COM4509/6509 Assignment 2023

Hello, this is programming assignment for *Machine Learning and Adaptive Intelligence*. This is worth 50% of the module grade, the remaining 50% will be assessed via the formal exam.

Deadline: 11th December 2023, 23:59

Please submit well before the deadline as there may be delays in the submission. Submission will be via Blackboard, the link will be made available closer to the deadline.

There are 2 parts to this assignment, covering different portions of the course. Both parts are worth 50 marks to give a combined total of 100 marks. Both contain a set of questions which will ask you to implement various machine learning algorithms that are covered throughout the course. You will receive marks for the correctness of your implementations, text based responses to certain questions and the quality of your code. Each question indicates how many marks are available for completing that questions.

Assignment help

If you are stuck and unsure what you need to do then please ask either in the lectures, labs or on the discussion board. There is a limit to what help we can provide but where possible we will give general guidance with how to proceed. We will also collect frequently asked questions here.

We are happy for you to discuss the assignment with other students but your code and test answers **must** be your own

What to submit

- You need to submit your **jupyter notebooks** and a **pdf** copy of it (not zipped together), named:

```
assignment_[username].ipynb
assignment_[username].pdf
```

replacing `[username]` with your username, e.g. `abc18de`.

- **Please execute the cells before your submission**. The **pdf** copy will be used as a backup in case the data gets corrupted and since we cannot run all the notebooks during marking. The best way to get a pdf is using Jupyter Notebook locally but if you are using Google Colab and are unable to download it to use Jupyter then you can use the Google Colab *file* → *print* to get a pdf copy.
- **Please do not upload** the data files used in this Notebook. We just want the python notebook *and the pdf*.

Late submissions

We follow the department's guidelines about late submissions, Undergraduate handbook link. PGT handbook link.

Use of unfair means

This is an individual assignment, while you may discuss this with your classmates, **please make sure you submit your own code**. You are allowed to use code from the labs as a basis of your submission.

"Any form of unfair means is treated as a serious academic offence and action may be taken under the Discipline Regulations." (from the students Handbook).

Reproducibility and readibility

Whenever there is randomness in the computation, you MUST set a random seed for reproducibility. Use your UCard number XXXXXXXXX (or the digits in your registration number if you do not have one) as the random seed throughout this assignment. You can set the seeds using torch.manual_seed(XXXXX) and np.random.seed(XXXXX). Answers for each question should be clearly indicated in your notebook. While code segments are indicated for answers, you may use more cells as necessary. All code should be clearly documented and explained. Note: You will make several design choices (e.g. hyperparameters) in this assignment. There are no “standard answers”. You are encouraged to explore several design choices to settle down with good/best ones, if time permits.

Enter your username (used for marking):

```
username = 'ACS23AB'
```

Part 1

Overview

This part of the assignment will focus on lecture 4.

This is the *first* of the two parts. Each part accounts for 50% of the overall coursework mark and this part has a total of 50 marks available. Attempt as much of this as you can. The questions below account for 45 marks. Your submitted code will also be scored based on conciseness, quality, efficiency and commenting (5 marks).

Assessment Criteria

The marks associated with each question are shown in square brackets. There are also 5 marks for code quality (including readability and efficiency).

You'll get marks for correct code that does what is asked and for text based answers to particular points. You should make sure any figures are plotted properly with axis labels and figure legends.

```
#We need to download a python file that contains some useful
functions.
```

```
!wget michaeltsmith.org.uk/assignment.py
```

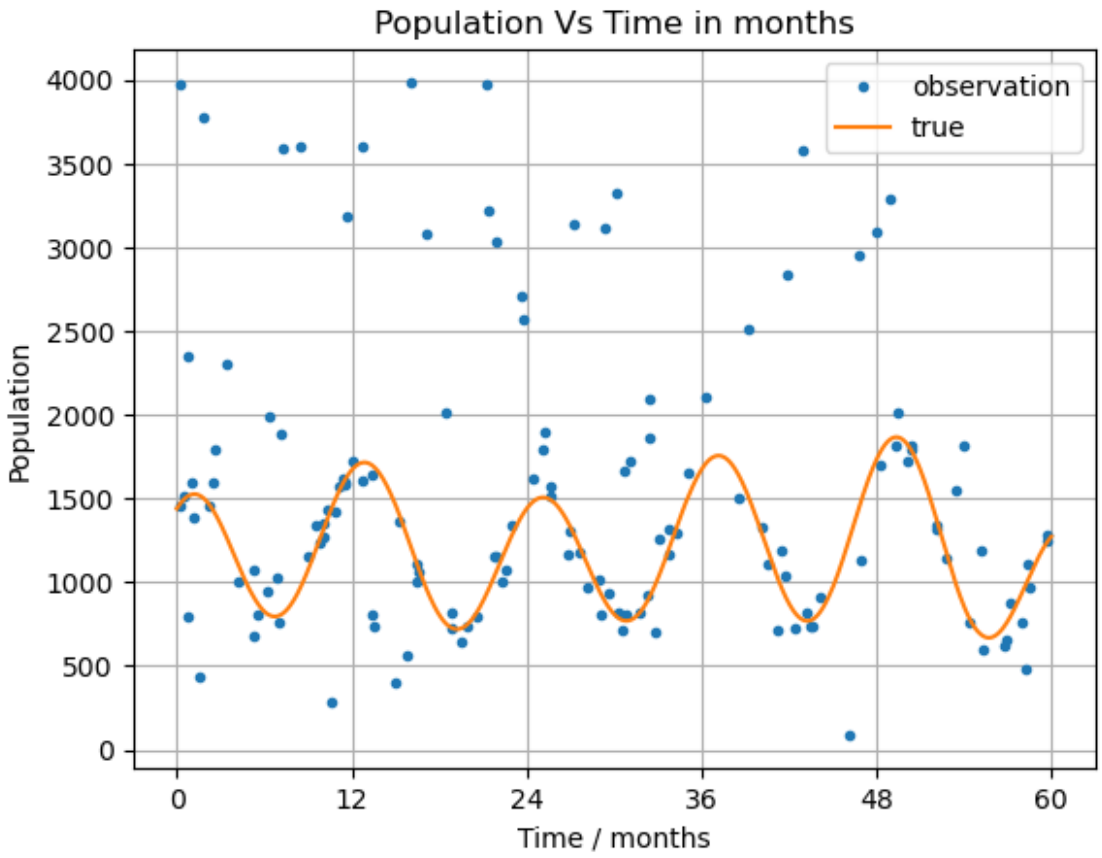
```
--2023-11-23 12:45:38--  http://michaeltsmith.org.uk/assignment.py
Resolving michaeltsmith.org.uk (michaeltsmith.org.uk)...
100.26.179.211
Connecting to michaeltsmith.org.uk (michaeltsmith.org.uk)|
100.26.179.211|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 17514 (17K) [text/x-python]
Saving to: 'assignment.py.5'

     0K .......... .......                                    100%
132K=0.1s

2023-11-23 12:45:39 (132 KB/s) - 'assignment.py.5' saved [17514/17514]
```

```
#and import some modules

import assignment
import numpy as np
import matplotlib.pyplot as plt
```

The Problem

Ecologists have monitored the population of Haggis on a particular mountain for five years. They have precise recordings (see `xtrue` and `ytrue`) and estimates from satellite remote sensing (`xdata` and `ydata`). They want to be able to *forecast* the *true population*, 12 months into the future.

```
xdata,ydata,xtrue,ytrue = assignment.data()

plt.plot(xdata,ydata,'.',label='observation')
plt.plot(xtrue,ytrue,'-',label='true')
plt.xticks(np.arange(0,61,12))
plt.xlabel('Time / months')
plt.ylabel('Population')
plt.title('Population Vs Time in months')
plt.legend()
plt.grid()
```



Question 1 [3 marks]

When developing your model for this problem, how could you split your data into training, validation and testing? (and why?) [max 30 words]

```
q1 = "The Haggis population dataset is split into training data
(excluding the last 12 months), true data(validation data) for
hyperparameter tuning (considering, last 12 months), and test data
(last 12 months)."
assignment.wc(q1)
```

```
30 words
```

Question 2: Gaussian Basis [9 marks]

In lab 4 you used a polynomial basis. The answer was of the form:

```
def polynomial(x, num_basis=4, data_limits=[-1., 1.]):
    Phi = np.zeros((x.shape[0], num_basis))
    for i in range(num_basis):
        Phi[:, i:i+1] = x**i
    return Phi
```

For this question, write a new function that creates a **Gaussian basis**.

Each basis function is of the form, $\exp\left[-\frac{(x-c)^2}{2w^2}\right]$. Where `c` is the centre of each Gaussian basis, and w is a constant (hyperparameter) that says how wide they are. You will want to space them uniformly across the domain specified by `data_limits`. So if `data_limits = [-2, 4]` and `num_basis = 4`. The centres will be at, -2 0 2 4.

Note: For now **we'll not have a constant term** (this will be ok if you standardise your data, as the mean will be zero).

```
def gaussian(x, num_basis=4, data_limits=[-1., 1.], width = 10):
    gaussian_means =
np.linspace(data_limits[0],data_limits[1],num_basis)
    #To do: Implement
    Phi = np.zeros((x.shape[0], num_basis))
    for i in range(num_basis):
        Phi[:,i] = np.exp(-(x - gaussian_means[i])**2/(2*width**2))
    return Phi

assignment.checkQ2(gaussian)
```

```
Success
```

Question 3: Ordinary Least Squares Regression [7 marks]

Rather than compute the closed form solution we will compute the gradient and use gradient descent for ridge regression (L2 regularisation).

First, write a function to compute the gradient of the sum squared error wrt a parameter vector w. Given it has L2 regularisation (with regularisation parameter λ).

To get you started, here is the $L2$ regularised cost function:

$$E=(y-\Phi w)^{\top}(y-\Phi w)+\lambda w^{\top} w$$

```
def grad_ridge(Phi,y,w,lam):
    """
    Return an D dimensional vector of gradients of w, assuming we want
to minimise the sum squared error
    using the design matrix in Phi; under ridge regression with
regularisation parameter lambda.
    Arguments:
     - Phi, N x D design matrix
     - y, training outputs
     - w, parameters (we are finding the gradient at this value of w)
     - lam, the lambda regularisation parameter.
    """
    dedw = 2*(-Phi.T@y + Phi.T@Phi@w + lam*w)
    return dedw #To do: Implement
```

```
assignment.checkQ3(grad_ridge)
```

```
Success
```

This `grad_descent` function uses gradient descent to minimise the cost function (optimise using an appropriate learning rate).

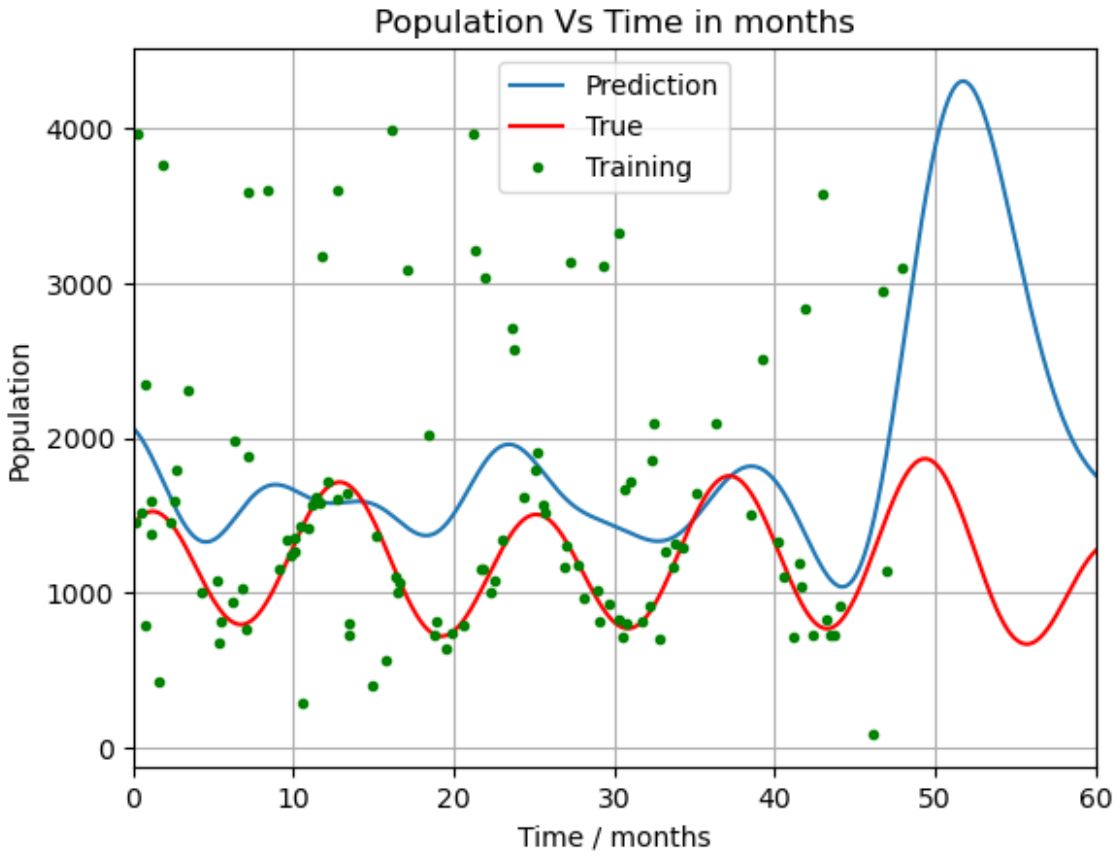
```
def grad_descent(grad_fn,Phi,y,lam):
    """
    Compute optimised w.
    Parameters:
       - grad, the gradient function
       - Phi, design matrix (shape N x D)
       - y, vector of observations (length N)
       - lam, regularisation parameter, lambda.
    Returns
       - w_optimsed, a vector (length D) that minimises the ridge
regression cost function
    """
    w = np.zeros(Phi.shape[1])
    for it in range(10000):
        g = grad_fn(Phi,y,w,lam)
        w-=0.0001*g
    return w
```

Let's see how we're doing...

