CMPE 257 : MACHINE LEARNING

PROJECT REPORT - CIFAR 100 IMAGE CLASSIFICATION

Submitted By: **Group 7**

Team members	Selected ML Algorithm
Qiao Liu - 013802893 **Ching-Min Hu - 013726154** **Dandan Zhao - 013795392** **Fernanda Bordin - 013800638** **Megha Rajam Rao - 013709488** **Rajasree Rajendran - 013774358**	**Bagging classifier**

Google Colab link:

Milestone 3:
https://colab.research.google.com/drive/1qvUDiAum3nDQGZR9SXCFloYzocwUMr8K?authuser=1#scrollTo=W40rmgMaOya2

Bonus:
https://colab.research.google.com/drive/16Z8o05r1rV1cMCS4jDn6bfw50buHnDq5?authuser=1#scrollTo=E1SREgSlQ4Ga

Task Assignment

Task	Description	Names
Data Preparation	Load, pre-process and visualize data	Megha, Ching-Min HU, Dandan zhao
ML methods	KNN classifier, Bagging classifier, Random Forest, Extra Decision Tree	Dandan Zhao, Ching-Min HU
Neural networks	Dense(feedforward) and Densenet121	Fernanda
Neural Networks	Convolutional Neural Network	Qiao, Rajasree
Powerpoint presentation	Input on Data preparation Input on ML algorithms Input on Neural network Input on CNN	Megha Dandan, Ching-Min Fernanda Qiao, Rajasree
Report (contributors)	Overall report coordination Input on Data Preparation Input on Neural network, overall analysis Input on ML algorithms Input on CNN Bonus feature	Megha, Rajasree Megha Rajasree Fernanda Dandan Zhao, Ching-Min Qiao

TABLE OF CONTENTS

Introduction

The CIFAR datasets are labeled subsets of the 80 million tiny images dataset collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The images are of size 32x32 pixels with 3 color channels (RGB). It comprises of 100 classes containing 600 images each (500 training and 100 testing). The classes (fine labels) are grouped into 20 super classes (coarse labels) and corresponding classes. In this report we filtered the CIFAR-100 dataset to select images from the super classes we chose, which are medium-sized mammals and small mammals.

The medium-sized mammals superclass includes the following classes:
fox, porcupine**,** possum, raccoon, skunk.

Small mammal's superclass includes the following classes:
Hamster, mouse, rabbit, shrew, squirrel.

Libraries

1. Numpy
2. Pandas
3. Keras
4. Sklearn
5. Matplotlib
6. Tensorflow
7. Math
8. Time
9. Seaborn

Softwares & Tools

1. Google Colaboratory
2. Python (language)
3. Powerpoint (presentation)
4. Word (report)
5. Google drive (document sharing)

Data Preparation

- In the initial step, Keras API and Dill package were installed. Keras is a higher-level library that streamlines the process of building deep learning networks. It is written in Python and operates over Theano or Tensorflow. Dill is intended to serialize and deserialize Python objects using the Pickle module.
- We imported all the necessary packages and modules and set the Tensorflow as the backend for Keras.
- The quintessential loading step was performed using ***cifar100.load_data()*** function**,** using which we gathered the training and testing data. Further, ***np.random.seed()*** function was used to seed and render the results reproducible.
- The chosen superclass pair is Medium-sized mammals and small mammals. Below are the subclasses in them.

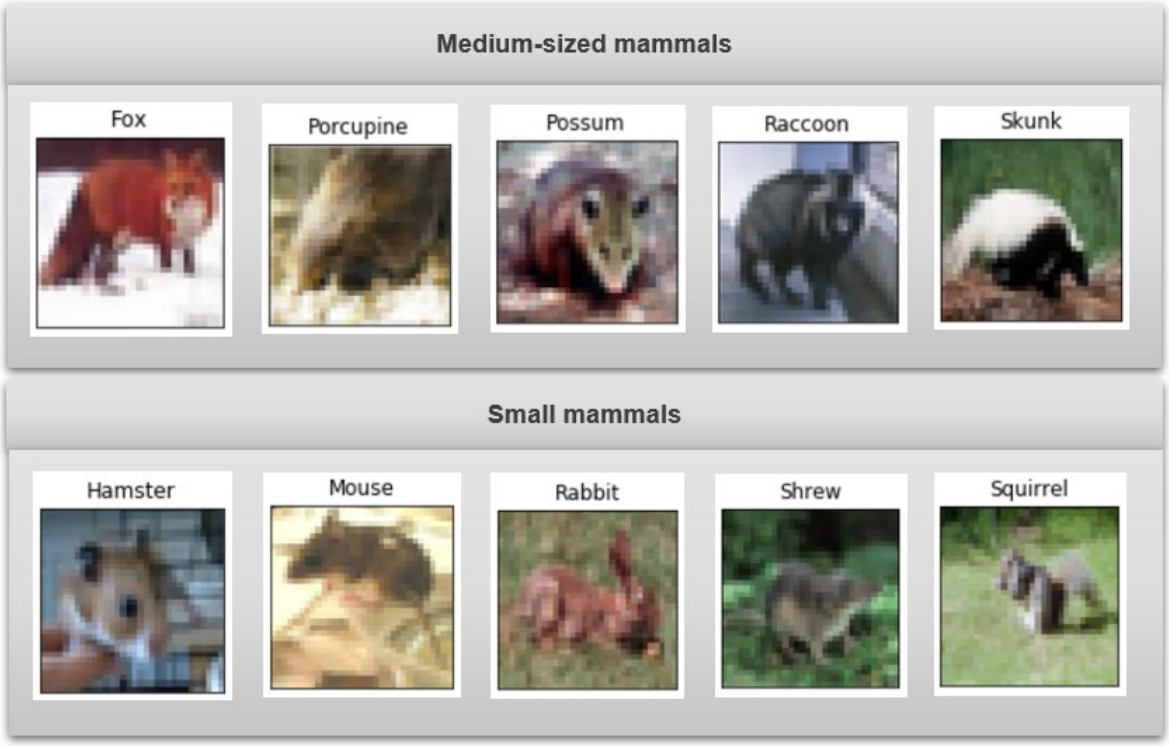


Figure 1 : The Superclass pair

- The class vectors were converted to binary class matrices using ***to_categorical()*** method (for application in neural networks).
- ***np.concatenate()*** function was used to swiftly combine the training and testing data.
- The assigned superclasses were filtered after much experimentation. Initially, we tried to manually find label names but it turned out be cumbersome and time-consuming. Hence, we defined the fine labels with individual class names and isolated their index. Using this information, we spliced the dataset to extract the

required sub-classes from the dataset using loops and ***enumerate()*** function.

Filter out the assigned superclasses

```
# First and foremost, define fine labels
fine_label = [
'apple',                                        # id 0
'aquarium_fish',
'baby',
'bear',
```

```
[ ]  # Splice the dataset to extract the relevant portion of data
     train_slice = np.array([ idx for idx, y in enumerate(y_train) if y[0] in target_index ])
     y_train = y_train[train_slice]
     x_train = x_train[train_slice]

     test_slice = np.array([ idx for idx, y in enumerate(y_test) if y[0] in target_index ])
     y_test = y_test[test_slice]
     x_test = x_test[test_slice]

     slice = np.array([ idx for idx, y in enumerate(y) if y[0] in target_index ])
     y = y[slice]
     x = x[slice]

     print (np.unique(y_train) )
     print (np.unique(y_test) )
```

```
[34 36 50 63 64 65 66 74 75 80]
[34 36 50 63 64 65 66 74 75 80]
```

- However, we had selected different subsets of data for the three milestones. For the first milestone, we worked with the entire dataset during training and testing. As we progressed, we chose smaller subsets as shown below, to mimic a real-world imperfect dataset.

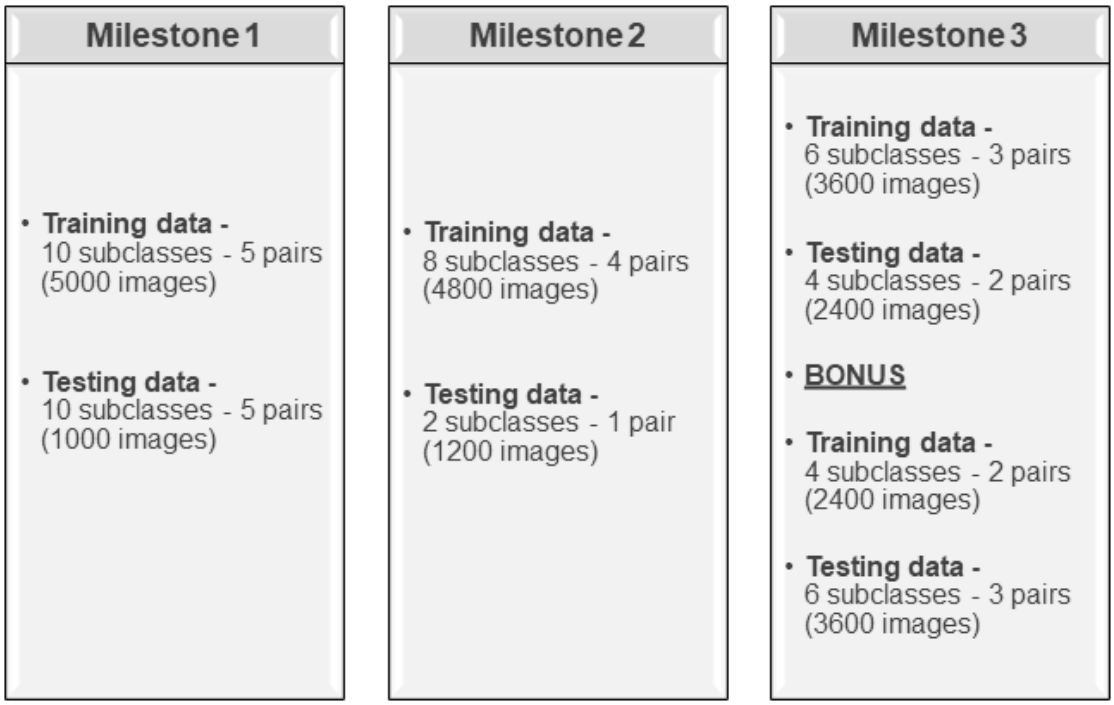


Figure 2 : Training and testing data proportion for each milestone

- For milestone 2, one subclass from each superclass was assigned exclusively as the testing set and the remaining data was assigned as the training set. Since we had already extracted the assigned superclasses (small mammals and medium-sized mammals), we used the

same filtered data to begin with Milestone 2.

- Thereafter, we generated **25 trials with each possible combination** of subclasses. For example, if we select Fox and squirrel as testing set, the remaining subclasses (in bold) were assigned as the training set.

  Medium-sized mammals: fox, **porcupine, possum, raccoon, skunk.**
  Small mammals: **hamster, mouse, rabbit, shrew,** squirrel.
- After a couple of attempts with a single combination, we realized the need to execute the algorithms for all the possible combinations. Finally, we created a user-defined function to splice and extract each combination with 1 subclass of small and medium mammals as testing set and remaining 8 subclasses as training set.
- Below are snippets from the code with the user-defined function and for loop that generated the 25 combinations. We used 'for' loops to rotate the subclasses for each combination, wherein we used the aforementioned user-defined function to extract the relevant data.
- The user-defined function was helpful as it can be called individually to summon any single combination of special interest. We aptly utilized it to separately run the algorithms and fine-tune hyperparameters for the combinations with the highest accuracy, as part of an extended experimentation.

