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* $\rightarrow$ *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):

In [ ]:
```
username = 'ACP23SD'
```

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.

In [ ]:
```
#We need to download a python file that contains some useful functions.

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

```
'wget' is not recognized as an internal or external command,
operable program or batch file.
```

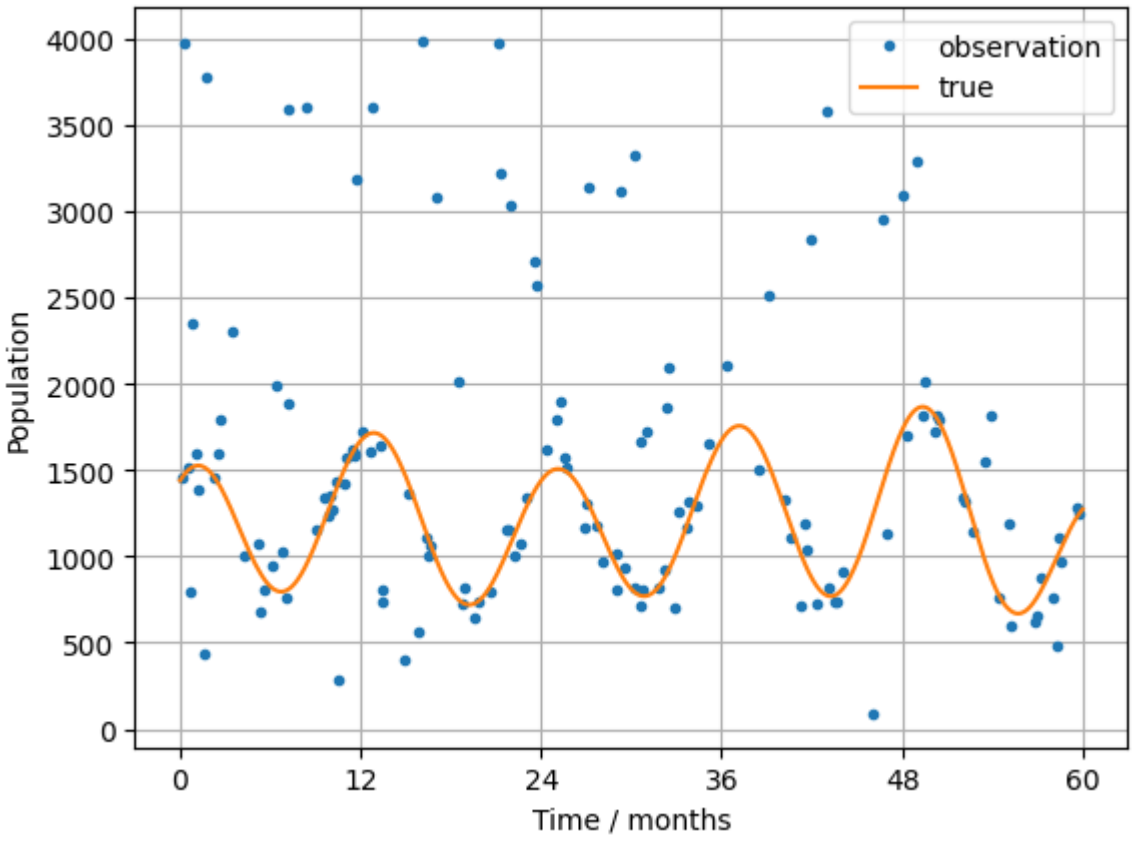
In [ ]:
```
#and import some modules

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.

In [ ]:
```
xdata,ydata,xtrue,ytrue = assignment.data()
```

In [ ]:
```
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.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]

In [ ]:
```
q1 = "Training -70-80%- Learn model patterns from labelled data, Validation -10-
assignment.wc(q1)
```

```
23 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, $\exp[-\frac{(x-c)^2}{2w^2}]$. 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).

In [ ]:
```
def gaussian(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
     - width, the 'spread' of the Gaussians in the basis
    """
    #To do: Implement
    Phi = np.zeros((x.shape[0], num_basis))
    for i in range(num_basis):
        Phi[:,i] = np.exp(-0.5*(((x-np.linspace(data_limits[0],data_limits[1],nu
    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:

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

In [ ]:
```
def grad_ridge(Phi,y,w,lam):
    """
    Return an D dimensional vector of gradients of w, assuming we want to minimi
    using the design matrix in Phi; under ridge regression with regularisation p
    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.
    """
    return 2*Phi.T@Phi@w - 2*Phi.T@y + 2*lam*w #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).

In [ ]:
```
def grad_descent(grad_fn,Phi,y,lam):
    """
    Compute optimised w.
    Parameters:
```

