Lab Exercise 3

Bing-Je Wu

IST 718 – big data analytics

Table of Contents

[Introduction 2](#_Toc34315702)

[Dataset 2](#_Toc34315703)

[Methods for Analysis 2](#_Toc34315704)

[Data and Data Processing 2](#_Toc34315705)

[Exploratory Analysis 3](#_Toc34315706)

[Data Modeling 5](#_Toc34315707)

[Linear Classification (SGDClassifier) 5](#_Toc34315708)

[Logistic Regression (LogisticRegression) 6](#_Toc34315709)

[Support Vector Machine (SVC) 7](#_Toc34315710)

[Naïve Bayes (GaussianNB) 8](#_Toc34315711)

[K-Nearest Neighbors (KNeighborsClassifier) 9](#_Toc34315712)

[Random Forest (RandomForestClassifier) 10](#_Toc34315713)

[Keras – ANN (Artificial Neural Network) 11](#_Toc34315714)

[Keras – CNN (Convolutional Neural Network) 13](#_Toc34315715)

[Final Model 15](#_Toc34315716)

[Questions 18](#_Toc34315717)

[Conclusion 19](#_Toc34315718)

[Reference 19](#_Toc34315719)

# Introduction

Computer Vision, often abbreviated as CV, is an interdisciplinary scientific field that deals with how computers can made to gain high-level understanding from digital images or videos. MINST database (National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. It is a hello-world challenge in the computer vision world for image recognition. To advance the image recognition ability, a Fashion-MNIST database is built for this task. The objectives for this lab is to use algorithms and compute to identify clothing items. The following research questions will be answered:

1. What is the accuracy of each method?
2. What are the trade-offs of each approach?
3. What is the compute performance of each approach?

# Dataset

This lab exercise contains 1 dataset from the sources : The Fashion-MNIST database is Zalando’s article images (<https://github.com/zalandoresearch/fashion-mnist>) ; (<https://research.zalando.com/welcome/mission/research-projects/fashion-mnist/>). The Fashion-MNIST is a dataset that consists of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28×28 grayscale image, associated with a label from 10 classes. Fashion-MNIST is intended to serve as a direct drop-in replacement of the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

# Methods for Analysis

Eight (8) algorithms were implemented for benchmarking machine learning algorithms during this lab exercise. Six (6) machine learning algorithm, 'Linear Classification', 'Logistics Regression', 'Support Vector', 'Naïve Bayes', 'K-Nearest Neighbors' and 'Random Forest' were applied from **scikit-learn**. An ‘Artificial Neural Network’ algorithm and a ‘Convolutional Neural Network’ were implemented in **TensorFlow** (2.0 version) by using **Keras**.

# Data and Data Processing

In order to complete the lab exercise for the image recognition task seamlessly, an environment setup must be addressed. The google Colab cloud services was chosen for this lab exercise to ease the effort of configuration and computation on running algorithm with GPU. According to the user guide of the Fashion-MNIST database, the dataset is stored in the Keras library. After loading the training images, training labels, test image, and test labels, some pre-processing steps should be applied, such as normalizing the images, reshaping the dataset, and shuffling index of training dataset. With the normalization on the images dataset (divide each pixel value of the image by the maximum number of pixel value at 255, it can help make sure that models will be trained faster and perform better. Reshaping the training dataset is a must-do process since it is in three-dimensions shape (number of instances, height, width). A standard format should be in two-dimension shape (number of instances, total number of pixels [height\*width]). Last, shuffling index helps adjust the combination of weights of each synapse when the neural network model is running in batches and preserve the properties of randomness. A four-dimensional shape dataset (number of instances, height, width, channels) had been converted for running convolutional neural network.

