Out[47]: Click here to toggle on/off the raw code.

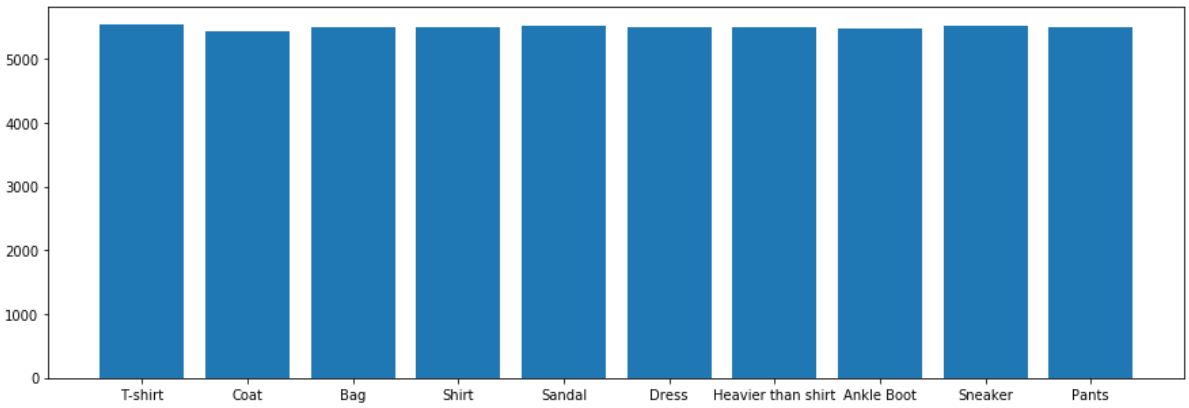
1 Exploratory Analysis (10 points)

I first try to understand the dataset and how it works. So this dataset is a dataset of fashion images. Each observation contains 784 columns which is to combine a 28x28 image matrix into one row. Our task is to classify each observation into categories based on their image information.

My english is not that good, therefore I use my own understandings to give each labels name based on their graphical representations, which is 0 represents as T shirt, 1 as pants, 2 as heavier than shirt, 3 as dress, 4 as coat, 5 as sandal, 6 as shirt, 7 as sneaker, 8 as bag and 9 as ankle Boot.

First I try to observe in the training set, how many numbers of observations do each class represent? Is it a biased dataset or it's equal weighted? I got the result:

```
Out[3]: {'T-shirt': 5543,
         'Coat': 5444,
         'Bag': 5496,
         'Shirt': 5499,
         'Sandal': 5512,
         'Dress': 5507,
         'Heavier than shirt': 5507,
         'Ankle Boot': 5488,
         'Sneaker': 5510,
         'Pants': 5494}
```



Therefore, we can see that we have a balanced dataset with approximately equal number of observations for each label. So what we are going to do is a balanced image classification dataset and we read information from the 28x28 pixels and classify the corresponding labels to each observation.

In order to get a more clear idea on how the image looks like, I visualize 5 sample graphs of each label class as below:

After reading a lot of paper and documents about the image classifications, I find out that there are two most efficients machine learning alogrithms that can classify the image the best. First is CNN which is the convolutional neural networks. And the second one is to use Random forest to classify the images.

Sample papers and documents are below that I refer: (There are also many others but just providing two samples here)

https://medium.com/intro-to-artificial-intelligence/simple-image-classification-using-deep-learning-deep-learning-series-2-5e5b89e97926 (https://medium.com/intro-to-artificial-intelligence/simple-image-classification-using-deep-learning-deep-learning-series-2-5e5b89e97926)

http://www.cs.huji.ac.il/~daphna/course/CoursePapers/bosch07a.pdf (http://www.cs.huji.ac.il/~daphna/course/CoursePapers/bosch07a.pdf)

Therefore, after doing the through expolaratory analysis I decide to Use two algorithms: CNN and Random Forest to deal with this problem.

2 Models

The first algorithm that comes to my head is CNN. Before looking at this assignment, I already heard about how CNN works well on the image classifications. In order to further prove my previous experiences, I research a lot online and conclude some of my personal perspectives: (These conclusions are all cited below):

1. the idea presented by Yann le cun in 1998: CNNs can be thought of automatic feature extractors from the image. If we just treat the images as vectors and do the machine learning algorithms, we may lose a lot of correlated information lie inside the matrix of the images. But CNN can solve that problem by effectively using adjacent pixel information to first make the image matrix simpler but remain the information of these matrix by using convolution and then use a dense layer to make the predictions.
2. Deep learning yields less error than simple machine learning, implying higher accuracy. Deep learning is not to say only perceptrons with the multi-layers, it still has a big advantage is his freedom. Deep learning can be treated as an architecture that contains a large freedom so people can always find the best "model" which best fir the dataset. Deep learning just mimic how human brain works.
3. Because CNN use multiple processing of the convolutional layer and the dense layer, it contains both the advantages of convolutional and deep neural network in the image classification.
4. Since Steinkrau showed the value of using GPUs for machine learning in 2005: "modern GPU computing and parallel computing methods have massively increased the ability to train CNN models, causing the rise of using CNN in image classification in the recent years."(cited Steinkrau,2005) Thus We can now train neural networks with very large dataset pretty efficiently with lower time cost and higher accuracy.

Therefore, the first algorithm I decide to use for the image classification is the CNN. Not only because of its higher accuracy than other algorithms (https://medium.com/intuitionmachine/why-deep-learning-is-radically-different-from-machine-learning-945a4a65da4d (https://medium.com/intuitionmachine/why-deep-learning-is-radically-different-from-machine-learning-945a4a65da4d)) but also because how it's suitable for the image classification (as described above and https://www.quora.com/Why-does-the-convolutional-neural-network-have-higher-accuracy-precision-and-recall-rather-than-other-methods-like-SVM-KNN-and-Random-Forest/answer/Haohan-Wang (https://www.quora.com/Why-does-the-convolutional-neural-network-have-higher-accuracy-precision-and-recall-rather-than-other-methods-like-SVM-KNN-and-Random-Forest/answer/Haohan-Wang) and https://arxiv.org/abs/1702.07800 (https://arxiv.org/abs/1702.07800)) and how it cost not that much time. (I tried later, usually takes above 2-3 hours, approximately similar time with SVM but much higher accuracy.)