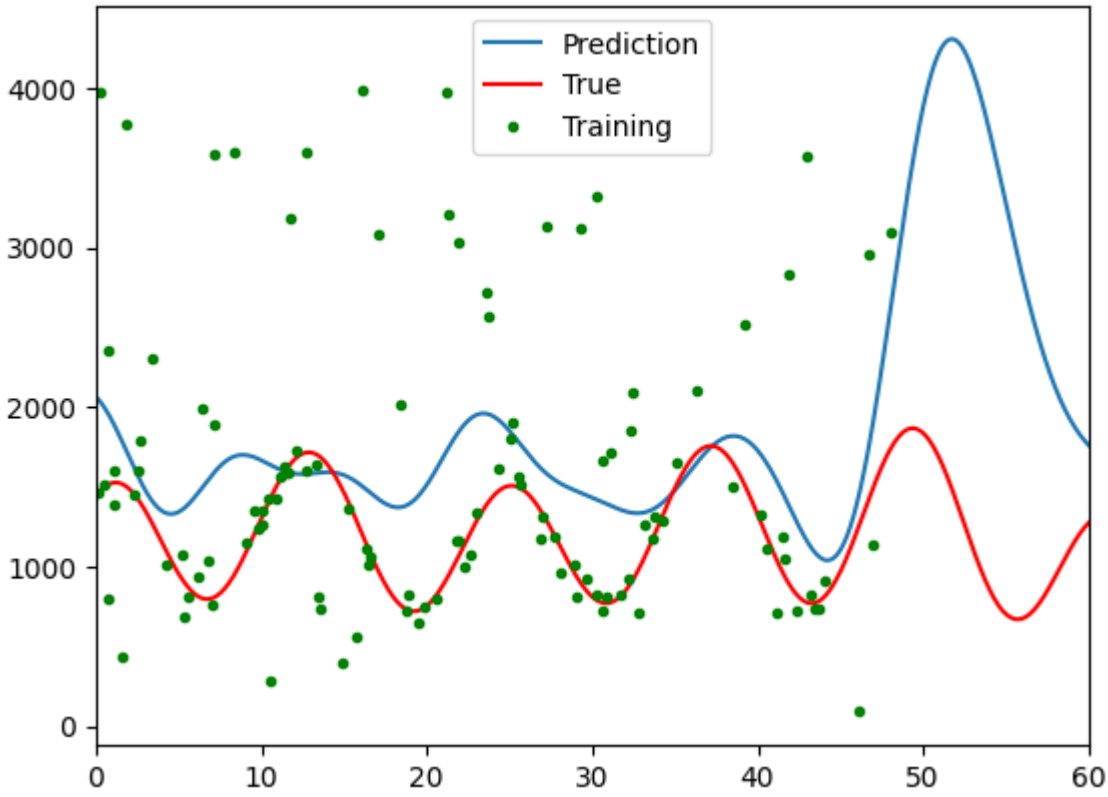
```
        - grad, the gradient function
        - Phi, design matrix (shape N x D)
        - y, vector of observations (length N)
        - lam, regularisation parameter, lambda.
    Returns
        - w_optimsed, a vector (length D) that minimises the ridge regression cos
    """
    w = np.zeros(Phi.shape[1])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

In [ ]:
```
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.legend()
plt.xlim([0,60])
```

Out[ ]:
```
(0.0, 60.0)
```



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:

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

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

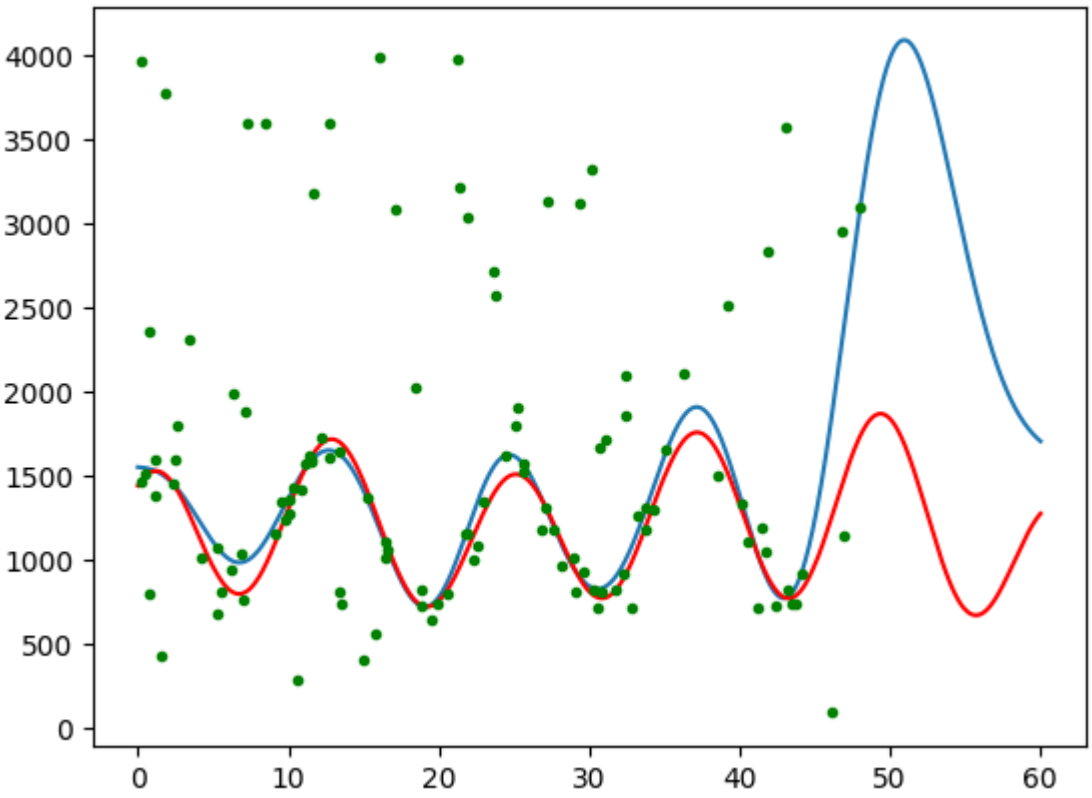
In [ ]:
```
def grad_abs(Phi,y,w,lam):
    """
    Return an D dimensional vector of gradients of w, assuming we want to minimi
    using the design matrix in Phi; under L2 regularisation parameter lambda.
    Arguments:
     - Phi, N x D design matrix
     - y, training outputs
     - w, parameters (we are finding the gradient at this value of w)
     - lam, the lambda regularisation parameter.
    """
    return Phi.T@np.sign(Phi@w - y) + 2*lam*w #To do: Implement

assignment.checkQ4(grad_abs)
```

```
Success
```

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

In [ ]:
```
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')
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x273fc057510>]
```



Question 5 [3 marks]

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

In [ ]:
```
q5 = "Absolute sum error is more resilient to outliers than sum squared error. T
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).

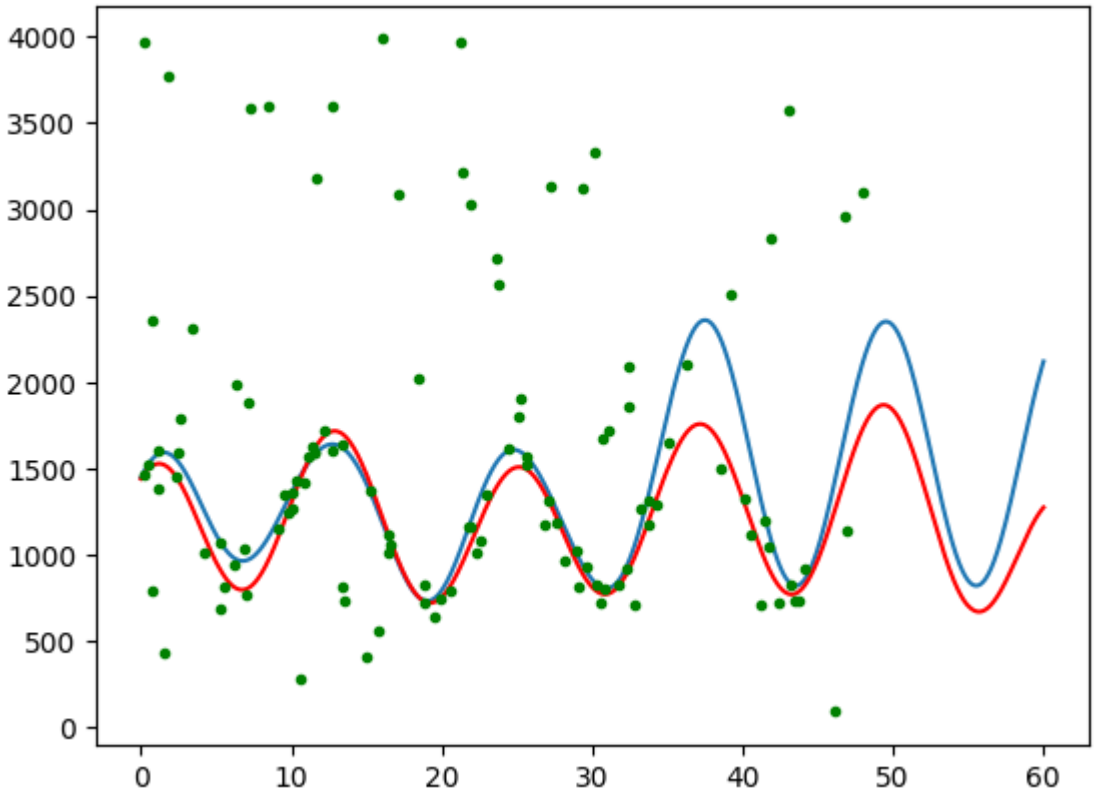
In [ ]:
```
def gaussian_and_sinusoidal(x, num_basis=4, data_limits=[-1., 1.], width = 10):
    """
```

```
    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
     - width, the 'spread' of the Gaussians in the basis

    Half the bases are Gaussian, half are evenly spaced cosines of 12 month peri
    """
    #To do: Implement
    Phi = np.zeros((x.shape[0], num_basis))
    period = 12
    for i in range(int(num_basis/2)):
        Phi[:,i] = np.exp(-0.5*(((x-np.linspace(data_limits[0],data_limits[1],nu
        Phi[:,int(num_basis/2+i)] = np.cos(2*np.pi/period*x-np.linspace(0,6,int(
    return Phi
```

Let's see how this has affected the result:

In [ ]:
```
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')
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x273fd0b48d0>]
```



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.