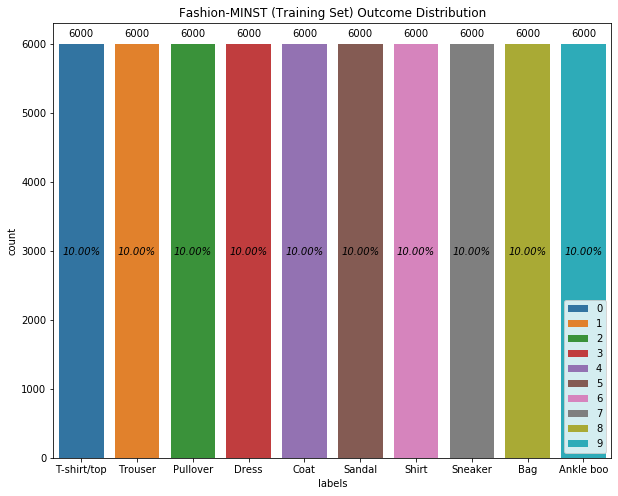
# Exploratory Analysis

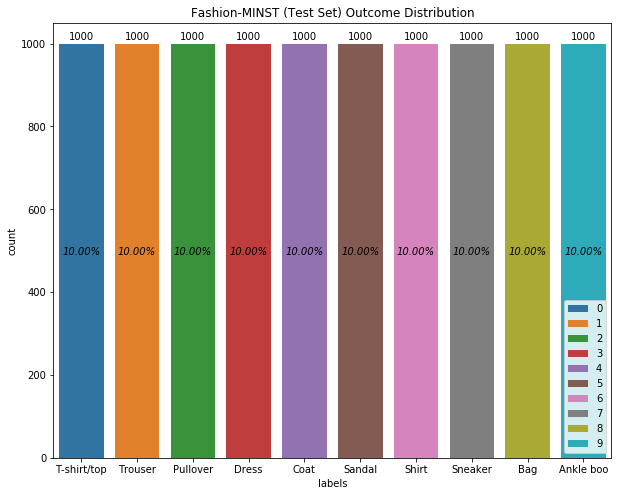
Images for each of clothes (with encoded code) and the variation of a cloth image (Coat) are shown below:





From the plot of outcome distribution for training set and test set, we can make sure that the dataset will not have any training bias during the training process. Each of classes is evenly distributed across training set and test set and the ratio are the same.





# Data Modeling

Eight (8) models, 'Linear Classification', 'Logistics Regression', 'Support Vector', 'Naïve Bayes', 'K-Nearest Neighbors', 'Random Forest', 'ANN (Artificial Neural Network)’ and ‘Convolutional Neural Network’, were built to classify the fashion items. Each model was built by training set with 60,000 records and validated by test set with 10,000 records.

## Linear Classification (SGDClassifier)

SGDClassifier is a linear classifier. This model implemented regularized linear models with stochastic gradient descent (SGD) learning (good for finding global minimum). The gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). This algorithm works best when the data have zero mean and unit variance. The model was built in default setting.

From the confusion matrix and classification report shown below, we can observe that **Trouser (encoded as 1)** was correctly classified at 96.2% of Recall (True positive rate) among others. And **Coat** **(encoded as 4)** has the worst performance on the classification at 48.2% of Recall (True positive rate). The overall model performance has 81.03% of accuracy.

|  |
| --- |
| [[770 5 22 19 1 4 168 0 11 0]  [ 4 962 8 17 1 0 5 1 2 0]  [ 14 3 771 4 56 0 144 1 7 0]  [ 41 32 27 749 9 0 134 0 8 0]  [ 1 5 179 20 482 0 303 0 10 0]  [ 3 0 2 0 0 919 0 43 11 22]  [111 2 136 17 37 0 676 0 21 0]  [ 1 0 0 0 0 38 0 913 2 46]  [ 5 1 7 6 1 7 31 3 938 1]  [ 1 1 0 2 0 33 4 35 1 923]] |

|  |
| --- |
| precision recall f1-score support  0 0.8097 0.7700 0.7893 1000  1 0.9515 0.9620 0.9567 1000  2 0.6693 0.7710 0.7165 1000  3 0.8981 0.7490 0.8168 1000  4 0.8211 0.4820 0.6074 1000  5 0.9181 0.9190 0.9185 1000  6 0.4614 0.6760 0.5485 1000  7 0.9167 0.9130 0.9148 1000  8 0.9278 0.9380 0.9329 1000  9 0.9304 0.9230 0.9267 1000  accuracy 0.8103 10000  macro avg 0.8304 0.8103 0.8128 10000  weighted avg 0.8304 0.8103 0.8128 10000 |

## Logistic Regression (LogisticRegression)

Logistic regression is a method for binary classification. It works to divide points in a dataset into two distinct classes, or categories. It creates a sort of S-curve (using the sigmoid function) which help show certainty, since the output from logistic regression is not just a one or zero. The multinomial logistic regression is a classification method that generalizes logistic regression to multiclass problems.

The Logistic Regression model from sklearn was implemented to classify the multiple classes, 10 classes of items from Fashion-MINST dataset. The algorithm from sklearn implements regularized logistic regression using the ‘liblinear’ library, ‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’ solvers. The model was built in default setting (solver = ‘lbfgs’, and penalty='l2') with adjustment on max\_iter=10000.

From the two metrics shown below, we can observe that **Trouser (encoded as 1)** was correctly classified at 95.8% of Recall (True positive rate) among others. And **Shirt (encoded as 6)** has the worst performance on the classification at 57.1% of Recall (True positive rate). The overall model performance has 84.41% of accuracy.

