# Computer Vision: Neural Networks vs. Kernel Methods

**Reuben Gezang**
Royal Institute of Technology
gezang@kth.se

**Petrea Norgren**
Royal Institute of Technology
petrea@kth.se

**Filip Niklasson**
Royal Institute of Technology
filnik@kth.se

**Vilhelm Karlin**
Royal Institute of Technology
vilhelmk@kth.se

## Abstract

This report investigates computer vision in statistical learning in an attempt to present strengths and weaknesses of different models for the task at hand. The project focuses on Neural Networks vs. Kernel Methods in classifying tumors in brain MRI scans, using a dataset found on Kaggle. Theory regarding Neural Networks, Convolutional Neural Networks, Support Vector Machines, Histogram of Oriented Gradients (HOG) and K-fold cross-validation is presented to understand the statistical strengths and weaknesses of the models. In addition, metrics such as Precision, Recall, F1-score and Expected Calibration Error are presented to evaluate the different models. In conclusion, both models were successful in the task and showed equal accuracy but with differences in how the errors were distributed. The kernel method model was more dependent on feature engineering, utilizing HOG in a efficient way while the CNN model was more dependent on tuning and choosing optimal hyperparameters.

## 1 Introduction

Traditionally, radiologists examine MRI scans manually but this process is time-consuming, subjective, and prone to error. Automating this process by applying machine learning techniques could accelerate diagnosis and potentially produce more reliable diagnoses while simultaneously allow doctors to devote more time to treatment. Moreover, precise and accurate automatic diagnostics could help reduce disparities in treatment quality due to geographical variations in competence by enabling hospitals to diagnose or at least recommend further investigation even in the absence of on-site specialists.

In this study, we developed two machine learning models to classify brain tumors from MRI scans: a traditional kernel-based method and a convolutional neural network (CNN). By comparing their performance using various metrics such as precision, recall and F1 score we aimed to assess the feasibility of using computer vision approaches for tumor detection and classification, and to identify which modeling strategy provides the most accurate and well-calibrated predictions.

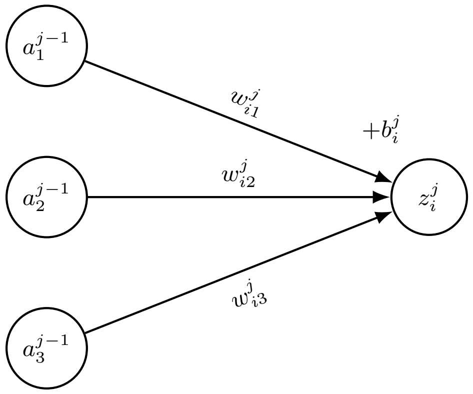
## 2 Methodology

The data[1] used in the project was a dataset consisting of 7022 images showing the results of brain MRI scans, of which 5022 contain brain tumors (categorized into three different types), and 2000 do not. Both a neural network and a support vector machine (SVM) were trained on the dataset, and the performance of the two trained models was evaluated and compared.

### 2.1 Neural Network model

One can think of a neural network as layers of nodes attached by edges where the value at each node is decided by the output signals from the previous layer, the weights of the edges and the bias attached to the current node. In this way the value at a single node, i, in layer, j, can be described as a vector equation in the format: $w_i^j a^{j-1} + b_i^j = z_i^j$ where $w_i^j$ is a vector that contains the weights of all edges connecting the node, i, to the previous layer, $j-1$, $a^{j-1}$ contains the output signals from all the nodes in the in that layer and $b_i^j$ is a scalar representing the bias at the node. The output signal from the node is then the result of an activation function on the result of the equation:

$$a_i^j = f_j^i(w_i^j a^{j-1} + b_i^j) = f_j^i(z_i^j)$$



By combining these vector equations of each node in a layer we can describe the transition between two layers as a matrix equation in the format:

$$z^j = W^j a^{j-1} + b^j = \begin{bmatrix} w_{1,1}^j & \cdots & w_{1,n}^j \\ \vdots & \ddots & \vdots \\ w_{m,1}^j & \cdots & w_{m,n}^j \end{bmatrix} \begin{bmatrix} a_1^{j-1} \\ \vdots \\ a_n^{j-1} \end{bmatrix} + \begin{bmatrix} b_1^{j-1} \\ \vdots \\ b_n^{j-1} \end{bmatrix}$$

[1]Link to dataset: `https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset/data`