In [ ]:
```
class popReg():
    def __init__(self,num_basis=120,lam=0.01,width=3):
        self.num_basis = num_basis
        self.lam = lam
        self.width = width

    def fit(self,x,y):
        Phi = gaussian_and_sinusoidal(x, num_basis=self.num_basis, width=self.wi
        y_standardized = (y - np.mean(y)) / np.std(y)
        self.w = grad_descent(grad_abs, Phi, y_standardized, self.lam)
        self.ymean = np.mean(y)
        self.ystd = np.std(y)
        return self

    def predict(self,x):
        Phi = gaussian_and_sinusoidal(x, num_basis=self.num_basis, width=self.wi
        return (Phi @ self.w)*self.ystd + self.ymean

    def get_params(self, deep=True):
        return {"num_basis": self.num_basis, "lam": self.lam, "width": self.widt

    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return self
```

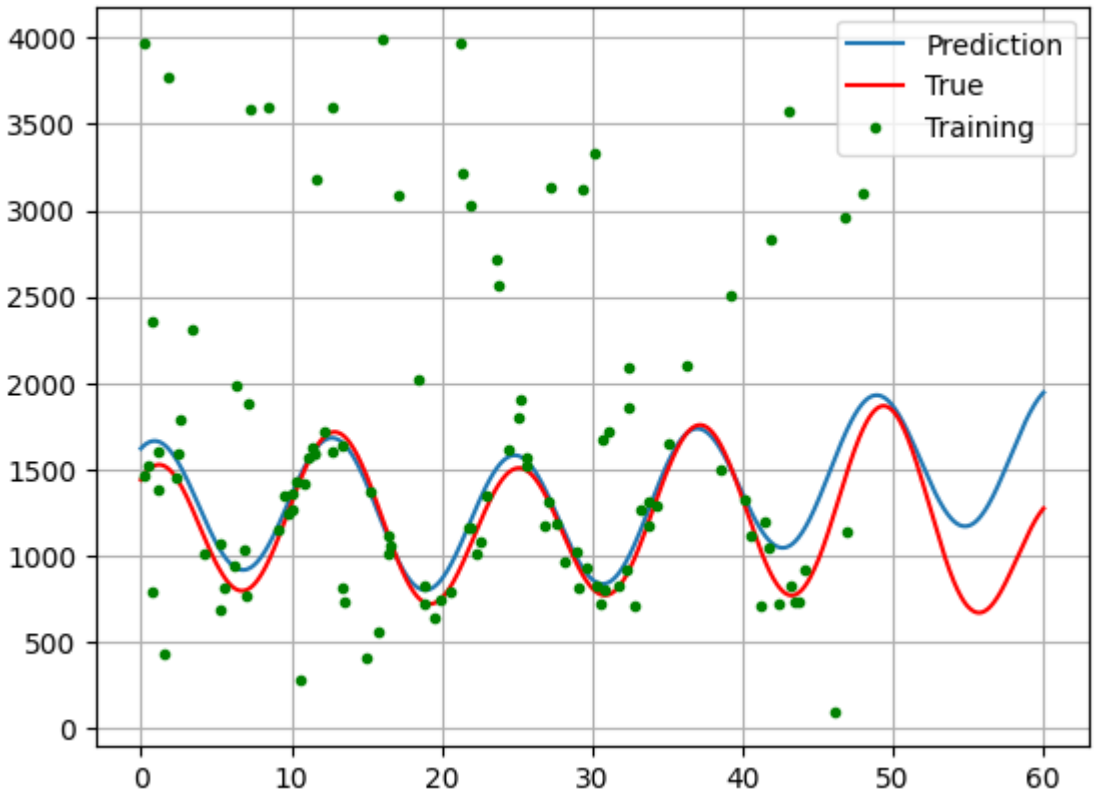
In [ ]:
```
#code here
xtrain = xdata[xdata<48]
ytrain = ydata[xdata<48]
xval = xtrue[xtrue>=48]
yval = ytrue[xtrue>=48]
```

In [ ]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {'num_basis':[30,60,120],
              'lam':[0.001,0.01,0.1],
              'width':[0.1,1,10]
              }
```

```
gcv = GridSearchCV(popReg(), param_grid, cv=5, scoring='neg_mean_absolute_error'
gcv.fit(xval,yval)

print(gcv.best_params_)
```

```
{'lam': 0.001, 'num_basis': 120, 'width': 10}
```

In [ ]:
```
data_mean = np.mean(ytrain)
data_std = np.std(ytrain)
ytrain_standardised = (ytrain - data_mean)/data_std

num_basis = gcv.best_params_['num_basis']
lam = gcv.best_params_['lam']
width = gcv.best_params_['width']

Phi = gaussian_and_sinusoidal(xtrain,num_basis,[0,60],width)
truePhi = gaussian_and_sinusoidal(xtrue,num_basis,[0,60],width)

w = grad_descent(grad_abs,Phi,ytrain_standardised,lam)
pred = truePhi @ w*data_std+data_mean

plt.plot(xtrue,pred,label='Prediction')
plt.plot(xtrue,ytrue,'-r',label='True')
plt.plot(xtrain,ytrain,'.g',label='Training')
plt.legend()
plt.grid()
```



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.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
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
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.

In [ ]:
```
x_train[:,0:5]
```

Out[ ]:
```
array([[ 0.28858451, -0.02029417, -0.13290514, -0.9952786 , -0.98311061],
       [ 0.27841883, -0.01641057, -0.12352019, -0.99824528, -0.97530022],
       [ 0.27965306, -0.01946716, -0.11346169, -0.99537956, -0.96718701],
       ...,
       [ 0.27338737, -0.01701062, -0.04502183, -0.21821818, -0.10382198],
       [ 0.28965416, -0.01884304, -0.15828059, -0.21913944, -0.11141169],
       [ 0.35150347, -0.01242312, -0.20386717, -0.26927044, -0.08721154]])
```

In [ ]:
```
np.isnan(x_train).all()
```

Out[ ]:
```
False
```

In [ ]:
```
x_train.max(),x_train.min()
```

Out[ ]:
```
(1.0, -1.0)
```

In [ ]:
```
from sklearn.cluster import *
from sklearn.metrics import silhouette_score
from sklearn.model_selection import GridSearchCV
import numpy as np
import time
```

In [ ]:
```
def silhouette_scorer(estimator, X):
    labels = estimator.fit_predict(X)
    return silhouette_score(X, labels)
```

Finding Compute time for algorithms

In [ ]:
```
compute_time = {}

algorithms = [KMeans(n_init=10),AffinityPropagation(),AgglomerativeClustering(),
#MeanShift(),SpectralClustering()

for algorithm in algorithms:
    print(f'Running {algorithm.__class__.__name__}')
    start_time = time.time()
    cluster = algorithm.fit(x_train)
    compute_time[algorithm.__class__.__name__] = np.round(time.time() - start_ti
```

```
Running KMeans
Running AffinityPropagation
Running AgglomerativeClustering
Running HDBSCAN
Running OPTICS
Running Birch
Running DBSCAN
Running MeanShift
Running MiniBatchKMeans
```

Implementing gridsearch for top 4 algorithms

In [ ]:
```
compute_time
```

Out[ ]:
```
{'KMeans': 2.05,
 'AffinityPropagation': 39.68,
 'AgglomerativeClustering': 10.8,
 'HDBSCAN': 54.48,
 'OPTICS': 113.09,
 'Birch': 13.04,
 'DBSCAN': 0.82,
 'MeanShift': 349.87,
 'MiniBatchKMeans': 0.35}
```

In [ ]:
```
# Sort the dictionary by value

sorted(compute_time.items(), key=lambda x: x[1])
```

Out[ ]:
```
[('MiniBatchKMeans', 0.35),
 ('DBSCAN', 0.82),
 ('KMeans', 2.05),
 ('AgglomerativeClustering', 10.8),
 ('Birch', 13.04),
 ('AffinityPropagation', 39.68),
 ('HDBSCAN', 54.48),
 ('OPTICS', 113.09),
 ('MeanShift', 349.87)]
```

In [ ]:
```
### Creating parameters for the top 3 models
```