|  |
| --- |
| [[806 2 11 53 4 2 111 0 11 0]  [ 4 958 3 25 4 0 3 1 2 0]  [ 24 4 739 10 124 0 86 1 12 0]  [ 24 17 18 861 30 0 39 0 11 0]  [ 0 2 115 36 763 0 77 0 7 0]  [ 0 0 0 1 0 922 0 48 7 22]  [143 2 123 38 100 0 571 0 23 0]  [ 0 0 0 0 0 35 0 939 0 26]  [ 7 1 7 14 5 6 21 5 934 0]  [ 0 1 0 0 0 12 1 38 0 948]] |

|  |
| --- |
| precision recall f1-score support  0 0.7996 0.8060 0.8028 1000  1 0.9706 0.9580 0.9643 1000  2 0.7274 0.7390 0.7331 1000  3 0.8295 0.8610 0.8449 1000  4 0.7408 0.7630 0.7517 1000  5 0.9437 0.9220 0.9327 1000  6 0.6282 0.5710 0.5982 1000  7 0.9099 0.9390 0.9242 1000  8 0.9275 0.9340 0.9307 1000  9 0.9518 0.9480 0.9499 1000  accuracy 0.8441 10000  macro avg 0.8429 0.8441 0.8433 10000  weighted avg 0.8429 0.8441 0.8433 10000 |

## Support Vector Machine (SVC)

Support vector machine (SVMs) are a set of supervised learning methods used for classification. The goal of SVM is to try and find a line or hyperplane to divide a dimensional space which best classifies the data points. The algorithm chooses the line/hyperplane with the maximum margin. Maximizing the margin will give us the optimal line to classify the data.

In sci-learn, SVC, NuSVC and LinearSVC are classes capable of performing multi-class classification on a dataset. The SVC class was implemented in this lab and is based on libsvm. The computation cost of this algorithm at least quadratically with the number of samples and it is impractical for dataset that has beyond tens of thousands of samples. The model was built in default setting (kernel=’rbf’ [Radial Basis Function for non-linearly separable data])

The confusion matrix and classification report show that **Bag (encoded as 8)** was correctly classified at 97.7% of Recall (True positive rate) among others. And **Shirt (encoded as 6)** has the worst performance on the classification at 65.5% of Recall (True positive rate). The overall model performance has 88.29% of accuracy.

|  |
| --- |
| [[857 0 16 28 3 2 85 0 9 0]  [ 4 962 2 25 3 0 4 0 0 0]  [ 11 2 816 16 88 0 65 0 2 0]  [ 27 3 11 890 33 0 32 0 4 0]  [ 1 1 87 32 815 0 61 0 3 0]  [ 0 0 0 1 0 951 0 33 1 14]  [135 1 103 27 68 0 655 0 11 0]  [ 0 0 0 0 0 21 0 955 0 24]  [ 3 1 1 5 2 2 4 5 977 0]  [ 0 0 0 0 0 11 1 37 0 951]] |

|  |
| --- |
| precision recall f1-score support  0 0.8256 0.8570 0.8410 1000  1 0.9918 0.9620 0.9766 1000  2 0.7876 0.8160 0.8016 1000  3 0.8691 0.8900 0.8794 1000  4 0.8053 0.8150 0.8101 1000  5 0.9635 0.9510 0.9572 1000  6 0.7222 0.6550 0.6869 1000  7 0.9272 0.9550 0.9409 1000  8 0.9702 0.9770 0.9736 1000  9 0.9616 0.9510 0.9563 1000  accuracy 0.8829 10000  macro avg 0.8824 0.8829 0.8824 10000  weighted avg 0.8824 0.8829 0.8824 10000 |

## Naïve Bayes ([GaussianNB](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB))

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes’ theorem with the “naive” assumption of conditional independence between every pair of features given the value of the class variable. It uses prior probability (probabilities we already know) to find the posterior probability (determine how to classify input). These probabilities are related to existing classes and what features they have.

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian. The model was built in default setting.

By observing from the two metrics shown below, we can see that **Sneaker (encoded as 7)** was correctly classified at 98.8% of Recall (True positive rate) among others. And **Shirt (encoded as 6)** has the worst performance on the classification at 4% of Recall (True positive rate) among all the algorithms. The overall model performance has 58.56% of accuracy. It is the lowest accuracy rate among others.

|  |
| --- |
| [[586 64 29 162 110 0 20 0 29 0]  [ 1 939 14 36 7 0 1 0 2 0]  [ 7 14 324 65 545 0 23 0 22 0]  [ 9 387 6 545 43 0 4 0 6 0]  [ 0 34 44 131 779 0 4 0 8 0]  [ 0 0 1 1 0 278 3 660 5 52]  [117 34 112 200 435 0 40 0 62 0]  [ 0 0 0 0 0 3 0 988 0 9]  [ 0 2 19 85 149 3 27 4 710 1]  [ 0 0 1 1 0 16 3 304 8 667]] |

|  |
| --- |
| precision recall f1-score support  0 0.8139 0.5860 0.6814 1000  1 0.6370 0.9390 0.7591 1000  2 0.5891 0.3240 0.4181 1000  3 0.4445 0.5450 0.4897 1000  4 0.3767 0.7790 0.5078 1000  5 0.9267 0.2780 0.4277 1000  6 0.3200 0.0400 0.0711 1000  7 0.5051 0.9880 0.6685 1000  8 0.8333 0.7100 0.7667 1000  9 0.9150 0.6670 0.7715 1000  accuracy 0.5856 10000  macro avg 0.6361 0.5856 0.5562 10000  weighted avg 0.6361 0.5856 0.5562 10000 |

