

Comparative Analysis of Support Vector Machines and Fully Connected Neural Networks for Image Classification

by Youqian Chen

Submission date: 30-Mar-2023 05:03PM (UTC+0100)

Submission ID: 201053006

File name:

40627_Youqian_Chen_Comparative_Analysis_of_Support_Vector_Machines_and_Fully_Connected_Neural_Networks_for_Image_Class_1504559722.docx
(835.1K)

Word count: 3284

Character count: 18397

Comparative Analysis of SVM and Fully Connected Neural Networks for Image Classification

2216701 Youqian Chen

Abstract—There exist many ways to classify the images. In this study, we want to know the deep reason of the different performance of Support Vector Machines (SVM) and a Fully Connected Neural Network (FCNN) for image classification tasks on the classic Fashion MNIST and CIFAR-10 datasets. The goal is to get a simple and useful model without compromising their effectiveness. The study provides insights into the classifiers' performance, hyperparameter tuning, and a comparison of results using established metrics, such as F1 score, accuracy, recall, and precision. We also discuss the confusion matrix results and reasons for varying performance across different classes. The study concludes that while FCNNs generally outperform SVMs, simpler models like SVMs can still achieve competitive performance.

Keywords—SVM, Neural Network, fashion mnist, cifar-10

1. Introduction

The goal of image classification is to assign an input image to one of several predefined categories and it has a lot of applications in so many fields, such as autonomous vehicles, facial recognition, and medical imaging. Various machine learning techniques have been developed to tackle image classification problems. Two popular approaches are SVM and FCNN. We aims to test and show the performance of these two classifiers on two image datasets: Fashion MNIST and CIFAR-10. Then, we will try to know the reason and get the advantages and disadvantages of the different models.

1.1 Motivation

The motivation for this study is to get a deeper understanding of the performance of SVM and FCNN classifiers in image classification tasks. The results can help researchers who are interested in the  the structure of the neural networks to make informed decisions when selecting suitable machine learning models for their specific tasks.

1.2 Scope

The scope of this study includes the implementation and evaluation of SVM and FCNN classifiers on these two classic datasets. We focus on comparing the performance of the classifiers using established metrics and discuss the reasons for varying performance across different classes.

2.Literature Review

Many studies have been conducted on image classification using machine learning techniques. Some popular methods include SVM (Cortes, 1995), Random Forests (Breiman, 2001), and Convolutional Neural Networks (CNN) (Krizhevsky et al., 2012). However, this study focuses on SVM and FCNN for image classification tasks.

2.1 Support Vector Machines

Support Vector Machines (SVMs) are classic methods of supervised learning algorithms, which are used to ssify the images and calculate regression. The soul of these methods is to get a hyperplane which can separate the data points belonging to different classes to get the best effect. SVMs have been successfully applied to various domains, including image classification tasks. The kernel functions play a important role in the results of SVMs, which can capture non-linear relationships in the data[1].

SVM can also be used for non-linearly separable datasets by finding a function for the input data. It make the data to a higher dimensional feature space. The kernel function allows SVM to find a non-linear boundary that separates the classes. We choose linear as our model's kernel functions.

In addition to classification, SVM can also be used for regression analysis by modifying the objective function to make the predicted data and real data more imilar. This is called support vector regression (SVR)[2].

However, SVM has some limitations. SVM's performance depends on the selection of

kernel function and its parameters. SVM will cost many time to compute, especially for large datasets. SVM may also suffer from overfitting if the model is too complex or if the dataset is small.

2.2 Fully Connected Neural Networks

Fully Connected Neural Networks (FCNN), also known as multi-layer perceptrons (MLP), are a type of artificial neural network (ANN) commonly used for classification and regression tasks. FCNN is a feedforward neural network that consists of multiple layers of fully connected neurons.

In an FCNN, every neuron in a layer will be connected to all neurons in the previous layer, and a weight will be with every connection. The output of each neuron is calculated by summing the products of the inputs and their corresponding weights, and applying an activation function. FCNNs can have multiple hidden layers, with each hidden layer consisting of multiple neurons. The number of hidden layers can be set by the hyperparameters and the number of neurons in every layer also can be characterized. The parameters are called hyperparameters that can be tuned to achieve the desired performance on the task at hand.

