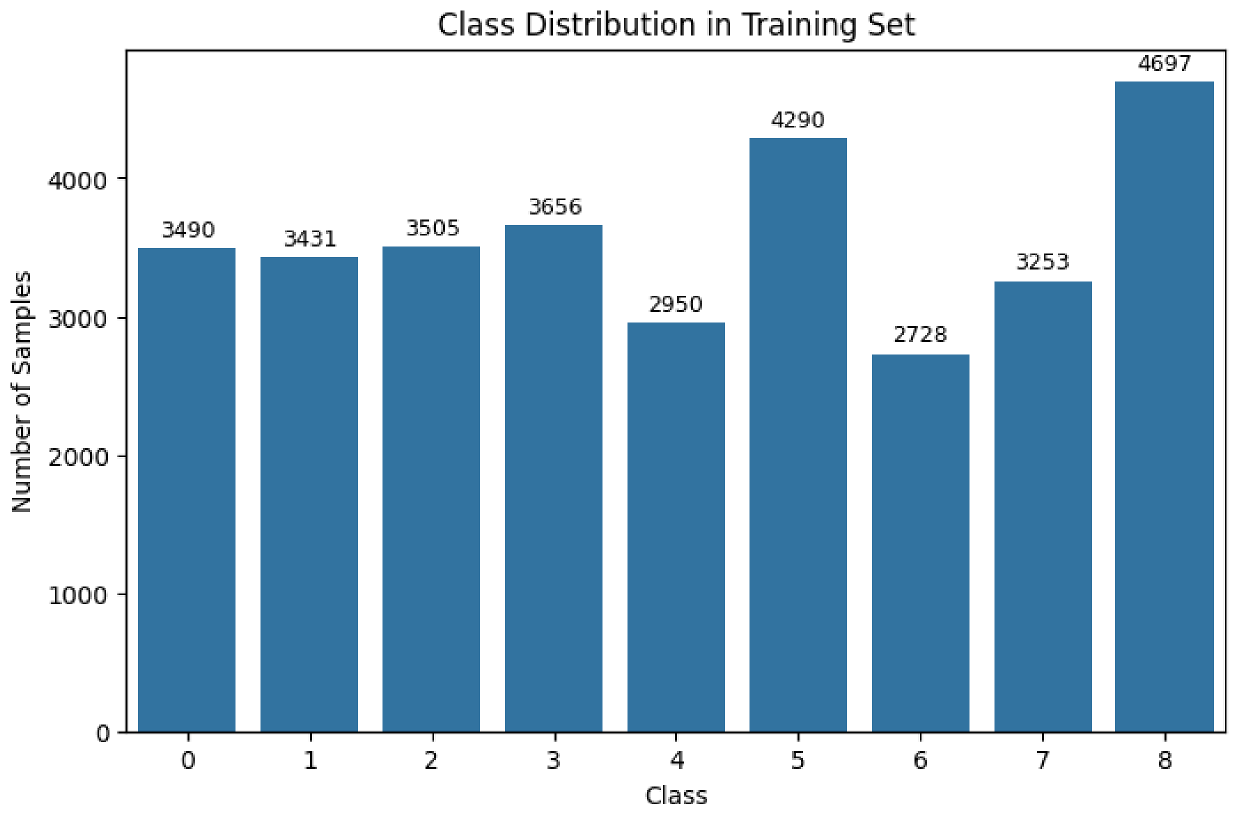


Figure 1

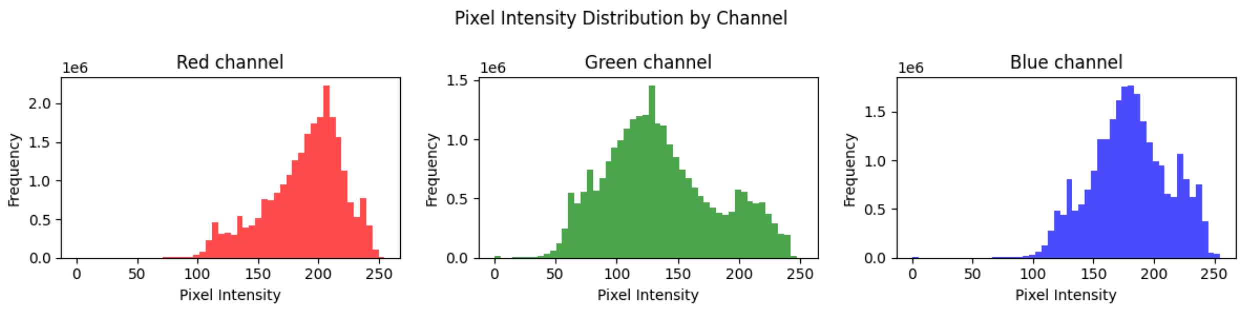


Figure 2

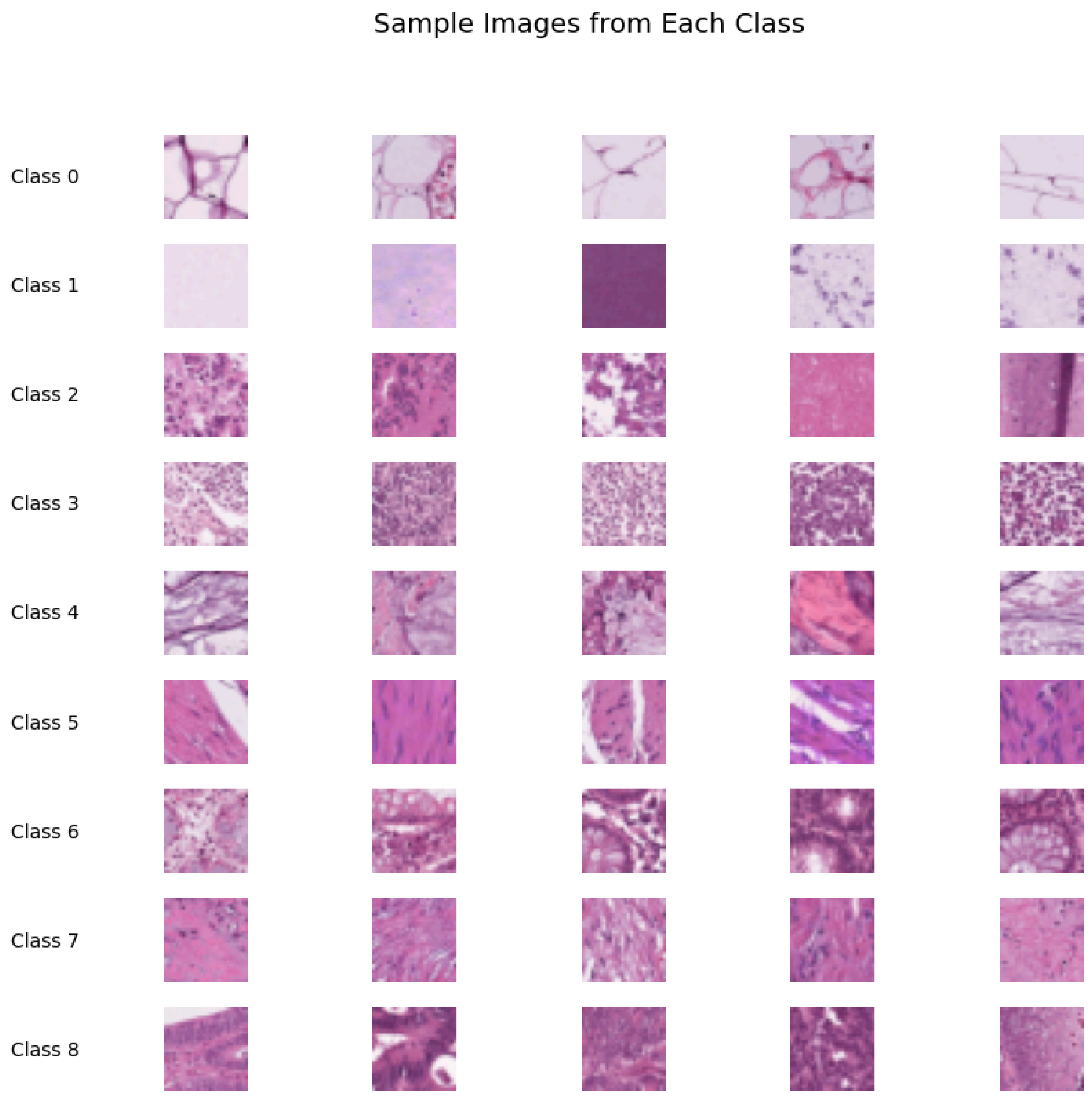


Figure 3

2.2. Pre-Processing

2.2.1. Techniques Used

Based on the findings in our data exploration, and supported by lecture content [3], we applied the following preprocessing techniques:

1. **Normalisation of Pixel Values (Min-Max Scaling to [0, 1])**
   All RGB pixel values were normalised from the original range of [0, 255] to [0, 1] by dividing each pixel value by 255. This transformation is essential for many machine

learning algorithms, especially those that rely on distance metrics or gradient-based optimization (e.g. KNN, MLP, CNNs). Normalization ensures that no single feature (color channel) disproportionately affects the model's performance and contributes to faster convergence in neural networks [3].

This choice was supported by insights from the pixel intensity histograms (see Figure 4), which showed skewed distributions across channels. Normalization brings all features to a common scale without distorting their relative differences.

2. **Stratified Subsetting**
   Given the size of the full training dataset (~31,000 samples as shown in Section 2.1.1 from initial exploration), training deep models such as CNNs across all combinations of hyperparameters was computationally expensive. To address this, a stratified sampling approach was used to extract a balanced subset of 5,000 examples. StratifiedShuffleSplit from sklearn was employed to ensure that the relative distribution of the 9 tissue classes was preserved, maintaining class representation fidelity during model training.

   This decision was based on visual analysis of the class distribution (see Figure 1), which revealed mild imbalance but not enough to necessitate oversampling or weighting. Stratified subsetting avoids introducing bias into training and still provides computational efficiency for grid search over multiple model architectures.

2.2.2. Techniques Considered But Not Unnecessary

The following pre-processing techniques were considered, but not used for the corresponding reasons.

1. **One-Hot Encoding**
   One-hot encoding was considered for the class labels but ultimately deemed unnecessary. This is because the neural network models (MLP and CNN) were compiled using the sparse_categorical_crossentropy loss function, which expects integer class labels rather than one-hot encoded vectors. Using this loss function simplifies the pipeline while preserving classification accuracy. As there was no theoretical or practical advantage in applying one-hot encoding under this configuration, it was omitted.

2. **Resizing**
   Image resizing was also considered, especially since resizing is commonly used to reduce computational cost or unify image dimensions. However, the PathMNIST dataset already provides images at a uniform and compact resolution of 28x28 pixels, which is well-suited for efficient training of deep learning models. Resizing down further would risk losing critical histological detail, while resizing up would increase computational cost without adding meaningful new information. Therefore, no resizing was applied.

3. **Grayscale Conversion**
   Converting images from RGB to grayscale can reduce dimensionality and simplify models, which is beneficial in some domains. However, in medical imaging tasks such as tissue classification, color features often carry important diagnostic information (e.g. staining patterns, tissue contrast). Exploratory analysis of sample images showed that class distinctions were visually supported by differences in color. As a result, retaining the full RGB channels was essential for preserving the biological signal, and grayscale conversion was not used.

4. **Augmentation or Rebalancing**
   Data augmentation and class rebalancing techniques (e.g. oversampling, SMOTE, weighted loss functions) were considered but ultimately not implemented. The class distribution in the original dataset (Figure 1) showed moderate imbalance but not severe enough to justify these interventions. Moreover, stratified subsetting was used to preserve class proportions in the reduced training set, mitigating the risk of bias. Given the project's scope and hardware constraints, the choice was made to prioritize efficient evaluation over aggressive rebalancing.

3. Methods

3.1. Theoretical Description of Algorithms

3.1.1. k-Nearest Neighbours (KNN)

KNN is a non-parametric, instance-based algorithm that classifies a new sample based on the majority label among its k closest training samples in feature space, typically using a distance metric such as Euclidean or Manhattan distance. It performs no explicit training, instead storing the dataset and making predictions at inference time. This makes KNN simple and interpretable, yet computationally intensive on large datasets [4] [5].

KNN was selected from the first six weeks as a baseline classifier due to its interpretability and minimal tuning requirements. Its behavior also provides a useful contrast to neural methods, especially when input images are flattened into feature vectors. Given the 28x28x3 input dimensionality, it serves to illustrate how distance-based methods perform on raw pixel data without spatial awareness. [4] [5].