In this code I standardise the training data labels, and use the methods you have written to make predictions for all the `true` data. Note that I'm holding out the last 12 months to see how the model looks for forecasting. I've also not used any validation, but instead have just used fixed value of the hyperparameters.

```
xtrain = xdata[xdata<48]
ytrain = ydata[xdata<48]
xval = xtrue[xtrue>=48]
yval = ytrue[xtrue>=48]

data_mean = np.mean(ytrain)
data_std = np.std(ytrain)
ytrain_standardised = (ytrain - data_mean)/data_std

Phi = gaussian(xtrain,120,[0,60],3)
w = grad_descent(grad_ridge,Phi,ytrain_standardised,0.01)
truePhi = gaussian(xtrue,120,[0,60],3)
plt.plot(xtrue,(truePhi @ w)*data_std+data_mean,label='Prediction')
plt.plot(xtrue,ytrue,'-r',label='True')
plt.plot(xtrain,ytrain,'.g',label='Training')
plt.xlabel('Time / months')
```

```
plt.ylabel('Population')
plt.title('Population Vs Time in months')
plt.legend()
plt.grid()
plt.xlim([0,60])
s = 0
for i in range(len(truePhi)):
    k = (truePhi[i] @ w)*data_std+data_mean
    s += (ytrue[i] - k)*(ytrue[i] - k)
sum_sqerror = s
```



There are two more tasks to do:

1) handle the outliers 2) Use a better basis

Question 4 [5 marks]

Let's use the sum of absolute errors, rather than the sum squared error, as the cost function. We will also keep the L2 regulariser. So the cost function can be:

$$E = \sum_{i=1}^{N} ¿ \left[\Phi \right)_i w - y_i \vee ¿ + \lambda w^{\top} w ¿$$

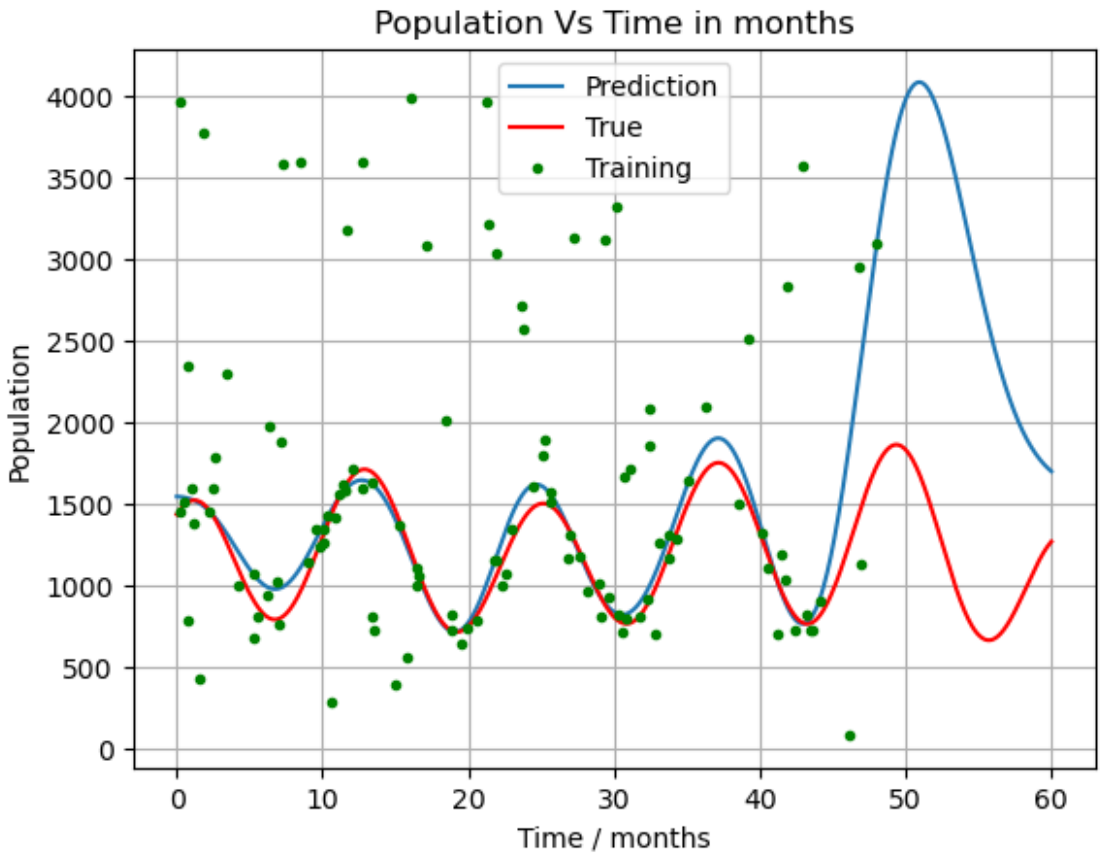
Write down a function that computes the gradient of this function wrt w.

```
def grad_abs(Phi,y,w,lam):
    """
    Return an D dimensional vector of gradients of w, assuming we want
to minimise the sum absolute error
    using the design matrix in Phi; under L2 regularisation parameter
lambda.
    Arguments:
     - Phi, N x D design matrix
     - y, training outputs
     - w, parameters (we are finding the gradient at this value of w)
     - lam, the lambda regularisation parameter.
    """
    gradient1 = np.zeros_like(w)
    Phi1 = np.zeros_like(Phi)
    for i in range(len(Phi)):
        c = Phi[i]@w - y[i]
        Phi1[i] = np.sign(c)*Phi[i]
        gradient1 += Phi1[i]

    dedw = gradient1 + 2*lam*w
    return dedw #To do: Implement

assignment.checkQ4(grad_abs)
```

```
Success
```

Let's see what the result looks like, using the absolute error:

```
Phi = gaussian(xtrain,120,[0,60],3)
w = grad_descent(grad_abs,Phi,ytrain_standardised,0.01)
truePhi = gaussian(xtrue,120,[0,60],3)
plt.plot(xtrue,(truePhi @ w)*data_std+data_mean,label='Prediction')
plt.plot(xtrue,ytrue,'-r',label='True')
plt.plot(xtrain,ytrain,'.g',label='Training')
plt.xlabel('Time / months')
plt.ylabel('Population')
plt.title('Population Vs Time in months')
plt.legend()
plt.grid()
s = 0
for i in range(len(truePhi)):
    s += np.abs((truePhi[i] @ w)*data_std+data_mean - ytrue[i])
sum_aerr = s
```



Question 5 [3 marks]

Comment on this result in terms why this result appears better than the sum-squared cost function [max 30 words]

```
q5 = "For datasets with outliers, sum-absolute cost-function could be
favoured over the sum-squared cost-function in training. Its
robustness to outliers makes it a better choice, minimizing their
impact during optimization."
assignment.wc(q5)
```

```
29 words
```

Question 6 [7 marks]

To improve its ability to forecast we observe that there seems to be an annual oscillation in the data. Can you create a basis that combines both Gaussian bases *AND* sinusoidal bases *of the appropriate wavelength*. Please use half of the `num_basis` for the Gaussian bases, and the other half for the sinusoidal ones. All the sinusoidal bases should have a 12 month period, but with a range of offsets (uniformly distributed between 0 and 6, but not including 6).

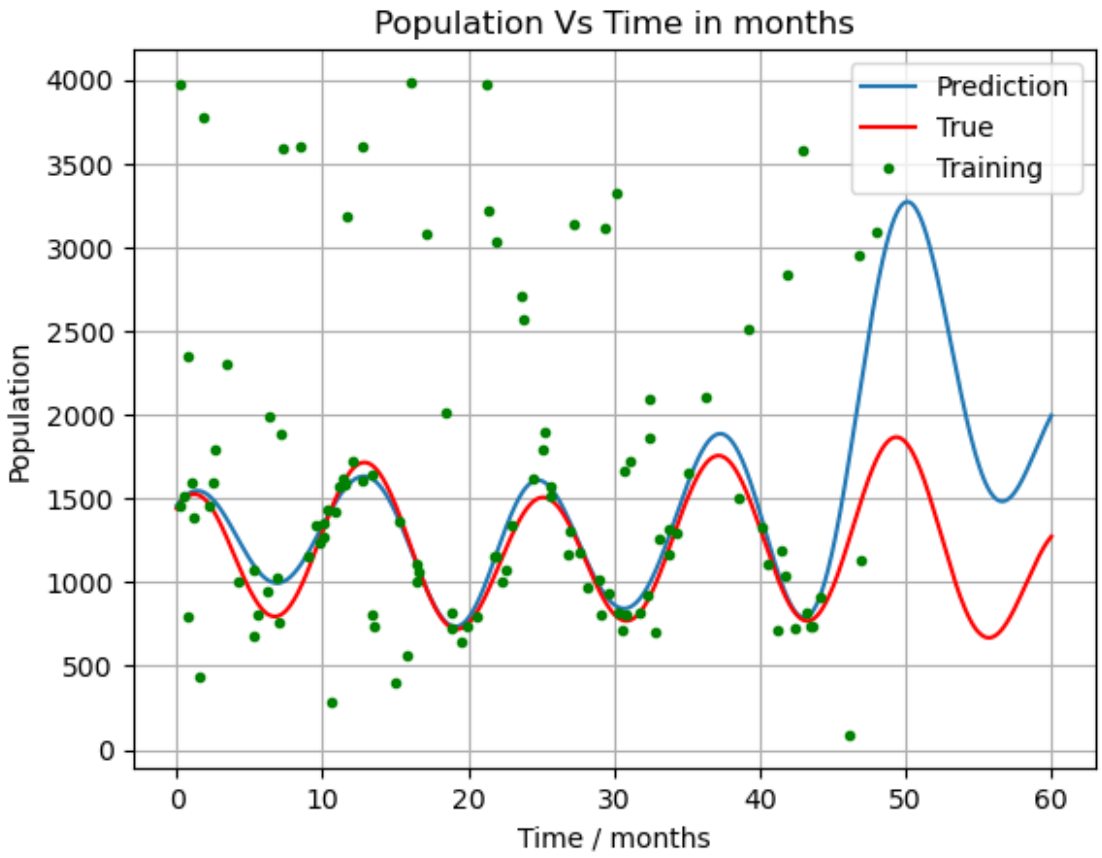
```
def gaussian_and_sinusoidal(x, num_basis=4, data_limits=[-1., 1.],
width = 10):
```

```
    """
    Return an N x D design matrix.
    Arguments:
     - x, input values (N dimensional vector)
     - num_basis, number of basis functions (specifies D)
     - data_limits, a list of two numbers, specifying the minimum and
maximum of the data input domain.
     - width, the 'spread' of the Gaussians in the basis

    Half the bases are Gaussian, half are evenly spaced cosines of 12
month period (offset by between 0 to 6 months)
    """
    #To do: Implement
    Phi = np.zeros((x.shape[0], num_basis))
    #Gaussian Basis Function
    Phi[:,0:num_basis//2] = gaussian(x, num_basis//2, data_limits,
width)
    offset = np.linspace(0,6,num_basis//2,endpoint=False)

    #sinusoidal basis function
    for i in range(num_basis//2):
        Phi[:,i+num_basis//2] = np.sin(2*np.pi*(x-offset[i])/12)
    return Phi
```

Let's see how this has affected the result:

```
Phi = gaussian_and_sinusoidal(xtrain,120,[0,60],3)
w = grad_descent(grad_abs,Phi,ytrain_standardised,0.01)
truePhi = gaussian_and_sinusoidal(xtrue,120,[0,60],3)
plt.plot(xtrue,(truePhi @ w)*data_std+data_mean,label='Prediction')
plt.plot(xtrue,ytrue,'-r',label='True')
plt.plot(xtrain,ytrain,'.g',label='Training')
plt.xlabel('Time / months')
plt.ylabel('Population')
plt.title('Population Vs Time in months')
plt.legend()
plt.grid()
s = 0
for i in range(len(truePhi)):
    s += np.abs((truePhi[i]@w)*data_std+data_mean - ytrue[i] )
sae = s #sum of absolute error
print(f'{sae:0.2f}')
```

```
305107.99
```



Question 7 [11 marks]

We now need to select the parameters.