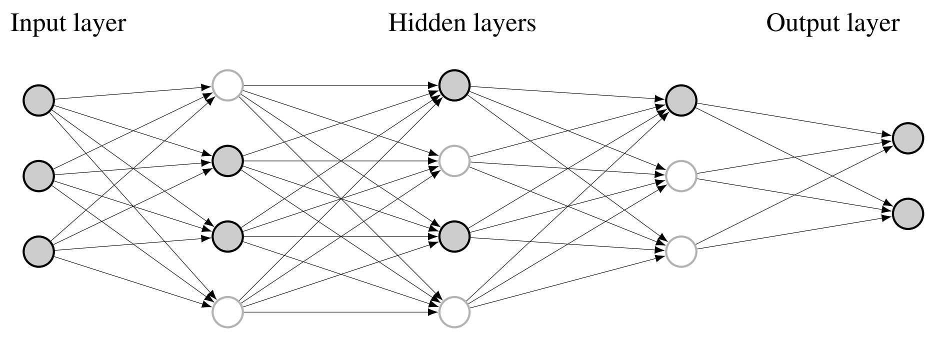
and by applying the activation function to the resulting vector we get the output signal, represented as a vector, of the entire layer:

$$a^j = f_j(z^j) = f(W^j a^{j-1} + b^j)$$

If we set the output of the first layer $a^0$ simply to be the initial input data $x$, it can easily be seen that the equation above forms a recursive formula of the output signals $a^j$ for all layers $j \in \{1, ...d\}$ where d represents the depth of the network: the number of layers excluding the input layer. Using this formula we can describe the entire network as a long series of nestled matrix multiplications with activation functions. Below let $\hat{y}$ be the final output of the network and let $x = a^0$ represent the initial input to the network:

$$\hat{y} = a^d = f_d(\underbrace{f_{d-1}(...\underbrace{f_1(x)}_{a^1}...)}_{a^{d-1}})$$

As mentioned above the first layer of a neural network: the input layer, is where we feed our input data and therefore the size: number of nodes, in this layer corresponds to the number of features in the input data. In our case the input data consisted of MRI scans of brains. However, before the data was sent into the neural network we used a convolutional neural network to extract the features and hence the size of the input layer was then a direct result of the design of the convolution neural network. For details on the CNN: see subsection 2.1.1. The size of the last layer: the output layer, is instead decided by the format of the output data. For example when designing a neural network for classification, as is the case in this project, the size of the output layer equals the number of classes. In our case the width/size of the last it was four corresponding to the three classes of tumors: meningioma, glioma, pituitary and the case of no tumor. The output of the last layer: $a^d$, which is a vector, contains a set of probabilities for how likely it is that that the input data belongs to each one of the classes and our prediction is then simply the class with the highest corresponding probability. When it comes to the layers in between: the hidden layers, both the number of layers and the size of each one of them is a question of design is to be decided by the engineer/-s.



When designing a Neural Network one has to weigh the benefits of a more complex model against the risk of overfitting/decreasing generalization and also take into consideration the the amount of training data available (a more complex model needs more training data) and the computational cost among other things. The final design of our Neural network has 4 convolutional layers, 4 pooling layers (see next subsection) and two linear layers. The training of an NN is a optimization problem where we want to adjust the weights and biases of the model in order to minimize a certain loss function. This was done trough backpropagation where the gradients of the loss function are calculated for the training data and the weights are adjusted accordingly. We used cross entropy loss:

$$L(f_\theta(x_i), y_i) = \sum_{j=1}^{4} \mathbf{1}_{y_i=j} \log(p_\theta(x_i)_j)$$

where above $y_i, x_i, f_\theta(x_i), p_\theta(x_i)_j$ represent the true label/class, the input, the output of the NN given the current weights and biases: $\theta$ and the predicted probability of class $j$ given the input $x_i$ given the current $\theta$ and in order to avoid the issue of a vanishing gradient[2] we used ReLu: $max(0, z)$ as our activation function that has a gradient of simply 0 or 1 w.r.t $z$. In order to ensure a more

[2]A common issue when training multilayered NNs that occurs due to the multiplication of numerous gradients with values close to zero as an effect of the chain rule.

robust model we added a 0.25 dropout rate between the two linear layers in our NN. This means that during training we randomly set $25\%$ of the output values from the nodes of the first linear layer to zero before sending the signals through to the next layer. In this way we make sure that the model is not too dependent on one node/intermediate feature in particular. A well calibrated NN should have about the same confidence as accuracy in its predictions[3] meaning that out of the predictions that were had $x\%$ confidence about $x\%$ of them should be accurate. One way to measure this is by looking at the ECE score: Expected calibration error (See subsection 2.4.3). When deciding upon the hyper parameters we used a randomly selected validation set from the training data. After tuning the parameters we then trained a new model using the chosen parameters with a new random validation set in order to make sure the results was not just the effects of a "happy split".