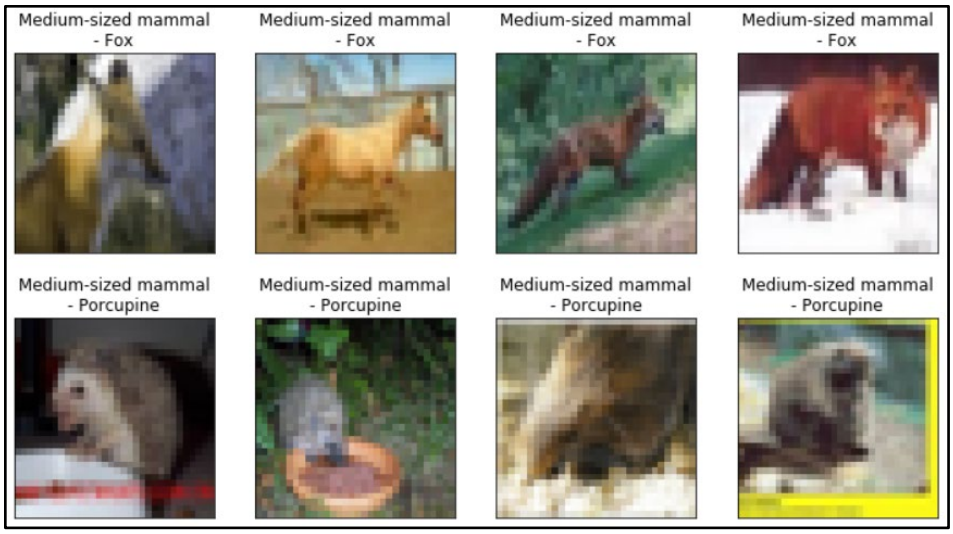
```
def train_test(small,medium): # test class for small and med
  small_mammals = ['hamster', 'mouse', 'rabbit', 'shrew', 'squirrel']
  small_mammals.remove(small)
  medium_sized_mammals = ['fox', 'porcupine', 'possum', 'raccoon', 'skunk']
  medium_sized_mammals.remove(medium)
```

```
# For training set
medium_ind = [ fine_label.index(x) for x in medium_sized_mammals ]
small_ind = [ fine_label.index(x) for x in small_mammals ]
target_ind = medium_ind + small_ind

#print ("Training set-\nNew index of Medium-sized mammals:", medium_ind,"\nNew index of Small mammals:", small_ind)

# For testing data -
medium_ind_2 = [ fine_label.index(x) for x in [medium] ]
small_ind_2 = [ fine_label.index(x) for x in [small] ]
target_ind_2 = medium_ind_2 + small_ind_2
#print ("\nTesting set-\nNew index of Medium-sized mammals:", medium_ind_2,"\nNew index of Small mammals:", small_ind_2)

# Splice the dataset to extract the relevant portion of data
train_slice3 = np.array([ idx for idx, y in enumerate(y) if y[0] in target_ind])
y_train3 = y[train_slice3]
x_train3 = x[train_slice3]

# Test set
test_slice3 = np.array([ idx for idx, y in enumerate(y) if y[0] in target_ind_2])
y_test3 = y[test_slice3]
x_test3 = x[test_slice3]

print ("Training set:", np.unique(y_train3))
print ("\nTesting set:", np.unique(y_test3))
#  Binary as we are working with two superclasses labels (or coarse labels)
y_train_bin3 = np.array([[int(y[0] in medium_ind)] for y in y_train3])
y_test_bin3 = np.array([[int(y[0] in medium_ind_2)] for y in y_test3])
y_bin3 = np.concatenate((y_train_bin3,y_test_bin3)) # for two superclass

return x_train3,x_test3,y_train_bin3,y_test_bin3
```

- The fine labels were defined and a blank list was initialized for the target and feature value of training and testing sets. Thereafter, the formerly defined function was used to splice the dataset and append to the initialized lists. The output indexes confirmed that relevant data was selected.

```
[ ]  # create list for train and test
     small_mammals = ['hamster', 'mouse', 'rabbit', 'shrew', 'squirrel']
     medium_sized_mammals = ['fox', 'porcupine', 'possum', 'raccoon', 'skunk']
     x_train3=list()
     x_test3=list()
     y_train_bin3=list()
     y_test_bin3=list()
     test_list=list()
     for i in range(0,5):
       for j in range(0,5):
         small=small_mammals[i]
         medium=medium_sized_mammals[j]
         x_train_temp,x_test_temp,y_train_temp,y_test_temp=train_test(small,medium)
         x_train3.append(x_train_temp)
         x_test3.append(x_test_temp)
         y_train_bin3.append(y_train_temp)
         y_test_bin3.append(y_test_temp)
         test_list.append([small,medium])
```

- Similarly, for milestone 3, we created a loop to test all possible combinations of the dataset using a similar ‘for’ loop and an user-defined function. Figure 2 succinctly explains the data proportions for milestone 3. The initial proportion was 3:2 (subclass pairs) for training and testing data. For bonus questions, we reversed the proportions; i.e.; a proportion of 2:3 (subclass pairs) in training and testing sets. The two sets are exclusive with no overlap of data.
- Visualization was an essential step towards validation of the filtered data. We created an user-defined function to print 4 random images from each subclass using ***matplotlib***. As instructed, the title included the verbal superclass name (aka coarse label) and subclass name (aka fine label as per the dataset terminology).



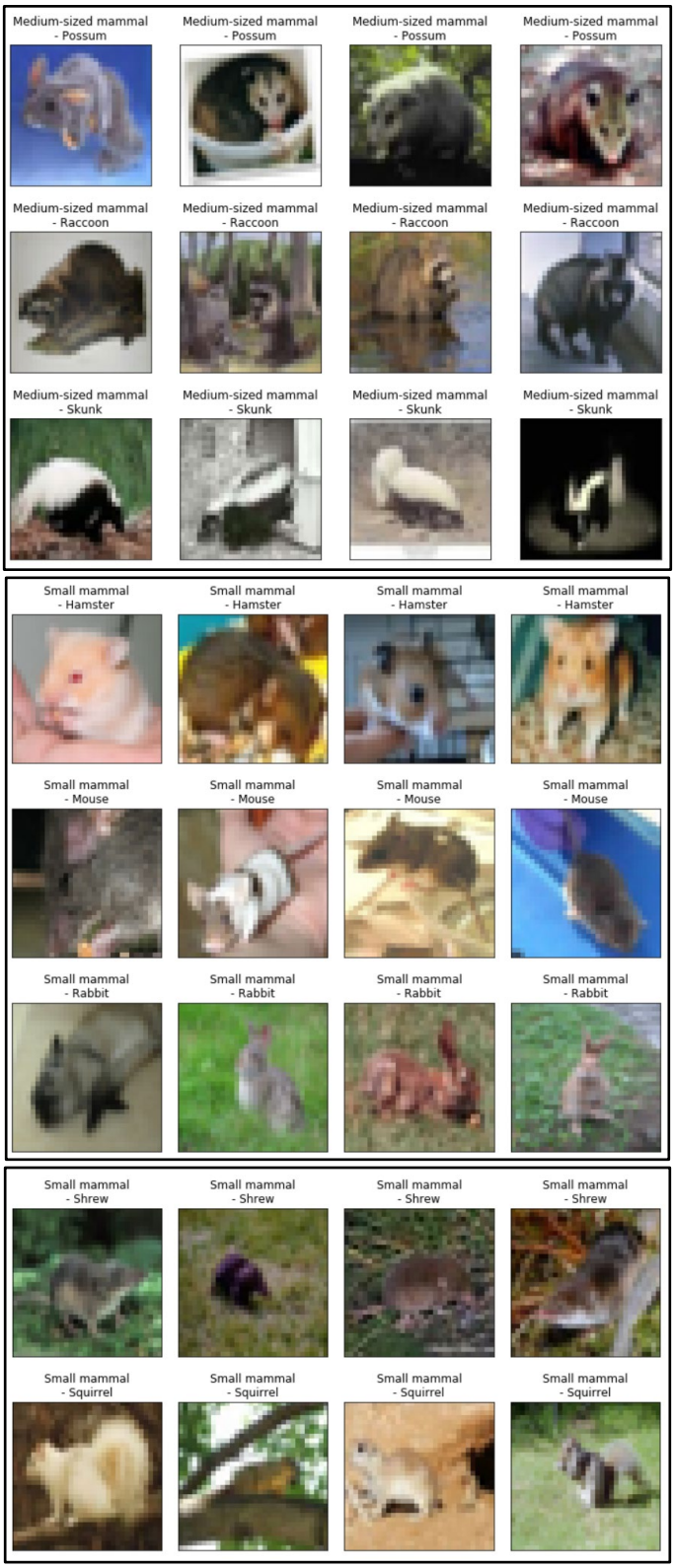


Figure 3 : Image validation

- Once the dataset was verified, validated and deemed ready, we proceeded ahead with the algorithms.

Milestone 1 - Prediction on randomly selected testing images

As a part of this project, we tried predicting randomly selected testing images. This segment of the project is considered as the Milestone 1. From the prepared data where 80% of data is selected as training data and 20% as testing data, we used several Machine Learning algorithms to train the model with training data, eventually using that to predict the testing data. In order to gain valuable insights of the data, we decided to try several simple algorithms before proceeding towards more complex algorithms. As the first step we tried a simple algorithm - Decision Tree Classifier.

Decision Tree Classifier:

A decision tree model is the best choice for handling tabular data with categorical or numerical features with less than hundreds of categories. Decision trees can understand non-linear interactions between features and target.