Write some code that:

- Selects good parameters
- Draws a graph of the result

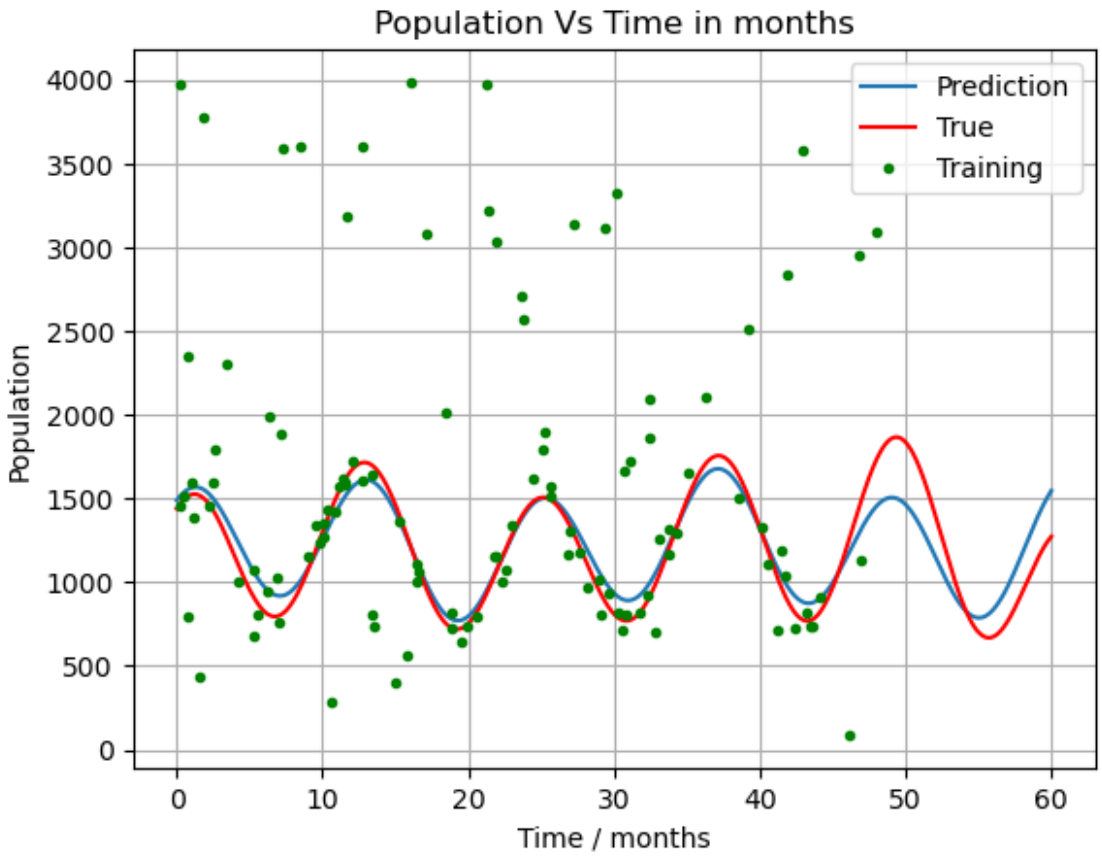
For this question you will need to:

- Decide on how you will select:
  - an appropriate number of bases
  - an appropriate Gaussian basis width
  - an appropriate regularisation term
- (you might want to use a validation set)
- Decide how you will split your data into training and validation. You could use the approach we used at the end of Q3. Remember: You are given the true underlying function, in `xtrue` and `ytrue`, so it is a comparison with that which matters. Remember also that you want to do well at **forecasting**!
- Plot a graph showing (a) the training points used; (b) the true population (`truex`, `truey`); and (c) your predictions.

```
#code here
#Train-Validation dataset split
xtrain = xdata[xdata<48]
ytrain = ydata[xdata<48]
xval = xtrue[xtrue>=48]
yval = ytrue[xtrue>=48]
x1 = xtrue[xtrue<48]
y1 = ytrue[xtrue<48]

data_mean = np.mean(ytrain)
data_std = np.std(ytrain)
ytrain_standardised = (ytrain - data_mean)/data_std

#Validation Data Hyper-parameter tuning
num = [50,100,200,300]
lamd = [0.005,0.01,0.05,0.1]
wid = [2.5,3,5.5,10]
Phi = gaussian_and_sinusoidal(xtrain,num[0],[0,60],wid[0])
w = grad_descent(grad_abs,Phi,ytrain_standardised,lamd[0])
valPhi = gaussian_and_sinusoidal(xval,num[0],[0,60],wid[0])
best_params = {}
s = 0
for i in range(len(valPhi)):
    s += np.abs((valPhi[i]@w)*data_std+data_mean - yval[i] )
e1 = s #sum of absolute error
print(f'Initialized Validation Error: {e1:0.2f}')
best_params['val_error'] = e1
best_params['num_basis'] = num[0]
best_params['lambda'] = lamd[0]
best_params['width'] = wid[0]
for i in num:
    for j in lamd:
        for k in wid:
            Phi = gaussian_and_sinusoidal(xtrain,i,[0,60],k)
            w = grad_descent(grad_abs,Phi,ytrain_standardised,j)
            valPhi = gaussian_and_sinusoidal(xval,i,[0,60],k)
            s = 0
            for l in range(len(valPhi)):
                s += np.abs((valPhi[l]@w)*data_std+data_mean - yval[l]
)
            e = s #sum of absolute error
            if e < e1 :
                best_params['val_error'] = e
                best_params['num_basis'] = i
                best_params['lambda'] = j
                best_params['width'] = k
                e1 = e
                print(f'Updated Validation Error: {e:0.2f}')

print('\nThe Best parameters obtained after validation:\
```

```
n',best_params)
num_basis = best_params['num_basis']
lam = best_params['lambda']
width = best_params['width']

#Testing on True data and forecasting for the last 12 months
Phi = gaussian_and_sinusoidal(xtrain,num_basis,[0,60],width)
w = grad_descent(grad_abs,Phi,ytrain_standardised,lam)
truePhi = gaussian_and_sinusoidal(xtrue,num_basis,[0,60],width)
plt.plot(xtrue,(truePhi @ w)*data_std+data_mean,label='Prediction')
plt.plot(xtrue,ytrue,'-r',label='True')
plt.plot(xtrain,ytrain,'.g',label='Training')
plt.xlabel('Time / months')
plt.ylabel('Population')
plt.title('Population Vs Time in months')
plt.legend()
plt.grid()
s = 0
for i in range(len(truePhi)):
    s += np.abs((truePhi[i]@w)*data_std+data_mean - ytrue[i] )
sae = s #sum of absolute error
print(f'\nThe Sum of absolute error for True data: {sae:0.2f}')
```

```
Initialized Validation Error: 122113.97
Updated Validation Error: 87986.17
Updated Validation Error: 52186.51
Updated Validation Error: 50323.05
Updated Validation Error: 45197.74
Updated Validation Error: 44823.77

The Best parameters obtained after validation:
 {'val_error': 44823.76644863585, 'num_basis': 300, 'lambda': 0.01,
'width': 10}

The Sum of absolute error for True data: 98474.09
```



Part 2

This is the *second* of the two parts. Each part accounts for 50% of the overall coursework mark and this part has a total of 50 marks available. Attempt as much of this as you can, each of the questions are self-contained and contain some easier and harder bits so even if you can't complete Q1 straight away then you may still be able to progress with the other questions.

Overview

This part of the assignment will cover:

- Q1: Dimensionality reduction and clustering (lectures 8 and 9)
- Q2: Classification and neural networks (lectures 6, 7 and 8)

Assessment Criteria

- The marks for this part are distributed as follows:
  - **Q1**: 20 marks
  - **Q2**: 25 marks
  - **Code quality** (including readability and efficiency): 5 marks

- You'll get marks for correct code that does what is asked and for text based answers to particular points. We are not overly concerned with model performance but you should still aim to get the best results you can for your chosen approaches. You should make sure any figures are plotted properly with axis labels and figure legends.

If you are unsure about how to proceed then please ask. We will compile a list of

Question 1: Clustering and dimensionality reduction [20 marks]

For this question you are asked apply a **clustering algorithm** of your choice (e.g K-means or spectral clustering) to a dataset with a large number of features, then apply a **dimensionality reduction** method (e.g PCA, Auto-encoder) to plot the clusters in a reduced feature space.

The dataset that you will be using is the UCI Human Activity Recognition dataset (link) which contains measurements using smartphone sensors during certain activities. The data has been pre-processed to give **561** features, representing many different aspects of the sensor dynamics. While this is a timeseries we will only consider individual samples, of which there are **7352** in the training set. This has been provided on Blackboard and can be downloaded as a compressed .npz file.

What you need to do

This question is split into 4 sub-parts, each will be marked based not only on the correctness of your code solution but a short text response to either justify the algorithms used or a discussion of the results of your code. The 4 parts to this questions are: 1) Choosing and applying a clustering algorithm to the data and justifying your approach. 2) Analysing the quality of the clustering solution and discussing the results. 3) Choosing and applying a dimensionality reduction technique and justifying your approach. 4) Plotting the clusters in the reduced feature space and discussing the plots.

```
import numpy as np
import matplotlib.pyplot as plt

dataset = np.load('./UCI_HAR.npz')

x_train = dataset['x_train']
y_train = dataset['y_train']

print(f'The training set contains {x_train.shape[0]} samples, each
with {x_train.shape[1]} features.')
print(f'There are {len(np.unique(y_train))} classes.')
```

```
The training set contains 7352 samples, each with 561 features.
There are 6 classes.
```

1.1 Clustering of the data [5 marks]

Choose a clustering algorithm (either one from class or an appropriate one from elsewhere) and apply it to this dataset. You will need to perform some analysis to select any necessary hyper-parameters.

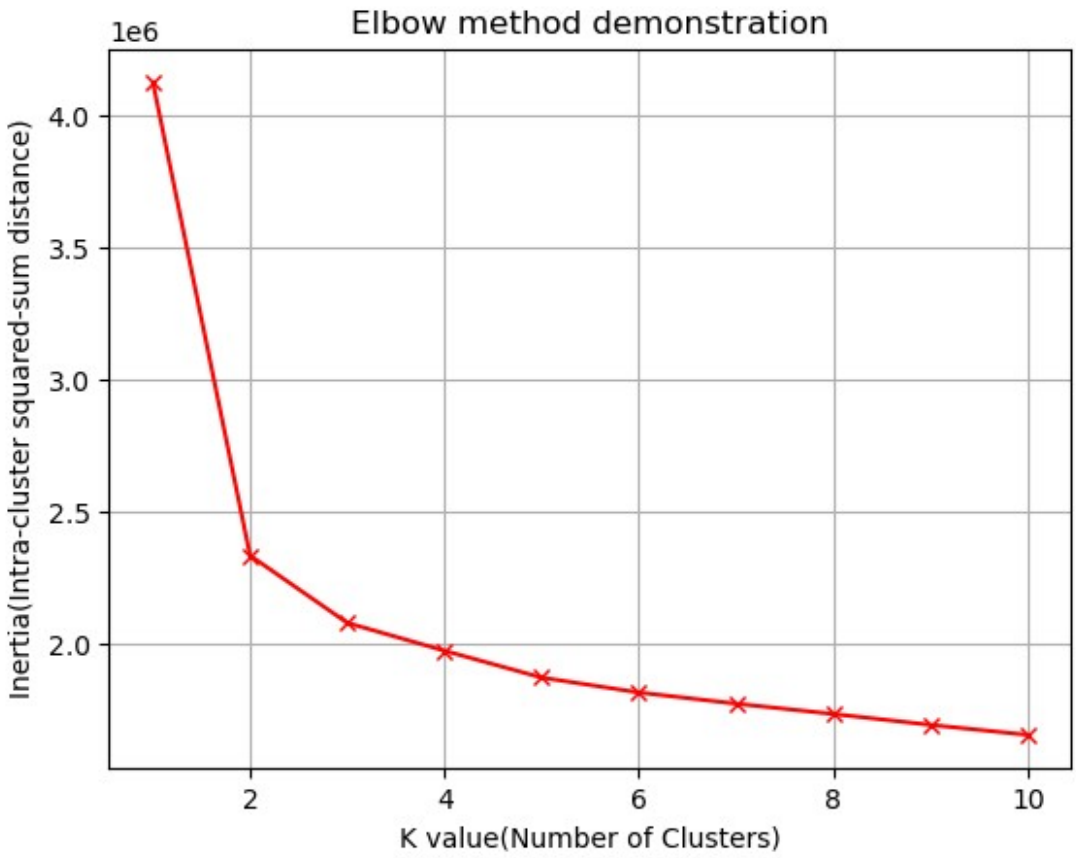
```
from sklearn.cluster import
KMeans,SpectralClustering,AgglomerativeClustering
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x_train_standard = scaler.fit_transform(x_train)
n_clusters = 6 # Assuming 6 classes

model1 =
KMeans(n_clusters=n_clusters,n_init=10,random_state=42,init='k-means+
+')
model2 =
SpectralClustering(n_clusters=n_clusters,affinity='nearest_neighbors')
model3 = AgglomerativeClustering(n_clusters=n_clusters)

scores = {}

scores[silhouette_score(x_train_standard,model1.fit_predict(x_train_st
andard))] = model1
scores[silhouette_score(x_train_standard,model2.fit_predict(x_train_st
andard))] = model2
scores[silhouette_score(x_train_standard,model3.fit_predict(x_train_st
andard))] = model3
print(scores)
print('\n')
model_best = scores[max(scores.keys())]

print('The best model obtained is:\n',model_best,'\nSilhoutte Score
=', max(scores.keys()))
```

```
{0.10864106318641077: KMeans(n_clusters=6, n_init=10,
random_state=42), 0.07337559327574601:
SpectralClustering(affinity='nearest_neighbors', n_clusters=6),
0.0833964923997774: AgglomerativeClustering(n_clusters=6)}


The best model obtained is:
 KMeans(n_clusters=6, n_init=10, random_state=42)
Silhoutte Score = 0.10864106318641077
```

