ECEN 758 Data Mining and Analysis Project

Yu-Sheng Chen
MS in Data Science
Texas A&M Univ.
Tx, U.S.A
chenyus0609@tamu.edu

Emily Chang
Dept. Elec, & Comp. Eng
Texas A&M Univ.
Tx, U.S.A
emilyiyc@tamu.edu

Jen Li Kao
Dept. Elec, & Comp. Eng.
Texas A&M Univ.
Tx, U.S.A
smallikao@tamu.edu

***Abstract*— This project compares two popular machine learning models - Deep Convolutional Neural Networks (CNNs) and Light Gradient Boosting Machine (LightGBM) - and their effectiveness in image classification using the Fashion-MNIST dataset. The main goal is to identify the strengths and weaknesses of each model and provide valuable insights for selecting the appropriate model for similar classification tasks. The project begins with an overview of the importance of model selection for successful machine-learning tasks. Our study found that CNNs are more effective in learning complex patterns in image data due to their multi-layered architecture. They outperformed LightGBM in classifying fashion items by approximately 3.35%. However, LightGBM provides a fast random forest framework for various data types. In our project, it was 730 seconds faster than CNN.**

Please find below the link to our code and website:
Code:https://colab.research.google.com/drive/1Y6aRmoUxRn_qqbfPkR97pJBtCZDRt1RZ?usp=sharing
Website: https://medium.com/@emilyjyc/comparison-of-cnn-and-light-gbm-with-fashion-mnist-77cd1644810e

Keywords—Fashion-MNIST, t-SNE, Deep CNNs, LightGBM, image classification

I. INTRODUCTION

Choosing the right model is vital to achieving optimal results in machine learning (ML). Our project is focused on comparing two distinct models, the Deep Convolutional Neural Networks (CNNs) and the Light Gradient Boosting Machine (LightGBM), to determine their effectiveness in image classification using the Fashion-MNIST dataset.

Therefore, through the comparative analysis of these models, we aim to contribute to the broader understanding of model selection in image classification tasks, shedding light on the nuanced interplay between model architecture and dataset characteristics. This exploration advances theoretical knowledge in the field and provides practical insights for informed model choices in real-world applications.

The structure of this report is as follows. In Section II, the implementation of data preparation and model selection are described. In Section III, the experimental results of each model are shown, and the conclusion is drawn in Section IV.

II. METHOD

A. Data Preparation and Analysis

We recognized the importance of data preparation for our project and took measures to ensure the quality and relevance of our dataset. Our analysis included scrutinizing data distributions, computing descriptive statistics, and using visualization techniques to understand our dataset's structure better.

(a) Data preparation

First, we import essential libraries for data processing and machine learning experiments: NumPy, Matplotlib, Seaborn, TensorFlow, Keras, and sci-kit-learn.

Subsequently, the Fashion-MNIST dataset is loaded, a collection of image data associated with various clothing categories. Leveraging the "fashion_mnist.load_data()" function, the training and testing are treated as grayscale, each with dimensions of 28x28 pixels.

In the data cleansing and transformation phase, the training and testing images undergo reshaping to ensure a single-channel representation (grayscale). Furthermore, a normalization step is implemented, scaling pixel values to the range [0, 1]. This normalization is pivotal for maintaining numerical stability during subsequent neural network training.

Finally, the dataset is partitioned into training and validation subsets, achieved through the "train_test_split" function from sci-kit-learn. Notably, 30% of the data is allocated for validation purposes.

(b) Data analysis

In exploring the Fashion-MNIST dataset, we meticulously conducted a multi-faceted data analysis to unravel its intricacies and characteristics.

First, we employed descriptive statistics to quantify the distribution of various fashion classes across the training, validation, and testing sets. This approach quantitatively characterizes the dataset by providing class-specific sample counts, illustrated in Fig 1.

Training Set:	Validation Set:	Testing Set:
[0]T-shirt/top: 4231 samples	[0]T-shirt/top: 1769 samples	[0]T-shirt/top: 1000 samples
[1]Trouser: 4165 samples	[1]Trouser: 1835 samples	[1]Trouser: 1000 samples
[2]Pullover: 4199 samples	[2]Pullover: 1801 samples	[2]Pullover: 1000 samples
[3]Dress: 4211 samples	[3]Dress: 1789 samples	[3]Dress: 1000 samples
[4]Coat: 4185 samples	[4]Coat: 1815 samples	[4]Coat: 1000 samples
[5]Sandal: 4217 samples	[5]Sandal: 1783 samples	[5]Sandal: 1000 samples
[6]Shirt: 4189 samples	[6]Shirt: 1811 samples	[6]Shirt: 1000 samples
[7]Sneaker: 4241 samples	[7]Sneaker: 1759 samples	[7]Sneaker: 1000 samples
[8]Bag: 4175 samples	[8]Bag: 1825 samples	[8]Bag: 1000 samples
[9]Ankle boot: 4187 samples	[9]Ankle boot: 1813 samples	[9]Ankle boot: 1000 samples