## K-Nearest Neighbors (KNeighborsClassifier)

K-Nearest Neighbors (KNN) is a basic classifier for machine learning. A classifier takes a labeled data set, and then it tries to label new data points into one of the categories. To do this, the closest points (neighbors) to the object will be examined. The class with the majority of neighbors will be the class that identifies the object to be in. The k is the number of nearest neighbors to the object. To find the neighbors, different methods of calculating distances can be implemented, such as ‘Euclidean distance’ and ‘Manhattan Distance’… etc.

KNeighborsClassifier is the classifier implementing the k-nearest neighbors vote based on the nearest neighbors of each query point. Classification is computed from a simple majority vote of the nearest neighbors of each point. The KNeighborsClassifier was implemented to build a model to classify the fashion-MINST dataset. The model was built in default setting. (n\_neighbors=5, metric = ‘minkowski’, p=2 [Euclidian\_distance], weights=’uniform’)

The confusion matrix and classification report show that **Trouser (encoded as 1)** and **Ankle boo (encoded as 9)** were both correctly classified at 96.8% of Recall (True positive rate) among others. And **Shirt (encoded as 6)** has the worst performance on the classification at 57.5% of Recall (True positive rate). The overall model performance has 85.54% of accuracy.

|  |
| --- |
| [[855 1 17 16 3 1 100 1 6 0]  [ 8 968 4 12 4 0 3 0 1 0]  [ 24 2 819 11 75 0 69 0 0 0]  [ 41 8 15 860 39 0 34 0 3 0]  [ 2 1 126 26 773 0 71 0 1 0]  [ 1 0 0 0 0 822 5 96 1 75]  [176 1 132 23 80 0 575 0 13 0]  [ 0 0 0 0 0 3 0 961 0 36]  [ 2 0 10 4 7 0 16 7 953 1]  [ 0 0 0 0 0 2 1 29 0 968]] |

|  |
| --- |
| precision recall f1-score support  0 0.7710 0.8550 0.8108 1000  1 0.9867 0.9680 0.9773 1000  2 0.7293 0.8190 0.7715 1000  3 0.9034 0.8600 0.8811 1000  4 0.7880 0.7730 0.7804 1000  5 0.9928 0.8220 0.8993 1000  6 0.6579 0.5750 0.6137 1000  7 0.8784 0.9610 0.9179 1000  8 0.9744 0.9530 0.9636 1000  9 0.8963 0.9680 0.9308 1000  accuracy 0.8554 10000  macro avg 0.8578 0.8554 0.8546 10000  weighted avg 0.8578 0.8554 0.8546 10000 |

## Random Forest (RandomForestClassifier)

Random Forest is an ensemble learning technique. It is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset (bootstrapped dataset) and uses averaging to improve the predictive accuracy and control over-fitting. The RandomForestClassifier was implemented to build a model. The model was built in default setting. (n\_estimators=100)

From the confusion matrix and classification report shown below, we can observe that **Bag (encoded as 8)** was correctly classified at 97.2% of Recall (True positive rate) among others. And **Shirt (encoded as 6)** has the worst performance on the classification at 59.2% of Recall (True positive rate). The overall model performance has 87.69% of accuracy.

|  |
| --- |
| [[862 0 11 31 2 1 83 0 10 0]  [ 3 961 2 21 5 0 6 0 2 0]  [ 12 0 803 10 117 0 54 0 4 0]  [ 18 2 11 909 30 0 28 0 2 0]  [ 1 0 94 33 820 0 50 0 2 0]  [ 0 0 0 1 0 956 0 32 1 10]  [155 1 117 28 91 0 592 0 16 0]  [ 0 0 0 0 0 17 0 949 0 34]  [ 1 1 6 2 5 2 7 4 972 0]  [ 0 0 0 0 0 11 0 41 3 945]] |

|  |
| --- |
| precision recall f1-score support  0 0.8194 0.8620 0.8402 1000  1 0.9959 0.9610 0.9781 1000  2 0.7692 0.8030 0.7857 1000  3 0.8783 0.9090 0.8934 1000  4 0.7664 0.8200 0.7923 1000  5 0.9686 0.9560 0.9623 1000  6 0.7220 0.5920 0.6505 1000  7 0.9250 0.9490 0.9368 1000  8 0.9605 0.9720 0.9662 1000  9 0.9555 0.9450 0.9502 1000  accuracy 0.8769 10000  macro avg 0.8760 0.8769 0.8756 10000  weighted avg 0.8760 0.8769 0.8756 10000 |