3.1.2. Multilayer Perceptron Neural Network (MLP)

MLPs are a class of feedforward neural networks that consist of an input layer, one or more hidden layers, and an output layer. Each neuron applies a weighted sum followed by a nonlinear activation function (e.g., ReLU), enabling the network to learn complex, non-linear functions. The network is trained via backpropagation and gradient descent to minimise a loss function such as cross-entropy. [6] [7]

In the context of this image classification task, MLPs serve as a foundational deep learning model, offering insights into how deep representations and multiple layers affect learning. They are well-suited for structured data but less effective for images unless combined with careful preprocessing, as they do not inherently capture spatial hierarchies. [6] [7]

3.1.3. Convolutional Neural Network (CNN)

CNNs are a type of deep neural network specifically designed for image data. Unlike MLPs, CNNs incorporate convolutional layers that learn spatial hierarchies of features through the use of local receptive fields, weight sharing, and pooling operations. These properties make CNNs more parameter-efficient and robust to translation and distortion in image data. [8] [9]

CNNs have become the dominant approach in image classification tasks due to their ability to automatically learn discriminative features. In this study, CNNs are used to demonstrate the effectiveness of architectures that exploit the 2D structure of image inputs and are expected to outperform both KNN and MLP in terms of classification accuracy and generalisation. [8] [9]

3.2. Relative Strengths & Weaknesses of Algorithms

3.2.1. Performance on Image Data

CNNs are generally the best performers on image data because they exploit spatial locality through the use of convolutional filters. These filters learn to detect visual patterns such as edges, textures, or shapes and preserve spatial hierarchies by applying local receptive fields. Pooling layers reduce dimensionality while maintaining relevant features, making CNNs highly effective at learning hierarchical representations [9]. In contrast, MLPs treat the input image as a flat vector, ignoring spatial structure, which often leads to lower accuracy and redundant weight usage. KNN does not perform feature learning; its success relies purely on the quality of the distance metric in the raw input space. While KNN can work well for simple image recognition tasks, its inability to learn abstract features typically limits its performance on complex image data like histopathological slides [5].

The PathMNIST dataset presents considerable variation in texture and colour across classes (e.g., connective tissue vs. cancer), which benefits from feature extraction that CNNs are specifically designed for. MLPs and KNN are less equipped to capture these spatial nuances. Thus, both in theory and in relation to this dataset, CNN has the highest performance, followed by MLP and finally KNN.

3.2.2. Overfitting and Generalisation

MLPs and CNNs both risk overfitting due to their large number of trainable parameters. However, CNNs are more robust to overfitting than MLPs due to weight sharing, which drastically reduces the number of parameters and acts as an implicit regulariser [10]. Dropout layers further mitigate overfitting in both architectures. KNN does not overfit in the traditional sense since it stores training data instead of learning parameters. However, it is prone to high variance and noise sensitivity, particularly when the data contains irrelevant features or overlapping classes [11].

Given that the PathMNIST dataset has modest class imbalance and noisy visual boundaries, models that can learn generalisable patterns while suppressing noise, like CNNs with dropout, are more suitable. KNN's lack of abstraction makes it more sensitive to borderline or ambiguous samples. Therefore, again both in theory and in relation to this dataset, CNN is the best for overfitting and generalisation, followed by MLP and KNN.

3.2.3. Runtime

From a computational standpoint, CNNs and MLPs require significant training time due to gradient-based optimisation over multiple epochs. CNNs are particularly computationally intensive due to their layered structure and convolutional operations. However, once trained, their inference speed is efficient. In contrast, KNN has negligible training time but very expensive inference, as it must compute distances to all training samples at prediction time. This makes KNN less scalable to large datasets, especially in high-dimensional feature spaces like flattened images [5]. Thus in theory, the order of runtime efficiency would be MLP, then CNN, then KNN.

However, in the PathMNIST dataset, this theoretical ordering did not hold: MLP achieved the fastest total runtime, but KNN was faster than CNN. This reversal is due to the relatively small training subset (5000 samples), which makes KNN's test-time distance calculations computationally feasible. Meanwhile, CNN's multi-layer structure and extended training time across 10 epochs led to the highest overall runtime. Thus, although CNNs are generally more efficient at inference when deployed at scale, the training overhead in small datasets can outweigh this advantage, allowing simpler algorithms like KNN to be more time-efficient in practice.

3.2.4. Number of Parameters

MLPs typically have a very large number of parameters due to fully connected layers, which assign a unique weight to each input-feature-to-node connection. For image data, this leads to redundancy and overparameterisation, since spatial structure is ignored. In contrast, CNNs mitigate this issue through convolutional layers that share weights across spatial regions. This architectural design dramatically reduces the number of parameters while preserving the ability to extract meaningful features [10].

KNN does not have any trainable parameters at all. It simply stores examples instead of learning weights. This can be viewed as having zero model parameters, though the trade-off is computational cost and memory during inference.

In the PathMNIST context, the high-dimensional pixel input (28x28x3 = 2352 features) means that MLPs require millions of parameters in just one dense layer, while CNNs can achieve comparable or better results with far fewer due to convolutional weight sharing.

Thus, both in theory and in the case of this study, KNN has the least number of parameters, followed by CNN then MLP which has the most.

3.2.5. Interpretability

KNN is the most interpretable model among the three, as predictions are made based on a simple, human-understandable rule: majority vote among the nearest neighbours, so it is straightforward to inspect which samples influence the prediction. MLPs and CNNs, on the other hand, operate as black-box models. While tools such as saliency maps can be used to interpret CNN predictions by highlighting influential regions of the input, these techniques offer only partial insight and are often difficult to validate [12].

For the medical context of PathMNIST, interpretability is valuable for understanding and justifying classifications. While CNNs outperform in accuracy, KNN provides more transparent reasoning which is a trade-off to consider in biomedical applications. Therefore, both in theory and in the case of this dataset, KNN has the highest interpretability, followed by CNN then MLP.

3.3. Architecture and Hyperparameters

3.3.1. Search Method

We used grid search with 3-fold cross-validation as our hyperparameter tuning strategy for all three models. This method exhaustively evaluates all combinations of specified parameter values using stratified data splits, ensuring fair and consistent evaluation. Cross-validation helps prevent overfitting to a single train-test split, especially with relatively small datasets like our 5000-sample subset, and is widely recommended for robust model selection. For these reasons, as well as the fact that this approach is supported in the lectures [13] and common practice in applied machine learning [14], we chose to use it.

3.3.2. k-Nearest Neighbours (KNN)

KNN serves as a simple, interpretable baseline. Although it does not learn parameters from the data, its performance depends heavily on hyperparameter choices and distance metrics. The following hyperparameters were tuned:

- n_neighbors: [1, 3, 5, 7, 9] – This determines how many training samples are considered for the majority vote. Smaller values increase sensitivity but risk overfitting, while larger values smooth the decision boundary. Odd values were used to avoid tie-breaking in classification decisions.
- weights: [‘uniform’, ‘distance’] – With 'uniform', all neighbours contribute equally. ‘Distance’ weighting emphasizes closer neighbours, potentially improving performance in dense regions. Including both allows us to test whether the model benefits from weighting local structure more heavily, particularly important in high-density clusters in the PathMNIST dataset.
- p: [1, 2] – With p=1 as Manhattan and p=2 as Euclidean distance. This controls the shape of the distance metric. Euclidean is more sensitive to outliers in high-dimensional space, while Manhattan can be more stable. Evaluating both helps determine the best fit for flattened image data, where features may be correlated across pixels.

We flattened the image data into 2D arrays as KNN requires fixed-length feature vectors and does not exploit spatial structure. Despite this limitation, we included KNN to provide a non-parametric, interpretable baseline.

3.3.3. Multilayer Perceptron (MLP)

The architecture chosen for MLP consisted of two fully connected hidden layers with ReLU activation, followed by a softmax output layer. This balances model complexity and training stability for moderately sized image datasets, hence why it was chosen. The hyperparameters tuned were:

- hidden_units1: [256, 512] – The number of neurons in the first layer. More units can capture more complex patterns, but increase overfitting risk. We chose powers of two in a moderate range to capture increasing feature abstraction while controlling the model size.

- hidden_units2: [128, 256] – The second hidden layer's size, controlling depth and feature abstraction. These values were chosen to mirror the scaling used in the first layer and assess diminishing returns in capacity.
- learning_rate: [1e-2, 1e-3] – Affects how quickly the model updates weights. Higher values train faster but risk overshooting; lower values provide stability. This is a typical starting range.
- optimizer: ['adam', 'rmsprop'] – Both optimizers are popular and adapt learning rates during training, but use different strategies. Adam generally converges faster for sparse gradients. Testing both helps find the best match for this task's loss landscape.
- batch_size: [32, 64] – Affects gradient update frequency. Smaller batches offer more updates but are noisier. Both are commonly used batch sizes that balance efficiency and performance.
- epochs: [5] – We fixed this to limit training time during tuning. In pilot tests, early convergence was observed within 5 epochs on this subset, justifying this cap.

Dropout layers (rates: 0.3 and 0.2) were also added between dense layers to regularize the network by randomly disabling neurons during training. This combats overfitting and has been shown to improve generalization [10], hence its inclusion. ReLu was also used for activation due to its efficiency and ability to mitigate vanishing gradients.

3.3.4. CNN Hyperparameters (CNN)

The CNN architecture consisted of two convolutional blocks with ReLU activation and max pooling, followed by a dense output layer. The hyperparameters tuned were:

- num_filters: [32, 64] – Controls the number of filters in the first convolutional layer. More filters allow learning a wider variety of features but increase computation. 32 is a typical baseline for small images, 64 enables capturing finer texture or cellular-level features.
- dropout_rate: [0.3, 0.5] – Prevents co-adaptation of neurons and reduces overfitting. Evaluates the trade-off between regularization and retained capacity. A higher rate provides stronger regularization, especially important in deep models with high parameter counts.
- learning_rate: [1e-3, 5e-4] – As with MLP, this governs convergence stability. 0.001 is a widely accepted default; 0.0005 allows finer convergence control.
- optimizer: ['adam', 'rmsprop'] – Evaluated for their adaptive learning behaviour, and again as with MLP both are popular.
- batch_size: [32, 64] – Similar justification as above.
- epochs: [10] – Increased from MLP due to CNNs typically requiring more time to converge.

The convolutional layers exploit the 2D spatial structure of the input data, making them ideal for image data. Pooling layers reduce dimensionality and increase translational invariance, which is beneficial in medical image classification like PathMNIST [9]. This architecture was based on canonical designs in image classification and was kept deliberately shallow due to the small input size (28x28) and limited subset size.

4. Results and Discussion

4.1. Hyperparameter Tuning

4.1.1. KNN Results

param_n_neighbors	param_p	param_weights	mean_fit_time
1	1	uniform	0.053720951
1	1	distance	0.00888896
1	2	uniform	0.008092562
1	2	distance	0.009957552
3	1	uniform	0.008464495
3	1	distance	0.007561048
3	2	uniform	0.009513696
3	2	distance	0.007041057
5	1	uniform	0.008647919
5	1	distance	0.007201989
5	2	uniform	0.009710073
5	2	distance	0.008954525
7	1	uniform	0.00838995
7	1	distance	0.009314617
7	2	uniform	0.009479682
7	2	distance	0.009220997
9	1	uniform	0.009862582
9	1	distance	0.006806374
9	2	uniform	0.007895311
9	2	distance	0.009572188

Mean fit time across different hyperparameters values

param_n_neighbors	param_p	param_weights	mean_test_score
1	1	uniform	0.3129985
1	1	distance	0.3129985
1	2	uniform	0.27599882
1	2	distance	0.27599882
3	1	uniform	0.29299842
3	1	distance	0.30379806
3	2	uniform	0.2561993
3	2	distance	0.26899878
5	1	uniform	0.29719878
5	1	distance	0.30239834
5	2	uniform	0.26819978
5	2	distance	0.27160042
7	1	uniform	0.29919958
7	1	distance	0.3037989
7	2	uniform	0.27300002
7	2	distance	0.27680046
9	1	uniform	0.30099994
9	1	distance	0.30659978
9	2	uniform	0.27879994
9	2	distance	0.27840038

Mean Accuracy across different hyperparameter values

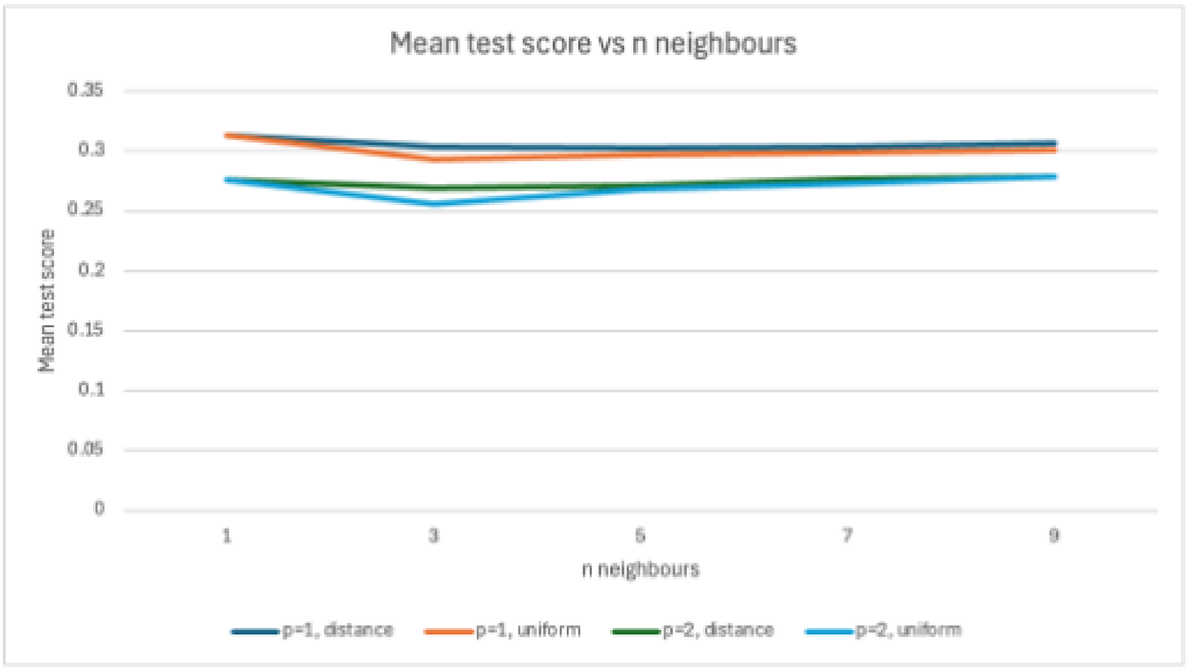
param_n_neighbors	param_p	param_weights	mean_score_time
1	1	uniform	2.54710809
1	1	distance	2.37219675
1	2	uniform	0.1790603
1	2	distance	0.19088856
3	1	uniform	2.29015533
3	1	distance	2.27804963
3	2	uniform	0.1784633
3	2	distance	0.18781638
5	1	uniform	2.26055463
5	1	distance	2.35185766
5	2	uniform	0.23823635
5	2	distance	0.20536613
7	1	uniform	2.34714866
7	1	distance	2.4561948
7	2	uniform	0.18319964
7	2	distance	0.17409722
9	1	uniform	2.266891
9	1	distance	2.25624069
9	2	uniform	0.18139958
9	2	distance	0.17507855

Mean score time across different hyperparameter values

4.1.2. KNN Discussion

The hyperparameter tuning results for K-Nearest Neighbours (KNN) revealed several noteworthy trends regarding model performance and runtime. The key hyperparameters tuned were the number of neighbours (n_neighbors), the distance metric (p), and the weighting scheme (weights). The performance metric used was mean cross-validation accuracy, while both mean fit time and score time were recorded to assess computational efficiency.

