# CS5062 – Machine Learning

## Assessment 1

## Task 1: Regression

## Data Import:

The dataset that we have used here is mostly used by many machine-learning algoritms especially regression models. The dataset here is aimed to explore the relationship between the level of prostate - specified antigen (PSA) and the various clinical measures in men who were about to undergo a radical prostatectomy. Let us have look at each column of data,

1. Log Cancer volume (**logCancerVol**) : The logarithm of the cancer volume is represented here. It lists the measurement of the size of the prostates cancerous tissue.
2. Log Cancer Weight (**logCancerWeight**): The logarithm of the weight of the prostate cancer is given here.
3. Age (**age**) : The age of the patients are given here. This is sometimes considered as an important factor as certain age groups are considered to have high risk of developing cancerous tissues.
4. Log of the amount of benign prostatic hyperplasia (**logBenighHP**) : The logarithm of the amount of Benign prostatic hyperplasia is given here. The BPH is a non-cancerous enlargement of the prostate. It can help to distinguish if the cancer is benign or malignant.
5. Seminal vesicle invasion (**svi**) : Seminal vesicle invasion is an important feature in diagnosing the cancer. Here, values between 0 and 1 represent whether or not there is a seminal vesicle invasion.
6. Log of capsular penetration (**logCP**) : The logarithm of the capsular penetration is represented here. It can help to know the extent of how much of the prostate's outer layer has been affected by the cancer.
7. Gleason Score (**gleasonScore**) : The doctors make use of this score to determine the aggressiveness of the prostate cancer.
8. Percent of Gleason Score 4 or 5 (**gleasonS45**) : The percentage of Gleason scores are given here. A value of 60 in the column indicates that 60% of the patient sample biopsy has Gleason scores of 4 or 5. Gleason score of 4 consists of abnormal glandular structures and Gleason score of 5 consists of small or poorly differentiated glands.
9. Log of cancer Antigen (**levelCancerAntigen**) : The Logarithm of the cancer antigen is given here. The antigen is produced by the prostate and it level of production can indicate the health of the prostate.

The data is being loaded into our programming environment with the help of pandas dataframe. Below is an example code snippet,

```
df= pd.read_csv('C:/Users/prade/Downloads/Task1_RegressionTask_CancerData.txt', delimiter="\t")
```

```
df
```

| | index | logCancerVol | logCancerWeight | age | logBenighHP | svi | logCP | gleasonScore | gleasonS45 | levelCancerAntigen | train |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | -0.579818 | 2.769459 | 50 | -1.386294 | 0 | -1.386294 | 6 | 0 | -0.430783 | T |
| **1** | 2 | -0.994252 | 3.319626 | 58 | -1.386294 | 0 | -1.386294 | 6 | 0 | -0.162519 | T |
| **2** | 3 | -0.510826 | 2.691243 | 74 | -1.386294 | 0 | -1.386294 | 7 | 20 | -0.162519 | T |
| **3** | 4 | -1.203973 | 3.282789 | 58 | -1.386294 | 0 | -1.386294 | 6 | 0 | -0.162519 | T |
| **4** | 5 | 0.751416 | 3.432373 | 62 | -1.386294 | 0 | -1.386294 | 6 | 0 | 0.371564 | T |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **92** | 93 | 2.830268 | 3.876396 | 68 | -1.386294 | 1 | 1.321756 | 7 | 60 | 4.385147 | T |
| **93** | 94 | 3.821004 | 3.896909 | 44 | -1.386294 | 1 | 2.169054 | 7 | 40 | 4.684443 | T |
| **94** | 95 | 2.907447 | 3.396185 | 52 | -1.386294 | 1 | 2.463853 | 7 | 10 | 5.143124 | F |
| **95** | 96 | 2.882564 | 3.773910 | 68 | 1.558145 | 1 | 1.558145 | 7 | 80 | 5.477509 | T |
| **96** | 97 | 3.471966 | 3.974998 | 68 | 0.438255 | 1 | 2.904165 | 7 | 20 | 5.582932 | F |