Fig 1. Statics of different datasets

Subsequently, we created a series of image plots from various classes in the training. Fig 2 displays a grid of 25 images labeled with its corresponding class name. This visual representation of the dataset provides a qualitative exploration,

offering insight into its diverse range of clothing items. The grid presents a snapshot of the dataset, making it easy to comprehend the variety of clothing items included.

Fig 2. Example images from classes

Additionally, we used a data analysis and dimensionality reduction technique called t-SNE (t-distributed stochastic neighbor embedding) with Python in sci-kit-learn[3]. We flattened the images and then applied t-SNE to reduce them to two dimensions. t-SNE will convert the distance between data in the original space to probabilities. To begin with, the process involves the calculation of conditional probabilities:

$$p_{j|i} = \frac{\exp\left(-\frac{d(x_i, x_j)}{2{\sigma_i}^2}\right)}{\sum_{i \neq k} \exp\left(-\frac{d(x_i, x_j)}{2{\sigma_i}^2}\right)}, p_{i|i} = 0 \quad (1)$$

which will be used to generate joint probabilities:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N} \quad (2)$$

The σ_i will be automatically determined. A heavy-tailed distribution will be used to measure similarities in the embedded space:

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l}(1 + \|y_k - y_l\|^2)^{-1}} \quad (3)$$

and then minimize the Kullback-Leibler divergence:

$$KL(P|Q) = \sum_{i \neq j} p_{ij} log \frac{p_{ij}}{q_{ij}} \quad (4)$$

between both distributions with gradient descent.[4]

This produced a visual representation of the dataset, as seen in Fig 3. Each point in the plot corresponds to an image, and its color tells us about its class label. The plot clearly distinguishes between Bag items and footwear, with the footwear items (Sneakers, Sandals, and Boots) being closer together. In addition, we can see that Trousers, Dresses, and T-shirts, among other clothing items, are clustered together. This technique helped identify potential patterns in the dataset and provided valuable insights into its structure.

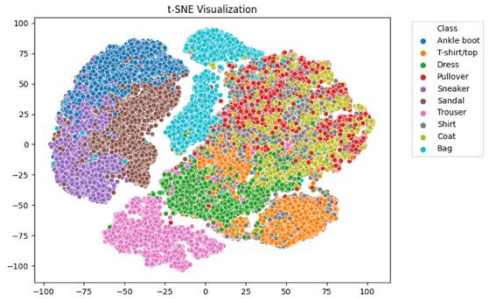

Fig 3. Dimensionality reduction visualization – t-SNE

B. *Model Selection*

Deep Convolutional Neural Networks (CNNs)[1] are highly effective tools for image-related tasks, leveraging their ability to learn hierarchical representations from raw pixel values automatically. In the context of image classification, CNNs have achieved remarkable success by capturing intricate features and patterns within images. Their ability to extract and analyze complex features makes them ideal for image-related tasks.

In contrast to Deep Convolutional Neural Networks (Deep CNNs), Light Gradient Boosting Machine (LightGBM)[2] is a gradient boosting framework that is particularly effective in managing tabular data. It has become increasingly popular because of its efficiency, scalability, and ability to handle massive data sets. LightGBM sequentially constructs decision trees, with each subsequent tree correcting the errors of the previous ones.

(a) Deep learning

Convolution Neural Network(CNN) is a concept that combines Neural Network and Convolution Layer Network. Neural networks consist of layers of interconnected nodes (neurons) to simulate the neurons in the human brain.

The typical concept of CNN is to modify an image to another shape. If we consider a 2-dimensional image *I* as our input, it is advisable to employ a 2-dimensional kernel *K*. The classical books about deep learning [1] purposes that:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \quad (5)$$

S(i,j) is given by the convoluted image which has numbers of height *m* and numbers of width *n*. The values *i* and *j* present the index that the kernel *K* has processed to the original image.