```
param_grid = {}
param_grid['KMeans'] = {'n_clusters': range(2,8)}
param_grid['AgglomerativeClustering'] = {'n_clusters': range(2,8)}
param_grid['Birch'] = {'threshold': np.linspace(0.3, 0.7, 3),'n_clusters': range
param_grid['MiniBatchKMeans'] = {'n_clusters': range(2,8)}
```

In [ ]:
```
algorithm_score = {}
best_params = {}

fast_algorithms = [KMeans(n_init=10),AgglomerativeClustering(),MiniBatchKMeans(n

for algorithm in fast_algorithms:
    print(f'Running {algorithm.__class__.__name__}')
    grid_search = GridSearchCV(algorithm, param_grid = param_grid[algorithm.__cl
    grid_search.fit(x_train)
    best_params[algorithm.__class__.__name__] = grid_search.best_params_
    cluster = grid_search.best_estimator_.fit(x_train)
    score = silhouette_score(x_train, cluster.labels_)
    algorithm_score[algorithm.__class__.__name__] = score
```

```
Running KMeans
Running AgglomerativeClustering
Running MiniBatchKMeans
Running Birch
```

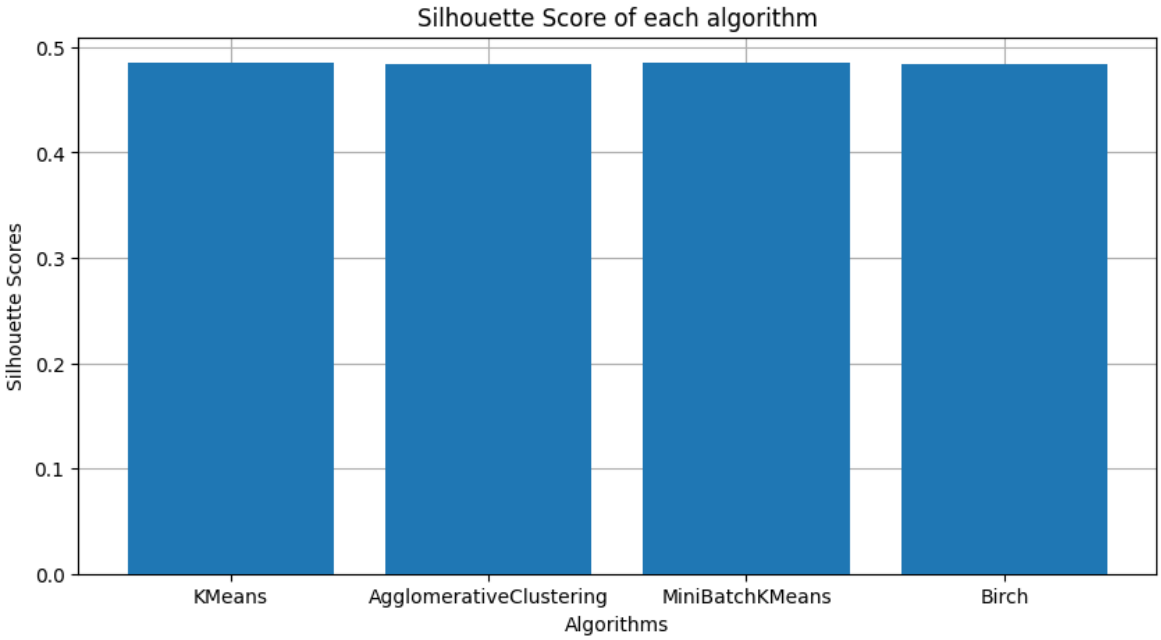
Plotting bar graph to rank silhouette scores of the top 3 models

In [ ]:
```
best_params
```

Out[ ]:
```
{'KMeans': {'n_clusters': 2},
 'AgglomerativeClustering': {'n_clusters': 2},
 'MiniBatchKMeans': {'n_clusters': 2},
 'Birch': {'n_clusters': 2, 'threshold': 0.3}}
```

In [ ]:
```
## Plotting bar graph to compare the scores of each algorithm

plt.figure(figsize=(10,5))
plt.grid()
plt.bar(algorithm_score.keys(), algorithm_score.values(),zorder=3)
plt.title('Silhouette Score of each algorithm')
plt.xlabel('Algorithms')
plt.ylabel('Silhouette Scores')
```

Out[ ]:
```
Text(0, 0.5, 'Silhouette Scores')
```



In [ ]:
```
#Final Model

n_clusters = best_params['KMeans']['n_clusters']
final_model = KMeans(n_clusters=n_clusters,n_init=10)
cluster = final_model.fit(x_train)
score = silhouette_score(x_train, cluster.labels_)
```

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

The algorithms are chosen limited to the sklearn package due to its reliability and stability.In order to filter out the 9 algorithms present in sklearn, compute time is considered as a constraint. The top 4 algorithms based on the time taken to fit the data are selected for Hyperparameter tuning using GridSearchCV. The hyperparameter-tuned model with the highest silhouette score is chosen as the final model.

David Bouldin computes the separation between clusters, but when dealing with more than 500 features, the clusters are bound to overlap, proving that the metric is inefficient. Calinski-Harabasz Index is not chosen as an accuracy metric owing to its assumption of distribution being Spherical. Hence Silhouette score is chosen over Davis Boulding and Calinski - Harabasz Index.

It can be seen that the top 4 shortest compute time are - K-Means,MiniBatchKmeans, Agglomerative and Birch Highest Silhouette score - K-Means

Final Model - K-Means

In [ ]:
```
compute_time
```

Out[ ]:
```
{'KMeans': 2.05,
 'AffinityPropagation': 39.68,
 'AgglomerativeClustering': 10.8,
 'HDBSCAN': 54.48,
 'OPTICS': 113.09,
 'Birch': 13.04,
 'DBSCAN': 0.82,
 'MeanShift': 349.87,
 'MiniBatchKMeans': 0.35}
```

In [ ]:
```
algorithm_score
```

Out[ ]:
```
{'KMeans': 0.4851541220680406,
 'AgglomerativeClustering': 0.484498415381901,
 'MiniBatchKMeans': 0.4851541220680406,
 'Birch': 0.484498415381901}
```

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.

In [ ]:
```
# Program your cluster quality metric here

#Cluster purity, Mutual Information, Fawkes Mallows Score, Adjusted Rand Index.

from sklearn.metrics import adjusted_mutual_info_score, adjusted_rand_score, fow

mis = adjusted_mutual_info_score(y_train,cluster.labels_)
ari = adjusted_rand_score(y_train,cluster.labels_)
fm = fowlkes_mallows_score(y_train,cluster.labels_)

print(f'Adjusted Mutual Information Score: {mis}')
print(f'Adjusted Rand Index: {ari}')
print(f'Fowlkes Mallows Score: {fm}')
```