97 rows × 11 columns

## Data Preprocessing:

Data Preprocessing is an important step as it helps to improve out models training and prediction process. After loading our dataset into our programming environment, we have to check for any missing values. If any missing values are found, then we have to either fill the cell with some relevant information or drop that row that contains the cell. You can see the below snippet on how to check for any missing values,

```
df.isnull().sum()
```

```
index              0
logCancerVol       0
logCancerWeight    0
age                0
logBenighHP        0
svi                0
logCP              0
gleasonScore       0
gleasonS45         0
levelCancerAntigen 0
train              0
dtype: int64
```

The next step of Data Preprocessing is Feature Scaling. Since we have a numerical dataset that has different scale of measures on each column we need to Normalize or standardize these features. We have used StandardScalar from sklearn to standardize the features. Below is the code Snippet,

The final step is to split the data, in our case we have a train column in our dataset. We have to split the data into train set and test set, from the train column in our dataset, the labels with 'T' are used as train set and the labels with 'F' values are used as test set.

```
train_set = df[df['train'] == 'T'].drop(['train'], axis = 1)
test_set = df[df['train'] == 'F'].drop(['train'], axis = 1)
```

**Regression Models:**

For our prediction and inference purpose, we have been asked to use multiple linear regression models. Let us have a look at out first regression model,

**Ridge Regression:**

The Ridge regression is implemented by importing it from the class sklearn.linear_model. It is a regression method that adds a penalty term (L2) to the objective function. It encourages the coefficients to be small but doesn't force them to be zero. The hyperparameter α, greatly controls how heavy the penalty should be. Let us have a look at how the model is built,

```
from sklearn.linear_model import Ridge
ridge_model = Ridge()
ridge_model.fit(X,Y)
```

The model is initialized with the default parameters. In order to improve the model, we need to customize the parameters using cross validation. The model is then fitted to out training and target data. Next the model is trained so that prediction can be done on the test set. We then make use of the Mean squared metric, which is commonly used in regression models to access if the model is a good fit or not. The MSE is calculated between the test values and predicted values. With the default parameters, the MSE for the ridge model using the cancer dataset is 0.489.

Now let us perform the cross validation to find the best possible alpha value, we are initializing a k-fold cross validation. In our case are performing a 10-fold cross validation. Following cross-validation, the best alpha value is given as 1.988. Now that the optimal alpha value has been included as a parameter in the Ridge model, following model training, the MSE value is 0.479.

```
ridge_model = Ridge(alpha = 1.988)
ridge_model.fit(X,Y)
y_pred_ridge = ridge_model.predict(Xt)
mse_ridge = mean_squared_error(Yt, y_pred_ridge)
print(f"Ridge Mean Squared Error: {mse_ridge}")
```

```
Ridge Mean Squared Error: 0.4793801585799768
```

**Lasso Regression:**

The Lasso regression is implemented by importing it from the class sklearn.linear_model. It is a regression method that adds a penalty term (L2) to the objective function. It encourages the coefficients to be to be exactly zero. The weight of the penalty has been. Let us have a look at how the model is built,

```
lasso_model = Lasso()
lasso_model.fit(X,Y)
y_pred_lasso = lasso_model.predict(Xt)
mse_lasso = mean_squared_error(Yt, y_pred_lasso)
print(f"Lasso Mean Squared Error: {mse_lasso}")
```

The Lasso follows the same procedure being done in the ridge regression. The cross validation is performed here to obtain the best alpha value, which is 0.007. Now we substitute the default parameter with the optimal alpha value. Following the completion of the model training, the MSE is 0.510.

```
from sklearn.metrics import mean_squared_error
lasso_model = Lasso(alpha=0.007)
lasso_model.fit(X_train, Y_train)
y_pred_lasso = lasso_model.predict(X_test)
mse_lasso = mean_squared_error(Y_test, y_pred_lasso)
print(f"Lasso Mean Squared Error: {mse_lasso}")
```

```
Lasso Mean Squared Error: 0.5106848638178022
```

**Linear Regression:**

The Linear regression is implemented by importing it from the class sklearn.linear_model. The model is initialized with the default parameters. No cross validation is being done here, as there is no valid parameters that can help improve the model perform better in our case. The model is then fitted and then the model is trained so that prediction can be done on the test set. We then make use of the Mean squared metric, which is commonly used in regression models to access if the model is a good fit or not. The MSE is calculated between the test values and predicted values. With the default parameters, the MSE for the ridge model using the cancer dataset is 0.508.