Accuracy Trends

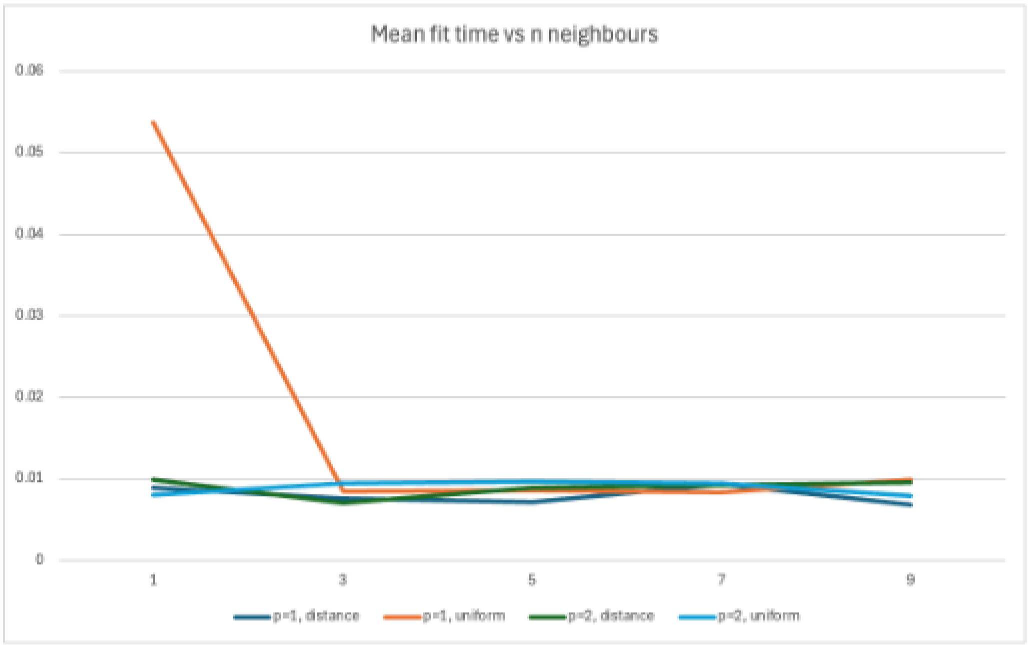


Overall, the highest classification accuracy (31.3%) was achieved using n_neighbors = 1, p = 1 (Manhattan distance), and both uniform and distance weighting schemes. This result aligns with the known behaviour of KNN, where smaller n_neighbors values tend to closely fit the training data, often leading to higher accuracy in datasets with distinct class boundaries. The accuracy across different hyperparameter settings remained relatively stable, with most configurations achieving comparable performance around 31%. Notably, both uniform weighting schemes with p = 1 and p = 2 exhibited a slight dip in accuracy at n_neighbors = 3, before recovering at higher n_neighbors values. This deviation may be attributed to the increased inclusion of more diverse neighbours at n = 3, which could have introduced conflicting class information, particularly under uniform weighting where all neighbours are treated equally regardless of their proximity.

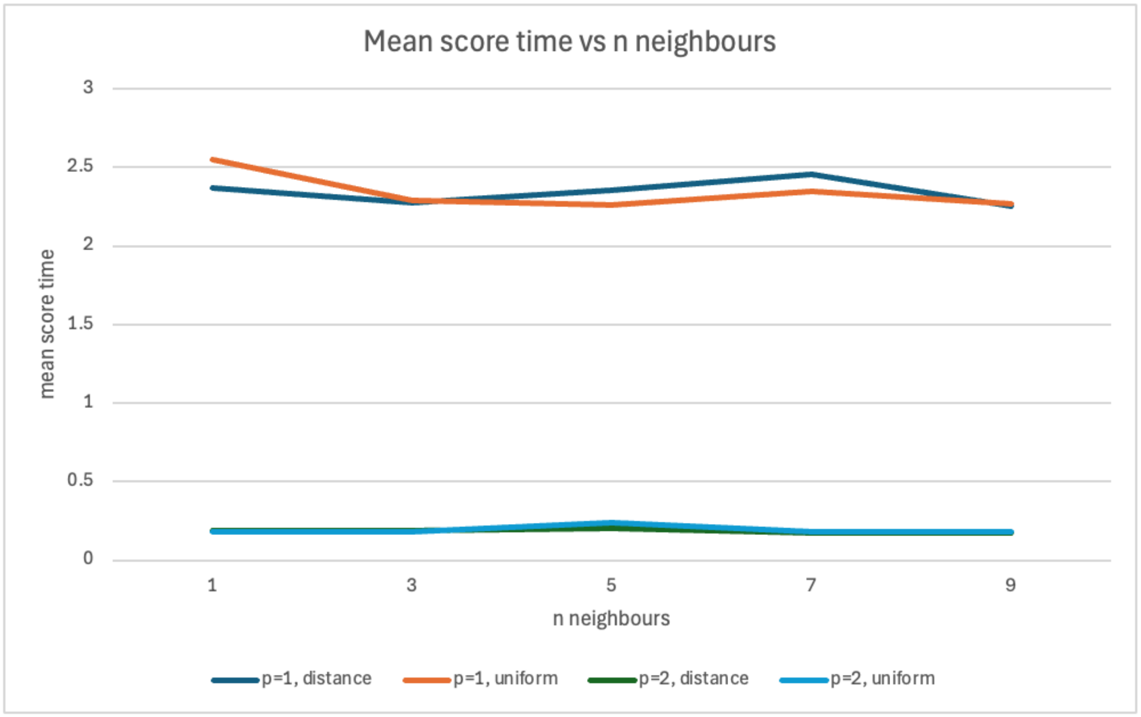
Moreover, models using p = 1 consistently outperformed those using p = 2 (Euclidean distance), indicating that the Manhattan distance was better suited for this dataset. This may be attributed to the high-dimensional pixel data, where Manhattan distance is less sensitive to the curse of dimensionality compared to Euclidean distance.

Overall, there was no significant evidence of accuracy deteriorating with increasing n_neighbors, which diverges slightly from the typical expectation of increased bias in KNN as more neighbours are included. This suggests that the dataset's structure might be less sensitive to the size of the neighbourhood within the tested range.

Runtime Trends



While KNN is a lazy learner with negligible fit time, the runtime differences across configurations were still observable. Fit times were generally low and stable across all hyperparameter settings, though an anomaly was noted where n_neighbors = 1 and p = 1 with uniform weighting exhibited unusually high fit time (0.0537 seconds) relative to others (typically around 0.008–0.01 seconds). This might be due to initialisation overhead or caching behaviour in the scikit-learn implementation.



Score time, in contrast, varied significantly. Configurations using p = 1 (Manhattan distance) consistently demonstrated much higher score times, averaging around 2.5 seconds, compared to approximately 0.18–0.20 seconds for configurations using p = 2 (Euclidean distance), regardless of the number of neighbours or weighting scheme. Interestingly, the number of neighbours (n_neighbors) itself appeared to have minimal impact on the score time within both distance metrics. This behaviour suggests that the computational overhead in this case is predominantly influenced by the distance calculation method rather than the number of neighbours considered during voting. This aligns with known characteristics of KNN where Manhattan distance can be computationally more intensive than Euclidean distance, especially when working with multi-dimensional data.

Performance vs Runtime Trade-off

The hyperparameter tuning results for KNN highlighted a clear trade-off between predictive performance and computational efficiency, primarily driven by the choice of distance metric rather than the number of neighbours. While the highest accuracy was achieved with p = 1 (Manhattan distance) at n_neighbors = 1, this configuration also resulted in the longest score times (approximately 2.5 seconds). Conversely, all configurations using p = 2 (Euclidean distance) achieved slightly lower but still comparable accuracies (approximately 27%–28%) with dramatically reduced score times (around 0.18–0.20 seconds). The number of neighbours had minimal effect on both accuracy and score time, suggesting that, for this dataset, the computational bottleneck lies in the distance calculation itself rather than in the neighbour voting process.

This observation implies that in scenarios where inference speed is critical, sacrificing a small margin of accuracy by opting for p = 2 could result in significant improvements in computational efficiency. Conversely, in offline analysis settings where prediction time is less critical, configurations using p = 1 may be justified to maximise predictive performance. The trade-off was more pronounced than initially anticipated, as the difference in score time between p = 1 and p = 2 exceeded an order of magnitude, while the associated accuracy difference remained modest.

Alignment with Expectations

The results largely align with theoretical expectations of KNN's behaviour, where lower n_neighbors is typically expected to improve accuracy at the expense of generalisability. However, the data showed that accuracy remained stable across most configurations, which was somewhat surprising and suggests that the dataset may have relatively well-separated classes that are not highly sensitive to the size of the neighbourhood within the tested range. Additionally, the performance advantage of Manhattan distance (p = 1) over Euclidean distance (p = 2) was consistent with known properties of high-dimensional data, where Manhattan distance can better preserve discriminative information.

The significant impact of the distance metric on score time also aligned with expectations, though the magnitude of the difference was larger than anticipated. This highlights the importance of considering computational costs associated with distance metrics when deploying KNN models, particularly in high-dimensional settings. Overall, the trends observed were consistent with theoretical principles, while also offering empirical insights into the cost-performance dynamics specific to the dataset.

4.1.3. MLP Results

param_batch_size	param_epochs	param_model__hidden_units1	param_model__hidden_units2	param_model__learning_rate	param_optimizer	mean_fit_time
32	5	256	128	0.01	adam	1.943217993
32	5	256	128	0.01	rmsprop	1.844264587
32	5	256	128	0.001	adam	2.037184874

32	5	256	128	0.001	rmsprop	2.231108268
32	5	256	256	0.01	adam	2.143074115
32	5	256	256	0.01	rmsprop	2.056999763
32	5	256	256	0.001	adam	1.977853616
32	5	256	256	0.001	rmsprop	2.240241369
32	5	512	128	0.01	adam	2.893040419
32	5	512	128	0.01	rmsprop	3.220081409
32	5	512	128	0.001	adam	3.823958556
32	5	512	128	0.001	rmsprop	3.003110806
32	5	512	256	0.01	adam	3.078799804
32	5	512	256	0.01	rmsprop	2.887222608
32	5	512	256	0.001	adam	2.781615655
32	5	512	256	0.001	rmspro	2.752

					p	476613
64	5	256	128	0.01	adam	1.263908625
64	5	256	128	0.01	rmsprop	1.294101318
64	5	256	128	0.001	adam	1.198752403
64	5	256	128	0.001	rmsprop	1.684503635
64	5	256	256	0.01	adam	1.279726744
64	5	256	256	0.01	rmsprop	1.252493302
64	5	256	256	0.001	adam	1.313447555
64	5	256	256	0.001	rmsprop	1.267518361
64	5	512	128	0.01	adam	1.745650053
64	5	512	128	0.01	rmsprop	1.813211759
64	5	512	128	0.001	adam	1.863037745
64	5	512	128	0.001	rmspro	1.955

					p	682357
64	5	512	256	0.01	adam	2.402721882
64	5	512	256	0.01	rmsprop	3.150268396
64	5	512	256	0.001	adam	4.505964677
64	5	512	256	0.001	rmsprop	4.617147207

Mean fit time

param_batch_size	param_epochs	param_model__hidden_units1	param_model__hidden_units2	param_model__learning_rate	param_optimizer	mean_score_time
32	5	256	128	0.01	adam	0.112623215
32	5	256	128	0.01	rmsprop	0.0998431055
32	5	256	128	0.001	adam	0.1048140533
32	5	256	128	0.001	rmsprop	0.119745016
32	5	256	256	0.01	adam	0.110917966
32	5	256	256	0.01	rmsprop	0.110855103

