FYS-STK4155 Project 3

Trude Halvorsen & Tiril Trondsen

University of Oslo, Department of Physics

(Dated: December 14, 2024)

I. ABSTRACT

The Fashion MNIST dataset is a challenging benchmark for evaluating image classification models, offering a more complex alternative to the traditional MNIST dataset of handwritten digits. In this project, we compared the performance of Convolutional Neural Networks (CNNs) and Support Vector Machines (SVMs) in classifying images from the dataset. CNNs, with their ability to capture hierarchical features, performed exceptionally well, achieving an overall F1-score of 0.92 and proving to be highly effective for image-based tasks. SVMs, while showing robustness in handling high-dimensional data and achieving a relatively high F1-score of 0.88, faced difficulties in distinguishing visually similar classes. These results highlight both the strengths and limitations of each method, providing valuable insights into their use for real-world classification problems.

II. INTRODUCTION

The Fashion MNIST dataset has become a widely used benchmark for evaluating machine learning models, particularly in image classification tasks. As an alternative to the original MNIST dataset of handwritten digits, Fashion MNIST provides a more complex set of grayscale images depicting various types of clothing and accessories [1]. This dataset offers an opportunity to test and improve classification algorithms on challenging, high-dimensional data.

In this project, we focus on Convolutional Neural Networks (CNNs) and Support Vector Machines (SVMs) to classify images from the Fashion MNIST dataset. CNNs were initially developed for character recognition tasks, and have since become central to the success of deep learning methods [2]. CNNs are different from other types of neural networks by their superior performance in image classifications [3]. Unlike traditional neural networks, CNN architectures are specifically designed with the assumption that the input data consists of images. They use specialized layers to extract features from raw pixel data, gradually getting an understanding of the input. This design allows certain features to be built into the structure, which helps the network process data more efficiently and use fewer parameters compared to traditional neural networks [2]. Moreover, SVMs provide a robust alternative for smaller and medium sized datasets, using hyperplanes and kernels to effectively solve classification problems. This enables us to compare the performance of CNNs and SVMs.

The report is organized as follows: The Methods section provides an overview of the CNN and SVM architectures, details the Fashion MNIST dataset, describes the training, evaluation and implementation process. Results and Discussion present the performance of each model, including metrics such as accuracy and F1-scores, and highlight key insights through confusion matrices. Finally, the Conclusion summarizes the main findings and outlines potential directions for future research.

This project aims to enhance our understanding of machine learning techniques for image classification by evaluating CNNs and SVMs performance on a challenging dataset, and applying theoretical concepts in practice.

III. METHODS

In this study, we employed two different machine learning techniques to model and study the Fashion MNIST dataset. In this section, we present the models used: Convolutional Neural Networks (CNNs) and Support Vector Machines (SVMs).

A. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) were implemented to analyze the Fashion MNIST dataset. As mentioned in the introduction, CNNs are particularly effective for image classification tasks because of their ability to capture patterns in image data [2]. As the data progresses layer by layer through the CNN, the network learns to recognize increasingly complex elements and shapes within the image, until it finally identifies the object image [3]. We can think of an image as a grid of numbers, where each represent the brightness of the pixel [4]. CNNs transform the input data, starting with raw pixel values, into feature representations that result in the final class scores [2]. This process allows CNNs to achieve high accuracy in problems like image recognition and classification.

1. Overview of CNNs Architecture

CNNs use various filters on pixel data to extract and learn features, which are then used for classification [4]. A typical CNN architecture consists of multiple layers, primarily including three main types: Convolutional Layers, Pooling Layers, and Fully-Connected Layers. A visualization of a CNN architecture is presented in Figure 1.

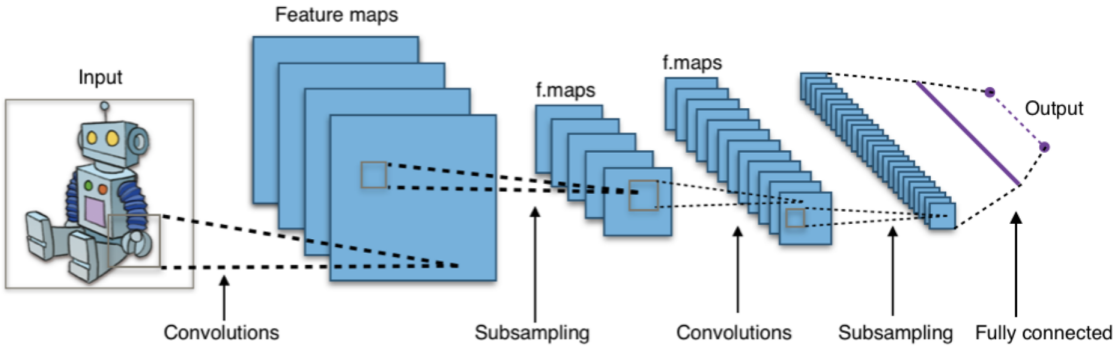

FIG. 1: Visualization of CNN. The image is adapted from [4].

The CNN architecture we made for this project is described below:

The convolutional layer is the first layer in our CNN. We apply convolutional operations to extract features from the input images. Each convolutional layer applies a set of filters to detect specific patterns. For each layer, the CNN identifies more of the image, earlier layers detect simpler features and as the data progresses through more layers, it recognizes even more complex shapes [3]. We used two convolutional layers, with the first layer containing 32 filters of size 3 x 3, with an input shape matching the shape of one image. The second covolutional layer containes 64 filters of size 3 x 3. Both layers use the non-linear Rectified Linear (ReLu) activation function, which "serves as the activation of the neurons in the first convolutional layer" [2]. Further details on the ReLU function are described in Project 2 [5].