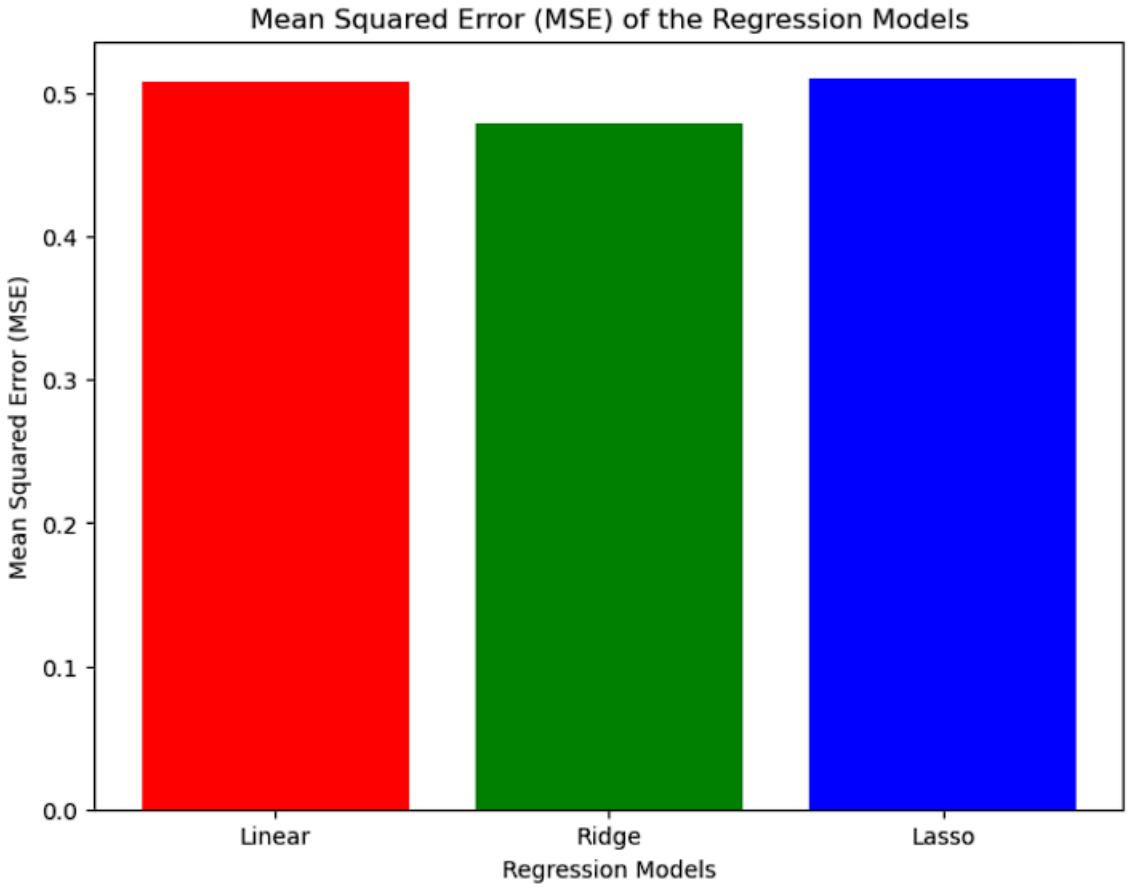
```
from sklearn.linear_model import LinearRegression
linear_model = LinearRegression()
linear_model.fit(X,Y)
y_pred_linear = linear_model.predict(Xt)
mse_linear = mean_squared_error(Yt, y_pred_linear)
print(f"Linear Mean Squared Error: {mse_linear}")
```

**Evaluation:**

To evaluate let us have a look at the mean squared error of each models, and determine which is the best model for the prediction.

| Regression Models | Alpha Value | Mean Squared Error(MSE) |
|---|---|---|
| Ridge Regression | Default (alpha = 1) | 0.489 |
| | Best value(alpha = 1.988) | 0.479 |
| Lasso Regression | Default (alpha = 1) | 1.056 |
| | Best value(alpha = 0.007) | 0.510 |
| Linear Regression | Default | 0.508 |

Now let us look at a visual representation of the models with the best alpha values and determine which has the least MSE.




