Classifying Images of Fashion Items using Deep CNN

Cheng Xi Tsou, Ningyu Chen, and Tim Betancur

1 Introduction

In 2020, around 30% of the total fashion retail sales in the US were through e-commerce. Countless new products were introduced during each season of the year. Listing new products into respective categories by human power could be time consuming and error-prone. We decided to solve this problem by building a classifier through a deep convolutional neural network that can efficiently and accurately classify images of fashion items. Our final model is able to classify images of fashion items with a 0.923 accuracy into 8 article types.

2 Data

The data we used to train our network were 80 by 60 RGB images from Kaggle[1]. The dataset came with 44,000 images labeled with 136 different article types and other basic descriptors of each image. After examining our dataset, we found an extreme imbalance between the classes, so we picked the classes that had at least 1700 samples and capped each class at 2500 samples, which resulted in 8 classes. We ended up with about 17,000 images with a 80-10-10 split between training, validation, and testing. Initially, we resized the images to 28 by 28 on grayscale but later we procured two more sets of data with dimensions 28x28 RGB and 80x60 RGB. For data preprocessing, we scaled each data value to be in the range of [0, 1] and changed our labels into categorical one-hot encoding.

Fig. 1 A 80x60 sample image

3 Experiments and Results

We started our experiments with a baseline model from Keras' Simple MNIST convnet[2] implementation as a reference and progressively introduced more complexity. This allowed us to learn whether a certain hyperparameter tuning was effective. The baseline model was a simple CNN with the following structure: (3, 3, 16) Conv, (2, 2) MaxPool, (3, 3, 64) Conv, (2, 2) MaxPool, (8) Dense. This model had a reasonable performance achieving 0.835 accuracy on the test set. The results indicated which classes were harder to classify and what the model was lacking.

Table 1 Evaluation of baseline model on testing data

Class name	f1-score	support
Sports Shoes	0.807107	203
Kurtas	0.885714	244
Tops	0.933687	188
Handbags	0.771144	194
Watches	0.993603	235
Shirts	0.854167	233
Tshirts	0.842520	263
Casual Shoes	0.983333	180
Accuracy	0.882759	N/A
Macro avg	0.883909	1740
Weighted avg	0.882869	1740

3.1 Experiment 1

In this experiment, we continuously added one layer, a Conv layer or dense layer, to our baseline model until the accuracy in the validation set did not improve anymore. Each succeeding model has one more layer than the previous. The images had low resolutions(28 x 28), so we did not introduce more pooling layers. We expanded the two stacks of Conv. layers by duplicating existing Conv layers. We added paddings to each Conv layer to maintain the input dimensions. It helped to elongate the network, which enabled us to add more Conv layers. We expanded the stacks of Conv layers and dense layers alternatively. We added more Conv layers when the model needed to capture more captures. We added more dense layers when the model needed more parameters to enhance its classification power. We stopped when the model was no longer improving and its computational time was too long. The validation accuracy of each model was compared, and we selected the model, exp1_10, with the highest validation accuracy. Model exp1_10 had 6 Conv layers equally divided between the stacks and three dense layers.

Table 2 The image above displays the performance of each model.

Model name	Accuracy	Validation Accuracy	Acc Diff
exp1_10	0.944862	0.900383	0.044
exp1_5	0.938192	0.897190	0.041
exp1_7	0.949262	0.896552	0.053
exp1_11	0.938689	0.895913	0.043
exp1_6	0.934644	0.894636	0.040
exp1_4	0.928683	0.893997	0.035
exp1_8	0.956713	0.892720	0.064
exp1_9	0.934076	0.885696	0.048
exp1_1	0.920380	0.884419	0.036
exp1_3	0.927689	0.884419	0.043
exp1_2	0.933579	0.883780	0.050

Model exp1_5 underperformed by around 0.3%. Its architecture was less complex than model exp_1_10; it was nearly two times faster. However, our goal for this experiment was to choose the model with the best generalization ability. Then we would apply simplification and regularization to the selected model in the later

experiments.