In addition to discrete convolutions, pooling operations are another essential component of CNNs. After the convolutional layers, a pooling layer is applied to downsample the feature maps along the spatial dimensions (width and height) [2]. This reduces the number of parameters and improves computational efficiency [3]. The most common type is max pooling, which selects the maximum value within a defined small region of the feature map [2]. We used a max pooling layer with a 2x2 filter. The feature maps produced by the pooling layers are flattened to prepare the data for the fully-connected layers.

Finally, at the end of the CNN architecture, we have the fully-connected layers, which act as a normal Feed Forward Neural Network (FFNN) [2]. Feed Forward Neural Networks were extensively described in our Project 2 [5]. These fully-connected layers act as classifiers and are responsible for performing the final classification output [6]. They compute the class scores by combining the features into predictions. For this project, we used one hidden dense layer with 128 units and the ReLU activation function, and an output layer with 10 units corresponding to the 10 classes we have in the Fashion MNIST dataset, with the softmax activation function. The softmax is also described in detail in Project 2 [5].

This combination of layers helps the CNN gradually understand the input images, starting with basic patterns and ending with more detailed features that allow it to classify images

accurately.

2. Mathematics of CNNs

The key component for CNNs lies in the mathematical operation of convolution. As described by Hjorth-Jensen, "convolution is represented by mathematical operations on two functions in order to produce a third function that expresses how the shape of one gets modified by the other" [2]. These operations enable CNNs to extract features from structures in the input data.

For one-dimensional data, convolution is mathematically defined as:

$$y(t) = \int x(a)w(t-a)da,$$

Here, $x(a)$ represents the input, $w(t-a)$ is the weight function and $y(t)$ is the resulting output. This integral can be compactly expressed as:

$$y(t) = (x * w)(t).$$

For discrete data, like in image analysis, the convolution operation becomes:

$$y(t) = \sum_{a=-\infty}^{a=\infty} x(a)w(t-a).$$

In the case of two-dimensional data, such as images, the convolution operation is generalized as:

$$Y(i,j) = (X * W)(i,j) = \sum_m \sum_n X(i-m, j-n)W(m,n).$$

Where X is the input image, W is the filter and Y is the resulting feature map.

B. Support Vector Machines

Support Vector Machines (SVMs) are "particularly well suited for classification of complex but small-sized or medium-sized datasets" [2]. They are machine learning algorithms widely used for both classification and regression tasks. One of the main advantages of SVMs in image classification is their ability to effectively handle high-dimensional data, such as images, by identifying hyperplanes that separate different classes. Additionally, SVMs are generally less prone to overfitting compared to algorithms like neural networks [7].

1. Theory of SVMs

The theory behind SVMs is based on the concepts of hyperplanes and margins to separate different classes in a classification problem. A hyperplane is the decision boundary that separates two classes of data points. In two dimensions, the hyperplane is a line, while in higher dimensions, it becomes a plane. SVM aims to maximize the margin, which is the distance between the data points from each class and the hyperplane (support vector). A larger margin makes a clear separation between classes, so the model is more likely to classify new data points correctly and less likely to make mistakes [2].

2. Mathematics of SVMs

The mathematics of SVM is based on describing hyperplanes, which are the decision boundary that separates different classes. As described by Hjorth-Jensen, "all points on one side of the plane will belong to class one and all points on the other side of the plane will belong to the second class two" [2]. In two dimensions, the hyperplane separating two classes is given by:

$$b + w_1 x_1 + w_2 x_2 = 0,$$

Where w is the vector orthogonal to the hyperplane and b is the intercept.

We can rewrite the above equation on vector form as:

$$\boldsymbol{x}^T \boldsymbol{w} + b = 0.$$

Where we have the vectors $\boldsymbol{x}$ and $\boldsymbol{w}$.

The margin M is the distance from the hyperplane to the closest data points (support vectors). To maximize M the constraints are:

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq M \; \forall i = 1, 2, \ldots, p.$$

By scaling, the margin is normalized to $M = \frac{1}{\|\mathbf{w}\|}$, leading to:

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 \; \forall i.$$

To allow for some misclassification, slack variables are introduced, and the constraints then become:

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - (1 - \xi_) \geq 0 \; \forall i.$$

The regularization parameter C controls the tradeoff between maximizing the margins, and minimizing the misclassification error. Using a higher C value will prioritize minimizing the classification error, while a lower C value will allow a larger margin, but then may lead to more misclassifications.

In SVM classification, there are several kernels being used to allow the algorithm to work in higher dimensions, by computing inner products. The most used kernel functions are:

- **Linear kernel:** $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$
- **Polynomial Kernel:** $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + c)^d$
- **Radial Basis function (RBF) Kernel:** $\exp(-\gamma||\mathbf{x}_i - \mathbf{x}_j||^2)$

The optimization can be rewritten using Lagrange multipliers, and the variable kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$:

$$\mathcal{L} = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

Subject to the constrain:

$$\sum_i \lambda_i y_i = 0, \lambda_i \geq 0$$

C. The Fashion MNIST Dataset

In this section, we will give an overview of the Fashion MNIST Dataset used in our project. We found the Fashion MNIST dataset on Kaggle, provided by Zalando Research, for training and evaluating our models [1].

Fashion MNIST is designed as an alternative to the original MNIST dataset, with the same image dimensions and training and testing structure. However, it provides a more challenging benchmark for machine learning algorithms because of the increased complexity of the clothing images, compared to the simpler handwritten digits in MNIST.

