Comparative Study of Machine Learning Models for Image Classification of Fashion Items

Joel Ang Lang Yi
Lok You Tan
Cheong Min Wei
Hu Wanqing
May Chan Shu Zhen
Kan Yip Keng

Abstract

There is a wide variety of supervised machine learning algorithms, each with its inspirations and roots, advantages and disadvantages. We seek to explore these algorithms in detail to gain a deeper understanding of them and how they perform compared to each other for image classification.

To do that, we compared the models' ability to classify images of fashion items and identified which of them are best suited to be employed in several situations that required emphasis in different aspects of performance.

Finally, using our acquired insights, we developed an original model JoNet-0 and achieved better accuracy than that achieved by the models we had previously implemented.

1. Introduction

There are indeed already an incredible amount of resources available that discusses different individual parts of what has been studied in this paper. The main aim of the paper is not to be the first to study these models, but to discuss several interesting ideas, new and old, and explore implementation details for each model. We will be studying 5 algorithms: Logistic Regression, k-Nearest Neighbors, Support Vector Machine, Multilayer Perceptron, and Convolutional Neural Network.

We decided to use the Fashion-MNIST dataset that has 60,000 28x28 grayscale images of 10 balanced classes of fashion items. With the rise of the Internet of Things, the surge in data influx, and increasingly widespread automation, image classification has many real-world applications today. Specifically, we believe being able to accurately classify fashion items has many significant applications in areas like security, search and rescue, retail, etc.

2. Experiment Framework

We will use python libraries, scikit-learn and Keras, to implement the traditional methods and deep learning methods respectively.

Traditional Machine Learning Algorithms

Grid search and k-fold cross validation will be used to tune these models. For each algorithm, there are sub-algorithms or techniques that will be further explored and compared. We will select the best sub-algorithm, after tuning the parameters available in scikit-learn, based on the F_1 score on the test set.

Deep Learning Methods

The first model we will develop is a Multilayer Perceptron (MLP), using "grad student descent" and validation to tune. We will experiment with image augmentation to see if it improves performance of the model.

Next, we will implement a model as close to LeNet-5, published by Yann LeCun [1], as possible to compare with the MLP model.

Finally, we will develop an original model JoNet-0 that is an improvement of LeNet-5 that will be tailored specifically for dataset we are using.

Metrics

The F_β score is the weighted harmonic mean of precision and recall. As our classes are well balanced, macro averaged F_1 score directly reflects the accuracy of the model and will therefore be our main metric for comparison. By changing the value β, we are able to change our emphasis on either precision or recall. We will be using $F_{0.5}$ and F_2 to evaluate the models as well to simulate different scenarios.

$$F_\beta = (1+\beta^2)\frac{\text{precision}\times\text{recall}}{(\beta^2\times\text{precision})\times\text{recall}}$$

Training and testing time might also be a key factor for comparison. Some applications require real-time training, or instant responses for large amounts of predictions.

3. Logistic Regression

Motivation

Logistic regression (LGR) is a classic model that is taught in any introductory machine learning module. We naturally want to further explore this model in terms of implementation details and compare it to other models.

Description

LGR is a binary classification technique. It attempts to classify the inputs using a linear equation, where the weights are maximum likelihood estimates of the true relationship between the predictors given the data using the sigmoid function as the probability density function of the data.

$$p(x|\theta) = \frac{1}{1+e^{-\theta^{\mathrm{T}}x}}$$

Multiclass-LGR uses the softmax function instead of the sigmoid.

$$g(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \text{ for j} = 1, \ldots, \text{K}$$

Finally, an iterative algorithm will be used to find a local optimal value of θ. There are several algorithms to choose from, we will implement each of them to evaluate their performances.

a. Solvers

Large Linear Classification

It is a library provided for large-scale linear classification. The algorithm is done by making a classification decision based on the value of a linear combination of the features. It uses coordinate descent, which successively approximates minimization along coordinate directions or coordinate hyperplanes.