3.2 Experiment 2

In this experiment, we tried to improve the performance of model exp1_10 by increasing the deeper Conv layers' kernel sizes. The convention of building a CNN architecture is to gradually increase the kernel size to reduce the feature space width to capture high-level information. The pattern of kernel sizes used for the second stack of Conv layers are listed as follow: (5x5) (5x5) (5x5), (3x3) (3x3) (5x5), (3x3)(5x5)(3x3), (5x5)(3x3)(3x3), and substituted (7x7) for (5x5). There are 8 different patterns and thus 8 models. According to the results, these alterations did not improve the generalization ability of model exp1_10, but in fact, they caused more overfitting on the testing set. Generally, larger kernel size should combat overfitting as it helps to capture generic features. However, the output dimensions of the first max pooling layer was (13x13). Therefore, we were unknowingly converting Conv layers to fully connected layers by using relatively big kernel sizes, which undermined the generalization ability of CNN and thus caused overfitting. As a result, we will proceed with exp1_10 as our current best model.

Table 3 The table displays the performance of each model. Please note: a model with higher label number is more complex due to layer additions

Model name	Accuracy	Validation Accuracy	Acc Diff
exp2_10_6	0.959	0.898	0.061
exp2_10_1	0.953	0.897	0.056
exp2_10_5	0.948	0.895	0.053
exp2_10_7	0.941	0.893	0.047
exp2_10_8	0.960	0.892	0.068
exp2_10_4	0.958	0.891	0.068
exp2_10_9	0.927	0.887	0.040

3.3 Experiment 3

In this experiment, the goal was to combat our issue of overfitting. The best model we have so far yielded a training accuracy of 0.944 and a validation accuracy of 0.9. Our goal is to reduce the difference between the two so that the model is less overfit on the data we have trained it on and able to be generalized for new inputs. The strategies that we have tried to employ are adding Dropout layers in the Dense layers, adding Dropout layers with low dropout rates in the convolutional layers, adding L2 regularization to the Dense layer's kernel weights, and data augmentation.

Our first model exp3_10_2 had two Dropout layers with dropout rate at 0.3 in the Dense layer, then we added Dropout layers with dropout rates at 0.2 to our convolutional layers for our model exp3_10_3. The results showed that both approaches still resulted in slight overfitting. After tuning the dropout rates in all layers, the results favored the model exp3_10_1 with two dropout layers with dropout rate of 0.5 in the dense layers. We hypothesize that this was because our image contained important low level features so any kind of dropout will cause the model to be less generalizable. We also tested a model exp3_10_4 with data augmentation with horizontal flips, rotations, and random offsets but the results showed a drop in training and validation accuracy.

Finally, we decided to try to simplify the model as we thought that the excessive number of layers caused our model not to generalize well. After removing a convolutional layer from each stack of convolutional layers and having two Dropout layers with dropout rates of 0.5 in the dense layer, we were able to finalize a model exp3_10_8 that had a training accuracy of 0.9145 and validation accuracy of 0.8978. Compared to other models in this experiment, exp3_10_8 converged much faster so it was less prone to overfitting. Comparing this exp3_10_8 with our best model from experiment exp1_10, the two models had a similar validation accuracy but exp3_10_8 is better generalized.

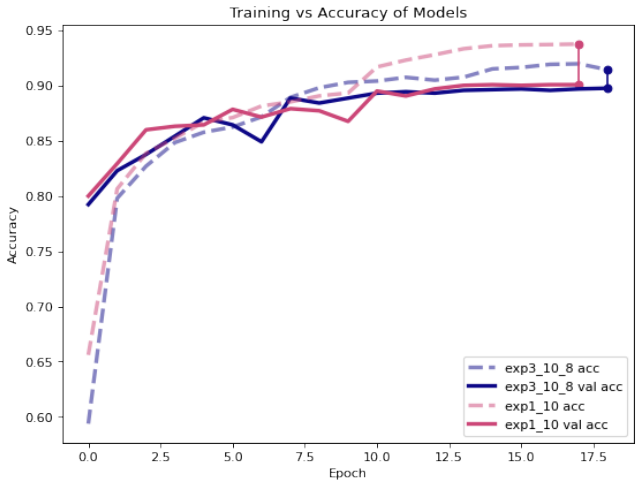

Fig. 2 Comparing the current best model with previous best model

Table 4 The table displays the performance of each model.

Model name	Accuracy	Validation Accuracy	Acc Diff
exp3_10_7	0.934	0.898	0.036
exp3_10_8	0.914	0.898	0.017
exp3_10_2	0.939	0.895	0.044
exp3_10_3	0.940	0.895	0.045
exp3_10_1	0.924	0.891	0.033
exp3_10_5	0.910	0.889	0.021
exp3_10_6	0.907	0.889	0.019
exp3_10_4	0.841	0.844	-0.003

3.4 Experiment 4

In this experiment, we wanted to try to change the width of the model so that our model is able to extract more features and find more patterns to classify our fashion items. Although we had hypothesized that our model was overfitting because it was too complex, we wanted to test our hypothesis. The strategies we will employ in this experiment are tuning the number of filters in the convolutional layers and tuning the number of units in the dense layer.