First, the initialization builds a 2D convolution over an input signal made up of multiple input planes. The output value of the layer with input size(N, C_{in}, H, W) and output $(N, C_{out}, H_{out}, W_{out})$ can be precisely defined as a formula[5]:

$$out\left(N_{in}, C_{out_j}\right) = bias\left(C_{out_j}\right) + \sum_{k=0}^{C_{in}-1} weight\left(C_{out_j}, k\right) * input(N_i, k) \quad (6)$$

Where * is the valid 2D cross-correlation operator, *C* is a batch size, denotes the channels used in hidden layers, $\boldsymbol{H}$ is the height of input planes in pixels, and $\boldsymbol{W}$ is width in pixels.

In this model, it is recommended to use 2 convolutional mapping layers processed by ReLU for solving the gradient issue[6] and max pooling layers before linear neuron connection as suggested by[7]. The first convolutional layer has 1 input and 16 output with kernel filter size 3×3 and the second one has 32 input and 64 output filters of size 3×3, and max pooling was performed on every 2×2 pixels for extracting the strong feature value presented in a specific region. The output of this was flattened and fed into a multi-layer connected network for classification. Therefore, our CNN model performs like Fig 4:

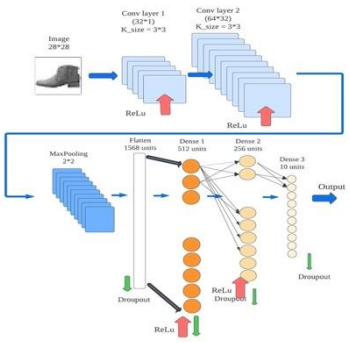

Fig 4. The scheme diagram of model CNN

Additionally, we use dropout rate 0.2 during training for regularization and preventing the co-adaptation of neurons given by [8].

Finally, one of our significant methodology beside model build used in training is parameter normalization provided by [9]:

Algorithm 1: Weight Parameter Normalization

Input: pixel values

Gain the class string name ***m***

If *the class name 'Linear' is not out of range:*

    n ← a value of feature in ***m***

    y ←1.0/sqrt(n)

    Constrain the value ***y*** in (0, y)

    bias ← 0

end if

Therefore, if the function detects pixel value is not equal to -1, the value will be normalized in 1.0/sqrt(n). It means that if n represents the number of nodes in a Linear Layer, then weights are given as a sample of normal distribution in the range (0, y). Here y represents standard deviation calculated as y=1.0/sqrt(n)

(b) LightGBM

LightGBM is a kind of Gradient Boosting Decision Trees (GBDT), which are widely-used in today's ML applications. However, GBDT faces challenges of the explosion of data quantities nowadays. To deal with this, LightGBM proposes 2 techniques: Gradient-based one-sided sampling (GOSS) and Exclusive Feature Bundling (EFB).

The basic concept for GOSS is to choose instances that contribute more to the information gain (having larger gradients). On the other hand, randomly drop those instances with small gradients. This is a way to preserve the information gain as much as possible while lessening the computational demands. The algorithm of GOSS is given as follows:

Algorithm 2: Gradient-based One-Side Sampling

Input: I: training data, d: iterations
Input: a: sampling ratio of large gradient data
Input:b: sampling ratio of small gradient data
Input: loss: loss function, L: weak learner
models ← {}, fact ← $\frac{1-a}{b}$
topN ← a×len(I) , randN ← b×len(I)

for *i = 1* ***to*** *d* **do**

    preds ← models.predict(I)

    g ← loss(I, preds), w ← {1,1,...}

    topSet ← sorted[1:topN]

    randSet ← RandomPick(sorted[topN:len(I)], randN)

    usedSet ← topSet + randSet

    w[randSet] × = fact ▷ Assign weight f act to the small gradient data

    newModel ← L(I[usedSet], − g[usedSet], w[usedSet])

    models.append(newModel)

EFB is a technique to "bundle" mutually exclusive variables, such that we can reduce the effective features, thus a way to counter the Curse of dimensionality. Here, mutually exclusive means that 2 features rarely take nonzero values simultaneously. The "bundle" is like merging two variables together as one, e.g. (1,0,0,2) can be bundled with (0,9,9,0) as (1,9,9,2), notice that this is just an instance, we shall expand this concept to variables. An efficient greedy algorithm is proposed:

Algorithm 3: Greedy Bundling

Input: F: features, K: max conflict count Construct graph G searchOrder ← G.sortByDegree() bundles ← {}, bundlesConflict ← {}

for *i* ***in*** *searchOrder* **do**

    needNew ← *True*

    for *j = 1* ***to*** *len(bundles)* **do**

        cnt ← ConflictCnt(bundles[j],F[i])

        if *cnt + bundlesConflict[i]* $\leq K$ **then**

            bundles[j].add(F[i]), needNew ← False

            break

    if *needNew* **then**

        Add *F[i]* as a new bundle to *bundles*

Output: *bundles*

(c) Early-Stopping

Early stopping is a crucial technique in model training that helps to improve efficiency and prevent overfitting. Its primary purpose is to monitor a model's performance on a validation dataset and stop training when further iterations are unlikely to improve performance. This technique helps to prevent overfitting, optimizes computational resources, accelerates training time, and promotes the development of robust models capable of generalizing well to new and unseen data.