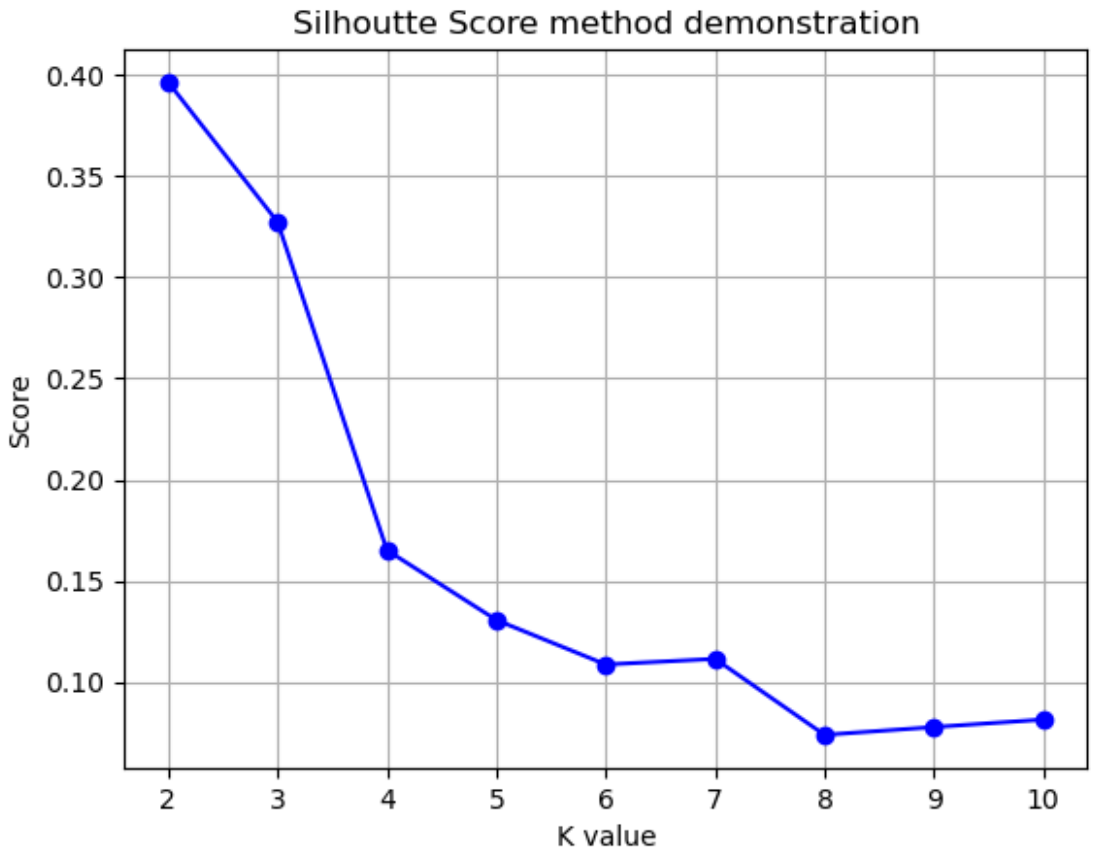
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
```

```
K = list(range(2,11))

score_model = {}
scaler = StandardScaler()
x_train_standard = scaler.fit_transform(x_train)

inertia_k = []
model1 = KMeans(n_clusters=1, n_init=10,random_state=42,init='k-means+
+')
model1.fit_predict(x_train_standard)
inertia_k.append(model1.inertia_)

for i in K:
    model1 = KMeans(n_clusters=i, n_init=10,random_state=42,init='k-
means++')
    model1.fit_predict(x_train_standard)
    score_model[silhouette_score(x_train_standard,model1.labels_)] =
model1
    inertia_k.append(model1.inertia_)

k = range(1,11)
plt.plot(k,inertia_k,'rx-')
plt.grid()
plt.xlabel('K value(Number of Clusters)')
plt.ylabel('Inertia(Intra-cluster squared-sum distance)')
plt.title('Elbow method demonstration')
```

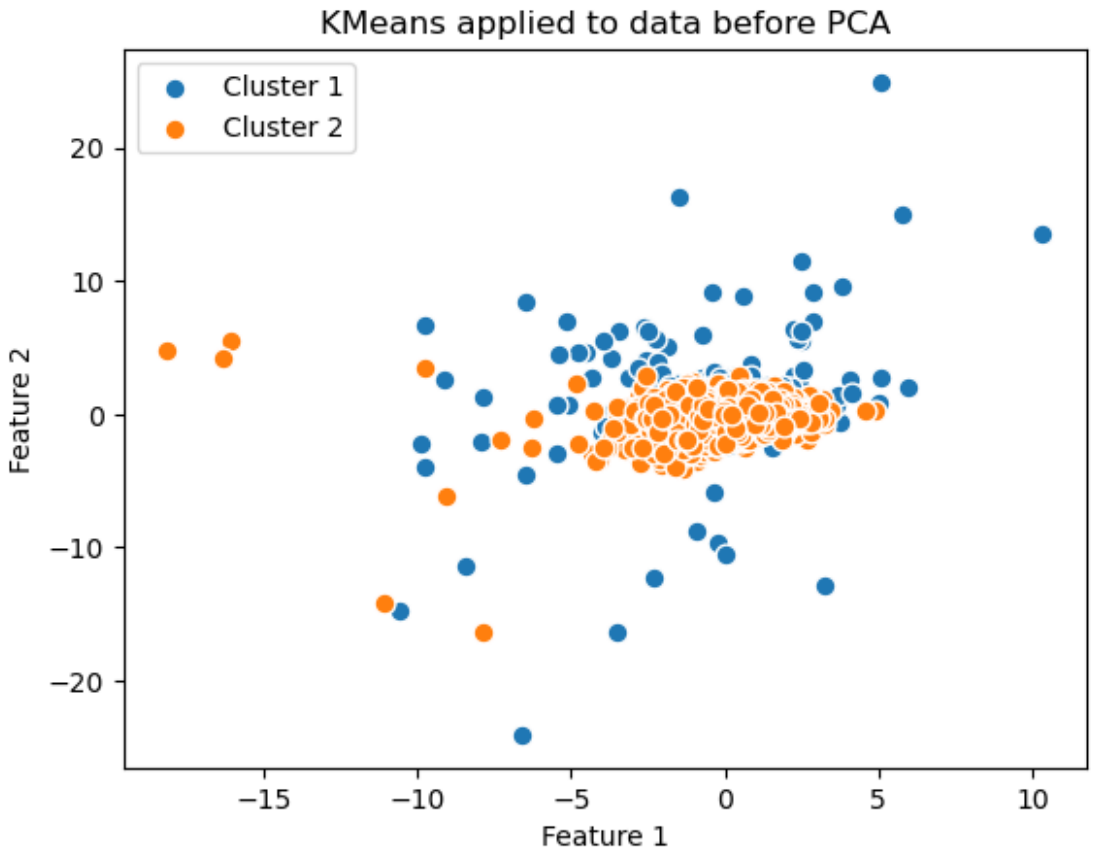
```
Text(0.5, 1.0, 'Elbow method demonstration')
```




```
k = range(2,11)
plt.plot(k,score_model.keys(),'bo-')
plt.grid()
plt.xlabel('K value')
plt.ylabel('Score')
plt.title('Silhoutte Score method demonstration')

Text(0.5, 1.0, 'Silhoutte Score method demonstration')
```




```
model = score_model[max(score_model.keys())]
model.fit_predict(x_train_standard)
clusters = np.unique(model.labels_)
for i in range(len(clusters)):
    cluster_points = x_train_standard[model.labels_ == i]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1],
label=f'Cluster {i + 1}', s=60, edgecolors='w')
plt.legend()
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('KMeans applied to data before PCA')

Text(0.5, 1.0, 'KMeans applied to data before PCA')
```



In the following markdown block, provide a justification of the algorithm that you selected and of any hyper-parameters that you have selected.

Write your answer here.

Justification of Clustering Algorithm and Hyperparameters used

In the initial comparison of clustering algorithms—K-Means, Spectral, and Agglomerative—K-Means emerged as the top performer, exhibiting the highest Silhouette Score for identical cluster numbers. Hyperparameter tuning for K-Means involved determining the optimal value of K (number of clusters). Utilising the Elbow method and Silhouette scoring across a range of clusters (2 to 10), K = 2 emerged as the optimal choice. The Elbow method demonstrated a diminishing rate of inertia value decrease (intra-cluster squared-sum distance) from K = 2 onwards, supporting its selection as the optimum value. Concurrently, Silhouette scores across different cluster values highlighted that the model with K = 2 clusters achieved the maximum score, reinforcing the conclusion that K = 2 is the most suitable number of clusters. Using both the Elbow method and Silhouette scoring together shows that K-Means clustering works well for this dataset's specific characteristics.

1.2 Analysis of the clustering quality [5 marks]

Using an appropriate analysis metric (e.g, cluster purity, the labels are available to use in the `y_train` array), measure the quality of the clustering.

```
# Program your cluster quality metric here
from sklearn.metrics import
normalized_mutual_info_score,homogeneity_score,adjusted_rand_score
nmi = normalized_mutual_info_score(y_train, model.labels_)
hom = homogeneity_score(y_train, model.labels_)
ars = adjusted_rand_score(y_train, model.labels_)

print(f'Normalized Mutual Information: {nmi:0.2f}')
print(f'Homogeneity Score: {hom:0.2f}')
print(f'Adjusted Rand Score: {ars:0.2f}')
```

```
Normalized Mutual Information: 0.54
Homogeneity Score: 0.38
Adjusted Rand Score: 0.33
```

Write a short discussion of these results commenting on the clustering performance, the relevance of your chosen analysis metric and any conclusions you have about the clustering of the data.

Write your discussion here

Purity Scores Analysis

The cluster purity scores offer insights into the clustering performance:

1) *Normalized Mutual Information (NMI)*: A NMI of 0.543 indicates a moderately good correlation between the true class labels (y_train) and the predicted cluster labels (model.labels_). NMI values closer to 1 signify a perfect correlation.

2) *Homogeneity Score*: The homogeneity score of 0.376 suggests that the purity of clusters is not particularly high, indicating the possibility of mixed classes within clusters. A desirable score approaches 1, indicating a homogeneous assignment of samples to clusters.

3) *Adjusted Rand Score*: The adjusted Rand score of 0.328 is moderately low, implying that the accuracy concerning ground truth labels is not optimal. Scores closer to 1 would indicate a better alignment between predicted clusters and actual class labels.

In conclusion, although the NMI shows a decent correlation, the homogeneity and adjusted Rand scores reveal areas to enhance. This suggests that some clusters have a mix of different classes, and the accuracy in aligning with the actual labels could be improved.

1.3 Training a dimensionality reduction method [5 marks]

Now you will need to choose a dimensionality reduction method that is able to reduce the number of features down to **3**. Again, where necessary you will need to select appropriate hyper-parameters.