Cited Reference:

https://www.quora.com/Why-is-CNN-used-for-image-classification-and-why-not-other-algorithms (https://www.quora.com/Why-is-CNN-used-for-image-classification-and-why-not-other-algorithms)

https://prakhartechviz.blogspot.com/2019/01/image-classification-keras.html (https://prakhartechviz.blogspot.com/2019/01/image-classification-keras.html)

https://en.wikipedia.org/wiki/Kernel_(image_processing)#Convolution (https://en.wikipedia.org/wiki/Kernel_(image_processing)#Convolution)

https://pdfs.semanticscholar.org/b8e3/613d60d374b53ec5b54112dfb68d0b52d82c.pdf (https://pdfs.semanticscholar.org/b8e3/613d60d374b53ec5b54112dfb68d0b52d82c.pdf)

https://web.stanford.edu/class/cs231a/prev_projects_2016/example_paper.pdf (https://web.stanford.edu/class/cs231a/prev_projects_2016/example_paper.pdf)

https://medium.com/intuitionmachine/why-deep-learning-is-radically-different-from-machine-learning-945a4a65da4d (https://medium.com/intuitionmachine/why-deep-learning-is-radically-different-from-machine-learning-945a4a65da4d)

https://www.quora.com/Why-does-the-convolutional-neural-network-have-higher-accuracy-precision-and-recall-rather-than-other-methods-like-SVM-KNN-and-Random-Forest/answer/Haohan-Wang (https://www.quora.com/Why-does-the-convolutional-neural-network-have-higher-accuracy-precision-and-recall-rather-than-other-methods-like-SVM-KNN-and-Random-Forest/answer/Haohan-Wang)

https://arxiv.org/abs/1702.07800 (https://arxiv.org/abs/1702.07800)

The second model I decide to use is the random forest. The reasons that I decide to use the random forest is because:

1. using random forests with appropriate parameters significantly reduce both the training cost and the test cost over SVM and at the same time, having a comparable performance which is similar accuracy.
2. "Random forests is an ensemble model which means that it uses the results from many different models to calculate the predicted label. Always the results getting from many models is better than only getting from one model".(Dahinden 2009).
3. Comparable results and accuracy with other machine learning models but not sensitive to the parameters chosen. And Easy to determine which parameters to be used(just used a cross validation can tune the parameter pretty well).
4. Avoid overfitting and avoid correlations.
5. The python code and library of random Forest is pretty easy to implement.

Therefore, Even though I dont expect the accuracy of Random forest can exceeds the CNN, but I do believe that in other ways the random forest should be a better model than other machine learning algorithms because of its less cost, less correlation effects, not very sensitive to the parameters, easy to tune the parameters, easier to implement, etc.

In other words, I believe that the deep learning (CNN) will yield the highest accuracy because the reasons and references I provide above, for the second Algorithm I personally believe other machine learning algorithms will yield similar accuracy. So I rather choose an algorithm with easy parameterization and less cost and can at the same time avoid correlations.

Thus I decide to use the random forest as my second model.

Cited Reference:

http://www.cs.huji.ac.il/~daphna/course/CoursePapers/bosch07a.pdf (http://www.cs.huji.ac.il/~daphna/course/CoursePapers/bosch07a.pdf)

http://wgrass.media.osaka-cu.ac.jp/gisideas10/papers/04aa1f4a8beb619e7fe711c29b7b.pdf (http://wgrass.media.osaka-cu.ac.jp/gisideas10/papers/04aa1f4a8beb619e7fe711c29b7b.pdf)

Data Splits (5 in the kaggle template)

I did the k-folds cross validation for my training data. And I set different K for CNN and random forest because their running time.

Since CNN model spent a significantly larger running time than Random forest, I dont want my training model to run forever. Therefore, I set K=5 for the CNN and K = 10 for the random forest.

Because the predictive classes in the training data are nearly evenly distributed, so I did not use the stratified K fold but only the simple K fold. For the CNN, I divided the training data to 5 equal pieces and run the cross validation (4 of 5 to be the training and the remaining one to be the validation). I output the average accuracy of 5 validation set. I think I did not overfit the training set because the each fold cross validation only train on 4 of 5 training set but not the left out validation set. So my final average accuracy is valid.

I did the same thing for the random forest and the only difference is that I divide the training data to 10 sub datasets not 5.

Code for CNN data splits:

kf = KFold(n_splits=5,random_state=2019,shuffle=True)

Code for RF data splits:

kf_rf = KFold(n_splits=10,random_state=2022,shuffle=True)

Training (CNN)

It's not an easy task to build a suitable deep learning model. Because there are so many choices and hyperparameters that can be used in the CNN, I have to make a lot of selections.

For the CNN model, there are several parameters (not hyperparameters that I tuned later) I want to explain why I was using these instead of others:

1. The structure of the neural network
2. activation function
3. kernel-initializer
4. kernel size
5. optimizer

1. The structure of the neural network

Based on the paper and referenced above, especially the article: https://medium.com/intro-to-artificial-intelligence/simple-image-classification-using-deep-learning-deep-learning-series-2-5e5b89e97926 (https://medium.com/intro-to-artificial-intelligence/simple-image-classification-using-deep-learning-deep-learning-series-2-5e5b89e97926), I decide to use layers of Conv2D with a Maxpooling and final two dense layers(the last one is the softmax to get the prediction classes) to be the structure of my deep learning model.

The other references of why I choose that structure is:

https://towardsdatascience.com/the-4-convolutional-neural-network-models-that-can-classify-your-fashion-images-9fe7f3e5399d (https://towardsdatascience.com/the-4-convolutional-neural-network-models-that-can-classify-your-fashion-images-9fe7f3e5399d)

https://www.geeksforgeeks.org/image-classifier-using-cnn/ (https://www.geeksforgeeks.org/image-classifier-using-cnn/)

https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks (https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks)

Based on these reference, I decide my neural network structure to be number of Conv2d, Maxpooling, dropout layers combine with Two Dense layers at the end. That was my basic structure of my neural network.

1. Activation function

The reason why I choose the activation = 'relu' is that the CNN applies a Rectified Linear Unit transformation to the convolved feature, in order to introduce the non-linearity. We know that if we dont add an appropriate activation function, the model will just like a linear regression which is not making sense. Relu: F(x) = max(0,x) returns all x for x>0 and 0 for x<0 and its purpose is to present non-linearity into the CNN model. the dataset we are using for the image classification needs first, non-linearity, and second, non-negative value. Therefore, relu is the most appropriate activation function for this dataset.

1. Kernel-initializer

I set kernel_initializer to be 'he_normal' basically inspired by: https://datascience.stackexchange.com/questions/13061/when-to-use-he-or-glorot-normal-initialization-over-uniform-init-and-what-are (https://datascience.stackexchange.com/questions/13061/when-to-use-he-or-glorot-normal-initialization-over-uniform-init-and-what-are) because I think that 'he_normal' is the most suitable initializer for our dataset.

And of course the dimension of each plot of our data is 28x28, so thats why we set the input shape to be (28,28,1).

1. kernel size

I have explained why I use relu and he_normal above. I use (2,2) for MaxPooling2D((2, 2) because based on the articles I referenced above, (3,3) is too much for the 28x28 image classification(too many information will be ignored) and (1,1) is meaningless for the MaxPooling. So (2,2) is the only suitable hyperparameter for the MaxPooling and I no longer to tune it anymore.

1. optimizer

I choose optimizer = adam because I referenced by this article: https://blog.algorithmia.com/introduction-to-optimizers/ (https://blog.algorithmia.com/introduction-to-optimizers/). It is the combination of Momentum and RMSprop (combine their advantages) and it's very useful and currently very popular for the neural network.

I also provide a running time below all of my training algorithms to compare their running times.

Remainder: all the running time below are the running time of 5-folds cross validation. So the actual running time of training will be much less than the running presented below. These are just for the running time comparisons between each models in order to tune the model.