Our project employs early stopping in both Deep CNNs and LightGBM models to balance high accuracy and effectiveness on new data. This method ensures our models are complex yet generalizable, enhancing accuracy while preventing overfitting.

(d) Feature Importance

The feature importance we use here is provided by the LightGBM package. It is calculated related to the loss improvement gain from splitting a node using a certain feature.

III. EXPERIMENTS

Hypothesis: For image classification, Deep CNNs are expected to outperform LightGBM in accuracy due to their specialized image processing capabilities. However, LightGBM is likely to be faster in training and prediction, benefiting from its efficiency with large datasets.

A. Model Building

(a) Deep learning

The training elements of CNN are built within:

- Criterion: CrossEntropy
- Optimizer: Adam
- Learning rate: 0.001
- Early-stopping rounds: 5

(b) LightGBM

A simple GBDT-based classifier is built.
Using hyper-parameters:

- Number of leaves in full tree: 31
- Learning rate: 0.05
- Part of feature to be used for each iteration: 0.9
- Part of data to be used for each iteration: 0.8
- Early-stopping round: 5

Note that due to resource and time limits, hyperparameter tuning methods such as GridSearch and RandomSearch were not applied.

B. Model Evaluation

Table 1. Performance Comparison between CNN and LightGBM

	Deep CNN	LightGBM
Testing Accuracy	92.25 %	89.26 %
Training time	1176 sec	446 sec

Table 1 displays that the distinguishing accuracy of CNN is better than LightGBM, but it obviously takes more time during training.

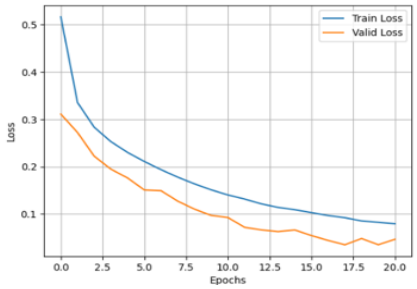

Fig 5. Training and validation loss curve - CNN

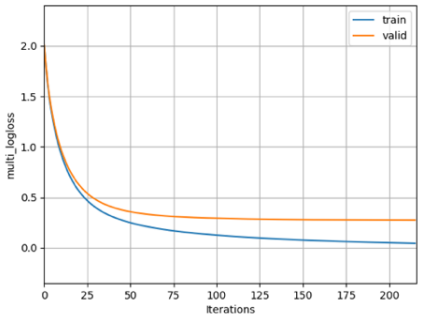

Fig 6. Training and validation loss curve – LightGBM

Classification report for CNN :

	precision	recall	f1-score	support
0	0.83	0.90	0.86	1000
1	1.00	0.97	0.99	1000
2	0.84	0.87	0.86	1000
3	0.90	0.92	0.91	1000
4	0.86	0.86	0.86	1000
5	0.98	0.99	0.98	1000
6	0.81	0.70	0.75	1000
7	0.96	0.97	0.97	1000
8	0.98	0.98	0.98	1000
9	0.97	0.96	0.97	1000
accuracy			0.91	10000
macro avg	0.91	0.91	0.91	10000
weighted avg	0.91	0.91	0.91	10000

Fig 7. Classification report (Test) - CNN

Classification Report for LightGBM:

	precision	recall	f1-score	support
0	0.83	0.88	0.85	1000
1	1.00	0.96	0.98	1000
2	0.80	0.82	0.81	1000
3	0.90	0.90	0.90	1000
4	0.81	0.83	0.82	1000
5	0.99	0.97	0.98	1000
6	0.73	0.66	0.69	1000
7	0.94	0.97	0.95	1000
8	0.97	0.97	0.97	1000
9	0.96	0.95	0.96	1000
accuracy			0.89	10000
macro avg	0.89	0.89	0.89	10000
weighted avg	0.89	0.89	0.89	10000