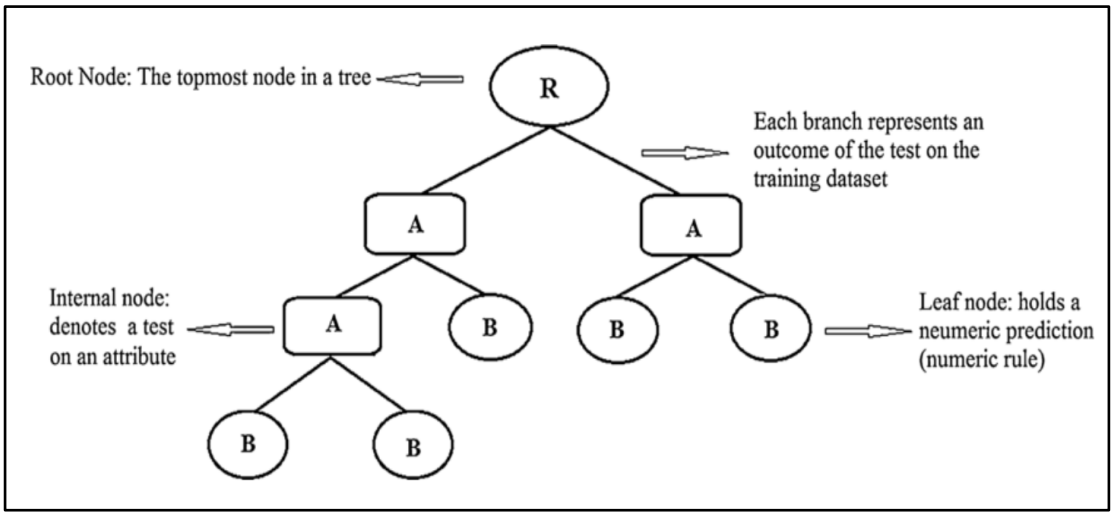


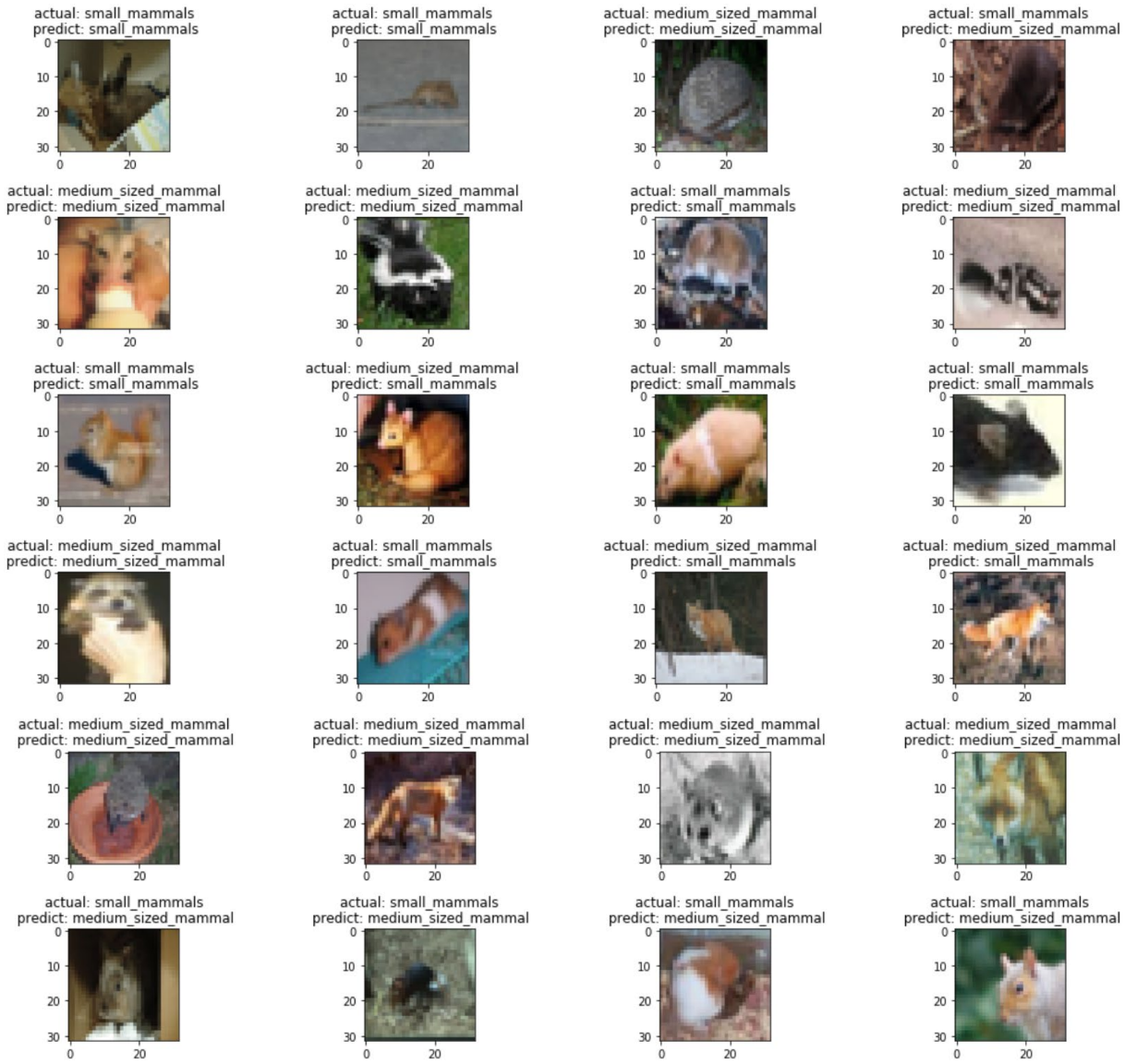
Figure 4 : Decision Tree architecture diagram.

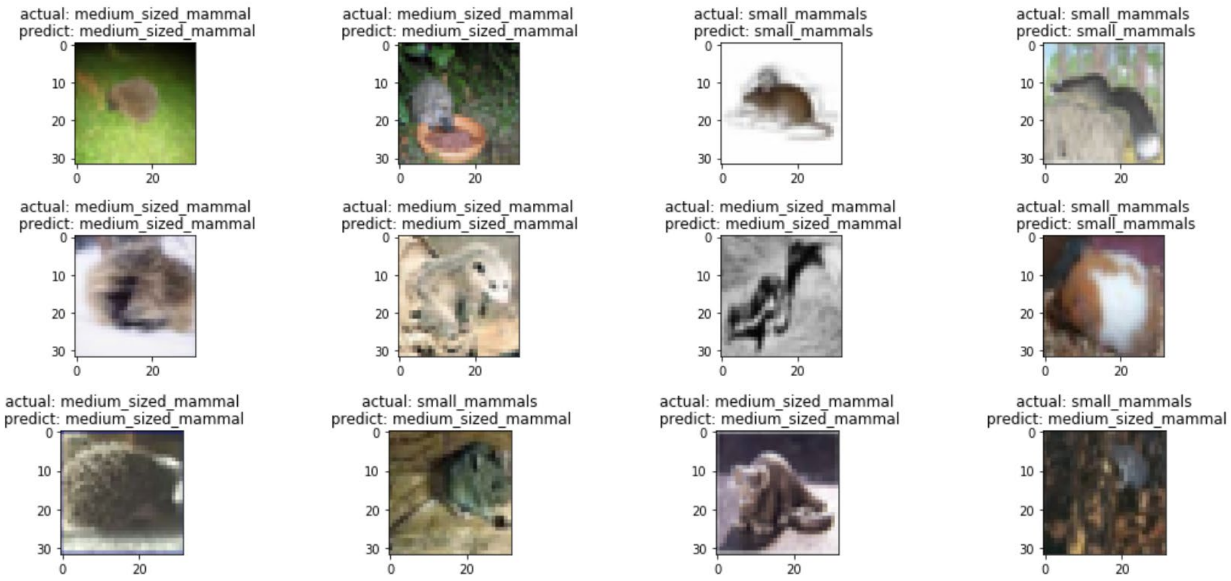
- Data was reshaped, by changing the train to test data ratio to 80/20.
- The model was defined and built, then fit using training data. The prediction was generated using ***.predict()*** and performance was evaluated by finding the score. The accuracy score for Decision Tree was found to be **57.49%**.

```
start = time.time()
tree = DecisionTreeClassifier()
tree.fit(x_train_1, y_train_bin)
tree_pred = tree.predict(x_test_1)
print ("DecisionTree Accuracy: {}%".format(tree.score(x_test_1, y_test_bin)*100))
end = time.time()
print(end - start)
```

```
DecisionTree Accuracy: 57.499999999999993%
15.552714824676514
```

We proceeded to print out **36 random images** with original and predicted labels using our algorithm. The results were as follows :

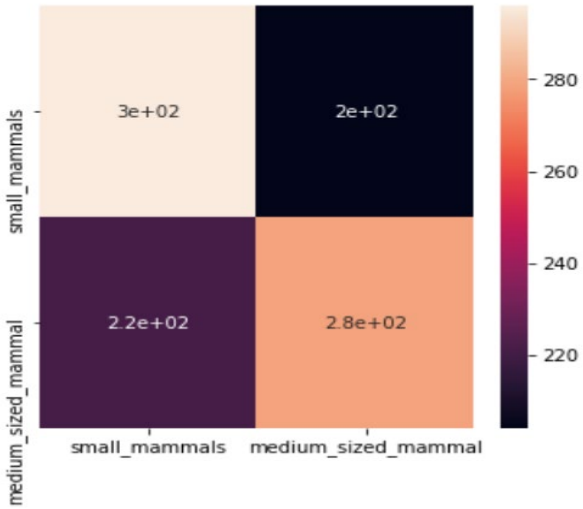




Model Evaluation :

From the result of the confusion matrix and classification report, the number of observations of the labeled classes is balanced, and F1 score of the small_mammals prediction is slightly higher than the medium_sized_mammals, and the heatmap of the confusion matrix also indicates the prediction of the small_mammals is better than medium_sized_mammals.

Confusion Matrix :



Classification Report :

	precision	recall	f1-score	support
0	0.57	0.59	0.58	500
1	0.58	0.56	0.57	500
micro avg	0.57	0.57	0.57	1000
macro avg	0.58	0.57	0.57	1000
weighted avg	0.58	0.57	0.57	1000

Extra Decision Tree Classifier:

We tried Extra Decision Tree classifier as the next step. Extra trees are also known as 'Extremely randomized trees' classifier.

- Data was reshaped, by changing the train to test data ratio to 5:1, and use the 5 fold split to do the cross-validation.
- The model was defined and built, then fit using training data. We also use the gridsearch to find out the best parameter. The prediction was generated using ***.predict()*** and performance was evaluated by finding the score. The accuracy score for extra decision tree was found to be **61.8%**, which is a better score than decision trees.

```
#using the default parameter
start = time.time()
Extratree = ExtraTreesClassifier()
Extratree.fit(x_train_1, y_train_bin)
Extratree_pred = Extratree.predict(x_test_1)
print ("ExtraTree Accuracy: {}%".format(Extratree.score(x_test_1, y_test_bin)*100))
end = time.time()
print(end - start)
```

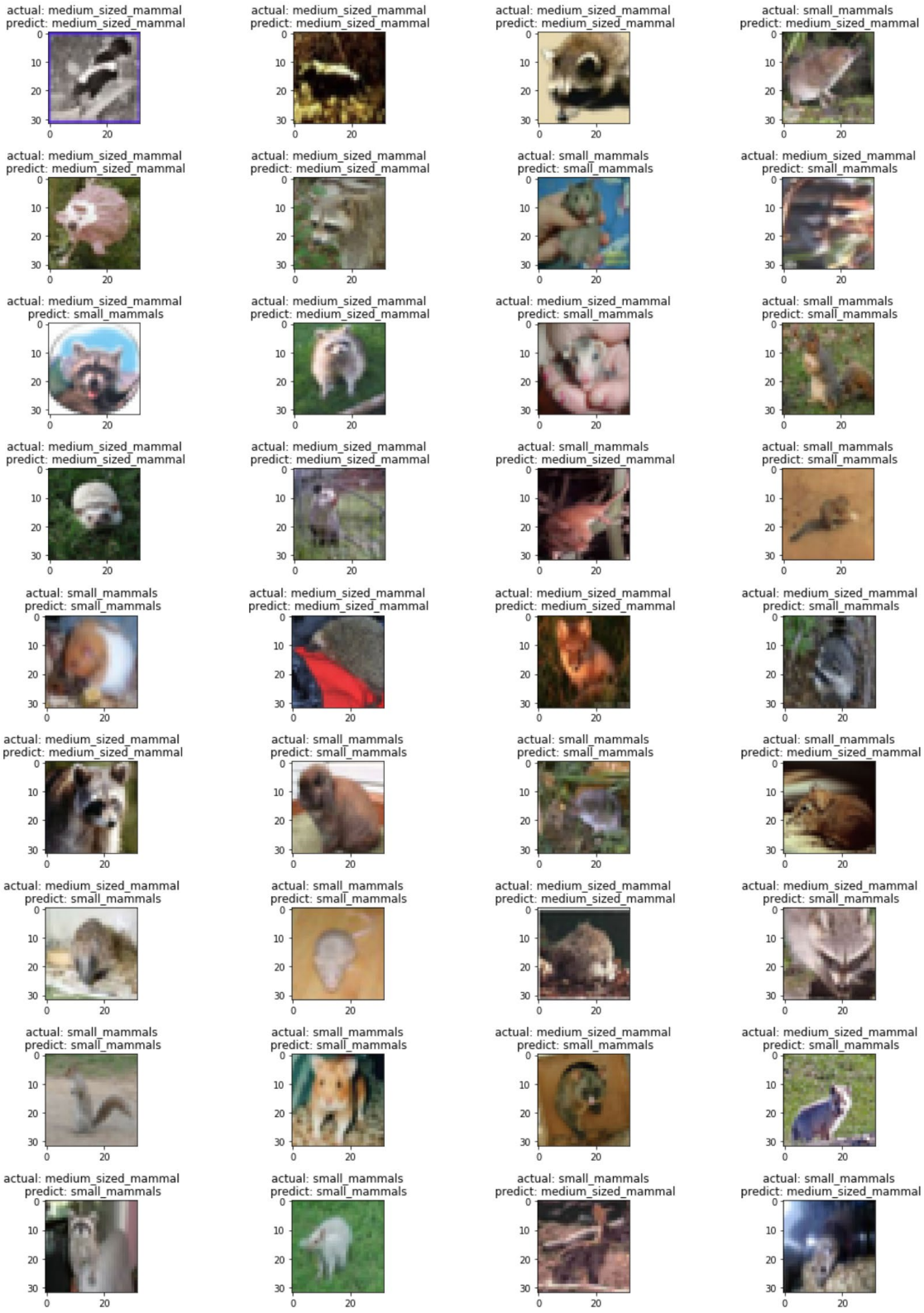
```
ExtraTree Accuracy: 61.8%
0.8412554264068604
```