#### 2.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a major success story for image classification. The reason for this is that the network can learn features using hierarchal fashion. The early layers detect simpler features such as corners and edges while intermediate layers combine these to detect shapes and patterns. Finally, deeper layers assemble complex structures to recognize the target objective, i.e the tumors. The hierarchy is enabled by convolutional- and pooling layers.

For this project, we are using 2D convolution for images:

$$(w * x)(k,l) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} w(i,j)x(k+i,l+j) \tag{1}$$

In equation (1), $W$ refers to the filter (kernel) and $x$ is the signal from the previous layer (or the image pixels in the first layer). The filters $W_{lk}$ are learned during training and are each a 2D matrix of size $m \times m$, in our case $m = 3$. The filters are then slid across the input computing a dot product at each location. If the results is large it means that the local patch matches the target and a low score means that there is a mismatch. Thus, we create a form of feature map that highlights patterns. This can be illustrated by:

$$\text{Input Image} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix} \quad \text{Convolution Filter} = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix} \tag{2}$$

$$\text{Convolved Image} = \begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta \\ g\alpha + h\beta + j\gamma + k\delta & h\alpha + i\beta + k\gamma + l\delta \end{bmatrix}. \tag{3}$$

After convolutional layers and using ReLU on the output we want to use pooling layers. Pooling is a way to build invariance through signal transformation. It was decided to use max pooling with stride $p = 2$. This means that each $2 \times 2$ block is replaced by its maximum. This reduces the spatial dimension with a factor of 2 in each dimension and sharpens feature identification by keeping the strongest activations.

$$\text{Max pool} \quad \begin{bmatrix} 5 & 1 & 2 & 8 \\ 3 & 4 & 5 & 5 \\ 17 & 2 & 3 & 8 \\ 7 & 3 & 6 & 9 \end{bmatrix} \longrightarrow \begin{bmatrix} 5 & 8 \\ 17 & 9 \end{bmatrix} \tag{4}$$

Using the theory presented in the lectures and after searching by ourselves, the final architecture of the Neural Network was decided and implemented using pytorch. The architecture can be found together with our choice of hyperparameters in the appendix.

### 2.2 Kernel Methods

For the second model, using a kernel method, we decided to use a support vector machine (SVM) with a Gaussian kernel (RBF) as the kernel. SVM's are are supervised learning models that attempts

[3]As mentioned in the text the output of a classifying NN is a vector of probabilities and the final prediction is the class with the highest probability.

to find the optimal separating hyperplane that maximizes the margin between different classes.

The optimization is based on minimizing the hinge loss function, defined as

$$L(y_i, f(x_i)) = max(0, 1 - y_i f(x_i)).$$

In totality, we can formulate the SVM dual problem, as described by the lecture notes

**SVM Dual Problem:**

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k k(\mathbf{x}_i, \mathbf{x}_k)$$

subject to

$$0 \leq \alpha_i \leq C, \quad \text{and} \quad \sum_{i=1}^{N} \alpha_i y_i = 0.$$

The final classifier is a non-linear function of $\mathbf{x}$:

$$\hat{f}(\mathbf{x}) = \text{sign}\left( \sum_{i=1}^{N} \hat{\alpha}_i y_i k(\mathbf{x}, \mathbf{x}_i) + \hat{\beta}_0 \right)$$

where the sum is only over the support vectors (observations for which $\hat{\alpha}_i > 0$). This loss only penalizes points that fall on the wrong side of the margin. For multiclass classification (four classes in our case), SVMs typically use either a one-vs-rest or one-vs-one strategy. In a one-vs-rest classification, which was the option we used, one trains a classical binary SVM per class, treating that class as positive and all others as negative. During prediction, all $k$ classifiers are evaluated and the class with highest decision score is chosen.

$$\hat{y} = argmax_{j \in \{1,..,k\}} f_j(x)$$

In the one-vs-one case the classifier is trained on each pair of classes, leading to $\frac{k(k-1)}{2}$ classifiers for $k$ classes. The predictions are then made by majority voting among the binary classifiers. Meaning that we have 6 classifiers and if $f_{ij}(x) > 0$ the vote goes to class $i$ and $j$ otherwise. The final decision becomes:

$$\hat{y} = argmax_{i \in \{1,...,k\}} \text{votes}(i).$$

In this project it was decided to implement scikit-learn's model for SVMs that uses the one-vs-rest approach.