The Fashion MNIST dataset contains a total of 70 000 grayscale images of Zalando's article images. All images consist of pixels arranged in order, where each image has dimesions of 28x28 pixels. Each pixel in an image has a value between 0 and 255, representing the pixel's intensity, meaning the lightness or darkness, where higher values indicate darker pixels.

These images are divided into:

- Training set: 60 000 images used to train our models.
- Testing set: 10 000 images for evaluating the performance of our models.

We have 10 classes, representing the different types of clothing items:

1. T-shirt/top
2. Trouser
3. Pullover
4. Dress

5. Coat
6. Sandal
7. Shirt
8. Sneaker
9. Bag
10. Ankle boot

Figure 2 provides a visualization of selected samples from the Fashion MNIST dataset, showing the variety of clothing items included.

FIG. 2: Selected samples from the Fashion MNIST dataset, illustrating the various clothing items.

We can visualize individual images from the Fashion MNIST dataset with their corresponding pixel intensity values, as shown in Figure 3. This gives an understanding of how the image data is structured.

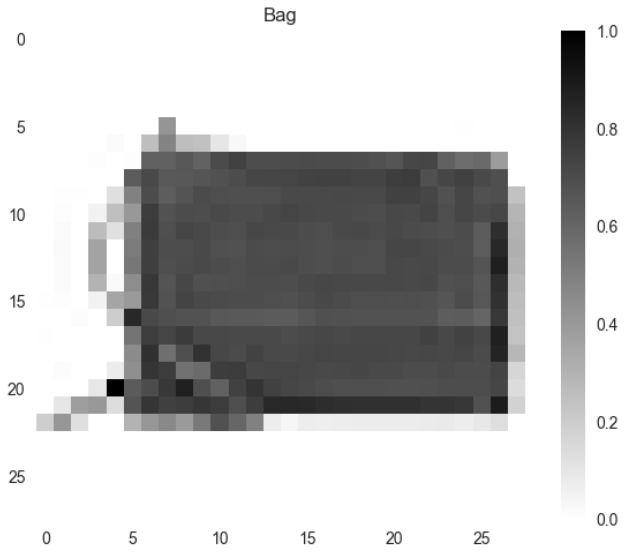

FIG. 3: Visualization of a sample image from the Fashion MNIST dataset, including its pixel intensity values. The image represents the class "Bag" and shows the grayscale intensity.

D. Training and Evaluation

To train and evaluate the CNN, we included hyperparameter tuning, accuracy analysis, and confusion matrix visualization. We used grid search over a range of learning rates η and regularization parameters λ to identify the optimal hyperparameter values. We experimented with a range of values:

Learning rates η of 0.00001, 0.0001 and 0.001 were used in the Adam optimizer, which is thoroughly described in Project 2 [5].

We evaluated the same values for the regularization parameters λ. A batch size of 256 was used to train the model efficiently. The model was trained for 10 epochs for each combination of η and λ.

We used heatmaps to study the impact of η and λ on testing and training accuracy. The accuracy is the percentage of correctly classified images in the data. The best performing model was selected based on the highest accuracy on the test set.

The best model, trained with the optimal hyperparameters, was tested on the test dataset. We used confusion matrix to compare the predictions to the true labels, showing

how well it classified the 10 classes in the dataset. We also included an early stopping callback. We used regularization techniques to reduce overfitting, by stopping the training once the performance on the validation set was getting worse for a specified number of epochs [8]. We sat the patience to 10 epochs. By stopping at this optimal point, we ensure that the model will have better generalization to unseen data.

The main methods we used to analyze the performance of our model were the accuracy and loss. Accuracy measures the proportion of correctly classified images, thereby giving a good indication of the predictive performance of the model. Loss, on the other hand, will tell us how well the model predictions match with the true labels. This will give an indication for the optimization process of the model. In this project, we used the cross-entropy function, which penalizes incorrect predictions more heavily.

Additionally, we used the classification report to study the model's performance in more detail. The classification report provides key metrics such as precision, recall, and F1-score for each class in the dataset. Precision measures the percentage of predicted labels that were correctly classified. Recall measures the percentage of correctly predicted images in that specific class. The F1-score, is the harmonic mean of the precision and the recall. A high F1-score indicates a good balance between prediction and recall, and the closer the score is to 1, the better the model's performance [9].

E. Implementation

We used the `TensorFlow Keras` [10] library to load the Fashion MNIST dataset.

Our CNN model was built using the `Sequential` functionality provided by TensorFlow Keras. By using predefined layers like `Dense`, `Conv2d` and `Maxpooling2D` we were able to adapt our network using different layers with different kernel sizes and regularization parameters, and various activation functions. For the last layer in our model, we used the `softmax` activation function, to asses the probability of classifying our 10 classes. For the CNN network, we also had to preprocess our input data, by normalizing the pixel values in our images, as well as adding a channel dimension, as `Keras` require a third dimension for the input data. When fitting our model, we used a validation split of 0.2, using 20% of the

testing data as validation.

The SVM model was implemented using the `SVC` functionality from `Scikit-learn` [11], allowing us to choose the kernel, adjust the regularization parameter C and the kernel coefficient (e.g. gamma for the RBF kernel), to optimize our classification model.

For the SVM funcionality, the input data needs to be represented as 1D vectors. Images with the original format of 28×28 were therefore flattened into vectors of length 784 using the `reshape` function. Scikit-learn's `StandardScaler` was used to ensure consistent feature scaling, as well as normalizing the pixel values by dividing by 255.

For the image labels, one-hot encoding was applied through the `to_categorical` function. This converts the class labels into a probabilistic representation.