From the graph, we can clearly see that the Ridge regression model has the best MSE value and it is suitable to infer the clinical measures mostly influencing the cancer antigen. In order to do that, we are calculating the coefficients for each features in the dataset with the help of the trained regression model. By printing the coefficients of each feature, it helps us to understand the importance and significance that each feature has made on the target variable. Now let us have a look at it,

```
Coefficient for logCancerVol:   0.5524452656962917
Coefficient for logCancerWeight:        0.3753558036166238
Coefficient for age:            -0.07981956428820025
Coefficient for logBenighHP:    0.24679674640069263
Coefficient for svi:            0.2746428230451138
Coefficient for logCP:          -0.13035375360622442
Coefficient for gleasonScore:   -0.1938997775794398
Coefficient for gleasonS45:     0.4302634166166712
```

The features that has the highest coefficients are considered to be important features. The value of the coefficients must be higher than zero. With respect to that, the features logCancerVol, logCancerWeight, logBenignHP, svi, gleasonS45 are considered to greatly influence the prediction of the cancer antigen.

## Task 2: Classification

### Data Preprocessing:

In the Data processing technique, first I have implemented the splitting data technique. We have been given two folders train and test, each of these folders contain the images of cats and dogs. We have first separated the images as cats and dogs and placed them in their respective folders. Then they are split into train and test dataset, each of them separate folders that contains the images of cats and dogs.

The next is that I have implemented data augmentation. Data augmentation is also considered as an data processing technique. The data augmentation is done to increase the amount of data available for the training, since we have less data. Having more images greatly improves the performance of the model. Here I have also split the data into train, test and validation.

```
datagen = ImageDataGenerator( rescale = 1.0/255,
                              rotation_range=10,
                              horizontal_flip=True,
                              vertical_flip=True,
                             validation_split = 0.2)

train_generator = datagen.flow_from_directory(train_path,
                                              batch_size=10,
                                              class_mode='binary',
                                              color_mode='rgb',
                                              shuffle=True,
                                              seed=123,
                                              target_size=(200, 200),
                                             subset='training')
validation_generator = datagen.flow_from_directory(train_path,
                                                   batch_size=10,
                                                   class_mode='binary',
                                                   color_mode='rgb',
                                                   shuffle=True,
                                                   seed=123,
                                                   target_size=(200, 200),
                                                  subset='validation')
test_datagen = ImageDataGenerator(rescale = 1./255)
test_generator = test_datagen.flow_from_directory(test_path,
                                                  batch_size=10,
                                                  class_mode='binary',
                                                  color_mode='rgb',
                                                  shuffle=False,
                                                  seed=123,
                                                  target_size=(200, 200))
```

```
Found 1602 images belonging to 2 classes.
Found 400 images belonging to 2 classes.
Found 400 images belonging to 2 classes.
```

**CNN Model:**

The Convolution neural network model is mainly used for Image Classification. Image classification in CNN helps us to categorize features and patterns. CNN makes use of its convolution layers to scan small parts of an image and extracts features from the image. The CNN also makes use of its pooling layer, it greatly reduces the size of the feature map by down sampling them. By doing so, it makes the information simple and manageable for further analysis. It also contains other layers which we will discuss briefly.

The dataset given to us contains images of cats and dogs. The images are labelled and are pre-processed to train our model to make our prediction. Now let us see the architecture of the CNN.