In the case where data is not linearly separable in the original feature space, the kernel trick allows SVMs to map the input into a higher-dimensional space where a linear separation between classes might exist. The Gaussian kernel (RBF) is defined as

$$K(x_i, x_J) = \exp(-\frac{||x_i - x_j||^2}{2\sigma^2}) \tag{5}$$

which can be veiwed as placing a Gaussian function around each datapoint. The hyperparameters for the model (to be decided by the experimenter) are $C$ (regularization), and $\gamma = \frac{1}{2\sigma^2}$ (Kernel width). Together they determine the bias-variance tradeoff. $C$ regulates the penalty for misclassification and $\gamma$ determines how tightly each training sample influences the decision boundary.

As mentioned, kernel methods rely on the idea of mapping input data into a high- or infinite-dimensional feature space where linear methods can capture nonlinear relationships. The key concept of the kernel trick, as mentioned in the lecture notes, is that instead of explicitly computing the mapping $\phi(x)$ we define a kernel function.

$$k(x, x') = \langle \phi(x), \phi(x') \rangle$$

Using RBF we can show that

$$\begin{aligned} k(x, x') &= \exp\left(-\frac{(x - x')^2}{2\sigma^2}\right) \\ &= \exp\left(-\frac{x^2}{2\sigma^2}\right) \exp\left(-\frac{x'^2}{2\sigma^2}\right) \sum_{m=0}^{\infty} \frac{(xx'/\sigma^2)^m}{m!} \\ &= \phi(x)^T \phi(x') \end{aligned}$$

where

$$\phi(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right)\left(1, \frac{x}{\sigma}, \frac{x^2}{\sigma^2\sqrt{2!}}, \frac{x^3}{\sigma^3\sqrt{3!}}, \dots\right)^T.$$

meaning that the feature map is infinite (!).

#### 2.2.1 Histogram of Oriented Gradients (HOG)

While it is theoretically possible to use the direct pixel values as the features to the model, there are better ways of constructing the features for use with SVMs for image classification. Some concrete problems with using direct pixel values is that each feature would become incredibly sensitive to e.g. small translations or rotations, and different contrasts or brightnesses across images. Furthermore, for a 128x128 pixel image, that would mean 49 152 features per image.

It is therefore more common to use HOG features. HOG has a few steps which seek to make each feature more informative, and to furthermore be less sensitive to the changes above.

1. Each image is first converted to grayscale and split into cells of $8 \times 8$ pixels, yielding a $16 \times 16$ grid of cells for a $128 \times 128$ image.
2. For each pixel, the gradient in both x and y directions is computed. The gradient magnitude and orientation are then given by

$$\text{Magnitude} = \sqrt{(\Delta x)^2 + (\Delta y)^2}, \quad \text{Orientation} = \arctan\left(\frac{\Delta y}{\Delta x}\right).$$

3. Within each cell, a histogram of gradient orientations is constructed. The orientation space $(0°, 180°)$ is divided into 9 bins, and each pixel's gradient magnitude votes for its corresponding orientation bin.
4. Cells are grouped into overlapping $2 \times 2$ blocks (yielding $15 \times 15$ blocks in total). For each block, the concatenated histogram (4 cells $\times$ 9 bins = 36 values) is normalized by its $L^2$ norm to achieve robustness to illumination and contrast changes.
5. The normalized histograms from all blocks are concatenated into a single feature vector of length $15 \times 15 \times 36 = 8100$.

This representation captures local edge structure while remaining invariant to small translations and robust to lighting variations, making it well-suited for SVM classification.

### 2.3 K-Fold Cross-Validation and Hyperparameter Selection

To select the optimal regularization parameter $C$ for the SVM, we employ 5-fold cross-validation. The training set is partitioned into 5 stratified folds, and for each candidate value of $C$, the model is trained on 4 folds and validated on the remaining fold. This process is repeated for all folds, and the $C$ yielding the highest average validation accuracy is selected. The kernel width parameter $\gamma$ is set using the heuristic $\gamma = \frac{1}{n_{\text{features}} \cdot \text{Var}(X)}$, corresponding to scikit-learn's `gamma='scale'` option. Initially we started with a wide spread set of alternative $C$ ranging from 0.01 to 10.0 and then refined the grid as we got closer to the optimal $C$ value. In the final version of the SVM we had $C = 4.5$.