The implementation process for our whole project is as follows:

1. **Data loading and preprocessing:** Load the dataset using keras, normalize. Add needed channel for CNN, flatten for SVM

2. **Model training:** Train our implemented methods for CNN and SVM on our training dataset.

3. **Parameter adjusting:** Use a range of CNN: eta, lambda, SVM: C (regularization parameter), gamma and kernel. Grid search to find optimal parameters.

4. **Evaluation:** Use the final, optimized model to analyze the performance on the test set.

IV. RESULTS AND DISCUSSION

In this section we present and discuss the results from our methods, starting with CNNs and moving on to SVMs.

A. CNNs

1. *Grid Search Results*

We used grid search to find the optimal learning rate η and regularization parameter λ for our CNN model. The results are visualized in heatmaps to study the impact of η and λ on training and testing accuracy, shown for training accuracy in Figure 4 and for test accuracy in Figure 5. The bright regions indicate higher accuracy.

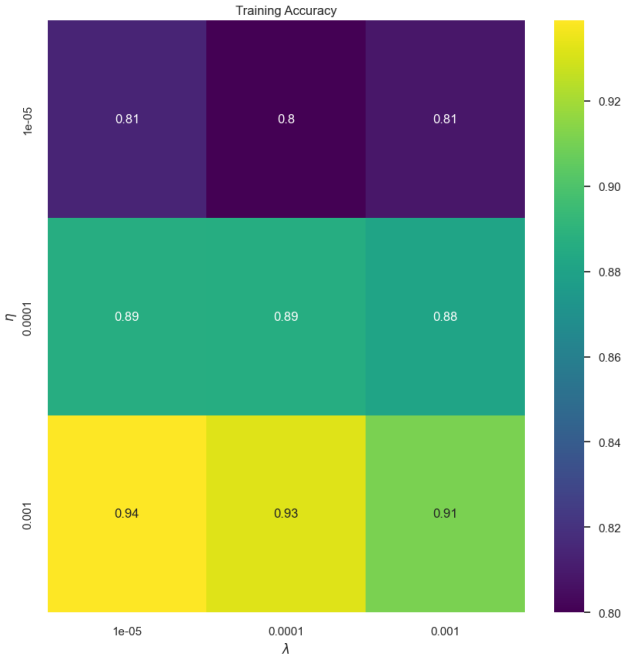

FIG. 4: Training accuracy heatmap for different combinations of learning rate (η) and regularization parameter (λ).

As shown in Figure 4, the model achieved the highest training accuracy at 94% with a learning rate of $\eta = 0.001$ and regularization parameter $\lambda = 1e-05$. The training data achieves a very high accuracy, but this is not representative for our model, as the model is evaluated on the same data that it was trained on. This means that the model already

knows the data, and therefore learns to fit it very well. The high accuracy can indicate that the model is overfitting, and therefore will not perform as good on unseen data. This grid search however, gives us a good indicator of the general trends in the model performance.

Moving further, we will therefore be looking at the grid search performed on the test data, which will give us a better indicator of how well the model actually performs on new, unseen data.

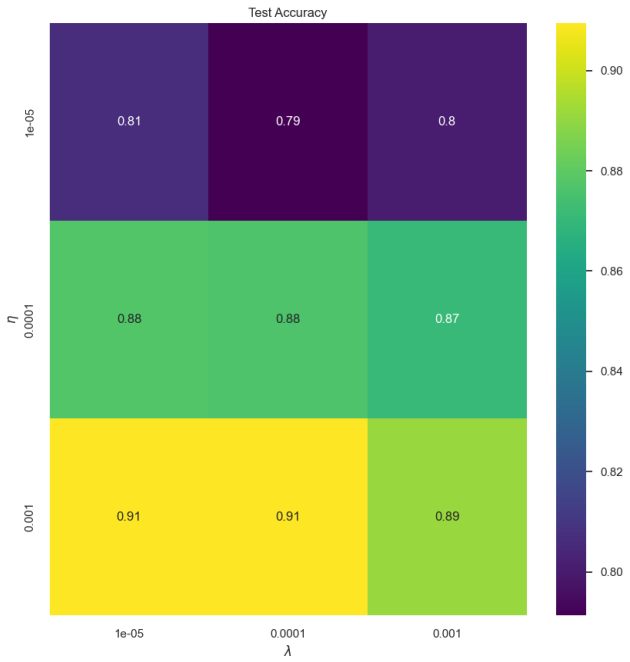

FIG. 5: Test accuracy heatmap for different combinations of learning rate (η) and regularization parameter (λ).

The test accuracy heatmap shown in Figure 5 highlights the performance of the model across the same range of hyperparameters. For smaller learning rates η the test accuracy is relatively low, around $80 - 81\%$, indicating that the model does not converge effectively.

As the learning rate increases to $\eta = 0.001$, the test accuracy improves significantly, achieving the highest test accuracy of 91%. Higher learning rates help the model converge faster, in this case yielding a higher accuracy.

The highest test accuracy of 91% was achieved with a learning rate of $\eta = 0.001$ and regularization parameter of either $\lambda = 1e - 05$ or $\lambda = 0.0001$. As λ increases to 0.001, the test accuracy drops slightly to 89%. Higher λ values reduce overfitting, but may also make

the model overly simple, reducing its test performance.

As shown in Figures 4 and 5, it is evident that the choice of hyperparameters significantly impacts both training and test accuracy. Moving forward, we will therefore be choosing a learning rate of 0.001 and a regularization parameter of 0.0001, which hopefully strikes a balance between overfitting and over simplifying, giving us a high accuracy.