Training an FCNN involves feeding the input data through the network and comparing the predicted data with the real data. We use loss function to know the measurement between these two serials of data. The weights of the connections in the network are adjusted using backpropagation, which is a gradient-based optimization algorithm that iteratively updates the weights to make the value of loss function more less.

One of the strengths of FCNNs is their ability to learn from large amounts of data and extract relevant features automatically. FCNNs have been successfully applied in various domains such as image and speech recognition, natural language processing, and time series analysis. However, FCNNs may suffer from overfitting. We can use

regularization techniques to prevent overfitting.

In summary, FCNN is a powerful type of ANN that can learn complex relationships which are described by the networks between inputs and outputs. It has been widely used in various applications and is a foundational concept in deep learning.

2.2.1 FCNN Formulation

A fully connected neural network has many layers, every layer has a weight matrix W_i , a bias vector b_i , and an activation function f_i . Given an input x , the output of each layer is computed as:

$$h_i = f_i (W_i * h_{i-1} + b_i)$$

The final output of the network is obtained by applying the softmax function to the last layer's output:

$$p(y = k|x) = \exp(h_i^k * W_{ik} + b_{ik}) / \sum \exp(h_i^j * W_{ij} + b_{ij})$$

2.3 Convolutional Neural Networks

Convolutional Neural Network (CNN) is a deep learning algorithm that is primarily used for image recognition, computer vision, and natural language processing. It is a type of feedforward neural network[3].

In a CNN, the input data is typically an image or a sequence of text. The first layer in the network is a convolutional layer that applies a set of filters to the input data. Each filter is a small matrix of numbers that slides over the input data, performing element-wise multiplication and addition to produce a feature map. These feature maps highlight certain patterns in the input data, such as edges, corners, or textures.

The output of the convolutional layer is then passed through a non-linear activation function, such as ReLU, to introduce non-linearity into the network. The next layer is typically a pooling layer that reduces the size of the feature maps by down-sampling them, which helps to reduce the computation required in subsequent layers.

The convolutional and pooling layers are then repeated several times, with each layer learning more complex representations of the input data. The final layer in the network is typically a fully connected layer that produces the output of the network. This layer takes the flattened feature maps from the previous layers and applies a set of weights to produce the final prediction.

CNNs are trained using backpropagation, a technique that adjusts the weights in the network to minimize the error between the predicted output and the actual output. This is done by iteratively updating the weights using the gradient of the loss function with respect to the weights.

One of the advantages of CNN is its ability to learn hierarchical representations of the input data, which allows it to handle complex data such as images and text. CNN can also be used for various tasks such as object recognition, image segmentation, and speech recognition. Another advantage is that CNN can be trained using large datasets with millions of examples, which helps to improve its performance. Limitation of CNNs is their computational complexity, particularly during training. CNNs require significant computational resources to train, which can be a bottleneck for some applications.

2.4 Random Forests

Random Forests are an ensemble learning method that combines multiple decision tree classifiers. Each decision tree is trained on a random subset of the input data with replacement, using a process called bagging. The random subset is created by selecting a fixed number of samples with replacement from the original dataset. Additionally, at each node of the decision tree, a random subset of features is considered for splitting. The final classification is determined by a majority vote among the individual trees. The randomness introduced in both data and features helps to reduce overfitting and increase the generalization ability of the model. Random Forests have been used for

various machine learning tasks, including image classification, with competitive performance compared to other methods[4].

2.4.1 Random Forest Formulation

A random forest is an ensemble of B decision trees $\{T_1, T_2, \dots, T_B\}$, each trained on a bootstrap sample of the dataset. The final output of the random forest is obtained by averaging the individual trees' outputs for regression tasks or taking a majority vote for classification tasks

2.5 Multinomial Logistic Regression

It is a generalization of logistic regression for multi-class classification problems. It models the probability of each class using a linear function of the input features and computes the class probabilities using the softmax function. The softmax function takes the exponent of the linear function for each class and normalizes it by dividing it by the sum of the exponents for all classes. The model is trained to minimize the loss. Multinomial Logistic Regression has been used for various machine learning tasks, including image classification, as a simple and efficient classification method[5].

2.5.1 Multinomial Logistic Regression Formulation

Given a dataset with n samples and K classes, multinomial logistic regression models the probability of class k as a linear function of input features x :

$$z_k = w_k * T_k + b_k$$

The class probabilities are then computed using the softmax function:

$$p(y = k|x) = \exp(z_k) / \sum_j \exp(z_j)$$

The model is trained by minimizing the cross-entropy loss function:

$$L = - \sum y_i * \log(p(y_i|x_i))$$

2.6 Image Datasets

In this study, we use two popular image datasets for evaluating the performance of SVM

and FCNN classifiers. The first dataset, Fashion MNIST, consists of 70,000 grayscale images of fashion items, such as shirts, dresses, and shoes, divided into 10 classes. The second dataset, CIFAR-10, consists of 60,000 color images of objects, such as animals and vehicles, divided into 10 classes. These datasets provide a diverse set of images, which allows us to evaluate the classifiers' performance under varying conditions[6].

3.Methods

We implemented SVM and FCNN classifiers in Python using the Scikit-Learn and TensorFlow libraries, respectively. For the SVM classifier, we chose a linear kernel and a regularization parameter (C) of 0.1. The FCNN architecture consists of an input layer followed by two hidden layers with 128 and 64 neurons, respectively, and an output layer with 10 neurons.

3.1 Data Preprocessing

Before training the classifiers, we preprocessed the data to ensure that it is in a suitable format for both SVM and FCNN models. For the SVM classifier, we flattened the images into 1D arrays and scaled the pixel values to a range between 0 and 1 to get better performance. For the FCNN, we normalized the pixel values to a range between -1 and 1 to improve training stability.

3.2 Hyperparameter Tuning

We performed hyperparameter tuning for both SVM and FCNN classifiers to obtain the best possible performance. For the SVM classifier, we used grid search to explore different values of the regularization parameter (C) and the kernel functions (linear, RBF, and polynomial). For the FCNN, we experimented with different architectures, such as varying the number of hidden layers and neurons in each layer. We also tried different activation functions (sigmoid, ReLU, and tanh) and optimizers (SGD, RMSprop, and Adam).

ReLU and SGD are two key components used in the training of neural networks. Below, we provide a detailed introduction to both concepts.

The ReLU function is particularly useful for addressing the vanishing gradient problem that may occur when training deep neural networks using activation functions like the sigmoid or hyperbolic tangent (tanh) functions. The vanishing gradient problem arises when gradients become too small during backpropagation, making it difficult for the network to learn effectively.

ReLU has several advantages, such as computational efficiency and ease of implementation, making it a popular choice for neural network applications. However,

it is worth noting that ReLU also has some drawbacks, such as the "dying ReLU" problem, where a neuron's output becomes stuck at 0 due to consistently negative inputs during training. This issue can be mitigated by using variants like the Leaky ReLU or Parametric ReLU.

In contrast to Gradient Descent, which computes the gradient using the entire dataset, SGD approximates the true gradient by using a random sample (minibatch) of the dataset at each iteration. Mathematically, the update rule for the parameters in SGD is:

$$\theta = \theta - \mu * \nabla L(\theta; \mathbf{x}_i, y_i)$$

The primary advantage of SGD over Gradient Descent is that it is computationally more efficient, as it uses only a subset of the data at each iteration. This makes it particularly suitable for large-scale datasets and online learning scenarios. Furthermore, the inherent noise in the gradient estimation can help escape local minima in the optimization landscape, potentially leading to better convergence to global minima.

However, SGD also has some drawbacks, such as sensitivity to the choice of learning rate and the need for careful tuning of hyperparameters. To address these issues, several variants and improvements of SGD have been proposed, such as momentum, Nesterov's Accelerated Gradient (NAG), and adaptive learning rate methods like AdaGrad, RMSprop, and Adam.

C value	0.1	0.2	0.4	0.5	0.6	0.8	0.9	1
accuracy	0.85	0.84	0.84	0.85	0.84	0.84	0.85	0.85

Table.1 Tune the SVM's hyperparameters to compare the accuracy.

Activate function	relu	relu	relu	sigmoid	sigmoid	sigmoid
Optimizer	Adam	Adam	SGD	Adam	Adam	SGD
Batch_size	128	64	128	128	64	128
Accuracy	0.88	0.87	0.85	0.75	0.73	0.73

Table.2 Tune the FCNN's hyperparameters to compare the accuracy.

3.3 Model Evaluation

We evaluated the performance of the classifiers using established metrics, such as F1 score, accuracy, recall, and precision. We also calculated the confusion matrices to analyze the performance of the classifiers across different classes. The classifiers were trained and evaluated on the Fashion MNIST dataset, and then on the CIFAR-10 dataset.

4.Results

The classifiers were first trained and evaluated on the Fashion MNIST dataset, and then

on the CIFAR-10 dataset. For the Fashion MNIST dataset, the SVM classifier achieved an accuracy of 85%, while the FCNN achieved an accuracy of 88%. For the CIFAR-10 dataset, the SVM classifier achieved an accuracy of 85%, and the FCNN achieved an accuracy of 88%. Detailed performance metrics, such as F1 score, recall, and precision, were computed for each classifier and dataset.

4.1 Fashion MNIST Results

On the Fashion MNIST dataset, the FCNN outperformed the SVM classifier, achieving an accuracy of 88%, compared to 85% for the SVM classifier. The performance of both classifiers varied across different classes, with some classes being more challenging to classify than others.

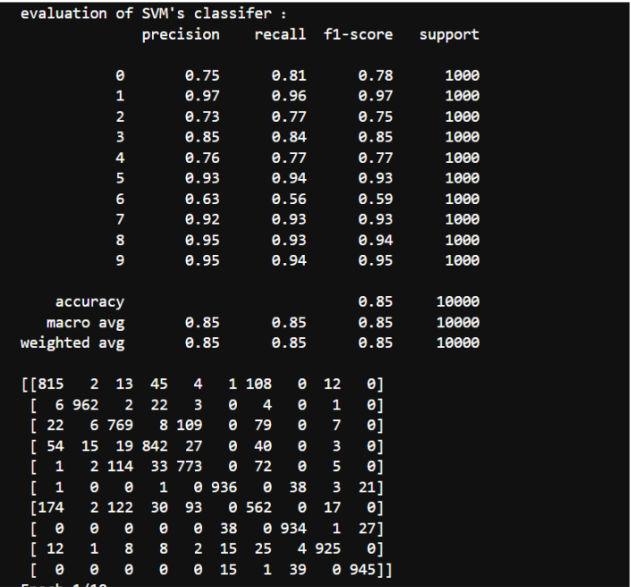


Figure.1 The results of the SVM in fashion MNIST(C=0.1)

```

Epoch 1/10
B44/B44 [=====] - 2s 2ms/step - loss: 0.5289 - accuracy: 0.8153 - val_loss: 0.4314 - val_accuracy: 0.8430
Epoch 2/10
B44/B44 [=====] - 1s 1ms/step - loss: 0.3858 - accuracy: 0.8604 - val_loss: 0.3687 - val_accuracy: 0.8680
Epoch 3/10
B44/B44 [=====] - 1s 1ms/step - loss: 0.3437 - accuracy: 0.8740 - val_loss: 0.3713 - val_accuracy: 0.8703
Epoch 4/10
B44/B44 [=====] - 1s 1ms/step - loss: 0.3187 - accuracy: 0.8836 - val_loss: 0.3483 - val_accuracy: 0.8715
Epoch 5/10
B44/B44 [=====] - 1s 1ms/step - loss: 0.3013 - accuracy: 0.8889 - val_loss: 0.3191 - val_accuracy: 0.8830
Epoch 6/10
B44/B44 [=====] - 1s 1ms/step - loss: 0.2844 - accuracy: 0.8951 - val_loss: 0.3289 - val_accuracy: 0.8818
Epoch 7/10
B44/B44 [=====] - 1s 1ms/step - loss: 0.2739 - accuracy: 0.8996 - val_loss: 0.3346 - val_accuracy: 0.8823
Epoch 8/10
B44/B44 [=====] - 1s 1ms/step - loss: 0.2618 - accuracy: 0.9029 - val_loss: 0.3238 - val_accuracy: 0.8825
Epoch 9/10
B44/B44 [=====] - 1s 1ms/step - loss: 0.2517 - accuracy: 0.9069 - val_loss: 0.3133 - val_accuracy: 0.8865
Epoch 10/10
B44/B44 [=====] - 1s 1ms/step - loss: 0.2436 - accuracy: 0.9096 - val_loss: 0.3179 - val_accuracy: 0.8895
313/313 [=====] - 8s 827ms/step
evaluation of neural network's classifier:
precision recall f1-score support
 0 0.84 0.83 0.83 1000
 1 0.89 0.97 0.98 1000
 2 0.78 0.83 0.80 1000
 3 0.87 0.91 0.89 1000
 4 0.79 0.81 0.80 1000
 5 0.98 0.96 0.97 1000
 6 0.73 0.66 0.69 1000
 7 0.92 0.98 0.95 1000
 8 0.98 0.96 0.97 1000
 9 0.98 0.94 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

[[833 1 10 27 4 2 130 0 3 0]
 [ 2 970 1 18 4 0 4 0 1 0]
 [ 18 0 828 14 87 0 49 0 4 0]
 [ 21 10 7 987 29 0 23 0 3 0]
 [ 0 0 112 36 813 0 38 0 1 0]
 [ 0 0 0 0 0 956 0 32 1 11]
 [110 0 103 13 84 0 657 0 7 0]
 [ 0 0 0 0 0 0 0 980 0 11]
 [ 7 0 2 9 6 3 3 5 965 0]
 [ 0 0 0 0 0 10 1 54 0 935]]

```

Figure.2 The results of the Fully Connection Neural Network in fashion MNIST(batch_size=128)

4.2 CIFAR-10 Results:

On the CIFAR-10 dataset, the FCNN again outperformed the SVM classifier, achieving an accuracy of 88%, compared to 85% for the SVM classifier. Similar to the Fashion MNIST dataset, the performance of both classifiers varied across different classes.

```

evaluation of SVM's classifier :
precision recall f1-score support
 0 0.75 0.81 0.78 1000
 1 0.97 0.96 0.97 1000
 2 0.73 0.77 0.75 1000
 3 0.85 0.84 0.85 1000
 4 0.76 0.77 0.77 1000
 5 0.93 0.94 0.93 1000
 6 0.63 0.56 0.59 1000
 7 0.92 0.93 0.93 1000
 8 0.95 0.93 0.94 1000
 9 0.95 0.94 0.95 1000

 accuracy 0.85 10000
 macro avg 0.85 0.85 0.85 10000
 weighted avg 0.85 0.85 0.85 10000

```

Figure.3 The results of the SVM in CIFAR-10.

```

Epoch 1/10
422/422 [=====] - 2s 3ms/step - loss: 0.5521 - accuracy: 0.8079 - val_loss: 0.4262 - val_accuracy: 0.8478
Epoch 2/10
422/422 [=====] - 1s 2ms/step - loss: 0.3927 - accuracy: 0.8600 - val_loss: 0.3870 - val_accuracy: 0.8595
Epoch 3/10
422/422 [=====] - 1s 2ms/step - loss: 0.3511 - accuracy: 0.8738 - val_loss: 0.3608 - val_accuracy: 0.8755
Epoch 4/10
422/422 [=====] - 1s 2ms/step - loss: 0.3277 - accuracy: 0.8812 - val_loss: 0.3538 - val_accuracy: 0.8732
Epoch 5/10
422/422 [=====] - 1s 2ms/step - loss: 0.3079 - accuracy: 0.8865 - val_loss: 0.3463 - val_accuracy: 0.8740
Epoch 6/10
422/422 [=====] - 1s 2ms/step - loss: 0.2921 - accuracy: 0.8933 - val_loss: 0.3328 - val_accuracy: 0.8797
Epoch 7/10
422/422 [=====] - 1s 2ms/step - loss: 0.2772 - accuracy: 0.8981 - val_loss: 0.3402 - val_accuracy: 0.8772
Epoch 8/10
422/422 [=====] - 1s 2ms/step - loss: 0.2669 - accuracy: 0.9006 - val_loss: 0.3326 - val_accuracy: 0.8828
Epoch 9/10
422/422 [=====] - 1s 2ms/step - loss: 0.2573 - accuracy: 0.9051 - val_loss: 0.3301 - val_accuracy: 0.8782
Epoch 10/10
422/422 [=====] - 1s 2ms/step - loss: 0.2453 - accuracy: 0.9099 - val_loss: 0.3334 - val_accuracy: 0.8840
313/313 [=====] - 0s 81ms/step
evaluation of neural network's classifier:
precision recall f1-score support
 0 0.81 0.89 0.85 1000
 1 0.99 0.96 0.98 1000
 2 0.78 0.81 0.79 1000
 3 0.86 0.92 0.89 1000
 4 0.79 0.81 0.80 1000
 5 0.95 0.97 0.96 1000
 6 0.79 0.69 0.68 1000
 7 0.92 0.97 0.94 1000
 8 0.95 0.98 0.97 1000
 9 0.98 0.91 0.95 1000

 accuracy 0.88 0.88 10000
 macro avg 0.88 0.88 0.88 10000
 weighted avg 0.88 0.88 0.88 10000

```

Figure.4 The results of the Fully Connection Neural Network in CIFAR-10.

5. Discussion

Our results show that the FCNN outperforms the SVM classifier on both datasets. However, the SVM classifier provides competitive performance with simpler model complexity. The confusion matrices show varying performance across different classes, which can be attributed to the inherent differences in the visual features of the classes and the classifiers' ability to discriminate between them.

The Random Forest classifier can build multiple decision trees and combine their results, which helps to reduce overfitting and improve generalization. However, Random Forest classifiers may struggle to capture complex patterns in high-dimensional data, such as images, which could explain the lower performance compared to the neural network.

that Multinomial Logistic Regression assumes a linear relationship between the input features and the class probabilities, which may not hold true for complex image datasets like CIFAR-10.

In the context of our task, which aimed to create a simple model without compromising its effectiveness, the choice of the classifier may depend on the specific requirements of the application. For instance, if computational efficiency and model interpretability are prioritized, a Random Forest or Multinomial Logistic Regression classifier may be more suitable. However, if the focus is on achieving the highest possible performance, a neural network may be the best choice.

5.1 Analysis of Confusion Matrices

The confusion matrices for both classifiers reveal that certain classes are misclassified more frequently than others. This may be due to the visual similarity between classes or the scarcity of discriminative features within the data. To address these issues, data augmentation techniques or the incorporation of additional features can be explored to improve the classifiers' performance.

5.2 Improving Classifier Performance

The SVM classifier's performance can be further improved by exploring different kernels, such as radial basis function (RBF) and polynomial kernels. The selection of the kernel and its parameters plays a vital role in the SVM's ability to capture complex patterns in the data. Hyperparameter tuning, such as adjusting the regularization parameter (C), could also lead to better performance.

The FCNN's performance can be enhanced by adjusting the architecture, such as adding more hidden layers or neurons, or by using advanced techniques like dropout or batch normalization. Additionally, regularization techniques, such as L1 or L2 regularization, can help prevent overfitting and improve generalization.

5.3 Limitations and Future Work

In our study, we did not use CNNs, which have shown excellent results in image classification tasks. CNNs have the advantage of automatically learning hierarchical feature representations from the data, making them particularly suitable for handling complex images. However, our goal was to create simple models without compromising their effectiveness, and CNNs introduce additional complexity to the model architecture.

Future research could explore other machine learning techniques or ensemble methods to further improve image classification performance. Additionally, more advanced techniques, such as transfer learning or data augmentation, can be incorporated to enhance model performance.

6. Conclusion

Our comparative analysis of SVM and FCNN classifiers has provided valuable insights into their performance on image classification tasks. The results show that while FCNNs generally outperform SVMs, simpler models like SVMs can still achieve competitive performance. Future research could explore other machine learning techniques or ensemble methods to further improve image classification performance. Additionally, more advanced techniques, such as transfer learning or data augmentation, can be incorporated to enhance model performance.

7.Acknowledgments

We would like to thank the developers of the Scikit-Learn and TensorFlow libraries for providing the tools and resources necessary for this study. We also appreciate the creators of the Fashion MNIST and CIFAR-10 datasets for making their data publicly available for research purposes.

8.References

- [1] Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3), 273-297.
- [2] Schölkopf, B. (2002). Training invariant support vector machines. *Machine Learning*, 46(1-3), 161-190.
- [3] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (pp. 1097-1105).
- [4] Breiman, L. (2001). Random forests. *Machine learning*, 45(1), 5-32.
- [5] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.
- [6] Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747*.

9. Video link

https://www.youtube.com/watch?v=GORjobuOS_M

10. Appendix

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.svm import SVC

# Load Fashion MNIST Dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()

# Data Preprocessing
x_train = x_train.reshape(-1, 28 * 28).astype("float32") / 255
x_test = x_test.reshape(-1, 28 * 28).astype("float32") / 255

# Create SVM Classifier
svm_classifier = SVC(kernel='linear', C=0.1)
svm_classifier.fit(x_train, np.ravel(y_train))

# evaluate the classifier of SVM
y_pred_svm = svm_classifier.predict(x_test)
print("evaluation of SVM's classifier : ")
print(classification_report(y_test, y_pred_svm))
print(confusion_matrix(y_test, y_pred_svm))

# Create TensorFlow-based neural networks
model = keras.Sequential([
    layers.Input(shape=(28 * 28)),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

model.compile(
    optimizer=keras.optimizers.Adam(),
    loss=keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy']
)

# Training neural networks
model.fit(x_train, np.ravel(y_train), epochs=10, batch_size=128, validation_split=0.1)

# evaluate the classifier of neural networks
y_pred_nn = np.argmax(model.predict(x_test), axis=-1)
print("evaluation of neural network's classifier: ")
print(classification_report(y_test, y_pred_nn))
print(confusion_matrix(y_test, y_pred_nn))
```

```
# Load Cifar10 Dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

# Data Preprocessing
x_train = x_train.reshape(-1, 32 * 32 * 3).astype("float32") / 255
x_test = x_test.reshape(-1, 32 * 32 * 3).astype("float32") / 255

# Retraining the SVM classifier
svm_classifier.fit(x_train, np.ravel(y_train))

# Re-evaluating the SVM classifier
y_pred_svm = svm_classifier.predict(x_test)
print("evaluate the classifier of SVM (Cifar10) : ")
print(classification_report(y_test, y_pred_svm))
print(confusion_matrix(y_test, y_pred_svm))

# Retraining neural networks
model.fit(x_train, np.ravel(y_train), epochs=10, batch_size=128, validation_split=0.1)

# Re-evaluating neural networks
y_pred_nn = np.argmax(model.predict(x_test), axis=-1)
print("evaluate the classifier of neural networks (Cifar10) : ")
print(classification_report(y_test, y_pred_nn))
print(confusion_matrix(y_test, y_pred_nn))
```

Comparative Analysis of Support Vector Machines and Fully Connected Neural Networks for Image Classification

GRADEMARK REPORT

FINAL GRADE

72/100

GENERAL COMMENTS

Instructor

Report structured and includes appropriate references. The review of related work could be a little more detailed in order to support the analysis of the results - eg compare with results obtained by others.

I asked you to avoid use of CNNs so the section discussing these is overly detailed. Also concentrate on the classifiers you chose - any two I asked for. I appreciate it is important to provide context and related models/ideas but I think you have used a lot of your word count here when you could have discussed other aspects in more detail - eg PCA/ dimensionality reduction.

Dimensionality reduction would be useful to apply and investigate, perhaps.

A good review of hyperparameters - well done. A very good discussion of results. Again, well done. Use some diagrams to help show your architectural design of the MLP.

Your results for CIFAR10 are very good. Double check these.

You could include additional refs if necessary.

PAGE 1



Comment 1

expand the abbreviation on first use

PAGE 2



Comment 2

used to



Comment 3

grammar

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

GRADING FORM: SMU GENERIC_GJ

YOUQIAN CHEN

0

RESEARCH-INFORMED LITERATURE

KNOWLEDGE AND UNDERSTANDING

ANALYSIS

PRACTICAL APPLICATION AND DEP.

SKILLS FOR PROFESSIONAL PRAC.