### 2.4 Metrics and Evaluation

In order to properly assess the performance of the models studied, their performance needs to be quantified. There are many methods of doing this, and different methods confer different advantages and disadvantages, and capture different aspects of model performance. This section will present the metrics used in this report for model evaluation, and comment on their specific advantages and disadvantages where deemed appropriate.

Prior to presenting the actual metrics, let us introduce some notation to be used in this section. Assume we aim to classify $N$ data points $\{x_i\}_{i=1}^N$, belonging to some space $X$, such that $x_i \in X$, into $K$ classes. Also assume there is some true classification of all $x_i$ into these $K$ classes. Let these classes be denoted by a unique number $k \in \{1, 2, 3, ..., K\} := Y$, and let $y_i \in Y$ be the true class of $x_i$. Now further assume that there is a predictor $\hat{f} : X \to Y$, taking in one data point and returning the class to which it is assigned by this predictor.

### 2.4.1 Precision and Recall

Precision and recall measures provide a tool for exploring the performance of a predictive classification model in more detail. The precision and recall measures can be calculated with respect to a specific class, and provide insights as to the performance of the model as it relates to that class, or with respect to the whole model. In the context of this report, the images are classified into four classes: one corresponding to the absence of a tumor, and three classes for different types of tumors.

Precision can be simply described as the proportion of units predicted to be part of a class which are actually part of that class, or as the ratio of true positives to all positives. In formal terms, precision might be described as follows:

The *precision* of $\hat{f}$ with respect to class $k$ is denoted $P_k$ and given by

$$P_k = \frac{\sum_{i\in\{j:\hat{f}(x_j)=k\}} \mathbf{1}(y_i = k)}{|\{j : \hat{f}(x_j) = k\}|} \tag{6}$$

where $\mathbf{1}$ is the indicator function. Recall is similarly defined as a ratio, but instead between the number of true positives to the total number of units in the class in question, or in other words the ratio of all instances of a class which were correctly classified as being members of that class. Formally, *recall* with respect to a class $k$, denoted $R_k$ is defined as

$$R_k = \frac{\sum_{i\in\{j:y_j=k\}} \mathbf{1}(\hat{f}(x_i) = k)}{|\{j : y_j = k\}|} \tag{7}$$

The definitions given above all relate precision and recall to specific classes. These metrics can, however, be expanded to the full dataset, by applying it to all classes. In such a case, the overall precision $P$ is given by

$$P = \frac{\sum_{k=1}^{K}[\sum_{i\in\{j:\hat{f}(x_j)=k\}} \mathbf{1}(y_i = k)]}{\sum_{k=1}^{K} |\{j : \hat{f}(x_j) = k\}|} = \frac{\sum_{i=1}^{N} \mathbf{1}(y_i = k)}{N} \tag{8}$$

As can be seen in (8), this is equivalent to overall prediction accuracy. For recall however, generalized to the entire dataset, denoted $R$, we get

$$R = \frac{\sum_{k=1}^{K} \sum_{i\in\{j:y_j=k\}} \mathbf{1}(\hat{f}(x_i) = k)}{\sum_{k=1}^{K} |\{j : y_j = k\}|} = \frac{\sum_{i=1}^{N} \mathbf{1}(y_i = k)}{N} \tag{9}$$

Once again, this is the overall prediction accuracy. Using this measure across all classes, the precision and recall assume the same value, which is the overall prediction accuracy.

### 2.4.2 F1-score

The F1-score combines precision and recall into one metric, reflecting the overall performance of the predictive model, possibly with respect to some class $k$. It is formally defined as the harmonic mean of the precision and the recall.

Let $P$ be the precision of a given test $T$, and let $R$ be the recall of $T$. The F1-score, denoted $F1$ is then given by

$$F1 = \frac{2}{P^{-1} + R^{-1}} \tag{10}$$

Note that since $P, R \in [0, 1]$, it follows that $F1 \in [0, 1]$. A larger F1 score is preferable.

### 2.4.3 Expected Calibration Error

So far, the metrics presented have assumed that the model assigns each data point to only one class. Many models, however, are also able to provide estimates concerning the confidence that a data point belongs to a certain class. The expected calibration error (ECE) is a metric showing how accurate these estimates are.

The ECE is calculated by comparing the confidence of the predictor with the proportion of correctly classified data points. If these match well, the model confidence estimates are considered reliable.