Fig 8. Classification report (Test) - LightGBM

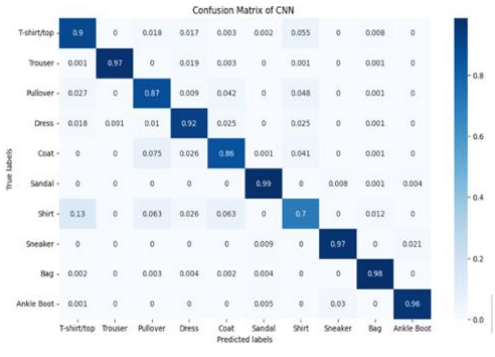

Fig 9. Confusion matrix (Test) - CNN

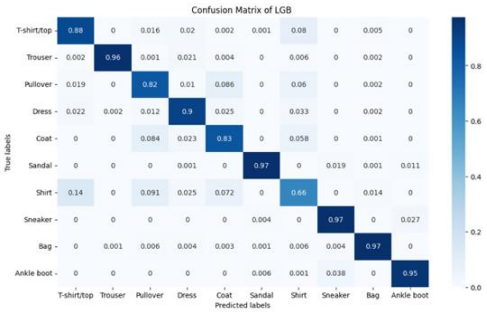

Fig 10. Confusion matrix (Test) - LightGBM

As above 2 confusion matrices Fig 9 and Fig 10 , we can observe that both models struggle to distinguish "Shirt" (label 6) the most, followed by "Coat" (label 4) and "Pullover" (label 2). This is reasonable due to the similar shapes of these clothing items and the waist width of label 6 being mistaken for label 4 or label 2.

C. Feature Importance

Using the feature importance method provided by LightGBM, we can have the features with top 15 Gain:

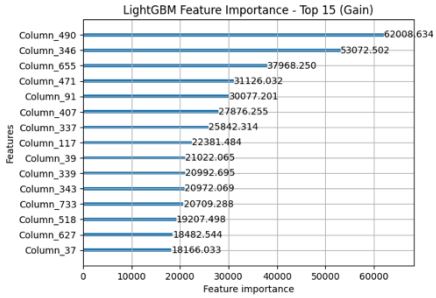

Fig 11. Feature importance of LightGBM

Fig 12. Example images of Top-10 important features

As shown in Fig 11, the LightGBM classifier suggests that pixels 490, 346, 655, 471, 91, 407, 337, 117, 39, 339 are the "most important" features when doing classification. (denoted as columns since they are flattened images). Fig 12 shows the first 25 images in the training Dataset with the top 10 important pixels indicated. Observe that these pixels can help distinguish shoes (sandals, sneakers, etc.) from clothes (pullover, shirt, etc.), thus it is reasonable that these pixels are the ones that have the largest gain.

IV. Conclusion

In conclusion, we found a delicate balance between model accuracy and computational efficiency after conducting a comparative analysis of Deep CNNs and LightGBM on the Fashion-MNIST dataset. Our results indicated that CNNs, with their multi-layered architecture, can learn hierarchical representations of features, making them ideal for tasks that involve complex patterns. They demonstrated proficiency in classifying fashion items, with an accuracy of approximately 3.35% better than LightGBM, showcasing their ability to capture intricate patterns in image data. Conversely, LightGBM adopts a GBDT framework to provide a boosting of decision trees, enabling it to handle various data types with speed advantages efficiently. In this project, it was 730 seconds faster than CNN.

The business insight of these two models is clear. For optimizing the online catalog, CNNs excel in image classification and tagging, enhancing the user experience by precisely organizing and presenting products. In scenarios requiring rapid processing of a large volume of product images, LightGBM's efficiency becomes crucial, especially for real-time tasks like flash sales or inventory updates, contributing to agile and responsive marketing strategies.

Understanding model capabilities is crucial for informed decisions. In the future, developing hybrid models that combine the strengths of CNNs and LightGBM could be a promising avenue for further research and improvement.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[2] G. Ke *et al.*, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree", Accessed: Nov. 21, 2023. [Online]. Available: https://github.com/Microsoft/LightGBM.

[3] "sklearn.manifold.TSNE — scikit-learn 1.3.2 documentation." Accessed: Nov. 20, 2023. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

[4] L. Van Der Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[5] "Conv2d — PyTorch 2.1 documentation." Accessed: Nov. 21, 2023. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

[6] "ReLU — PyTorch 2.1 documentation." Accessed: Nov. 21, 2023. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html

[7] W. Hu, Y. Huang, L. Wei, F. Zhang, and H. Li, "Deep convolutional neural networks for hyperspectral image classification," *J Sens*, vol. 2015, 2015, doi: 10.1155/2015/258619.

[8] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors".

[9] "Fashion MNIST | Machine Learning Master." Accessed: Nov. 21, 2023. [Online]. Available: https://nvsyashwanth.github.io/machinelearningmaster/fashion-mnist/