Hyperparameter Selection (CNN)

After deciding all the parameters above, There are still several parameters in the CNN model that I have to tune, and I will try to tune these parameters below. Because of the long running time of the Neural network models, I cannot made a list and tune all of them at the same time it will cost very very long time to run, so I have to try to tune them one by one:

1. Number of layers in the model
2. Drop out coefficients
3. coefficients in Conv2D
4. coefficients in Dense Layer

The first model I used is:

model1 = Sequential()

model1.add(Conv2D(32, kernel_size=(3, 3),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1)))

model1.add(MaxPooling2D((2, 2)))

model1.add(Dropout(0.25))

model1.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))

model1.add(MaxPooling2D(pool_size=(2, 2)))

model1.add(Dropout(0.25))

model1.add(Flatten())

model1.add(Dense(64, activation='relu'))

model1.add(Dropout(0.3))

model1.add(Dense(10, activation='softmax'))

model1.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy'])

The average accuracy is about 0.945 and running time is slightly less than 3 hours.

```
CPU times: user 6h 50min 37s, sys: 54min 28s, total: 7h 45min 5s
Wall time: 2h 49min 22s
```

```
Out[133]: 0.9451272727446123
```

This is a decent accuracy and running time but I still want to tune the parameter to see if it can work better. I keep all other things equal and set the first dense layer to be 128:

model11 = Sequential()

model11.add(Conv2D(32, kernel_size=(3, 3),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1)))

model11.add(MaxPooling2D((2, 2)))

model11.add(Dropout(0.25))

model11.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))

model11.add(MaxPooling2D(pool_size=(2, 2)))

model11.add(Dropout(0.25))

model11.add(Flatten())

model11.add(Dense(128, activation='relu'))

model11.add(Dropout(0.3))

model11.add(Dense(10, activation='softmax'))

model11.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy'])

Now the new accuracy is about 0.959 and running time still slightly less than 3 hours. Implying that increasing Dense layer can improve the performance of the model.

```
CPU times: user 6h 58min 33s, sys: 56min 31s, total: 7h 55min 5s
Wall time: 2h 57min 48s

Out[137]: 0.959199999852614
```

Then I tried to tune the kernel size and add two more conv layers into the model to see what will happen:

```
model111 = Sequential()

model111.add(Conv2D(32, kernel_size=(5, 5),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1)))

model111.add(Conv2D(32, kernel_size=(5, 5),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1)))

model111.add(MaxPooling2D((2, 2)))

model111.add(Dropout(0.25))

model111.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))

model111.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))

model111.add(MaxPooling2D(pool_size=(2, 2)))

model111.add(Dropout(0.25))

model111.add(Flatten())

model111.add(Dense(128, activation='relu'))

model111.add(Dropout(0.3))

model111.add(Dense(10, activation='softmax'))

model111.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy'])
```

The resulting accuracy did not improve, which is about 0.9537 but the running time is significantly larger, which is more than 12 hours. So I conclude that increasing kernel size and add more layers is meaningless and too time costly. So I decide just to set kernel_size to be (3,3) and keep the original structure.

```
CPU times: user 1d 4min 5s, sys: 4h 37min 44s, total: 1d 4h 41min 49s
Wall time: 12h 18min 11s

Out[140]: 0.9537454546755011
```

Then I tried to tune the convolutional layer. I tried to add one more convolutional layer: Conv2D(16, kernel_size=(3, 3),activation='relu',kernel_initializer='he_normal',input_shape=(28, 28, 1)) to the first model I tried and to see whether the performance will be improved:

model2 = Sequential()

model2.add(Conv2D(16, kernel_size=(3, 3),activation='relu',kernel_initializer='he_normal',input_shape=(28, 28, 1)))

model2.add(MaxPooling2D((2, 2)))

model2.add(Dropout(0.25))

model2.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))

model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Dropout(0.25))

model2.add(Conv2D(64, (3, 3), activation='relu'))

model2.add(Dropout(0.4))

model2.add(Flatten())

model2.add(Dense(64, activation='relu'))

model2.add(Dropout(0.3))

model2.add(Dense(10, activation='softmax'))

model2.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy'])

And I discovered that the accuracy actually largely decrease. So that's not a good way to build the model.

```
CPU times: user 3h 32min 38s, sys: 56min 16s, total: 4h 28min 54s
Wall time: 1h 39min 25s

Out[144]: 0.9199272725625471
```

Then I tried to add the number of neurons in each conv layer by multiplying by 2 to see how it works:

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',kernel_initializer='he_normal',input_shape=(28, 28, 1))

model.add(MaxPooling2D((2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(Dropout(0.3))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.3))

model.add(Dense(10, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy'])
```

The running time is still around 3 hours (actually more than 3 hours) but the accuracy is 0.948 (not anything improving from the second model I used. Since the only difference between this model and my second model is I add a 128 conv layer into the model. So I concluded it's meaningless to add any further conv layers to my model. (adding both 16 and 128 does not improve the performance.)

```
CPU times: user 7h 41min 35s, sys: 1h 39min 46s, total: 9h 21min 21s
Wall time: 3h 15min 39s

Out[147]: 0.9483818182078275
```

So After all of these tryings, I decide to use my second model to be as my currently bese model. I have decided the number of the layers, the structure, the parameters in the Conv2D and Maxpooling, then I begin to tune the Dropout terms.

I tried to tune the dropout term for the 128 dense layer from 0.3 to 0.25 to see if there's any difference:

model3 = Sequential()

model3.add(Conv2D(32, kernel_size=(3, 3),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1)))

model3.add(MaxPooling2D((2, 2)))

model3.add(Dropout(0.25))

model3.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))

model3.add(MaxPooling2D(pool_size=(2, 2)))

model3.add(Dropout(0.25))

model3.add(Flatten())

model3.add(Dense(128, activation='relu'))

model3.add(Dropout(0.25))

model3.add(Dense(10, activation='softmax'))

model3.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy'])

The running time and the accuracy is very very similar to my second model therefore I think that dropout term does not make a large difference on the performance.

```
CPU times: user 6h 58min 52s, sys: 57min 21s, total: 7h 56min 13s
Wall time: 2h 47min 17s

Out[151]: 0.9579090908830816
```

After deciding all of these, I made a last move on the dense layers. I tune the dense layers from 128 to be 256 and to see what will happen: (will it be a large improvement on the accuracy?)

model3_2 = Sequential()

model3_2.add(Conv2D(32, kernel_size=(3, 3),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1)))

model3_2.add(MaxPooling2D((2, 2)))

model3_2.add(Dropout(0.25))

model3_2.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))

model3_2.add(MaxPooling2D(pool_size=(2, 2)))

model3_2.add(Dropout(0.25))

model3_2.add(Flatten())

model3_2.add(Dense(256, activation='relu'))

model3_2.add(Dropout(0.25))

model3_2.add(Dense(10, activation='softmax'))

model3_2.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy'])

It turns out that it will have a large improvement on the accuracy: the accuracy being larger than 0.965 and not cost very much running time. So I find out that adding dense layer may be the best way to improve the performance of the model.