32	5	256	256	0.001	adam	0.125574748
32	5	256	256	0.001	rmsprop	0.152227163
32	5	512	128	0.01	adam	0.464336952
32	5	512	128	0.01	rmsprop	0.170769612
32	5	512	128	0.001	adam	0.182956616
32	5	512	128	0.001	rmsprop	0.138166189
32	5	512	256	0.01	adam	0.127358119
32	5	512	256	0.01	rmsprop	0.121966362
32	5	512	256	0.001	adam	0.115048567
32	5	512	256	0.001	rmsprop	0.122520049
64	5	256	128	0.01	adam	0.106906017
64	5	256	128	0.01	rmsprop	0.090176662
64	5	256	128	0.001	adam	0.1106

						81295
64	5	256	128	0.001	rmsprop	0.0930384
64	5	256	256	0.01	adam	0.0871097250
64	5	256	256	0.01	rmsprop	0.0864596370
64	5	256	256	0.001	adam	0.089855194
64	5	256	256	0.001	rmsprop	0.118436337
64	5	512	128	0.01	adam	0.158435901
64	5	512	128	0.01	rmsprop	0.113319635
64	5	512	128	0.001	adam	0.119162242
64	5	512	128	0.001	rmsprop	0.132053057
64	5	512	256	0.01	adam	0.175993045
64	5	512	256	0.01	rmsprop	0.252776225
64	5	512	256	0.001	adam	0.306053956
64	5	512	256	0.001	rmsprop	0.31840578

						7

Mean score time

param_batch_size	param_epochs	param_model__hidden_units1	param_model__hidden_units2	param_model__learning_rate	param_optimizer	mean_test_score
32	5	256	128	0.01	adam	0.146799932
32	5	256	128	0.01	rmsprop	0.146799932
32	5	256	128	0.001	adam	0.251213863
32	5	256	128	0.001	rmsprop	0.251213863
32	5	256	256	0.01	adam	0.146799932
32	5	256	256	0.01	rmsprop	0.146799932
32	5	256	256	0.001	adam	0.273002302
32	5	256	256	0.001	rmsprop	0.273002302
32	5	512	128	0.01	adam	0.146799932
32	5	512	128	0.01	rmspro	0.146

					p	799932
32	5	512	128	0.001	adam	0.2952054 25
32	5	512	128	0.001	rmsprop	0.2952054 25
32	5	512	256	0.01	adam	0.1467999 32
32	5	512	256	0.01	rmsprop	0.1467999 32
32	5	512	256	0.001	adam	0.3018119 07
32	5	512	256	0.001	rmsprop	0.3018119 07
64	5	256	128	0.01	adam	0.2169858 95
64	5	256	128	0.01	rmsprop	0.2169858 95
64	5	256	128	0.001	adam	0.2313852 96
64	5	256	128	0.001	rmsprop	0.2313852 96
64	5	256	256	0.01	adam	0.1467999 32
64	5	256	256	0.01	rmspro	0.146

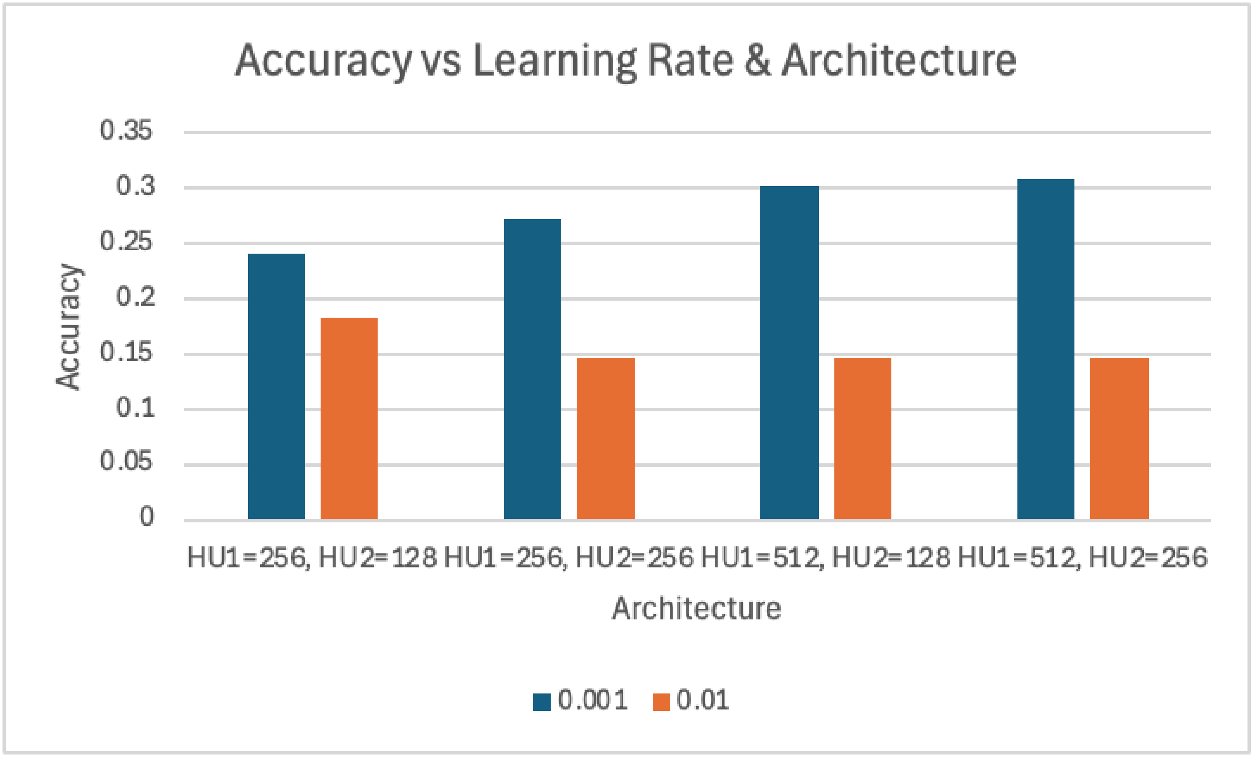
					p	799932
64	5	256	256	0.001	adam	0.2704035422
64	5	256	256	0.001	rmsprop	0.2704035422
64	5	512	128	0.01	adam	0.1467999932
64	5	512	128	0.01	rmsprop	0.1467999932
64	5	512	128	0.001	adam	0.3078086666
64	5	512	128	0.001	rmsprop	0.3078086666
64	5	512	256	0.01	adam	0.1467999932
64	5	512	256	0.01	rmsprop	0.1467999932
64	5	512	256	0.001	adam	0.3152053866
64	5	512	256	0.001	rmsprop	0.3152053866

Accuracy

4.1.4. MLP Discussion

The hyperparameter tuning results for the Multilayer Perceptron (MLP) classifier revealed several notable trends concerning model performance and runtime. The key hyperparameters tuned included the architecture size (hidden_units1, hidden_units2), learning rate (learning_rate), optimizer (adam, rmsprop), and batch size (32, 64). The models were trained for a fixed 5 epochs to maintain consistent comparison. Performance was evaluated using mean cross-validation accuracy, alongside measurements of mean fit time and mean score time.

Accuracy Trends



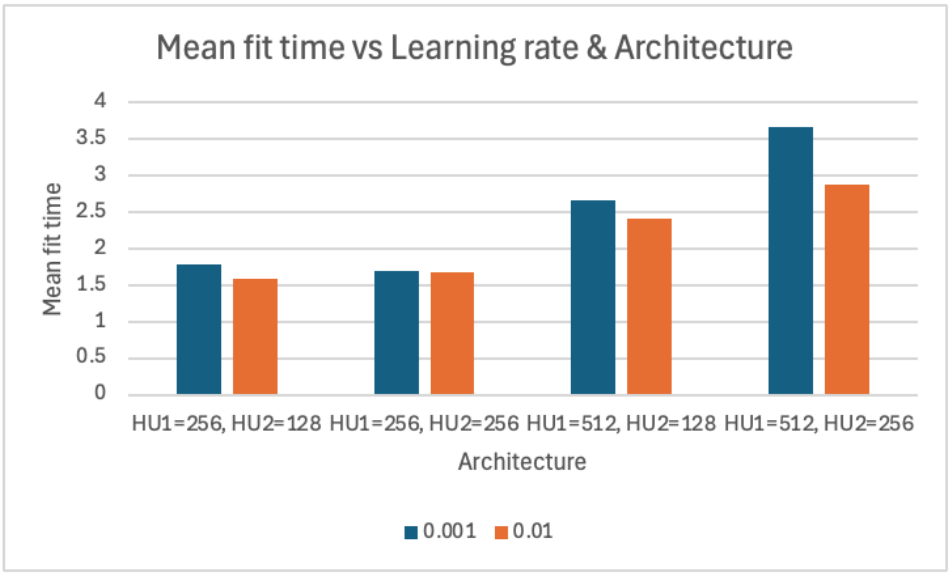
The tuning results highlighted learning rate as the most influential hyperparameter impacting classification accuracy. Models trained with a learning rate of 0.001 consistently outperformed those trained with 0.01, regardless of the optimizer, hidden unit configuration, or batch size. Specifically, the highest observed accuracy (approximately 31.5%) was achieved using the configuration with 512-256 hidden units, learning_rate = 0.001, rmsprop optimizer, and batch_size = 64. In contrast, models trained with 0.01 learning rate consistently failed to surpass approximately 14.7% accuracy, suggesting the higher learning rate hindered effective convergence within the limited number of epochs.

Model architecture also influenced accuracy, albeit to a lesser extent than the learning rate. Larger architectures (512-256) generally achieved higher accuracies than smaller ones

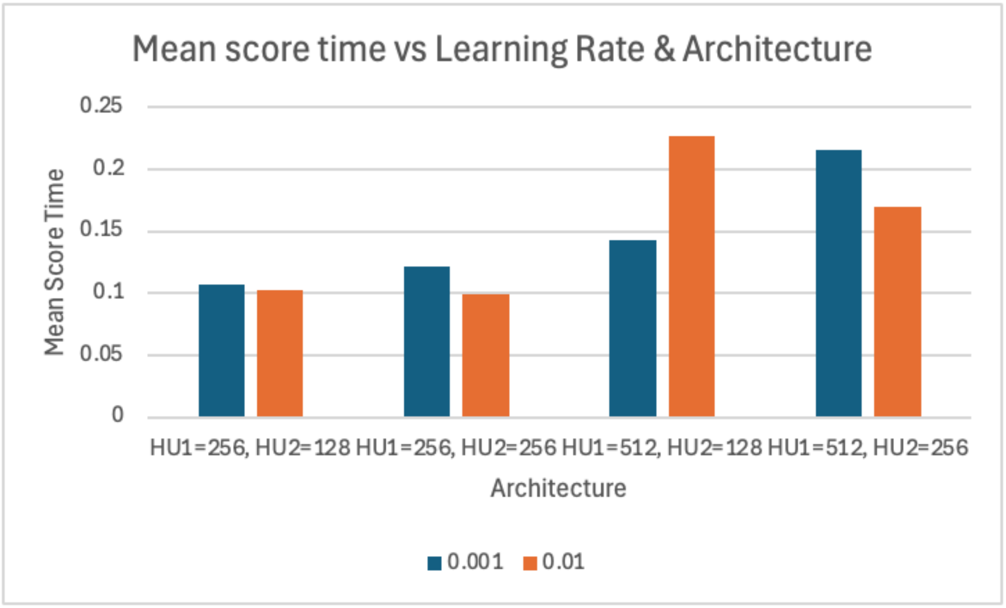
(256-128), but the margin was modest. This suggests that increasing model capacity offered some gains in representation power, but the key bottleneck remained the learning rate and convergence within 5 epochs.

Optimizer choice (adam vs rmsprop) had negligible impact on accuracy, with both optimizers producing similar results across equivalent configurations. Likewise, batch size showed only marginal influence on accuracy, though configurations using batch_size = 64 with learning_rate = 0.001 slightly outperformed their counterparts with batch_size = 32.

Runtime Trends



Mean fit time was strongly affected by both model complexity and learning rate. Configurations with larger architectures (512-256) and learning_rate = 0.001 exhibited the longest fit times, averaging more than 3.5 seconds, compared to 1.8-2.3 seconds for simpler models (256-128). This is expected, as larger models have more parameters to update per batch, and lower learning rates result in slower convergence per epoch due to smaller weight updates.



Mean score time was generally low and stable across most configurations, typically ranging between 0.08 and 0.18 seconds. This indicates that inference time in MLP models is not substantially affected by hyperparameters such as model size, learning rate, or optimizer, which aligns with theoretical expectations. As MLP inference involves only forward propagation without any backpropagation or gradient calculations, the computational cost is relatively predictable and efficient.

However, a few outliers were observed. Specifically, configuration with 512-128 hidden units, batch_size = 32, learning_rate = 0.01, and adam optimizer reached approximately 0.46 seconds. This suggests that, while generally negligible, the increased number of parameters in these deeper and wider models can introduce marginal additional computational overhead during inference. Nevertheless, these variations remained within an acceptable range and are unlikely to pose practical limitations, particularly in non-real-time applications.