Stochastic Average Gradient

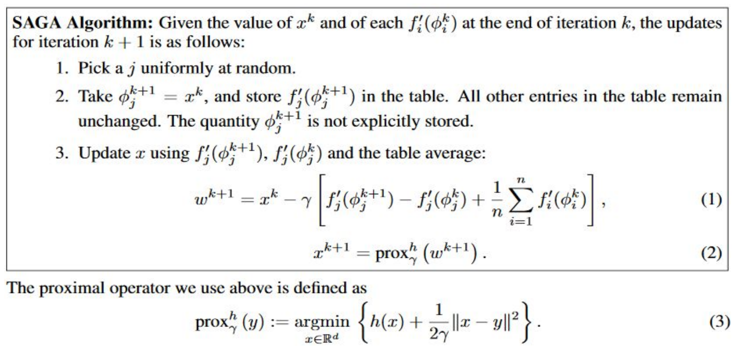
A randomized variant of the incremental aggregated gradient method, SAG has low iteration cost but through incorporating a memory of previous gradient values, the SAG method achieves a faster convergence rate. The iterations take the form:

$$x^{k+1} = x^k - \frac{\alpha_k}{n}\sum_{i=1}^{n} y_i^k,$$

where at each iteration, a random index is selected and we set:

$$y_i^k = \begin{cases} f_i'(x^k) & \text{if } i = i_k, \\ y_i^{k-1} & \text{otherwise} \end{cases}$$

SAGA



SAGA Algorithm: Given the value of x^k and of each $f_i'(\phi_i^k)$ at the end of iteration k, the updates for iteration $k+1$ is as follows:

1. Pick a j uniformly at random.
2. Take $\phi_j^{k+1} = x^k$, and store $f_j'(\phi_j^{k+1})$ in the table. All other entries in the table remain unchanged. The quantity ϕ_j^{k+1} is not explicitly stored.
3. Update x using $f_j'(\phi_j^{k+1})$, $f_j'(\phi_j^k)$ and the table average:

$$w^{k+1} = x^k - \gamma \left[f_j'(\phi_j^{k+1}) - f_j'(\phi_j^k) + \frac{1}{n} \sum_{i=1}^{n} f_i'(\phi_i^k) \right], \quad (1)$$

$$x^{k+1} = \operatorname{prox}_\gamma^h \left(w^{k+1} \right). \quad (2)$$

The proximal operator we use above is defined as

$$\operatorname{prox}_\gamma^h (y) := \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \left\{ h(x) + \frac{1}{2\gamma} \|x - y\|^2 \right\}. \quad (3)$$

(Defazio et al., 2014 [2])

A variant of the SAG that supports l1-norm regularization penalty.

For both SAG and SAGA, training times were lower than others, further cementing their utility in large datasets.

Newton's Method

Newton's method is an iterative gradient descent approach for finding successively better approximations to the roots (or zeros) of a real-valued function. It uses the knowledge of a function's second derivative. The function is as below:

$$x_{n+1} = x_n - \frac{f(x)}{f'(x)}$$

Limited-Memory BFGS algorithm

Limited-Memory BFGS uses a similar approach compared to Newton's method, but uses less memory.

b. Results

The five methods produced nearly the same F_1 score which is around 0.84 with a difference of less than 0.01. Newton's F_1 score results in 0.01 point higher than the others.

Solvers	F_1 Score	Training Time	Testing Time
Large Linear Classification	0.8389	420.5 sec	0.06169 sec
Stochastic Average Gradient	0.8433	214.7 sec	0.04845 sec
SAGA	0.8433	558.2 sec	0.05196 sec
Newton's Method	0.8454	13831 sec	0.05228 sec
Limited-Memory BFGS	0.8432	476.9 sec	0.05127 sec

Newton's method confusion matrix:

Class	Precision	Recall	F_1score
0	0.80	0.81	0.81
1	0.98	0.95	0.97
2	0.73	0.74	0.73
3	0.83	0.87	0.85
4	0.74	0.77	0.75
5	0.94	0.92	0.93
6	0.63	0.57	0.60
7	0.91	0.94	0.92
8	0.93	0.94	0.94
9	0.95	0.94	0.95
Mean	0.85	0.85	0.85

Although there is only a slight difference in their F_1 scores, their training time performances varies significantly.

SAG has the best time performance, while Newton's Method took much longer to train (approximately 4 hours).

c. Analysis

F_1 score

Most classes' classification have obtained an F_1 score between 0.7 and 0.9, however, class 6 (shirt) performs comparatively much worse than the other data sets, which obtained a score of 0.6. Such phenomenon might be due to the class sharing much similarity with class 0 (t-shirt). The difference between the 2 classes are mainly presence of buttons and texture of clothes, which are hard to be recognized using a 28x28 picture.