```
CPU times: user 7h 18min 50s, sys: 1h 34min 9s, total: 8h 52min 59s
Wall time: 3h 31min 3s
```

```
Out[20]: 0.967672727281397
```

I discover increasing the number of dense layer maybe a best way to improve the model performance, so I switch dense layer from 256 to 512 and see what happens:

model4 = Sequential()

model4.add(Conv2D(32, kernel_size=(3, 3),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1)))

model4.add(MaxPooling2D((2, 2)))

model4.add(Dropout(0.25))

model4.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))

model4.add(MaxPooling2D(pool_size=(2, 2)))

model4.add(Dropout(0.25))

model4.add(Flatten())

model4.add(Dense(512, activation='relu'))

model4.add(Dropout(0.5))

model4.add(Dense(10, activation='softmax'))

model4.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy'])

it turns out that the accuracy keep increasing to 0.97+. And the running time does not significantly increases. Total time is nearly the same but my conputer shut down for a moment. So the real running time is still about 3 hours.

For here I add my dropout term to be 0.5 because higher dense layer implies larger probability to be overfitting so I add a higher dropout term to avoid overfitting. For here I did not turn much more about the dropout term because i think it's not that useful based on the previous runnings. 0.45,0.5,0.55 will not make a large difference.

```
CPU times: user 7h 53min 36s, sys: 1h 34min 43s, total: 9h 28min 20s
Wall time: 9h 33min 53s
```

```
Out[14]: 0.9701454546581616
```

Then I made a last try to tune the number of dense layers to 1028, the accuracy improves a little bit but costs much more time(about two more hour). So I decide to use dense layer = 512 for this dataset.

```
CPU times: user 9h 21min 35s, sys: 1h 54min 39s, total: 11h 16min 14s
Wall time: 15h 17min 17s
```

```
Out[24]: 0.9730000000173396
```

At last I add the BatchNormalization for each layer to see if there will be any improvements. However, the accuracy decreases. And it costs much much more time to run the alrogithm.

Therefore, I decide not to add the BatchNormalization term.

```
CPU times: user 19h 40min 23s, sys: 3h 16min 4s, total: 22h 56min 28s
Wall time: 1d 10h 14min 2s
```

```
Out[25]: 0.9679999999566512
```

At last, I decide my final CNN model to be:

model4 = Sequential()

model4.add(Conv2D(32, kernel_size=(3, 3),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1)))

model4.add(MaxPooling2D((2, 2)))

model4.add(Dropout(0.25))

model4.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))

model4.add(MaxPooling2D(pool_size=(2, 2)))

model4.add(Dropout(0.25))

model4.add(Flatten())

model4.add(Dense(512, activation='relu'))

model4.add(Dropout(0.5))

model4.add(Dense(10, activation='softmax'))

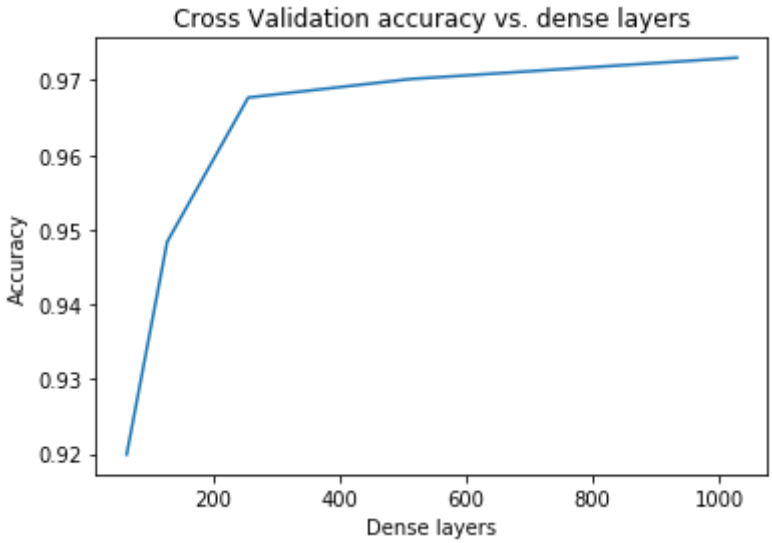
model4.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy'])

```
CPU times: user 1h 52min 2s, sys: 23min 3s, total: 2h 15min 6s
Wall time: 51min 25s
```

This final model takes 51min to train the whole training dataset and I think it's an appropriate time.

After all of these trainings (tuning hyperparameters), I discover that the most important hyperparameter may be the number of layers in the dense.

Therefore, in order to present a more clear picture on how the number of dense layers will affect the accuracy (performance of the model), I made a plot:



Training (Random Forest)

There are several parameters in the Random Forest that I want to explain why I was using these:

1. criterion (Gini and Entropy)
2. max_features
3. bootstrap
4. n-jobs

1. Criterion:

"Gini measurement is the probability of a random sample being classified incorrectly if we randomly pick a label according to the distribution in a branch."

"Entropy is a measurement of information (or rather lack thereof). We calculate the information gain by making a split. Which is the difference in entripies. This measures how you reduce the uncertainty about the label."

Above two sentences is reference to: https://www.quora.com/What-is-difference-between-Gini-Impurity-and-Entropy-in-Decision-Tree (https://www.quora.com/What-is-difference-between-Gini-Impurity-and-Entropy-in-Decision-Tree). I think it's a very good explanations for these two contents and it's what we learned the definition of these two in the class. So I copy them here. (Cited)

For here I use the Gini (which is the default of the random forest function here), because for my personal opinion, Gini works better on continuous variables and Entropy is better for discrete attributes. And Gini tends to minimize misclassification and Gini usually run faster.

1. max_features

Based on the class note, The best maximum features for each bootstrapping will be the square root of the number of features. Thus I choose the sqrt max_features based on what we learned in class.

1. bootstrap

I set true for this parameter because it can add more randomness to our model in order to avoid overfitting and correlation errors.

1. n-jobs

I set this paramter to -1 so we use all processors to the number of jobs to run in parallel for both fit and predict. It can help our algorihm to run faster.

Hyperparameter Selection (Random Forest)

In this section I will tune the parameter of the Random Forest. Based on the class notes, class readings, and homeworks, and at the same time avoid long-time running, I decide only to tune the n_estimators, which is the number of trees and max_depth, which is the depth of each trees.

I use the cross validation to tune the hyperparameter. Unlike the CNN which I divide the training into 5 pieces, this time I divide the training into 10 pieces. I tested n_estimators from [50,100,200,400,700,1000,1500,2000,3000,5000] and max_depth from [4,5,6,8].

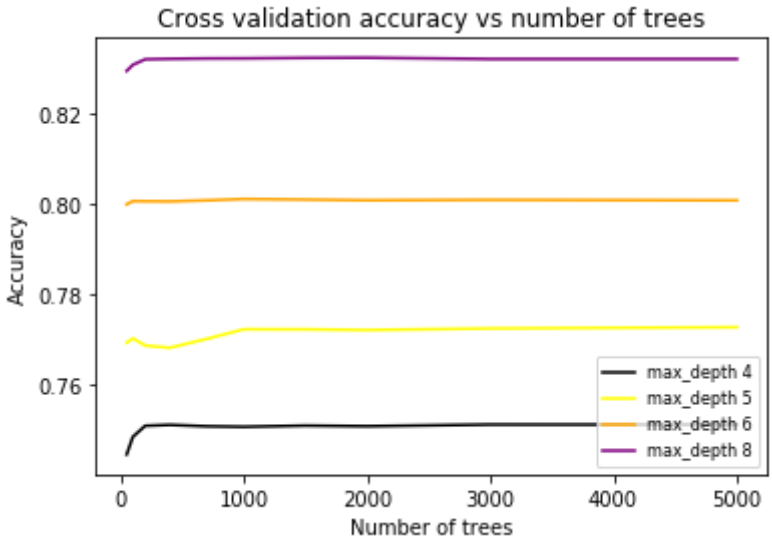
The whole cross validation tuning process takes more than a day, it's especially slower when the number of trees and max_depth getting larger.

The result can be seen below, it turns out that the number of trees did not affect the accuracy very much, max_depth turns out to be a much important hyperparameter. The highest accuracy occurs at max_depth = 8 and n_estimators = 5000. It shows the trend that higher max_depth resulting higher accuracy. But much lower than CNN.

I did not continue tune the parameter for the random forest because obviously when the max_depth and n_estimators get larger the running time of random forest will be very much like the Final CNN model I defined above.

However, the accuracy of Random Forest model still much lower than CNN's. Therefore, I can conclude that at similar time, CNN's accuracy is much higher than the Random Forest. Therefore, I choose CNN as my final model decision.