- We could find that the prediction score on testing data with best estimator was found to be 67.0% with 'max_depth': 30, 'n_estimators': 500.

```
print ('prediction score on testing data with the best estimator: %.2f%%' % (Extratree.best_estimator_.score(x_test_1,y_test_bin)*100))
```

```
prediction score on testing data with the best estimator: 67.00%
```

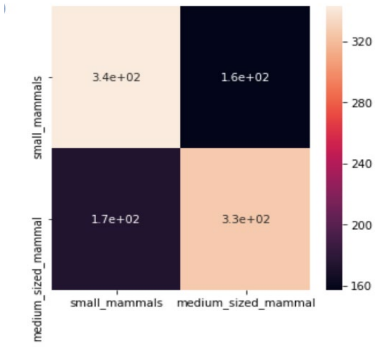
- We proceeded to print out **36 random images** with original and predicted labels using our algorithm. The results were as follows :


actual: medium_sized_mammal
predict: medium_sized_mammal
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: small_mammals
predict: medium_sized_mammal
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: small_mammals
predict: small_mammals
actual: medium_sized_mammal
predict: small_mammals
actual: medium_sized_mammal
predict: small_mammals
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: medium_sized_mammal
predict: small_mammals
actual: small_mammals
predict: small_mammals
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: small_mammals
predict: medium_sized_mammal
actual: small_mammals
predict: small_mammals
actual: small_mammals
predict: small_mammals
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: medium_sized_mammal
predict: small_mammals
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: small_mammals
predict: small_mammals
actual: small_mammals
predict: small_mammals
actual: small_mammals
predict: medium_sized_mammal
actual: medium_sized_mammal
predict: small_mammals
actual: small_mammals
predict: small_mammals
actual: small_mammals
predict: small_mammals
actual: medium_sized_mammal
predict: small_mammals
actual: small_mammals
predict: small_mammals
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: medium_sized_mammal
predict: small_mammals
actual: small_mammals
predict: small_mammals
actual: small_mammals
predict: small_mammals
actual: medium_sized_mammal
predict: small_mammals
actual: small_mammals
predict: medium_sized_mammal
actual: small_mammals
predict: medium_sized_mammal

Model Evaluation :

From the result of the confusion matrix and classification report, the number of observations of the labeled classes is balanced, and F1 score of the small_mammals prediction is slightly higher than the medium_sized_mammals, and the heatmap of the confusion matrix also indicates the prediction of the small_mammals is better than medium_sized_mammals.

Confusion Matrix :



Classification Report :

	precision	recall	f1-score	support
0	0.66	0.69	0.68	500
1	0.68	0.65	0.66	500
micro avg	0.67	0.67	0.67	1000
macro avg	0.67	0.67	0.67	1000
weighted avg	0.67	0.67	0.67	1000

SVM with ‘RBF’ Classifier:

We tried **Support Vector Machine (SVM)** is a supervised machine learning algorithm used mostly in classification problems. A Support Vector Machine algorithm outputs a hyperplane by categorizing examples. A hyperplane can be described as a line which divides a plane into two sections, with the two classes are in either side of the line. Separation of classes is a major property of SVMs. Radial Basis Function(RBF) is a kernel which is commonly used with SVM. RBF kernels are mostly used in image classification problems.

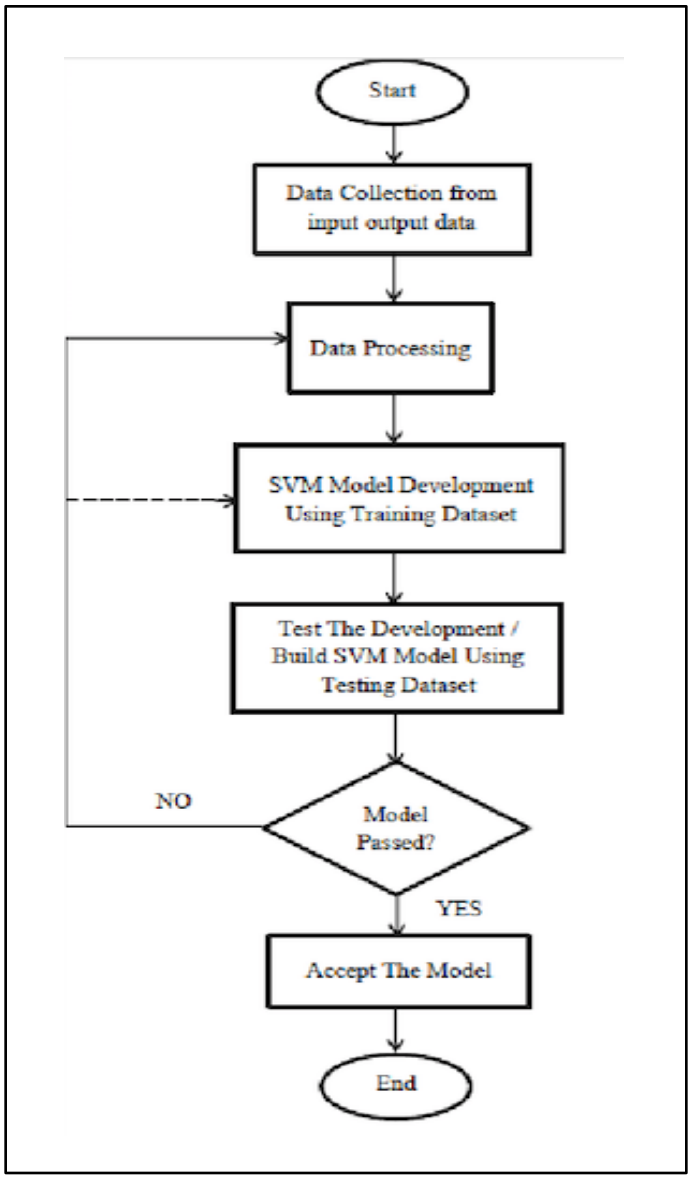


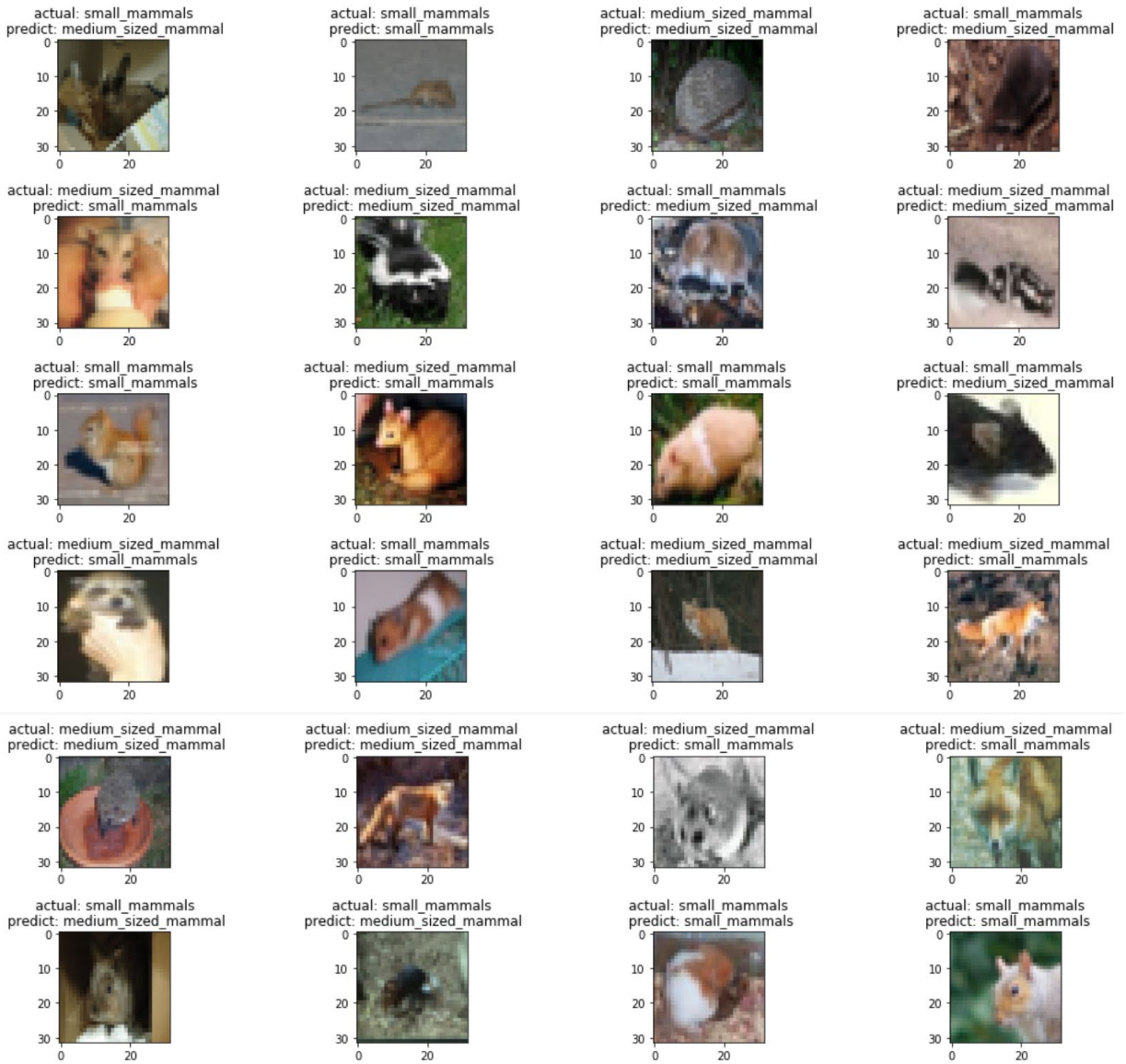
Figure 5 : Workflow diagram of SVM Classifier