2. *Confusion Matrix*

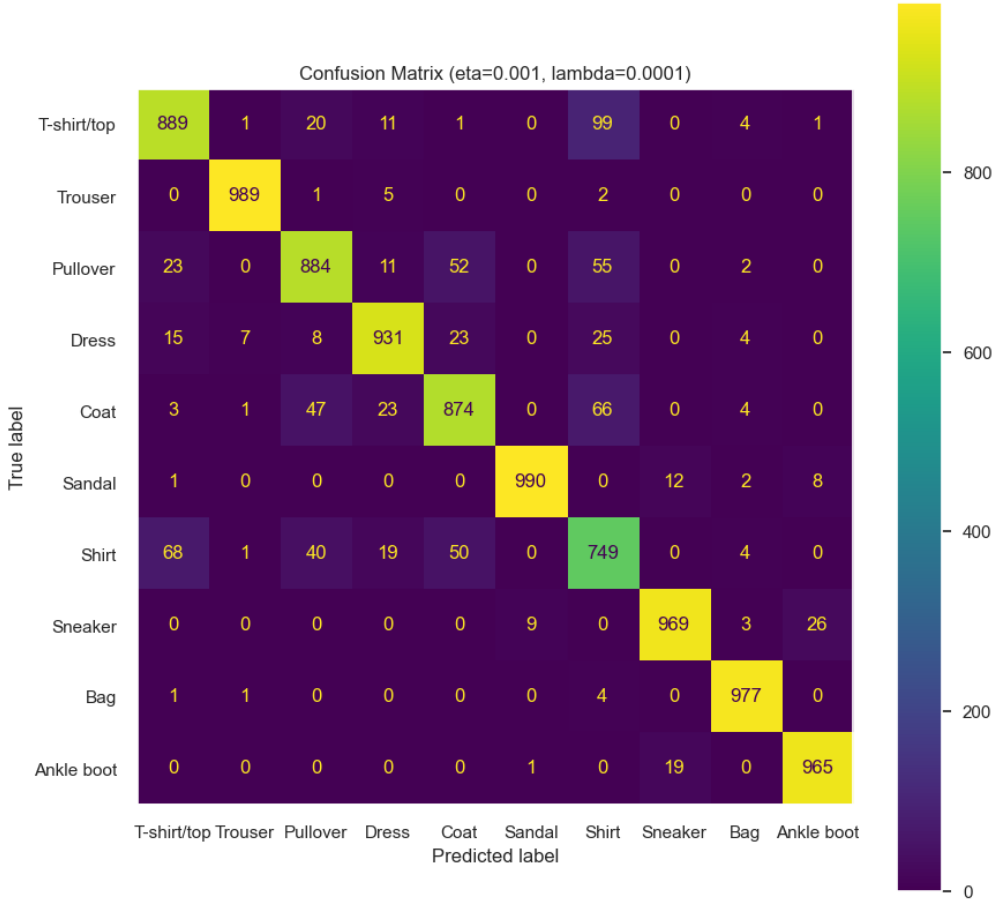

FIG. 6: CNN Confusion Matrix - $\eta = 0.001$, $\lambda = 0.0001$.

The confusion matrix in Figure 6 shows the model's performance across the 10 classes in the Fashion MNIST dataset. The rows represent the true labels, while the columns represent the predicted labels. The diagonal values indicate correct predictions, and off-diagonal values show misclassifications. The diagonal of the matrix indicates strong overall performance, with high precision scores for most classes. However, we got some misclassifications, mostly

between the "T-shirt/top", "Shirt", "Coat" and "Pullover" categories, which is likely due to similarities (similar features) between these categories.

3. Model Performance

We are further investigating the performance of our model using the accuracy and loss score for training and validation. Figure 7 visualizes the training and validation accuracy across epochs.

FIG. 7: Training and validation accuracy.

The training accuracy increases as the number of epochs grows, reaching an accuracy of over 96% for the highest amount of epochs. The accuracy is constantly increasing, and the growing gap between the training and validation accuracy increases, indicating overfitting on the training data.

The validation accuracy increases rapidly over the first 5 epochs, reaching an accuracy of around 91%. With increasing number of epochs, the accuracy fluctuates, eventually stabilizing at an accuracy of around 92% after approximately 17 epochs. This indicates that the model reaches its optimal performance at around 5-7 epochs, after which the accuracy will not improve more with the given hyperparameters.

Figure 8 shows the training and validation loss of the model across increasing epochs.

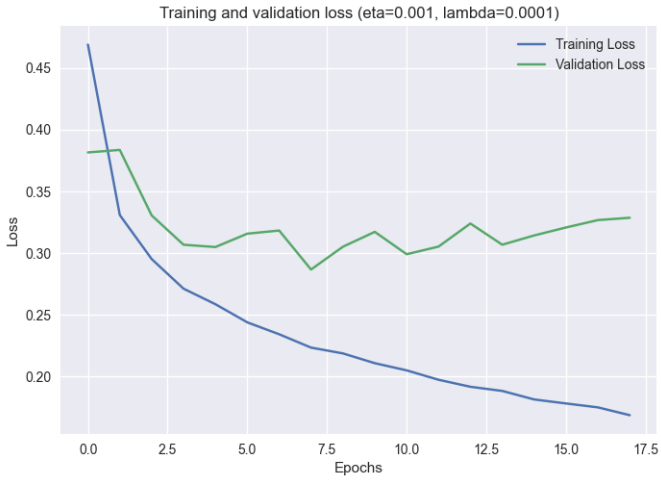

FIG. 8: Training and validation loss.