## Keras – ANN (Artificial Neural Network)

Keras is a high-level API meant to be for TensorFlow. ‘tf.keras.Sequential()’ function was implemented to build a stack of layers for the ANN model, also known as fully connected network. Six (6) layers were built in the neural network. The neural network model consists of three (3) dense layer, two (2) dropout layer and one (1) output layer [another dense layer]. Each of layers was set up (with different number of neurons and activation function) for tuning the best result. The first layer consists of 128 neurons and ‘relu’ activation function. The second layer is a dropout layer as a regularization technique where neurons will be set to zero in layer. In that case, it will the chosen neuron will not be updated on the weight of synapse while doing the back propagation. The third layer consists of 64 neurons with ‘relu’ activation function. The fourth layer is another dropout layer. The fifth layer consist of 64 neurons and ‘linear’ activation function. And the last layer is the output layer with 10 neurons as our output classes. The total number of weights (parameters) is 113,546. The ANN model was built with ‘adam’ optimizer (for first-order gradient-based optimization of stochastic objective functions) and using ‘softmax’ as the metric of the loss function. 30 batches (epochs) were set during the training process.

|  |
| --- |
| ANN.summary():  Model: "sequential"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  dense (Dense) (None, 128) 100480  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout (Dropout) (None, 128) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_1 (Dense) (None, 64) 8256  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_1 (Dropout) (None, 64) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_2 (Dense) (None, 64) 4160  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_3 (Dense) (None, 10) 650  =================================================================  Total params: 113,546  Trainable params: 113,546  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

The confusion matrix and classification report show that **Bag (encoded as 8)** was correctly classified at 97.9% of Recall (True positive rate) among others. And **Shirt (encoded as 6)** has the worst performance on the classification at 69.8% of Recall (True positive rate). The overall model performance has 88.67% of accuracy.

|  |
| --- |
| [[789 2 9 52 9 1 129 0 9 0]  [ 0 975 1 17 4 0 2 0 1 0]  [ 16 2 797 12 111 0 61 0 1 0]  [ 17 6 8 908 41 0 16 0 4 0]  [ 0 1 81 28 848 0 40 0 2 0]  [ 0 0 0 0 0 959 0 24 1 16]  [ 78 1 88 46 78 0 698 0 11 0]  [ 0 0 0 0 0 13 0 974 0 13]  [ 1 0 0 6 3 1 7 3 979 0]  [ 0 0 0 0 0 7 1 52 0 940]] |

|  |
| --- |
| precision recall f1-score support  0 0.8757 0.7890 0.8301 1000  1 0.9878 0.9750 0.9814 1000  2 0.8100 0.7970 0.8034 1000  3 0.8494 0.9080 0.8777 1000  4 0.7751 0.8480 0.8099 1000  5 0.9776 0.9590 0.9682 1000  6 0.7317 0.6980 0.7144 1000  7 0.9250 0.9740 0.9489 1000  8 0.9712 0.9790 0.9751 1000  9 0.9701 0.9400 0.9548 1000  accuracy 0.8867 10000  macro avg 0.8874 0.8867 0.8864 10000  weighted avg 0.8874 0.8867 0.8864 10000 |

## Keras – CNN (Convolutional Neural Network)

Keras provides the function to build a Convolutional Neural Network (CNN), also known as deep learning model. The CNN model has deeper layers than ANN model. There were thirteen (13) layers were built in this neural network. The convolutional neural network model consists of four (4) convolutional layers [feature filters], two (2) pooling layers, one (1) flatten layer, three (3) dense layer, two (2) dropout layer and one (1) output layer [another dense layer]. The first and second convolution layers consist of 32 filters with the kernel matrix size of 3 by 3, ‘same’ padding setup (dimension stays the same) and ‘relu’ activation function. Then, it follows a Max Pooling layer with pool size of 2 by 2, strides at 2, and no padding added. The following 2 convolution layers consist of 64 filters with the kernel matrix size of 3 by 3, ‘same’ padding setup (dimension stays the same) and ‘relu’ activation function. Another Max Pooling layer with strides at 2, and no padding setting was added. The next layer is Flatten layers to flatten the matrixes in order to move forward to the fully connected neural network. The structures are the same as the ANN with miner changes on hyper-parameters. The first dropout layer was changed on the rate from 0.2 to 0.4. And the number of epochs was set up as 30. The rest are remained the same. The total number of weights (parameters) is 477,194. The ANN model was built with ‘adam’ optimizer (for first-order gradient-based optimization of stochastic objective functions) and using ‘softmax’ as the metric of the loss function. 30 batches (epochs) were set during the training process.