The various solvers have comparable results. One reason could that with the use of grid search, we have optimized each solver to give their best results. Hence, no specific solver performs worse than the others.

Time performance

Although Newton's method performs better than other 4 solvers, it takes a significantly much more time compared to other models as well. This could be because it uses computationally expensive functions (Hessian Matrix) and second derivatives. Also, Newton's method has a hard time handling saddle points, resulting in slow convergence of functions. Meanwhile, SAG only takes 214.7 seconds to train, which can be explained by SAG's faster convergence rate and fewer iterations.

4. k-Nearest Neighbors

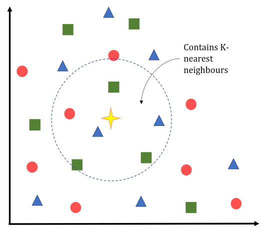
Motivation

k-Nearest Neighbors is the representative of the non-parametric algorithms for our study. Unlike the other models, there are no assumptions of the data and its relationship to the labels.

Description

There are no parameters to train in this algorithm, therefore its training time is distinctively short. However, due to the necessity to search for nearest neighbors for every classification, the algorithm has an exceptionally long testing time. The number of neighbors, k, controls model flexibility and can be adjusted to balance the bias-variance tradeoff.

There are 2 common data structures that can be used to help facilitate the finding of nearest neighbors: the Ball tree, and the k-d tree. We should not expect any difference in F_1 score when either data structures are used since how points are compared are the same in both algorithms. The only difference might be the training or testing time. Brute force search is also an option, possibly even outperforming these tree-based algorithms under certain conditions. Thus, there is usually a tradeoff when constructing these trees, where buckets of certain sizes are left at the leave nodes for brute force search to take over, instead of continuously constructing the tree until single element leaf nodes are left.



A distance metric is used to compare the similarity of each example in the test set and all examples in the training set are computed, after which, the test set is assigned the majority label of its k closest training set examples. A common distance metric to use is the Minkowski distance for continuous data and the Hamming distance for discrete data.

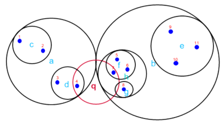
$$D\left(x_i, x_j\right) = \left(\sum_{i=1}^{n} \left|x_i - x_j\right|^p \right)^{1/p}$$

For our experiments, we will be using the Euclidean distance which is a special case of the Minkowski distance with degree 2.

a. Data Structures

Ball tree

A ball tree is a binary tree with a hierarchical structure, where two D-dimensional "hyperspheres", where points are recursively subdivided into two sub-clusters until a certain chosen depth. The time complexity to construct the tree is *O[D log(N)]* and to traverse it is *O[log(n)]*.



(Dolatshah et al., 2015 [3])

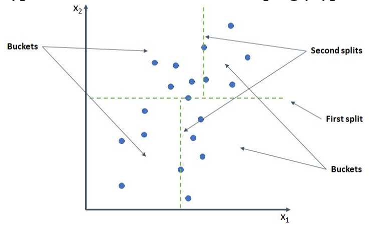
More specifically, each node in the tree defines the smallest ball that contains all data points in its subtree.

$$D^B(t) = \begin{cases} \max(|t - B.pivot| - B.radius, D^{B.parent}), & \text{if } B \neq Root \\ \max(|t - B.pivot| - B.radius, 0), & \text{if } B = Root \end{cases}$$

This gives rise to the useful property that, for a given test point t, the distance to any point in a ball B in the tree is greater than or equal to the distance from t to the ball.

k-d tree

The k-d tree uses the divide and conquer approach to search for the nearest neighbors. It is essentially an index tree that recursively partitions the sample space using a different unique variable at each split in the subtree, eventually ending with buckets of samples as the leaves. The complexity to construct the tree is *O[kn log(n)]* and to traverse it is *O[log(n)]*.



b. Results

Data Structures	F_1 Score	Training Time	Testing Time
Ball Tree	0.8546	20.48 sec	1045 sec
k-d Tree	0.8546	19.53 sec	1111 sec

Ball tree confusion matrix:

Class	Precision	Recall	F_1score
0	0.77	0.85	0.81
1	0.99	0.97	0.98
2	0.73	0.82	0.77
3	0.90	0.86	0.88
4	0.79	0.77	0.78
5	0.99	0.82	0.90
6	0.66	0.57	0.61
7	0.88	0.96	0.92
8	0.97	0.95	0.96
9	0.90	0.97	0.93
Mean	0.86	0.86	0.85

c. Analysis

Expectedly, the 2 data structures have identically performances. The F_1 score should not be affected by the way the data is stored and structured, and the training and testing time should not be too different given that both data structures have similar time complexities.