In Figure 8 we can observe the cross-entropy loss for the training and validation data. This shows us how well the model is minimizing the difference between the predicted and true values. We see the same general trend as for the training and validation accuracy. The training loss decreases steady to a minimum value of 0.2, while the validation loss decreases initially and reaches a minimum around 0.3 and then stabilizes across higher epochs.

Table I shows the classification report, which summarizes the performance of our CNN model for each class in the dataset.

Class	Precision	Recall	F1-score	Support
T-shirt/top	0.87	0.89	0.88	1000
Trouser	0.99	0.99	0.99	1000
Pullover	0.86	0.88	0.87	1000
Dress	0.92	0.93	0.92	1000
Coat	0.86	0.87	0.87	1000
Sandal	0.98	0.99	0.98	1000
Shirt	0.80	0.75	0.78	1000
Sneaker	0.96	0.97	0.97	1000
Bag	0.99	0.98	0.99	1000
Ankle boot	0.98	0.96	0.97	1000
Accuracy			0.92	10000
Macro avg	0.92	0.92	0.92	10000
Weighted avg	0.92	0.92	0.92	10000

TABLE I: Classification report CNN (eta=0.001, lambda=0.0001)

From Table I, we see that the model achieved an overall weighted F1-score of 0.92. The highest F1-score was obtained for "Bag" and "Trouser" with 99%, meaning the model could classify these classes very well. We also see a slight drop in performance for certain categories like "Shirt" (F1 = 0.78), because of similarities with other classes. These results align with the confusion matrix analysis, where similar categories had lower F1-scores.

B. Support Vector Machine

For the implementation of the Support Vector Machine classifier, we used the Radial Basis Function kernel, exploring different regularization parameters C. The different values of the regularization parameter that we used were $C = [1.0, 10, 100, 1000]$. We first explored how C alone influenced the performance of the model, while using the RBF-kernel, and $\gamma = 0.0001$.

Table II visualize the classification report and Figure 9 shows the confusion matrix and for the SVM model with $C = 1$, kernel = "rbf" and "gamma = 0.0001".

Class	Precision	Recall	F1-score	Support
T-shirt/top	0.78	0.83	0.81	1000
Trouser	0.99	0.95	0.97	1000
Pullover	0.75	0.74	0.74	1000
Dress	0.82	0.88	0.85	1000
Coat	0.73	0.78	0.75	1000
Sandal	0.93	0.91	0.92	1000
Shirt	0.64	0.53	0.58	1000
Sneaker	0.89	0.92	0.91	1000
Bag	0.95	0.96	0.95	1000
Ankle boot	0.93	0.93	0.93	1000
Accuracy			0.84	10000
Macro avg	0.84	0.84	0.84	10000
Weighted avg	0.84	0.84	0.84	10000

TABLE II: Classification Report Metrics (C=1.0, kernel='rbf', gamma=0.0001)

This setting has higher regularization and simpler decision boundaries, compared to a

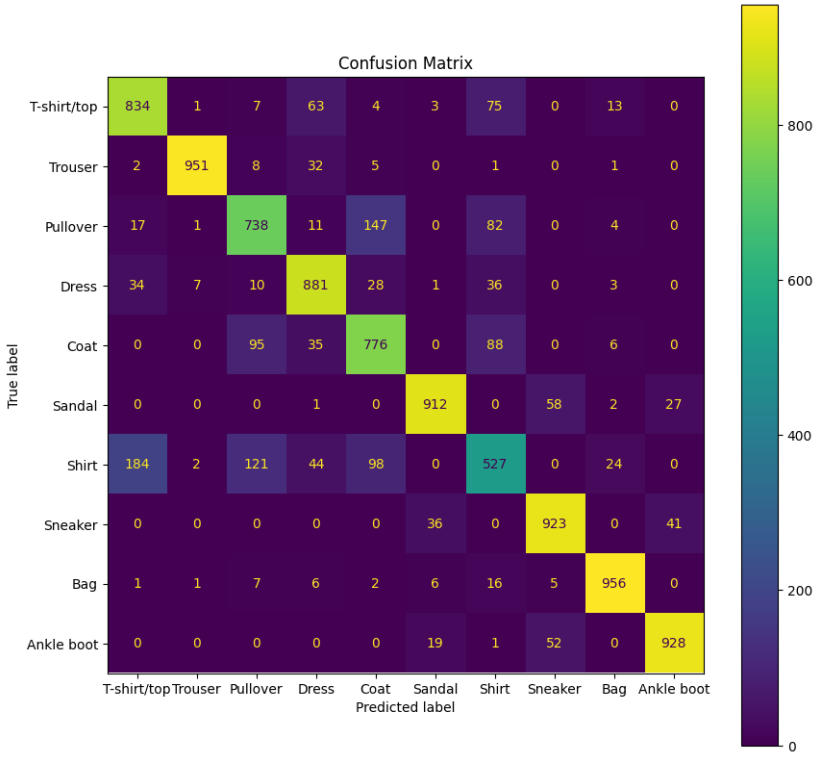

FIG. 9: Confusion matrix for SVC with C=1.0, kernel='rbf', gamma=0.0001

higher C-value. From Table II we see that the accuracy F1-score is 0.82. This model achieves moderate accuracy, but certain classes have significant misclassifications. "Shirt" has the most confusion, with only 527 correctly classified. Many shirts are misclassified as "T-shirt/top" (184) or "pullover" (121), likely due to visual similarities between the classes. The model performs very well for classes like "Trouser" with 951 correct predictions and little misclassification. The accuracy is decent, but the model meets some problems with differentiating visually similar classes due to the simpler decision boundaries enforced by higher regularization. For $C = 1$ the model can tolerate more errors, simplifying the model.

