Assignment 2

Machine Learning, SS23

Team members		
Last name	First name	Matriculation Number
Heng	Chloe Yi Ning	12239126
Lim	Yan Qi	12239271

1 Neural Networks

1.1 PCA Classification

1.1.1 PCA for dimensionality reduction

N components used: 400

Variance explained

Total explained variance obtained: 96.461%

Variance explained for each selected component:

```
Explained variance of each n component: [0.1200027  0.07080331 0.04303719 0.0413679  0.03782501 0.03369365
 0.02749453 0.02149849 0.02011629 0.01876554 0.01779723 0.01751738
 0.01561228 0.01424105 0.01391119 0.01323543 0.01256689 0.01187404
 0.01134885 0.01031115 0.00943928 0.00918016 0.00879083 0.0084098
 0.0077773  0.00717117 0.00686497 0.00657955 0.00624481 0.00587618
 0.00565918 0.00546852 0.00526848 0.00513563 0.00494943 0.00463522
 0.00462215 0.00438199 0.00435642 0.00418157 0.00412519 0.00385322
 0.0037907  0.00373974 0.00366797 0.00353858 0.00347053 0.00339049
 0.00323328 0.00315134 0.0030939  0.00304344 0.00300488 0.00289043
 0.00284076 0.00280113 0.0027045  0.00268166 0.00262692 0.00251884
 0.00248914 0.00240061 0.00237763 0.00233699 0.00227566 0.00221927
 0.00218734 0.00215791 0.00209739 0.00204603 0.0020206  0.00199305
 0.00197481 0.00195415 0.00187989 0.00185077 0.00183293 0.00180116
 0.00178363 0.00174104 0.00173521 0.00171649 0.00167054 0.00162601
 0.00161882 0.00159264 0.00157108 0.00154468 0.00151318 0.00148413
 0.00147499 0.00145632 0.00142591 0.00141488 0.00140861 0.00138949
 0.00136723 0.00134421 0.00133249 0.00131423 0.00130995 0.0012924
 0.00126111 0.00126015 0.0012382  0.00121171 0.00119473 0.00118233
 0.00116649 0.00115643 0.00114669 0.00112048 0.00111559 0.00110955
 0.00108096 0.00107881 0.00106234 0.00104252 0.00103672 0.00102
 0.00100061 0.00098887 0.00098601 0.00097833 0.00096135 0.00095366
 0.00094133 0.0009284  0.00091536 0.00089575 0.00089068 0.00088593
 0.0008781  0.00086338 0.00085997 0.00085026 0.000838   0.00083405
 0.00081648 0.00081464 0.00081133 0.00080547 0.0007969  0.00078784
 0.00077268 0.00076895 0.00076284 0.00076022 0.00075059 0.00074122
 0.00073031 0.00072635 0.00071901 0.00071013 0.00070242 0.00069609
 0.0006926  0.00068809 0.00068496 0.00067315 0.00066929 0.00066436
 0.00065703 0.0006503  0.00064143 0.00063784 0.00063216 0.00062593
 0.00062232 0.00061728 0.00061225 0.00061053 0.0006021  0.00059924
 0.00059424 0.00058867 0.00058396 0.00057984 0.0005758  0.00057229
 0.00056584 0.00056445 0.00055498 0.00054908 0.00054436 0.00053916
 0.00053656 0.00053083 0.00052625 0.00051798 0.00051608 0.00051211
 0.00050859 0.00050613 0.00050218 0.00049494 0.00049399 0.00048748
 0.00048345 0.00048008 0.00047756 0.00047184 0.00046954 0.00046752
 0.00046352 0.00046165 0.00045592 0.00045327 0.00044879 0.0004469
 0.00044257 0.00044137 0.00043616 0.00043274 0.0004306  0.00042863
 0.00042007 0.0004178  0.00041281 0.00041145 0.00040604 0.00040418
 0.00040235 0.0003979  0.00039569 0.00039282 0.00039092 0.00039012
 0.00038495 0.00038316 0.00038153 0.00037894 0.0003766  0.00037337
 0.00037    0.00036905 0.00036521 0.00036291 0.0003604  0.00035742
 0.00035458 0.00035387 0.00035314 0.00034971 0.00034518 0.00034362
 0.00034162 0.00033587 0.00033426 0.00033233 0.00033138 0.00033072
 0.00032627 0.00032381 0.00032291 0.00031972 0.00031864 0.00031571
 0.00031282 0.00031131 0.0003089  0.00030566 0.0003053  0.00030233
 0.00030102 0.00030078 0.00029853 0.00029549 0.00029395 0.00029099
 0.00028993 0.00028774 0.00028562 0.00028517 0.00028156 0.00028014
 0.00027974 0.0002773  0.00027474 0.00027338 0.00027249 0.00027086
 0.00027052 0.00026826 0.00026691 0.00026403 0.00026147 0.00026032
```

```
 0.00025932 0.00025735 0.00025586 0.00025434 0.00025306 0.00025295
 0.00025109 0.00024861 0.00024756 0.00024569 0.00024285 0.00024246
 0.00024034 0.00024009 0.00023867 0.00023831 0.00023601 0.00023498
 0.0002336  0.00023169 0.0002297  0.00022821 0.00022712 0.00022641
 0.00022454 0.00022235 0.00022131 0.00021994 0.00021904 0.00021783
 0.00021609 0.00021395 0.00021252 0.0002116  0.00020955 0.00020848
 0.00020814 0.00020748 0.00020598 0.00020551 0.0002042  0.00020283
 0.0002015  0.0002004  0.00019884 0.00019694 0.00019547 0.00019538
 0.00019363 0.00019279 0.00019175 0.00019133 0.00018883 0.00018708
 0.00018686 0.00018598 0.00018483 0.00018474 0.00018274 0.00018241
 0.00018154 0.00018054 0.00017913 0.00017801 0.00017736 0.00017646
 0.00017616 0.00017394 0.00017336 0.0001718  0.00017092 0.00017048
 0.0001693  0.00016785 0.00016667 0.00016474 0.00016399 0.00016264
 0.00016216 0.00016163 0.00016128 0.00016063 0.0001591  0.00015824
 0.00015737 0.00015647 0.00015469 0.00015413 0.00015303 0.00015126
 0.00015055 0.00014958 0.00014901 0.00014785 0.00014669 0.00014571
 0.00014445 0.0001441  0.00014319 0.00014168 0.00014025 0.00013904
 0.00013827 0.00013723 0.00013681 0.0001356  0.00013411 0.00013289
 0.00013197 0.00013139 0.00013049 0.00012872]
(2062, 400)
```