```
CPU times: user 2d 16h 54min 2s, sys: 25min 37s, total: 2d 17h 19min 40
s
Wall time: 1d 1h 12min 56s
```



Based on the plot above we can see that number of trees does not make a big difference but max_depth makes a very significant differences.

```
Out[53]: [8, 5000, 0.8320727272727273]
```

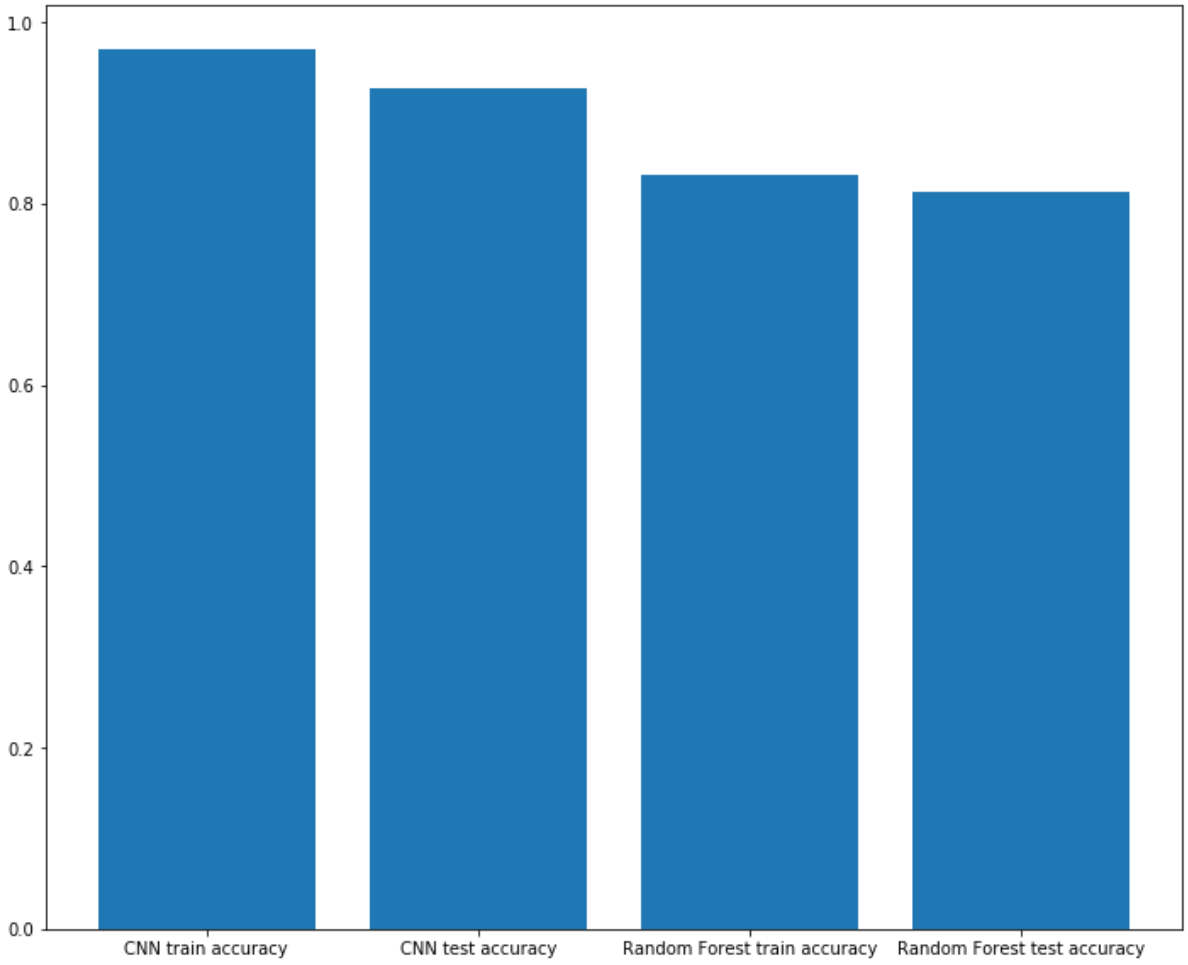
Above is the hyperparameters with the best accuracy, which is tree = 5000 and max_depth = 8.