|  |
| --- |
| CNN.summary():  Model: "sequential\_1"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  conv2d (Conv2D) (None, 28, 28, 32) 320  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_1 (Conv2D) (None, 28, 28, 32) 9248  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling2d (MaxPooling2D) (None, 14, 14, 32) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_2 (Conv2D) (None, 14, 14, 64) 18496  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_3 (Conv2D) (None, 14, 14, 64) 36928  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling2d\_1 (MaxPooling2 (None, 7, 7, 64) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  flatten (Flatten) (None, 3136) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_4 (Dense) (None, 128) 401536  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_2 (Dropout) (None, 128) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_5 (Dense) (None, 64) 8256  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_3 (Dropout) (None, 64) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_6 (Dense) (None, 32) 2080  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_7 (Dense) (None, 10) 330  =================================================================  Total params: 477,194  Trainable params: 477,194  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

The confusion matrix and classification report show that **Bag (encoded as 8)** was correctly classified at 99% of Recall (True positive rate) among others. And **Shirt (encoded as 6)** has the worst performance on the classification at 74% of Recall (True positive rate). The overall model performance has 92.52% of accuracy. It is the highest accuracy rate among others

|  |
| --- |
| [[897 1 14 8 1 0 75 0 4 0]  [ 0 985 0 9 0 0 4 0 2 0]  [ 18 2 909 8 32 0 30 0 1 0]  [ 23 0 11 933 9 0 23 0 1 0]  [ 0 0 59 28 871 0 41 0 1 0]  [ 0 0 0 0 0 978 0 13 0 9]  [118 2 47 22 66 0 740 0 5 0]  [ 0 0 0 0 0 8 0 982 0 10]  [ 1 0 0 4 1 2 1 1 990 0]  [ 0 0 0 0 0 5 1 27 0 967]] |

|  |
| --- |
| precision recall f1-score support  0 0.8486 0.8970 0.8721 1000  1 0.9949 0.9850 0.9899 1000  2 0.8740 0.9090 0.8912 1000  3 0.9219 0.9330 0.9274 1000  4 0.8888 0.8710 0.8798 1000  5 0.9849 0.9780 0.9814 1000  6 0.8087 0.7400 0.7728 1000  7 0.9599 0.9820 0.9708 1000  8 0.9861 0.9900 0.9880 1000  9 0.9807 0.9670 0.9738 1000  accuracy 0.9252 10000  macro avg 0.9249 0.9252 0.9247 10000  weighted avg 0.9249 0.9252 0.9247 10000 |

## Final Model

The comparison charts of the algorithm classification performance are shown below. The first chat compares the performance of training time and testing with eight approaches. The x-axis is the computation time in second, and the y-axis is the name of the algorithms used in the lab exercise. The second chart combines the training time and testing time together with the score rank in color coded. The last chart shows the accuracy rate for each of approaches with rounded percentage.

From the three charts below, among the eight (8) algorithms, we can observe that Support Vector Machine model has the highest computation cost in overall. Although it has pretty good performance on the score, its total computation time still has 788 seconds. According to the document of the sckit-learn, support vector machine (SVC) is not good for huge number of the dataset since its computation time is growing exponentially. It supports the reason why the SVC model is not the best choice among others.

K-Nearest Neighbors model has less computation cost on training process because it simply stores instances of the training data. But the computation is costly on prediction phase. It is not a practical practice in the production. There is a philosophy for choosing a model for production. We would prefer an algorithm requires longer computation cost and have the highest accuracy rather than a model has higher accuracy but requires longer computation time in predicting.

CNN (Convolutional Neural Network) model has a quite long computation time on training phase but no computation cost on testing phase at all. Its performance on the score is phenomenal, almost 93% of accuracy rate. Its recalls (True Positive Rate) for each of classes are higher than 70%. It fits the philosophy that we have mentioned earlier. CNN model is so far the best model we have now.

Logistic model has higher computation cost on training phase as well. It has no computation cost on testing phase, and the accuracy rate is about 84%. Although its performance looks well, the classification report gives us a clue that the Logistic model is not the best model. It has poor performance on classifying the class 6, Shirt, with the true positive rate less than 60%.

ANN (Artificial Neural Network) model has the less computation cost and satisfying result. However, its performance on classification report is not as good as CNN model.

Random Forest model has pretty good performance in overall. It has less computation cost and high accuracy rate at 88%. But, by comparing the true positive rate (recall), ANN and CNN models have overall higher rate for each of classes than random forest model.

Linear Classification (SGDClassifier) works fine for the fashion-MINST dataset. It has at least 81% of accuracy rate. But it is hard to say this algorithm will be suitable for another dataset.

Naïve Bayes (GaussianNB) seems to have the least computation time on both training phase and testing phase, less than 1 second, because of the property of the Naïve Bayes Algorithm. But its performance is the worst among others. Thus, we can exclude GaussianNB model from our option of the best model.

With Consideration of all the factors, I would recommend the CNN (Convolutional Neural Network) model if we can accept cost on training phase in order to get the highest prediction. The second choice will be ANN (Artificial Neural Network) model.