Table III visualize the classification report and Figure 10 shows the confusion matrix for SVM with $C = 100$, RBF kernel and $\gamma = 0.0001$.

This setting reduces the regularization, allowing the model to fit the data better with more complex decision boundaries. From Table III we read that the accuracy F1-score is now 0.88, which is higher in comparison with our other model with $C = 1$. We see that the class "Trouser" has excellent performance with 962 correctly classified instances. We also

Class	Precision	Recall	F1-score	Support
T-shirt/top	0.80	0.85	0.82	1000
Trouser	0.98	0.96	0.97	1000
Pullover	0.79	0.82	0.80	1000
Dress	0.86	0.88	0.87	1000
Coat	0.81	0.81	0.81	1000
Sandal	0.96	0.95	0.95	1000
Shirt	0.71	0.65	0.68	1000
Sneaker	0.93	0.95	0.94	1000
Bag	0.97	0.95	0.96	1000
Ankle boot	0.97	0.95	0.96	1000
Accuracy			0.88	10000
Macro avg	0.88	0.88	0.88	10000
Weighted avg	0.88	0.88	0.88	10000

TABLE III: Classification Report Metrics (C=100, kernel='rbf', gamma=0.0001

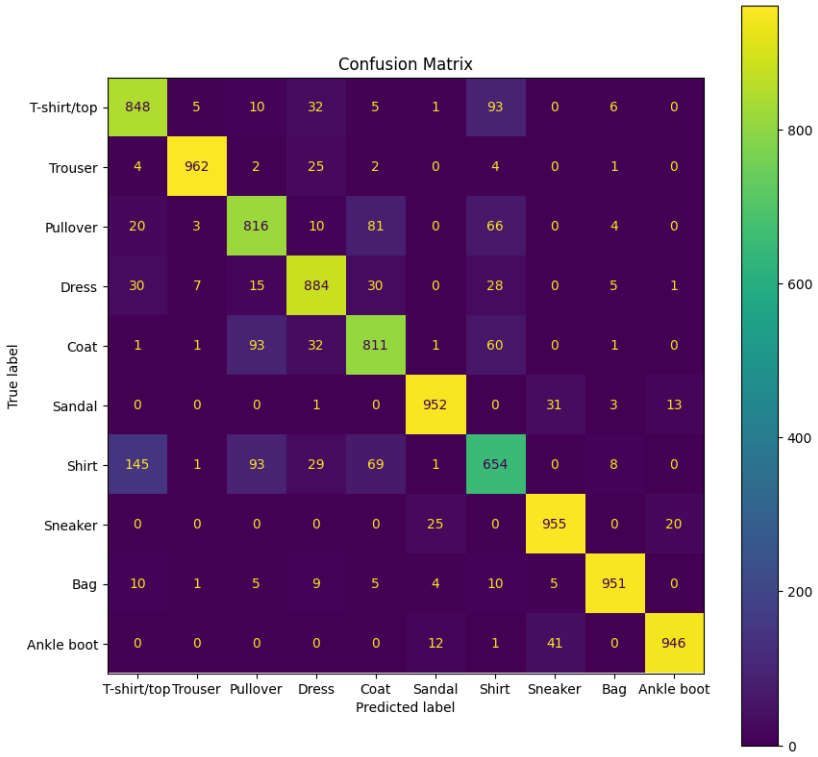

FIG. 10: SVC with C=100, kernel='rbf', gamma=0.0001

see some improvements. The number of correctly classified "shirt"-images improves, with 654 correctly classified instances compared to 527 in the $C = 1$ model. Misclassifications

are reduced, although they are still notable.

Increasing C from 1 to 100, improves performance, as higher value of C allows the model to better adapt to the dataset, creating more flexible decision boundaries and reducing misclassification. An additional Figure for $C = 10$ can be found in the Appendix. We also performed additional runs with some different values for γ, but they proved to not achieve much improvement. All test runs can be found on our GitHub repository [12].

Lastly, we compare the performance of CNNs and SVMs on the Fashion MNIST dataset, finding some key differences. As described earlier, CNNs are specifically designed for image data, which is reflected in the performance. CNNs achieve higher accuracy with an F1-score of 0.92, showing excellent performance in distinguishing classes like "Bag" and "Trouser" with an F1-score of 0.99 for these classes.

In contrast, SVMs with RBF kernel give us a simpler approach to classification. We studied different values for the regularization parameter C, observing an F1-score of 0.88 with $C = 100$. The confusion matrix for SVMs shows more misclassifications in comparison with CNNs, particularly for the similar looking categories. The performance of the SVM model could however be optimized further, by tuning hyperparameters.

Even though both of the models are classifying certain categories with distinct features well, both CNNs and SVMs are struggling with classifying visually similar classes like "Shirt" and "T-shirt/top". Overall, we have found that the CNN performs better than SVMs in terms of accuracy.

V. CONCLUSION

In this project, we studied the performance of CNNs and SVMs on the Fashion MNIST dataset, a benchmark designed for evaluating classification algorithms on challenging image data. Our findings show that CNNs are very effective for image classification, achieving an F1-score of 0.92, with high performance in distinguishing most classes, especially "Bag" and "Trouser." This superior performance is due to CNNs ability to learn hierarchical features through multiple specialized layers. However, CNNs struggled to differentiate visually similar categories such as "Shirt" and "T-shirt/top," indicating room for improvement in handling similar features for class separation.

SVMs, using the Radial Basis Function (RBF) kernel, achieved an F1-score of 0.88, showing robustness in handling high dimensional data with moderate performance. Despite achieving acceptable results, SVMs had more difficulty separating visually similar classes compared to CNNs. Hyperparameter tuning, particularly of the regularization parameter C, played an important role in improving the performance, highlighting the importance of optimization.