```
Adjusted Mutual Information Score: 0.5484570161739734
Adjusted Rand Index: 0.3299555315656998
Fowlkes Mallows Score: 0.5764668346638834
```

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.

Clustering Performance

Adjusted mutual information score measures the similarity of the cluster labels and true labels using mutual information and entropy of true and predicted classes. The score of 0.54 indicates a moderate similarity between the true and predicted clusters.

Adjusted Rand Index (ARI) measures the accuracy of pair-wise similarities between predictions and the ground truth. ARI lies in the range -1 to 1, indicating that 0.32 is an above average score.

The Fowlkes Mallows score evaluates the mean of recall and precision (dissimilar pairs) contrasting to ARI where it computes the similarity. With a range between 0 and 1, a higher Fowlkes Mallwos score indicate better clustering performance. The score of 0.57 from the clustering above can be interpreted as a good result, especially considering the extensive feature space on which the model was trained.

Conclusions

1. Spectral Clustering does not perform effectively when a high feature space is involved.
2. Large number of features resulting sub-par clustering performance
3. Compute time of each algorthm played a critical filtering technique
4. K -Means performed the best out of the rest of the clustering algorithms, although it depended heavily on the range of clusters given in the grid search

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.

In [ ]:
```
# Program your dimensionality reduction here.

from sklearn.decomposition import PCA

pca = PCA(n_components=3,random_state=42)

x_train_pca = pca.fit_transform(x_train)

x_train_pca

best_params = {'n_clusters': range(4,8)}

grid_search = GridSearchCV(KMeans(n_init=10), param_grid = best_params, cv=5, sc
grid_search.fit(x_train_pca)
print(grid_search.best_params_)
grid_search.best_estimator_.fit(x_train_pca)
score = silhouette_score(x_train_pca, grid_search.best_estimator_.labels_)
print(score)
```

```
{'n_clusters': 4}
0.4797572714880095
```

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.

Sklearn presents 2 dimensionality reduction techniques - Prinicipal Component Analysis (PCA) and Random Projections (Feature Agglomeration applied to Hierarchichal clustering only).The prinicipal component analysis method is chosen due to 2 reasons -

1. Linearity - PCA performs a linear (rotation) transformation while random projections performs a non-linear reduction. Linearity aids in interpreting the transformation.
2. Variance - PCA ensures the maximum variance of the features are retained after the reduction while Random projections retains pairwise distances between the samples in the dataset.

PCA also has it's disadvantages such as the assumption of a gaussian distribution and outlier sensitivity.

GridsearchCV is utilized for selecting the hyperparameters for the K-Means clustering with the silhouette score as the accuracy metric. Silhouette score is a measure of the similarity of samples within a cluster. Although with advantages such as cluster shape insensitivity, it has disadvantages such as distance dependent and outlier sensitivity which are to be considered.

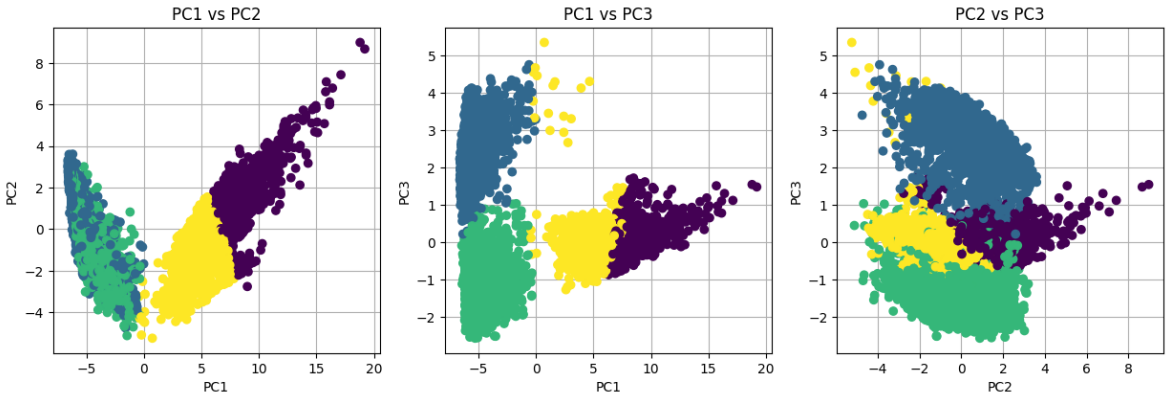
1.4 Plotting the clusters in the reduced feature space [5 marks]

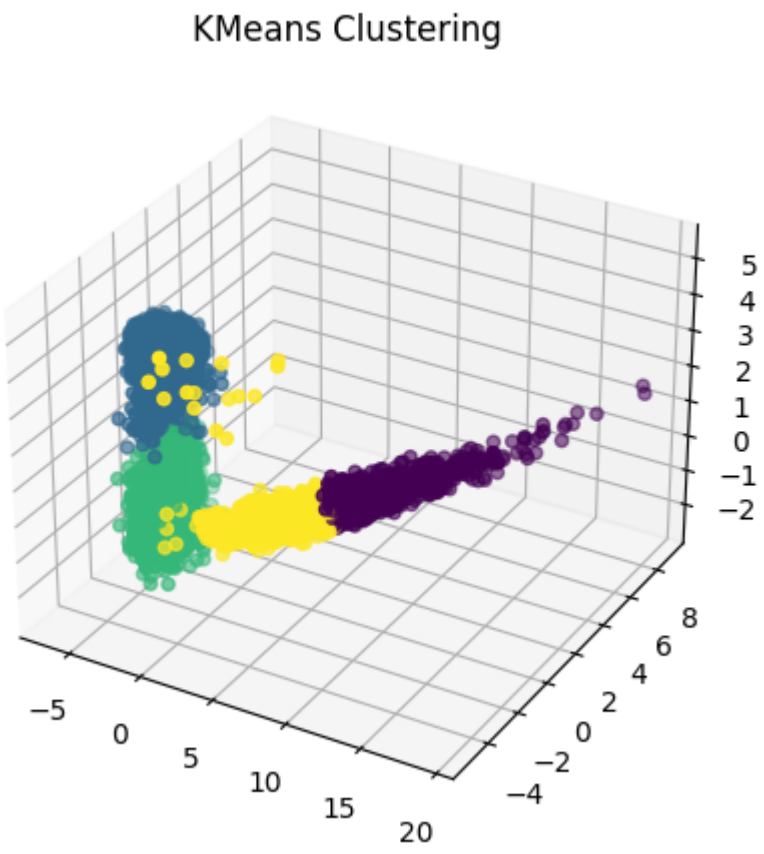
Now that you have transformed your data into 3 dimensions, create a set of plots to show the clusters in these reduced dimensions. Make separate plots using the clustering labels from part 1.1 and also the ground truth labels to show how well it has been clustered. Where possible combine the figures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

In [ ]:
```
# Program your plots here

from mpl_toolkits.mplot3d import Axes3D

## plotting 2D scatter plots for the combinations of the 3 principal components

fig = plt.figure(figsize=(15,15))
ax = fig.add_subplot(3,3,1)
ax.scatter(x_train_pca[:,0],x_train_pca[:,1],c=grid_search.best_estimator_.label
plt.title('PC1 vs PC2')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.grid()


ax = fig.add_subplot(3,3,2)
ax.scatter(x_train_pca[:,0],x_train_pca[:,2],c=grid_search.best_estimator_.label
plt.title('PC1 vs PC3')
plt.xlabel('PC1')
plt.ylabel('PC3')
plt.grid()

ax = fig.add_subplot(3,3,3)
ax.scatter(x_train_pca[:,1],x_train_pca[:,2],c=grid_search.best_estimator_.label
plt.title('PC2 vs PC3')
plt.xlabel('PC2')
plt.ylabel('PC3')
```