1.1.2 Varying the number of neurons

For each n hid, report the accuracy on the train and test set, and the loss.

```
----- Task 1.1.2 -----
-----  2 neurons -----
Train accuracy: 0.5609. Test accuracy: 0.3123
Loss: 1.1337
-----  10 neurons -----
Train accuracy: 0.9970. Test accuracy: 0.4600
Loss: 0.0167
-----  100 neurons -----
Train accuracy: 1.0000. Test accuracy: 0.5981
Loss: 0.0043
-----  200 neurons -----
Train accuracy: 1.0000. Test accuracy: 0.5448
Loss: 0.0035
```

How do we know (in general) if the model does not have enough capacity (the case of underfitting)?

- ➔ In this case, the train accuracy and test accuracy are both low and both values did not reach the maximum value possible seen in the entire process of learning.

How do we know (in general) if the model starts to overfit?

- ➔ The model will start to overfit when the train and test accuracy have reached their corresponding maximum values and in the next iteration, the test accuracy falls. In other words, the model will start to overfit in this iteration when the test accuracy has just started to decrease right after reaching its maximum value.

Does that happen with some architectures/models? (If so, say with what number of neurons that happens).

- ➔ Yes, this does happen with some models especially when a very powerful function is being used, or a very high number of neurons is being used. In this case, it happened when the number of neurons was 200.

Which model would you choose here and why?

- ➔ I would choose the model with 100 neurons. As this model is just right and based on the previous explanation on when underfitting and overfitting occurs, this model did not underfit or overfit. It has the highest train and test accuracy.

1.1.3 Overfitting prevention: Regularization and/or early stopping

Option (a) where alpha = 0.1 works the best in my opinion.