In order to arrive at an ECE value in practice, certain approximations and estimates have to be made. Firstly, the data points $\{x_i\}_{i=1}^N$ are classified by the predictor $\hat{f}$, and assigned to $M$ bins $B_m$, $m = 1, 2, ..., M$ based on the highest confidence associated with each data point, or in other words, the confidence that the data point belongs to the class it was classified to. Let $\hat{p}_i$ be the confidence given by $\hat{f}$ that $x_i$ belongs to class $\hat{f}(x_i)$. Now bins of arbitrary number and size, although usually of equal width, are introduced, together spanning all of $[0, 1]$, and the data points are assigned to bins in accordance with

$$\begin{cases} a_m \leq \hat{p}_i < b_m \implies x_i \in B_m & m < M \\ a_m \leq \hat{p}_i \leq b_m \implies x_i \in B_m & m = M \end{cases} \tag{11}$$

where $a_m, b_m \in [0, 1]$ define the lower and upper boundaries of bin $m$, respectively. In our implementation we used 10 bins with the following spans: $[0, 0.1), [0.1, 0.2), ..., [0.9, 1]$. The ECE is then given by

$$\sum_{m=1}^{M} \frac{|B_m|}{N} \left| \frac{1}{|B_m|} \sum_{x_i \in B_m} \mathbf{1}(\hat{f}(x_i) = y_i) - \frac{1}{|B_m|} \sum_{x_i \in B_m} \hat{p}_i \right| \tag{12}$$

In plain terms, this is the average estimated discrepancy between actual prediction accuracy and the confidence in the predictions given by the model. Note that in this case, the ECE is only applicable to the neural network model, since the SVM only results in one single class prediction for each data point, and thus gives no output from which confidences can be estimated.

## 3 Results

This section presents the performance metrics of the trained models on the previously unseen test dataset.

| Test data results | | | | |
|---|---|---|---|---|
| Category | Test Accuracy | Correct | Incorrect | ECE(10 bins) |
| CNN | 97.64% | 1280/1311 | 31/1311 | 0.0436 |
| Kernel Method | 97.64% | 1280/1311 | 31/1311 | NA |

Table 1: Results for both models on the test data

| CNN metrics | | | | |
|---|---|---|---|---|
| Category | Precision | Recall | F1 score | Support |
| Glioma | 0.99 | 0.95 | 0.97 | 300 |
| Meningioma | 0.95 | 0.96 | 0.95 | 306 |
| No tumor | 0.99 | 1.00 | 0.99 | 405 |
| Pituitary | 0.98 | 0.99 | 0.99 | 300 |
| Accuracy | | | 0.98 | 1311 |
| Macro avg | 0.98 | 0.97 | 0.98 | 1311 |
| Weighted avg | 0.98 | 0.98 | 0.98 | 1311 |

Table 2: Key performance metrics of the trained neural network.

| SVM metrics | | | | |
|---|---|---|---|---|
| Category | Precision | Recall | F1 score | Support |
| Glioma | 1.00 | 0.92 | 0.96 | 300 |
| Meningioma | 0.92 | 0.99 | 0.95 | 306 |
| No tumor | 0.99 | 1.00 | 1.00 | 405 |
| Pituitary | 0.99 | 0.99 | 0.99 | 300 |
| Accuracy | | | 0.98 | 1311 |
| Macro avg | 0.98 | 0.97 | 0.97 | 1311 |
| Weighted avg | 0.98 | 0.98 | 0.98 | 1311 |

Table 3: Key performance metrics of the trained support vector machine.