Future work should focus on improving classification performance for challenging classes that struggle due to visual similarities with other categories. Strategies can include using deeper CNN architectures with more layers to extract more complex features. For SVMs, using alternative kernels, further tuning hyperparameters, and using scaling methods could improve the performance. This project highlights both the strengths and limitations of CNNs and SVMs, and opens for further study of machine learning techniques in image classification.

VI. APPENDIX

A. Additional Figures

Class	Precision	Recall	F1-score	Support
T-short/top	0.80	0.84	0.82	1000
Trouser	0.98	0.96	0.97	1000
Pullover	0.78	0.79	0.78	1000
Dress	0.84	0.88	0.86	1000
Coat	0.77	0.80	0.78	1000
Sandal	0.96	0.94	0.95	1000
Shirt	0.70	0.60	0.65	1000
Sneaker	0.92	0.95	0.93	1000
Bag	0.97	0.97	0.97	1000
Ankle boot	0.96	0.94	0.95	1000
Accuracy			0.87	10000
Macro avg	0.87	0.87	0.87	10000
Weighted avg	0.87	0.87	0.87	10000

TABLE IV: Classification Report (C=10, kernel='rbf', gamma=0.0001)

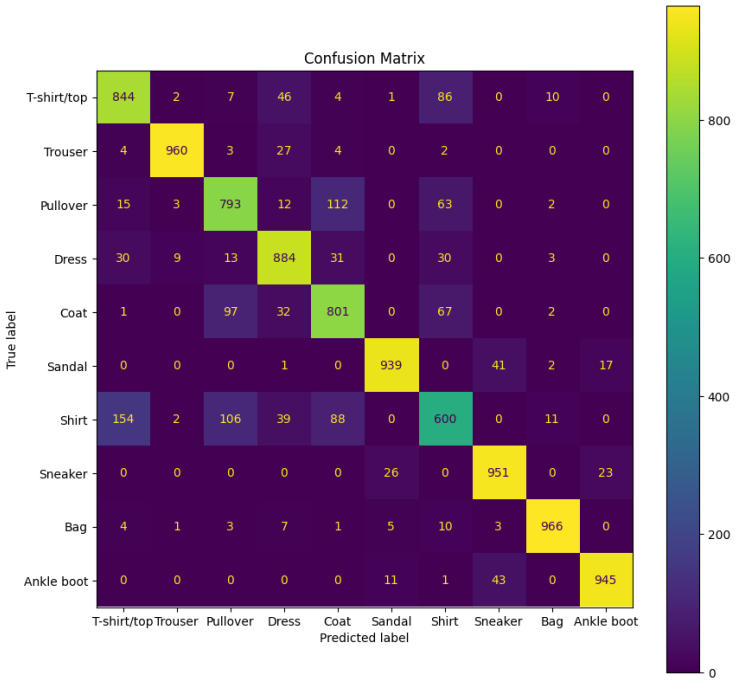

FIG. 11: SVC with C=10, kernel='rbf', gamma = 0.0001

B. GitHub

All code can be found at Github: https://github.com/trudehalv/FYS_STK4155_project_3.

VII. ACKNOWLEDGEMENTS

Throughout this project, we got a lot of inspiration for our codes from Professor Morten Hjorth-Jensen's lecture notes [2]. Thank you!

Chat GPT provided assistance to clarify some coding issues and to improve our language. This tool was used to support our independent work.

REFERENCES

[1] Zalando Research. Fashion mnist dataset. https://www.kaggle.com/datasets/zalando-research/fashionmnist, 2024. Accessed December, 2024.

[2] Morten Hjorth-Jensen. Machine learning lecture notes. https://github.com/CompPhysics/MachineLearning/tree/master/doc/pub, 2023. GitHub repository.

[3] IBM. Convolutional neural networks. https://www.ibm.com/topics/convolutional-neural-networks, 2024. Web article.

[4] Milliams.com. Convolutional neural networks. https://milliams.com/courses/neural_networks/Convolutional%20neural%20networks.html, 2024. Accessed December 11, 2024.

[5] Tiril Trondsen, Trude Halvorsen, Lars-Martin Gihle, and Jonas Båtnes. Fys-stk 4155 project 2. https://github.com/MrSBR/FYS-STK4155_project2, 2024.

[6] Vaibhav Rastogi. Fully connected layer. https://medium.com/@vaibhav1403/fully-connected-layer-f13275337c7c, 2023. Accessed: 2024-12-12.

[7] GeeksforGeeks. Image classification using support vector machine (svm) in python. https://www.geeksforgeeks.org/image-classification-using-support-vector-machine-svm-in-python/, 2024. Accessed: 2024-12-12.

[8] GeeksforGeeks. Regularization by early stopping. https://www.geeksforgeeks.org/regularization-by-early-stopping/, 2023. Accessed: 2024-12-12.

[9] Zach Bobbitt. How to interpret the classification report in scikit-learn. https://www.statology.org/sklearn-classification-report/, 2022. Accessed: 2024-12-12.

[10] TensorFlow Datasets. Fashion mnist dataset. https://www.tensorflow.org/datasets/catalog/fashion_mnist, 2024. Accessed: 2024-12-12.

[11] Scikit-Learn. Svc. https://scikit-learn.org/1.5/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC, 2024. Accessed: 2024-12-12.

[12] Trude Halvorsen Tiril Trondsen. Fys-stk4155 - project 3. https://github.com/trudehalv/FYS_STK4155_project_3, 2024. Accessed December 13, 2024.