Assignment 2 – Group 114

Tutors: Tahsin Samia, Siwen Luo
Group members: Yichen Chen(yche3494), Long Cheng(lche4115)

1 Abstract

Both Cifar-10 and Cifar-100 are datasets collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton for object recognition and classification. CIFAR was founded in 1982 as the Canadian Institute for Advanced Research and brought together top scientists in various fields worldwide to solve complex problems like deep learning. This report will briefly introduce this data set, compare it with similar data sets, and then use a series of pre-processing and machine school methods to calculate three different machine learning models. Through the accuracy and cross-validation estimates of the three models, compare the performance of other models. Finally, the report also discusses possible improvements to the model and some suggestions for further research in the future.

2 Introduction

In this assignment, we decided to choose CIFAR-100 as our research project. This dataset contains 100 classes and 600 images in each class. Looking each class includes 500 training images and 100 testing images. In the meantime, the 100 classes in the dataset are grouped into 20 super-classes, which contributes the 'coarse' label to the image. The other label name 'fine' belongs to the class. Totally, the CIFAR-100 dataset is made up of 60000 32*32 color images and also has 50000 training images and 10000 test images.
Nowadays, portrait recognition technology has matured, but there are still great challenges for object recognition because the data set contains many different features, noises, and objects of different proportions. If objects can be identified quickly and accurately, it will help the deep learning of future machines.
Briefly, for KNN and SVM models, we utilize 'minmaxscale' as our first pre-processing method, which can help with accelerating gradient descent, helping convergence, and improving accuracy. Since the size of each image in the data set is 32x32, there are three RGB channels. Following the RGB channel order and the order of the row, a training sample corresponds to a row with 32*32*3=3072 values. The next pre-processing step is principal component analysis (PCA), which helps dimensionality reduction and data compression. For CNN, we utilize 'ImageDateGenerator' to boost our data set as pre-processing. KNN and SVM are the most commonly used in machine learning. In KNeighborsClassifier, only the number of neighbors is changed, and the classification results of the optimal K value of each data set are stored. The main focus of this report is to study CNN, which is very good at image processing. CNN will be divided into many parts in the method section to discuss the convolution layer, activation layer, batch normalization, pooling, dropout, weight decay, and SGD.

3 Previous work

We have studied this literature[1], "Big Transfer (BiT): General Visual Representation Learning," published in 2020 (Alexander Kolesnikov, 2020). The literature expanded the pre-training and proposed a simple method called Big Transfer (BiT). By combining some selected components,

starting from the three variables of data set: model size, model structure, and model training super parameters. That experiments aim to prove: pre-train on a large supervised source dataset and fine-tuning the weights on the target task.
The tasks for fine-tuning are ILSVRC-2012， CIFAR-10/100, Oxford-IIIT Pet, etc. And the training models are deep residual networks (ResNet). Through the literature, it can be concluded that combining 'GroupNorm' (GN) and weight standardization as 'GroupNorm' will benefit upstream pre-processing. Moreover, methods as downlinking resizing and random cropping of details are also important. If the image is larger, we will adjust it to a larger fixed size to benefit from fine-tuning at a higher resolution.
Compared to our works in this assignment, there exist main differences:
1) The ResNet network is based on the VGG19 network, modified based on it, and added the residual unit through the short-circuit mechanism, which maintains the complexity of the network layer.
2) In the stage of pre-processing data, for random cropping of pictures, we only do horizontal flipping and rotation of images.
3) Use weight standardization in all convolutional layers.
4) Save training time, reduce memory usage, and improve model accuracy.

4 Methods:

4.1 Minmaxscale:

For the KNN and SVM model, the 'MinMaxScaler' has been imported from 'sklearn.preprocessing' can transform features by scaling each feature or the maximum absolute value of each value to a given range between zero and one. The formula is given by:

$$x_{\text{scaled}} = \frac{x - min(x)}{max(x) - min(x)}$$

The purpose of using the 'MinMaxScaler' is to robust features to very small standard deviations and preserve zero entries in sparse data. Moreover, the next step is PCA, which is also affected by normalization.

4.2 Principal components analysis (PCA)

The dataset is about high dimensional 'RGB images, so it is essential to use PCA to reduce the dimensionality of such datasets, increase interpretability, and minimize information loss to around 20 percentage. However, we have to apply explained variance ratio to decide how much information will be lost. From figure 1 below, approximate 50 components have to be kept to prevent the loss of the explained variance.

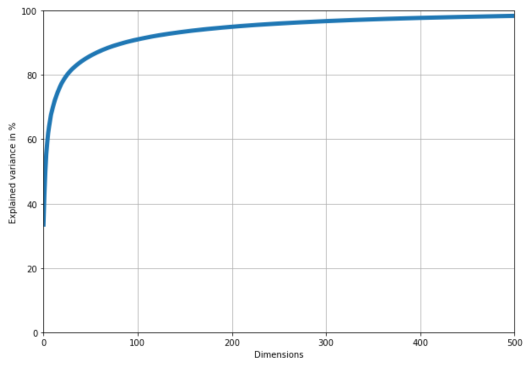


Figure 1: Explained variance ratio versus dimensions

Then we can set n_components=50 and execute PCA again. But there is a better option: instead of setting the number of principal components to be retained, set n_components to 0.8 as the amount of variance ratio we want to retain.

4.3 ImageDataGenerator

ImageDataGenerator is located in the keras.preprocessing.image module and can be used for data augmentation. It performs a set of specified transformations (rotation_range=30, width_shift_range=0.2, height_shift_range=0.2 horizontal_flip=True) on the original image, thereby constructing a new training set that includes the original image and its transformation. The ImageDataGenerator function is used with streaming functions that generate batch image data.

4.4 KNN

The KNN algorithm is a simple supervised machine learning algorithm that can be used to solve classification and regression problems. Its principle is very simple, but there is a major drawback, that is, as the scale of the data used increases, the speed will be significantly slower. The working principle of KNN is very simple: Given a test sample, find the k training samples closest to it in the training set based on a certain distance metric, and then make predictions based on the information of these k "neighbors" and vote for the most frequent label (In the case of classification) or the average label (in the case of regression).

The main reasons to choose the KNN algorithm:

1) The principle is simple and easy to implement.
2) The training time is shorter compared with the other algorithm like SVM.
3) No assumptions about the data, high accuracy, insensitive to outlier points.

More efficiency when dealing large sample size classification model.

4.5 SVM

Support Vector Machine is a supervised learning and non-parametric method. The decision boundary of SVM is maximum-margin hyperplane.

To define a hyperplane, we use formula like this:

$$w^T x + b = 0$$

And what is maximum-margin hyperplane? That means we have to find a hyperplane that has a maximum distance between different labels group (or sides, for example, blue and red sides below).

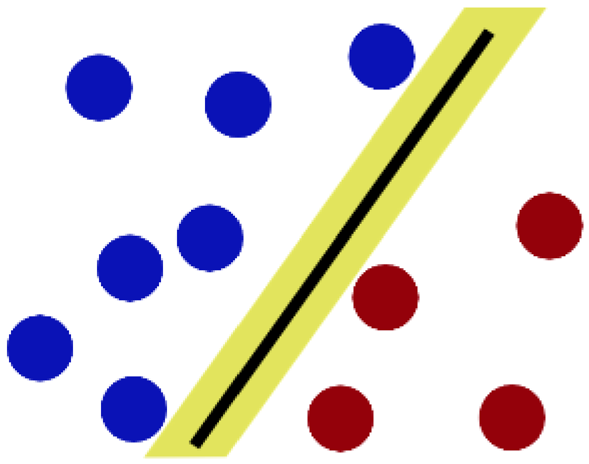


Figure2: maximum-margin hyperplane

What are support vectors? They are the points that exactly have the distance to maximum-margin hyperplane is d, where d is 1/2 maximum-margin.

For the point of the other, we have:

$$\begin{cases} \dfrac{w^T x + b}{||w||} \geq d & y = 1 \\ \dfrac{w^T x + b}{||w||} \leq -d & y = -1 \end{cases}$$

where y is the label.

For the convenience of calculation, we just set ||ω||·d = 1, so we have:

$$y(w^T x + b) \geq 1$$

The reason why we choose SVM is to compare with simple model KNN, as SVM is more complex.

For SVM, we design candidate hyper-parameters:

{'kernel' : ["rbf"], "C" : [5,10,15],"gamma":[0.001,0.01,0.1]}

We choose rbf kernel to try to solve this non-linear classification problem.

4.6 CNN

How and why we design our structure of CNN will be discuss in detail in part 5.5.2. In general, we designed such a structure aim to increase the operating speed and increase the accuracy.

4.6.1 Convolution layer

The convolution layer uses a convolution kernel1 continuously scans the pixel matrix on each layer, and the value scanned each time will be multiplied by the number in the corresponding position in the convolution kernel. Then add and sum, the values obtained will generate a new matrix. In other words, it first extracts every detail of the original image and then generates a new matrix (image) from the extracted details. The size of this detail depends on the size of the convolution kernel. And, how to define a particular kernel or filter? For 3x3 filter, we have 9 weights, and after one iteration, we just use backpropagation to update the weights to get lower loss.[2]

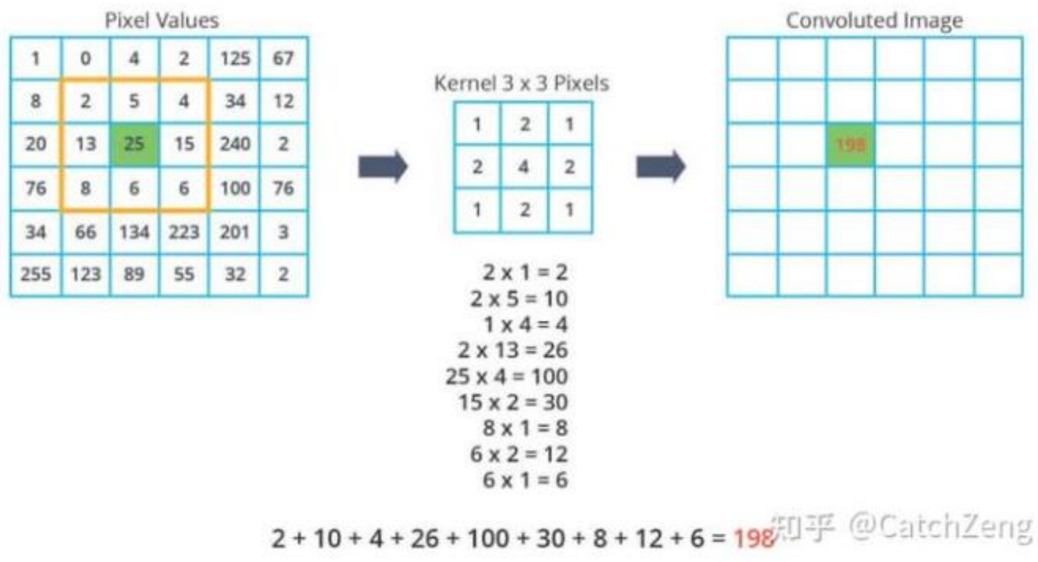


Figure 3: convolution layer

Convolution kernel1: also called a filter, the most common size are 3x3 and 5x5.

4.6.2 Activation layer

The activation layer is designed to increase the nonlinearity of a neural network. The most common activation functions are tanh, relu. A neural network without activation layer essentially is still a linear function, no matter how deep the depth is, the output is a linear combination of the input. In this case, we cannot expect neural network to perform better than the simple linear estimators. [3]

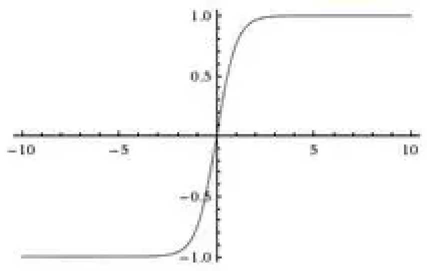


Figure4: tanh activation function

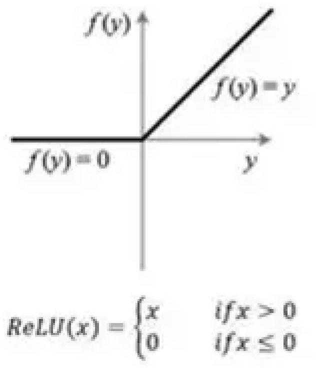


Figure 5: relu activation function

4.6.3 Batch Normalization

“Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout.” This Batch Normalization layer performs like a normalization pre-processing part, but it can apply for every convolution layers. The principle can simply conclude as, as mentioned above, after one convolution and activation functions, we just get a new matrix, for these numbers in matrix, some of them could be very large or small which need to be normalized, so we just apply this normalization layers for all convolution layers. [5]

4.6.4 Pooling

The pooling layer is a straightforward way to reduce the size of the image but retaining important features, in order to reduce the impact of overfitting. The following pictures are two ways to retain important features, the first is max pool, the second is average pool.

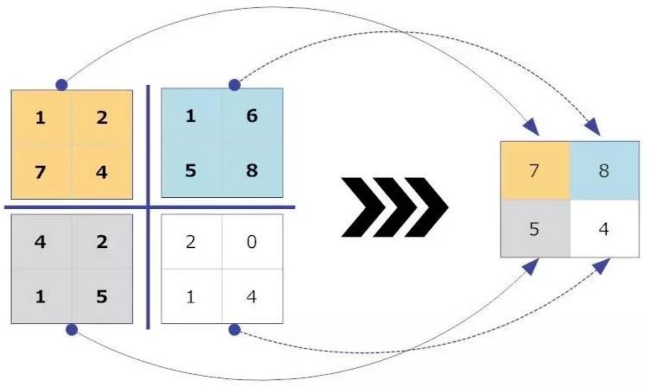


Figure 6: max pool

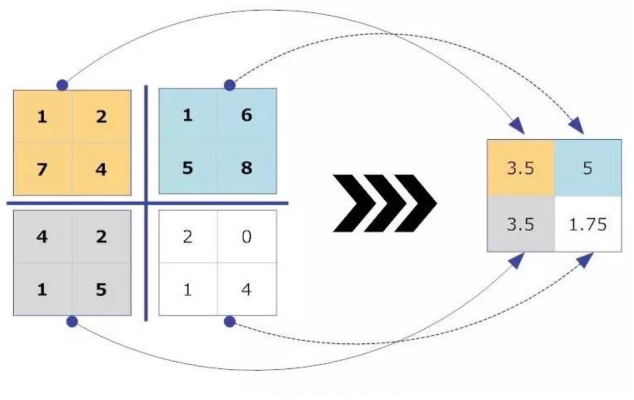


Figure 7: average pool [6]

4.6.5 Dropout layer

Another straightforward way is to reduce the impact of overfitting. It just randomly dropout features in a layer of neural network. Only one parameter probability to set how many features you want to dropout in this layer. For example, dropout (0.3) means the features in this layer have 30% probability to be dropout.

4.6.6 Weight decay

Another technic to prevent overfitting. Weight decay is a coefficient of the regularization term. If we have large value of weight decay, we will get higher loss if the neural network is complex. The larger value weight decay, the larger loss in same complexity.

4.6.7 SGD

In fact, it is Mini-batch gradient descent in keras package. Set the batch size from 1 to the n_samples, for examples, we choose 100, then it chooses 100 samples in a batch, for every batch, we update the weight once.

$$g_t = \nabla_{\theta_{t-1}} f(\theta_{t-1})$$

$$\Delta\theta_t = -\eta * g_t$$

Where η is the learning rate, g_t is the gradient.

5 Experiments:

5.1 Basic description:

CIFAR100 data contains 60000 rows in total, 50000 for training and 10000 for testing. For simple classifiers like KNN SVM, we use cross-validation to average the validation accuracy to improve our estimators by tuning parameters, use test data to evaluate final model via evaluation metrics (accuracy, precision, recall) and confusion matrix. For CNN model, we split training data into validation data and training data with proportion 90% for training and 10% for valid. Use this validation data to evaluate the loss and accuracy of current network by plotting the line chart of loss and line chart of accuracy to determine whether continue to decrease learning rate and proceed next stage of training (in the case they do not converge).

5.2 Hardware specifications:

Processor: Intel Core i5 10400 2.9Ghz
RAM: 16GB 2667Mhz
GPU: NVIDIA RTX2060 6G

5.3 Software specifications:

Environment: Python3 Jupyternotebook
Packages: Pandas 1.1.3; NumPy 1.19.2; Scikit - Learn 1.0; TensorFlow – gpu 2.2.0

5.4 Runtime:

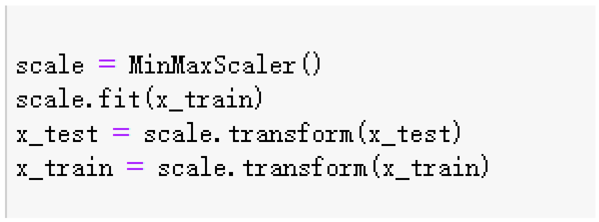
Scale data: 1.5 s
PCA: 73.4 s
KNN: Tuning parameters 427.3 s. Model fitting 0.2 s.
SVM: Tuning parameters 5168.0 s. Model fitting 128.3 s.
CNN: (101 s + 65 s + 41 s + 31 s +26 s) * 50 = 13200 s, for each learning rate with different batchsize run 50 epochs.

5.5 Process and results:

5.5.1 KNN and SVM:

Preprocessing:

We apply MinMaxScale to standardize data.



```
scale = MinMaxScaler()
scale.fit(x_train)
x_test = scale.transform(x_test)
x_train = scale.transform(x_train)
```

After that we use PCA to reduce the dimensions from 3072 to 26 to retain 80% of the variance.

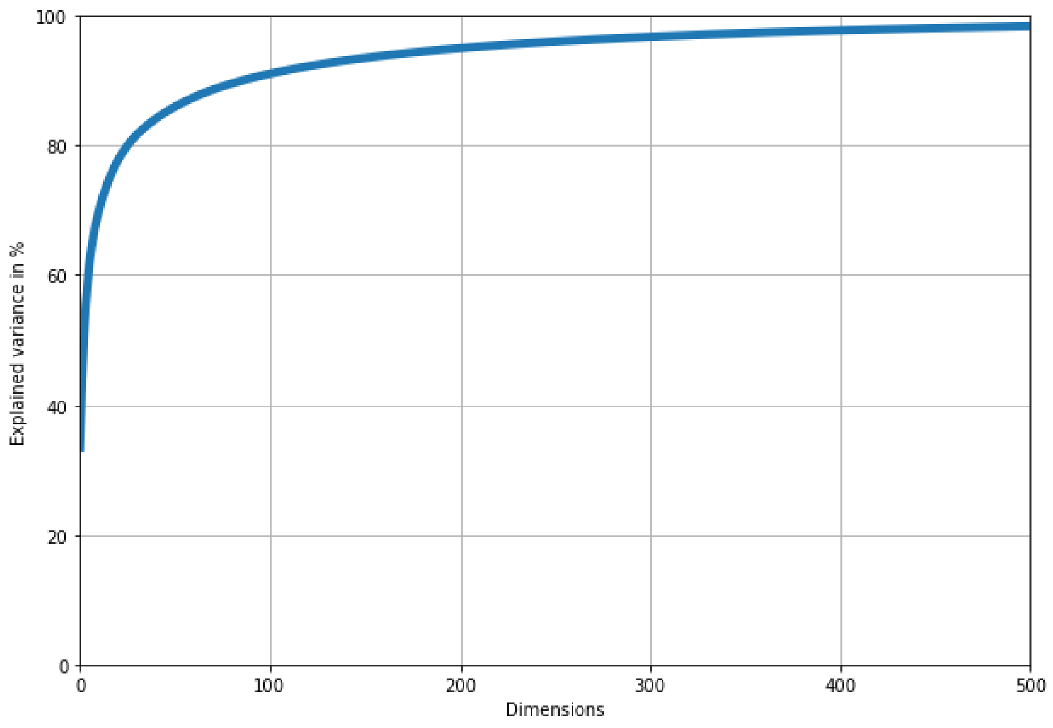
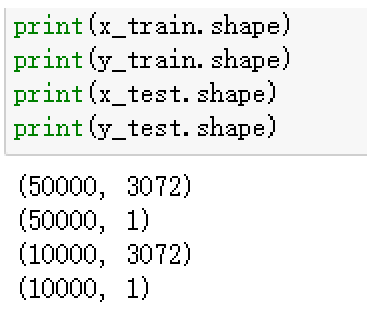


Figure7: PCA

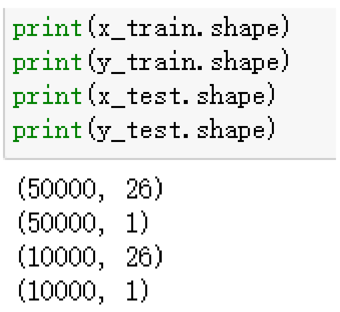
Before:



```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(50000, 3072)
(50000, 1)
(10000, 3072)
(10000, 1)
```

After:



```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(50000, 26)
(50000, 1)
(10000, 26)
(10000, 1)
```

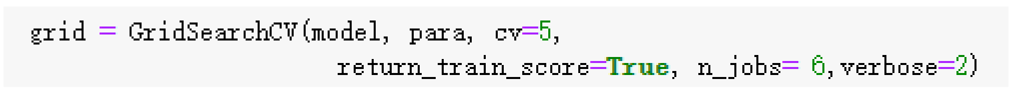
We use 5 – fold cross- validation to find the best hyper-parameters of these models. The candidate hyper-parameters we choose for KNN is:

{'n_neighbors': [5,10,15,20,25,30,35]}

The candidate hyper-parameters we choose for SVM is:

{'kernel' : ["rbf"], "C" : [5,10,15],"gamma":[0.001,0.01,0.1]}

We use n_jobs = 6 to apply parallelizing to speeding – up our code:



```
grid = GridSearchCV(model, para, cv=5,
                    return_train_score=True, n_jobs= 6,verbose=2)
```

Then we find that the best parameter or combination for KNN and SVM is:

KNN: KNeighborsClassifier (n_neighbors=30) with validation accuracy 0.17 and training accuracy 0.25.

SVM: SVC (C=5, gamma=0.01) with validation accuracy 0.25 and training accuracy 0.76.

Then we use these two estimators to predict the labels of test data and import classification report from sklearn.metrics to evaluate these two estimators by accuracy, average precision and recall:

```
KNN version evaluation:                      SVM version evaluation:

Accuracy: 0.1839                             Accuracy: 0.2659

Precision: 0.20468359364468694               Precision: 0.2636055170633384

Recall: 0.18389999999999998                  Recall: 0.26589999999999997
```

For confusion matrix, consider there are 100 labels in this task which is extremely difficult to show the details by using normal confusion matrix table, we decide to visualize this table via heatmap, and here are the results:

KNN version:

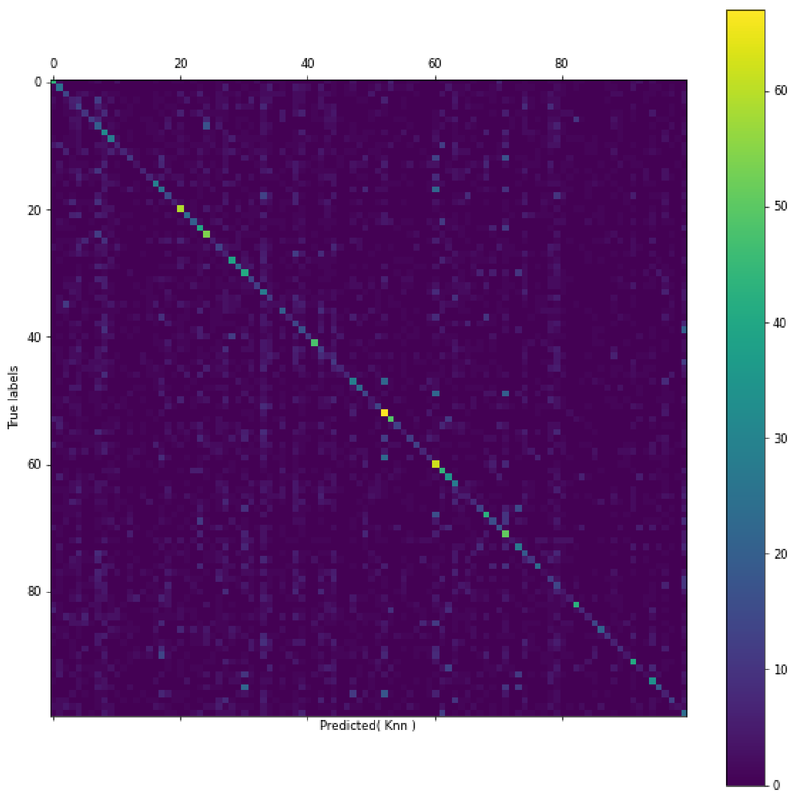


Figure8: confusion matrix of KNN

SVM version:

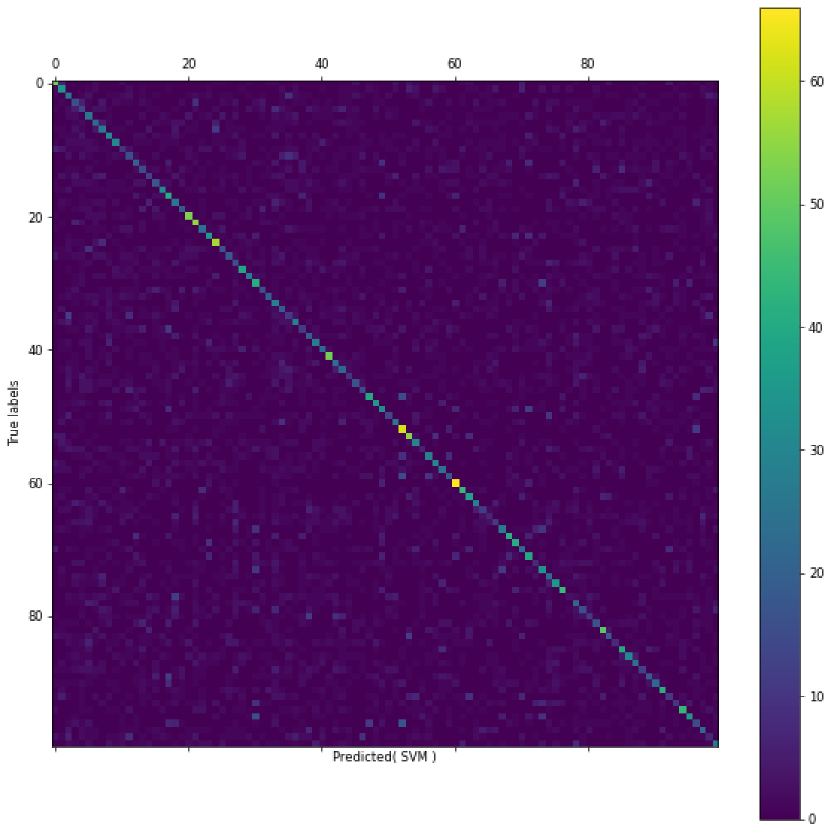
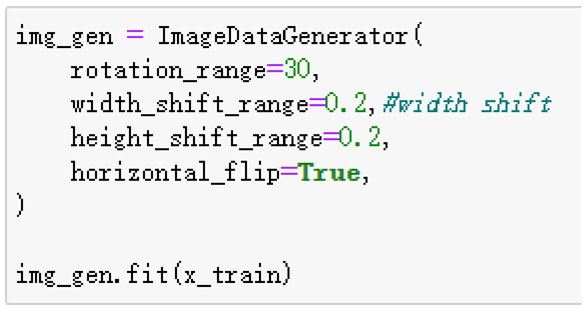


Figure9: confusion matrix of SVM

5.5.2 CNN:

5.5.2.1 Our effort to improve accuracy:

We try to use ImageDataGenerator to do data augmentation.



```
img_gen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,#width shift
    height_shift_range=0.2,
    horizontal_flip=True,
)

img_gen.fit(x_train)
```

We set rotation_range = 30, which will make image randomly rotate in 30 degree. And we set width and height shift. Then we hope we will get some mirror flip image, so we finally set horizontal_flip = True.

According to the previous report or literature of VGG16 work on the similar data set cifar – 10 and previous work of alexander 2020 in cifar – 100, we judge that we need to increase the complexity of the CNN structure since we are facing a 100 labels' dataset rather than 10 labels. Thus, we refer to the VGG16 network structure and add two convolutional layers, two "relu" activation layers, Batch Normalization layers for each convolutional layers and dropout layers (dropout 0.3 and 0.4) for each convolutional layers to increase the complexity to capture more details of the dataset.[4]

```
model.add(layers.Conv2D(512, (3, 3), padding='same',kernel_regularizer=regularizers.l2(weight_decay)))
model.add(layers.Activation('relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.4))

model.add(layers.Conv2D(512, (3, 3), padding='same',kernel_regularizer=regularizers.l2(weight_decay)))
model.add(layers.Activation('relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.3))
```

(Additional layers)

And we also find that if we modify the final dropout layer's parameter from 0.5 to 0.3, it may perform better in validation accuracy. We decide to do this modification because in the training history we find that training accuracy and validation accuracy is close enough, which indicate no overfitting. So we reduce this value to retain more features.

```
model.add(layers.Dropout(0.3)) # from 0.5 to 0.3
model.add(layers.Dense(100,activation='softmax'))
```

In the progress of training, we also find that if we reduce batchsize to 16 or lower, the accuracy become higher than large batchsize like 200, 300.

```
model_sgd_2 = model
sgd = SGD(lr=0.03, decay = 0.0001, momentum=0.9, nesterov=True)
model_sgd_2.compile(loss='categorical_crossentropy', optimizer=sgd,metrics=['accuracy'])
history2 = model_sgd_2.fit(img_gen.flow(x_train, y_train, batch_size = 16), epochs=50,
                    validation_data = (x_val, y_val), verbose=1)
```

Since it would take several hours to reproduce this process just to change one parameter, we only mentioned it instead of providing grid search evidence. In fact, using grid search for CNN is a nightmare.

For learning rate, we set a decay and momentum for SGD optimizer, we also manually choose a lower learning rate to continue the training to get higher validation accuracy if accuracy and loss stop changing.

5.5.2.2 Our effort to speeding-up code:

In fact, manually choosing a lower learning rate which is mentioned above is also one of the way to speed up our training. Ideally, to get the highest accuracy we can set a appropriate decay for SGD and increase the epochs to a large value to wait it to converge. But we are not sure about how decay parameter of SGD exactly work and how many epochs is sufficient. So we try to detect the progression of training and manually change the learning rate. We plot the history of one specific learning rate like this:

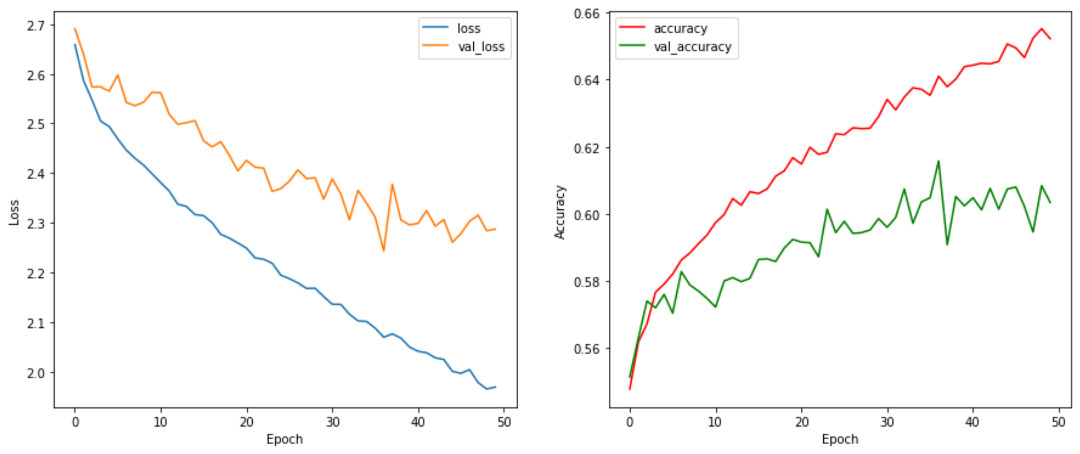


Figure10: history plot for learning rate = 0.0005

If the loss and accuracy of one model with particular learning rate converge, we just manually choose next level of learning rate, like 0.1 to 0.01.

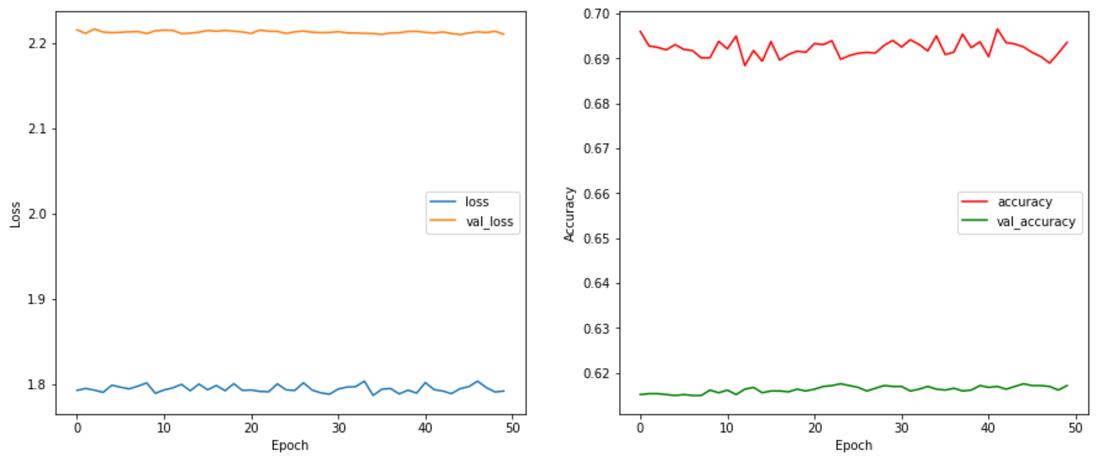


Figure 11: history plot for our final learning rate = 0.000005

For history of learning rate equal to 5e-6 above, we find that both of loss and accuracy do not change anymore (even though the start of the line), which means we have to stop training. We manually change learning rate for 5 times, which should probably save much plenty of time.

5.5.2.3 Evaluation of CNN:

In final 50 epochs, we got training accuracy and validation accuracy:

```
- loss: 1.7922 - accuracy: 0.6936 - val_loss: 2.2109 - val_accuracy: 0.6172
```

Then we use this CNN to predict the labels of test data and import classification report from sklearn.metrics to evaluate this estimator using accuracy, average precision recall:

```
CNN version evaluation:

Accuracy: 0.6229000091552734

Precision: 0.6410205738913081

Recall: 0.6236
```

Confusion Matrix of CNN:

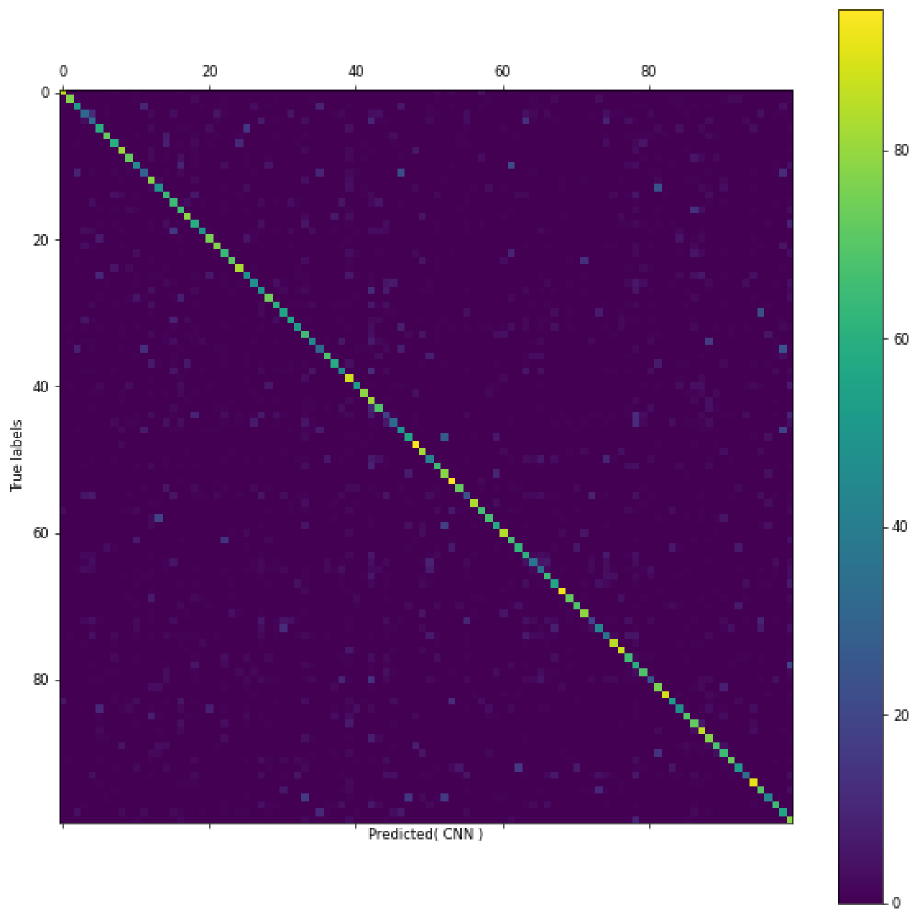


Figure12: confusion matrix of CNN

5.5.3 Summrize:

To comment these heatmaps of confusion matrix, we can apparently see that CNN's diagonal is the most conspicuous one, which indicate it has the greatest number of samples being correctly classified. The performance of KNN and SVM in confusion matrix is similar, although they are worse compared to CNN, you can still distinguish a diagonal very clearly. In fact, accuracy about 20% is much better than the result of randomly guess a label for 100 labels' dataset.

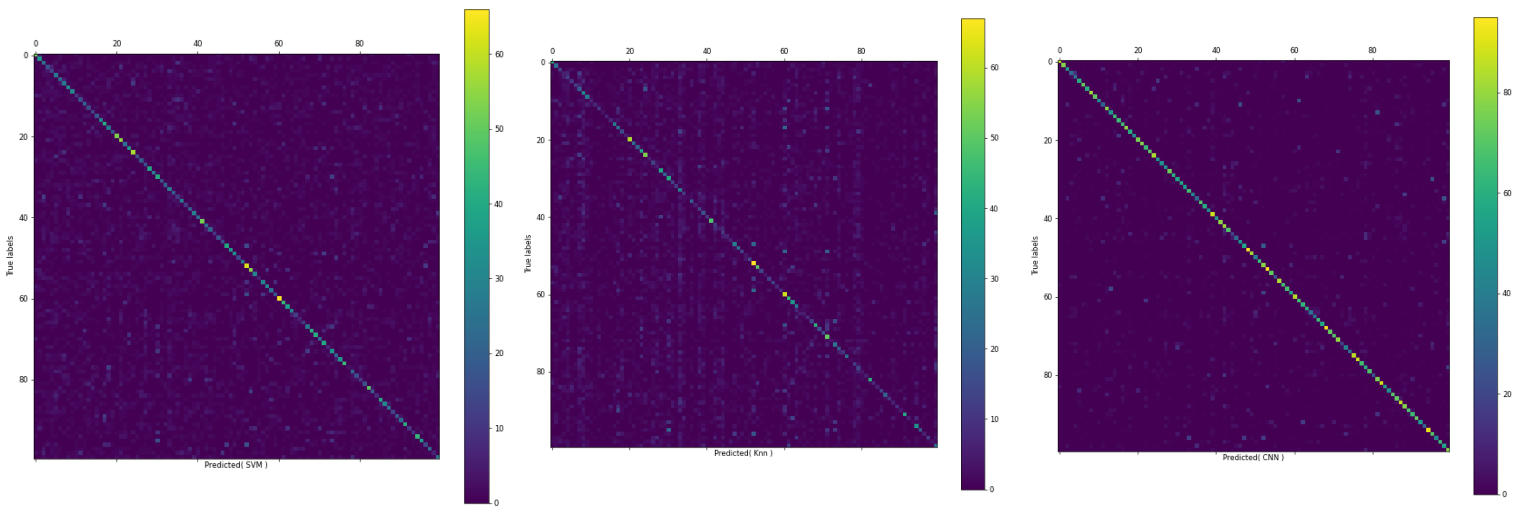


Figure13: Comparison of confusion matrix (left to right: KNN SVM CNN)

For the information given by below table, we can determine that, for accuracy, CNN can predict 62% correct labels, which is the best one, while KNN, SVM can only predict 18% and 27% correct labels.

The precision indicates the probability of when an estimator predict a sample and claim it's positive, and in fact it's true. That is, when CNN guessing a particular label, there's 64% chance it is indeed the label in average. And for KNN, SVM it's 20% and 26%, which is lower.

For recall, that is the ability to find a particular label. That means that when a sample is label A, CNN have 62% chance to predict A (for all the labels in average). While KNN, SVM is 18% and 27%, which is apparently lower. Recall and Precision trade off each other.

If an estimator has higher recall, then it's more likely to have lower precision. However, CNN have both highest value of them, which means CNN has the best generalization.

For training accuracy and validation accuracy, we can see that there's a large gap between training accuracy and validation accuracy of SVM, which definitely indicate overfitting. And for KNN, it has the lowest training accuracy and validation accuracy, which is the evidence of underfitting. For CNN, it has both training accuracy and validation accuracy converging to a high value, and they are quite close, which indicate no overfitting or underfitting.

For fitting time, since CNN have to refit the weight of every parameters for data, it cost 100 times of time consuming compared to SVM. And for KNN, because the main calculation happens while predicting, it cost the lowest time.

	Accuracy	Precision	Recall	Training acc	Valid acc	Fitting time, s
KNN	0.184	0.204	0.184	0.25	0.17	0.2
SVM	0.266	0.264	0.266	0.76	0.25	128.3
CNN	0.623	0.641	0.624	0.69	0.62	13200

Figure14: table of metrics

6 Conclusions:

Combining the algorithms of the three models and the volume of the data set, we can conclude that for KNN, this algorithm is a little bit simple and lazy for a 100-label data set, it is obviously underfitting. Perhaps reducing the value of k can reduce the impact of underfitting and increase the accuracy of the training set, but the accuracy of the validation set is likely to be lower. In general, I think KNN is not suitable for this data set. For SVM, the accuracy has increased relative to KNN, but it is still not ideal, and it suffered from very serious overfitting. It seems that underfitting and overfitting occur simultaneously on this model. Our analysis may be SVM is essentially a linear classifier. Although we are trying to use Gaussian kernal to convert non-linear classification problem into linear classification problem, the effect does not seem to be ideal. As CNN has enough ability to deal with nonlinear classification problems with activation functions like "relu", its accuracy is significantly higher than other classifiers. And for a large depth neural network, it has enough complexity to get a more ideal result, as we dropout the features in time, there is no obvious overfitting as well.

As suggestion for meaningful future work, we would consider using different type of kernel for SVM. For CNN, make the structure much more complex, we may increase the depth of the network, or try different size of filters, since the gap between valid accuracy and training accuracy is not large. We can also do vertical flip for augmentation to improve accuracy by increasing sample variety. And maybe we can select partial data to find a suitable structure first, which may save a lot of time. The performance of partial data and full data are supposed to be similar, so that we can speed up our code without losing accuracy.

7 References:

1. Kolesnikov A, Beyer L, Zhai X, Puigcerver J, Yung J, Gelly S, et al. Big Transfer (BiT): General Visual Representation Learning. arXiv:191211370 [cs] [Internet]. 2020 May 5 [cited 2021 Nov 7]; Available from: https://arxiv.org/abs/1912.11370

2. CNN 卷积层 [Internet]. 知乎专栏. Available from:

https://zhuanlan.zhihu.com/p/386704448

3. 全连接层和激活层_choushi5845的博客-CSDN博客 [Internet]. blog.csdn.net. Available from: https://blog.csdn.net/choushi5845/article/details/100747110

4. 机器学习实验 | VGG16 + cifar-10 + Keras+ 训练保存模型 +特征图可视化_Liaojiajia-Blog-CSDN博客 [Internet]. blog.csdn.net. Available from:

https://blog.csdn.net/mary_0830/article/details/106451690?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_title~default-0.no_search_link&spm=1001.2101.3001.4242.0

5. Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.

6. 最大池化层和平均池化层图解_微电子学与固体电子学-CSDN博客_平均池化层 [Internet]. yuchi.blog.csdn.net. [cited 2021 Nov 7]. Available from:

https://yuchi.blog.csdn.net/article/details/86571089?spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.no_search_link&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.no_search_link

8 Appendix

8.1 How to run our code:

group114_best_algorithm1:
It's necessary to import all the packages. After that you are supposed to run the data loading , preprocessing, split data, image augmentation step by step.
The next part is establish the structure of CNN model and plot the training history.
If it is not necessary for you to run the training progress, you can just ignore and jump to the Evaluation part, which can directly load our pre-trained CNN model. Then you just shift + enter block by block to evaluate our model.

group114_other_algorithms:
Just like group114_best_algorithm1, it's necessary to run import packages and data preprocessing like scale data and PCA. But tuning estimators' hyper – parameters part is not

necessary if you don’t want run the tuning code.
Then you just go to Evaluation part, run the code block by block to get the results of evaluation.