**CNN Model Architecture:**

```
model_cnn = Sequential()

model_cnn.add(Conv2D(filters=32, kernel_size=(3,3), input_shape=(200,200,3),padding='same',activation='relu'))
model_cnn.add(MaxPool2D(pool_size=(2,2),strides=2))

model_cnn.add(Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu'))
model_cnn.add(MaxPool2D(pool_size=(2,2),strides=2))

model_cnn.add(Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu'))
model_cnn.add(MaxPool2D(pool_size=(2,2),strides=2))

model_cnn.add(Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu'))
model_cnn.add(MaxPool2D(pool_size=(2,2),strides=2))
model_cnn.add(Flatten())
model_cnn.add(Dense(512, activation='relu'))
model_cnn.add(Dropout(0.25))
model_cnn.add(Dense(256, activation='relu'))
model_cnn.add(Dropout(0.25))
model_cnn.add(Dense(1, activation='sigmoid'))
```

The first layer is a convolution layer, the parameters used here contains 32 filters with a kernel size of 3 x 3 and an input shape of 150 x 150 x 3 is given. For this layer, we have ReLU as the activation function. The feature maps are produced as an output in this layer and it is passed on top the next layer. The ReLU activation function takes care of the vanishing gradient problem faced by other activation functions. The next layer is the Max Pool layer, lets us have a look at what parameters are being used and how it influences the model. It contains a pool size of 2 x 2, it helps in dividing the image into non overlapping 2 x 2, it greatly decreases the spatial size of the image. The strides are set to 2, it moves the window 2 pixels each time. This layer selects the most important features from the image.

The next three layers are made up of the same Convolution layers and max pool layers. The Max pool has similar parameters being set to it, the pool size is kept as 2 x 2 and the strides have a value of 2 through all of its layer. For the convolution layer, all the parameters except the value of filters are kept same. The filter values are 64, 128 and 256 respectively for the second, third and fourth layer.

The next after that is the Flatten layer, it is an most important layer as it helps to flatten the input data into a 1D array. In our case its takes an input data of shape (9,9,128) and outputs shape of (10368,). The next layer is the dense layer, it contains 256 neurons and ReLU is being kept as the activation function. The final layer is also a Dense layer, it contains about 128 neurons and it uses Sigmoid as its activation function. The Sigmoid activation function is appropriate for a two-class classification problem since it produces a probability distribution over the two classes of the problem.

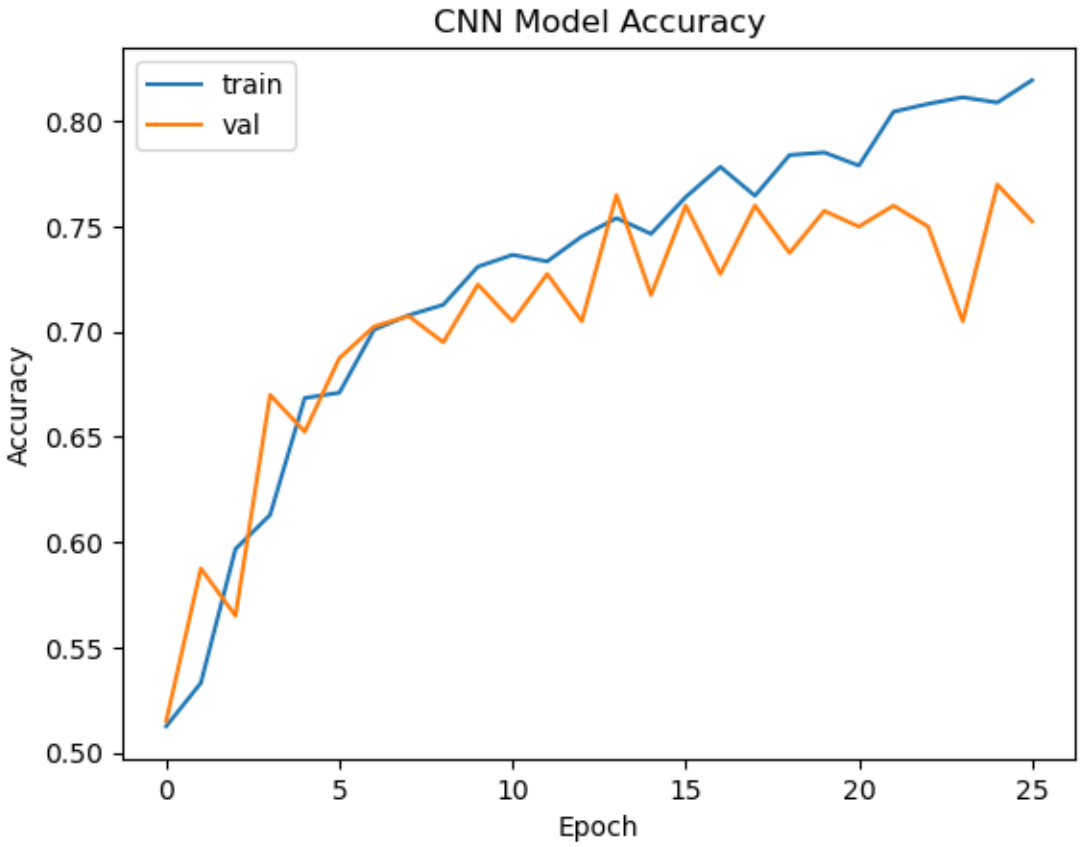
```
model_cnn.compile(optimizer= tf.keras.optimizers.Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
```