First, we wanted to see if the model needed to extract more features by using 64 and 128 as our number of filters in our

model exp4_10_3_1 but this resulted in an overfit. Then, we decided to lower our number of filters for models exp4_10_3_2 and exp4_10_3_3. While this did not cause the model to overfit, the model's validation accuracy went down. In the following models exp4_10_3_4 to exp4_10_3_7, we wanted to test if our model needed more classification power by increasing the units in our dense layers and experimented with different numbers of filters in our convolutional layers. The result was still overfitting which aligns with our original hypothesis that our model did not need to be more complex. We will keep the previous best model and proceed with exp3_10_3 in experiment 5.

Table 5 The table displays the performance of each model.

Model name	Accuracy	Validation Accuracy	Acc Diff
exp4_10_8_5	0.926	0.900	0.026
exp4_10_8_4	0.924	0.898	0.026
exp4_10_8_1	0.956	0.895	0.061
exp4_10_8_7	0.928	0.892	0.004
exp4_10_8_6	0.930	0.891	0.004
exp4_10_8_2	0.898	0.887	0.011
exp4_10_8_3	0.909	0.882	0.027

3.5 Experiment 5

We were unable to increase the model's validation accuracy by hyperparameter tuning. So in this experiment we changed the dataset. The original dataset is 80x60x3, but we chose to use grayscale for faster training time as we thought colors wouldn't be relevant and would confuse the model. However upon revision, we hypothesized that RGB would matter as different article types may have a set color palette that designers use. In this experiment, we will test and refine our model using image dimensions of 28x28x3 and 80x60x3.

First, we trained our model on the input dimensions 28x28x3. This resulted in an increase of validation accuracy to a new high of 0.9138. Then, we increased the number of units in our dense layer for models exp5_10_8_2, exp5_10_8_3 as we thought with 3 channels, there would be more classification power needed. However, the validation accuracy did not improve. We then trained our model exp5_10_8_4 with increased units in the Dense layers on our data with dimensions 80x60x3 and further increased our validation accuracy. However, the training time took a long time as the number of parameters for the model was 10 million. In model exp5_10_8_5, we added in a group of convolutional layers with max pooling so the data is downsampled and added a Dropout layer with dropout rate of 0.1 after each group of convolutions to combat overfitting. This reduced the parameters of our model to 2 million and achieved a final validation accuracy of 0.928.

Table 6 The table displays the performance of each model.

Model name	Accuracy	Validation Accuracy	Acc Diff
exp5_10_8_5	0.961	0.928	0.032
exp5_10_8_4	0.959	0.921	0.037
exp5_10_8_3	0.944	0.917	0.027
exp5_10_8_1	0.935	0.914	0.021
exp5_10_8_2	0.940	0.914	0.026

Table 7 Composition of exp5_10_8_5

Layer (type)	Output Shape	Param #
Conv	(80, 60, 16)	448
Conv	(80, 60, 16)	2320
Max Pool	(40, 30, 16)	0
Dropout	(40, 30, 16)	0
Conv	(40, 30, 32)	4640
Conv	(40, 30, 32)	9248
Max Pool	(20, 15, 32)	0
Dropout	(20, 15, 32)	0
Conv	(20, 15, 64)	18494
Conv	(20, 15, 64)	36928
Max Pool	(10, 7, 64)	0
Dropout	(10, 7, 64)	0
Flatten	(4480)	0
Dense	(512)	2294272
Dropout	(512)	0
Dense	(512)	262656
Dropout	(512)	0
Dense	(512)	262656
Dense	(8)	4104

Comparing the best models of each input dimension, we found that the validation accuracy steadily increased. For the final model, we chose to use exp5_10_8_5 with input dimensions of 80x60x3. An evaluation of our final model on testing data showed an accuracy of 0.913. This was much better than our baseline model which had an accuracy of 0.835.

