Using different models in Images Classification

Table of Contents:

Title and author:

1. **Title**:

Using different models in Images Classification

2. **Author**:

- Name: Thai Quoc Hoang
- Date of birth: January 09, 2000
- Email: quocthai9120@gmail.com (mailto:quocthai9120@gmail.com) / qthai912@uw.edu (mailto:qthai912@uw.edu)

Summary of research questions and Results:

Part 0: Using k-Nearest-Neighbors approach:

1. How accurate the model classify and does the model predict outputs with high confidence (that is the probability of confidence is high)?

- We want to know whether the model is a good model for the dataset by finding the accuracy of correct predictions the model can make for each dataset in this project. We also want to know whether our model can really classify the object with high confidence or every labels in the dataset have the nearly the same probabilities. With the conclusions, we can determine whether to use another model for a particular dataset or improve the model by preprocessing data or tuning hyperparameters.
- Approach:
  - Implement kNN model to classify images in both MNIST Fashion dataset and CIFAR_10 dataset.
  - Visualize the images in the validation and test set with real label and predicted labels to verify that no error occurs. Also use several images as examples and visualize all k-nearest neighbors to analyze the similarity.
  - To answer how accurate the model classify the outputs, get the accuracy of correct predictions for all testing data in each dataset.
  - To compare the probability of each class appear in k-nearest neighbors, use several testing images as examples then get the scores for all distinct labels appear in k-nearest neighbors of the testing instance than analyze these scores.
- Results:
  - CIFAR_10 dataset:
    - How accurate the model classify: Using 10000 unseen images as testing instances, our kNN model gives predictions with 23.56% correct accuracy.
    - Does the model predict outputs with high confidence: For most images in this dataset, kNN model gives predictions with low confidence because the probabilities of several labels are similar to the resulted label's probability.
  - Fashion MNIST dataset:
    - How accurate the model classify: Using 10000 unseen images as testing instances, our kNN model gives predictions with 85.97% correct accuracy.
    - Does the model predict outputs with high confidence: For most images in this dataset, kNN model gives predictions with high confidence because the probability of the resulted label is much more higher compare to other labels.

Part 1: Using Convolutional Neural Network approach:

1. How accurate the model classify and does the model predict outputs with high confidence (that is the probability of confidence is high)? How does CNN model work in each dataset compare to kNN model?

    - We want to know whether the model is a good model for the dataset by finding the accuracy of correct predictions the model can make for each dataset in this project. We also want to know whether our model can really classify the object with high confidence or every labels in the dataset have the nearly the same probabilities. Moreover, we also want to know how much different in accuracy of correct predictions made by each model (kNN versus CNN) for each dataset so that we can consider the model that fit our purpose for a particular dataset. With the conclusions, we can determine whether to use another model for a particular dataset or improve the model by preprocessing data or tuning hyperparameters.
    - Approach:
        - Implement a Convolutional Neural Network model for both datasets. To be fair, in this research we will use the same configuration of model for both dataset and analyze the different between the two datasets.
        - Visualize the images in the validation and test set with real label and predicted labels using to show how the model works.
        - To answer how accurate the model classify the outputs, get the accuracy of correct predictions for all testing data in each dataset.
        - To answer does the model predicts outputs with high confidence: use several images as example and get all probability the model predicts for each class.
        - Compare the accuracy for each dataset while using CNN with the result while using kNN in both datasets to give a conclusion about advantages and disadvantages for each model in each dataset.
    - Results:
        - CIFAR_10 datset:
            - How accurate the model classify: Using 10000 unseen images as testing instances, our CNN model gives predictions with 70.09% correct accuracy.
            - Does the model predict outputs with high confidence: For most images in this dataset, CNN model gives predictions with high confidence because the probability of the resulted label is much more higher compare to other labels.
            - How does CNN model work in CIFAR_10 dataset compare to kNN model: For CIFAR_10 dataset, CNN model works much better compare to kNN model with the different of 46.53% correct accuracy for 10000 testing images.
        - Fashion MNIST dataset:
            - How accurate the model classify: Using 10000 unseen images as testing instances, our CNN model gives predictions with 90.55% correct accuracy.
            - Does the model predict outputs with high confidence: For most images in this dataset, CNN model gives predictions with high confidence because the probability of the resulted label is much more higher compare to other labels.
            - How does CNN model work in CIFAR_10 dataset compare to kNN model: In this dataset, if the clients only need the correct accuracy of 85.97% or more (if the clients only need to classify easier images), kNN is a good enough model for this dataset. CNN is still a better model in this dataset.
2. What does the model learn in each convolutional layer?

- Understanding what happen in each layer can help us know how to tuning hyperparameters and help us avoid time consuming because of training bad model. In this question, we only focus on CIFAR_10 dataset because CIFAR_10 dataset is much more complex compare to Fashion MNIST, which mean we can understand Fashion MNIST dataset by unnderstanding CIFAR_10 dataset.
- Approach:
  - We need to visualize the filters in each convolutional layer. As Convolutional layers are just special case of dense layers (layers after flatten), We can use the fact that the weights in each convolutional layer are the filters we use. We can ignore bias at this point because we only want to get the filters. Therefore, we can visualize the filters in each convolutional layers of the model by getting the all the weights of each layer and plot them.
  - As our filters only have size (5x5) or (3x3), maybe it will be hard for us to recognize the feature the model learn in each filter. Therefore, we can visualize the outputs after each layer and analyze these outputs by observation to see what feature does it learn. Use serval images as training data and visualize the activations of these images through different layers in the model. Then we can check back by comparing with the filters we plots above to give conclusion.
- Results:
  - The deeper the layers, the more abstract the model learns.
  - Each filter has a characteristic that is different to others. The model learns features from local positions of the images. There are also a few images with every pixels are black, which mean the model learns nothing by applying a filter to the image to get that particular activation (dead filters).

Motivation and background:

Nowadays, we are living in a world with richer data than ever since. Visual data such as images or videos are also one of them. According to https://www.omnicoreagency.com (https://www.omnicoreagency.com), every minute YouTube users upload about 300-hours of videos, Facebook users upload 243,000 photos, and every day Instagram users upload more than 100 million photos. With that enormous amount of data, we cannot think how much time we need to spend if we want to analyze all of these data to take additional actions such as creating a list of recommendation or clustering these data to different categories.

On the other hand, with the advantage that we can collect data much easier than before with the support of technologies and internet and with the advantage the computer hardware is much stronger and therefore allows us to do a significant amount of work in a small amount of time.

With that reason, it is reasonable for us to let computers do human's task that is analyzing and processing visual data. By analyzing visual data, computers can help us clustering or labeling data then putting these data in the category that most fit them.

To analyze data, especially images, we need to recognize the objects inside the images. One of the choices to recognize objects inside images is to first classify a number of objects so that the computer would able to understand the feature of each object. A core key to classify accurate objects

is the choice of model and the understanding of that model so that we can improve the model to let the machine learns.

In this project, I am going to do research on the k-Nearest-Neighbors model and Convolutional Neural Network Model in image classification. For each model, I will analyze how the model works for the different datasets and try to give conclusions about choosing an accurate model in image classification. Therefore, by understanding how a model works and how well it does in a particular dataset, people will be able to consider whether a model is fit in a particular dataset or not. Besides, by understanding deeply how a CNN model learn in each layer, developments can understand there model better for tuning hyperparameters to make the model better. With that, we can train machines to classify images and therefore it can do for us further extensive tasks such as object detection or object tracking,...

Datasets:

In this project, I will use both Fashion MNIST dataset and CIFAR_10 dataset to analyze the choice of models in different datasets. The Fashion MNIST and CIFAR_10 datasets are very different (in Fashion MNIST dataset, all objects have the same scale and there are no background to mislead from training; in CIFAR_10 dataset, the objects in the dataset do not necessarily have the same scale and there are background to mislead from training).

Fashion MNIST:

Source and Information:

Fashion-MNIST dataset is a dataset of Zalando's article images. The datset contains 60000 training samples and 10000 testing samples and each sample is a 28x28 grayscale image, associated with a label from 10 classes in the dataset.

10 classes of Fashion MNIST dataset are:

0. T-shirt/top
1. Trouser
2. Pullover
3. Dress
4. Coat
5. Sandal
6. Shirt
7. Sneaker
8. Bag
9. Ankle boot

Dataset layout:

The dataset used in this project is the csv version of the original fashion MNIST dataset, which is downloaded from Kaggle in this link: https://www.kaggle.com/zalando-research/fashionmnist (https://www.kaggle.com/zalando-research/fashionmnist). The training data ('fashion-mnist_train.csv')

has 60000 rows and 784 columns that represent 28x28 images. The testing data ('fashion-mnist_test.csv') has 10000 rows and 784 columns that represents 28x28 images.

The dataset can be download through the link (Kaggle) provided above or can be directly download here: https://www.kaggle.com/zalando-research/fashionmnist/downloads/fashionmnist.zip/4 (https://www.kaggle.com/zalando-research/fashionmnist/downloads/fashionmnist.zip/4).

CIFAR_10 dataset:

Source and Information:

The CIFAR-10 dataset contains 60000 32x32 colour images in 10 classes (6000 images/class) that are randomly shuffled and are divided into 2 parts:

- 50000 images for training (5000 images for each class).
- 10000 images for testing (1000 images for each class).

The classes are completely mutually exclusive and there is no overlap between classes.

10 classes of CIFAR-10 dataset are:

0. airplane
1. automobile
2. bird
3. cat
4. deer
5. dog
6. frog
7. horse
8. ship
9. truck

The dataset were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton and can be accessed through the link provided here : https://www.cs.toronto.edu/~kriz/cifar.html (https://www.cs.toronto.edu/~kriz/cifar.html) or directly download here: https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz (https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz).

Dataset layout:

The CIFAR-10 datset used in this project is the Python Version.

The download compressed file contains 7 files: data_batch_1, data_batch_2, data_batch_4, data_batch_5,test_batch, and batches.meta, which are Python "pickled" objects.

The source of the dataset (https://www.cs.toronto.edu/~kriz/cifar.html (https://www.cs.toronto.edu/~kriz/cifar.html)) provides a function called 'unpickle(file)' to open such files and returns the data as dictionaries.

Methodology:

Part 0: Using k-Nearest-Neighbor Approach:

1. How accurate the model classify and does the model predict outputs with high confidence (that is the probability of confidence is high)?

    - Implement kNN model to classify images in both MNIST Fashion dataset and CIFAR_10 dataset by the same method as below:
        - Define a function to calculate the distances between the training images and the testing images by using the distance formula: $|(\vec{A} - \vec{B})|^{\frac{1}{k}}$, where k is the norm the client wants to use (in common, norm is $k = 2$).
        - Define a function to get k elements that nearest the testing images by get through all training images and for each image compute the distance of the current training image and the testing image by the distance formula above. Then gets k elements that has the smallest distances.
        - Define a function to get the best result from k nearest elements using a function to compare k-nearest neighbors. Currently, the function to calculate the score of each distinct label $a[i]$ that appears in k-nearest neighbors is: $\frac{n}{k}$, where n is the total times $a[i]$ appears in k-nearest neighbors and k is the number of nearest neighbors we use. In words, the score of each label is the number of times that each label appears in k-nearest neighbors divided by k.
        - The distinct label with highest score will be the predicted result.
    - Visualize the images in the validation and test set with real label and predicted labels to verify the results. Also use several images as examples and visualize all k-nearest neighbors to analyze the similarity.
    - Get through all testing instances and predict label for each instance by the model in order to compute the correct accuracy (for both datasets).
    - For each dataset, use several images as examples. For each image, look at the returned scores for all labels for each testing image to see how different the score for each label is and analyze that whether the predicted label has a big different in score than the rest or all labels have similar scores and the resulted label just has a slightly higher score.
    - **Conculsion**:
        - If the model has the correct accuracy in any of these 2 datasets at least 80%, then kNN can be considered as a good model for that dataset (because we can still improve the accuracy by pre-processing dataset or combine with other models). Otherwise, we have to think about whether we can combine with other algorithm to make the kNN model better or we have to use another model for the current dataset.
        - For each image, if all distinct labels in k-nearest neighbors have almost the same score (nearly about $\frac{1}{k}$), which mean that they all have the same probability, then the model does not predicts outputs with high confidence for that single image. This means the model classifies well for the current image. If the resulted label has a much higher score than the remaining labels, then the model does predicts output with high confidence (this is different to whether it predicts accurate or not) for that single image. This means the model does not really classifies well for the current image and we need to improve the model so that it can predicts other images better.

Part 1: Using Convolutional Neural Network approach:

1. How accurate the model classify and does the model predict outputs with high confidence (that is the probability of confidence is high)? How does CNN model work in each dataset compare to kNN model?

    - Divide the trainig data to smaller training data and validation data (ratio 80:20).
    - Implement a Convolutional Neural Network model. To be fair, in this research we will use the same configuration of model for both dataset and analyze the different. The models for both dataset will have the layers belows:
        - Convolutional Layer: 32 filters with kernel size (5x5) and stride 1.
        - Activation Layer: Use ReLU activation function.
        - MaxPooling Layer: Use max pooling with size (2x2) and stride 2.
        - Convolutional Layer: 64 filters with kernel size (5x5) and stride 1.
        - Activation Layer: Use ReLU activation function.
        - MaxPooling Layer: Use max pooling with size (2x2) and stride 2.
        - Convolutional Layer: 128 filters with kernel size (3x3) and stride 1.
        - Activation Layer: Use ReLU activation function.
        - MaxPooling Layer: Use max pooling with size (2x2) and stride 2.
        - Fully-connected layer: Flatten the current input.
        - Hidden Layer: 256 neurals
        - Activation Layer: Use ReLU activation function.
        - Drop out layer: Use drop out to drop 50% units.
        - Output Layer: number of neurals is the number of classes (here we use the activation softmax).
    - Use Stochastic gradient descent optimizer to optimize the categorical cross entropy loss function. Then train the data with the batch size 32 until the loss of validation data stops decreasing.
    - Predicts the results in test data and compute the correct accuracy.
    - Visualize the images in the validation and test set with real label and predicted labels to show how the model works.
    - To answer how accurate the model classify the outputs, get the accuracy of correct predictions for all testing data in each dataset.
    - To answer does the model predicts outputs with high confidence: use several images as example and get all probability the model predicts for each class.
    - Compare the accuracy for each dataset while using CNN with the result while using kNN in both datasets to give a conclusion about advantages and disadvantages for each model in each dataset.
    - **Conculsion**:
        - If the CNN model gives a correct accuracy for a dataset that at least 80%, then the model is a good model for the current dataset. Otherwise, analyze solution to improve accuracy of the model by try regularization.
        - For each image, if the return probability of the resulted class is much higher than the rest (how much higher is depend on the weights and bias so we cannot tell at this point), then the model gives prediction with high confidence for this single image. If all classes have the same probability, then the model does not gives prediction with high confidence for this single image.

        - If CNN model works much more better than kNN in each dataset (> 10% correct accuracy), then CNN model is a better model for the current dataset. Otherwise, because kNN is not really bad for that dataset because we can also improve the dataset using PCA or regularization for pre-processing part so that kNN will able to work better.

2. What does the model learn in each convolutional layer?

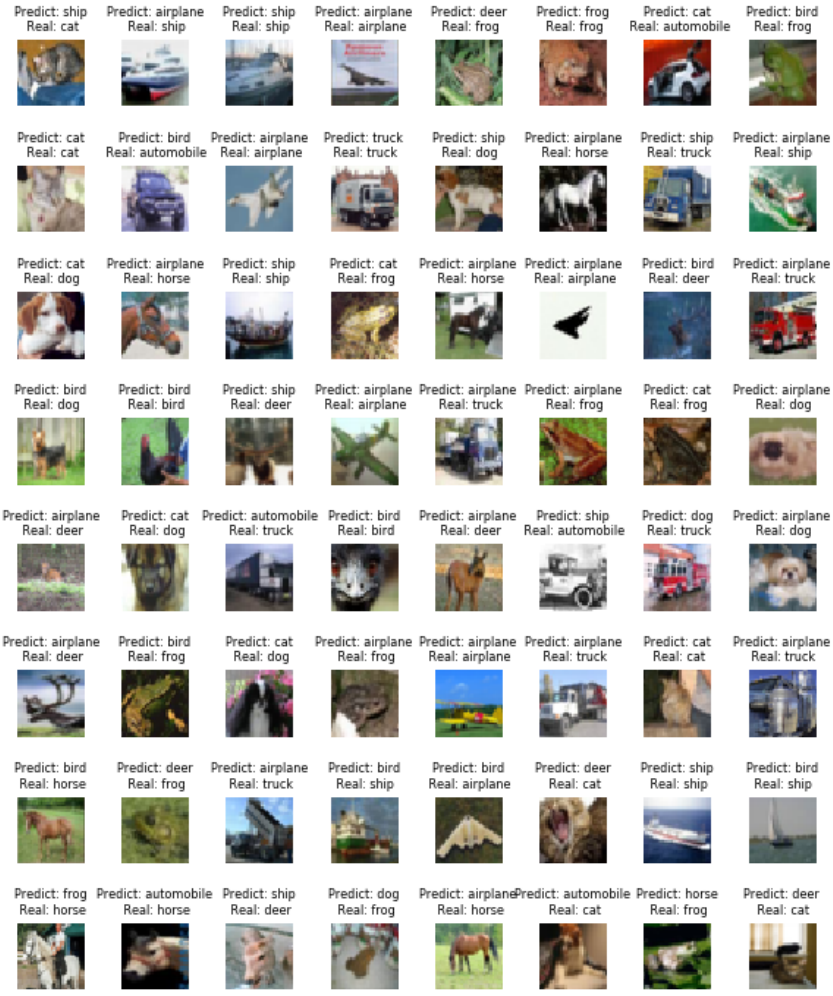
    - We need to visualize the filters in each convolutional layer. As Convolutional layers are just special case of dense layers (layers after flatten), We can use the fact that the weights in each convolutional layer are the filters we use. We can ignore bias at this point because we only want to get the filters. Therefore, we can visualize the filters in each convolutional layers of the model by getting the all the weights of each layer and plot them.
    - To visualize the outputs after each convolutional layer, we can use several images as examples. Set up a sub-model that has input as the orignal input and the output is a convolutional layer in our CNN model. Predict the input image with all weights and biases we already trained to get the output in the desired output layer we want to visualize. In an output layers, there are n outputs, where n is the number of filters in the convolutional layer. The n outputs therefore have shapes of 2-dimensional matrix and so we can plot to visualize all the output as grayscale images. Use the convention that the more brighter the pixel, the closer the value to 255 to give conclusion about feature that each filter learned.
    - Compare between the filter plots and the output plots to verify the conlusion.
    - **Conculsion**: After visualize the filters and outputs after each convolutional layer, we can understand exactly what feature does each filter learns after we trained the model. If we find a layer that the model cannot learns anything (for instance, all plotted images are completely black), we need to change our layer, such as change the kernel size, to fix the current issue or the result after that layer will be strongly affected.

Results:

Part 0: Using k-Nearest-Neighbors approach:

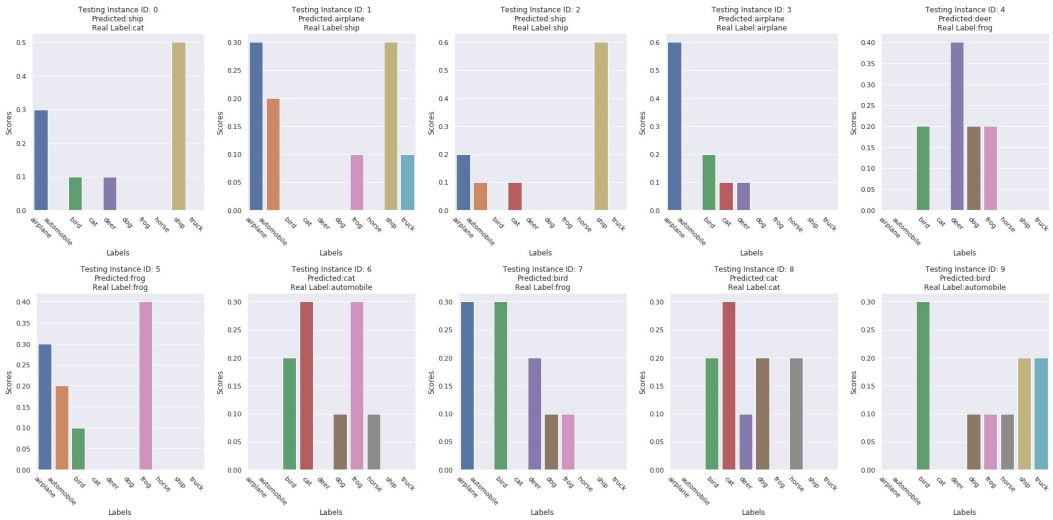
1. How accurate the model classify and does the model predict outputs with high confidence (that is the probability of confidence is high)?

    - **Conclusions**:
        - **CIFAR_10 dataset**:
            - How accurate the model classify: Using 10000 unseen images as testing instances, our kNN model gives predictions with 23.56% correct accuracy. Using several examples to test the result, we got:



As the examples as shuffled randomly, the images can represent the trend of the CIFAR_10 dataset. We can see the trend that the model gives a low percentage of correct predictions even though it is better than randomly guessing. This gives an intuition that the kNN model is not a good model for the CIFAR_10 dataset.

- Does the model predict outputs with high confidence: Using the first ten examples in test set, we got the result:

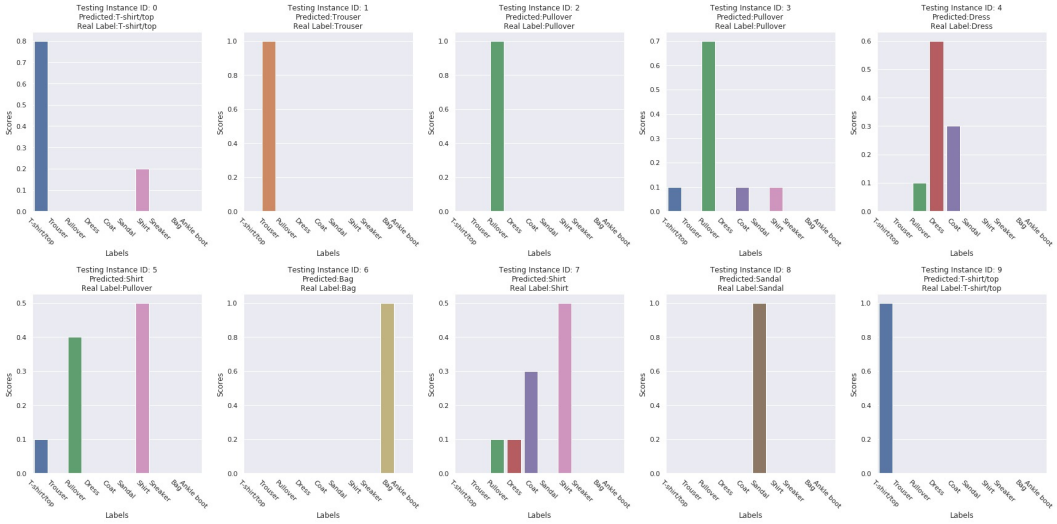


As the 10 examples are shuffled randomly, the results can represent the trend of the CIFAR_10 dataset. From the results we have produced, we recognize the fact that for most images in this dataset, kNN model gives predictions with low confidence because the probabilities of several labels are similar to the resulted label's probability.

- **Fashion MNIST dataset**:
  - How accurate the model classify: Using 10000 unseen images as testing instances, our kNN model gives predictions with 85.97% correct accuracy. Using several examples to test the result, we got:

As the examples as shuffled randomly, the images can represent the trend of the Fashion MNIST dataset. We can see the fact that the model only gives wrong predictions with a few images where it is really confusing to detect even with human eyes. Therefore, as the model gives us 85.97% correct accuracy, the model can be considered as a good model for this dataset. We can still improve the model to get better predictions.

  - Does the model predict outputs with high confidence: Using the first ten examples in test set, we got the result:



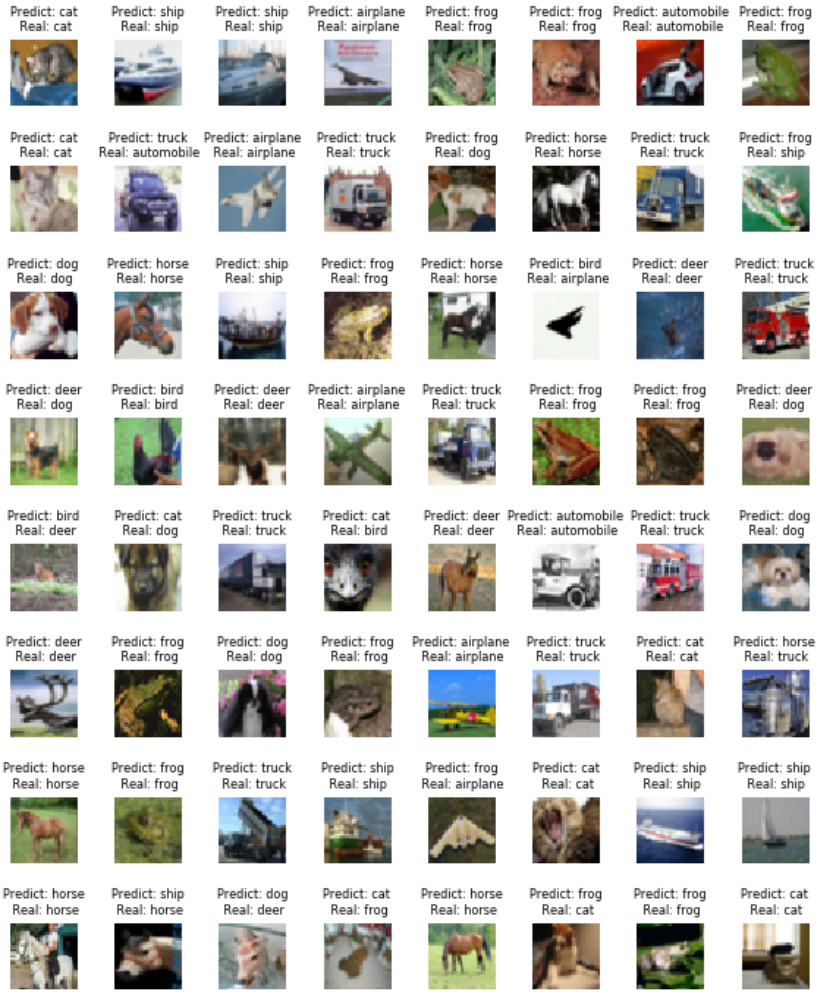
As the 10 examples are shuffled randomly, the results can represent the trend of the Fashion MNIST dataset. From the results we have produced, we recognize the fact that for most images in this dataset, kNN model gives predictions with high confidence because the probability of the resulted label is much more higher compare to other labels.

- **Discussions**:
  - The kNN model works much better for the Fashion MNIST Dataset compared to the CIFAR_10 dataset. This is reasonable because of the way the kNN model works (calculating distances between corresponding pixels in all training images and testing image and get the k-nearest neighbors and get the label that appears most in that k-nearest neighbors). As the images in Fashion MNIST dataset are grayscale, the model will predict better because it does not have to consider different in color much. Moreover, the images in Fashion MNIST dataset are centered and all objects have the similar shape (not deform) and all images have the same background. Therefore, the distance between pixels of similar object will be much smaller than other objects and so the model can works well. In constrast, the CIFAR_10 dataset has images with different forms for the same object and they all have different backgrounds and positions and colors. Therefore, just comparing the distance between corresponding pixel cannot give accurate predictions.
  - With the reason above, we should not choose kNN model to predict the images with different shapes, backgrounds, and positions as CIFAR_10 dataset. For datasets that are similar to Fashion MNIST, which the images have similar shapes, backgrounds, and positions, the kNN model would work better.

Part 1: Using Convolutional Neural Network approach:

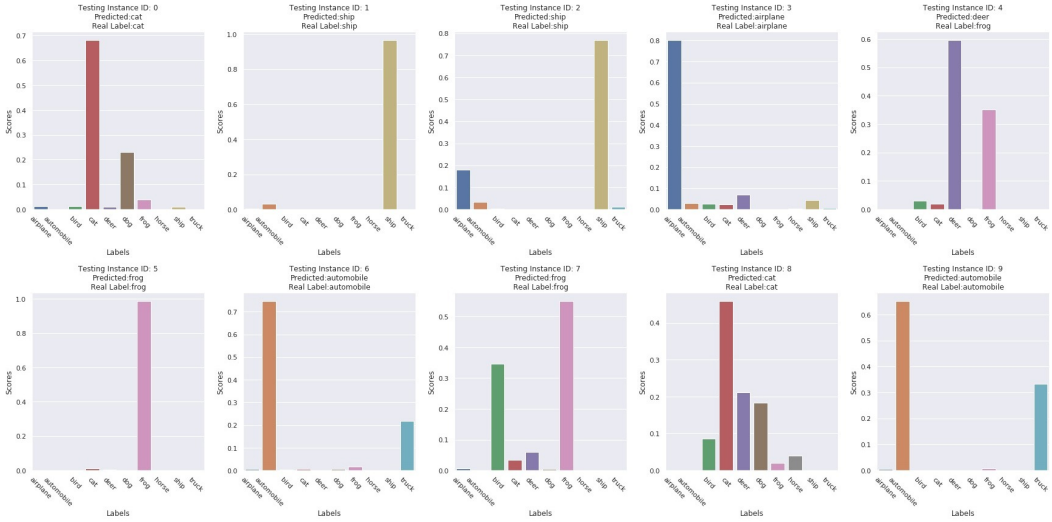
1. How accurate the model classify and does the model predict outputs with high confidence (that is the probability of confidence is high)? How does CNN model work in each dataset compare to kNN model?

- **Conclusions**:
  - **CIFAR_10 dataset**:
    - How accurate the model classify: Using 10000 unseen images as testing instances, our CNN model gives predictions with 70.09% correct accuracy. Using several examples to test the result, we got:



As the examples as shuffled randomly, the images can represent the trend of the CIFAR_10 dataset. We can see the trend that the model gives most correct predictions for images contains object appear clearly and sometimes give incorrect predictions with images contain objects in the forms that are hard to detect (even with human eyes). Therefore, even though the model only gives 70.09% correct accuracy, it currently is a good model enough and we can still improve the model to give better predictions.

- Does the model predict outputs with high confidence: Using the first ten examples in test set, we got the result:



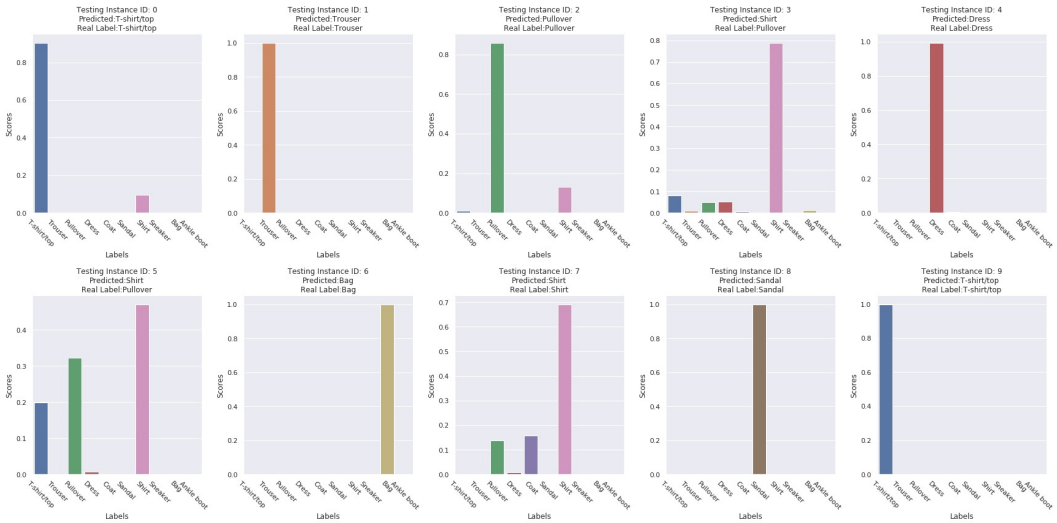
As the 10 examples are shuffled randomly, the results can represent the trend of the CIFAR_10 dataset. From the results we have produced, we recognize the fact that for most images in this dataset, CNN model gives predictions with high confidence because the probability of the resulted label is much more higher compare to other labels.

- How does CNN model work in CIFAR_10 dataset compare to kNN model: For this dataset, using CNN model gives 70.09% correct accuracy with testing data, while using kNN model only gives us 23.56% correct accuracy with the same testing data. It means that for CIFAR_10 dataset, CNN model works much better compare to kNN model with the huge different of 46.53% correct accuracy for 10000 testing images. Depends on how the clients consider what is enough for a dataset, we can gives different opinions about 46.53% of different in correct accuracy for 10000 testing images. However, because our kNN model currently only gives 23.56% correct accuracy, we cannot use kNN model separately to classify this dataset. For this dataset, CNN is a much better model to use to classify objects.

- **Fashion MNIST dataset**:
  - How accurate the model classify: Using 10000 unseen images as testing instances, our CNN model gives predictions with 90.55% correct accuracy. Using several examples to test the result, we got:

As the examples as shuffled randomly, the images can represent the trend of the Fashion MNIST dataset. We can see the fact that the model only gives wrong predictions with a few images where it is really confusing to detect even with human eyes. Therefore, as the model gives us 90.55% correct accuracy, the model can be considered as a good model for this dataset. We can still improve the model to get better predictions.

- Does the model predict outputs with high confidence: Using the first ten examples in test set, we got the result:



As the 10 examples are shuffled randomly, the results can represent the trend of the Fashion MNIST dataset. From the results we have produced, we recognize the fact that for most images in this dataset, CNN model gives predictions with high confidence because the probability of the resulted label is much more higher compare to other labels.

    - How does CNN model work in CIFAR_10 dataset compare to kNN model: For this dataset, using CNN model gives us 90.55% correct accuracy with testing data, while using kNN model gives us 85.97% correct accuracy with the same testing data. Depends on how the clients consider what is enough for a dataset, we can gives different opinions about 4.58% of different in correct accuracy for 10000 testing images. However, compare to CIFAR_10 dataset, this different is much smaller. Therefore, in this dataset, if the clients only need the correct accuracy of 85.97% or more (if the clients only need to classify easier images), kNN is a good enough model for this dataset. CNN is still a better model in this dataset.

- **Discussions**:
    - It is reasonable that CNN works better in both dataset since the way CNN model works is using convolutional to keep more information about positions of pixels and the power of continuously feedfowarding and using backpropagation to update weights and biases to minimize loss between predictions and real labels while kNN model only ranking labels based on nearest neighbors of each testing instances with the scores being compute by using distances between each corresponding pixel between training images and testing instances.
    - For CIFAR_10 dataset, each image for each class has a different form, shape, and position of the object. Moreover, there are background in each image. Therefore, as the kNN model only compute different value of each corresponding pixel between testing image and each training image, these factors will be the noise and make the model work not well for this dataset. On the other hand, as CNN model uses convolution, it can use the relationships of nearby pixels to learn more complex features. Therefore, even though the deformation of object or the different in positions of object in different training images, the CNN model can still be able to detect the features of each object and use to classify testing instances.
    - For Fashion MNIST dataset, each image for each class has similar form, shape, and position (all flatten, positioned at center of all images, similar size, and grayscale image). Therefore, the kNN model can work well in this dataset because the differences between corresponding pixel between testing image and each training image would be small for object in the same class and therefore the distance formula performs much more accurate compare to CIFAR_10 dataset. CNN model always better because of its characteristic (discussed above).
    - Generalize, if the client only want to classify labels for images with the similar form, shape, and position for each class, kNN model maybe a good model enough. Otherwise, if the client want to classify real world images that have distraction by background, deformation, shapes,..., then CNN model would be a better choice of model.
    - Even though CNN model is still better, it requires much more deeper knowledge and more complex implementations.
    - kNN model can be described as lazy learning since it can predict images instantly without training process. This may be better compare to CNN model if the clients just want to classify several images instantly.

- On the other hand, CNN model takes initial training time but it will predict much faster after training process. Therefore, if the clients want to classify a huge number of testing instances, CNN model might be better.
- kNN model maybe more flexible because if we want to add several more classes, we do not need to change anything from the current model and the training time will not be different (if we add more classes but keep the number of images be the same).
- Contrarily, CNN model needs to train the whole model again if the clients add more classes to the dataset.

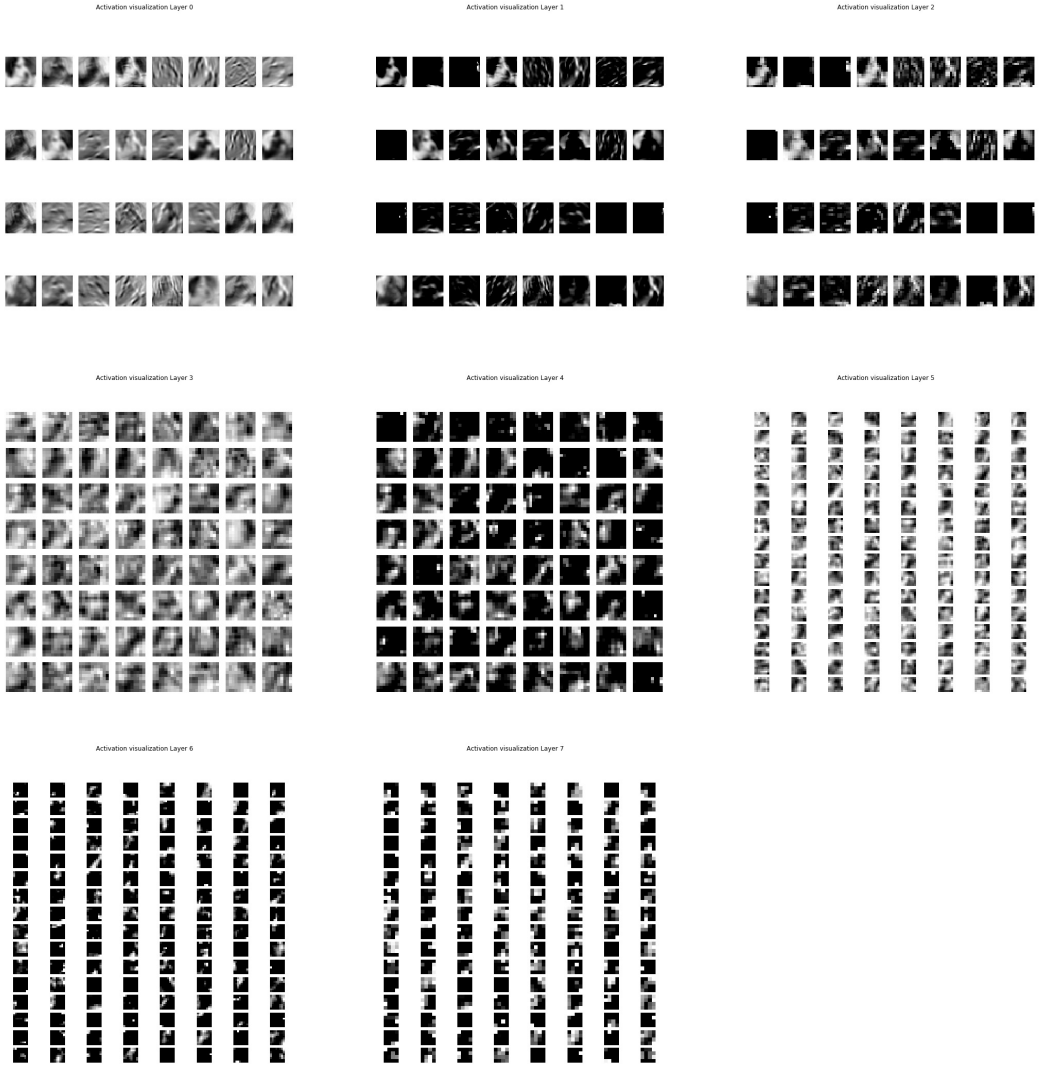
2. What does the model learn in each convolutional layer?

   Note: Here we only use the CIFAR_10 dataset

- **Conclusions**:
  - The deeper the layers, the more abstract the model learns. As we visualize the filters from our model after training, we got the result:

  Using the images above as reference, we can see that in the first few layers, it is easier to detect the features the model learns compare to the layers after that. For example, in the first layer we can see many filters with bright pixels form lines and in the second layer we can see bright pixels form curves,...)

- If we apply the filters from the model on a image (testing data, image 16th, which is a photo of a dog):



  As we apply the filters to a particular photo, in this case is the photo of a dog, we can see that different images, which represents different filters apply to activations, have different structures. Each image has a characteristic that is different to others. For instance, in layer 0 and 1, we can see a few activations with the edges of the dog, a few with the nose,...) The more it get deeper, the more black pixels we can see in each image, and the bright pixels seem to arange as group. This can be reasoning that the model learns features from local positions of the images (ear, nose, ... of a dog's image in this example). There are also a few images with every pixels are black, which mean the model learns nothing by applying a filter to the image to get that particular activation (dead filters).

- **Discussions**: By visualizing the filters and activations, we can understand more about the model we use. Visualizing filters and activations allows us to diagnose our model if it has any problems (slowly decreasing loss,...). We can use the visualizations to analyze how our current model works and therefore improve our model (For example, if there are many dead filters, we need to change the hyperparameters).

Reproducing Results:

NOTE:

- We are using seed while shuffling data to make the data indexes consistence for each run. However, as the training for CNN model is done by using Keras Sequential model, with SGD as optimizer and drop-out layer, it is reasonable to have a small different between each run. Even though there will be some differences, the results will not be affect much because the differences are negligible.
- There are many similarities between the codes for two datasets. However, the codes are not combined on purpose. As each dataset has different characteristics, these datasets may need different configurations to have better predictions. At this period, as we are currently focusing on comparing two datasets, we are setting them similar to each other. Seperating code files for each dataset as different files allows us to improve our model for each dataset easier in the future.

Install Neccessary Libraries:

The codes are built with Python 3.

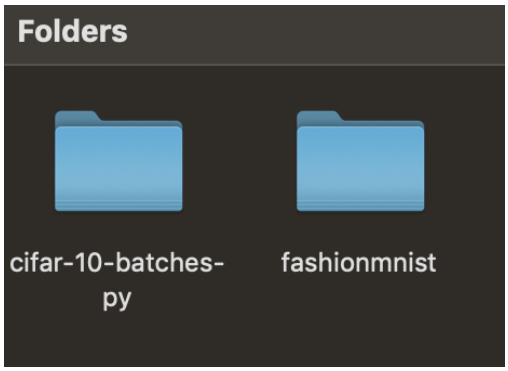
To run the codes for the project, the client needs to install the libraries below:

- numpy
- sklearn
- pandas
- matplotlib
- seaborn
- Tensorflow (to run Keras)
- Keras

The client should create a new environment and install all libraries above in that environment. All the codes for the project needed to be run using that environment.

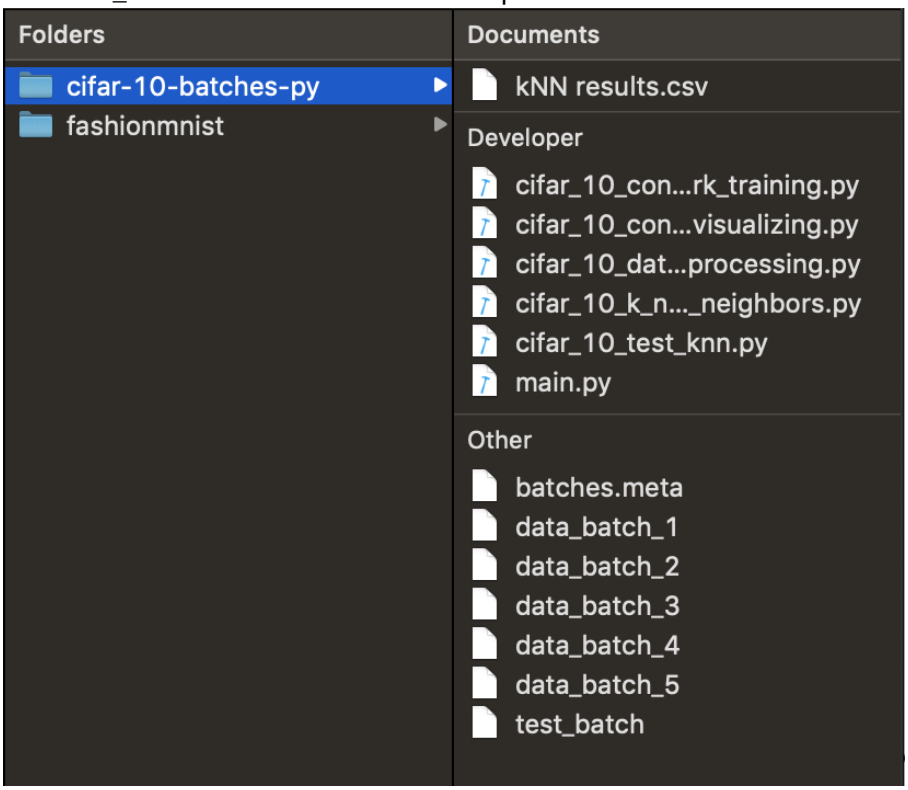
Download the datasets:

- CIFAR_10 dataset: The client can download the CIFAR_10 dataset by clicking the link https://www.cs.toronto.edu/~kriz/cifar.html (https://www.cs.toronto.edu/~kriz/cifar.html) then choose the version 'CIFAR-10 python version' and download it or directly download here: https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz (https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz).
- Fashion MNIST dataset: The client can download the Fashion MNIST dataset by clicking the link https://www.kaggle.com/zalando-research/fashionmnist (https://www.kaggle.com/zalando-research/fashionmnist) and click the download on the upper right to download the dataset as a zip.
- After downloading two dataset above, the client need to unzip each dataset and keep them separately as two folder. The result would be similar to this:



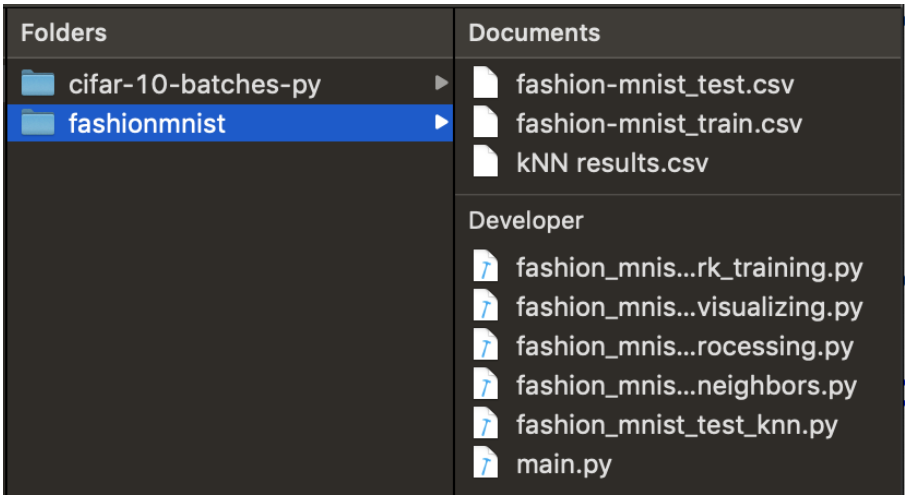
Download the code files and supporting files for the project and store in datasets' folders:

- The code files and supporting files will be submitted with the project. In addition, the client can directly download all codes and supporting files as a zip with the link below: https://drive.google.com/file/d/1M3FUn9xzLqQPFoxcdqTscvFMn6iNZHCh/view?usp=sharing (https://drive.google.com/file/d/1M3FUn9xzLqQPFoxcdqTscvFMn6iNZHCh/view?usp=sharing).
- After getting all the code files, the client needs to move the code files to the dataset's folder above so as to make the code run. To access the files, the client needs to get to the 'CODES' directory and selects the directory corresponding to each dataset:
  - For CIFAR_10 dataset, there are 6 codes file, including one main file to run all the code and one test file to test the kNN code. In addition, there is one csv file called 'kNN results.csv', which is the predicted result using kNN model for CIFAR_10 dataset. The client needs to move all the code files and the csv file to the folder of CIFAR_10 dataset after unzip the dataset. The CIFAR_10 dataset folder after this step would look similar to this:



  - For Fashion MNIST dataset, there are 6 codes file, including one main file to run all the code and one test file to test the kNN code. In addition, there is one csv file called 'kNN results.csv', which is the predicted result using kNN model for Fashion MNIST dataset. The

client needs to move all the code files and the csv file to the folder of CIFAR_10 dataset after unzip the dataset. The Fashion MNIST dataset folder after this step would look similar to this:



Run the whole project:

This section provides direction to run the whole project at once. The current codes for predicting the datasets using kNN are commented out because the kNN training part of both datasets takes a lot of time (about 12 hours with GPU on Google Colab/dataset). We provide a predictions result (predicted previously) for that part so it will not affect the project.

If the client wants to re-predict using kNN model, the next section provides several directions to use the KNN model to re-predict the testing instances.

CIFAR_10 dataset:

To run the whole analysis for the CIFAR_10 dataset, after completing previous steps (install libraries, download dataset, download codes, move code files and kNN result csv file to datasets' folders), the client only needs to run the analysis by running the 'main.py' file in the folder for CIFAR_10 dataset with the environment the client created and installed all required libraries.

Fashion MNIST dataset:

To run the whole analysis for the Fashion MNIST dataset, after completing previous steps (install libraries, download dataset, download codes, move code files and kNN result csv file to datasets' folders), the client only needs to run the analysis by running the 'main.py' file in the folder for Fashion MNIST dataset with the environment the client created and installed all required libraries.

Specific Results Per Research Question:

In this part, we are providing directions to running the analysis for each research question.

Preprocessing data:

Note: This is the most important part if the client wants to run each python files separately. The data preprocessing part allows the client to create a directory called 'Models' and exports all necessary data in that directory to use in all other supporting files.

- CIFAR_10 dataset: to preprocess the data for CIFAR_10 dataset, the client needs to run the file 'cifar_10_data_preprocessing.py'. The file will preprocesses the data and create a new folder named 'Models' in the current directory and add all the data in that folder.
- Fashion MNIST dataset: to preprocess the data for Fashion MNIST dataset, the client needs to run the file 'fashion_mnist_data_preprocessing.py'. The file will preprocesses the data and create a new folder named 'Models' in the current directory and add all the data in that folder.

Note: For all sections below, the client needs to finished the Preprocessing data part first, which means that for each dataset, the coresponding dataset's folder needs to include a directory named 'Models' that included the file below:

```
+ label_names.npy
+ x_train.npy
+ x_val.npy
+ x_test.npy
+ y_train.npy
+ y_val.npy
+ y_test.npy
```

Part 0: Using k-Nearest-Neighbors approach:

1. How accurate the model classify and does the model predict outputs with high confidence (that is the probability of confidence is high)?

    - CIFAR_10 dataset: to run the kNN code for the CIFAR_10 dataset, the client needs to run the file 'cifar_10_k_nearest_neighbors.py'. As running the whole program will takes a lot of time to use the model predicting testing instances, we already done the training part and export to a file named 'kNN results.csv'. By moving the supporting files to the dataset's folder as above, the client already has this file in the dataset directory.
        - Note: If the client want to re-predict the testing instances, the client can remove the comment signs for the part of using model to predict testing instances in the main method of the file 'cifar_10_k_nearest_neighbors.py':

        ```
        243     '''
        244     print('Testing model')
        245     testing_result = predict(x_train, y_train, x_test, y_test, label_names)
        246     testing_result.to_csv('kNN results.csv')
        247     '''
        ```

        Now, by running the 'cifar_10_k_nearest_neighbors.py', the client can run the predicting process.
    - Fashion MNIST dataset: to run the kNN code for the Fashion MNIST dataset, the client needs to run the file 'fashion_mnist_k_nearest_neighbors.py'. As running the whole program will takes a lot of time to use the model predicting testing instances, we already done the training part and export to a file named 'kNN results.csv'. By moving the supporting files to the dataset's folder as above, the client already has this file in the dataset directory.
        - Note: If the client want to re-predict the testing instances, the client can remove the comment signs for the part of using model to predict testing instances in the main

method of the file 'fashion_mnist_k_nearest_neighbors.py':

```
243     '''
244     print('Testing model')
245     testing_result = predict(x_train, y_train, x_test, y_test, label_names)
246     testing_result.to_csv('kNN results.csv')
247     '''
```

Now, by running the 'fashion_mnist_k_nearest_neighbors.py', the client can run the predicting process.

- **Results**:
  - The results for how accurate the model classify will be printed out to show the percentage of correct predictions. The program also provides several examples to verify the predictions as 8x8 images of testing instances. The visualization will then be saved as a png file.
  - The results for does the model predict outputs with high confidence will be showed as 10 plots of distribution of probabilities for each label in the dataset for each testing instances (using 10 testing instances as examples). The results will then be saved as png files.
- **Testing kNN codes**:
  - Note: In this part, even though the code files for each dataset are similar, they still have several modifications to fit the characteristic of each dataset. Therefore, to ensure we do not make any error, we still need to test each code file for each dataset. Currently, although the codes for testing each dataset are similar, we still organize them separately for future modifications.
  - CIFAR_10 dataset: To test the codes written for kNN model, we have provided a file called 'cifar_10_test_knn.py'. To run the test, the client needs to move the test file to the CIFAR_10 dataset folder with the other code files then runs the main method of this file.
  - Fashion MNIST dataset: To test the codes written for kNN model, we have provided a file called 'fashion_mnist_test_knn.py'. To run the test, the client needs to move the test file to the Fashion MNIST dataset folder with the other code files then runs the main method of this file.

Part 1: Using Convolutional Neural Network approach:

1. How accurate the model classify and does the model predict outputs with high confidence (that is the probability of confidence is high)? How does CNN model work in each dataset compare to kNN model?

   - CIFAR_10 dataset: To run the CNN initializing and training code for the CIFAR_10 dataset, the client needs to run the file 'cifar_10_convolutional_neural_network_training.py' in the CIFAR_10 dataset folder.
   - Fashion MNIST dataset: To run the CNN initializing and training code for the Fashion MNIST dataset, the client needs to run the file 'fashion_mnist_convolutional_neural_network_training.py' in the Fashion MNIST dataset folder.
   - **Results**:
     - The results for how accurate the model classify will be printed out to show the percentage of correct predictions. The program also provides several examples to verify the predictions as 8x8 images of testing instances. The visualization will then be saved as a png file.

        - The results for does the model predict outputs with high confidence will be showed as 10 plots of distribution of probabilities for each label in the dataset for each testing instances (using 10 testing instances as examples). The results will then be saved as png files.
        - The results for how does CNN model work in each dataset compare to KNN model will be interpreted by the accuracy of correct predictions and the layout of images in each dataset.
2. What does the model learn in each convolutional layer?

    - CIFAR_10 dataset: To run the CNN visualization code for the CIFAR_10 dataset, the client needs to run the file 'cifar_10_convolutional_neural_network_visualizing.py' in the CIFAR_10 dataset folder.
        - Note: The above part gives directions to re-train the CNN model. This will gives a small different between the visualizations of each run. Even though this will not affect the results much, if the client want exact results as the 'Results' part, we also provide a file called 'cnn_model' in the 'Model and Data/CIFAR_10' directory. The client can comment out the 'cifar_10_convolutional_neural_network_training.py' in the 'main.py' file and copy the 'cnn_model' file to the 'Models' folder of the CIFAR_10 dataset folder. By doing this, the client can then run the visualization part again and will get the exact results as the 'Results' part.
    - Fashion MNIST dataset: To run the CNN visualization code for the Fashion MNIST dataset, the client needs to run the file 'fashion_mnist_convolutional_neural_network_visualizing.py' in the Fashion MNIST dataset folder.
        - Note: The above part gives directions to re-train the CNN model. This will gives a small different between the visualizations of each run. Even though this will not affect the results much, if the client want exact results as the 'Results' part, we also provide a file called 'cnn_model' in the 'Model and Data/Fashion MNIST' directory. The client can comment out the 'fashion_mnist_convolutional_neural_network_training.py' in the 'main.py' file and copy the 'cnn_model' file to the 'Models' folder of the Fashion MNIST dataset folder. By doing this, the client can then run the visualization part again and will get the exact results as the 'Results' part.
    - **Results**: Running the program described above will generate several plots. The results will be interpreted by analyzing the plots generated by the codes.

Work Plan Evaluation:

Data Preprocessing:

Time estimate: 1 hour - 2 hours

Evaluation: This part is accurate. As the two datasets have the clear layouts, I did not need to spend a lot of time on data preprocessing part.

In this part, we will try to pre-process 2 datasets to have a similar shape of inputs so that it will be much easier for further parts. Therefore, even though the codes might not really be optimized for each individual dataset, it would be convenient for the future's modifications.

Fashion MNIST dataset:

- After download the dataset by the link provided in "Dataset" section, we need to get the training data from 'fashion-mnist_train.csv' and testing data from 'fashion-mnist_test.csv'.
- Read training data and testing data to DataFrames using Pandas and divides the data to input columns (x) and output columns (y) by the column 'label'.
- For consistence with CIFAR_10 and so as to fit in Keras Sequential, put both training data and testing data to numpy array then reshape each image in each row to (number of images, 28, 28, 1) then divides all pixel values by 255 so that the value of each pixel will be from 0 to 1. We can understand (28, 28, 1) represents an image with 28x28 pixels and 1 channel.
- Split the training data to smaller training data and validation data by ratio (80:20).
- Define a list store labels name and add all labels into the list so that the index of each label correspond to the index of that label in the dataset's output.

CIFAR_10 dataset:

- After download the dataset by the link provided in "Dataset" section, we need to get the training data from 'data_batch_1' to 'data_batch_5' and testing data from 'test_batch' by using the 'unpickle(file)' function from the link below: https://www.cs.toronto.edu/~kriz/cifar.html (https://www.cs.toronto.edu/~kriz/cifar.html).
- Put all image in $['b'data']$ column to numpy array then reshape it to (number of images, 3, 32, 32) then use numpy.transpose to permute the color channel as (1, 2, 0) so as to get the shape (number of images, 32, 32, 3) with correct color channel. Then divides all pixel values by 255 so that the value of each pixel will be from 0 to 1.
- Split the training data to smaller training data and validation data by ratio (80:20).
- Define a list store labels name and add all labels into the list so that the index of each label correspond to the index of that label in the dataset's output.

Answer Research Questions:

Part 0: Using k-Nearest-Neighbors approach

Time estimate: 24 - 48 hours (because of training kNN requires a lot of time - about 120000 training images in the two datasets (incuded validation data))

Evaluation: This part took me a lot longer time because after each modification, I need to re-run the prediction part. With the fact that it takes about 24-48 hours to finish the predicting part, I need much longer time to complete this part with a good result. I also deleted the second research question in this part because it is overlap with the first research question in part 0 and part 1.

1. How accurate the model classify and does the model predict outputs with high confidence (that is the probability of confidence is high)?

    - Define core function for both model:
        - Distance function: use numpy to compute the distance function by the formula given above.
        - Define a function to get k-nearest neighbors: call distance function for each instance in training data to compute the distance between that instant and the current testing

image.
    - Define a function to get the score of k-nearest neighbors: Use a dictionary to store the number each distinct class appears in k-nearest neighbors then get the score of each label.
- Run through all testing instances and predicts the label based on the given function. Store the result to a DataFrame for further visualization.
- Visualize several images predicted before using matplotlib.
- To answer how accurate the model classify the outputs, get the accuracy of correct predictions for all testing data in each dataset.
- To compare the probability of each class appear in k-nearest neighbors, prints out the score achieved from the function we used to get the score then compare them.

Part 1: Using Convolutional Neural Network approach

Time estimate: 48 - 96 hours

Evaluation: This part took me longer time because I need to re-train the models for each dataset several time to avoid overfitting by keeping track the rate of validation loss decreasing. As each time of training takes a lot of time (5-10 minutes with GPU or 30-45 minutes with CPU), I spent a lot of time on this part to get an accurate result.

1. How accurate the model classify and does the model predict outputs with high confidence (that is the probability of confidence is high)? How does CNN model work in each dataset compare to kNN model?

    - Use Keras Sequential model to implement a Convolutional Neural Network for both dataset. The only different between 2 dataset is the input shape for each image of Fashion MNIST dataset is (28, 28, 1) and for CIFAR_10 dataset is (32, 32, 3) and so they will have different number of parameters after each convolutional layers.
    - Add all layers described above to the CNN model using model.add($[layer]$).
    - Compile the model then fit the model with training data and validate with validation data.
    - To answer how accurate the model classify the outputs, get the accuracy of correct predictions for all testing data in each dataset.
    - To answer does the model predict outputs with high confidence, use model.predict() with keyword argument 'verbose=1' to show the probability of each class that the model predict for several images.
2. What does the model learn in each convolutional layer?

    - To visualize filters of a convolutional layer:
        - Get all weights in a particular convolutional layer
        - Use matplotlib to plot all filters achieved above as a single figure.
    - To visualize outputs after applying filters after a convolutional layer:
        - Use Keras.Model.model to get a model with input is the input image and output is a convolutional layer (if want to visualize the output after apply an activation after convolutional layer then select the layer after applying activation).
        - Use model.predict to get the output of the interested output layer we need to visualize.
        - Use matplotlib to visualize the output achieved above.
    - Compare the filters plot and outputs plot by looking at bright pixels in the plots to give conclusion.

Discussion about the models:

Time estimate: 12 hours

Evaluation: This part took me a longer time to finish because after visualizing the results generated above, I need to look closer to the visualizations and understand how these visualizations represent.

- Look back all analysis to raise discussion about limitation or configuration of the models in the project.
- Try several ways of tuning hyperparameters.

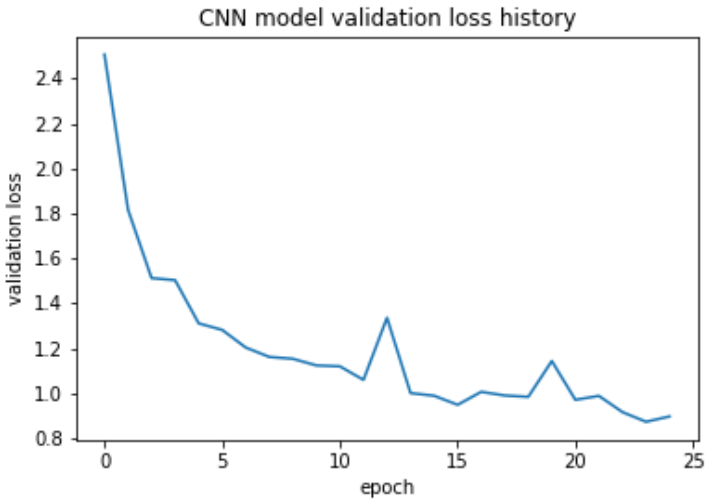
Testing:

PART 0: Using k-Nearest Neighbors Approach:

- We defined functions to predict results in this part without using libraries. Therefore, we need to test our codes to verify the correctness. With that reason, we use assert statements with small examples that are easy to verify by hand for each function to verify the correctness of these functions. All tests for this part is located in 'cifar_10_knn_test.py' and 'fashion_mnist_knn_test.py', which are provided in the zip file contains all code files. As we can verify the testing examples by hand, we can verify the correctness of the functions in this part and therefore can trust the results that are provided by these codes.
- To run the test for kNN for CIFAR_10 dataset, move the 'cifar_10_knn_test.py' to the same folder of the CIFAR_10 dataset after unzip the dataset. Then run the main method of that test file and all the test will be run immediately, as described in the 'Reproducing Results' part.
- To run the test for kNN for Fashion MNIST dataset, move the 'fashion_mnist_knn_test.py' to the same folder of the Fashion MNIST dataset after unzip the dataset. Then run the main method of that test file and all the test will be run immediately, as described in the 'Reproducing Results' part.

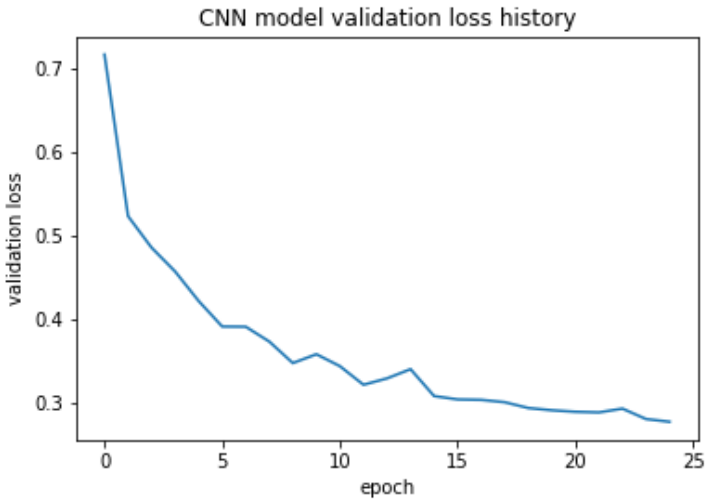
PART 1: Using Convolutional Neural Network Approach:

- **CIFAR_10 dataset**:
  - Overfitting Test: As we plot a graph of history of fitting model with our training data and validate with validation data, we understand the loss in validation data in our model is decreasing throughout training process and just slightly increase at the final epochs. Our final history of validation data loss shows that we are stopping when the validation loss is decreasing very slowly and as experimenting while working on the project, the validation loss would start to increase after a few epochs if we continue to fit the model. As we stop at the moment the loss of validation data start increasing, we can confirm our model is not overfitting.



- Data Input Test: To make sure everything we have done in this part is correct, we need to test the inputs. Therefore, in the main method of 'cifar_10_convolutional_neural_network_training.py', we used assert statements after loading data in CNN training file to verify the shapes of the data so that it will not give unexpected behaviors during training and visualizing process. As the program passed the assert statements, we understand that our inputs are the same thing that we expected.
- Data Output Test: After training our model and using that to predict results, we need to verify that the predictions are accurate. Therefore, we need to visualize several examples to see the trend of predictions and compare with the correct accuracy to verify our process (plots on 'Results' part). As the examples we use in visualization are randomly shuffled (as the dataset's source provides that the data are randomly shuffled), we can compare the trend in the visualization to the model's prediction accuracy. By comparing the accuracy of the visualizations to the correct predicting accuracy, we can verify our program works correctly and we can trust the prediction's accuracy of the model.

- **Fashion MNIST dataset**:
  - Overfitting Test: As we plot a graph of history of fitting model with our training data and validate with validation data, we understand the loss in validation data in our model is decreasing throughout training process and just slightly increase at the final epochs. Our final history of validation data loss shows that we are stopping when the validation loss is decreasing very slowly and as experimenting while working on the project, the validation loss would start to increase after a few epochs if we continue to fit the model. As we stop at the moment the loss of validation data start increasing, we can confirm our model is not overfitting.



- Data Input Test: To make sure everything we have done in this part is correct, we need to test the inputs. Therefore, in the main method of 'fashion_mnist_convolutional_neural_network_training.py', we used assert statements after loading data in CNN training file to verify the shapes of the data so that it will not give unexpected behaviors during training and visualizing process. As the program passed the assert statements, we understand that our inputs are the same thing that we expected.
- Data Output Test: After training our model and using that to predict results, we need to verify that the predictions are accurate. Therefore, we need to visualize several examples to see the trend of predictions and compare with the correct accuracy to verify our process (plots on 'Results' part). As the examples we use in visualization are from the test set, we can use the assumption that the author of the dataset has already shuffled the test set to publish a good test set. Therefore, we can compare the trend in the visualization to the model's prediction accuracy. By comparing the trend of the visualizations to the correct predicting accuracy, we can verify our program works correctly and we can trust the prediction's accuracy of the model.