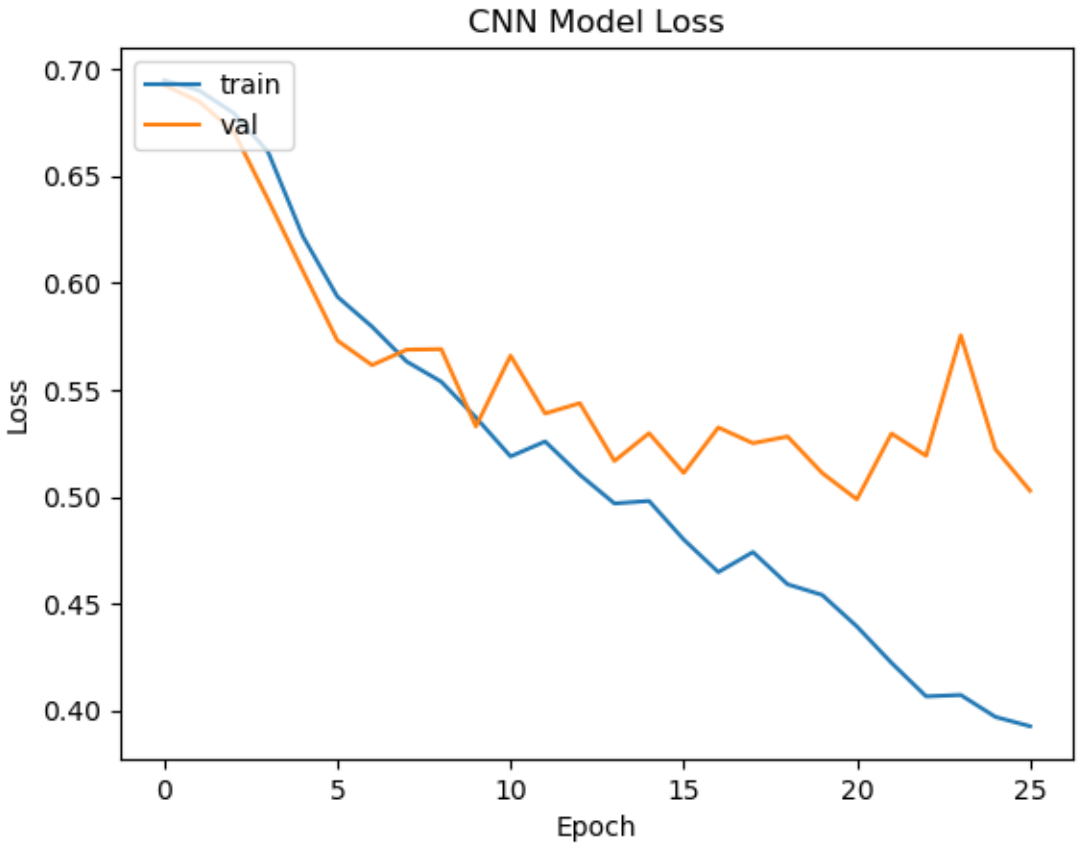
```
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=1,
                                           mode="auto", baseline=None, restore_best_weights=True)
checkpoint_cnn = keras.callbacks.ModelCheckpoint("cnn_best_model.h5", save_best_only=True)
history_cnn = model_cnn.fit(train_generator, epochs=20, validation_data=validation_generator,
                            verbose=1, callbacks=[early_stop,checkpoint_cnn])
```