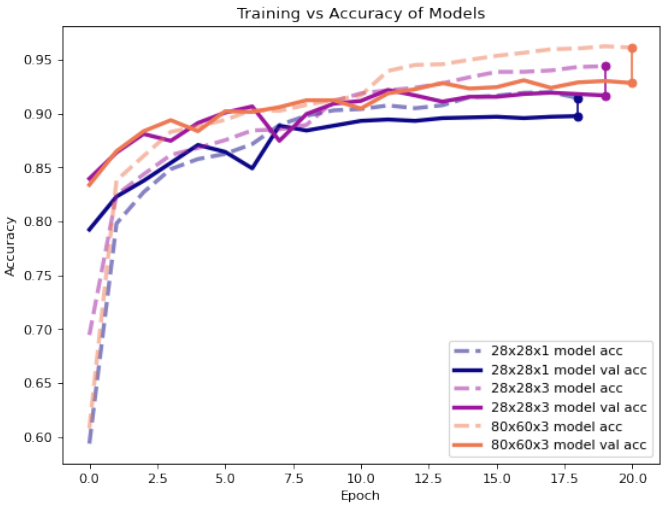

Fig. 3 Comparing the best models of each input dimension

Looking at a confusion matrix (see Figure 4 below) of predictions on our final model, we found that the model did very well for classes with more distinct shapes, and poorly on classes that were similar in shape. Most notably, the pairs of classes that had the highest proportion of mislabeling as one another were Shirts and Tops, and Kurtas and T-shirts.

4 Retrospective

We learned the conventions of building a CNN architecture and fine-tuning hyperparameters to obtain the best model. We saw the importance of the quality of the input images. The performance of

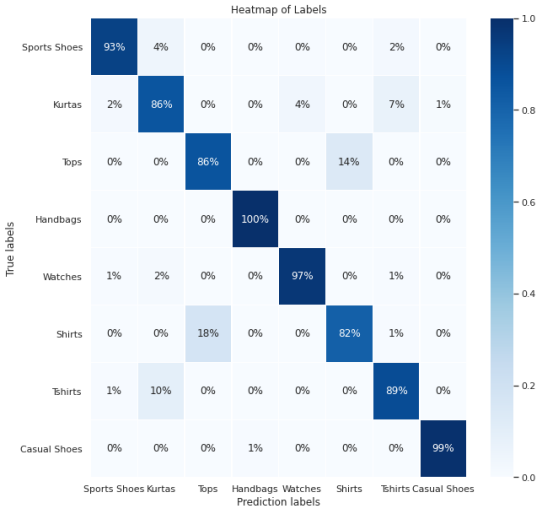

Fig. 4 Comparing the best models of each input dimension

the models increased the greatest when input images were colored instead of grayscale. We had limited computation resources on Google Colab, so we traded the resolutions and colors of input images for training efficiency. Experiment 5 demonstrated that the colors enhanced the models' classification ability. So better performance could be achieved if we did not down sample the images.

5 Appendix

First, go to our github to download the neccessary files. The github link is found here[3]. Download all the files inside the "files" folder. There should be 4 zips. Download the notebook "FashionCNN_final.ipynb" in the "notebook" folder to run the final model for our project. For a more detailed walkthrough of our project including all experiment and data preprocessing, download the notebook "FashionCNN_detailed.ipynb". Instructions to run the detailed version will be found in the detailed notebook.

5.1 Running FashionCNN_final.ipynb

After unzipping the zip files in the "files" folder, there were be four folders. "assets" will contain assets of the final model. "variables" will contain variables of the final model. "model_histories" will contain the model histories of all our experiments and models. "saved_files" will contain all our preprocessed data. There are instructions in the detailed notebook on how to reproduce all the data, but due to the run time, we will be loading all our variables and data in this notebook.

Please change the paths to each folder in the notebook to wherever you placed these folders. If you are running the notebook on Google Colab, you can upload the zips to your Drive and unzip these folders in the Colab. Make sure to run the appropriate shell commands given in the notebook if you want to mount your Drive. Before running the code, make sure you have the following libraries: Numpy, Pandas, Seaborn, Matplotlib, scikit-learn, TensorFlow Follow the instructions given in the notebook and you should be able to reproduce our model, the evaluation results, and the confusion matrix.

Notes and references

1 *Fashion Product Images (small)*, `https://www.kaggle.com/paramaggarwal/fashion-product-images-small`, Accessed: 2021-06-12.

2 *Simple MNIST convnet*, `https://keras.io/examples/vision/mnist_convnet/`, Accessed: 2021-06-15.

3 `https://github.com/chengxi600/FashionImageClassifier`.