Chart 1 & 2: Computation Cost Comparison

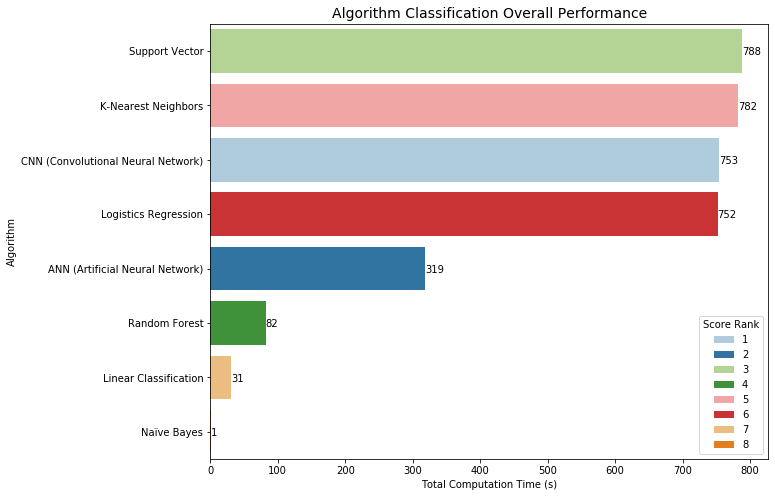
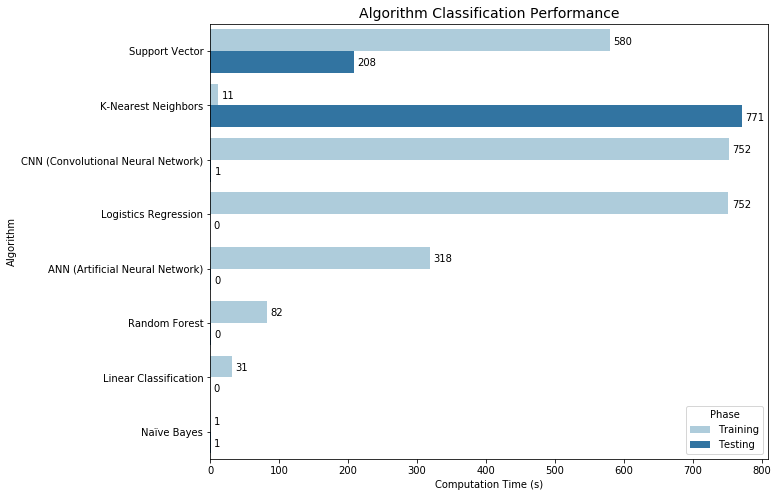
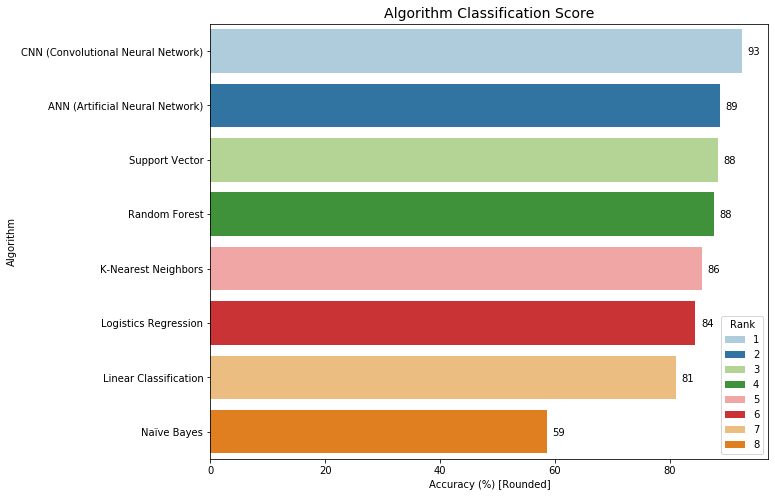


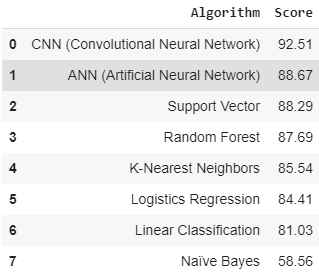
Chart 3: Score (accuracy rate) Comparison



# Questions

1. **What is the accuracy of each method?**

The model accuracy rate is shown below:



1. **What are the trade-offs of each approach?**

Each approach has its own advantage and disadvantage.

Convolutional Neural Network works the best among others. However, it comes with the package that the computation cost on training could be higher depending on how deep the network is. Artificial Neural Network works the same but has less computation cost and slightly off on the accuracy since it cannot detect every aspect of features as Convolutional Neural Network does.

Support Vector Machine (SVM) works effectively in classifying higher dimensional space and saves spaces on memory because it only uses the support vectors to create the optimal line. It is the best classifier when data points are separable. But, SVM performs poorly when the classes are overlapping, such as non-separable data points, and it is limited to small dataset. The bigger the training data, the higher the computation cost. Therefore, in our case, it has higher computation cost and higher accuracy rate.