For the third subtask, let us try to justify the training process for the CNN model. After building the CNN model, we have to compile it. In order to compile it we are utilising the compile function, which has several crucial arguments to it. The optimizer is used for adjusting the internal parameters such as the weight and bias, here we have made of Adam

as our optimizer. It is commonly used optimizer in machine learning. The Adam optimizer is given a learning rate of 0.0001. Next is the loss parameter, here we have assigned the binary crossentropy as our loss function. This needs to be specified because it will help us in the computation of our model's loss function. The metric is assigned as accuracy, this will help us to monitor the accuracy of the training model.

The next step after this is to fit out model. This is a crucial step in our machine learning process. It helps the model to learn from the training and target dataset. The fit function also need some parameter tuning. Some of the most important parameters to take into consideration are Early Stopping and Checkpoint. Early stopping helps to stop the training when a larger value of epoch is given, it stops the model when the loss function on the target set degrades, this also prevents the model from overfitting. It helps in saving a lot off computing power. The restore_best_weights parameter helps in restoring the weight from the best epoch. The next is the checkpoint, it stores the best set of weights seen during training in a file with a.h5 extension.






Let us have a look at the accuracy and loss graph, the number of epochs is plotted along the x axis and the accuracy or loss are plotted along the y axis. In both of the graph, we can say that train set's accuracy and loss is performing well as we cannot see many fluctuations in the line. It means that there are enough datasets for the training. The accuracy and loss of the validation fluctuates a lot, it can be due to the relatively limited volume of available images in the dataset. There is no significant gap between the train set and the val set, suggesting that the overfitting is relatively small.

**Evaluation:**

For our evaluation purpose, I have taken two machine learning algorithms one is CNN and the other one is FNN. Now let us have a look at the accuracy and loss of the models when evaluated with the test dataset,

| Models | Accuracy | Loss |
|---|---|---|
| CNN | 75% | 0.598 |
| FNN | 49% | 0.702 |

From the table, we can clearly see that CNN performs well compared to the other model. The CNN has achieved an accuracy of 75 % which means that it can predict a substantial portion of the test dataset. Despite the fact that the loss value of CNN is lower compared to the FNN

model. We can say the result of lower loss of 0.598 is significant, in context to the small dataset given to us. The FNN has demonstrated an accuracy of 49 % which suggests that the prediction of the test dataset is very difficult, and it may not be able to comprehend the underlying data patterns. The FNN has recorded a loss of 0.702, which implies that the predictions may deviate more substantially from the actual target values.

We can even improve the performance even more significantly for both of the models. Some of the layers between the CNN and FNN can be optimized and generalized to increase the accuracy and lower the loss to make predictions more accurate.

Let us have a look at the precision of the predicted values using the test dataset,

**Classification Report**

```
]: #CNN
print(classification_report(true_labels, predicted_labels, target_names=['cat','dog']))
```

```
              precision    recall  f1-score   support

         cat       0.73      0.74      0.74       200
         dog       0.74      0.72      0.73       200

    accuracy                           0.73       400
   macro avg       0.74      0.73      0.73       400
weighted avg       0.74      0.73      0.73       400
```

```
]: #FNN
print(classification_report(true_labels, predicted_labels1, target_names=['cat','dog']))
```

```
              precision    recall  f1-score   support

         cat       0.50      1.00      0.67       200
         dog       0.00      0.00      0.00       200

    accuracy                           0.50       400
   macro avg       0.25      0.50      0.33       400
weighted avg       0.25      0.50      0.33       400
```

For the CNN, we see that the predicted values have a precision of nearly 75% for both the classes. The recall and the f1-score is also consistently high, hovering around 75%. While the FNN has only 50% precision for the cat class. From this we can infer that the CNN model will perform the prediction well for the given dataset.

**Reference:**


1. For CNN and FNN, I referred some parts from the previous coursework, Data Mining with Deep Learning.
2. Referred the code used in practical day 5, for doing the cross validation in Task-1.