With an F_1 score of 0.8546, k-NN is evenly matched with LGR. It seems to face the same problems that LGR was facing, that the clothing that have similar sizes and shapes are misclassified often. This might be due to these similar clothing having very small distances from each other, thus overlapping each other very often.

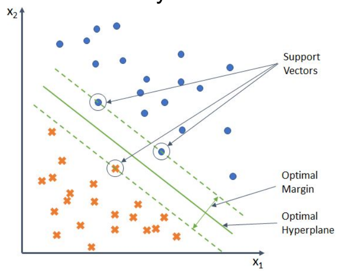
5. Support Vector Machines

Motivation

Support Vector Machines are a very interesting family of machine learning algorithms as they not only find a separating hyperplane for classification, but also maximizes the leeway for input noise.

Description

"The support-vector network implements the following idea: it maps the input vectors into some high dimensional feature space Z through some non-linear mapping chosen a priori. In this space, a linear decision surface is constructed with special properties that ensure high generalization ability of the network." [4]



After the input vectors have been mapped to a feature space such that it is separable by a hyperplane, the goal is to find a hyperplane, parameterized by weights θ and bias b, such that the distances between the points from each class closest to the hyperplane (ie. support vectors) are the largest.

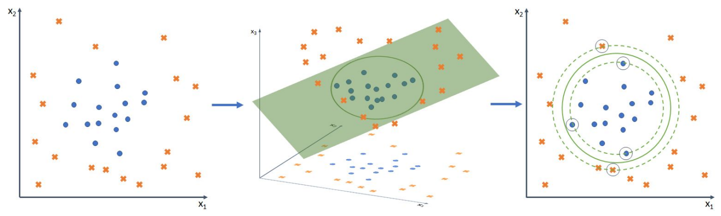
$$\theta, b = \arg\max_{\theta, b} \min_{x^{(j)} \in X} y^{(j)} \frac{\theta^{T} x^{(j)} + b}{\|\theta\|}$$

To improve generality, we will allow some points to be within the margin, while incurring some penalty. This soft-margin svm will therefore have the following training objective.

$$\theta, b = \arg\min_{\theta, b, \xi} \left(\lambda \|\theta\|^2 + \frac{1}{m} \sum_{j=1}^{m} \xi_j \right)$$

$$s.t.\ \forall j,\ y^{(j)} \left(\theta^{T} x^{(j)} + b \right) \geq 1 - \xi_j \text{ and } \xi_j \geq 0$$

Now, we are left with a final important decision to make - the choice of kernel to use that will make the input space into a linearly separable hyperspace. There are several options but in this study, we will only be comparing the sigmoid, polynomial, and radial-basis functions.



a. Kernels

Sigmoid

The sigmoid function was originally popular due to its roots from logistic regression and its widespread use in neural networks. The sigmoid is not a positive semi-definite (PSD) kernel. However, it is conditionally positive definite (CPD) under some constraints.

$$K\left(x_i, x_j\right) = \tanh\left(a x_i^{T} x_j + r\right)$$

Polynomial

A polynomial function is a simple choice for adding an additional dimension to the latent space. The degree of polynomial, d, can be adjusted to have a simple linear projection, or allow for curved surfaces. Increasing the degree allows for greater flexibility in the model but might introduce increased error from variance due to overfitting. This is a PSD kernel and therefore satisfies Mercer's theorem.

$$K\left(x_i, x_j\right) = \left(a x_i^{T} x_j + r\right)^{d}$$

Radial-basis function

The RBF kernel is a gaussian kernel smoothed by a radial (exponential) function. It essentially maps the distance between nodes to the new dimensional space while favoring small distances. Similar to the polynomial kernel, it is a PSD kernel and therefore satisfies Mercer's theorem.

$$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$$

d. Results

Kernels	**F_1 Score**	**Training Time**	**Testing Time**
Sigmoid	0.02000	1457 sec	5.742 sec
Polynomial	0.8434	148.7 sec	0.0881 sec
Radial-basis function	0.9038	968.3 sec	336.9 sec

Radial-basis function confusion matrix:

Class	Precision	Recall	F_1score
0	0.84	0.85	0.85
1	0.99	0.98	0.98
2	0.82	0.84	0.83
3	0.91	0.91	0.91
4	0.84	0.84	0.84
5	0.98	0.98	0.98
6	0.75	0.73	0.74
7	0.96	0.97	0.96
8	0.98	0.98	0.98
9	0.97	0.97	0.97
Mean	0.90	0.90	0.90

e. Analysis

The performance of this kernel was surprisingly underwhelming. The model was exceptionally difficult to tune and the results were abysmal. The F_1 score obtained was 0.02, suggesting that the model is very sensitive to tuning, and still poorly tuned despite numerous attempts to tune it. We suspect that the non-PSD characteristic of the kernel is the cause of this result.

The ideal degree for the polynomial kernel turns out to be 1 (ie. the linear kernel).

With an F_1 score of 0.9038, the RBF kernel not only outperforms all the other kernels significantly, it also outperforms all other traditional machine learning models we have implemented. This is perhaps due to the fact that there are pieces of clothing that have very similar pixel values and patterns and the RBF focuses more on the small differences.

6. Multilayer Perceptron

Motivation

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. They were popular in the 1980s [5], but computational difficulties relegated them to the backseat in favor of other simpler models. Within the last decade, interest in MLP returned as advances in hardware and algorithms sped up training by orders of magnitude.

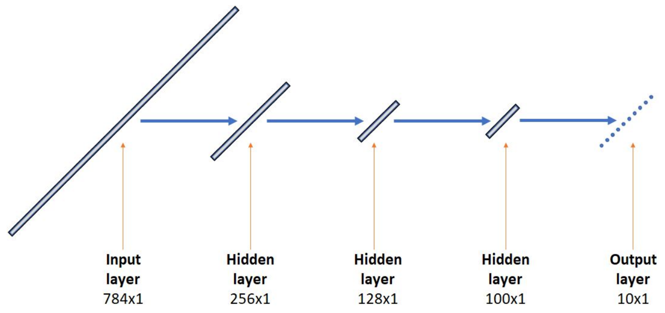
As the 'vanilla' neural network, we will be implementing a MLP as our first foray into deep learning on our image classification task. We will see just how much of an improvement deep learning can bring over non deep learning techniques.

Our MLP will also serve as a baseline for comparison with the other class of deep feedforward artificial neural networks, convolutional neural network, that is most commonly applied to analyzing visual imagery and image recognition systems.

We will also explore some augmentation of our images, introducing noise to our training data in the hopes that our network will generalize better.

a. Base model

Architecture



Characteristics

1. Densely connected layers
2. Input and hidden layer activation: Hyperbolic Tangent
3. Output activation: Sigmoid
4. Dropout
5. Stochastic Gradient Descent

Results

Our model managed to achieve an F_1 score of 0.8956, a sizeable lead over most of our previous traditional machine learning methods.

Class	Precision	Recall	F_1score
0	0.83	0.88	0.85
1	0.99	0.97	0.98
2	0.84	0.81	0.82
3	0.89	0.91	0.90
4	0.82	0.84	0.83
5	0.98	0.95	0.97
6	0.73	0.70	0.71
7	0.94	0.97	0.95
8	0.98	0.97	0.97
9	0.96	0.96	0.96
Mean	0.90	0.90	0.90

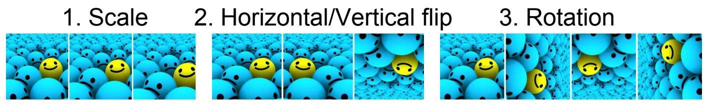
b. Image Augmentation

Description

The idea behind image augmentation is to introduce some noise to the training data to hopefully reduce overfitting and help our model generalize better. Image augmentation artificially expands the data-set, and is especially helpful when we have limited data.

Here, we augmented our training set randomly and added it to our original dataset for a combined total of 120,000 training images.

These are a few common ways to augment images.



https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced

Results

After augmenting our images with rotation and shifts, our model achieved an F_1 score of 0.8974.

Class	Precision	Recall	F_1score
0	0.85	0.84	0.85
1	0.99	0.98	0.98
2	0.84	0.81	0.82
3	0.89	0.92	0.90
4	0.82	0.83	0.83
5	0.98	0.96	0.97
6	0.72	0.74	0.73
7	0.94	0.96	0.95
8	0.98	0.97	0.98
9	0.96	0.96	0.96
Mean	0.90	0.90	0.90

c. Analysis

Image augmentation did not improve our model as much as we'd hoped for. There are a few reasons for this.