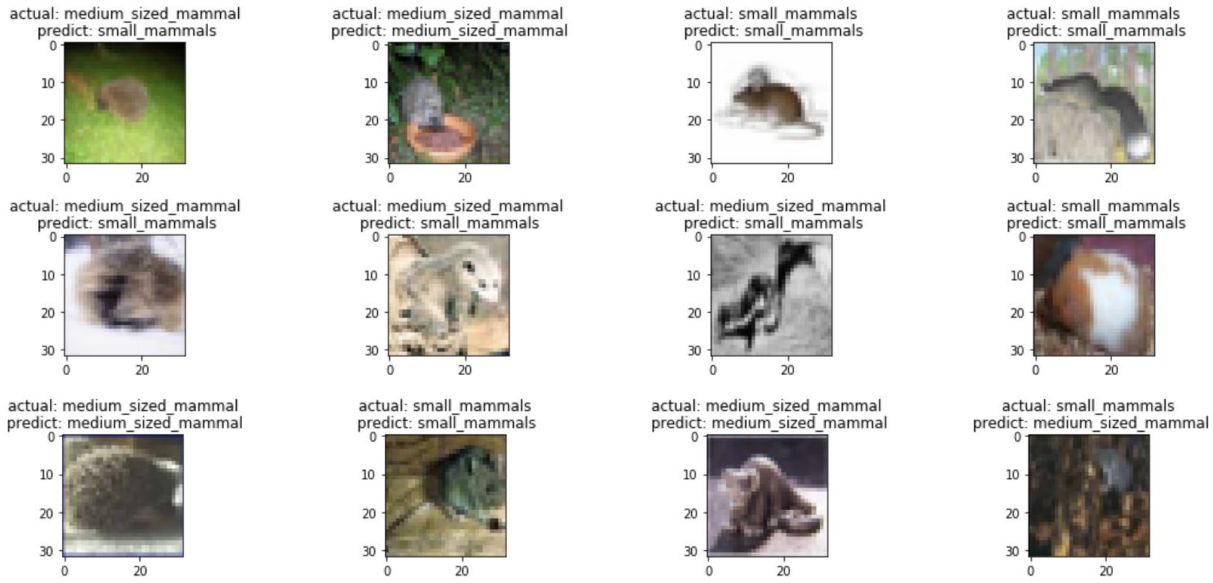
- Data was reshaped, by changing the train to test data ratio to 5:1. and use the 5 fold split to do the cross-validation.
- The model was defined and built, then fit using training data. Prediction was generated using ***.predict()*** and performance was evaluated by finding the score.
- We could generate the accuracy as **63.2%** with default parameter, and 63.5% with C= 10.

```
start = time.time()
rbf = SVC(kernel='rbf', C=5)
rbf.fit(x_train_1, y_train_bin)
rbf_pred=rbf.predict(x_test_1)
print ("SVM with rbf kernel Accuracy: {}%".format(rbf.score(x_test_1, y_test_bin)*100))
end = time.time()
print('{} seconds'.format(end - start))
```

```
SVM with rbf kernel Accuracy: 63.2%
157.40500020980835 seconds
```

We printed out **36 random images** with original and predicted labels using our algorithm. The results were as follows :

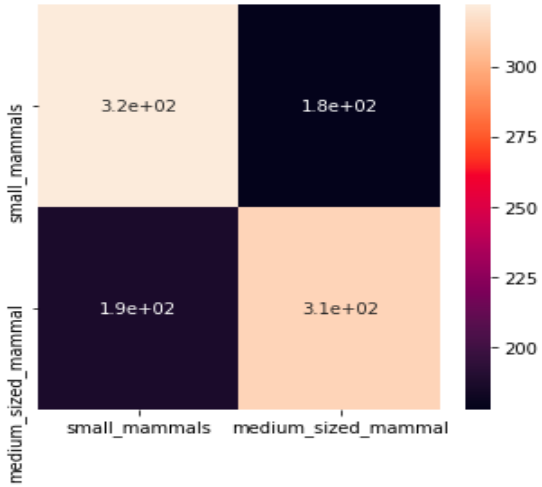




Model Evaluation :

From the confusion matrix and classification report (0: small mammals, 1: medium_sized mammals), the number of observations of labeled classes is balanced and F1 score of small_mammals prediction is better than medium_sized_mammals. The heatmap of the confusion matrix also indicates that the prediction of small_mammals is better than medium_sized_mammals.

Confusion Matrix :



Classification Report :

	precision	recall	f1-score	support
0	0.63	0.64	0.64	500
1	0.64	0.63	0.63	500
micro avg	0.64	0.64	0.64	1000
macro avg	0.64	0.64	0.63	1000
weighted avg	0.64	0.64	0.63	1000

After trying multiple basic Machine Learning algorithms, we moved on to more complex ML algorithms such as Neural Networks, since this is an image classification problem.

Neural Network for Image Recognition:

Neural networks mimic the functioning of human brain - by taking in raw images and processing the output directly. Neural networks can classify texts, images, videos, etc. Neural networks are a collection of interconnected ‘neurons’, where each neuron ingests an input data and applies a computation called the ‘activation function’ to produce a result. Each neuron is provided with a numerical weight by which the result is affected. This result is input to more neural layers until the neural network generates a prediction. This is repeated for all the images and thus the network learns which weights are generating best and accurate predictions, and the process is called ‘back propagation’. After the model is trained, a test set of images are fed to the model to test the accuracy of the model. This model can then be used for image classification.

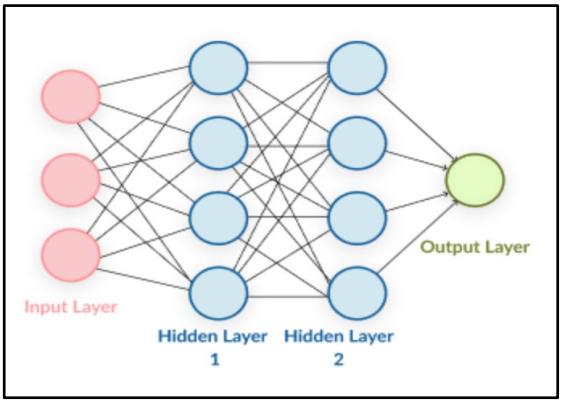


Figure 6 : Neural Network architecture diagram

- Data setup was done by separating the data into two groups - predictors and targets.
- The target data is one-hot encoded, where the categorical variables are converted into binary form so that machine learning algorithms can do a better job at predicting them. For this, ***to_categorical()*** function was used.
- While working with color images, they should be flattened from RGB format(6000,

32, 32, 3) to (6000, 30172) . This was achieved by using ***x.reshape(x.shape[0], -1)***. (Berhane, 2016)

- The dataset was spliced to filter the assigned superclass pair.
- The data was validated by visualizing four random images from each sub-class. (Kinli, 2018)
- In order to setup the neural network model, first the model was defined using ***model = Sequential()***.
- Then we created an architecture by adding input layers, where input_shape is big enough to represent each pixel(32x32). The activation function was chosen to be 'relu' due to its flexibility. The number of neurons was determined through experimentation.

The code used was: ***model.add(Dense(500,activation="relu",input_shape=(3072,)))***

- Hidden layers were added using # of layers = # of samples / (alfa *(# neurons of outputs + # neurons of inputs)), where alpha is a arbitrary scaling factor, 2 to 10. (codename, 2018)
- Overfitting issues were dealt with by adding a dropout layer. (Tutorial, 2018)
- Model Compilation was done by defining the optimizer, which was chosen to be "adam", since it adapts its learning rate throughout the process and has a very flexible application. The learning rate was changed in order to avoid big jumps. (Brownlee, 2017)
- The loss function was defined as categorical_crossentropy. The code used was :

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

- Fitting the model was done as the next step. An early stop was set up, so that the computer stops computing in case the model is no longer evolving.
- Model was fit, and we used 30% test and 70% train split for validation and number of epochs were defined. The code used was :

hist= model.fit(x_flatten, y_target,validation_split=0.3, epochs=20, callbacks=[early_stop])

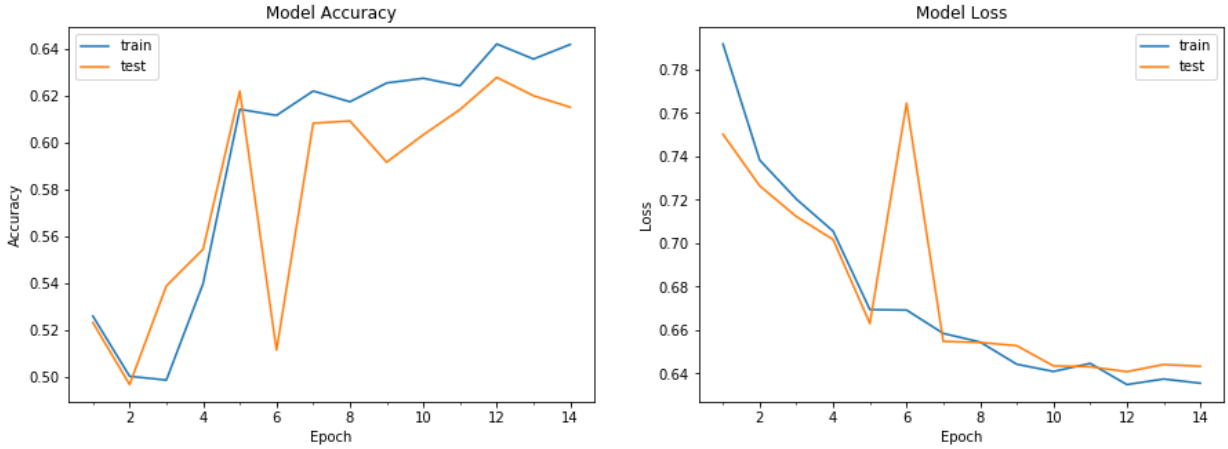
Model Evaluation :

On comparing the predictions with the actual data, the score was found to be **61.51%**.

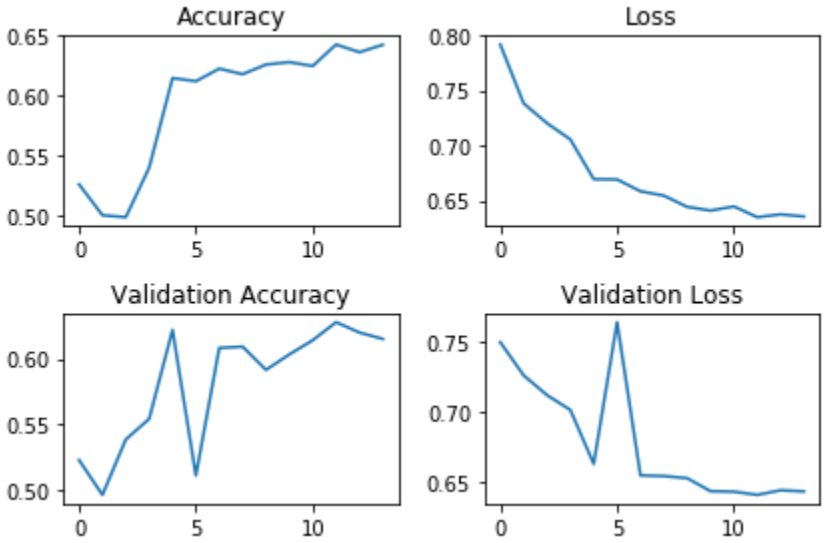
```
scores = model.evaluate(X_flatten_test, y_target_test, verbose=0)
print("NN score: %.2f%%" % (scores[1]*100))
```

```
NN score: 61.51%
```

The results were plotted using ***matplotlib***. As the graphics show there is a lot of instability. Reducing the learning rate didn't have a positive effect leading the mode to overfit. Dropout layers and kernel_regularizer were tried to avoid overfitting, with relative success.



The model accuracy and loss were also plotted using ***matplotlib***.



To avoid long training cycles, once we found a model we were comfortable with, we saved it using ***model.save()***. Given that we couldn't get the score above 62% without overfitting the model, we don't recommend using a simple feedforward neural network.