```
plt.grid()

fig = plt.figure(figsize=(5,5))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x_train_pca[:,0],x_train_pca[:,1],x_train_pca[:,2],c=grid_search.best
plt.title('KMeans Clustering')
plt.show()
```





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?

It is observed in the 3D projection that K-Means clustering has efficiently clustered the reduced components. 2-dimensional plots showcase overlap of the 4 clusters across the 3 dimensions. A major inference to be noted is that dimenionality reduction methods such as PCA and Random projections aid in simplifying complex number of features. After the reduction process, visualization help in better interpretation and understanding of the features.

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.

In [ ]:
```
import numpy as np
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/bloodmni

# 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
```

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

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

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.

The first model selected is a set of 3 fully connected layers. Linearly connected layers aid in learning hierarchichal features and patterns present throughout the sample/image. This acts as a baseline model for the comparison of CNN and Deep-CNN. The second architecture comprises of 1 convolutional layer followed by a a linear layer combining the kernel outputs. Convolutional layers are ideal for image classification helping in finding specific features in the dataset. One layer of CNN identifies the edges suitable for finding the output objects. The third model chosen is the Deep Convolutional Neural Networks consisting of 2 convolutional layers followed by a linear layer combining all the resuls. The second convolutional layer aids in identifying abstract object parts in the image samples to find image classification. The final architecture considered for the classification problem is the Support Vector Machine (SVM). Being memory efficient along with versatile outlier handling makes SVM a good choice in classification problems.

Cross Entropy Loss is used throughout the problem as the loss critereon along with Stochastic Gradient Descent (SGD) used as an optimizer owing to the effective computation of classification of images.

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.

In [ ]:
```
# Program your models here. You can use as many cells as necessary but aim to be
```

In [ ]:
```
### Importing the necessary libraries

from torch import nn
import torch
from sklearn.preprocessing import StandardScaler
import torch.optim as optim
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import torch.nn.functional as F
import itertools
```

Data Preparation

In [ ]:
```
## Preparing the data

x_train = dataset['train_images']
y_train = dataset['train_labels']
x_val = dataset['val_images']
y_val = dataset['val_labels']
x_test = dataset['test_images']
y_test = dataset['test_labels']

## Reshaping the data - FCN and SVM

x_train_1d = x_train.reshape(len(x_train),-1)
x_val_1d = x_val.reshape(len(x_val),-1)
x_test_1d = x_test.reshape(len(x_test),-1)

y_train = y_train.reshape(len(y_train),-1).squeeze()
y_val = y_val.reshape(len(y_val),-1).squeeze()
y_test = y_test.reshape(len(y_test),-1).squeeze()

## Standardizing on-dimensional data

scaler = StandardScaler()
x_train_1d = scaler.fit_transform(x_train_1d)
x_val_1d = scaler.transform(x_val_1d)
x_test_1d = scaler.transform(x_test_1d)

## Converting the data into tensors

x_train = torch.from_numpy(x_train).float()
x_train_1d = torch.from_numpy(x_train_1d).float()
y_train = torch.from_numpy(y_train).long()

x_val = torch.from_numpy(x_val).float()
x_val_1d = torch.from_numpy(x_val_1d).float()
y_val = torch.from_numpy(y_val).long()

x_test = torch.from_numpy(x_test).float()
x_test_1d = torch.from_numpy(x_test_1d).float()
y_test = torch.from_numpy(y_test).long()

### Standardizing data

x_train = (x_train - x_train.mean(dim=(1,2,3),keepdim=True))/x_train.std(dim=(1,
x_val = (x_val - x_val.mean(dim=(1,2,3),keepdim=True))/x_val.std(dim=(1,2,3),kee
x_test = (x_test - x_test.mean(dim=(1,2,3),keepdim=True))/x_test.std(dim=(1,2,3)

## Shufffling columns from conv2d suitable input

x_train_perm = x_train.permute(0,3,1,2)
x_val_perm = x_val.permute(0,3,1,2)
x_test_perm = x_test.permute(0,3,1,2)
```

Architecture - 1 - Fully Connectecd Network

In [ ]:
```
class FCN(nn.Module):
    def __init__(self,lin1=128,lin2=256,lin3=128):
        super(FCN, self).__init__()
        self.fc1 = nn.Linear(28*28*3, lin1)
```

```
        self.fc2 = nn.Linear(lin1, lin2)
        self.fc3 = nn.Linear(lin2, lin3)

    def forward(self, x):
        x = x.reshape(-1, 28*28*3)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.softmax(x,dim=1)

    def train(self,x_train,y_train,epochs, optimizer, loss):
        loss_list = []
        accuracy_list = []
        for epoch in range(epochs):

            optimizer.zero_grad()
            output = self.forward(x_train)
            loss_val = loss(output, y_train)
            loss_val.backward()
            optimizer.step()

            loss_list.append(loss_val.item())
            accuracy_list.append(accuracy_score(y_train,output.argmax(dim=1)))
            #print(f'Epoch: {epoch}, Loss: {loss_val.item()}')
        return loss_list, accuracy_list

    def evaluate(self,loss,x_val,y_val):
        with torch.no_grad():
            output = self.forward(x_val)
            loss_val = loss(output, y_val)
            accuracy = accuracy_score(y_val,output.argmax(dim=1))
            #print(f'Accuracy: {accuracy}, Loss: {loss_val.item()}')
            return accuracy, loss_val.item()
```

In [ ]:
```
lr = 0.0001
loss_func = nn.CrossEntropyLoss()

fcnTrain = FCN()
optimizer3 = optim.Adam(fcnTrain.parameters(), lr=lr)
loss_list1,accuracy_list1 = fcnTrain.train(x_train_1d,y_train,25,optimizer3,loss

fcnTrain.evaluate(loss_func,x_val_1d,y_val)
```

Out[ ]:
```
(0.42348130841121495, 4.806222915649414)
```

Architectures - 2 - CNN

In [ ]:
```
class CNN(nn.Module):
    def __init__(self,convKern1=2,convKern2=3):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, convKern1)
        self.maxpool = nn.MaxPool2d(2)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(64*(int((29-convKern1)/2)**2), 16)


    def forward(self, x):
        x = self.flatten(self.maxpool(self.conv1(x)))
```

```
        x = F.relu(self.fc1(x))
        return x


    def train(self,x_train,y_train,epochs, optimizer, loss):
        loss_list = []
        accuracy_list = []
        for epoch in range(epochs):

            optimizer.zero_grad()
            output = self.forward(x_train)
            loss_val = loss(output, y_train)
            loss_val.backward()
            optimizer.step()

            loss_list.append(loss_val.item())
            accuracy_list.append(accuracy_score(y_train,output.argmax(dim=1)))
            #print(f'Epoch: {epoch}, Loss: {loss_val.item()}')
        return loss_list, accuracy_list

    def evaluate(self,loss,x_val,y_val):
        with torch.no_grad():
            output = self.forward(x_val)
            loss_val = loss(output, y_val)
            accuracy = accuracy_score(y_val,output.argmax(dim=1))
            #print(f'Accuracy: {accuracy}, Loss: {loss_val.item()}')
            return accuracy, loss_val.item()
```

In [ ]:
```
lr = 0.001
epoch = 30
loss_func = nn.CrossEntropyLoss()

neuralNet1 = CNN()
optimizer1 = optim.Adam(neuralNet1.parameters(), lr=lr)
neuralNet1.train(x_train_perm,y_train,epoch,optimizer1,loss_func)

neuralNet1.evaluate(loss_func,x_val_perm,y_val)
```

Out[ ]:
```
(0.7535046728971962, 0.7220208048820496)
```

In [ ]:
```
### Hyperparameter Tuning