Our dataset is already 'clean' - all images are centered, converted to grayscale, and always in the same orientation. Our data has a normalized orientation, so rotations were not particularly helpful here. Shifts were not helpful as well, since our data is centered.

The takeaway here is that image augmentation is a powerful preprocessing step, but it depends on the problem and the dataset. It is not very helpful in every case.

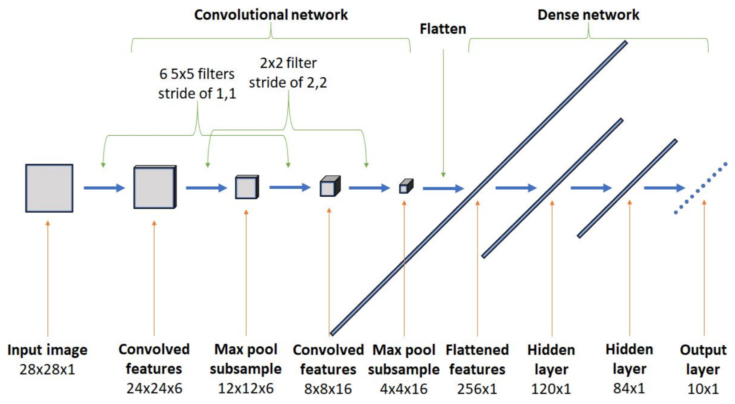
7. Convolutional Neural Network

Motivation

CNNs are specialized neural networks that uses spatially relative information to automatically engineer features that will then be used for prediction. This family of neural networks is naturally a perfect choice to further explore image recognition. We will implement LeNet-5, published by Yann LeCun back in 1998 pioneering CNNs, and use it as a springboard for our exploration into this area of study. We will then develop our own CNN, JoNet-0, with increased complexity and features that take advantage of our specific dataset to obtain a better F1-score.

a. LeNet-5

Architecture



Characteristics

1. Inner layers activation: Hyperbolic tangent (tanh)
$$g(z) = \tanh(z)$$
2. Output activation: Softmax
$$g(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \text{ for } j = 1, \ldots, K$$
3. Weight initialization: LeCun Uniform
$$\theta_k \sim U\left(-\frac{\sqrt{3}}{\sqrt{n_j}}, \frac{\sqrt{3}}{\sqrt{n_j}}\right)$$
4. Optimization algorithm: Mini-Batch Gradient Descent
$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta)$$

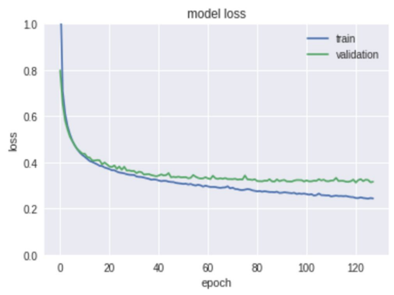
Description

LeNet-5 is a relatively small model with a mere 47,000 parameters (for our 28x28 images). It a simple model with only 2 convolution layers, each with a max pooling layer, followed by 2 fully connected layers. We will be implementing this exact architecture and will not be using any regularization methods like dropout and penalties. This will be our baseline model and we will be focusing on optimizing our custom CNN.

Results

With an F1-score of 0.8823, LeNet-5 performs decently, but fares slightly worse than Support Vector Machines. There are some problems that the model is facing that can be addressed to improve its performance.

Class	Precision	Recall	F_1score
0	0.84	0.86	0.85
1	0.99	0.97	0.98
2	0.78	0.81	0.79
3	0.88	0.91	0.90
4	0.79	0.78	0.79
5	0.98	0.96	0.97
6	0.71	0.65	0.68
7	0.93	0.97	0.95
8	0.96	0.96	0.96
9	0.97	0.95	0.96
Mean	0.88	0.88	0.88

