ICDSS Challenge: Fashion MNIST Classification

Tan Si Yu (Lionel)

1. Overview

Requirements:

Choose the latest versions of any of the dependencies below:

- pandas
- numpy
- matplotlib
- sklearn
- keras
- tensorflow

Files:

Excluding the original files provided,

1. visualization_icdss.ipynb → Notebook to show t-SNE of fashion MNIST
2. train_4cnn_icdss.ipynb → Notebook to train 4 layer CNN
3. train_resnet_icdss.ipynb → Notebook to train Resnet Model
4. train_inception_icdss.ipynb → Notebook to train transfer learning with InceptionV3
5. evaluate_icdss.ipynb → Notebook to evaluate performance on test set and view other accuracy matrices

2. Dataset

Fashion-MNIST (1) is an image dataset compiled by Zalando Research to replace the MNIST dataset. It contains 70,000 28x28 grayscale images of clothing items from 10 labelled classes.

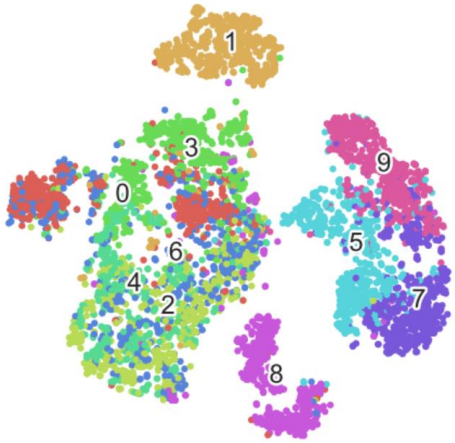
The dataset is divided into a 60,000-image train set and 10,000-image test set. The data is provided in 2 .csv files contained in data.tar.gz file.

2.1 Visualisation

I did dimensionality reduction using t-Distributed Stochastic Neighbour Embedding (t-SNE). In machine learning, dimensionality reduction is the process of reducing the number of random variables under consideration by obtaining a set of principal variables.

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a non-linear technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets.

t-Distributed stochastic neighbor embedding (t-SNE) minimizes the divergence between two distributions: a distribution that measures pairwise similarities of the input objects and a distribution that measures pairwise similarities of the corresponding low-dimensional points in the embedding. In this way, t-SNE maps the multi-dimensional data to a lower dimensional space and attempts to find patterns in the data by identifying observed clusters based on similarity of data points with multiple features.



2.2 Data Preprocessing

After using the provided reader to import the data as NumPy arrays in Python, I transformed the NumPy arrays into a float32 array of shape (60000, 28 * 28) and included an extra dimension to the array as the dimension for image channels.

I split the original training data into a training set of 48,000 images (80%) and a validation, of 12000 images (20%). The training, as elaborated later, will use both training and validation set to optimize the classifier, while keeping the test data unseen from the model so as to do a final evaluation. This allows us to tell if we are over fitting on training data by seeing if training accuracy is much higher than validation accuracy or if we should lower learning rate and train for more epochs if validation accuracy is higher than validation accuracy.

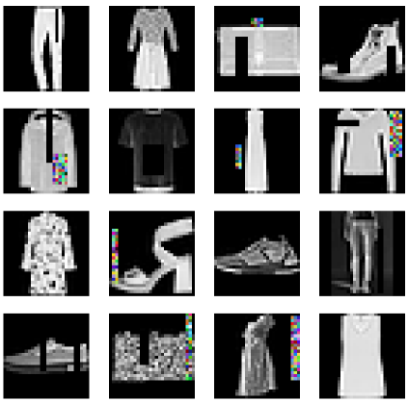
2.3 Data Augmentation

First, I rescaled the image arrays 1./255 so that values are between 0 and 1. By normalizing all of our inputs to a standard scale, we're allowing the network to more quickly learn the optimal parameters for each input node as all parameters can be updated in equal proportions.

Then, I used common data augmentation techniques such as horizontal flipping, shifting and zooming on the training samples, artificially expanding the training samples.

I also included a pre-processing function, which I learnt about during my internship. Random Erasing randomly selects a rectangle region in an image and erases its pixels with random

values. Random Erasing is complementary to commonly used data augmentation techniques such as random cropping and flipping. It is very effective in combatting occlusion, but it also allows us to train our model to not be overly dependent on a certain feature in the image to classify it. Do refer to this paper for more information on this: https://arxiv.org/abs/1708.04896



3. Models

In this work, I will train a Convolutional Neural Network classifier with 4 convolution layer using Tensorflow 2.0 with Keras deep learning library.

3.1 4 Convolutional Layer CNN

This CNN takes as input tensors of shape (28, 28, 1).

Types of layers used:

- For the **Conv2D** layers, I used 3 x 3 sized kernels as the convolution filter which extracts features from the input images by sliding over the input to produce a feature map. I also added zero padding to prevent shrinking of outputs and loosing information on the corners of image. For the activation, I used ReLU. In the past, nonlinear functions like tanh and sigmoid were used, but researchers found out that **ReLU layers** work far better because the network can train a lot faster due to computational efficiency without making a significant difference to the accuracy. It also helps to alleviate the vanishing gradient problem as it increases the nonlinear properties of the model.
- To normalize the input layers, I use the **BatchNormalization** layers to adjust and scale the activations. Batch Normalization reduces the amount by what the hidden unit values shift around (covariance shift). It improves gradient flow and acts as a form of regularization as it normalizes each input data by mean of that batch.
- I used 2 x 2 **MaxPooling2D** layers to reduces the dimensionality of each feature, which helps shorten training time and reduce number of parameters.

I stacked four of the Conv2D-BatchNormalization-MaxPooling2D blocks while incrementing number of filters at each block in order to obtain higher level features. The features in lower layers are primitive while those in upper layers are high-level abstract features made from combinations of lower-level features.

Next, I flattened the last 3D output tensor to a 1D tensor to input into a stack of Dense (Fully Connected layers). To combat overfitting, I use the Dropout layers, a powerful regularization technique. Dropout is an extremely effective and simple regularization technique that complements the other methods. While training, dropout is implemented by only keeping a neuron active with some probability pp (a hyper parameter) or setting it to zero otherwise. Randomly setting some neurons to zero in each forward pass, forces the network to have a redundant representation, preventing co adaptation of features.

My final Dense layer does the 10-way classification with a Softmax activation to calculate output based on the probabilities. Each class is assigned a probability and the class with the maximum probability is the model's output for the input.

3.2 ResNet Architecture

This model uses the famous ResNet architecture to train but due to the small image sizes, I took out the maxPooling layer. As a model depth increases, a deeper model should technically be able to perform at least as well as the shallower model as it could just learn the same feature mappings as the previous layers, but in reality this is very hard to achieve and it is extremely hard to optimize a deeper model. ResNet solves this by using residual blocks.

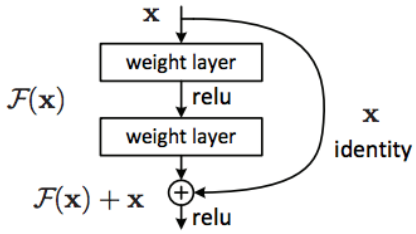


Figure 2. Residual learning: a building block.

The 'Skip Connection', identity mapping. This identity mapping does not have any parameters and is just there to add the output from the previous layer to the layer ahead. However, sometimes x and F(x) will not have the same dimension. Recall that a convolution operation typically shrinks the spatial resolution of an image, e.g. a 3x3 convolution on a 32 x 32 image results in a 30 x 30 image. The identity mapping is multiplied by a linear projection W to expand the channels of shortcut to match the residual. This allows for the input x and F(x) to be combined as input to the next layer. The Skip Connections between layers add the outputs from previous layers to the outputs of stacked layers. This results in the ability to train much deeper networks than what was previously possible.

3.3 Transfer Learning with Inception_V3

Unfortunately, I did not have the time to do this, but the script is included in the file as well.

Transfer Learning is to use pre trained models out there and train upon that model instead of re training from the beginning where there is a larger margin of error in achieving convergence. The pre-trained model I used is Inception V3 model architecture that was trained on ImageNet dataset. ImageNet is a dataset that contains 14 million images with over 1,000 classes.

The images of fashionMNIST are black and white, while the required input for Inception V3 must be coloured images. Thus, I convert the images into coloured ones with 3 channels R, G, B and resized them to 150 x 150 x 3.

I set my base model to be the pre trained Inception V3 model removing the original top dense layer and build upon it with a global average pooling2d, a dense layer with a ReLU activation, a dropout layer and a final dense classification layer.

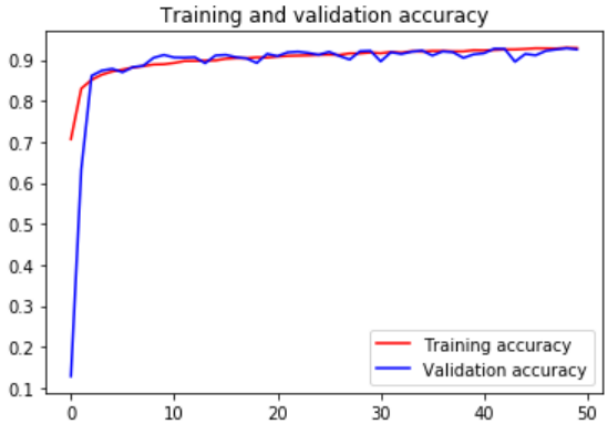
4. Training

Google Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. Training is done on the Google Colab platform using the GPU hardware accelerator.

For both models, I used Adam as optimizer, which is an improvement from SGD. The optimizer is responsible for updating the weights of the neurons via backpropagation. It calculates the derivative of the loss function with respect to each weight and subtracts it from the weight. I compiled the model with keras callbacks to save model checkpoints and to reduce learning rate upon plateau. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates.

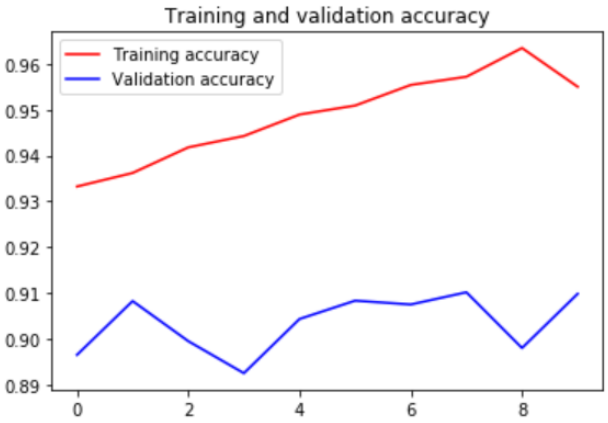
4.1 4 Convolutional Layer CNN

I used a grid search to find the most optimal training parameters for the model which includes hyperparameters such as learning rate, batch size, dropout percentage and dense layer sizes, which prompted me to use adam with a learning rate of 0.01 and a dropout percentage of 0.5 for the next training, achieving a test accuracy of 0.9135.



4.2 Training ResNet Architecture

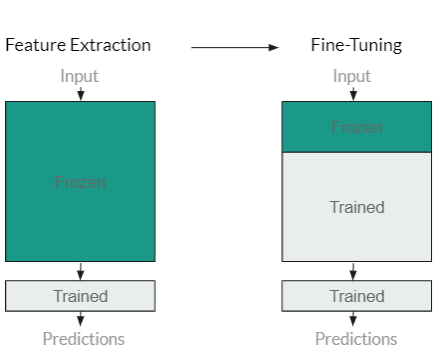
I used the same training process as the one I used for 4-layer convolutional layer. However, this time I had both model fitting to data that went through data augmentation and non-augmented data. Non augmented data was a lot faster to train and attained convergence quickly, albeit a large difference between training and validation accuracy. I believe data augmentation will be able to regularize the model and prevent over fitting but due to the lack of time, I was not able to train it till convergence. With non-augmented data, I was able to attain 0.9099 test accuracy after just 10 epochs of training.

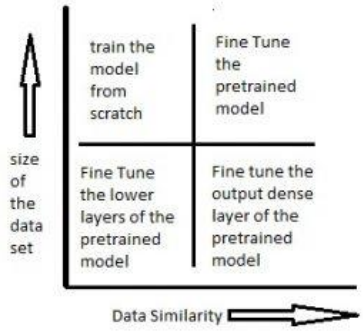


4.3 Training Transfer Learning

Unfortunately, I did not have the time to do this, but this was the planned training process.

With a pretrained model, there is no need to retrain the whole model with the new dataset as the first few convolutional layers are usually used to detect features like edges and curves, which are still applicable to our dataset.





In this case, since we have a large dataset, our neural network training would be effective. However, since the data we have is quite different as compared to the data used for training our pretrained models. The predictions made using pretrained models would not be effective. Hence, its best to train the neural network from scratch according to your data.

Instead, I did a two-pronged approach where I first did feature extraction, freezing all the weights in the InceptionV3 model and just train the dense layers that I added on to the base model. After a few epochs, I did fine tuning where I unfreeze more top layers to train it to train the deeper feature extractions at the higher convolutional layers to our dataset. This is because ImageNet is not exactly like the fashion MNIST dataset and since we have quite a big dataset to train on, we can afford to unfreeze and train larger parts of the model but on a smaller learning rate. Here is where I find SGD tend to be a better optimizer than Adam.

In fact, there has been research papers supporting this argument:

https://arxiv.org/abs/1712.07628

5. Evaluation

5.1 Results

Test accuracy is not sufficient as an indicator of whether the model is a good model. We need to analyse if the accuracy is balanced across all classes.

I used a classification Report which shows precision, recall and f1 score of each class.

Precision is true positive/(true positive and false positive). A high precision would mean classifier has a low false positive rate. Recall is true positive/(true positive and false negative. A high Recall would mean a low false negative rate. F1 score is basically 2 * precision*recall/(precision + recall). It is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. From the result it seems that classes are relatively balanced, with shirt being the hardest for the model to identify.

	precision	recall	f1-score	support
t_shirt	0.83	0.86	0.85	1000
trouser	0.98	0.99	0.99	1000
pullover	0.89	0.84	0.87	1000
dress	0.91	0.93	0.92	1000
coat	0.88	0.84	0.86	1000
sandal	0.99	0.96	0.97	1000
shirt	0.75	0.77	0.76	1000
sneaker	0.94	0.98	0.96	1000
bag	0.98	0.98	0.98	1000
ankle_boots	0.96	0.95	0.96	1000
micro avg	0.91	0.91	0.91	10000
macro avg	0.91	0.91	0.91	10000
weighted avg	0.91	0.91	0.91	10000

I also displayed the confusion matrix, where the diagonal elements represent the number of points for which the predicted label is equal to the true label, while off-diagonal elements are those that are mislabelled by the classifier. The higher the diagonal values of the confusion matrix the better, indicating many correct predictions.

For the 4 layer CNN model, I also tried to extracting feature maps at each layers to understand the intermediate outputs of the CNN model. In order to extract the feature maps we want to look at, we'll create a Keras model that takes batches of images as input, and outputs the activations of all convolution and pooling layers. To do this, I used the Keras class Model functional api.

5.2 Possible Improvements

- For a simple dataset like Fashion MNIST, perhaps better to use a simpler model and to retrain rather than doing transfer learning which is more suited for complex datasets.
- Use ensemble learning
    - Model Ensemble is a technique that can boost the model's ability to generalise and give accurate results. There were various model ensembling methods, including a simple average ensemble, voting ensemble and stacked average ensemble.

    - To achieve a stack ensemble, we need to train several individual trained models and train an additional meta-learner that takes in the outputs from the 4 models. Here, the algorithm takes the outputs of sub-models as input and attempts to learn how to best combine the input predictions to make a better prediction. The corroboration of results from the 4 models reduces bias and variance as compared to using a single model alone, resulting in an improved validation accuracy.

- Leaky ReLu instead of ReLu