### Hyperparameter tuning with VALIDATION SET
hyperparameter_grid = {
    'convKern1': [2,3]
}

all_hyperparameter_combinations = list(itertools.product(*hyperparameter_grid.va

criterion = nn.CrossEntropyLoss()

for hyperparameter_values in all_hyperparameter_combinations:

    current_hyperparameters = dict(zip(hyperparameter_grid.keys(), hyperparamete


    model = CNN(**current_hyperparameters)
    optimizer = optim.Adam(model.parameters(), lr=0.1)
    num_epochs = 10
```

```
    for epoch in range(num_epochs):
        model.train(x_train_perm,y_train,epoch,optimizer1,loss_func)

    validation_loss,validation_acc = model.evaluate(loss_func,x_val_perm,y_val)

    if lowest_loss < validation_loss:
        lowest_loss = validation_loss
        best_hyperparameters = current_hyperparameters
```

In [ ]:
```
best_hyperparameters
```

Out[ ]:
```
{'convKern1': 2}
```

In [ ]:
```
bestCNN = CNN(**best_hyperparameters)
optimizer = optim.Adam(bestCNN.parameters(), lr=0.001)
loss_list2, accuracy_list2 = bestCNN.train(x_train_perm,y_train,15,optimizer,los
```

Architecture - 3 - Deep CNN

In [ ]:
```
class DeepCNN(nn.Module):
    def __init__(self,convKern1=2,convKern2=3):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, convKern1)
        self.conv2 = nn.Conv2d(64, 32, convKern2)

        self.maxpool = nn.MaxPool2d(2)
        self.avgpool = nn.AvgPool2d(3)
        self.flatten = nn.Flatten()

        self.fc1 = nn.Linear(288, 16)

    def forward(self, x):
        x = self.maxpool(self.conv1(x))
        x = self.avgpool(self.conv2(x))
        x = self.fc1(self.flatten(x))
        x = F.relu(x)
        return x


    def train(self,x_train,y_train,epochs, optimizer, loss):
        loss_list = []
        accuracy_list = []
        for epoch in range(epochs):

            optimizer.zero_grad()
            output = self.forward(x_train)
            loss_val = loss(output, y_train)
            loss_val.backward()
            optimizer.step()

            loss_list.append(loss_val.item())
            accuracy_list.append(accuracy_score(y_train,output.argmax(dim=1)))
            #print(f'Epoch: {epoch}, Loss: {loss_val.item()}')
        return loss_list, accuracy_list

    def evaluate(self,loss,x_val,y_val):
        with torch.no_grad():
```

```
            output = self.forward(x_val)
            loss_val = loss(output, y_val)
            accuracy = accuracy_score(y_val,output.argmax(dim=1))
            #print(f'Accuracy: {accuracy}, Loss: {loss_val.item()}')
            return accuracy, loss_val.item()
```

In [ ]:
```
lr = 0.001
epoch = 25
loss_func = nn.CrossEntropyLoss()

deepCNN = CNN()
optimizer2 = optim.Adam(deepCNN.parameters(), lr=lr)
loss_list3,accuracy_list3 = deepCNN.train(x_train_perm,y_train,epoch,optimizer2,

deepCNN.evaluate(loss_func,x_val_perm,y_val)
```

Out[ ]:
```
(0.7441588785046729, 0.9448304176330566)
```

Architecture - 4 - Support Vector Machine

In [ ]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

mod_svm =  SVC(kernel='linear', C=1, random_state=42)
mod_svm.fit(x_train_1d,y_train)

accuracy_score(y_val,mod_svm.predict(x_val_1d))
```

Out[ ]:
```
0.8183411214953271
```

Testing

In [ ]:
```
### Testing the best model on the test set

arch1_acc = accuracy_score(fcnTrain.forward(x_test_1d).argmax(dim=1),y_test)

arch2_acc = accuracy_score(bestCNN.forward(x_test_perm).argmax(dim=1),y_test)

arch3_acc = accuracy_score(deepCNN.forward(x_test_perm).argmax(dim=1),y_test)

arch4_acc = accuracy_score(y_test,mod_svm.predict(x_test_1d))

print(f'Accuracy of FCN: {arch1_acc}')
print(f'Accuracy of CNN: {arch2_acc}')
print(f'Accuracy of Deep CNN: {arch3_acc}')
print(f'Accuracy of SVM: {arch4_acc}')
```

```
Accuracy of FCN: 0.4238526746565332
Accuracy of CNN: 0.6971645717626425
Accuracy of Deep CNN: 0.7322420344928383
Accuracy of SVM: 0.7866121017246419
```

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.

It can be noted that the fully connected layer performed sub-par with respect to the other algorithms. The grid search for deep CNN was not performed as the number of neurons between the layers are interlinked and increase complexity when parameter tuning is performed. In the case of simple CNN, the linear layer's input channels is directly related to the kernel size of the conv2d layer. Max pooling and average pooling are utilized as sub-sampling methods to extract specific features. Support Vector Machine works intuitively with the C score and the kernel type. As SVM does not have epochs, they are not plotted in terms of iterations. Batch processing using data loader could also be performed for better performance and efficiency.

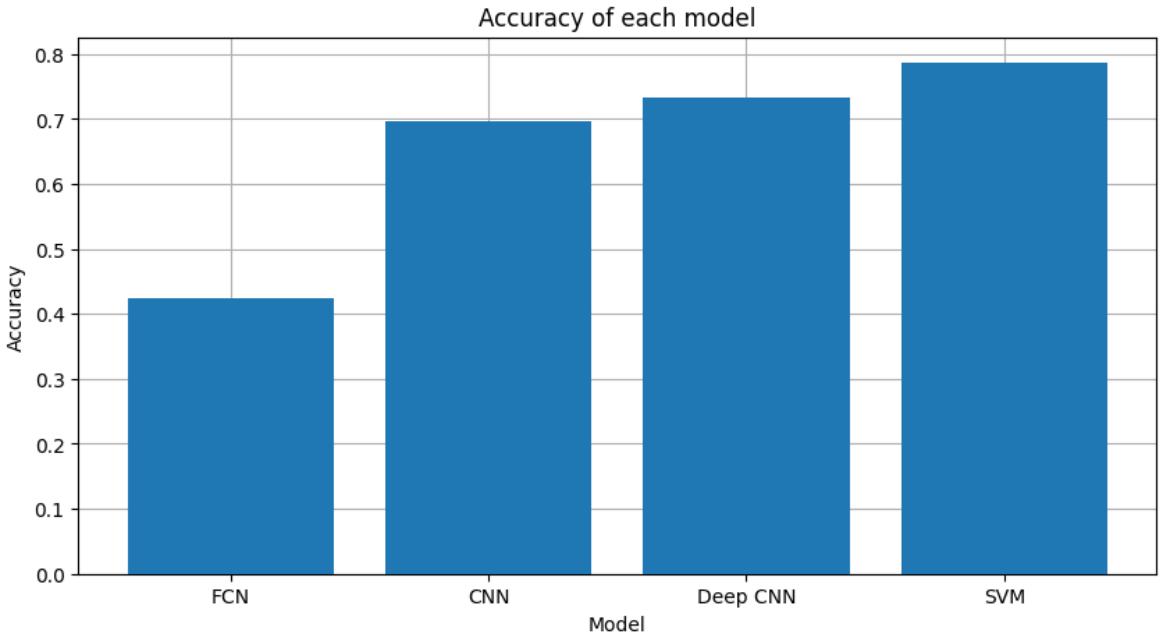
2.3 Classification results based on the test data [10 marks]

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.