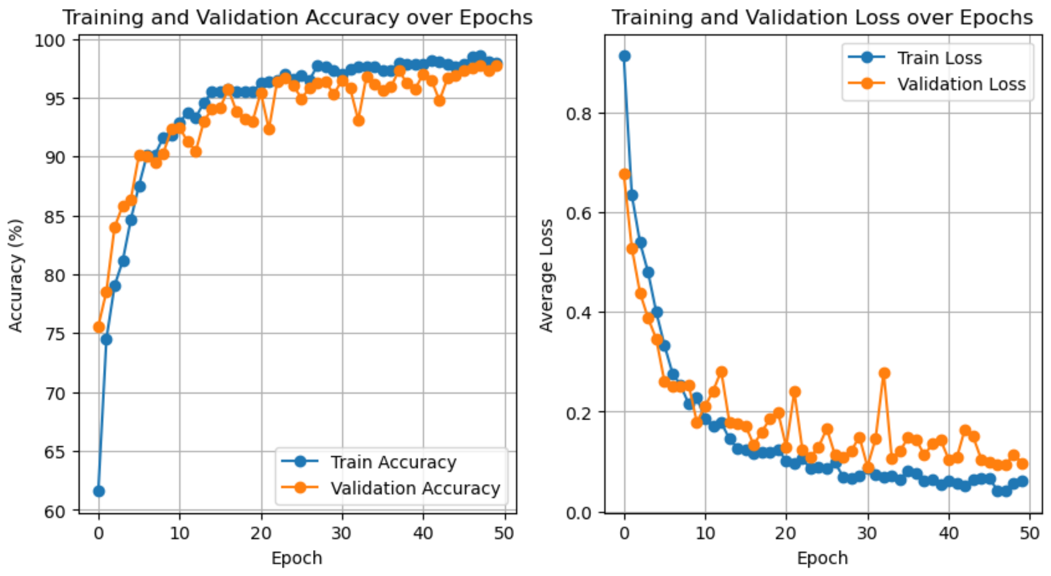
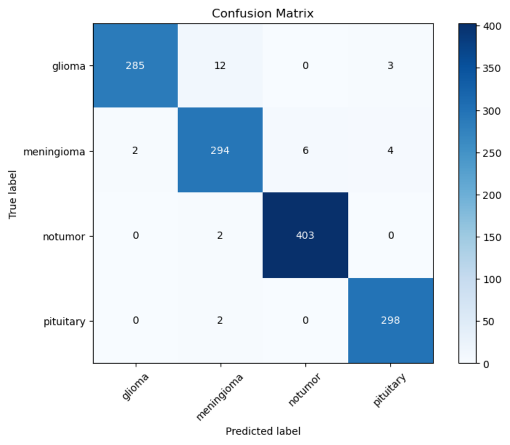
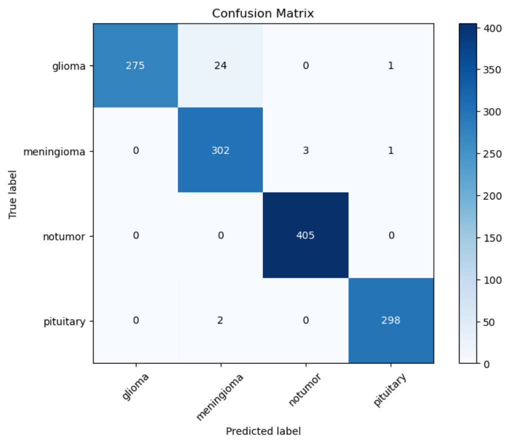


Figure 1: Training and validation loss/accuracy over epochs for the neural network model. The loss function used was cross-entropy loss.



(a) Confusion matrix for trained CNN



(b) Confusion matrix for trained kernel method (SVM)

Figure 2: Confusion matrices for the CNN and SVM models (Test dataset).

## 4 Conclusion

In this study, we compared a Convolutional Neural Network (CNN) and a Support Vector Machine (SVM) in classifying brain tumors from MRI scans. Both approaches were found to be highly effective, achieving identical test accuracies of 97.64%. This success underscores the viability of both more modern deep learning and classical machine learning paired with robust feature engineering, for medical image classification.

While both models have identical overall accuracies, a more granular analysis of the confusion matrices reveals certain important distinctions. For instance, the SVM's errors were heavily concentrated, with its misclassification of gliomas as meningiomas accounting for nearly 80% of its total errors (24 out of 31), while the CNN's errors were somewhat more evenly distributed. This shows that the models have distinct classification boundaries and that equal accuracy does not imply equal decision strategies and error patterns. If we assume that the worst scenario is that a model predicts that no tumor is present while in fact it is then the SVM model seems preferable in this case since this happens in three vs six cases for the respective models. On the other hand the more evenly distributed error pattern in the CNN case suggests more stable and balanced decision boundaries.

The success of both models can be attributed to their underlying statistical principles. The CNN possesses inherent inductive bias for spatial hierarchies, allowing it to learn relevant features automatically. While this end-to-end learning offers a "plug-and-play" advantage, it comes at the cost of a much larger hyper-parameter space which we did not exhaustively explore and optimize. The CNN, with its vast number of parameters, is a high-variance model prone to overfitting, and so regularization via dropout was essential for prevent overfitting and ensure good generalization. The CNN's predictions were shown to be well-calibrated, with a relatively low Expected Calibration Error (ECE).