```
# Program your dimensionality reduction here.
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=3)
x_train_pca = pca.fit_transform(x_train_standard)
```

Theoretical Representation of Principal Component Analysis(PCA)

```
#Creating Covariance matrix
covar_mat = np.cov(x_train_standard,rowvar=False)

#Creating Eigen Values and Eigen Vectors from the matrix
eig_val,eig_vec = np.linalg.eig(covar_mat)

#Arranging the Eigen Vectors in Descending order of Eigen Values
si = np.argsort(eig_val)[::-1]
eig_vec_sorted = eig_vec[:, si]

#Obtaining Projection Matrix
n_components = 3
pro_mat = eig_vec_sorted[:,:n_components]

#Projecting the data to the Projected Matrix
pro_mat = np.array(pro_mat)
x_train_pca_theo = -x_train_standard@pro_mat
```

In the following markdown block, provide a justification for the dimensionality reduction technique that you have used and (if any) how you selected your hyper-parameters. Be clear as to the advantages and disadvantages to your approach.

Write your discussion here

Justification for the dimensionality reduction technique

Principal Component Analysis (PCA) was chosen for dimensionality reduction due to its notable advantages:

1) PCA transforms original features into uncorrelated components, capturing maximum variance and allowing effective feature compression.

2) The linear combinations of features obtained from PCA are easily interpretable. In contrast, autoencoders may lack interpretability as their hidden layers may not have a clear relationship with the original features.

3) PCA is suited for unsupervised learning, requiring no labeled data, while autoencoders rely on labeled data, making PCA more versatile.

However, consider the following disadvantages of PCA:

1) PCA assumes a linear relationship between features, which may limit its efficiency in capturing non-linear patterns in the data.

2) PCA is sensitive to outliers, impacting the determination of principal components as it aims to maximize variance.

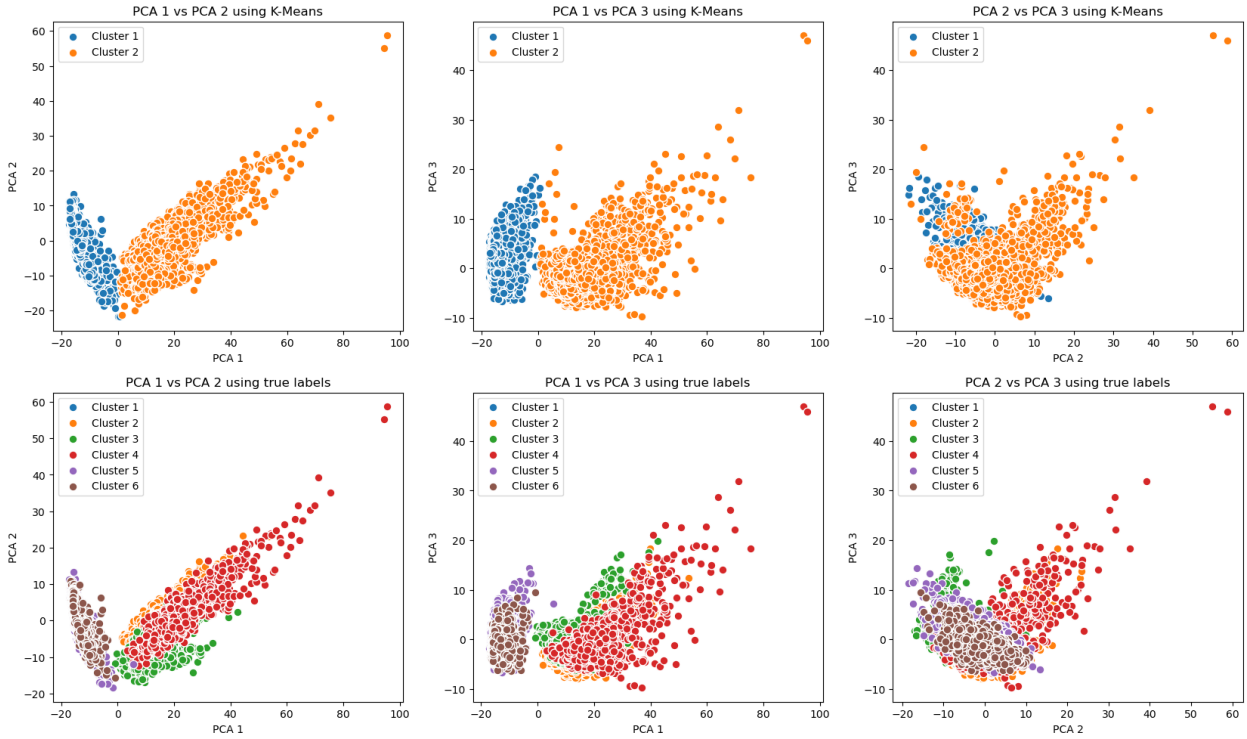
In summary, while PCA offers interpretability, versatility, and efficiency in linear relationships, its sensitivity to outliers and limitation in capturing non-linear patterns should be considered based on the characteristics of the dataset and the objectives of dimensionality reduction.

1.4 Plotting the clusters in the reduced feature space [5 marks]

Now that you have transformed your data into 3 dimensions, create a set of plots to show the clusters in these reduced dimensions. Make separate plots using the clustering labels from part 1.1 and also the ground truth labels to show how well it has been clustered. Where possible combine the figures in sensible ways using subplots.

Plot these as a set of 2d plots of the combinations of all the reduced dimensions. You may additionally plot this as a 3d plot, if this helps with the visualisation.

```
# Program your plots here
from itertools import combinations
list1 = list(range(0,3))
combination_pairs = list(combinations(list1, 2))
plt.figure(figsize=[5,5])
#plotting using the 1.1 labels
clusters = np.unique(model.labels_)
for pairs in combination_pairs:
    for i in range(len(clusters)):
        plt.subplot(2,3,combination_pairs.index(pairs)+1)
        cluster_points = x_train_pca[model.labels_ == i]
        plt.scatter(cluster_points[:, pairs[0]], cluster_points[:,
pairs[1]], label=f'Cluster {i + 1}', s=60, edgecolors='w')
        plt.xlabel(f'PCA {pairs[0] + 1}')
        plt.ylabel(f'PCA {pairs[1] + 1}')
        plt.title(f'PCA {pairs[0] + 1} vs PCA {pairs[1] + 1} using K-
Means')
        plt.legend()

#plotting using ground truth labels
clusters = np.unique(y_train)
for pairs in combination_pairs:
    for i in range(len(clusters)):
        plt.subplot(2,3,combination_pairs.index(pairs)+4)
        cluster_points = x_train_pca[y_train == i]
        plt.scatter(cluster_points[:, pairs[0]], cluster_points[:,
pairs[1]], label=f'Cluster {i + 1}', s=60, edgecolors='w')
        plt.xlabel(f'PCA {pairs[0] + 1}')
        plt.ylabel(f'PCA {pairs[1] + 1}')
        plt.title(f'PCA {pairs[0] + 1} vs PCA {pairs[1] + 1} using
true labels')
        plt.legend()

plt.subplots_adjust(left=0.95, bottom=0.2, right=4.1, top=2)
```



Write a short comment on your plots, evaluating the performance of the dimensionality reduction and how well the clustering has done in this visualisation. Are there any key conclusion spanning the whole question that you can draw?

Write your discussion here

Analysis of the plots for K-Means Clustering Algorithm

The plots above show the outcomes of using the K-Means Clustering labels and the actual (ground-truth) labels on data that has been simplified using a method called PCA (dimensionality reduction).

The first three graphs demonstrate how the algorithm organized the simplified data into two groups according to the labels assigned by the K-Means Algorithm before using PCA. The bottom three graphs show the clustering of the same data, but this time using the real labels (y_train).

Looking at the dataset (UCI Human Activity Recognition) and examining the ground-truth labels, we find that there are six different classes representing various activities. However, K-Means seems to have simplified this complexity by identifying two main groups in the data: 'movement' and 'no movement.' It appears that the algorithm has merged distinct classes into broader clusters, oversimplifying the true class structure.

The ground-truth labels indicate that the classes are not entirely separated in the reduced dimensional space, suggesting some overlap between them. While PCA effectively captures a significant amount of variance in the first few components, the linear relationships between these components may not fully account for the underlying separability of the classes.

Question 2: Classification and neural networks [25 marks]

This second questions will look at implementing classifier models via supervised learning to correctly classify images. We will be using images from the MedMNIST dataset which contains a range of health related image datasets that have been designed to match the shape of the original digits MNIST dataset. Specifically we will be working with the BloodMNIST part of the dataset. The code below will download the dataset for you and load the numpy data file. The data file will be loaded as a dictionary that contains both the images and labels already split to into training, validation and test sets. The each sample is a 28 by 28 RGB image and are not normalised. You will need to consider any necessary pre-processing.

Your task in this questions is to train **at least 4** different classifier architectures (e.g logistic regression, fully-connected network etc) on this dataset and compare their performance. These can be any of the classifier models introduced in class or any reasonable model from elsewhere. You should consider 4 architectures that are a of suitable variety i.e simply changing the activation function would score lower marks than trying different layer combinations.

This question will be broken into the following parts:

1. A text description of the model architectures that you have selected and a justification of why you have chosen them. Marks will be awarded for suitability, variety and quality of the architectures.
2. The training of the models and the optimisation of any hyper-parameters.
3. A plot comparing the accuracy and error (or loss), on separate graphs, of the different architectures and a short discussion of the results.

```
import numpy as np
import matplotlib.pyplot as plt
import urllib.request
import os

# Download the dataset to the local folder
if not os.path.isfile('./bloodmnist.npz'):

urllib.request.urlretrieve('https://zenodo.org/record/6496656/files/
bloodmnist.npz?download=1', 'bloodmnist.npz')

# Load the compressed numpy array file
dataset = np.load('./bloodmnist.npz')

# The loaded dataset contains each array internally
for key in dataset.keys():
    print(key, dataset[key].shape, dataset[key].dtype)
```

```
train_images (11959, 28, 28, 3) uint8
train_labels (11959, 1) uint8
val_images (1712, 28, 28, 3) uint8
val_labels (1712, 1) uint8
test_images (3421, 28, 28, 3) uint8
test_labels (3421, 1) uint8
```

```
print(dataset['train_images'].shape)
```

```
(11959, 28, 28, 3)
```

```
from keras.utils import to_categorical
import warnings

warnings.filterwarnings('ignore')

train_images = dataset['train_images'].astype('float32')/255.0
train_labels = to_categorical(dataset['train_labels'],num_classes=
len(np.unique(dataset['train_labels'])))
val_images = dataset['val_images'].astype('float32')/255.0
val_labels = to_categorical(dataset['val_labels'],num_classes=
len(np.unique(dataset['val_labels'])))
test_images = dataset['test_images'].astype('float32')/255.0
test_labels = to_categorical(dataset['test_labels'],num_classes=
len(np.unique(dataset['val_labels'])))
```