Convolutional Neural Network for image recognition:

After doing the basic neural network model, we tried Convolutional neural networks (CNN), since they are specifically designed for image classification problems and are scalable for usage with large datasets. CNN extracts features from images and there is no need for feature engineering. CNN converts the extracted features into lower dimension without

losing its characteristics. Also, CNN gives better accuracy for image classification problems. A CNN has five layers as seen from the below diagram.

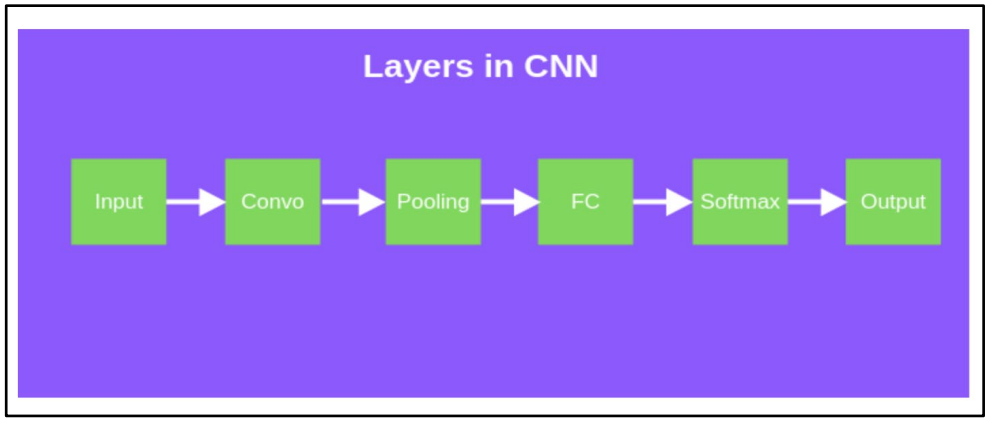


Figure 7 : Layers in CNN

- Data was prepared by splitting into training and testing data.
- A CNN involves four blocks which are a convolution layer, pooling layer, flattening layer and an output layer. We initialized the CNN using the ***.sequential()*** function.
- After that, convolution, pooling, flattening and model compiling was done using the following code:

```
]
  model_cnn1 = Sequential()

  model_cnn1.add(Conv2D(32, 3, 3, border_mode='same', input_shape=x_train.shape[1:], activation='relu', ))

  model_cnn1.add(MaxPooling2D(pool_size=(2, 2)))
  #model.add(Dropout(0.5))

  model_cnn1.add(Conv2D(128, 3, 3, border_mode='same', activation='relu'))
  model_cnn1.add(MaxPooling2D(pool_size=(2, 2)))
  model_cnn1.add(Dropout(0.5))
  #model.add(Conv2D(128, 3, 3, border_mode='same', activation='relu'))
  #model.add(MaxPooling2D(pool_size=(2, 2)))

  model_cnn1.add(Flatten())
  model_cnn1.add(Dense(256, activation='relu'))
  model_cnn1.add(Dropout(0.5))
  model_cnn1.add(Dense(2, activation='softmax'))
  opt = Adam(lr=0.0004, decay=1e-6)
  model_cnn1.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

- After that, image data generator class was initialized, for data augmentation to reduce overfitting on models by increasing the amount of training data.

```
print('Using real-time data augmentation.')
datagen = ImageDataGenerator(
featurewise_center=False,
samplewise_center=False,
featurewise_std_normalization=False,
samplewise_std_normalization=False,
zca_whitening=False,
rotation_range=0,
width_shift_range=0,
height_shift_range=0,
horizontal_flip=True,
vertical_flip=False)

datagen.fit(x_train)
```

- Now that the model is created, it is time to train the model. Better accuracy can be obtained by increasing the number of epochs.

```
epochs = 10
batch_size=32
early_stop=EarlyStopping(patience=2)
hist = model_cnn1.fit_generator(datagen.flow(x_train, y_train_c,batch_size=batch_size,shuffle=True),
  steps_per_epoch=x_train.shape[0] // batch_size,
  epochs=epochs,
  validation_data=(x_test, y_test_c),
  workers=4,
  callbacks=[early_stop])
```

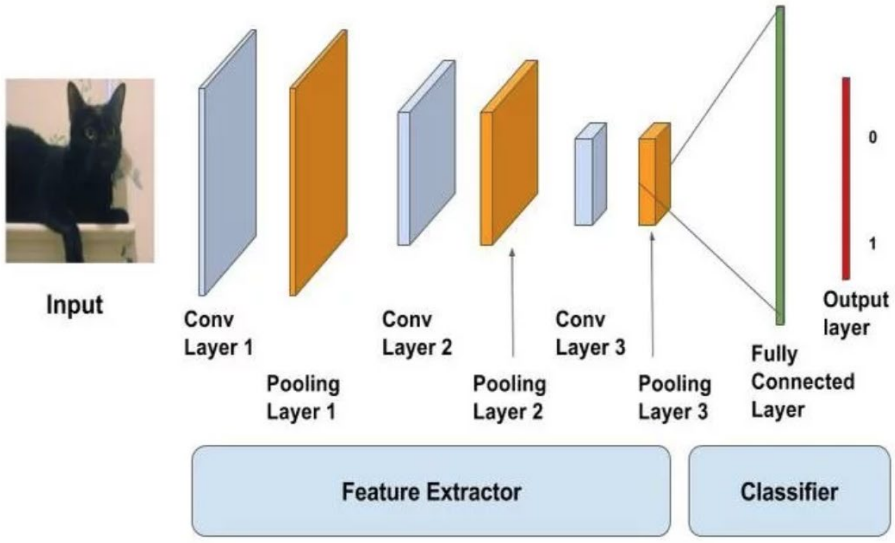


Figure 8 : CNN Model architecture

Model Evaluation :

The scores were evaluated using model.evaluate() function, and the score was found to be **74.10%**.

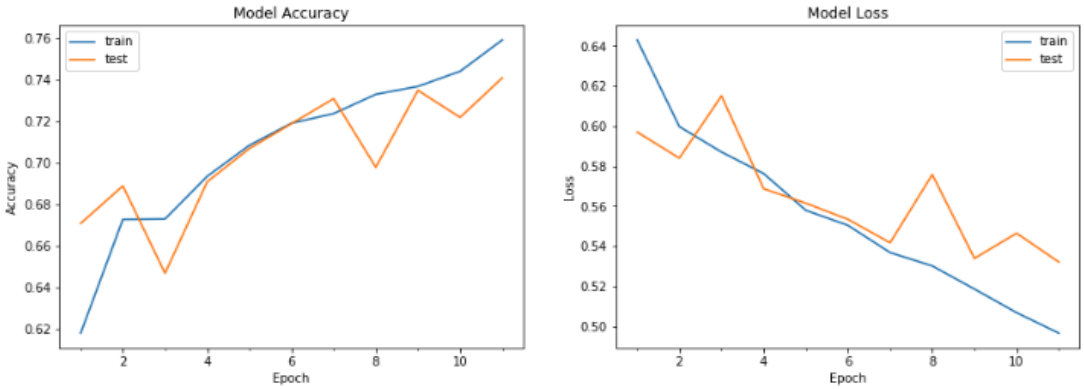
```
scores = model_cnn1.evaluate(x_test, y_test_c, verbose=0)
print("CNN score: %.2f%%" % (scores[1]*100))
```

```
CNN score: 74.10%
```

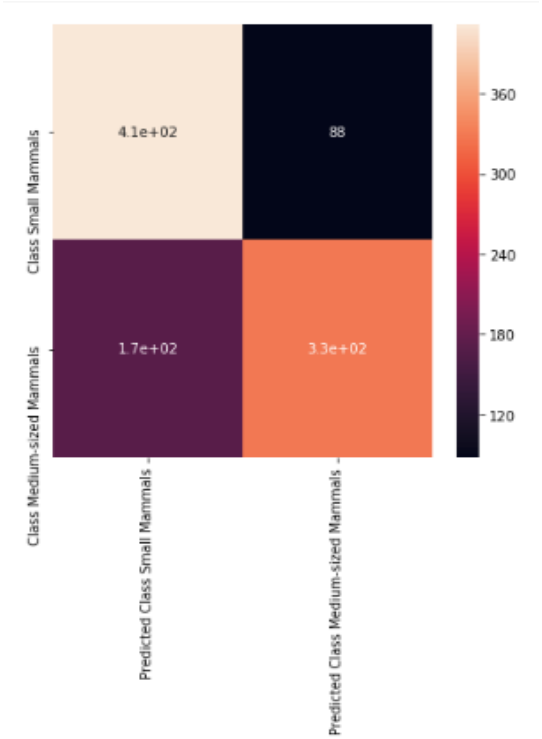
```
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

```
Test loss: 0.5321442775726318
Test accuracy: 0.741
```

The model accuracy and loss were plotted using ***matplotlib.***



Confusion Matrix :



Classification Report :

	precision	recall	f1-score	support
Small Mammals	0.71	0.82	0.76	500
Medium-sized Mammals	0.79	0.66	0.72	500
micro avg	0.74	0.74	0.74	1000
macro avg	0.75	0.74	0.74	1000
weighted avg	0.75	0.74	0.74	1000

The F1-score of small_mammals is slightly bigger than that of medium_sized_mammals. From the F1-score, we can see that small_mammals are more positively predicted by the model than the medium-sized mammals. From the precision, we can see that medium_sized_mammals were predicted more precisely by this model.

In this milestone, we used multiple algorithms to train models to recognize small sized mammals and medium sized mammals. During the project, we understood the importance of evaluating our model. By comparing training loss and testing loss, we can adjust the model to avoid overfitting. By looking at the confusion matrix and confusion report, we can infer the cause of our errors, and fine tune our model accordingly. The most important thing for a machine learning/deep learning model is not just a high score, but also robustness so it can predict for different cases. For the milestone one (image classification for super classes: small and medium mammals) we selected **convolutional neural network** as the most promising, since it delivered a good score (74.10%) spending a reasonable amount of time processing (80.53s).

Milestone 2 - Prediction on one testing subclass from each of the two superclasses

After careful analysis of Milestone 1 results, we selected the ML algorithms which showed better performance to do the Milestone 2. We used the **GridSearch** method to tune the parameters of the model and got the best parameters, which showed better accuracy. They are:

- Extra decision tree
- Random forest
- Gradient boosting
- SVM with ‘rbf’ kernel
- Bagging
- Logistic regression.

Analyzing the results of Milestone 1, we decided to test the algorithms which showed good results for Milestone 1, in Milestone 2 as well. As the first step, we tried Random Forest Classifier.

Random Forest :

Random Forest is an ensemble algorithm that creates a set of decision trees and combines them together to get an accurate, stable prediction.

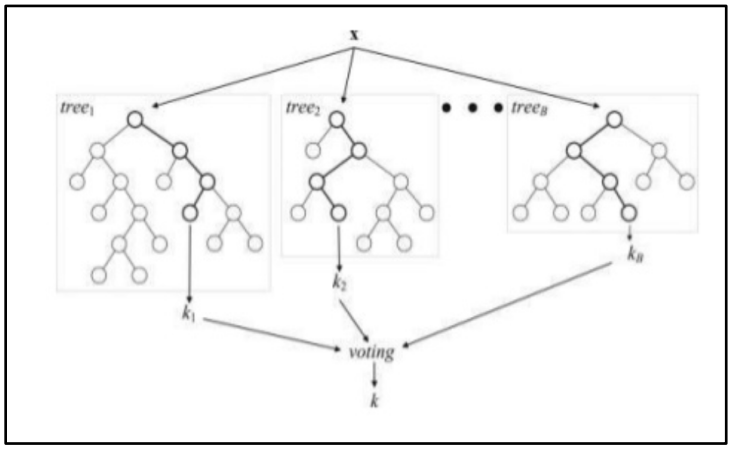


Figure 9 : Random forest Architecture

- The model was defined and built, then fit using training data. The prediction was generated using ***.predict()*** and performance was evaluated by finding the score.

```
# find the baseline performance for the random Forest
start = time.time()
logit = RandomForestClassifier()
logit.fit(x_train2, y_train_bin)
logit_pred=logit.predict(x_test2)
scores = cross_val_score(logit, x_train2,y_train_bin, cv=5)
print(scores)
print ("RandomForest Accuracy: {}%".format(logit.score(x_test2, y_test_bin)*100))
end = time.time()
print(end - start)
```

```
[0.64583333 0.66770833 0.66354167 0.66875    0.646875  ]
RandomForest Accuracy: 47.91666666666667%
9.257304430007935
```

The prediction score on testing data with best estimator was found to be 48.17%.