The SVM, being a structurally simpler model, largely owes its success to the effectiveness of the HOG features. The comparable results on this dataset indicates that the HOG features managed to successfully encode the critical edge and shape information necessary for the kernel method to operate. Being a structurally simpler model, the SVM is less prone to overfitting, and the bias-variance trade-off was effectively managed by optimizing the regularization parameter $C$ on the validation set.

## Appendix

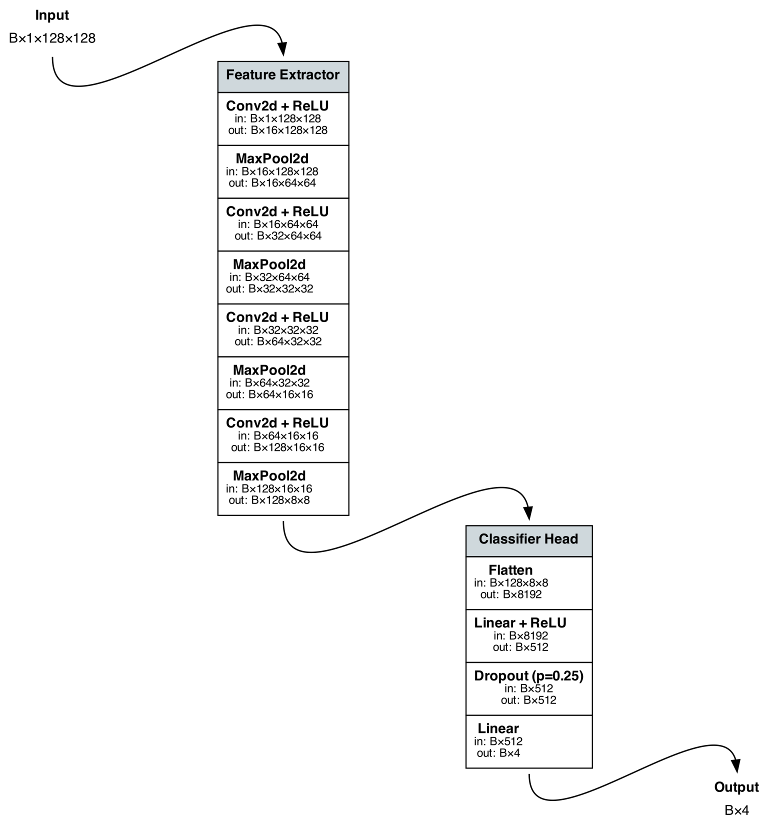


Figure 3: Final Architecture of CNN

| **Hyperparameter** | **Value** |
|---|---|
| Number of Epochs | 50 |
| Optimizer | Adam |
| $\lambda$ (Learning rate) | 0.001 |
| Loss function | Cross-Entropy loss |
| Drop-out | $p = 0.25$ |
| Batch size | 32 |
| 2D convolutional padding | 1 |
| 2D convolutional kernel | 3 |
| Max pool kernel size | 2 |
| Max pool stride | 2 |

Table 4: Hyperparameters chosen for the Neural Network

### 4.1 Dataset

The dataset used is accessible at `https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset/data`.

### 4.2 Disclosure of AI use

This text has been formatted using AI tools such as ChatGPT and Claude. These have been primarily used for polishing written text and reformulating thoughts. Furthermore these AI tools helped with making plots/tables and structuring python-code.

### 4.3 Git Repository

For this project the coding was done using python. There is a git repository containing the code used.https://gits-15.sys.kth.se/gezang/SF2935-Project

## References

[1] Lim, S. H. (2025). Modern Methods of Statistical Learning: lecture slides. Available at https://canvas.kth.se/courses/56658.

[2] PyTorch Contributors. (2025). PyTorch: An open source machine learning framework. Available at: https://pytorch.org/ (Accessed: October 17, 2025).

[3] Scikit-learn Developers. (2025). Scikit-learn: Machine learning in Python. Available at: https://scikit-learn.org/stable/ (Accessed: October 17, 2025).

[4] Bishop, C. M. & Bishop, H. (2024). Deep Learning: Foundations and Concepts.

[5] Dalal, N. and Triggs, B. (2005). *Histograms of Oriented Gradients for Human Detection.* Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), vol. 1, pp. 886–893. Available at: https://doi.org/10.1109/CVPR.2005.177

[6] KTH Royal Institute of Technology. (2025). DD1420 Lecture Notes. Available at: https://dd1420.notion.site/DD1420-Lecture-Notes-b555e017345a4119950ce8fd67133275 (Accessed: October 17, 2025).