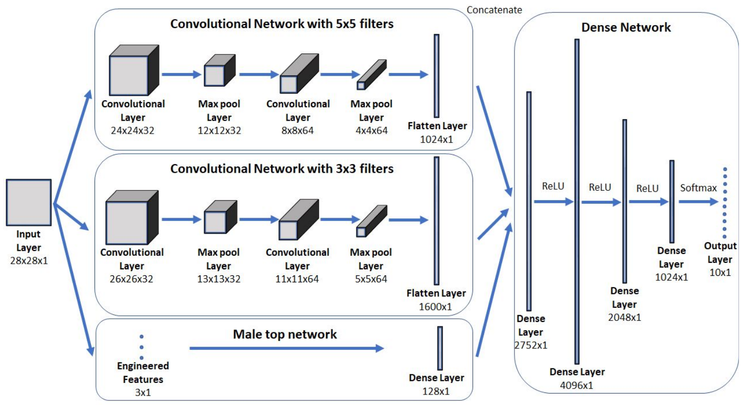


The slight divergence of the validation loss indicates that the model is overfitting the data slightly. This can be addressed by adding dropout layers and/or regularization penalties. The model also seems to be having a difficult time identifying classes 0, 2, 4, and 6, which correspond to *T-shirt/top*, *Pullover*, *Coat*, and *Shirt* respectively (corresponding to male tops). This is probably due to the similar shape of all of these pieces of clothing. If we could come up with a solution to help the model differentiate these pieces of clothing, the score should increase significantly.

With these lessons learnt, we shall now develop JoNet-0 to hopefully outperform LeNet-5.

b. JoNet-0

Architecture



Characteristics

1. Inner layers activation: Rectified linear unit (ReLU)
$$g(z) = \max(0, z)$$
2. Output activation: Softmax
3. Weight initialization: Glorot Uniform
$$\theta_j \sim U\left(-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right)$$
4. Optimization algorithm: Adaptive Moment Estimation (Adam)
$$m_t = \beta_1 m_{t-1} + \left(1 - \beta_1\right) g_t,\ \hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$v_t = \beta_2 v_{t-1} + \left(1 - \beta_2\right) g_t^2,\ \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$
$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t$$

Description

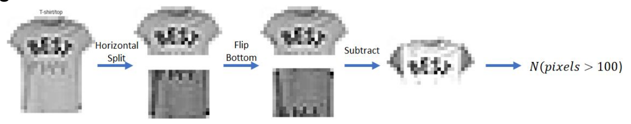
First, we dramatically increased the size of the network, requiring over 20 million parameters to be tuned. We also replace old techniques with more modern ones: ReLU activation - to deal with vanishing gradients in the significantly larger network, Glorot uniform initializer - to approximately maintain activation variances and back-propagated gradients variance as one moves up or down the network [6], and Adam optimizer - an empirically better optimizer that combines momentum and Root Mean Square Propagation [7].

Next, to deal with overfitting, we will add a dropout layer as well as an l2-norm regularization penalty to each convolutional and dense inner layers.

Finally, in an attempt to tackle the problem of differentiating between male tops, we added 2 networks to run in parallel to the original convolutional network with 5x5 filters: An additional convolutional network with 3x3 filters in the hopes that it picks up on the print patterns on the t-shirts and coats, and a “Male top network” which takes 3 engineered features as inputs that will hopefully help the system differentiate between the male tops.

Male top network

The first input is a measure of how horizontally symmetrical the image is. We will be using the simple method for calculating this measure shown below. The idea is that some tops are more horizontally symmetrical than others, mainly due to the sleeve length and the existence of collars or hoods.



The second input is a measure of the intensity of the color of the top. We have observed that *Coat*s and *Pullover*s are generally darker than the other tops. This is done by simply averaging the pixel value of the image.

The third input is a measure of the size of the top. We have observed that *Coat*s and *Pullover*s tend to be larger than *T-shirt/top*s. This is done by counting the number of pixels with a value greater than 100.

This gives us 3 inputs per image that we will feed into a dense layer for learning to take place.

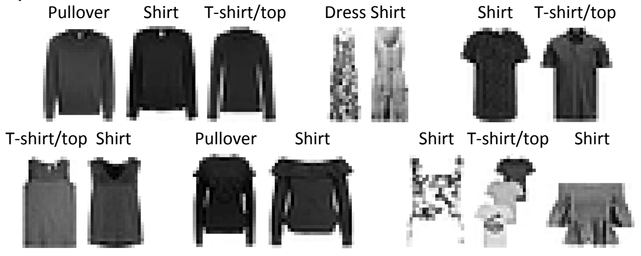
Results

With an F1-score of 0.9230, JoNet-0 manages to not only perform better than LeNet-5, but better than any other model we have implemented.