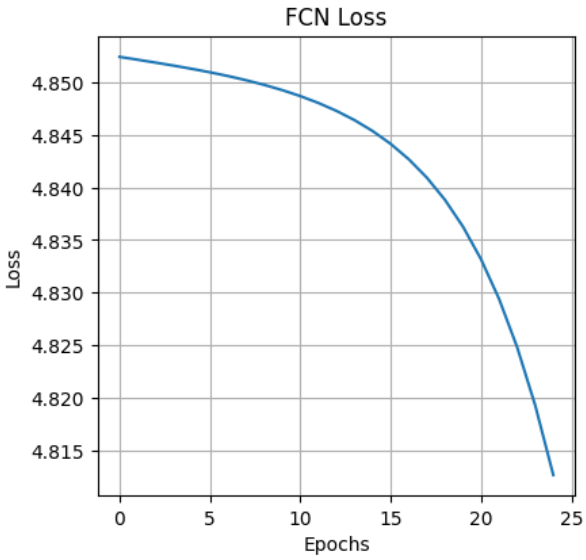
In [ ]:
```
# Program your plots here.

## Bar graph to compare accuracy of each model

plt.figure(figsize=(10,5))
plt.grid()
plt.bar(['FCN','CNN','Deep CNN','SVM'], [arch1_acc,arch2_acc,arch3_acc,arch4_acc
plt.title('Accuracy of each model')
plt.xlabel('Model')
plt.ylabel('Accuracy')


plt.figure(figsize=(20,15))

fig,ax = plt.subplots(3,2,figsize=(10,15))

for axi in ax.flatten():
    axi.grid(True)

ax[0,0].plot(loss_list1)
ax[0,0].set_title('FCN Loss')
ax[0,0].set_xlabel('Epochs')
ax[0,0].set_ylabel('Loss')

ax[0,1].plot(accuracy_list1)
ax[0,1].set_title('FCN Accuracy')
ax[0,1].set_xlabel('Epochs')
ax[0,1].set_ylabel('Accuracy')


ax[1,0].plot(loss_list2)
ax[1,0].set_title('CNN Loss')
ax[1,0].set_xlabel('Epochs')
ax[1,0].set_ylabel('Loss')

ax[1,1].plot(accuracy_list2)
ax[1,1].set_title('CNN Accuracy')
ax[1,1].set_xlabel('Epochs')
ax[1,1].set_ylabel('Accuracy')
```

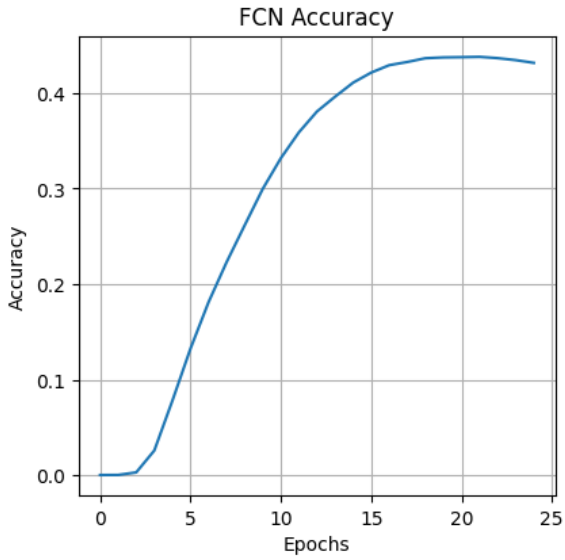
```
ax[2,0].plot(loss_list3)
ax[2,0].set_title('Deep CNN Loss')
ax[2,0].set_xlabel('Epochs')
ax[2,0].set_ylabel('Loss')

ax[2,1].plot(accuracy_list3)
ax[2,1].set_title('Deep CNN Accuracy')
ax[2,1].set_xlabel('Epochs')
ax[2,1].set_ylabel('Accuracy')
```

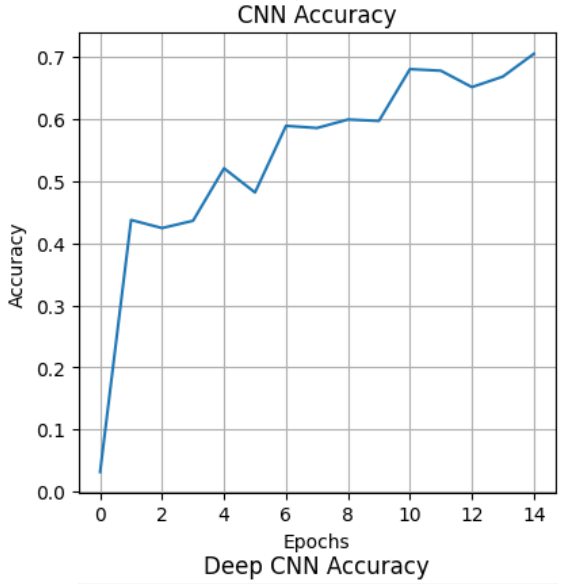
Out[ ]: Text(0, 0.5, 'Accuracy')

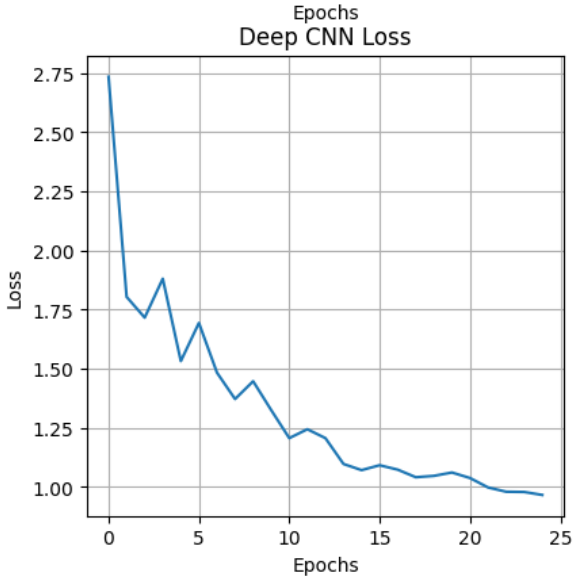


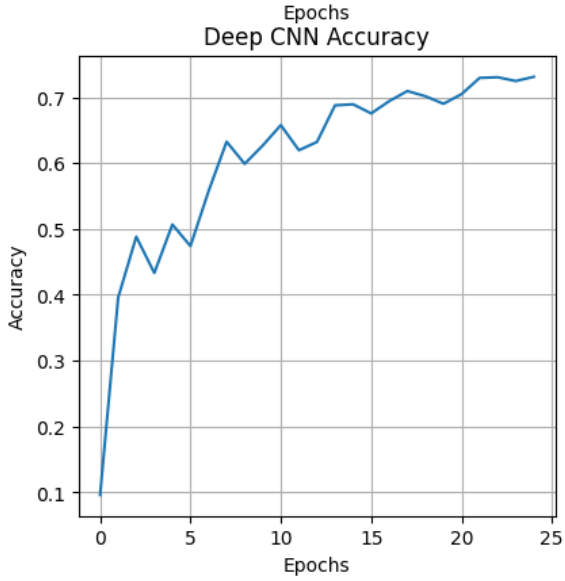
<Figure size 2000x1500 with 0 Axes>











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:

It can be clearly seen that the Deep CNN and SVM perform better in the classification problem. The simple CNN is unable to capture and extract the features from edges of the images completely. Fully connected network performs sub-par relative to CNN due to the inability to capture features from images.