These findings confirm that for MLP models, score time is relatively insensitive to most tuning parameters, and runtime considerations should focus primarily on fit time rather than inference time when evaluating performance trade-offs.

Performance vs Runtime Trade-off

The hyperparameter tuning highlighted a clear trade-off between predictive performance and computational cost during training, primarily driven by the learning rate and model architecture. The best-performing configurations (512-256, learning_rate = 0.001) required over double the fit time compared to simpler architectures, yet only provided a 1.5–2% increase in accuracy. This suggests that while deeper and wider networks can offer performance improvements, the returns may diminish relative to the additional training cost, especially under strict computational constraints.

Given that inference time remained efficient and relatively unaffected by hyperparameters, the main consideration in deployment scenarios would be the training cost versus performance requirements. For applications where training time and resource efficiency are critical, configurations such as 256-128, learning_rate = 0.001, and batch_size = 64 might offer a practical balance, achieving around 25% accuracy with roughly half the fit time of the best model.

Alignment with Expectations

The observed trends are largely consistent with theoretical expectations of MLP training dynamics. Lower learning rates facilitated better convergence, avoiding the instability and underfitting seen with higher learning rates (0.01), particularly within the limited 5-epoch regime. The incremental accuracy improvements from larger model architectures were expected, albeit the gains were somewhat modest, reflecting the potential diminishing returns of increasing depth and width without extending the number of training epochs.

The stable inference time across all configurations aligned with the known efficiency of forward passes in MLPs, regardless of the hyperparameters used during training. Overall, the results validated the importance of carefully tuning the learning rate and model size, while showing that batch size and optimizer selection had limited impact under the current settings.

4.1.5. CNN Results

param_batch_size	param_epochs	param_model__dropout_rate	param_model__learning_rate	param_model__num_filters	param_optimizer	mean_test_score
32	10	0.3	0.001	32	adam	0.600601968
32	10	0.3	0.001	32	rmsprop	0.600601968
32	10	0.3	0.001	64	adam	0.59261113
32	10	0.3	0.001	64	rmsprop	0.59261113
32	10	0.3	0.0005	32	adam	0.599204529

32	10	0.3	0.0005	32	rmsprop	0.5992045299
32	10	0.3	0.0005	64	adam	0.6589991733
32	10	0.3	0.0005	64	rmsprop	0.6589991733
32	10	0.5	0.001	32	adam	0.5742058077
32	10	0.5	0.001	32	rmsprop	0.5742058077
32	10	0.5	0.001	64	adam	0.5946044888
32	10	0.5	0.001	64	rmsprop	0.5946044888
32	10	0.5	0.0005	32	adam	0.5352078444
32	10	0.5	0.0005	32	rmsprop	0.5352078444
32	10	0.5	0.0005	64	adam	0.611399929
32	10	0.5	0.0005	64	rmsprop	0.611399929
64	10	0.3	0.001	32	adam	0.6185940488
64	10	0.3	0.001	32	rmsprop	0.61859404

						8
64	10	0.3	0.001	64	adam	0.644601932
64	10	0.3	0.001	64	rmsprop	0.644601932
64	10	0.3	0.0005	32	adam	0.533202003
64	10	0.3	0.0005	32	rmsprop	0.533202003
64	10	0.3	0.0005	64	adam	0.61700505
64	10	0.3	0.0005	64	rmsprop	0.61700505
64	10	0.5	0.001	32	adam	0.538200403
64	10	0.5	0.001	32	rmsprop	0.538200403
64	10	0.5	0.001	64	adam	0.528410524
64	10	0.5	0.001	64	rmsprop	0.528410524
64	10	0.5	0.0005	32	adam	0.487614962
64	10	0.5	0.0005	32	rmsprop	0.487614962

64	10	0.5	0.0005	64	adam	0.512805602
64	10	0.5	0.0005	64	rmsprop	0.512805602

Accuracy

param_batch_size	param_epochs	param_model__dropout_rate	param_model__learning_rate	param_model__num_filters	param_optimizer	mean_fit_time
32	10	0.3	0.001	32	adam	12.34956066
32	10	0.3	0.001	32	rmsprop	9.825862567
32	10	0.3	0.001	64	adam	20.6390264
32	10	0.3	0.001	64	rmsprop	19.61846773
32	10	0.3	0.0005	32	adam	8.916782935
32	10	0.3	0.0005	32	rmsprop	8.603474617
32	10	0.3	0.0005	64	adam	19.14226325
32	10	0.3	0.0005	64	rmsprop	19.26753259
32	10	0.5	0.001	32	adam	8.69612685

						8
32	10	0.5	0.001	32	rmsprop	8.401114305
32	10	0.5	0.001	64	adam	18.39667455
32	10	0.5	0.001	64	rmsprop	21.483793334
32	10	0.5	0.0005	32	adam	8.246713718
32	10	0.5	0.0005	32	rmsprop	9.799476624
32	10	0.5	0.0005	64	adam	25.438020077
32	10	0.5	0.0005	64	rmsprop	21.819497277
64	10	0.3	0.001	32	adam	8.285310904
64	10	0.3	0.001	32	rmsprop	8.371483405
64	10	0.3	0.001	64	adam	18.748152577
64	10	0.3	0.001	64	rmsprop	18.636166577
64	10	0.3	0.0005	32	adam	8.071890036

64	10	0.3	0.0005	32	rmsprop	8.197940667
64	10	0.3	0.0005	64	adam	20.341548 28
64	10	0.3	0.0005	64	rmsprop	19.161026 95
64	10	0.5	0.001	32	adam	9.853767316
64	10	0.5	0.001	32	rmsprop	8.984645526
64	10	0.5	0.001	64	adam	20.46109597
64	10	0.5	0.001	64	rmsprop	20.40574233
64	10	0.5	0.0005	32	adam	9.728930712
64	10	0.5	0.0005	32	rmsprop	9.545789003
64	10	0.5	0.0005	64	adam	22.13006798
64	10	0.5	0.0005	64	rmsprop	21.68926303

Mean fit time

param_	param	param_model	param_model	param_mode	param_	mean

batch_size	_epochs	__dropout_rate	__learning_rate	l__num_filters	optimizer	_score_time
32	10	0.3	0.001	32	adam	0.3097986388
32	10	0.3	0.001	32	rmsprop	0.2802046933
32	10	0.3	0.001	64	adam	0.4366919999
32	10	0.3	0.001	64	rmsprop	0.4523432255
32	10	0.3	0.0005	32	adam	0.2269672555
32	10	0.3	0.0005	32	rmsprop	0.23268199
32	10	0.3	0.0005	64	adam	0.4342820644
32	10	0.3	0.0005	64	rmsprop	0.4264804522
32	10	0.5	0.001	32	adam	0.2186810977
32	10	0.5	0.001	32	rmsprop	0.2142773477
32	10	0.5	0.001	64	adam	1.3125785999
32	10	0.5	0.001	64	rmspro	0.282

					p	310327
32	10	0.5	0.0005	32	adam	0.193651676
32	10	0.5	0.0005	32	rmsprop	0.269419909
32	10	0.5	0.0005	64	adam	0.643721422
32	10	0.5	0.0005	64	rmsprop	0.478785038
64	10	0.3	0.001	32	adam	0.245014032
64	10	0.3	0.001	32	rmsprop	0.226912657
64	10	0.3	0.001	64	adam	0.525326411
64	10	0.3	0.001	64	rmsprop	0.403774261
64	10	0.3	0.0005	32	adam	0.235764424
64	10	0.3	0.0005	32	rmsprop	0.236992041
64	10	0.3	0.0005	64	adam	0.531977495
64	10	0.3	0.0005	64	rmspro	0.445

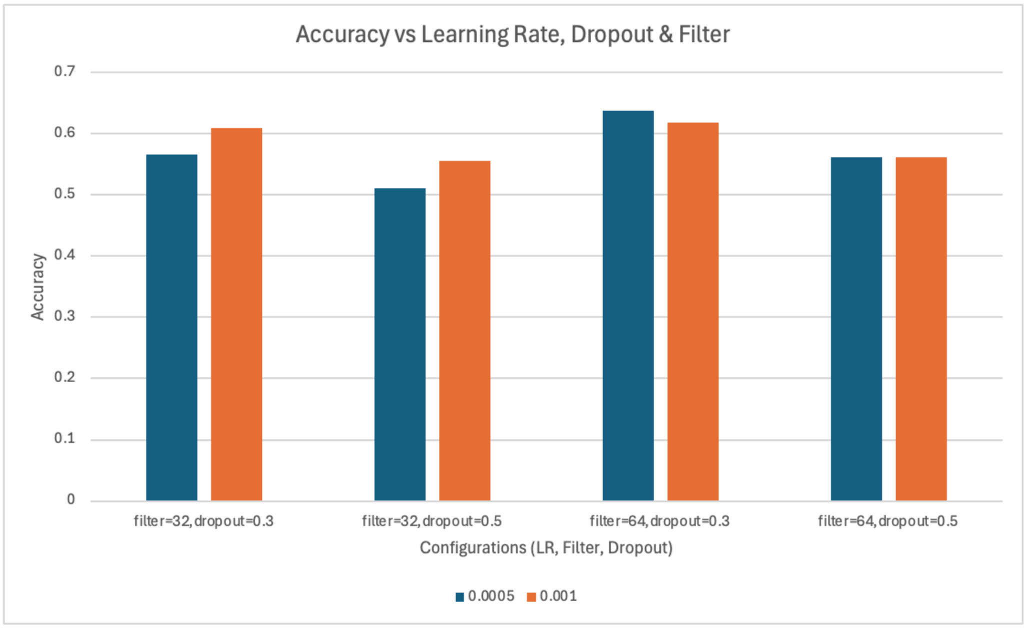
					p	033073
64	10	0.5	0.001	32	adam	0.307887952
64	10	0.5	0.001	32	rmsprop	0.277279139
64	10	0.5	0.001	64	adam	0.487537384
64	10	0.5	0.001	64	rmsprop	0.53481698
64	10	0.5	0.0005	32	adam	0.305229346
64	10	0.5	0.0005	32	rmsprop	0.279200713
64	10	0.5	0.0005	64	adam	0.508195718
64	10	0.5	0.0005	64	rmsprop	0.515769561

Mean score time

4.1.6. CNN Discussion

The hyperparameter tuning results for the Convolutional Neural Network (CNN) classifier revealed clear patterns in model performance and runtime. The key hyperparameters tuned included the number of filters in the first convolutional layer (num_filters), dropout rate (dropout_rate), learning rate (learning_rate), optimizer (adam, rmsprop), and batch size (32, 64). The models were trained for a fixed 10 epochs to ensure comparability. Model performance was evaluated using mean cross-validation accuracy, with mean fit time and mean score time recorded to assess computational efficiency.

Accuracy Trends



The tuning results showed that both learning rate and model capacity had notable effects on classification accuracy, with some interesting interactions observed between the parameters.

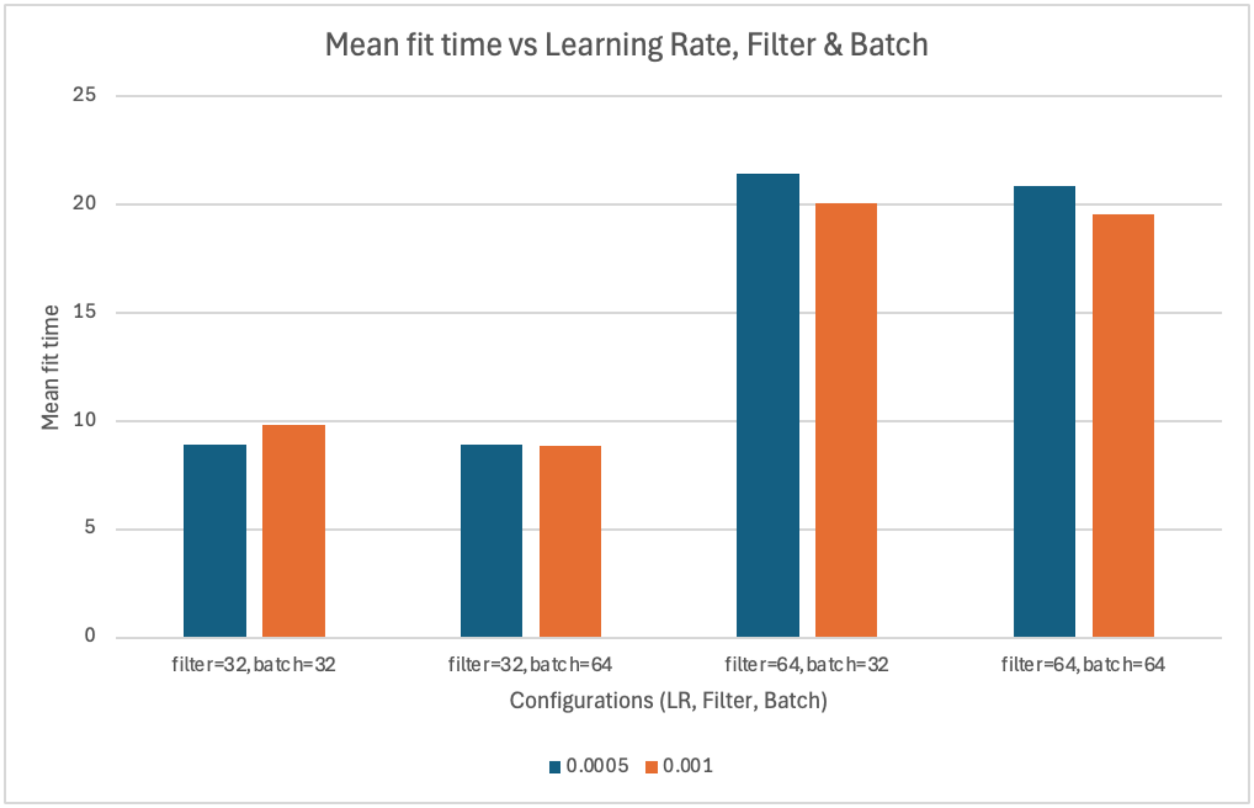
Contrary to expectations, models trained with a learning rate of 0.0005 consistently outperformed those with 0.001 in several configurations, particularly when using 64 filters. For example, the highest accuracy was achieved with 64 filters, 0.3 dropout rate, 0.0005 learning rate, batch_size = 32, and either adam or rmsprop optimizer, reaching approximately **65.9%**. This suggests that the lower learning rate allowed the CNN to converge more effectively within the 10 epochs, especially in the larger model configurations.