```
CPU times: user 53min 50s, sys: 15.2 s, total: 54min 5s
Wall time: 18min 58s
```

Predictive accuracy

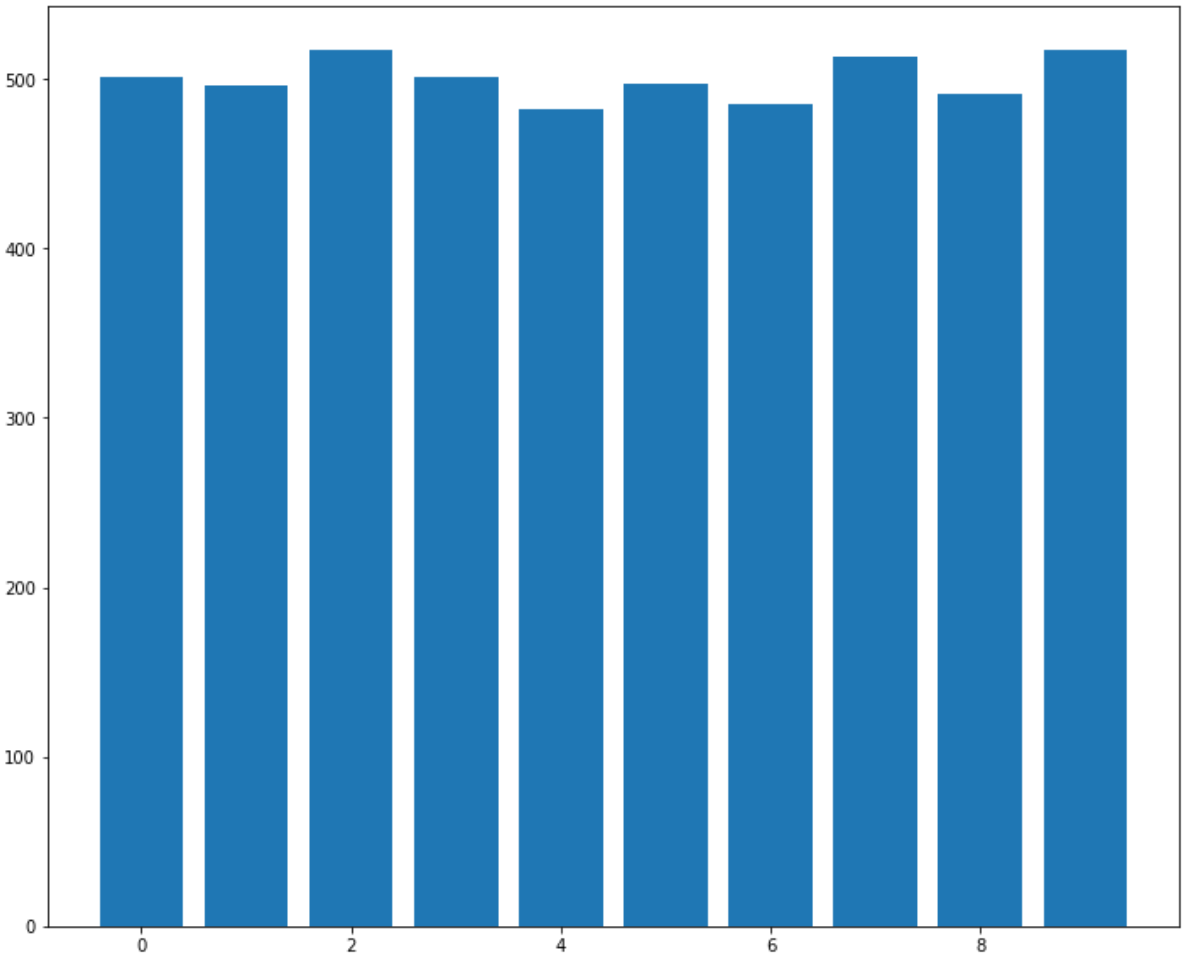
After submitted my both model predictions to the kaggle I got two accuracy: 0.9266 for CNN and 0.8122 for Random Forest. Although they are both slightly lower than their performance in the train set, it's acceptable. I did not find the severe problem about overfitting.

I made a barplot to visualize the accuracy. For here I use accuracy instead of classification error because I think it's basically the same: accuracy = 1- classification error and I think presenting accuracy may be more straightforward.



From the plot above we can see that CNN is significantly better than the Random Forest model, for both train data and the test data.

Then I visualize the predictions:



From the visualization above we can see that the prediction for the test test is nearly uniformly distributed, with class 2,7 and 9 slightly more than others.

Errors and the mistakes

I think the largest error/mistake I have made during this competition is that I underestimate the running time of some CNN models and the random Forest model. Therefore, I spent a lot of time running these model and waiting for them to be done.

Especially the random forest when I was tuning the n_estimators and the max depth I spent more than a day to run the algorithm. If I know it will take that long I will not include that much potential parameters in the tuning step.

I think there are two hardest parts of this competetion. First is that there is a large freedom building the CNN model. There are so many potential hyperparameters (number of layers, numbers of neurons in each layer, combinations, etc). So it's definitely a hard task to find the optimal model. The second hardest part is the running time of the model. Since it usually takes a long time to run the CNN and large random forest model, especially I also have to run the cross validation. So It really cost a lot of time in model trainings.

Codes