```
train_labels.shape
```

```
(11959, 8)
```

2.1 What models/architectures have you chosen to implement [5 marks]

In the following block, write a short (max 200 words) description and justification of the architectures that you have chosen to implement. You should also think about any optimisers and error or loss functions that you will be using and why they might be suitable.

Write your answer here.

Justification for choice of Architectures of training models and use of optimizer and loss function

In the pursuit of training the provided image dataset, four key architectures come into play: Convolutional Neural Networks (CNN), Feed-forward Neural Networks (FNN), K-Nearest Neighbors (KNN), and Logistic Regression.

- CNNs, tailored for image data, automatically learn spatial hierarchies and patterns, eliminating the need for extensive manual feature engineering.
- FNNs, renowned for their versatility, extend their utility across various data types, including images. Their interpretability provides a nuanced grasp of how input features contribute to output, uncovering less apparent features.
- KNN, characterized by simplicity, proves intuitive for image data with minimal parameter tuning. Its non-linear nature excels in capturing intricate patterns.

- Logistic Regression, valued for simplicity, yields clear and interpretable outcomes while showing resilience against overfitting, particularly in high-dimensional datasets.

For both CNN and FNN, the adaptive Adam optimizer aids in efficient learning rate adjustments, with Categorical Crossentropy as a fitting loss function for multi-class image classification.

2.2 Implementation and training of your models. [10 marks]

You should now implement the models that you have introduced above, train them and optimise any hyper-parameters using the validation set. You may wish to store any training results for the next sub-question.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import EarlyStopping,History,Callback
import warnings

warnings.filterwarnings("ignore")

class Test_Accuracy_Log(Callback):
    def __init__(self, test_data):
        super().__init__()
        self.test_data = test_data
        self.test_loss = []
        self.test_accuracy = []

    def on_epoch_end(self, epoch, logs=None):
        # Evaluate the model on the test data and log the test
accuracy
        test_loss, test_accuracy =
self.model.evaluate(self.test_data[0], self.test_data[1], verbose=0)
        self.test_loss.append(test_loss)
        self.test_accuracy.append(test_accuracy)
        print(f'\nEpoch {epoch + 1}: Test Loss: {test_loss}, Test
Accuracy: {test_accuracy*100:0.2f}%')

# Define the model creation function
def build_model():
    model = models.Sequential()
    # Convolutional layers
    model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(28, 28, 3)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
```

```
    # Dense layers
    model.add(layers.Flatten())
    model.add(layers.Dense(32, activation='relu'))
    model.add(layers.Dense(8, activation='softmax'))  # Assuming you
have 8 classes

    optimizer = keras.optimizers.Adam(learning_rate=0.01)
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model


# Define hyperparameters for tuning
batch_size =  [32,64,128]
epochs =  [5,15,22]
acc1 = 0
best_params = {}

for i in batch_size:
    for j in epochs:
        model = build_model()
        model.fit(train_images,train_labels, epochs=j,
batch_size=i,verbose=0)
        loss, acc = model.evaluate(val_images, val_labels,verbose=0)
        if acc>acc1:
            best_params['epochs'] = j
            best_params['batch_size'] = i

# Print the best parameters and accuracy
history = History()
modelcnn = build_model()
test_accuracy_logger = Test_Accuracy_Log(test_data=(test_images,
test_labels))
modelcnn.fit(train_images,train_labels,validation_data=(val_images,
val_labels),
                      epochs = best_params['epochs'],batch_size =
best_params['batch_size'],
                      callbacks=[history,test_accuracy_logger])
training_accuracy = history.history['accuracy']
validation_accuracy = history.history['val_accuracy']
training_loss = history.history['loss']
validation_loss = history.history['val_loss']
test_loss = test_accuracy_logger.test_loss
test_accuracy = test_accuracy_logger.test_accuracy
print('Best parameters for CNN are:',best_params)

Epoch 1/22
92/94 [============================>.] - ETA: 0s - loss: 1.4069 -
```

```
accuracy: 0.4681
Epoch 1: Test Loss: 0.9360322952270508, Test Accuracy: 63.26%
94/94 [==============================] - 7s 47ms/step - loss: 1.3991 -
accuracy: 0.4704 - val_loss: 0.9123 - val_accuracy: 0.6472
Epoch 2/22
93/94 [============================>.] - ETA: 0s - loss: 0.8649 -
accuracy: 0.6734
Epoch 2: Test Loss: 0.757265567779541, Test Accuracy: 72.29%
94/94 [==============================] - 4s 38ms/step - loss: 0.8648 -
accuracy: 0.6730 - val_loss: 0.7341 - val_accuracy: 0.7243
Epoch 3/22
92/94 [============================>.] - ETA: 0s - loss: 0.7271 -
accuracy: 0.7318
Epoch 3: Test Loss: 0.6784824728965759, Test Accuracy: 75.56%
94/94 [==============================] - 4s 38ms/step - loss: 0.7259 -
accuracy: 0.7323 - val_loss: 0.6665 - val_accuracy: 0.7518
Epoch 4/22
92/94 [============================>.] - ETA: 0s - loss: 0.6533 -
accuracy: 0.7570
Epoch 4: Test Loss: 0.6831179857254028, Test Accuracy: 75.45%
94/94 [==============================] - 4s 38ms/step - loss: 0.6532 -
accuracy: 0.7575 - val_loss: 0.6540 - val_accuracy: 0.7558
Epoch 5/22
92/94 [============================>.] - ETA: 0s - loss: 0.6170 -
accuracy: 0.7734
Epoch 5: Test Loss: 0.6244463324546814, Test Accuracy: 77.84%
94/94 [==============================] - 4s 38ms/step - loss: 0.6166 -
accuracy: 0.7736 - val_loss: 0.6041 - val_accuracy: 0.7722
Epoch 6/22
93/94 [============================>.] - ETA: 0s - loss: 0.5813 -
accuracy: 0.7858
Epoch 6: Test Loss: 0.5537170171737671, Test Accuracy: 80.33%
94/94 [==============================] - 4s 40ms/step - loss: 0.5811 -
accuracy: 0.7859 - val_loss: 0.5388 - val_accuracy: 0.7950
Epoch 7/22
92/94 [============================>.] - ETA: 0s - loss: 0.5616 -
accuracy: 0.7941
Epoch 7: Test Loss: 0.5579591393470764, Test Accuracy: 79.63%
94/94 [==============================] - 4s 43ms/step - loss: 0.5603 -
accuracy: 0.7945 - val_loss: 0.5422 - val_accuracy: 0.7973
Epoch 8/22
93/94 [============================>.] - ETA: 0s - loss: 0.5049 -
accuracy: 0.8100
Epoch 8: Test Loss: 0.4813830852508545, Test Accuracy: 82.43%
94/94 [==============================] - 4s 37ms/step - loss: 0.5049 -
accuracy: 0.8099 - val_loss: 0.4764 - val_accuracy: 0.8218
Epoch 9/22
92/94 [============================>.] - ETA: 0s - loss: 0.4965 -
accuracy: 0.8186
```

```
Epoch 9: Test Loss: 0.5805486440658569, Test Accuracy: 79.04%
94/94 [==============================] - 4s 40ms/step - loss: 0.4958 -
accuracy: 0.8186 - val_loss: 0.5647 - val_accuracy: 0.7862
Epoch 10/22
92/94 [============================>.] - ETA: 0s - loss: 0.4581 -
accuracy: 0.8328
Epoch 10: Test Loss: 0.46898186206817627, Test Accuracy: 83.08%
94/94 [==============================] - 4s 37ms/step - loss: 0.4577 -
accuracy: 0.8332 - val_loss: 0.4582 - val_accuracy: 0.8353
Epoch 11/22
92/94 [============================>.] - ETA: 0s - loss: 0.4218 -
accuracy: 0.8421
Epoch 11: Test Loss: 0.532650887966156, Test Accuracy: 79.36%
94/94 [==============================] - 3s 37ms/step - loss: 0.4193 -
accuracy: 0.8434 - val_loss: 0.5143 - val_accuracy: 0.7961
Epoch 12/22
92/94 [============================>.] - ETA: 0s - loss: 0.4047 -
accuracy: 0.8493
Epoch 12: Test Loss: 0.4672936201095581, Test Accuracy: 82.99%
94/94 [==============================] - 3s 37ms/step - loss: 0.4039 -
accuracy: 0.8493 - val_loss: 0.4622 - val_accuracy: 0.8236
Epoch 13/22
92/94 [============================>.] - ETA: 0s - loss: 0.3728 -
accuracy: 0.8608
Epoch 13: Test Loss: 0.38925111293792725, Test Accuracy: 86.23%
94/94 [==============================] - 3s 37ms/step - loss: 0.3726 -
accuracy: 0.8610 - val_loss: 0.3790 - val_accuracy: 0.8686
Epoch 14/22
93/94 [============================>.] - ETA: 0s - loss: 0.3682 -
accuracy: 0.8637
Epoch 14: Test Loss: 0.4238761067390442, Test Accuracy: 84.45%
94/94 [==============================] - 3s 37ms/step - loss: 0.3673 -
accuracy: 0.8640 - val_loss: 0.4224 - val_accuracy: 0.8382
Epoch 15/22
94/94 [==============================] - ETA: 0s - loss: 0.3601 -
accuracy: 0.8645
Epoch 15: Test Loss: 0.37700778245925903, Test Accuracy: 85.68%
94/94 [==============================] - 4s 38ms/step - loss: 0.3601 -
accuracy: 0.8645 - val_loss: 0.3837 - val_accuracy: 0.8639
Epoch 16/22
93/94 [============================>.] - ETA: 0s - loss: 0.3700 -
accuracy: 0.8643
Epoch 16: Test Loss: 0.39756014943122864, Test Accuracy: 86.23%
94/94 [==============================] - 3s 37ms/step - loss: 0.3701 -
accuracy: 0.8643 - val_loss: 0.4138 - val_accuracy: 0.8540
Epoch 17/22
93/94 [============================>.] - ETA: 0s - loss: 0.3320 -
accuracy: 0.8763
Epoch 17: Test Loss: 0.36501702666282654, Test Accuracy: 86.99%
```

```
94/94 [==============================] - 4s 41ms/step - loss: 0.3323 -
accuracy: 0.8762 - val_loss: 0.3537 - val_accuracy: 0.8750
Epoch 18/22
92/94 [============================>.] - ETA: 0s - loss: 0.3373 -
accuracy: 0.8798
Epoch 18: Test Loss: 0.33832159638404846, Test Accuracy: 87.52%
94/94 [==============================] - 4s 44ms/step - loss: 0.3363 -
accuracy: 0.8797 - val_loss: 0.3280 - val_accuracy: 0.8849
Epoch 19/22
93/94 [============================>.] - ETA: 0s - loss: 0.3025 -
accuracy: 0.8866
Epoch 19: Test Loss: 0.43049153685569763, Test Accuracy: 84.07%
94/94 [==============================] - 5s 51ms/step - loss: 0.3028 -
accuracy: 0.8866 - val_loss: 0.4264 - val_accuracy: 0.8394
Epoch 20/22
94/94 [==============================] - ETA: 0s - loss: 0.3146 -
accuracy: 0.8806
Epoch 20: Test Loss: 0.34791359305381775, Test Accuracy: 87.43%
94/94 [==============================] - 5s 49ms/step - loss: 0.3146 -
accuracy: 0.8806 - val_loss: 0.3551 - val_accuracy: 0.8791
Epoch 21/22
92/94 [============================>.] - ETA: 0s - loss: 0.2979 -
accuracy: 0.8894
Epoch 21: Test Loss: 0.3565422296524048, Test Accuracy: 87.49%
94/94 [==============================] - 5s 48ms/step - loss: 0.2981 -
accuracy: 0.8895 - val_loss: 0.3507 - val_accuracy: 0.8727
Epoch 22/22
93/94 [============================>.] - ETA: 0s - loss: 0.2848 -
accuracy: 0.8947
Epoch 22: Test Loss: 0.3426642417907715, Test Accuracy: 88.16%
94/94 [==============================] - 5s 48ms/step - loss: 0.2847 -
accuracy: 0.8948 - val_loss: 0.3635 - val_accuracy: 0.8785
Best parameters for CNN are: {'epochs': 22, 'batch_size': 128}
```

```python
# Define the model creation function
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import EarlyStopping,History,Callback
import warnings

warnings.filterwarnings("ignore")

class Test_Accuracy_Log(Callback):
    def __init__(self, test_data):
        super().__init__()
        self.test_data = test_data
        self.test_loss = []
        self.test_accuracy = []
```

```
    def on_epoch_end(self, epoch, logs=None):
        # Evaluate the model on the test data and log the test
accuracy
        test_loss, test_accuracy =
self.model.evaluate(self.test_data[0], self.test_data[1], verbose=0)
        self.test_loss.append(test_loss)
        self.test_accuracy.append(test_accuracy)
        print(f'\nEpoch {epoch + 1}: Test Loss: {test_loss}, Test
Accuracy: {test_accuracy*100:0.2f}%')
def build_fnn_model():
    model = models.Sequential()
    model.add(layers.Flatten(input_shape=(28, 28, 3)))
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(8, activation='softmax'))  # Assuming you
have 8 classes

    optimizer = keras.optimizers.Adam(learning_rate=0.01)
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model

# Define hyperparameters for tuning
batch_size =  [32,64,128]
epochs =  [5,15,22]
acc1 = 0
best_params = {}

for i in batch_size:
    for j in epochs:
        model = build_fnn_model()
        model.fit(train_images,train_labels, epochs=j,
batch_size=i,verbose=0)
        loss, acc = model.evaluate(val_images, val_labels,verbose=0)
        if acc>acc1:
            best_params['epochs'] = j
            best_params['batch_size'] = i

history1 = History()
modelfnn = build_fnn_model()
test_accuracy_logger = Test_Accuracy_Log(test_data=(test_images,
test_labels))
modelfnn.fit(train_images,train_labels,validation_data=(val_images,
val_labels),
                       epochs = best_params['epochs'],batch_size =
best_params['batch_size'],
                       callbacks=[history1,test_accuracy_logger])
training_accuracy1 = history1.history['accuracy']
```

```
validation_accuracy1 = history1.history['val_accuracy']
training_loss1 = history1.history['loss']
validation_loss1 = history1.history['val_loss']
test_loss1 = test_accuracy_logger.test_loss
test_accuracy1 = test_accuracy_logger.test_accuracy
print('Best parameters for FNN are:',best_params)
```

```
Epoch 1/22
93/94 [============================>.] - ETA: 0s - loss: 2.7131 -
accuracy: 0.3788
Epoch 1: Test Loss: 1.2034344673156738, Test Accuracy: 57.15%
94/94 [==============================] - 2s 18ms/step - loss: 2.7066 -
accuracy: 0.3793 - val_loss: 1.2033 - val_accuracy: 0.5759
Epoch 2/22
91/94 [============================>.] - ETA: 0s - loss: 1.0577 -
accuracy: 0.6264
Epoch 2: Test Loss: 0.9131712317466736, Test Accuracy: 66.94%
94/94 [==============================] - 1s 14ms/step - loss: 1.0547 -
accuracy: 0.6261 - val_loss: 0.8974 - val_accuracy: 0.6676
Epoch 3/22
90/94 [============================>..] - ETA: 0s - loss: 0.9057 -
accuracy: 0.6674
Epoch 3: Test Loss: 0.947585940361023, Test Accuracy: 64.34%
94/94 [==============================] - 1s 13ms/step - loss: 0.9086 -
accuracy: 0.6661 - val_loss: 0.9123 - val_accuracy: 0.6472
Epoch 4/22
89/94 [============================>..] - ETA: 0s - loss: 0.8394 -
accuracy: 0.6954
Epoch 4: Test Loss: 0.8555501103401184, Test Accuracy: 67.76%
94/94 [==============================] - 1s 14ms/step - loss: 0.8389 -
accuracy: 0.6952 - val_loss: 0.8373 - val_accuracy: 0.6723
Epoch 5/22
94/94 [==============================] - ETA: 0s - loss: 0.7977 -
accuracy: 0.7037
Epoch 5: Test Loss: 0.7300787568092346, Test Accuracy: 72.20%
94/94 [==============================] - 1s 13ms/step - loss: 0.7977 -
accuracy: 0.7037 - val_loss: 0.6992 - val_accuracy: 0.7354
Epoch 6/22
94/94 [==============================] - ETA: 0s - loss: 0.7346 -
accuracy: 0.7271
Epoch 6: Test Loss: 0.6972822546958923, Test Accuracy: 73.78%
94/94 [==============================] - 1s 15ms/step - loss: 0.7346 -
accuracy: 0.7271 - val_loss: 0.6787 - val_accuracy: 0.7442
Epoch 7/22
88/94 [============================>..] - ETA: 0s - loss: 0.7180 -
accuracy: 0.7343
Epoch 7: Test Loss: 0.6465342044830322, Test Accuracy: 76.97%
94/94 [==============================] - 1s 15ms/step - loss: 0.7164 -
accuracy: 0.7336 - val_loss: 0.6270 - val_accuracy: 0.7722
Epoch 8/22
```

```
90/94 [==========================>..] - ETA: 0s - loss: 0.6927 -
accuracy: 0.7423
Epoch 8: Test Loss: 0.7185612320899963, Test Accuracy: 73.37%
94/94 [==============================] - 1s 14ms/step - loss: 0.6957 -
accuracy: 0.7405 - val_loss: 0.6997 - val_accuracy: 0.7459
Epoch 9/22
94/94 [==============================] - ETA: 0s - loss: 0.7068 -
accuracy: 0.7369
Epoch 9: Test Loss: 0.7361541390419006, Test Accuracy: 70.56%
94/94 [==============================] - 1s 13ms/step - loss: 0.7068 -
accuracy: 0.7369 - val_loss: 0.6974 - val_accuracy: 0.7202
Epoch 10/22
90/94 [==========================>..] - ETA: 0s - loss: 0.6361 -
accuracy: 0.7652
Epoch 10: Test Loss: 0.5822147727012634, Test Accuracy: 78.16%
94/94 [==============================] - 1s 14ms/step - loss: 0.6360 -
accuracy: 0.7649 - val_loss: 0.5650 - val_accuracy: 0.7775
Epoch 11/22
90/94 [==========================>..] - ETA: 0s - loss: 0.6010 -
accuracy: 0.7786
Epoch 11: Test Loss: 0.5966836214065552, Test Accuracy: 77.76%
94/94 [==============================] - 1s 14ms/step - loss: 0.6039 -
accuracy: 0.7782 - val_loss: 0.5840 - val_accuracy: 0.7833
Epoch 12/22
91/94 [==========================>.] - ETA: 0s - loss: 0.6156 -
accuracy: 0.7705
Epoch 12: Test Loss: 0.7419215440750122, Test Accuracy: 71.85%
94/94 [==============================] - 1s 15ms/step - loss: 0.6163 -
accuracy: 0.7702 - val_loss: 0.7297 - val_accuracy: 0.7220
Epoch 13/22
94/94 [==============================] - ETA: 0s - loss: 0.5805 -
accuracy: 0.7894
Epoch 13: Test Loss: 0.5527311563491821, Test Accuracy: 79.57%
94/94 [==============================] - 1s 16ms/step - loss: 0.5805 -
accuracy: 0.7894 - val_loss: 0.5427 - val_accuracy: 0.7944
Epoch 14/22
89/94 [==========================>..] - ETA: 0s - loss: 0.6297 -
accuracy: 0.7649
Epoch 14: Test Loss: 0.6374648213386536, Test Accuracy: 76.38%
94/94 [==============================] - 1s 15ms/step - loss: 0.6305 -
accuracy: 0.7654 - val_loss: 0.6307 - val_accuracy: 0.7617
Epoch 15/22
88/94 [==========================>..] - ETA: 0s - loss: 0.5699 -
accuracy: 0.7945
Epoch 15: Test Loss: 0.6068854331970215, Test Accuracy: 77.43%
94/94 [==============================] - 1s 14ms/step - loss: 0.5703 -
accuracy: 0.7945 - val_loss: 0.5819 - val_accuracy: 0.7897
Epoch 16/22
91/94 [==========================>.] - ETA: 0s - loss: 0.5669 -
```

```
accuracy: 0.7907
Epoch 16: Test Loss: 0.5575553178787231, Test Accuracy: 78.87%
94/94 [==============================] - 1s 15ms/step - loss: 0.5681 -
accuracy: 0.7895 - val_loss: 0.5325 - val_accuracy: 0.8143
Epoch 17/22
88/94 [===========================>..] - ETA: 0s - loss: 0.5544 -
accuracy: 0.7937
Epoch 17: Test Loss: 0.5762784481048584, Test Accuracy: 79.98%
94/94 [==============================] - 1s 15ms/step - loss: 0.5499 -
accuracy: 0.7948 - val_loss: 0.5468 - val_accuracy: 0.8067
Epoch 18/22
91/94 [============================>.] - ETA: 0s - loss: 0.5392 -
accuracy: 0.8015
Epoch 18: Test Loss: 0.510707676410675, Test Accuracy: 80.94%
94/94 [==============================] - 1s 14ms/step - loss: 0.5391 -
accuracy: 0.8015 - val_loss: 0.4893 - val_accuracy: 0.8195
Epoch 19/22
92/94 [============================>.] - ETA: 0s - loss: 0.5869 -
accuracy: 0.7869
Epoch 19: Test Loss: 0.6094915866851807, Test Accuracy: 76.59%
94/94 [==============================] - 2s 17ms/step - loss: 0.5879 -
accuracy: 0.7862 - val_loss: 0.5764 - val_accuracy: 0.7880
Epoch 20/22
90/94 [===========================>..] - ETA: 0s - loss: 0.5531 -
accuracy: 0.7951
Epoch 20: Test Loss: 0.601076602935791, Test Accuracy: 78.72%
94/94 [==============================] - 2s 18ms/step - loss: 0.5547 -
accuracy: 0.7956 - val_loss: 0.5633 - val_accuracy: 0.7979
Epoch 21/22
91/94 [============================>.] - ETA: 0s - loss: 0.5753 -
accuracy: 0.7910
Epoch 21: Test Loss: 0.5748487710952759, Test Accuracy: 77.87%
94/94 [==============================] - 1s 15ms/step - loss: 0.5742 -
accuracy: 0.7914 - val_loss: 0.5509 - val_accuracy: 0.7856
Epoch 22/22
93/94 [============================>.] - ETA: 0s - loss: 0.5121 -
accuracy: 0.8122
Epoch 22: Test Loss: 0.5756575465202332, Test Accuracy: 78.40%
94/94 [==============================] - 2s 16ms/step - loss: 0.5118 -
accuracy: 0.8122 - val_loss: 0.5352 - val_accuracy: 0.7944
Best parameters for FNN are: {'epochs': 22, 'batch_size': 128}
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,mean_absolute_error
n_neighbors = range(1,50,2)
best_params = {}
train_images_flattened = train_images.reshape(train_images.shape[0], -
1)
train_labels1 = dataset['train_labels'].reshape(-1)
val_images_flattened = val_images.reshape(val_images.shape[0], -1)
```

```
val_labels1 = dataset['val_labels'].reshape(-1)
test_images_flattened = test_images.reshape(test_images.shape[0], -1)
test_labels1 = dataset['test_labels'].reshape(-1)
acc1 = 0
for i in n_neighbors:
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(train_images_flattened,train_labels1)
    pred = knn.predict(val_images_flattened)
    acc = accuracy_score(val_labels1,pred)
    if acc>acc1:
        best_params['n_neighbors'] = i
        acc1 = acc

knn1 = KNeighborsClassifier(n_neighbors=best_params['n_neighbors'])
knn1.fit(train_images_flattened,train_labels1)
pred2 = knn1.predict(train_images_flattened)
pred1 = knn1.predict(test_images_flattened)
test_acc_best = accuracy_score(test_labels1,pred1) * 100
train_acc_best = accuracy_score(train_labels1,pred2) * 100
test_err_best = mean_absolute_error(test_labels1,pred1)
train_err_best = mean_absolute_error(train_labels1,pred2)

# Print the best parameters and accuracy
print(f"Best for KNN Classifier:
{accuracy_score(test_labels1,pred1)*100 : 0.2f}% using {best_params}")
```

```
Best for KNN Classifier:  76.53% using {'n_neighbors': 23}
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,mean_absolute_error
import warnings

warnings.filterwarnings("ignore")

best_params = {}
C = [0.01, 0.1, 1, 10, 100]
max_iter = [200, 500, 1000]
train_images_flattened = train_images.reshape(train_images.shape[0], -
1)
train_labels1 = dataset['train_labels'].reshape(-1)
val_images_flattened = val_images.reshape(val_images.shape[0], -1)
val_labels1 = dataset['val_labels'].reshape(-1)
test_images_flattened = test_images.reshape(test_images.shape[0], -1)
test_labels1 = dataset['test_labels'].reshape(-1)
acc1 = 0
for i in C:
    for j in max_iter:
        logr = LogisticRegression(C=i,max_iter=j)
        logr.fit(train_images_flattened,train_labels1)
        pred = logr.predict(val_images_flattened)
```

```
        acc = accuracy_score(val_labels1,pred)
        if acc>acc1:
            best_params['C'] = i
            best_params['max_iter'] = j
            acc1 = acc

logr1 =
LogisticRegression(C=best_params['C'],max_iter=best_params['max_iter']
)
logr1.fit(train_images_flattened,train_labels1)
pred2 = logr1.predict(train_images_flattened)
pred1 = logr1.predict(test_images_flattened)
test_acc_best1 = accuracy_score(test_labels1,pred1) * 100
train_acc_best1 = accuracy_score(train_labels1,pred2) * 100
test_err_best1 = mean_absolute_error(test_labels1,pred1)
train_err_best1 = mean_absolute_error(train_labels1,pred2)

# Print the best parameters and accuracy
print(f"Best for Logistic Regression Classifier:
{accuracy_score(test_labels1,pred1)*100 : 0.2f}% using {best_params}")
```

```
Best for Logistic Regression Classifier:  82.58% using {'C': 0.1,
'max_iter': 1000}
```