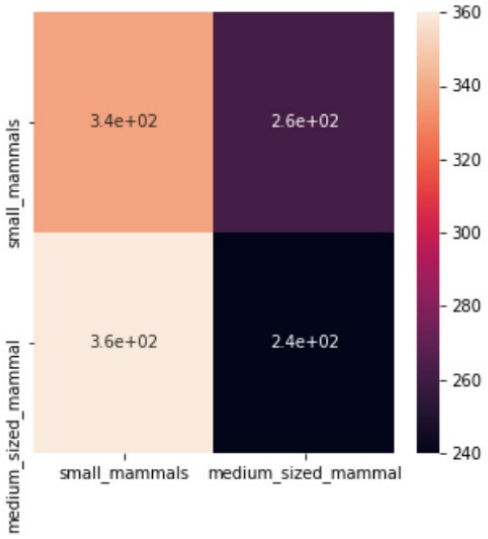
```
print ('prediction score on testing data with the best estimator: %.2f%%' % (logit.best_estimator_.score(x_test2,y_test_bin)*100))
```

```
prediction score on testing data with the best estimator: 48.17%
```

The predicted data can be seen here:

```
test_label = ['squirrel','fox']
logit_pred=logit.predict(x_test2)
indices = [np.random.choice(range(len(x_test2))) for j in range(8)]
#cifar_grid(x_test2, y_test_bin2, indices,4,logit_pred)
cifar_grid1(x_test2, y_test_bin, indices,4,logit_pred,test_label)
```

Confusion Matrix :



Classification Report :

	precision	recall	f1-score	support
0	0.48	0.56	0.52	600
1	0.48	0.40	0.44	600
micro avg	0.48	0.48	0.48	1200
macro avg	0.48	0.48	0.48	1200
weighted avg	0.48	0.48	0.48	1200

We moved on to the next method since this did not provide much accuracy. The next method we tried was Extra Decision Tree method.

Extra Decision Tree :

- The model was defined and built, then fit using training data. The prediction was generated using ***.predict()*** and performance was evaluated by finding the score.

```
# find the baseline performance
start = time.time()
Extratree = ExtraTreesClassifier()
Extratree.fit(x_train2,y_train_bin)
Extratree_pred=Extratree.predict(x_test2)
scores = cross_val_score(Extratree, x_train2, y_train_bin, cv=5)
print(scores)
print ("ExtraTree Accuracy: {}%".format(Extratree.score(x_test2, y_test_bin)*100))
end = time.time()
print(end - start)
```

```
[0.67395833 0.659375   0.66979167 0.67291667 0.62395833]
ExtraTree Accuracy: 49.666666666666664%
4.041752576828003
```

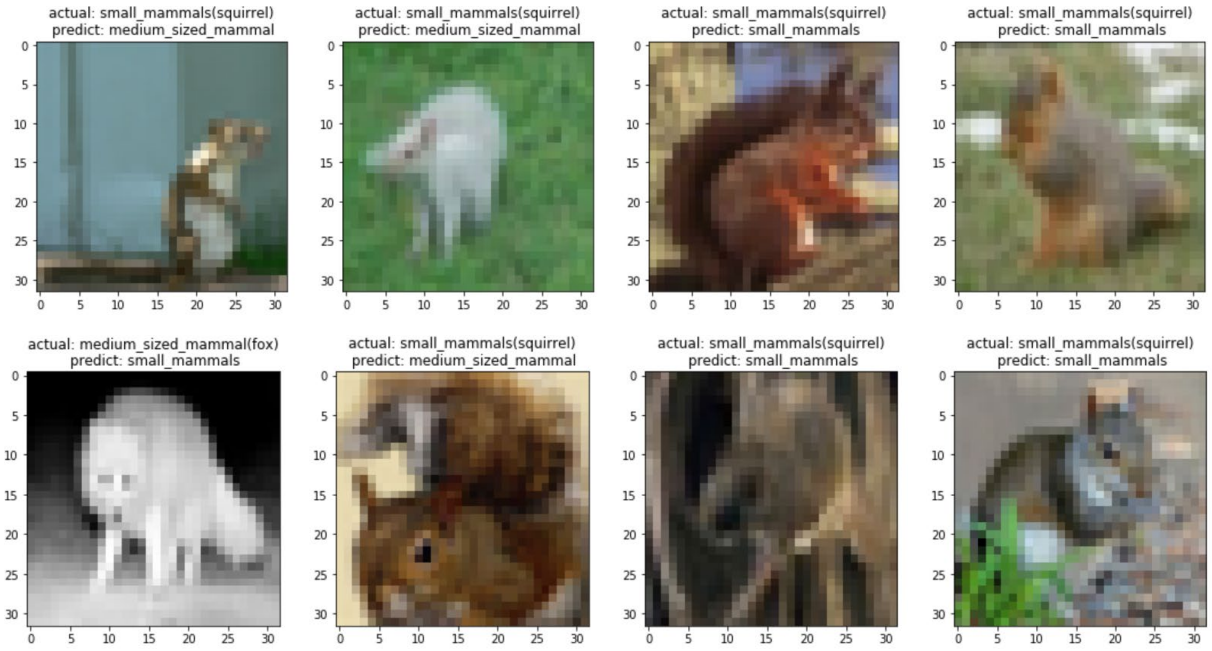
The prediction score on testing data with best estimator was found to be 48.50%.

```
print ('prediction score on testing data with the best estimator: %.2f%%' % (Extratree.best_estimator_.score(x_test2,y_test_bin)*100))
```

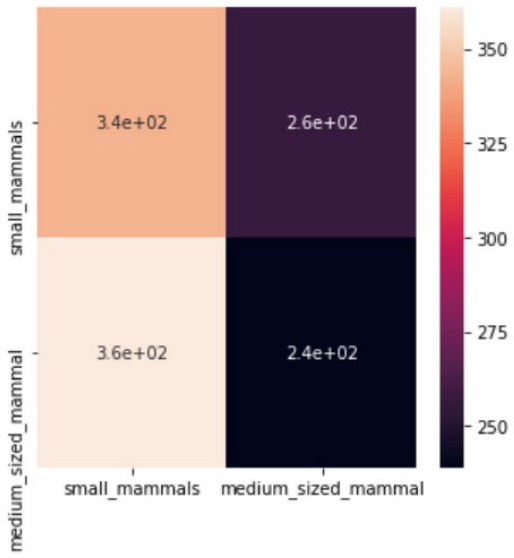
```
prediction score on testing data with the best estimator: 48.50%
```

```
test_label = ['squirrel','fox']

Extratree_pred=Extratree.predict(x_test2)
indices = [np.random.choice(range(len(x_test2))) for j in range(8)]
cifar_grid1(x_test2, y_test_bin, indices,4,Extratree_pred,test_label)
```

The predicted data can be seen here:



Confusion Matrix :



Classification Report :

	precision	recall	f1-score	support
0	0.49	0.57	0.53	600
1	0.48	0.40	0.44	600
micro avg	0.48	0.48	0.48	1200
macro avg	0.48	0.48	0.48	1200
weighted avg	0.48	0.48	0.48	1200

Since this was again not giving better results, we tried Gradient Boosting algorithms.

Gradient Boosting :

Gradient Boosting is an algorithm mainly used for regression and classification problems. Gradient Boosting has three elements which are optimizing the loss function, a weak learning to make predictions and another model to minimize loss function.

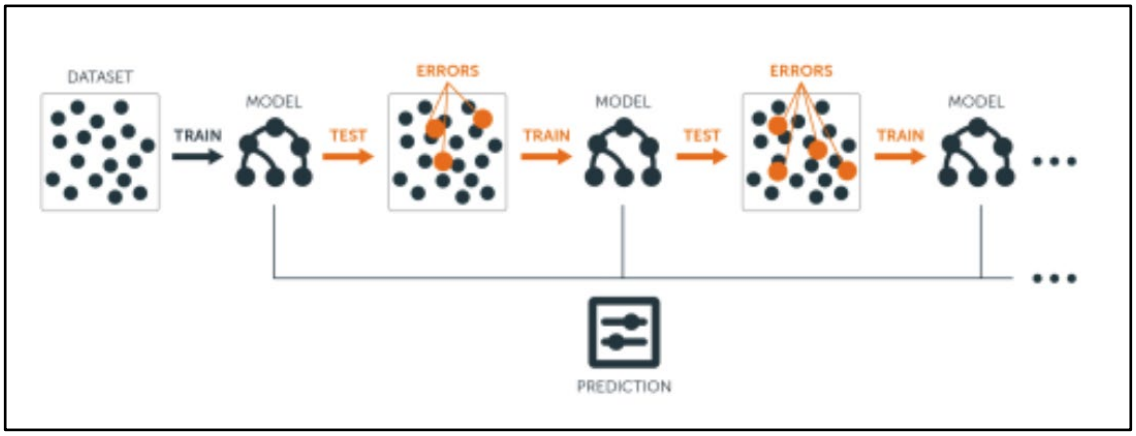


Figure 10 : Gradient Boosting workflow diagram

- The model was defined and built, then fit using training data. The prediction was generated using ***.predict()*** and performance was evaluated by finding the score.

```
# baseline performance
start = time.time()
Gradient = GradientBoostingClassifier()
Gradient.fit(x_train2, y_train_bin)
Gradient_pred=Gradient.predict(x_test2)
scores = cross_val_score(Gradient, x_train2, y_train_bin, cv=5)
print ("Gradient Boosting Accuracy: {}%".format(Gradient.score(x_test2, y_test_bin)*100))
end = time.time()
print(end - start)
```

```
Gradient Boosting Accuracy: 41.083333333333336%
767.4611730575562
```