Report the train and test accuracy, and the loss for n hidden ∈ {2, 10, 100, 200} for setup (a), (b) and (c).

```
----- Task 1.1.3 -----
-----  2 neurons -----
(a) alpha = 0.1
Train accuracy: 0.5537. Test accuracy: 0.3002
Loss: 1.1891
(b) early_stopping = true
Train accuracy: 0.1401. Test accuracy: 0.1162
Loss: 2.1439
(c) both -> early_stopping = True, alpha = 0.1
Train accuracy: 0.1395. Test accuracy: 0.1162
Loss: 2.1459
```

```
-----  10 neurons -----
(a) alpha = 0.1
Train accuracy: 0.9958. Test accuracy: 0.5012
Loss: 0.0713
(b) early_stopping = true
Train accuracy: 0.9127. Test accuracy: 0.3729
Loss: 0.2210
(c) both -> early_stopping = True, alpha = 0.1
Train accuracy: 0.8702. Test accuracy: 0.3414
Loss: 0.4090
```

```
-----  100 neurons -----
(a) alpha = 0.1
Train accuracy: 1.0000. Test accuracy: 0.6804
Loss: 0.0423
(b) early_stopping = true
Train accuracy: 0.9454. Test accuracy: 0.4915
Loss: 0.0453
(c) both -> early_stopping = True, alpha = 0.1
Train accuracy: 0.9454. Test accuracy: 0.4988
Loss: 0.1135
```

```
-----  200 neurons -----
(a) alpha = 0.1
Train accuracy: 1.0000. Test accuracy: 0.6707
Loss: 0.0392
(b) early_stopping = true
Train accuracy: 0.9557. Test accuracy: 0.4818
Loss: 0.0087
(c) both -> early_stopping = True, alpha = 0.1
Train accuracy: 0.9551. Test accuracy: 0.4867
Loss: 0.1013
```

Does this improve the results from the previous step? Which model would you choose now?

- ➔ Setup (b) and (c) does not improve the results from the previous step. Since for all number of hidden neurons, it performs worse than when no such setup was used. As for setup (a), it does improve the results from the previous step but only in certain cases. In general, as the number of hidden neurons increases, there is a greater improvement in performance when using setup (a) compared to not using it at all.

- ➔ The model I would choose now would be when the number of hidden neurons is 100 and alpha = 0.1. This is where the highest train and test accuracy was achieved out of all the models seen here.

1.1.4 Variability of the performance

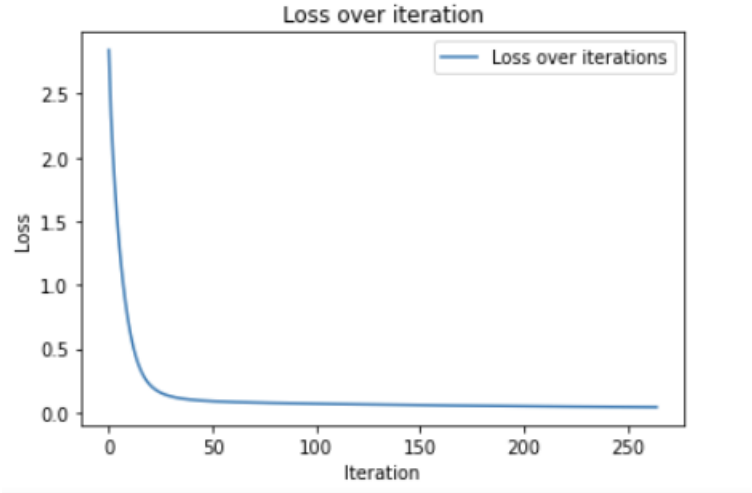
When changing the seed, what exactly changes?

➔ When changing the seed, the test accuracy varies the most. The train accuracy remains the same, whereas the loss does change as well but has lesser fluctuation compared to the test accuracy.

Report the minimum and maximum accuracy, and mean accuracy over 5 different runs and standard deviation i.e., (mean ± std).

```
Mean Accuracy on the train set: 1.0000 +/- 0.0000
Mean Accuracy on the test set: 0.6538 +/- 0.0080
Min accuracy on the train set: 1.0000 & Max accuracy on the train set: 1.0000
Min accuracy on the test set: 0.6392 & Max accuracy on the test set: 0.6610
```

1.1.5 Loss over iteration curve



1.1.6 Results from predictions on the test set

Classification report

```
Predicting on the test set
              precision    recall  f1-score   support

           0       0.82      0.80      0.81        40
           1       0.74      0.70      0.72        44
           2       0.60      0.58      0.59        48
           3       0.56      0.46      0.51        39
           4       0.71      0.81      0.76        37
           5       0.73      0.81      0.77        43
           6       0.49      0.50      0.49        44
           7       0.64      0.57      0.60        37
           8       0.49      0.60      0.54        30
           9       0.79      0.75      0.77        51

    accuracy                           0.66       413
   macro avg       0.66      0.66      0.66       413
weighted avg       0.66      0.66      0.66       413
```

Confusion Matrix

```
[[32  2  1  0  2  1  1  1  0  0]
 [ 0 31  1  2  0  1  6  1  1  1]
 [ 0  1 28  2  3  5  5  1  3  0]
 [ 1  1  4 18  0  0  6  0  9  0]
 [ 0  2  2  0 30  1  0  0  1  1]
 [ 1  1  2  1  0 35  2  1  0  0]
 [ 2  3  3  5  0  3 22  0  2  4]
 [ 3  0  2  0  3  1  0 21  3  4]
 [ 0  0  3  4  4  1  0  0 18  0]
 [ 0  1  1  0  0  0  3  8  0 38]]
```

How could you calculate yourself recall from support and confusion matrix entries?

- ➔ Recall is the number of true positives divided by the sum of the true positives and the false negatives.
  Thus, to calculate the recall for the i-th class, we will use the confusion matrix. We will take the i-th value in the i-th row; this will be the number of true positives.
  The false negatives will be all other values in its ith row, except for the i-th value.
  Hence, the sum of the true positives and false negatives would be the sum of all values in its i-th row. This sum is also called the support value in the classification report which is the total number of examples out of all examples in which the model predicted to be this class regardless of its correct/wrong prediction.

- ➔ Example: Class 0 (the 1st class)
  1st value in the 1st row: 32 (True positives)
  All other values in its ith row, except for the ith value: 2+1+0+2+1+1+1+0+0 = 8 (False negatives)
  Support: 32 + 8 = 40 (Same as the support score as seen in the classification report for class 0)

Recall: 32 / 40= 0.8 (Same as the recall score as seen in the classification report for class 0)

Explain in words what is recall.

➔ This means the rate at which the model was able to predict the truly relevant examples correctly.

What is the most misclassified image? State which class (digit) it was, and how you concluded that.

➔ Class 6 is the most misclassified image. As it has the lowest f1 score, where the f1 score considers both the precision and recall, since there is a tradeoff between them.
Therefore, in a whole, class 6 would be the most misclassified.

1.2 Model selection using GridSearch

1.2.1 Architectures

Number of architectures that will be checked: $C_1^2 \times C_1^4 \times C_1^2 \times C_1^2 = 32$

1.2.3 Best parameter and score

Best parameter:

```
{'activation': 'relu', 'alpha': 10, 'hidden_layer_sizes': 200, 'solver': 'adam'}
```

Best score: 0.6573657548125633

1.2.4 Test Accuracy

Train accuracy: 0.9406. Test accuracy: 0.7070

1.2.5 Theoretical Questions

What is the difference between hyperparameters and parameters of a model (in general)?

- ➔ Parameters are values that the model learns on its own. Hyperparameters are parameters set explicitly by the user to control the model.

Explain then the difference using the example of neural networks (i.e., name a few hyperparameters and parameters of neural networks)

- ➔ The number of layers and units, learning rate are considered hyperparameters of neural networks while weights, biases are considered as parameters.

2 Regression with Neural Networks

2.2 Function *calculate_mse*:

```
def calculate_mse(targets, predictions):
    """
    :param targets:
    :param predictions: Predictions obtained by using the model
    :return:
    """
    mse = mean_squared_error(y_pred=predictions, y_true=targets)
    return mse
```

2.3 Using GridSearch

```
n_hidden_neurons_list = [40, 48, 56]
```

Parameters:
`{'alpha': [0.002, 0.003, 0.004], 'hidden_layer_sizes': n_hidden_neurons_list, 'learning_rate_init: [0.0003, 0.0005, 0.0008], 'activation': ['relu', 'logistic'], 'solver': ['adam', 'lbfgs', 'sgd']}`

Best Model parameters:
`{'activation': 'logistic', 'alpha': 0.002, 'hidden_layer_sizes': 56, 'learning_rate_init': 0.0003, 'solver': 'lbfgs'}`
Best score: 0.9989232429730069

2.4 Final choice of model accuracy

Train MSE: 0.0004. Test MSE: 0.0004