In the following block, comment on the success of the training process and provide a description of how you have selected or optimised any hyper-parameters.

Write your answer here.

Analysis of acurracies for each modelling architecture and hyper-parameters optimisation

In the modeling process, we adjust hyperparameters based on validation accuracy and create a comprehensive approach to make the most of each architecture's strengths in handling the unique challenges of the image dataset.

The hyperparameters tuned for CNN and FNN included the batch size and the number of epochs, while for KNN, the optimization revolved around the number of nearest neighbors (K-value). In the case of logistic regression, the hyperparameters 'max_iter'(maximum number of iterations for convergence) and 'C' (regulation strength, aiding in avoiding overfitting) were subject to optimization.

Looking at the test accuracies for each model:

- *CNN*: Achieving a test accuracy of 88.16%, CNN shows it's good at understanding complex patterns in the image data, making it a strong contender for image classification.

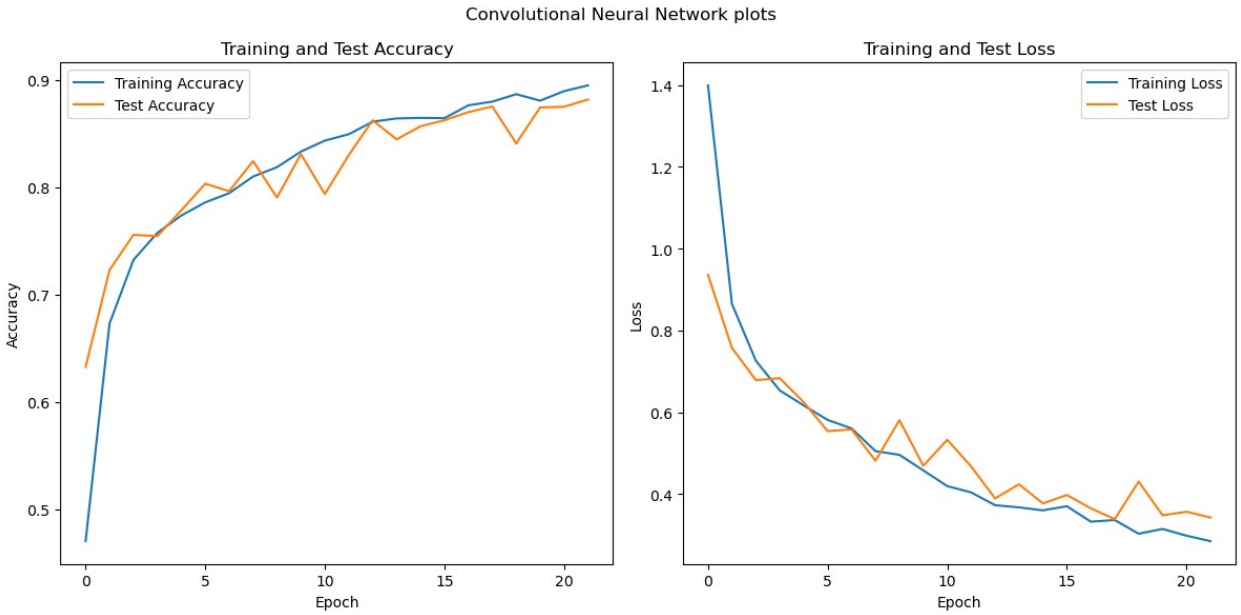
- *FNN*: While FNNs are versatile, their test accuracy of 78.4% indicates they might not be as effective as CNNs in grasping spatial relationships crucial for image tasks.
- *KNN*: With a test accuracy of 76.53%, KNN seems to struggle with the high-dimensional and complex patterns in image data.
- *Logistic Regression*: Achieving a test accuracy of 82.58%, logistic regression captures some patterns but might not handle intricate relationships as well as neural networks.

In conclusion, CNN proves to be the best architecture for image classification due to its ability to capture complex spatial relationships, automatic learning of hierarchical features, and a significantly higher accuracy of 88.16%. This suggests that CNN is well-suited for the demands of image classification tasks.