Class	Precision	Recall	F_1score
0	0.87	0.88	0.88
1	1.00	0.98	0.99
2	0.87	0.88	0.88
3	0.94	0.93	0.93
4	0.87	0.88	0.88
5	0.98	0.99	0.99
6	0.79	0.88	0.77
7	0.96	0.98	0.97
8	0.98	0.98	0.98
9	0.98	0.96	0.97
Mean	0.92	0.92	0.92

c. Analysis

The F_1 scores of the classes 0, 2, 4, and 6 (male tops) have significantly improved. The largest increase being class 6 (*Shirt*) from 0.68 to 0.77 - a good sign that our model is more capable in differentiating between the male tops. There is still much room for improvement when it comes to classifying the male tops, however, based on our observations of these images, it is clear why the model struggles to classify them perfectly. Many of the classes have images that are very similar to each other, while some could be questioned whether to even be included in the dataset at all.



Ultimately, like all neural networks, it is difficult to pinpoint exactly which part of the neural network is actually helping to classify these difficult classes. Perhaps, it is simply the significantly increased scale of the network that is the main cause of the improvement.

8. Model Comparison

a. Summary

Model	F_1 Score	$F_{0.5}$ Score	F_2 Score	Training Time	Testing Time
Logistic Regression (Newton’s Method)	0.8454	0.8451	0.8458	13831 sec	**0.05228 sec**
k-Nearest Neighbours (Ball tree)	0.8546	0.8561	0.8565	**21.62 sec**	1017 sec
Support Vector Machines (RBF)	0.9038	0.9038	0.9039	968.3 sec	336.9 sec
Multilayer Perceptron (with Image Augmentation)	0.8974	0.8967	0.8966	3800 sec	0.6800 sec
Convolutional Neural Network (JoNet-0)	**0.9230**	**0.9231**	**0.9231**	59200 sec	34.00 sec

b. Scenarios

F_β score

Different scenarios require different emphasis on precision and recall. In the context of security, precision might be of higher importance, as false positives are costly due to overhead in deploying security personnel. On the other hand, in the context of fashion, recall might be more important as a large breadth in types of fashion items detected is desirable.

The $F_{0.5}$ scores and F_2 scores are generally very similar to the F_1 score. This is unsurprising. Looking at the confusion matrices of all the models, the precision and recall are usually very similar, so adding a weight to either would not change the outcome much. Therefore, we should simply pick the best performing model, JoNet-0, regardless of the emphasis on either precision or recall.

Training and Testing Time

Two other important metrics to compare our models is the training and testing time. Some applications of machine learning models require either real-time training (eg. online learning), or instantaneous predictions of large amounts of examples. In the former case, k-Nearest Neighbors might be the ideal model as it has the characteristically short training time. For the latter, logistic regression might be the ideal model as it is able to predict thousands of examples in mere milliseconds.

Support Vector Machine seems to be the best generalist and could be the ideal model if all metrics are of equal importance.

9. Conclusion

Having successfully implemented all of the models that we hoped to, we have obtained greater insights into the implementation details of each algorithm. We have also successfully developed an interesting new model that makes use of the specificity of the data to improve the performance. It would have been naive to think there would be a single best model that would perform the best in all scenarios. It is therefore great to have a guide for which models might be most suitable in certain cases.

References

[1] LeCun, Bottou, Bengio, Haffner (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*.

[2] Defazio, Bach, Lacoste-Julien (2014). SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives. arXiv:1407.0202.

[3] Dolatshah, Hadian, Minaei-Bidgoli (2015). Ball*-tree: Efficient spatial indexing for constrained nearest-neighbor search in metric spaces.*arXiv:1511.00628v1*.

[4] Cortes, Vapnik (1995). Support-Vector Networks. *Machine learning*.

[5] LeCun, Bottou, Orr, Müller (1998). Efficient BackProp. *Springer, Berlin, Heidelberg*.

[6] Glorot, Bengio (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*.

[7] Ruder (2016). An overview of gradient descent optimization algorithms. *arXiv:1609.04747*

He, Zhang, Ren, Sun (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. International Conference on Computer Vision (ICCV), 2015.

Lin, Lin (2003). A Study on Sigmoid Kernels for SVM and the Training of non-PSD Kernels by SMO-type Methods. *submitted to Neural Computation***.**

Liu, T.; Moore, A. & Gray, A. (2006). New Algorithms for Efficient High-Dimensional Nonparametric Classification. Journal of Machine Learning Research. 7: 1135–1158.