Random forest has pretty good overall performance. It can handle both linear and non-linear data. But, one thing must be aware of. That is the higher depth of trees could bring the issue of the overfitting.

K-Nearest Neighbors has higher computation cost on predicting It is not good for production since it requires more computation cost in predicting than training.

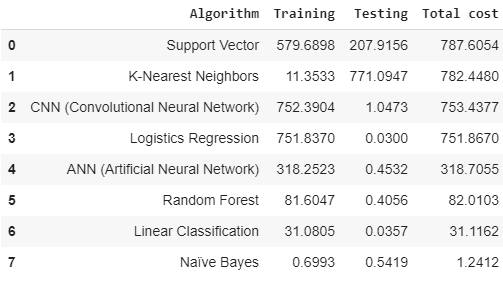
Logistic regression works fairly and it can be used in practice but it may require more computation cost.

Linear classification works just fine on the fashion-MINST dataset, but its performance is depending on the dataset. Linear classification works better on linear dataset with scaled input.

Naïve Bayes is the simplest approach and known for faster computation compared to more sophisticated methods. When the training data contains continuous attribute, Gaussian Naïve Bayes is the choice. When the feature vectors represent frequencies, Multinomial Naïve Bayes should be implemented. When, the input variables are independent Booleans (binary variables), the Bernoulli Naïve Bayes is the tool to use. Overall, Naïve Bayes can perform will when the input variable is normally distributed and its predictors are independent to each other. But, clearly, it is not the case for our fashion-MINST dataset.

1. **What is the compute performance of each approach?**

The computation cost for each approach is shown below:



# Conclusion

The fashion-MINST dataset was used for the eight algorithms. Among the 10 classes, 'T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', and 'Ankle boo', it is easy to find that ‘Shirt’ is the class that is hard to be classified by each of approaches. The reason could be that ‘Shirt’ shares the similar features with 'T-shirt/top’, ‘Pullover', and ‘Coat’. We can tell that by observing 'T-shirt/top’, ‘Pullover' and ‘Coat’ are not the top easy to be classified classes. ‘Trouser’ and ‘Bag’ are the classes that are easy to be classified because they have the clear features that are clearly different from others. In order to improve the ability of images recognition, the complicated neural network, Convolution Neural Network (CNN), should be built deeper for learning the varies of features with feature detector.

# Reference

Tensorflow.org. [Web Link](https://www.tensorflow.org/tutorials/keras/classification) ; 2 Udemy – TensorFlow 2.0. [Web Link](https://github.com/bing020815/TensorFlow/blob/master/2_Building_an_Artificial_Neural_Network_in_TensorFlow_2_0.ipynb)  
3 MNIST database, from Wikipedia. [Web Link](https://en.wikipedia.org/wiki/National_Institute_of_Standards_and_Technology) ; 4 SGDClassifier, from scikit-learn. [Web Link](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier)  
5 LogisticRegression, from scikit-learn. [Web Link](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html?highlight=logist#sklearn.linear_model.LogisticRegression)1 ; 6 SVC, from scikit-learn. [Web Link](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)1 , [Web Link](https://scikit-learn.org/stable/modules/svm.html#svm-classification)2   
7 Naive Bayes, from scikit-learn. [Web Link](https://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes) ; 8 KNeighborsClassifier, from scikit-learn. [Web Link](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier)  
9 RandomForestClassifier, from scikit-learn. [Web Link](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlight=random%20forest#sklearn.ensemble.RandomForestClassifier) ; 10 Keras, from TensorFlow. [Web Link](https://www.tensorflow.org/api_docs/python/tf/keras/Sequential)   
11 Stochastic gradient descent, from Wikipedia. [Web Link](https://en.wikipedia.org/wiki/Stochastic_gradient_descent) ; 12 Precision and recall, from Wikipedia. [Web Link](https://en.wikipedia.org/wiki/Precision_and_recall)   
13 Adam: A Method for Stochastic Optimization. [Web Link](https://arxiv.org/abs/1412.6980v8) ; 14 2D Visualization of a Convolutional Neural Network. [Web Link](https://www.cs.ryerson.ca/~aharley/vis/conv/flat.html)   
15 Naive Bayes classifier, from Wikipedia. [Web Link](https://en.wikipedia.org/wiki/Naive_Bayes_classifier) ; 16 Machine Learning Course with Python. [Web Link](https://github.com/machinelearningmindset/machine-learning-course)   
17 Understanding Convolutions for Deep Learning. [Web Link](https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1) ; 18 Machine Learning & Deep Learning Fundamentals [Web Link](https://deeplizard.com/learn/video/gZmobeGL0Yg)   
19 About Convolutional Layer and Convolution Kernel. [Web Link](https://www.sicara.ai/blog/2019-10-31-convolutional-layer-convolution-kernel) ; 20 Neural networks [Web Link](https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi)