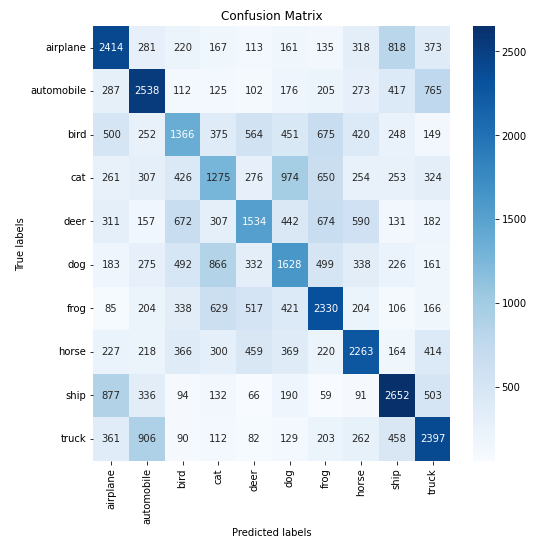
109550087

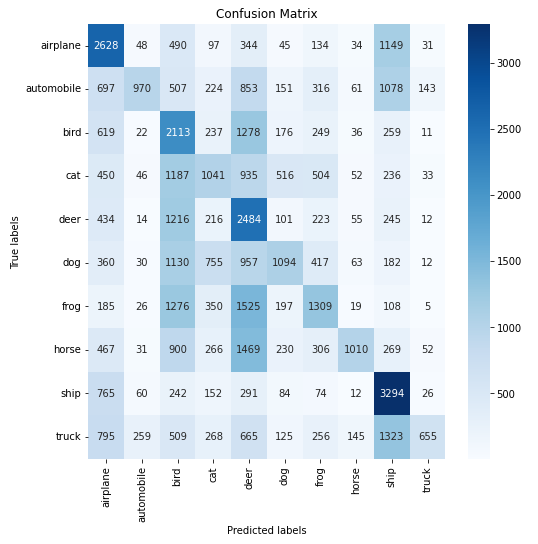
Programming Assignment #1

**A public image dataset (Cifar10)**

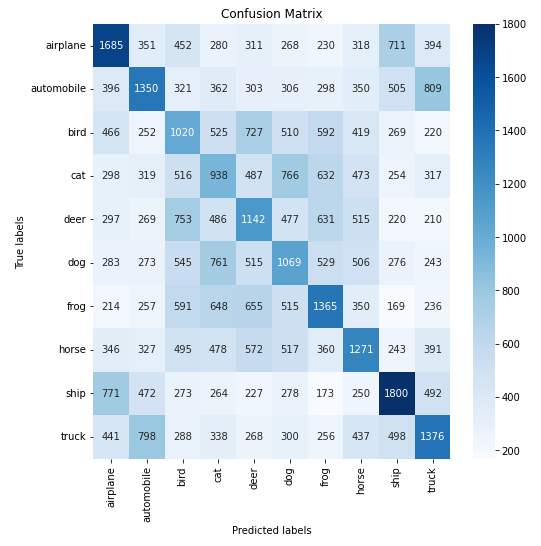
* First classifier: logistic regression (max\_iter=100)



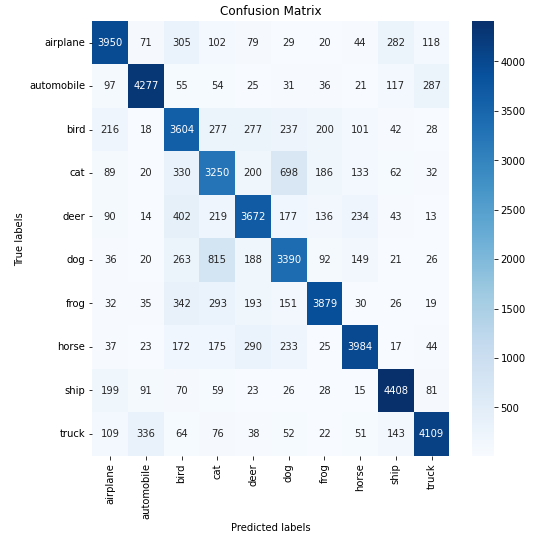
* Second classifier: KNN (n\_neighbors=5)



* Third classifier: decision tree (max\_depth=none)



* Fourth classifier: CNN



**using different amounts of training data**

Logistic regression (max\_iter=100)

|  |  |  |
| --- | --- | --- |
| Number of training data | All (50000) | Half (25000) |
| accuracy | 0.4079 | 0.3955 |
| Precision | 0.4026 | 0.3916 |
| Recall | 0.4079 | 0.3955 |
| F1-score | 0.4036 | 0.3913 |
| ROC AUC score | 0.8172 | 0.8119 |

KNN (n\_neighbors=5)

|  |  |  |
| --- | --- | --- |
| Number of training data | All (50000) | Half (25000) |
| accuracy | 0.3320 | 0.3131 |
| Precision | 0.4238 | 0.4039 |
| Recall | 0.3320 | 0.3130 |
| F1-score | 0.3180 | 0.2974 |
| ROC AUC score | 0.7264 | 0.7116 |

Decision tree (max\_depth=none)

|  |  |  |
| --- | --- | --- |
| Number of training data | All (50000) | Half (25000) |
| accuracy | 0.2603 | 0.2477 |
| Precision | 0.2613 | 0.2489 |
| Recall | 0.2604 | 0.2478 |
| F1-score | 0.2606 | 0.2481 |
| ROC AUC score | 0.5891 | 0.5821 |

CNN

|  |  |  |
| --- | --- | --- |
| Number of training data | All (50000) | Half (25000) |
| accuracy | 0.7705 | 0.7298 |
| Precision | 0.7802 | 0.7314 |
| Recall | 0.7705 | 0.7298 |
| F1-score | 0.7717 | 0.7273 |
| ROC AUC score | 0.9731 | 0.9626 |

**using different classifiers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| model | Logistic regression | KNN | Decision tree | CNN |
| accuracy | 0.4079 | 0.3320 | 0.2603 | 0.7705 |
| Precision | 0.4026 | 0.4238 | 0.2613 | 0.7802 |
| Recall | 0.4079 | 0.3320 | 0.2604 | 0.7705 |
| F1-score | 0.4036 | 0.3180 | 0.2606 | 0.7717 |
| ROC AUC score | 0.8172 | 0.7264 | 0.5891 | 0.9731 |

**using different settings and/or hyper-parameters**

Logistic regression

|  |  |  |  |
| --- | --- | --- | --- |
| max\_iter | 100 | 200 | 400 |
| accuracy | 0.4079 | 0.4058 | 0.3945 |
| Precision | 0.4026 | 0.4024 | 0.3912 |
| Recall | 0.4079 | 0.4059 | 0.3946 |
| F1-score | 0.4036 | 0.4029 | 0.3922 |
| ROC AUC score | 0.8172 | 0.8173 | 0.8108 |

KNN

|  |  |  |  |
| --- | --- | --- | --- |
| n\_neighbors | 3 | 5 | 10 |
| accuracy | 0.3232 | 0.3320 | 0.3289 |
| Precision | 0.4229 | 0.4238 | 0.4361 |
| Recall | 0.3233 | 0.3320 | 0.3290 |
| F1-score | 0.3103 | 0.3180 | 0.3121 |
| ROC AUC score | 0.6993 | 0.7264 | 0.7581 |

Decision tree

|  |  |  |  |
| --- | --- | --- | --- |
| max\_depth | none | 10 | 20 |
| accuracy | 0.2603 | 0.2925 | 0.2632 |
| Precision | 0.2613 | 0.2920 | 0.2646 |
| Recall | 0.2604 | 0.2927 | 0.2633 |
| F1-score | 0.2606 | 0.2894 | 0.2636 |
| ROC AUC score | 0.5891 | 0.6071 | 0.5907 |

CNN

|  |  |  |
| --- | --- | --- |
| layers | simple | complex |
| accuracy | 0.7705 | 0.9038 |
| Precision | 0.7802 | 0.9038 |
| Recall | 0.7705 | 0.9038 |
| F1-score | 0.7717 | 0.9034 |
| ROC AUC score | 0.9731 | 0.9932 |

For the first dataset, I use a popular image dataset – cifar10. It consists of 60000 32x32 color images in 10 classes, with 6000 images per class. In detail, it has 50000 for training and 10000 for testing.

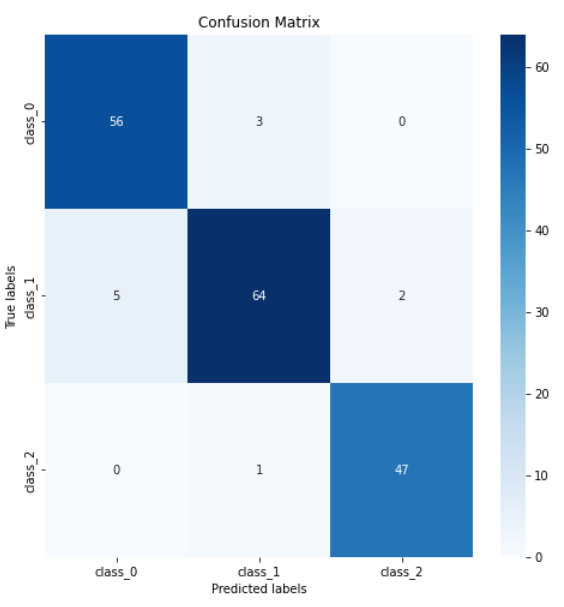
To begin with, I use four different model, logistic regression, KNN, decision tree and CNN, all using kfold(n=5), and all of their performance are affected by the amount of training data (50000 to 25000). As a result, we can tell that it is important to have a larger size of dataset.

Further more, as we can see, KNN and Decision tree perform badly on cifar10. I think it is because both of the models aren’t suitable for complex problems, and 10 classes are too hard for them to predict correctly. As for logistic regression, since we can apply as many iterations as we want, making the model more “fit” on the dataset, its performance is slightly higher than previous models. Last, I use a CNN with simple architecture(detail in appendix), and it outperforms any other model. For one thing, CNN is good at image classification. For another, its architecture can be designed by coders, so we can choose those layers that are suitable for certain dataset. In other word, if we build a CNN with randomly chosen layers, its performance may drop significantly.

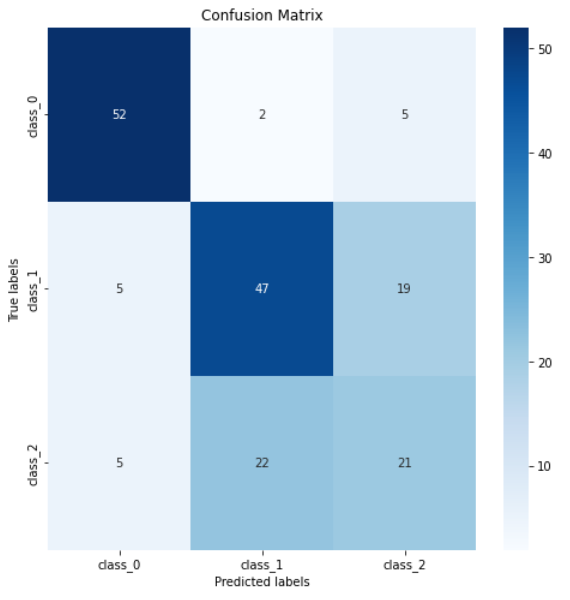
Next, I use different hyper-parameters on different models. For logistic regression, KNN and decision tree, different hyper-parameters seem not to affect the performance significantly. However, the architecture of CNN does affect the performance greatly (from 0.77 to 0.9). I think it is because the second architecture I used can fetch the feature more accurately, which lead to a higher accuracy.

**A public non-image dataset (wine)**

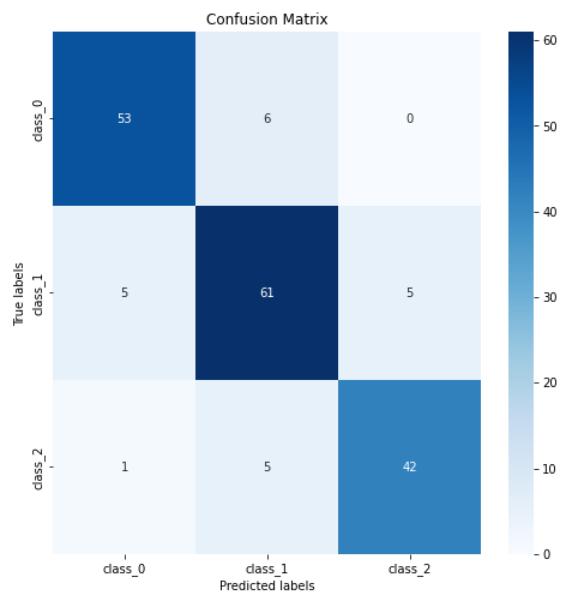
* First classifier: logistic regression (max\_iter=100)



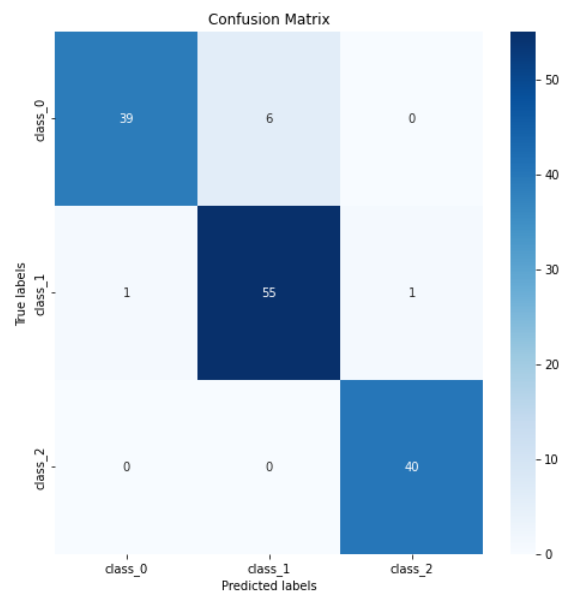
* Second classifier: KNN (n\_neighbors=5)



* Third classifier: decision tree (max\_depth=none)



* Fourth classifier: CNN



**using different amounts of training data**

Logistic regression (max\_iter=100)

|  |  |  |
| --- | --- | --- |
| Number of training data | All (178) | Half (89) |
| accuracy | 0.9383 | 0.7022 |
| Precision | 0.9431 | 0.4774 |
| Recall | 0.9420 | 0.6411 |
| F1-score | 0.9404 | 0.5454 |
| ROC AUC score | 0.9927 | 0.9401 |

KNN (n\_neighbors=5)

|  |  |  |
| --- | --- | --- |
| Number of training data | All (178) | Half (89) |
| accuracy | 0.6746 | 0.6854 |
| Precision | 0.6628 | 0.4767 |
| Recall | 0.6599 | 0.6248 |
| F1-score | 0.6556 | 0.5366 |
| ROC AUC score | 0.8603 | 0.8996 |

Decision tree (max\_depth=none)

|  |  |  |
| --- | --- | --- |
| Number of training data | All (178) | Half (89) |
| accuracy | 0.8763 | 0.6962 |
| Precision | 0.8828 | 0.4656 |
| Recall | 0.8794 | 0.6390 |
| F1-score | 0.8786 | 0.5380 |
| ROC AUC score | 0.9096 | nan |

CNN

|  |  |  |
| --- | --- | --- |
| Number of training data | All (178) | Half (89) |
| accuracy | 0.9931 | 0.9771 |
| Precision | 0.9917 | 0.9750 |
| Recall | 0.9933 | 0.9773 |
| F1-score | 0.9920 | 0.9733 |
| ROC AUC score | 1.0 | 0.9959 |

**using different classifiers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| model | Logistic regression | KNN | Decision tree | CNN |
| accuracy | 0.9383 | 0.6746 | 0.8763 | 0.9931 |
| Precision | 0.9431 | 0.6628 | 0.8828 | 0.9917 |
| Recall | 0.9420 | 0.6599 | 0.8794 | 0.9933 |
| F1-score | 0.9404 | 0.6556 | 0.8786 | 0.9920 |
| ROC AUC score | 0.9927 | 0.8603 | 0.9096 | 1.0 |

**using different settings and/or hyper-parameters**

Logistic regression

|  |  |  |  |
| --- | --- | --- | --- |
| max\_iter | 100 | 50 | 20 |
| accuracy | 0.9383 | 0.9100 | 0.6802 |
| Precision | 0.9431 | 0.9196 | 0.7144 |
| Recall | 0.9420 | 0.9103 | 0.6477 |
| F1-score | 0.9404 | 0.9115 | 0.6313 |
| ROC AUC score | 0.9927 | 0.9807 | 0.8756 |

KNN

|  |  |  |  |
| --- | --- | --- | --- |
| n\_neighbors | 3 | 5 | 10 |
| accuracy | 0.7083 | 0.6746 | 0.6802 |
| Precision | 0.7013 | 0.6628 | 0.6645 |
| Recall | 0.7022 | 0.6599 | 0.6603 |
| F1-score | 0.6965 | 0.6556 | 0.6576 |
| ROC AUC score | 0.8502 | 0.8603 | 0.8631 |

Decision tree

|  |  |  |  |
| --- | --- | --- | --- |
| max\_depth | none | 3 | 2 |
| accuracy | 0.8763 | 0.8876 | 0.8600 |
| Precision | 0.8828 | 0.8970 | 0.8731 |
| Recall | 0.8794 | 0.8848 | 0.8651 |
| F1-score | 0.8786 | 0.8865 | 0.8582 |
| ROC AUC score | 0.9096 | 0.9270 | 0.9156 |

CNN

|  |  |  |
| --- | --- | --- |
| layers | simple | complex |
| accuracy | 0.9931 | 0.9443 |
| Precision | 0.9917 | 0.9524 |
| Recall | 0.9933 | 0.9411 |
| F1-score | 0.9920 | 0.429 |
| ROC AUC score | 1.0 | 0.9822 |

For the Second dataset, I use a non-image dataset – wine. These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

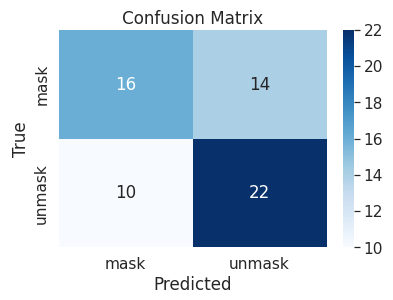
In this part, I use the same four models, but I changed the architecture of CNN to be more suitable in this dataset(detail in appendix). Similarly, most of the models’ performances drop after I decrease the amount of training data. What’s, more, since this dataset is much smaller than cifar10, so cutting the amount of training data into half has a larger impact on performances.

As I mentioned above, wine dataset is a small dataset. Consequently, except KNN, all models perform well on this dataset. For logistic regression, smaller amount makes it easier to find a linear pattern; for decision tree, only three classes are in the dataset, so it is more likely to distinguish from one class to another. As for KNN, I think it is the dataset’s characteristic that makes KNN’s performance poor, maybe the distribution of the data isn’t dense enough for the model to predict. Also, CNN still performs best among all the models. I think the reasons are similar: I can decide what layer to use, which is a advantage of using CNN.

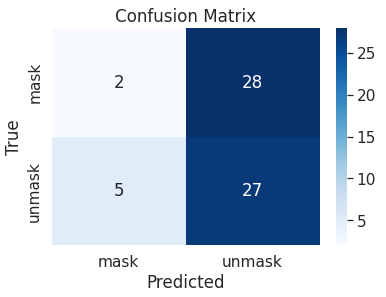
Last, the different hyper-parameters part. First, logistic regression, we can see that max\_iter=20 is not enough for this situation, and the more iteration we apply, the better performance we get. Next, KNN, the performance of the model actually doesn’t change a lot when using different n\_neighbors, maybe it’s because KNN already perform poor on wine dataset, using different hyper-parameters won’t affect its performance, I guess. Third, decision tree, since the dataset is small enough, so max\_depth=3 is already enough for a acceptable accuracy, which in fact has a higher accuracy than max\_depth=5 one. Last but not least, CNN. In this dataset, a complex architecture performs poorer than a simple one. In my opinion, maybe I add a useless layer, or superfluous layer in the complex one. Nonetheless, both of the architectures still have high accuracy.

**A self-made dataset**

* First classifier: logistic regression (max\_iter=100)

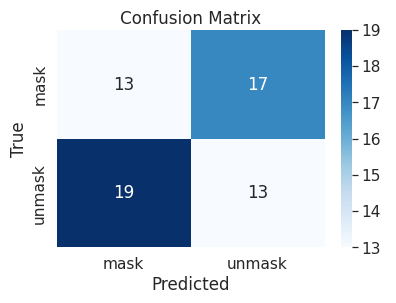


* Second classifier: KNN (n\_neighbors=5)



* Third classifier: decision tree

(max\_depth=none)



* Fourth classifier: CNN

|  |
| --- |
|  |

**using different amounts of training data**

Logistic regression (max\_iter=100)

|  |  |  |
| --- | --- | --- |
| Number of training data | All (310) | Half (168) |
| accuracy | 0.6129 | 0.5 |
| Precision | 0.6111 | 0.44 |
| Recall | 0.6875 | 0.7857 |
| F1-score | 0.6471 | 0.5641 |
| ROC AUC score | 0.6104 | 0.5429 |

KNN (n\_neighbors=5)

|  |  |  |
| --- | --- | --- |
| Number of training data | All (310) | Half (168) |
| accuracy | 0.4677 | 0.4118 |
| Precision | 0.4909 | 0.4062 |
| Recall | 0.8438 | 0.9286 |
| F1-score | 0.6207 | 0.5652 |
| ROC AUC score | 0.4552 | 0.4893 |

Decision tree (max\_depth=none)

|  |  |  |
| --- | --- | --- |
| Number of training data | All (310) | Half (168) |
| accuracy | 0.4194 | 0.4706 |
| Precision | 0.4333 | 0.4091 |
| Recall | 0.4062 | 0.6429 |
| F1-score | 0.4194 | 0.5 |
| ROC AUC score | 0.4198 | 0.4964 |

CNN

|  |  |  |
| --- | --- | --- |
| Number of training data | All (310) | Half (168) |
| accuracy | 0.6774 | 0.4904 |

**using different classifiers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| model | Logistic regression | KNN | Decision tree | CNN |
| accuracy | 0.6129 | 0.4677 | 0.4194 | 0.6774 |
| Precision | 0.6111 | 0.4909 | 0.4333 |  |
| Recall | 0.6875 | 0.8438 | 0.4062 |  |
| F1-score | 0.6471 | 0.6207 | 0.4194 |  |
| ROC AUC score | 0.6104 | 0.4552 | 0.4198 |  |

**using different settings and/or hyper-parameters**

Logistic regression

|  |  |  |  |
| --- | --- | --- | --- |
| max\_iter | 100 | 20 | 10 |
| accuracy | 0.6129 | 0.629 | 0.5806 |
| Precision | 0.6111 | 0.6364 | 0.575 |
| Recall | 0.6875 | 0.6562 | 0.7188 |
| F1-score | 0.6471 | 0.6462 | 0.6389 |
| ROC AUC score | 0.6104 | 0.6281 | 0.576 |

KNN

|  |  |  |  |
| --- | --- | --- | --- |
| n\_neighbors | 5 | 3 | 1 |
| accuracy | 0.4677 | 0.4032 | 0.4355 |
| Precision | 0.4909 | 0.451 | 0.4681 |
| Recall | 0.8438 | 0.7188 | 0.6875 |
| F1-score | 0.6207 | 0.5542 | 0.557 |
| ROC AUC score | 0.4552 | 0.3927 | 0.4271 |

Decision tree

|  |  |  |  |
| --- | --- | --- | --- |
| max\_depth | none | 3 | 1 |
| accuracy | 0.4194 | 0.4677 | 0.5323 |
| Precision | 0.4333 | 0.4783 | 0.7143 |
| Recall | 0.4062 | 0.3438 | 0.1562 |
| F1-score | 0.4194 | 0.4 | 0.2564 |
| ROC AUC score | 0.4198 | 0.4719 | 0.5448 |

In this part, I choose to classify whether a person is wearing a face mask or not. As a result, my dataset has two kind of images: people with face masks and without face masks. The former has 161 images, and the latter has 148. All of the images are gathered from Internet.

In this part, we can also see the effect of decreasing the amount of training data, most of the models have lower accuracy with half amount of training data. In terms of using different classifiers, logistic regression and CNN perform better than the remaining two. However, they are only slightly higher than random guess (0.5, since my dataset is a binary classification). The reason of this, I think, is because of the quality and the size of the dataset. Since the images are from many websites, some of them may have complex background, making it more difficult to specify where the face is. Also, I only gathered 310 images in total, which is a very small amount. Another point I want to point out is, the performance of KNN, it still performs the worst among four models, no matter how I change its hyper-parameter.

**Discussion**

In this homework, I implement four different models of all the datasets, making it easier to compare their performance under different circumstances. In terms of their performances, CNN performs as good as I expected, which outperforms other three models on all the datasets. Nevertheless, KNN’s performance is way lower than I expected, and it has low accuracy but high recall score, which is equivalent to the True Positive rate. After some research, I notice “the curse of dimensionality”, which means that KNN performs best with a low number of features. When the number of features increases, then it requires more data. When there’s more data, it creates an overfitting problem because no one knows which piece of noise will contribute to the model. To be more specific, we need less features in dataset to have a better KNN model.

Well, there are still many experiments that can be done. If time is available, I would do grid search on every model on every dataset to find the best combination of different settings/hyper-parameters. Also, I would try more different architectures of CNN, different loss function(MSE, cross-enropy…) or different optimizer, and a more complicated model such as SVM, which is somehow time-consuming and makes me didn’t add the model into my implementation.

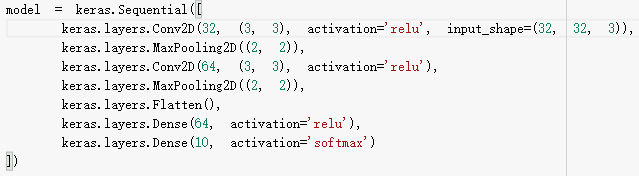
Last thing I want to note is, data preprocessing. Data preprocessing is an important part in machine learning, including feature concatenation, feature extraction, sampling, and etc. I wonder how different preprocessing steps would affect the final output. Maybe if I preprocess the data correctly, simple model like logistic regression can also catch on the performance of neural network? Who knows!

Appendix

**CNN architecture**

*(part1)*

Simple layers

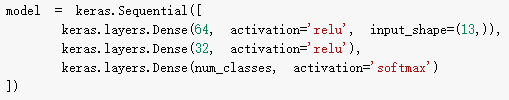


Complex layers

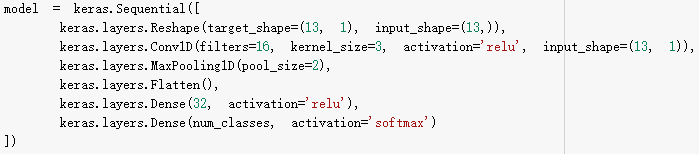


*(part2)*

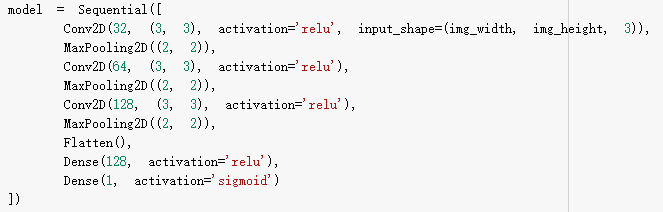
Simple layers



Complex layers



*(part3)*



**Code**

**Logistic regression**

# Load the CIFAR-10 dataset

(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()

# Flatten the images and scale pixel values to range [0, 1]

X\_train = X\_train.reshape(X\_train.shape[0], -1) / 255.0

X\_test = X\_test.reshape(X\_test.shape[0], -1) / 255.0

# Define the Logistic Regression model

model = LogisticRegression(max\_iter=100)

# Define the k-fold cross-validation object

kf = KFold(n\_splits=5, shuffle=True, random\_state=42)

# Initialize lists to store the evaluation metrics

accuracy\_scores = []

precision\_scores = []

recall\_scores = []

f1\_scores = []

roc\_auc\_scores = []

conf\_matrices = []

# Loop over the folds of the cross-validation

for train\_index, val\_index in kf.split(X\_train):

    # Split the training set into a training subset and a validation subset

    X\_train\_sub, X\_val = X\_train[train\_index], X\_train[val\_index]

    y\_train\_sub, y\_val = y\_train[train\_index], y\_train[val\_index]

    # Train the model on the training subset

    model.fit(X\_train\_sub, y\_train\_sub.ravel())

    # Predict the labels on the validation subset

    y\_pred = model.predict(X\_val)

    # Evaluate the model using various metrics

    accuracy\_scores.append(accuracy\_score(y\_val, y\_pred))

    precision\_scores.append(classification\_report(y\_val, y\_pred, output\_dict=True)['macro avg']['precision'])

    recall\_scores.append(classification\_report(y\_val, y\_pred, output\_dict=True)['macro avg']['recall'])

    f1\_scores.append(classification\_report(y\_val, y\_pred, output\_dict=True)['macro avg']['f1-score'])

    roc\_auc\_scores.append(roc\_auc\_score(y\_val, model.predict\_proba(X\_val), multi\_class='ovr'))

    conf\_matrices.append(confusion\_matrix(y\_val, y\_pred))

**KNN**

# Load CIFAR-10 dataset

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

# Reshape input data

X = x\_train.reshape((x\_train.shape[0], -1))

y = y\_train.reshape((-1,))

# Create a KNN classifier object

knn = KNeighborsClassifier(n\_neighbors=5)

# Create KFold cross-validation object

kf = KFold(n\_splits=5, shuffle=True, random\_state=42)

# Perform cross-validation

scores = cross\_val\_score(knn, X, y, cv=kf, scoring='accuracy')

# Compute precision, recall, f1-score, and roc\_auc\_score for each fold

precision\_scores = []

recall\_scores = []

f1\_scores = []

roc\_auc\_scores = []

confusion\_matrices = []

for train\_idx, test\_idx in kf.split(X):

    X\_train, X\_test = X[train\_idx], X[test\_idx]

    y\_train, y\_test = y[train\_idx], y[test\_idx]

    knn.fit(X\_train, y\_train)

    y\_pred = knn.predict(X\_test)

    precision\_scores.append(precision\_score(y\_test, y\_pred, average='macro'))

    recall\_scores.append(recall\_score(y\_test, y\_pred, average='macro'))

    f1\_scores.append(f1\_score(y\_test, y\_pred, average='macro'))

    roc\_auc\_scores.append(roc\_auc\_score(y\_test, knn.predict\_proba(X\_test), multi\_class='ovr'))

    confusion\_matrices.append(confusion\_matrix(y\_test, y\_pred))

# Print the mean accuracy and mean scores for each metric

print("Accuracy:", np.mean(scores))

print('Precision:', round(np.mean(precision\_scores), 4))

print('Recall:', round(np.mean(recall\_scores), 4))

print('F1-score:', round(np.mean(f1\_scores), 4))

print('ROC AUC score:', round(np.mean(roc\_auc\_scores), 4))

**Decision tree**

# Load CIFAR-10 dataset

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

# Reshape input data

X = x\_train.reshape((x\_train.shape[0], -1))

y = y\_train.reshape((-1,))

# Initialize decision tree classifier

tree = DecisionTreeClassifier(random\_state=42)

# Define k-fold cross-validation with 5 folds

kf = KFold(n\_splits=5, shuffle=True, random\_state=42)

# Initialize lists to store evaluation metrics for each fold

accuracy\_list = []

precision\_list = []

recall\_list = []

f1\_list = []

roc\_auc\_list = []

conf\_matrix\_list = []

# Perform k-fold cross-validation

for train\_index, test\_index in kf.split(X):

    # Split data into training and testing sets

    X\_train, X\_test = X[train\_index], X[test\_index]

    y\_train, y\_test = y[train\_index], y[test\_index]

    # Fit model on training set

    tree.fit(X\_train, y\_train)

    # Make predictions on testing set

    y\_pred = tree.predict(X\_test)

    # Convert y\_test and y\_pred to one-hot encoding

    y\_test = to\_categorical(y\_test)

    y\_pred = to\_categorical(y\_pred)

    # Compute accuracy, precision, recall, f1-score, ROC AUC score, and confusion matrix for this fold

    accuracy = accuracy\_score(y\_test.argmax(axis=1), y\_pred.argmax(axis=1))

    precision = precision\_score(y\_test, y\_pred, average="macro")

    recall = recall\_score(y\_test, y\_pred, average="macro")

    f1 = f1\_score(y\_test, y\_pred, average="macro")

    roc\_auc = roc\_auc\_score(y\_test, y\_pred, average="macro", multi\_class="ovo")

    conf\_matrix = confusion\_matrix(y\_test.argmax(axis=1), y\_pred.argmax(axis=1))

    # Append evaluation metrics to lists

    accuracy\_list.append(accuracy)

    precision\_list.append(precision)

    recall\_list.append(recall)

    f1\_list.append(f1)

    roc\_auc\_list.append(roc\_auc)

    conf\_matrix\_list.append(conf\_matrix)

**CNN**

# Load the CIFAR10 dataset

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

# Normalize pixel values between 0 and 1

x\_train = x\_train.astype('float32') / 255

x\_test = x\_test.astype('float32') / 255

# Convert labels to one-hot encoding

y\_train = keras.utils.to\_categorical(y\_train, 10)

y\_test = keras.utils.to\_categorical(y\_test, 10)

# Define the CNN model architecture

model = keras.Sequential([

    keras.layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)),

    keras.layers.MaxPooling2D((2, 2)),

    keras.layers.Conv2D(64, (3, 3), activation='relu'),

    keras.layers.MaxPooling2D((2, 2)),

    keras.layers.Flatten(),

    keras.layers.Dense(64, activation='relu'),

    keras.layers.Dense(10, activation='softmax')

])

# Compile the model with categorical crossentropy loss and Adam optimizer

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Define KFold cross-validation

kf = KFold(n\_splits=5, shuffle=True, random\_state=42)

# Initialize arrays to store results

accuracy\_scores = []

precision\_scores = []

recall\_scores = []

f1\_scores = []

roc\_auc\_scores = []

conf\_matrices = []

# Train and evaluate the model for each fold

for train\_idx, test\_idx in kf.split(x\_train):

    x\_train\_fold, y\_train\_fold = x\_train[train\_idx], y\_train[train\_idx]

    x\_test\_fold, y\_test\_fold = x\_train[test\_idx], y\_train[test\_idx]

    model.fit(x\_train\_fold, y\_train\_fold, epochs=10, batch\_size=64)

    y\_pred\_fold = model.predict(x\_test\_fold)

    y\_pred\_fold\_classes = np.argmax(y\_pred\_fold, axis=1)

    y\_test\_fold\_classes = np.argmax(y\_test\_fold, axis=1)

    accuracy\_scores.append(accuracy\_score(y\_test\_fold\_classes, y\_pred\_fold\_classes))

    precision\_scores.append(classification\_report(y\_test\_fold\_classes, y\_pred\_fold\_classes, output\_dict=True)['weighted avg']['precision'])

    recall\_scores.append(classification\_report(y\_test\_fold\_classes, y\_pred\_fold\_classes, output\_dict=True)['weighted avg']['recall'])

    f1\_scores.append(classification\_report(y\_test\_fold\_classes, y\_pred\_fold\_classes, output\_dict=True)['weighted avg']['f1-score'])

    roc\_auc\_scores.append(roc\_auc\_score(y\_test\_fold, y\_pred\_fold, multi\_class='ovr'))

    conf\_matrices.append(confusion\_matrix(y\_test\_fold\_classes, y\_pred\_fold\_classes))

# Compute the mean of the evaluation scores over all folds

mean\_accuracy = np.mean(accuracy\_scores)

mean\_precision = np.mean(precision\_scores)

mean\_recall = np.mean(recall\_scores)

mean\_f1 = np.mean(f1\_scores)

mean\_roc\_auc = np.mean(roc\_auc\_scores)

mean\_conf\_matrix = np.mean(conf\_matrices, axis=0)