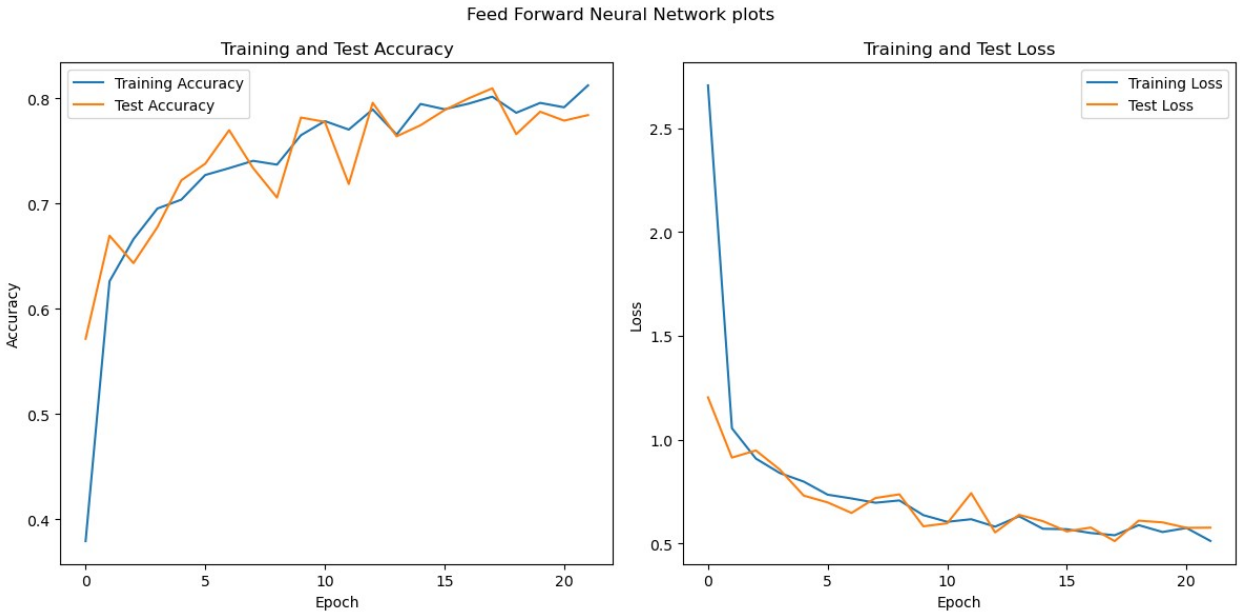
2.3 Classification results based on the test data [10 marks]

You should now plot the accuracy and error (or loss), on separate graphs, for the training and testing set. You may also undertake any other performance analysis of your models.

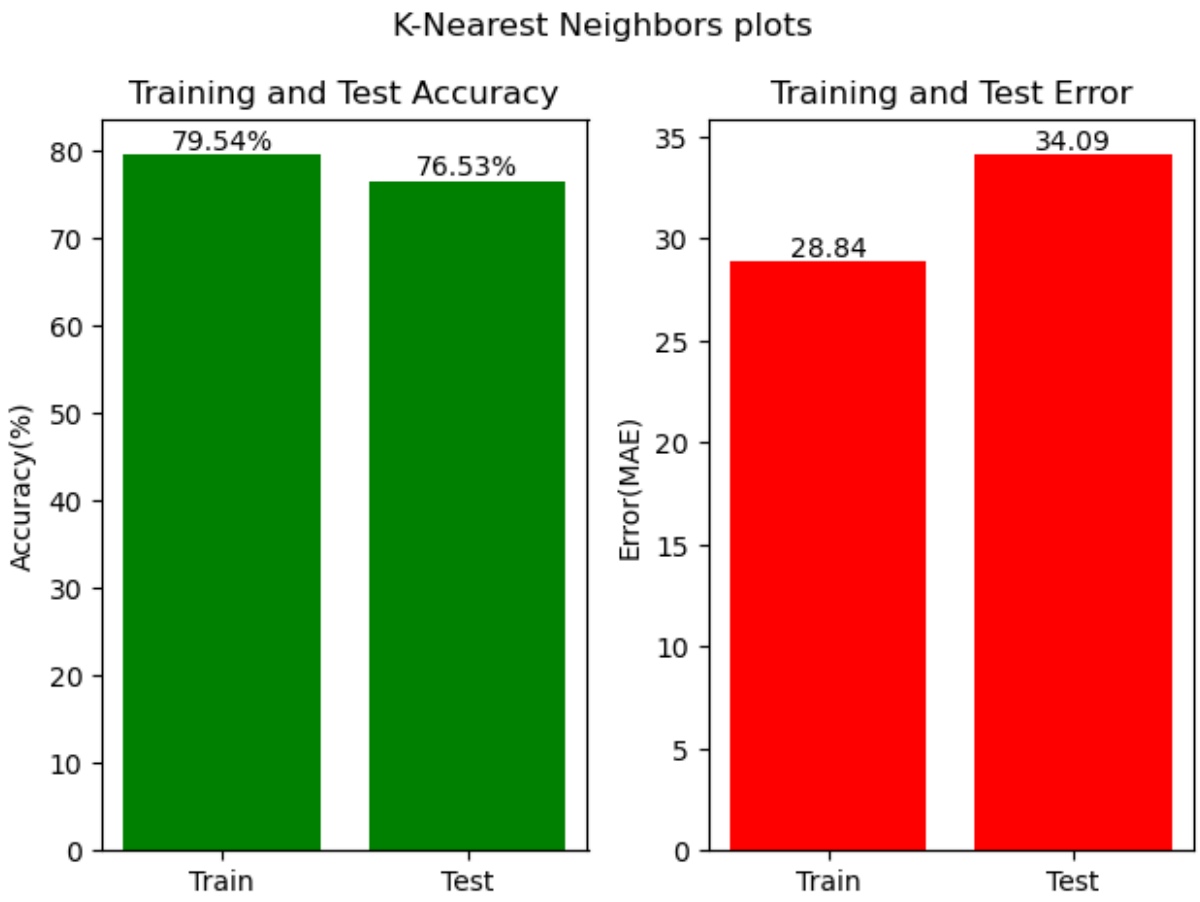
```
# Program your plots here.
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(training_accuracy, label='Training Accuracy')
plt.plot(test_accuracy, label='Test Accuracy')
plt.title('Training and Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot training and validation loss
plt.subplot(1, 2, 2)
plt.plot(training_loss, label='Training Loss')
plt.plot(test_loss, label='Test Loss')
plt.title('Training and Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.suptitle('Convolutional Neural Network plots')
plt.tight_layout()
plt.show()
```



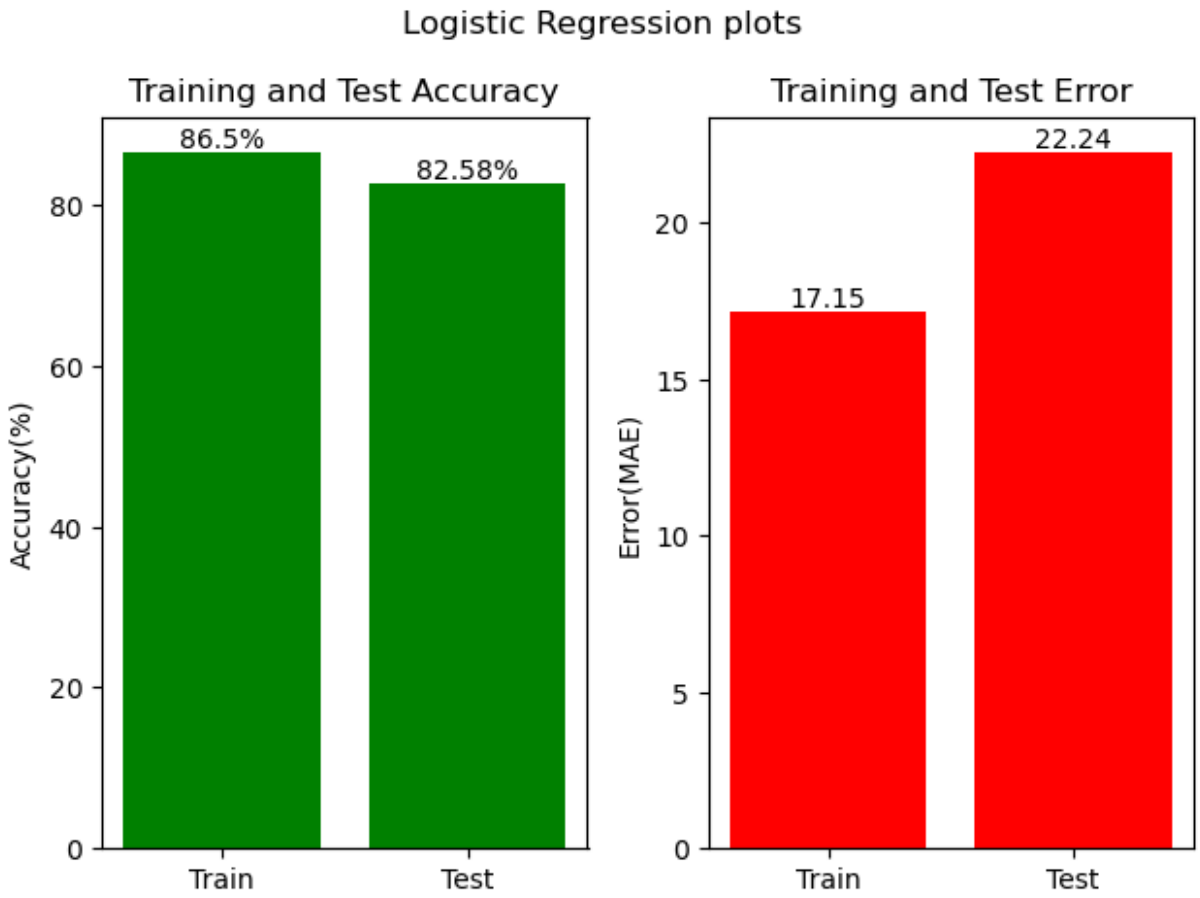

```
# Program your plots here.
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(training_accuracy1, label='Training Accuracy')
plt.plot(test_accuracy1, label='Test Accuracy')
plt.title('Training and Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot training and validation loss
plt.subplot(1, 2, 2)
plt.plot(training_loss1, label='Training Loss')
plt.plot(test_loss1, label='Test Loss')
plt.title('Training and Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.suptitle('Feed Forward Neural Network plots')
plt.tight_layout()
plt.show()
```




```
plt.subplot(1,2,1)
acc = [np.round(train_acc_best,2),np.round(test_acc_best,2)]
err = [np.round(train_err_best,2),np.round(test_err_best,2)]
data_type = ['Train','Test']
plt.bar(data_type,acc,color = 'green')
plt.text(data_type[0],acc[0],f'{acc[0]}%',ha = 'center',va = 'bottom')
plt.text(data_type[1],acc[1],f'{acc[1]}%',ha = 'center',va = 'bottom')
plt.title('Training and Test Accuracy')
plt.ylabel('Accuracy(%)')

plt.subplot(1,2,2)
plt.bar(data_type,err,color = 'red')
plt.text(data_type[0],err[0],err[0],ha = 'center',va = 'bottom')
plt.text(data_type[1],err[1],err[1],ha = 'center',va = 'bottom')
plt.title('Training and Test Error')
plt.ylabel('Error(MAE)') #MAE for Mean Absolute Error
plt.suptitle('K-Nearest Neighbors plots')
plt.tight_layout()
plt.show()
```




```
plt.subplot(1,2,1)
acc = [np.round(train_acc_best1,2),np.round(test_acc_best1,2)]
err = [np.round(train_err_best1,2),np.round(test_err_best1,2)]
data_type = ['Train','Test']
plt.bar(data_type,acc,color = 'green')
plt.text(data_type[0],acc[0],f'{acc[0]}%',ha = 'center',va = 'bottom')
plt.text(data_type[1],acc[1],f'{acc[1]}%',ha = 'center',va = 'bottom')
plt.title('Training and Test Accuracy')
plt.ylabel('Accuracy(%)')

plt.subplot(1,2,2)
plt.bar(data_type,err,color = 'red')
plt.text(data_type[0],err[0],err[0],ha = 'center',va = 'bottom')
plt.text(data_type[1],err[1],err[1],ha = 'center',va = 'bottom')
plt.title('Training and Test Error')
plt.ylabel('Error(MAE)') #MAE for Mean Absolute Error
plt.suptitle('Logistic Regression plots')
plt.tight_layout()
plt.show()
```



Now provide a short discussion evaluating your results and the architectures that you have used. Provide any conclusions that you can make from the data:

Write your answer here.

Visualisation of the results and performance of modeling architectures

- The CNN model performs impressively, achieving a high accuracy of 89.48% during training and maintaining competitiveness with 88.16% on new data at the end of epochs. This similarity suggests it's adept at grasping new image patterns effectively.
- The FNN model exhibits a training accuracy of 81.2% and a test accuracy of 78.4% at the end of epochs. The slight difference implies there's room for improvement, possibly through further adjustments or regularization techniques.
- KNN achieves a training accuracy of 79.54% and a test accuracy of 76.53%. These numbers are decent, but it faces challenges with complex image patterns compared to neural networks.
- Logistic Regression attains a training accuracy of 86.5% and a test accuracy of 82.58%. While it performs well, it falls slightly behind the CNN in terms of accuracy.