```
# import pandas as pd import numpy as np import matplotlib.pyplot as plt import tensorflow as tf import keras from keras.models import Sequential from keras.layers import Dense, Flatten, Conv2D, Dropout, MaxPooling2D,BatchNormalization from sklearn.model_selection import train_test_split from sklearn.model_selection import StratifiedKFold from sklearn.model_selection import KFold from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import accuracy_score from collections import Counter # Codes to load the dataset train = np.load('train_images.npy') valid = np.load('test_images.npy') train_label = np.load('train_labels.npy') # change the data into pandas dataframe train = pd.DataFrame(train) valid = pd.DataFrame(valid) train_label = pd.DataFrame(train_label) train_label.columns = ['labels'] #create the dictionary for each labels #used to output the different fashion styles numbers data_label = {0 : "T-shirt", 1: "Pants", 2: "Heavier than shirt", 3: "Dress", 4: "Coat", 5: "Sandal", 6: "Shirt", 7: "Sneaker", 8: "Bag", 9: "Ankle Boot"} label_c = train_label["labels"].value_counts() #count the different label appearance time label_c.keys()[1] d = {} for i in range(len(label_c)): #append the value counts to the corresponding label names in the data_label dic created above index = label_c.keys()[i]# get the keys l = data_label[index] #corresponding to the label name d[l] = label_c[i]# match the value counts d #Create the visualization of the dic above dd = pd.DataFrame(d,index=[0]).T dd.columns = ['num'] fig = plt.subplots(figsize=(15,5)) plt.bar(dd.index,dd.num) pass labels = np.load('train_labels.npy') train2 = train.copy() train2['labels'] = labels plt_images = [] plt_labels = [] for i in label_c.keys(): #for all labels i in the dataset s = train2[train2["labels"] == i] # get all observations with the corresponding label i sample = s.sample(n=5) #randomly sample 5 observations from this label for j, s in enumerate(sample.values): img = np.array(sample.iloc[j, :-1]).reshape(28,28) #reshape each observation back to 28x28 pixel matrix plt_images.append(img) #append the images, used to visualizing plt_labels.append(sample.iloc[j, -1]) #append the label #plot these sample images for each label f, ax = plt.subplots(5,10, figsize=(16,12)) for i, img in enumerate(plt_images): ax[i//10, i%10].imshow(img, cmap='Blues') ax[i//10, i%10].axis('off') ax[i//10, i%10].set_title(data_label[plt_labels[i]]) plt.show() y = keras.utils.to_categorical(train_label.labels, 10) #alternate the dataset ready for CNN num_images = train.shape[0] x_as_array = train.values X = train.values.reshape(num_images, 28, 28, 1)# get ready the X for CNN kf = KFold(n_splits=5,random_state=2019,shuffle=True)# 5 folds split the data model1 = Sequential()# define the model #first add 32 Conv layer model1.add(Conv2D(32, kernel_size=(3, 3),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1))) #Add a maxpooling layer model1.add(MaxPooling2D((2, 2))) # Add dropout term to avoid overfitting model1.add(Dropout(0.25)) # Then add 64 conv layer model1.add(Conv2D(64, kernel_size=(3, 3), activation='relu')) # Add a Maxpooling model1.add(MaxPooling2D(pool_size=(2, 2))) #Add dropout term to avoid overfitting model1.add(Dropout(0.25)) #Flatten to prepare for the dense layer
```

model1.add(Flatten()) #Add a 64 layer dense layer model1.add(Dense(64, activation='relu')) #Add dropout term to avoid overfitting model1.add(Dropout(0.3)) # use softmax to classify to each labels model1.add(Dense(10, activation='softmax')) # define the loss, optimizer and metrics. model1.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy']) %%time #to calculate the time used overall_accu1 = [] for train, test in kf.split(X,y): # transform train and valid x,y to numpy array train_x = np.array(X)[train] train_y = np.array(y)[train] valid_x = np.array(X)[test] valid_y = np.array(y)[test] # fit the model train_model1 = model1.fit(train_x, train_y, batch_size=128, epochs=50, verbose=0, validation_data=(valid_x, valid_y)) # get the validation accuracy for each fold acc1 = train_model1.history['val_acc'][-1] overall_accu1.append(acc1) sum(overall_accu1)/5 #basically same code as above model11 = Sequential() model11.add(Conv2D(32, kernel_size=(3, 3),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1))) model11.add(MaxPooling2D((2, 2))) model11.add(Dropout(0.25)) model11.add(Conv2D(64, kernel_size=(3, 3), activation='relu')) model11.add(MaxPooling2D(pool_size=(2, 2))) model11.add(Dropout(0.25)) model11.add(Flatten()) model11.add(Dense(128, activation='relu')) model11.add(Dropout(0.3)) model11.add(Dense(10, activation='softmax')) model11.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy']) %%time overall_accu11 = [] for train, test in kf.split(X,y): train_x = np.array(X)[train] train_y = np.array(y)[train] valid_x = np.array(X)[test] valid_y = np.array(y)[test] train_model11 = model11.fit(train_x, train_y, batch_size=128, epochs=50, verbose=0, validation_data=(valid_x, valid_y)) acc11 = train_model11.history['val_acc'][-1] overall_accu11.append(acc11) sum(overall_accu11)/5 #basically same code as above model111 = Sequential() #add two consecutive 32 Conv2D layer model111.add(Conv2D(32, kernel_size=(5, 5),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1))) model111.add(Conv2D(32, kernel_size=(5, 5),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1))) model111.add(MaxPooling2D((2, 2))) model111.add(Dropout(0.25)) #add two consecutive 64 Conv2D layer model111.add(Conv2D(64, kernel_size=(3, 3), activation='relu')) model111.add(Conv2D(64, kernel_size=(3, 3), activation='relu')) model111.add(MaxPooling2D(pool_size=(2, 2))) model111.add(Dropout(0.25)) model111.add(Flatten()) model111.add(Dense(128, activation='relu')) model111.add(Dropout(0.3)) model111.add(Dense(10, activation='softmax')) model111.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy']) %%time overall_accu111 = [] for train, test in kf.split(X,y): train_x = np.array(X)[train] train_y = np.array(y)[train] valid_x = np.array(X)[test] valid_y = np.array(y)[test] train_model111 = model111.fit(train_x, train_y, batch_size=128, epochs=50, verbose=0, validation_data=(valid_x, valid_y)) acc111 = train_model111.history['val_acc'][-1] overall_accu111.append(acc111) sum(overall_accu111)/5 #basically similar code as above model2 = Sequential() model2.add(Conv2D(16, kernel_size=(3, 3),activation='relu',kernel_initializer='he_normal',input_shape=(28, 28, 1))) model2.add(MaxPooling2D((2, 2))) model2.add(Dropout(0.25)) model2.add(Conv2D(32, kernel_size=(3, 3), activation='relu')) model2.add(MaxPooling2D(pool_size=(2, 2))) model2.add(Dropout(0.25)) model2.add(Conv2D(64, (3, 3), activation='relu')) model2.add(Dropout(0.4)) model2.add(Flatten()) model2.add(Dense(64, activation='relu')) model2.add(Dropout(0.3)) model2.add(Dense(10, activation='softmax')) model2.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy']) %%time overall_accu2 = [] for train, test in kf.split(X,y): train_x = np.array(X)[train] train_y = np.array(y)[train] valid_x = np.array(X)[test] valid_y = np.array(y)[test] train_model2 = model2.fit(train_x, train_y, batch_size=128, epochs=50, verbose=0, validation_data=(valid_x, valid_y)) acc2 = train_model2.history['val_acc'][-1] overall_accu2.append(acc2) sum(overall_accu2)/5 #basically similar code as above model = Sequential() model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',kernel_initializer='he_normal',input_shape=(28, 28, 1))) model.add(MaxPooling2D((2, 2))) model.add(Dropout(0.25)) model.add(Conv2D(64, kernel_size=(3, 3), activation='relu')) model.add(MaxPooling2D(pool_size=(2, 2))) model.add(Dropout(0.25)) model.add(Conv2D(128, (3, 3), activation='relu')) model.add(Dropout(0.3)) model.add(Flatten()) model.add(Dense(128, activation='relu')) model.add(Dropout(0.3)) model.add(Dense(10, activation='softmax')) model.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy']) #basically similar code as above %%time overall_accu = [] for train, test in kf.split(X,y): train_x = np.array(X)[train] train_y = np.array(y)[train] valid_x = np.array(X)[test] valid_y = np.array(y)[test] train_model = model.fit(train_x, train_y,

batch_size=128, epochs=50, verbose=0, validation_data=(valid_x, valid_y)) acc = train_model.history['val_acc'][-1] overall_accu.append(acc) sum(overall_accu)/5 #basically similar code as above model3 = Sequential() model3.add(Conv2D(32, kernel_size=(3, 3),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1))) model3.add(MaxPooling2D((2, 2))) model3.add(Dropout(0.25)) model3.add(Conv2D(64, kernel_size=(3, 3), activation='relu')) model3.add(MaxPooling2D(pool_size=(2, 2))) model3.add(Dropout(0.25)) model3.add(Flatten()) model3.add(Dense(128, activation='relu')) model3.add(Dropout(0.25)) model3.add(Dense(10, activation='softmax')) model3.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy']) #basically similar code as above %%time overall_accu3 = [] for train, test in kf.split(X,y): train_x = np.array(X)[train] train_y = np.array(y)[train] valid_x = np.array(X)[test] valid_y = np.array(y)[test] train_model3 = model3.fit(train_x, train_y, batch_size=128, epochs=50, verbose=0, validation_data=(valid_x, valid_y)) acc3 = train_model3.history['val_acc'] [-1] overall_accu3.append(acc3) sum(overall_accu3)/5 #basically similar code as above model3_2 = Sequential() model3_2.add(Conv2D(32, kernel_size=(3, 3),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1))) model3_2.add(MaxPooling2D((2, 2))) model3_2.add(Dropout(0.25)) model3_2.add(Conv2D(64, kernel_size=(3, 3), activation='relu')) model3_2.add(MaxPooling2D(pool_size=(2, 2))) model3_2.add(Dropout(0.25)) model3_2.add(Flatten()) model3_2.add(Dense(256, activation='relu')) model3_2.add(Dropout(0.25)) model3_2.add(Dense(10, activation='softmax')) model3_2.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy']) #basically similar code as above %%time overall_accu3_2 = [] for train, test in kf.split(X,y): train_x = np.array(X)[train] train_y = np.array(y)[train] valid_x = np.array(X)[test] valid_y = np.array(y) [test] train_model3_2 = model3_2.fit(train_x, train_y, batch_size=128, epochs=50, verbose=0, validation_data= (valid_x, valid_y)) acc3_2 = train_model3_2.history['val_acc'][-1] overall_accu3_2.append(acc3_2) sum(overall_accu3_2)/5 #basically similar code as above model4 = Sequential() model4.add(Conv2D(32, kernel_size=(3, 3),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1))) model4.add(MaxPooling2D((2, 2))) model4.add(Dropout(0.25)) model4.add(Conv2D(64, kernel_size=(3, 3), activation='relu')) model4.add(MaxPooling2D(pool_size=(2, 2))) model4.add(Dropout(0.25)) model4.add(Flatten()) model4.add(Dense(512, activation='relu')) model4.add(Dropout(0.5)) model4.add(Dense(10, activation='softmax')) model4.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy']) #basically similar code as above %%time overall_accu4 = [] for train, test in kf.split(X,y): train_x = np.array(X)[train] train_y = np.array(y)[train] valid_x = np.array(X)[test] valid_y = np.array(y)[test] train_model4 = model4.fit(train_x, train_y, batch_size=128, epochs=50, verbose=0, validation_data=(valid_x, valid_y)) acc4 = train_model4.history['val_acc'] [-1] overall_accu4.append(acc4) sum(overall_accu4)/5 #basically similar code as above model6 = Sequential() model6.add(Conv2D(32, kernel_size=(3, 3),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1))) model6.add(MaxPooling2D((2, 2))) model6.add(Dropout(0.25)) model6.add(Conv2D(64, kernel_size=(3, 3), activation='relu')) model6.add(MaxPooling2D(pool_size=(2, 2))) model6.add(Dropout(0.25)) model6.add(Flatten()) model6.add(Dense(1028, activation='relu')) model6.add(Dropout(0.5)) model6.add(Dense(10, activation='softmax')) model6.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy']) #basically similar code as above %%time overall_accu6 = [] for train, test in kf.split(X,y): train_x = np.array(X)[train] train_y = np.array(y)[train] valid_x = np.array(X)[test] valid_y = np.array(y)[test] train_model6 = model6.fit(train_x, train_y, batch_size=128, epochs=50, verbose=0, validation_data=(valid_x, valid_y)) acc6 = train_model6.history['val_acc'] [-1] overall_accu6.append(acc6) sum(overall_accu6)/5 #basically similar code as above model5 = Sequential() model5.add(Conv2D(32, kernel_size=(3, 3),activation='relu', kernel_initializer='he_normal',input_shape=(28, 28, 1))) model5.add(BatchNormalization()) model5.add(MaxPooling2D((2, 2))) model5.add(Dropout(0.25)) model5.add(Conv2D(64, kernel_size=(3, 3), activation='relu')) model5.add(BatchNormalization()) model5.add(MaxPooling2D(pool_size=(2, 2))) model5.add(Dropout(0.25)) model5.add(Flatten()) model5.add(Dense(512, activation='relu')) model5.add(BatchNormalization()) model5.add(Dropout(0.5)) model5.add(Dense(10, activation='softmax')) model5.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy']) %%time #basically similar code as above overall_accu5 = [] for train, test in kf.split(X,y): train_x = np.array(X)[train] train_y = np.array(y)[train] valid_x = np.array(X)[test] valid_y = np.array(y)[test] train_model5 = model5.fit(train_x, train_y, batch_size=128, epochs=50, verbose=0, validation_data=(valid_x, valid_y)) acc5 = train_model5.history['val_acc'][-1] overall_accu5.append(acc5) sum(overall_accu5)/5 %%time

```
train_model4 = model4.fit(X, y, batch_size=128, epochs=50, verbose=0) # append all cross validation accuracy to a single list accus = np.array([overall_accu2,overall_accu,sum(overall_accu3_2)/5,sum(overall_accu4)/5,sum(overall_accu6)/5]) # append the corresponding dense layer number dense = np.array([64,128,256,512,1028]) # plot the plot plt.plot(dense,accus) plt.ylabel('Accuracy') plt.xlabel('Dense layers') plt.title('Cross Validation accuracy vs. dense layers') pass #prepare the dataset train = np.load('train_images.npy') valid = np.load('test_images.npy') train_label = np.load('train_labels.npy') train_rf = pd.DataFrame(train) valid_rf = pd.DataFrame(valid) train_y_rf = pd.DataFrame(train_label) kf_rf = KFold(n_splits=10,random_state=2022,shuffle=True) # set the 10 folds cross validation l1 = [50,100,200,400,700,1000,1500,2000,3000,5000] # set the number of trees for tuning parameter l2 = [4,5,6,8] # set the max depth of trees for tuning parameter %%time overall_accu_rf = [] for i in l2: for j in l1: #for different max depth and different number of trees tmp_acc = [] # define the randomforest classifier rf = RandomForestClassifier(n_estimators=j, max_depth=i,n_jobs=-1,random_state=20192019) for train, test in kf_rf.split(train_rf,train_label): # change the data to numpy arrays train_x = np.array(train_rf)[train] train_y = np.array(train_label)[train] valid_x = np.array(train_rf)[test] valid_y = np.array(train_label)[test] # fit the training data rf.fit(train_x,train_y) # pred the validation set pred = rf.predict(valid_x) acc_rf = accuracy_score(valid_y, pred) tmp_acc.append(acc_rf) overall_accu_rf.append([i,j,sum(tmp_acc)/10]) acc_plt = [] n_tree = [] acc_plt4 = [] acc_plt5 = [] acc_plt6 = [] acc_plt8 = [] n_tree4 = [] n_tree5 = [] n_tree6 = [] n_tree8 = [] labels = [4,5,6,8] colors = ["black","yellow","orange","purple"] #define the different color for different max depth for n in range(40): #the first 10 are max_depth 4's accuracy if n < 10: acc_plt4.append(overall_accu_rf[n][2]) n_tree4.append(overall_accu_rf[n][1]) #the 11-20 are max_depth 5's accuracy elif n < 20 and n >= 10: acc_plt5.append(overall_accu_rf[n][2]) n_tree5.append(overall_accu_rf[n][1]) #the 21-30 are max_depth 5's accuracy elif n < 30 and n >= 20: acc_plt6.append(overall_accu_rf[n][2]) n_tree6.append(overall_accu_rf[n][1]) #the 31-40 are max_depth 5's accuracy elif n < 40 and n >= 30: acc_plt8.append(overall_accu_rf[n][2]) n_tree8.append(overall_accu_rf[n][1]) #append the accuracy and n_tree lists to the overall lists use for visualization acc_plt.append(acc_plt4) acc_plt.append(acc_plt5) acc_plt.append(acc_plt6) acc_plt.append(acc_plt8) n_tree.append(n_tree4) n_tree.append(n_tree5) n_tree.append(n_tree6) n_tree.append(n_tree8) for i, color in zip(range(4),colors): plt.plot(n_tree[i], acc_plt[i], color=colors[i],label='max_depth {0}' #plot the different max_depth's accuracy ''.format(labels[i])) #differentiated by the color plt.legend(loc="lower right",prop={'size': 8}) plt.xlabel('Number of trees') plt.ylabel('Accuracy') plt.title('Cross validation accuracy vs number of trees') pass max_acc = 0 max_num = 0 for n in overall_accu_rf: if n[2] > max_acc: max_num = n # Get the maximum accuracy combination n %%time rf = RandomForestClassifier(n_estimators=5000, max_depth=8,n_jobs=-1,random_state=20192019) rf.fit(train_rf,train_label) num_images = valid.shape[0] X_test = valid.values.reshape(valid.shape[0], 28, 28, 1) predicted_classes = model5.predict_classes(X_test) predicted_classes = pd.DataFrame(predicted_classes) accu_list = [0.9701454546581616,0.9266,0.8320727272727273,0.8122] #use to compare the performance of two model name_list = ['CNN train accuracy','CNN test accuracy','Random Forest train accuracy','Random Forest test accuracy'] fig = plt.figure(figsize=(12,10)) plt.bar(x = name_list,height=accu_list) pass Counter(predicted_classes[0]) fig = plt.figure(figsize=(12,10)) plt.bar(x = list(Counter(predicted_classes[0]).keys()),height=list(Counter(predicted_classes[0]).values())) pass
```