Increasing the number of filters from 32 to 64 led to substantial accuracy improvements across all settings, confirming the importance of model capacity for feature extraction in CNNs. In contrast, using only 32 filters limited performance to around **60%**, regardless of the other hyperparameters.

Dropout rate also influenced accuracy, with models using 0.3 dropout achieving higher accuracy than those using 0.5. This suggests that the higher dropout rate may have introduced excessive regularisation, preventing the model from effectively fitting the data, particularly in the lower-capacity configurations.

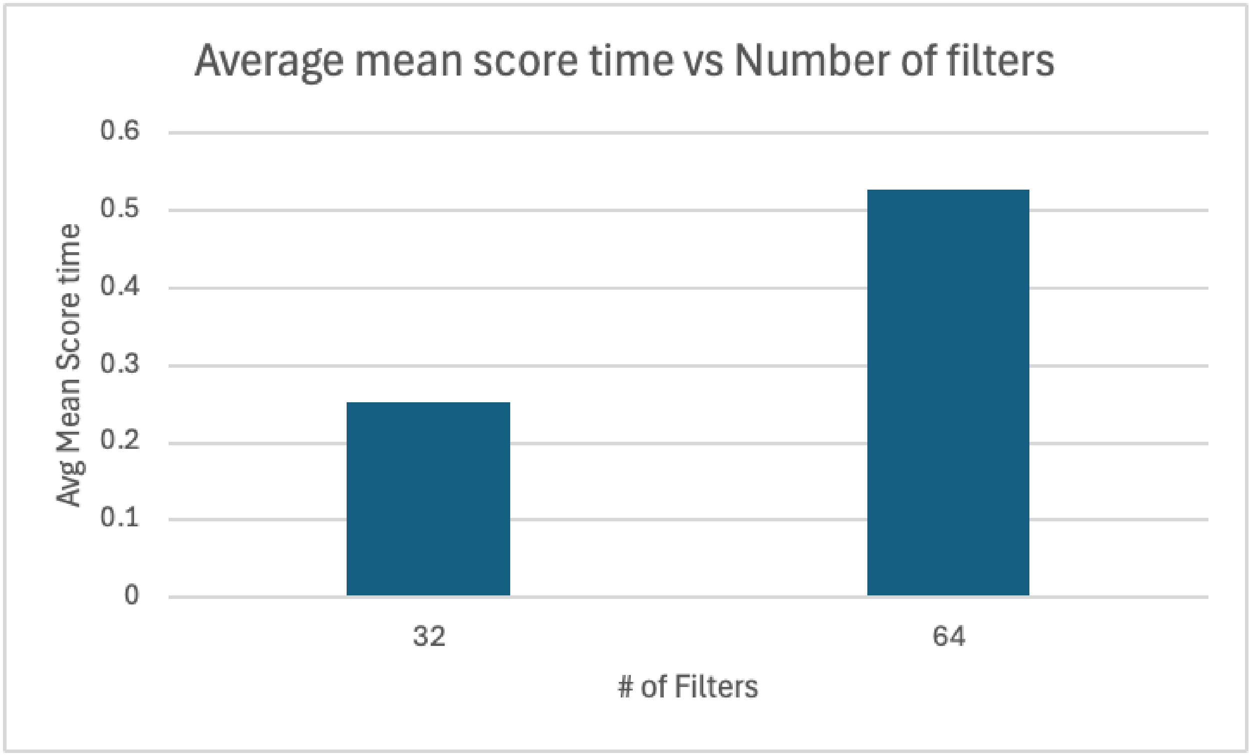
Optimizer choice (adam versus rmsprop) had minimal effect on accuracy across all settings, with both optimizers producing nearly identical performance. Similarly, batch size (32 versus 64) showed some influence, where batch_size = 32 consistently achieved slightly higher accuracy than 64. This indicates that smaller batch sizes may have offered more stable updates and better generalisation for this dataset.

Runtime Trends



Fit time increased consistently with the number of filters and lower learning rates. Configurations using 64 filters exhibited the highest fit times, often exceeding 20 seconds, especially when combined with a lower learning rate of 0.0005 and batch size of 32. The longest observed fit time was 25.4 seconds for the configuration using 64 filters, 0.5 dropout rate, 0.0005 learning rate, adam optimizer, and batch size 32. This reflects the increased number of convolution operations and parameter updates required for larger models and smaller learning rates, where the smaller step sizes prolonged the convergence process.

Batch size also had an observable impact on fit time, where batch size 64 generally reduced fit times by around 10-20% compared to batch size 32, due to more efficient processing of larger batches and fewer updates per epoch. However, this trend was less noticeable in larger models, where the computational cost remained high regardless of batch size.



Score time trends mirrored fit time, with larger models incurring longer inference times. Configurations using 64 filters consistently exhibited higher score times, with the highest reaching 1.31 seconds for the configuration with 64 filters, 0.5 dropout, 0.001 learning rate, adam optimizer, and batch size 32. In contrast, models with 32 filters typically had score times around 0.2 to 0.3 seconds across all configurations. This substantial difference highlights the heavier inference burden of larger CNN models, where the number of filters directly affects the size of the feature maps and the computational load during convolutional operations.

Some configurations showed anomalies where score time for 64 filters with certain dropout and learning rate combinations was unexpectedly lower (e.g. 0.4–0.5 seconds), suggesting that other factors such as internal optimizations, batch normalization behaviour, or specific data characteristics might have played a role.

Overall, the analysis confirmed that both fit time and score time scaled significantly with model complexity in CNNs, and these costs need to be considered carefully in scenarios requiring fast inference or limited training time.

Performance vs Runtime Trade-off

The tuning results for CNN highlighted a clear trade-off between performance and computational cost. The best accuracy, approximately 65.9%, was achieved using 64 filters, 0.3 dropout, 0.0005 learning rate, and batch size 32, but this also resulted in the highest fit and score times, with fit times exceeding 25 seconds and score times over 0.6 seconds. In contrast, smaller models with 32 filters and higher dropout achieved lower accuracy (around 60%) but with fit times below 10 seconds and score times under 0.3 seconds.

Batch size 64 generally reduced fit times by processing larger batches more efficiently, though this effect diminished in the largest models where computational demands remained high.

These findings suggest that higher-capacity models with lower learning rates offer better accuracy but come with significant runtime costs. For scenarios where computational efficiency is prioritised, using 32 filters with 0.3 dropout and 0.001 learning rate offers a practical compromise, reducing runtime while maintaining moderate accuracy.

Alignment with Expectations

The results aligned with CNN theory, where increasing model size and lowering learning rates improved accuracy but at higher computational costs. The stronger-than-expected advantage of 0.0005 learning rate, particularly in larger models, suggests the dataset benefited from more gradual convergence. The impact of dropout was also consistent, with higher dropout leading to underfitting, especially in smaller models.

Score time increased notably with model size, reflecting the heavier inference demands of CNNs compared to MLPs. Overall, the trends confirmed known behaviours of CNNs, reinforcing the need to balance model complexity against runtime requirements in deployment scenarios.

4.2. Models

4.2.1. Result

Model	Best Hyperparameters	Test Accuracy	Fit Time	Score Time
KNN	{'n_neighbors': 1, 'p': 1, 'weights': 'distance'}	0.312998505	0.00888896	2.372196754
MLP	{'batch_size': 64, 'epochs': 5, 'model__hidden_units1': 512, 'model__hidden_units2': 256, 'model__learning_rate': 0.001, 'optimizer': 'adam'}	0.315205386	4.505964677	0.306053956
CNN	{'batch_size': 32, 'epochs': 10, 'model__dropout_rate': 0.3, 'model__learning_rate': 0.0005, 'model__num_filters': 64, 'optimizer': 'adam'}	0.658999173	19.14226325	0.434282064

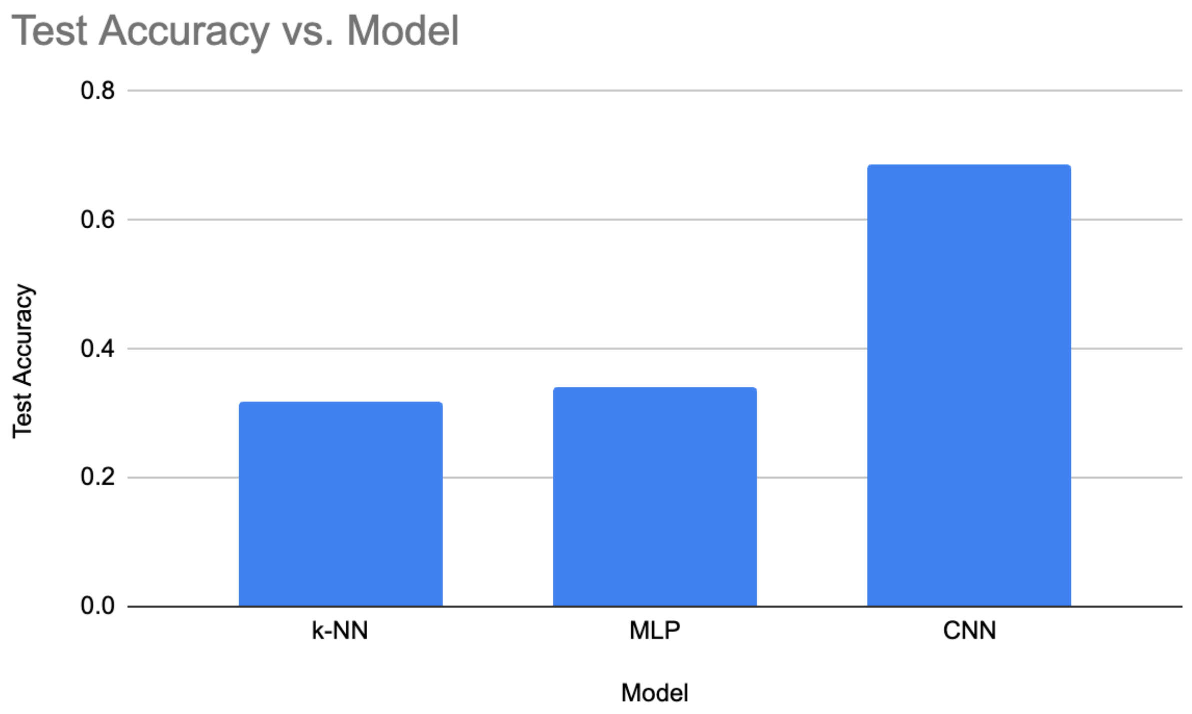
Results on Training set via 3-fold cross validation

Model	Test Accuracy	F1 Score (macro)	Train Time (s)	Test Time (s)
k-NN	0.317	0.25963	0.014563	15.888074
MLP	0.33925	0.242078	4.462969	0.799005
CNN	0.687375	0.674622	22.606919	2.161633

Results on test set

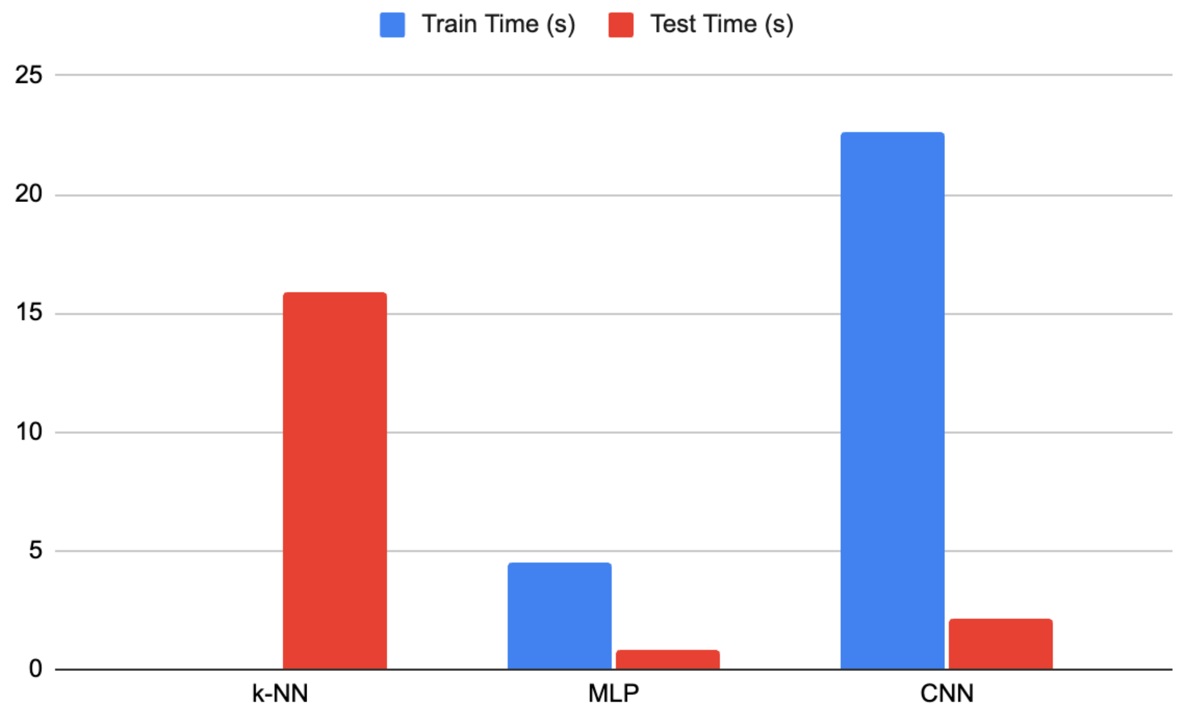
4.2.2. Model Comparison and Discussion

Performance Comparison



The CNN substantially outperformed both KNN and MLP, achieving the highest test accuracy of approximately 68.7% and a macro F1-score of 67.5%. In comparison, MLP achieved a moderate test accuracy of 33.9% and a macro F1-score of 24.2%, whereas KNN exhibited the lowest performance, with a test accuracy of 31.7% and macro F1-score of 26.0%. These outcomes align closely with theoretical expectations. CNN's superior performance can be attributed to its ability to effectively leverage spatial locality through convolutional layers and pooling operations, thus accurately capturing complex visual patterns present in biomedical images. Conversely, the MLP's moderate performance highlights its limitations in processing image data, as it treats the input images as flattened vectors and fails to utilize spatial information. Similarly, the performance of KNN reflects its fundamental constraint of relying purely on distance metrics in high-dimensional pixel space, without any capacity for abstract feature learning, resulting in limited effectiveness on complex datasets such as PathMNIST.

Runtime Analysis

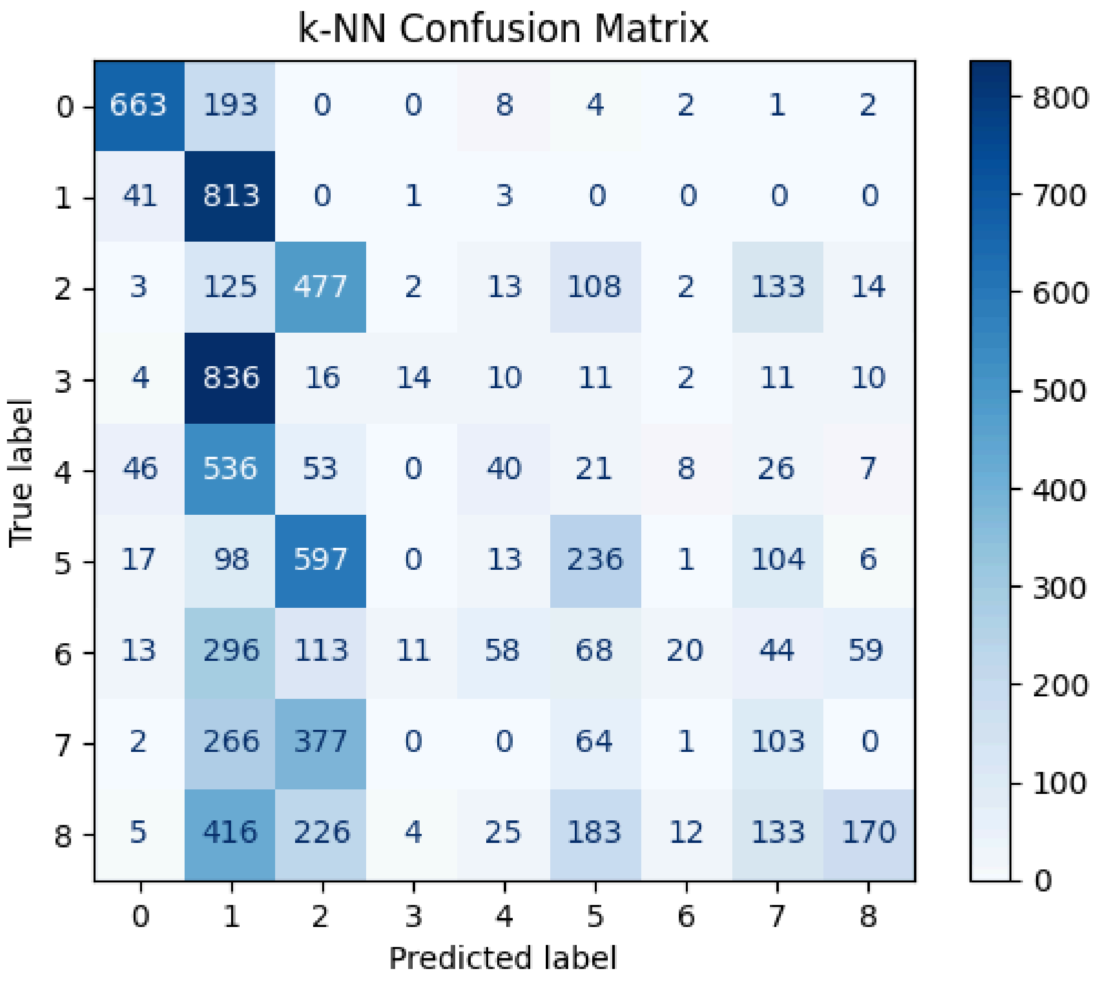


Distinct runtime characteristics observed were consistent with theoretical expectations. KNN exhibited the shortest training time (approximately 0.015 seconds), as it involves no explicit training phase. However, KNN had the highest inference time (approximately 15.89 seconds), consistent with its computationally intensive nature during prediction due to extensive distance calculations. The MLP required a moderate training duration (approximately 4.46 seconds), reflecting the time taken for gradient-based optimization over multiple hidden layers, but it demonstrated highly efficient inference speed (approximately 0.80 seconds) due to the simplicity of forward propagation. The CNN required the longest training duration (approximately 22.61 seconds) due to the complexity inherent in convolutional operations, multiple layers, and training over several epochs. Despite its longer training time, CNN had a comparatively efficient inference time (approximately 2.16 seconds), reflecting its optimized computation for forward propagation in convolutional architectures.

Interestingly, while theoretical insights suggested that CNNs should exhibit the highest computational complexity, practical runtime results showed CNN inference was notably more efficient than KNN. This difference underscores the significant computational overhead KNN faces when calculating distances in high-dimensional feature spaces. Thus, despite CNN's relatively higher complexity, it remains highly practical and effective for deployment scenarios involving image classification tasks, where inference speed is crucial alongside model accuracy.

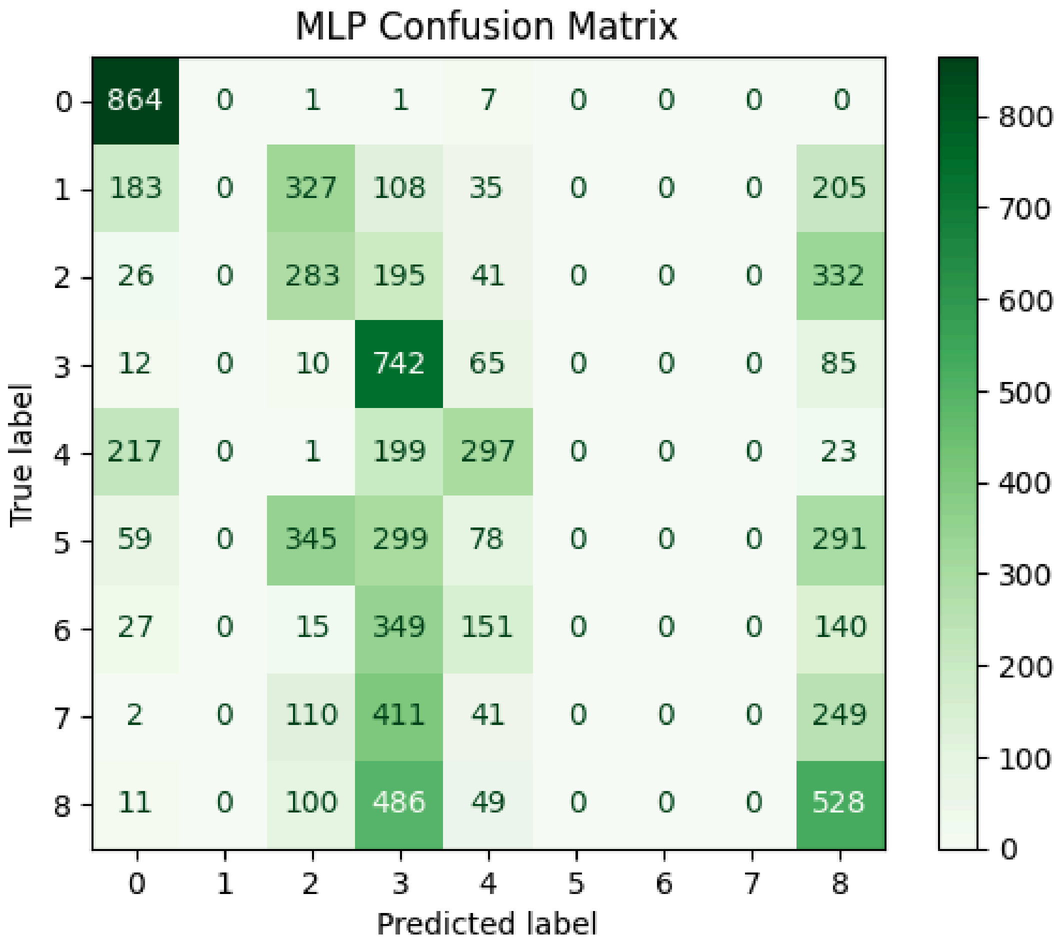
Types of Errors and Class-wise Performance

Confusion matrices highlighted crucial differences in the types of mistakes each model made:

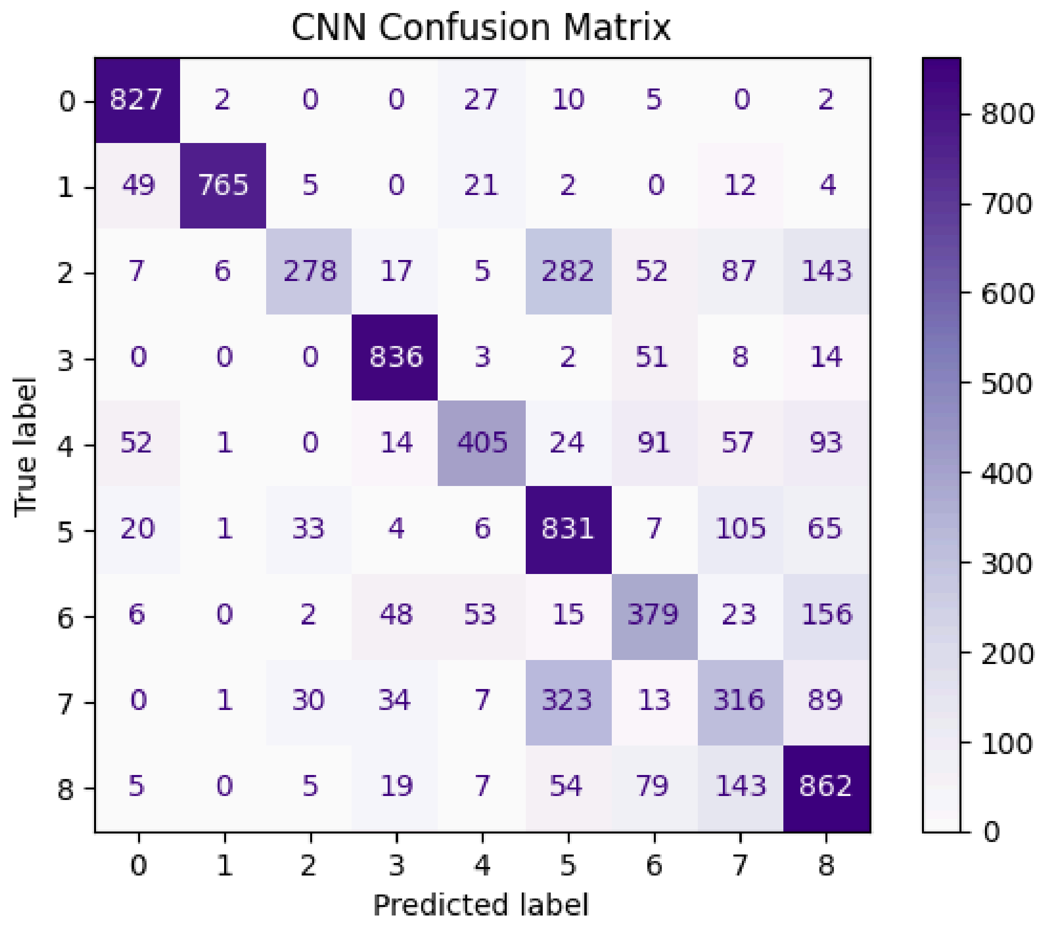


True label \ Predicted label	0	1	2	3	4	5	6	7	8
0	663	193	0	0	8	4	2	1	2
1	41	813	0	1	3	0	0	0	0
2	3	125	477	2	13	108	2	133	14
3	4	836	16	14	10	11	2	11	10
4	46	536	53	0	40	21	8	26	7
5	17	98	597	0	13	236	1	104	6
6	13	296	113	11	58	68	20	44	59
7	2	266	377	0	0	64	1	103	0
8	5	416	226	4	25	183	12	133	170

- The confusion matrix for k-NN revealed significant confusion among classes, consistent with its theoretical limitations. The classifier notably struggled to discriminate visually similar classes, frequently misclassifying samples across classes with overlapping features, such as class 7 and class 5. For example, class 7 was often misclassified as class 2 or class 1, highlighting k-NN's inability to capture abstract spatial patterns due to its reliance solely on raw pixel distances. These errors underline k-NN's known sensitivity to noise and ambiguous visual boundaries, particularly evident in the PathMNIST dataset.



- MLP's confusion matrix demonstrated widespread misclassifications across most classes. While class 0 and class 3 achieved somewhat higher correct classifications, substantial confusion persisted between most other classes. This broad distribution of errors indicates MLP's theoretical weakness in capturing spatial hierarchies, as it treats image data as flattened vectors without spatial context. This confirms expectations that without careful preprocessing or spatial feature encoding, MLP's capacity to distinguish subtle class differences, particularly in visually complex histopathological images, is limited.



- CNN exhibited a far more balanced and accurate confusion matrix, correctly classifying the majority of samples for most classes, such as classes 0, 1, 3, and 5. Misclassifications were generally fewer and more systematically structured, such as confusion between visually similar classes (e.g., class 4 with class 6 or class 2 with class 4), likely due to subtle textural similarities. These results strongly align with theoretical predictions, demonstrating CNN's effective use of convolutional layers and pooling operations to capture complex spatial patterns, textures, and shapes within the PathMNIST images. Despite some inevitable errors due to inherent class ambiguities, CNN notably reduced misclassification frequency and severity compared to KNN and MLP.

Alignment with Theoretical Expectations

The updated results closely aligned with theoretical expectations. CNN significantly outperformed both MLP and k-NN, achieving the highest test accuracy (68.7%) and macro F1-score (67.5%). This supports theoretical claims about CNN’s advantage due to convolutional layers that effectively exploit spatial locality and hierarchical visual patterns within images.

CNN's confusion matrix further highlighted its superior ability to correctly classify challenging classes with subtle differences in texture and structure.

The performance of MLP improved slightly to 33.9% accuracy, but remained relatively weak, confirming the theoretical expectation that treating images as flattened vectors without spatial context severely limits performance. Errors were broadly spread across many classes, indicating difficulty in learning discriminative features from raw pixel inputs without spatial structure.

The k-NN model remained least effective (31.7% accuracy, 26% F1-score), consistent with theoretical predictions. Its reliance on pixel-wise distance calculations resulted in substantial confusion between visually overlapping classes, validating its known sensitivity to noise and limited ability to handle complex image data.

Regarding runtime, results also matched theoretical predictions. k-NN demonstrated negligible training time (0.015 s) but high test-time cost (15.9 s), reflecting its computationally expensive inference. CNN exhibited the longest training (22.6 s) due to the complexity of convolution operations, but maintained reasonable inference speed (2.16 s), as anticipated given efficient forward-pass processing. MLP had moderate training (4.46 s) and fast inference (0.8 s), aligning with its intermediate complexity relative to CNN and k-NN.

Interestingly, despite CNN's higher computational cost, its superior accuracy justified the trade-off for applications prioritising performance, such as medical diagnosis.

Additional Trends and Interesting Observations

- **CNN’s Robustness:** CNN not only achieved higher accuracy but also more evenly distributed errors, effectively distinguishing classes that were problematic for other models. This confirms CNN’s theoretical robustness and superior generalisation capabilities due to convolutional layers and pooling.

- **MLP’s Performance Gains:** The slight performance improvement in MLP compared to earlier results suggests that careful hyperparameter tuning and possibly extended training could moderately improve MLP’s utility. Still, fundamental limitations due to lack of spatial awareness remain evident.

- **k-NN Inference Cost:** Despite minimal training overhead, k-NN’s high inference cost highlights the substantial computational burden imposed by instance-based prediction, especially problematic for high-dimensional image data. This underscores theoretical warnings about scalability and efficiency.

5. Conclusion

5.1. Main Findings

This study compared three machine learning algorithms: k-Nearest Neighbours (KNN), Multi-Layer Perceptron (MLP), and Convolutional Neural Network (CNN), in the context of an image classification task. Performance evaluation considered accuracy, macro-averaged F1 scores, and runtime metrics, along with interpretability and practical deployment implications.

Performance and Accuracy

The CNN achieved the highest test accuracy (68.7%) and macro-averaged F1 score (67.5%), significantly outperforming both KNN and MLP. This result aligns with theoretical expectations, highlighting CNN's strengths in capturing complex spatial structures and hierarchical features through convolutional and pooling operations.

The MLP demonstrated moderate accuracy (33.9%) and a relatively low macro F1 score (24.2%), reflecting its limited capability in exploiting spatial information due to its fully-connected architecture. The KNN algorithm exhibited the lowest performance, with an accuracy of 31.7% and a macro F1 score of 26.0%, confirming its limitations in handling high-dimensional pixel data and inability to perform feature extraction.

Runtime and Practicality

The KNN algorithm had the shortest training duration (approximately 0.015 seconds) but exhibited the longest prediction runtime (15.89 seconds), significantly impacting its practicality for real-time or large-scale applications. The CNN, despite having the longest training time (22.61 seconds), delivered relatively efficient predictions (2.16 seconds), enhancing its suitability for practical deployment scenarios.

The MLP showed intermediate training time (4.46 seconds) and fast prediction runtime (0.80 seconds). However, its moderate predictive performance limits its practical utility, despite computational efficiency.

Interpretability

Interpretability is a critical aspect when selecting models for practical applications. The KNN algorithm is inherently interpretable due to its simplicity and transparent instance-based decision-making. In contrast, the CNN and MLP models lack straightforward interpretability, as their decision processes are concealed within complex network structures. This could limit their applicability in domains requiring transparency, such as healthcare or financial services.

Limitations

The study had several limitations, including:

- **Hyperparameter Tuning Limitations**: Due to computational constraints, hyperparameter optimisation was limited, potentially missing optimal configurations and underestimating true model performance.

- **Dataset Constraints**: The dataset's size, diversity, and representativeness could have affected generalisability and might not fully reflect real-world scenarios.

- **Lack of Interpretability**: The CNN's superior predictive performance is offset by lower interpretability, potentially hindering acceptance in regulated or sensitive domains.

5.2. Future Work Suggestions

To address the identified limitations, future research could focus on several concrete strategies:

Enhanced Hyperparameter Optimisation

Future studies should implement advanced hyperparameter tuning methods, such as Bayesian optimisation or genetic algorithms. These techniques efficiently explore large hyperparameter spaces, potentially uncovering better-performing configurations and leading to improved predictive outcomes.

Improved Feature Extraction and Dimensionality Reduction for KNN

Applying dimensionality reduction and advanced feature extraction techniques, such as Principal Component Analysis (PCA), autoencoders, or embeddings from pre-trained CNN models, could significantly enhance the predictive performance and runtime efficiency of KNN. This would make KNN more viable for large-scale or real-time applications by decreasing prediction latency.

Interpretability Techniques for CNN

To overcome interpretability concerns with the CNN model, future research should integrate explainability techniques such as SHAP (SHapley Additive exPlanations) or Grad-CAM. Such tools provide insight into CNN decision-making processes, making the model more transparent and facilitating its use in sensitive applications requiring trust and regulatory compliance.

Expanded and Diversified Dataset Evaluation

Future experiments should utilise larger, more diverse, and more representative datasets to test the robustness and generalisability of the evaluated models. Increased dataset diversity could provide deeper insights into model capabilities and limitations, ensuring more realistic assessments of model suitability for deployment.

Real-time Deployment and Practical Testing

Lastly, it would be valuable to explicitly assess model performance within real-time deployment contexts, taking into consideration key metrics such as latency, throughput, scalability, and computational resources. This will help identify and address practical constraints associated with model deployment, thereby enhancing overall applicability and usefulness of the developed solutions.

6. Reflection

6.1. Reflection for 520463341

The most important thing I learned during this assignment was how to critically evaluate and compare machine learning models not only in terms of predictive performance but also through practical trade-offs such as runtime efficiency, interpretability, and scalability. While CNNs achieved the highest accuracy on the dataset, implementing KNN and MLP revealed how architectural differences directly impact training behaviour and generalisation ability.

Additionally, I gained a deeper appreciation for the challenges of working with image data, particularly around preprocessing decisions like normalization and dimensionality reduction, and the importance of balancing model complexity with computational cost. Exploring hyperparameter tuning with grid search across all models reinforced the value of systematic experimentation, and I learned how tuning choices such as learning rate and dropout in neural networks can drastically influence convergence and overfitting. Collaboratively working through these design and analysis decisions helped me better understand the practical application of theory and the iterative nature of effective model development.

6.2. Reflection for 520494068

Through this assignment, I realised how important it is to design model architectures that actually suit the type of data and the problem we're trying to solve. Working with the PathMNIST dataset really showed me the strengths of convolutional neural networks (CNNs) - how they can pick up spatial patterns in medical images in a way that simpler models like MLPs or KNN just can't. I found it especially valuable to break down the components of a CNN, like filters, pooling layers, and dropout, and see firsthand how each one impacts performance and generalisation.

I also got to practise data preprocessing techniques like stratified sampling and normalisation, which helped make the training process fairer and more stable. One of the biggest takeaways for me was learning to look beyond just accuracy when evaluating models. Using other metrics like F1-score and also factoring in runtime helped me make more informed conclusions. Overall, the hands-on experience of tuning hyperparameters and comparing different models helped me connect the theory we learned in class to real-world machine learning workflows.

7. References

[1] J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, and B. Ni, “MedMNIST v2: A large-scale lightweight benchmark for 2D and 3D biomedical image classification,” Scientific Data, vol. 10, no. 1, pp. 1-15, 2023. https://doi.org/10.1038/s41597-022-01721-8

[2] M. Ahasan, “PathMNIST Image Classification_CNNmodels_PyTorch,” Kaggle, 2023. [Online]. Available: https://www.kaggle.com/code/mubtasimahasan/pathmnist-image-classification-cnnmodels-pytorch

[3] I. Koprinska, “Data,” COMP5318/COMP4318 Machine Learning and Data Mining Lecture Slides, Week 1b, University of Sydney, 2025.

[4] I. Koprinska, “Nearest Neighbor algorithm. Rule-Based Algorithms: 1R and PRISM,” COMP5318/COMP4318 Machine Learning and Data Mining Lecture Slides, Week 2, University of Sydney, 2025.

[5] T. Cover and P. Hart, “Nearest neighbor pattern classification,” IEEE Trans. Inf. Theory, vol. 13, no. 1, pp. 21-27, Jan. 1967. https://doi.org/10.1109/TIT.1967.1053964.

[6] I. Koprinska, “Deep Learning I: Feedforward Neural Networks,” COMP5318 Machine Learning and Data Mining Lecture Slides, Week 7, University of Sydney, 2025.

[7] S. Haykin, “Neural Networks and Learning Machines”, 3rd ed. Upper Saddle River, NJ, USA: Pearson, 2008. https://dai.fmph.uniba.sk/courses/NN/haykin.neural-networks.3ed.2009.pdf

[8] I. Koprinska, “Deep Learning II: Convolutional and Recurrent Networks,” COMP5318 Machine Learning and Data Mining Lecture Slides, Week 8, University of Sydney, 2025.

[9] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” Nature, vol. 521, pp. 436-444, May 2015. https://doi.org/10.1038/nature14539

[10] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, MIT Press, 2016. [Online]. Available: https://www.deeplearningbook.org

[11] C. M. Bishop, Pattern Recognition and Machine Learning, 1st ed., Springer, 2006. https://www.microsoft.com/en-us/research/wp-content/uploads/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf

[12] B. Subhash, "Explainable AI: Saliency Maps," Medium, Mar. 6, 2022. [Online]. Available: https://medium.com/@bijil.subhash/explainable-ai-saliency-maps-89098e230100

[13] I. Koprinska, “Probability-based Learning: Naïve Bayes. Evaluating Machine Learning Methods.,” COMP5318 Machine Learning and Data Mining Lecture Slides, Week 4, University of Sydney, 2025.

[14] A. Müller and S. Guido, Introduction to Machine Learning with Python, O’Reilly Media, 2016. https://www.nrigroupindia.com/e-book/Introduction%20to%20Machine%20Learning%20with%20Python%20(%20PDFDrive.com%20)-min.pdf