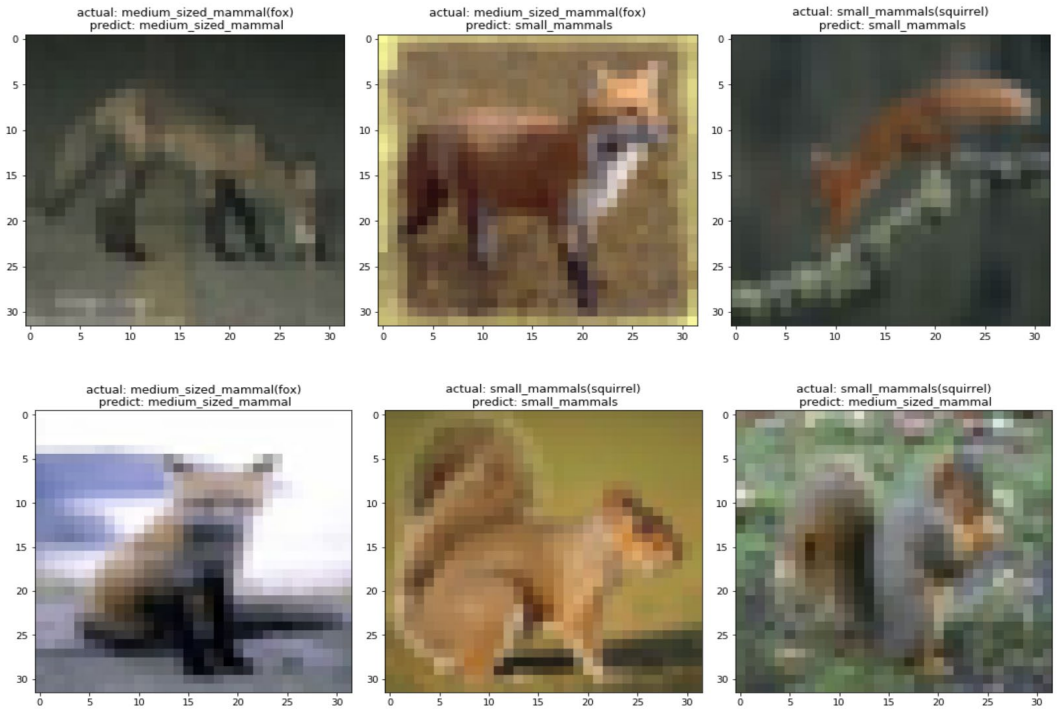
The prediction score on testing data with best estimator was found to be 47.00%.

```
print ('prediction score on testing data with the best estimator: %.2f%%' % (Gradient.best_estimator_.score(x_test2,y_test_bin)*100))
```

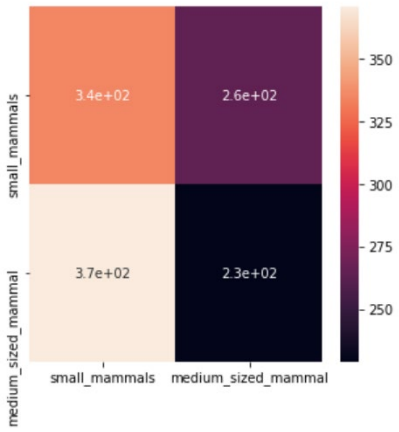
```
prediction score on testing data with the best estimator: 47.00%
```

The predicted data can be seen here:

```
test_label = ['squirrel','fox']
Gradient_pred=Gradient.predict(x_test2)
indices = [np.random.choice(range(len(x_test2))) for j in range(6)]
cifar_grid1(x_test2, y_test_bin,indices,3,Gradient_pred,test_label)
```



Confusion Matrix :



Classification Report :

	precision	recall	f1-score	support
0	0.47	0.56	0.51	600
1	0.46	0.38	0.42	600
micro avg	0.47	0.47	0.47	1200
macro avg	0.47	0.47	0.47	1200
weighted avg	0.47	0.47	0.47	1200

Due to the low accuracy score, we moved on to the next algorithm - SVM with 'rbf' kernel.

SVM with 'RBF' kernel :

- The model was defined and built, then fit using training data. The prediction was generated using ***.predict()*** and performance was evaluated by finding the score.

```
start = time.time()
rbf = SVC(kernel='rbf')
rbf.fit(x_train2, y_train_bin)
rbf_pred=rbf.predict(x_test2)
print ("SVM - rbf Accuracy: {}%".format(rbf.score(x_test2, y_test_bin)*100))
end = time.time()
print('{} seconds'.format(end - start))
```

```
SVM - rbf Accuracy: 44.5%
303.22016644477844 seconds
```

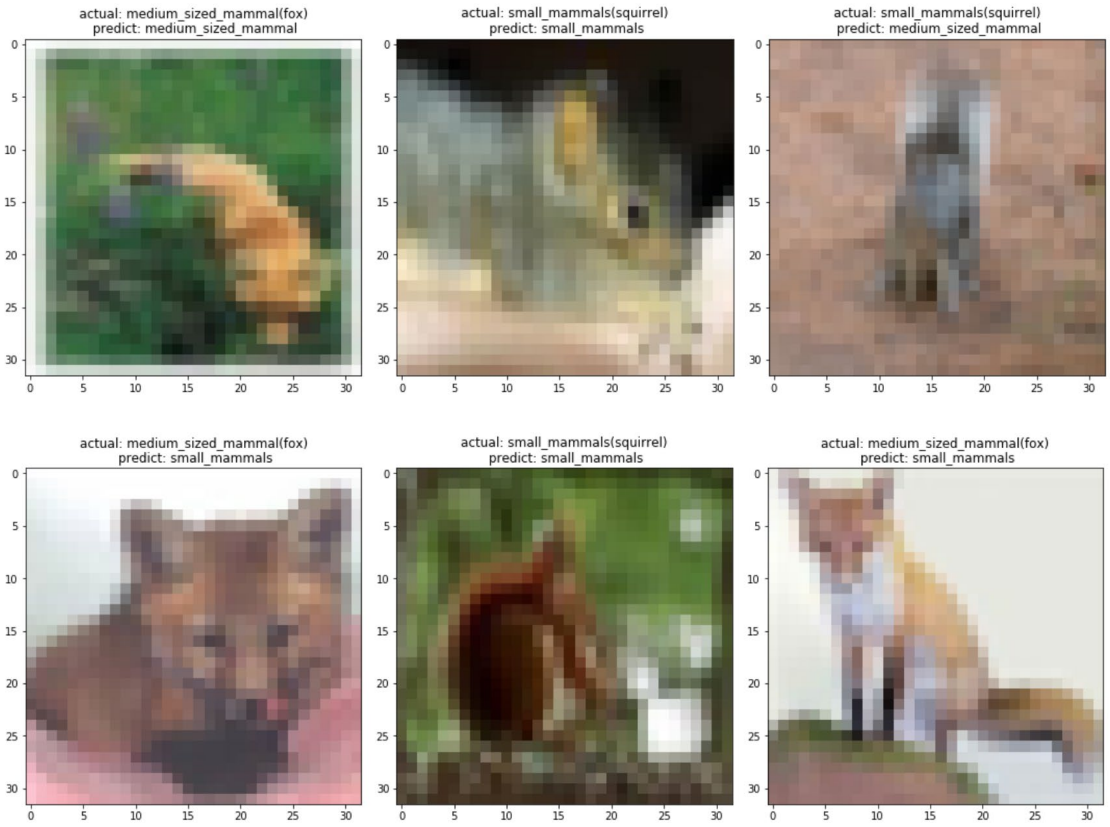
The prediction score on testing data with best estimator was found to be 42.50%.

```
print ('prediction score on testing data with the best estimator: %.2f%%' % (rbf_clf.best_estimator_.score(x_test2,y_test_bin)*100))
```

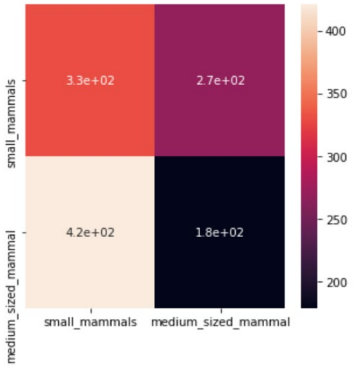
```
prediction score on testing data with the best estimator: 42.50%
```

The predicted data can be seen here:

```
test_label = ['squirrel','fox']
rbf_clf_pred=rbf_clf.predict(x_test2)
indices = [np.random.choice(range(len(x_test2))) for j in range(6)]
cifar_grid1(x_test2, y_test_bin,indices,3,rbf_clf_pred,test_label)
```



Confusion Matrix :



Classification Report :

	precision	recall	f1-score	support
0	0.44	0.55	0.49	600
1	0.40	0.30	0.34	600
micro avg	0.42	0.42	0.42	1200
macro avg	0.42	0.42	0.42	1200
weighted avg	0.42	0.42	0.42	1200

Hoping that logistic regression will give better results, we tried that.

Logistic Regression :

Logistic Regression is a predictive analysis, where predictions are made using logistic function and is used in classification problems. In logistic regression, coefficients are estimated using maximum likelihood and returns a probability value. Data preparation for Logistic regression is very similar to that of linear regression.

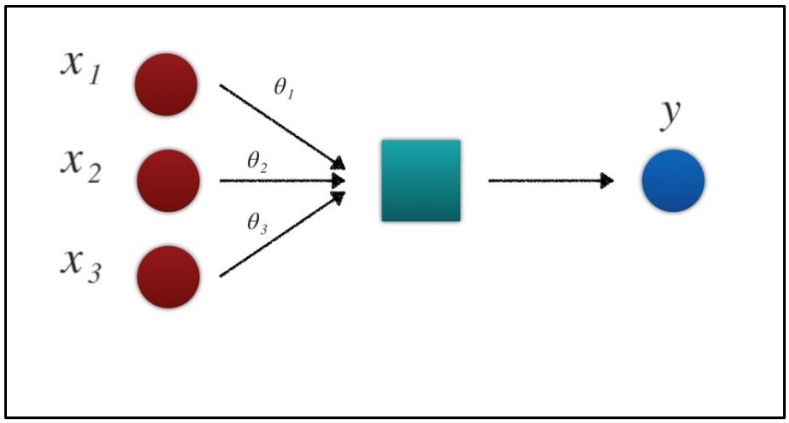


Figure 11 : Logistic regression model

- The model was defined and built, then fit using training data. The prediction was generated using ***.predict()*** and performance was evaluated by finding the score.

```
start = time.time()
lr = LogisticRegression()
lr.fit(x_train2, y_train_bin)
lr_pred=lr.predict(x_test2)
print ("Logistic Regression Accuracy: {}%".format(lr.score(x_test2, y_test_bin)*100))
end = time.time()
print('{} seconds'.format(end - start))
```

```
Logistic Regression Accuracy: 39.75%
51.61750555038452 seconds
```

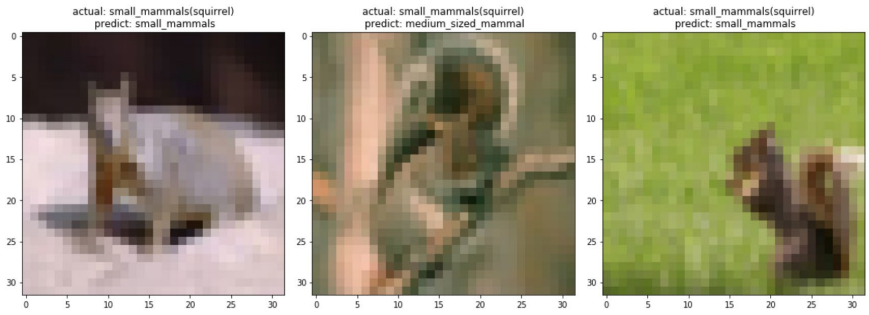
The prediction score on testing data with best estimator was found to be 44.75%.

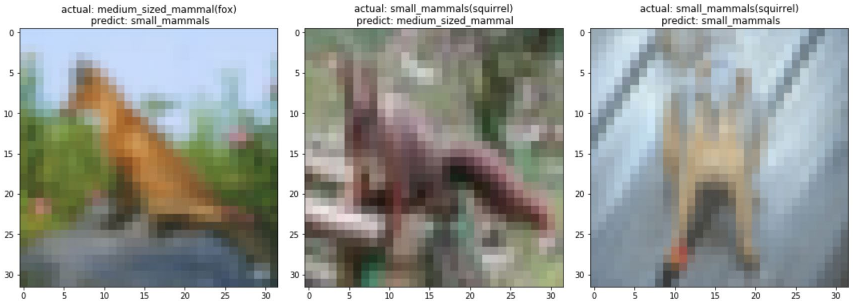
```
print ('prediction score on testing data with the best estimator: %.2f%%' % (lr_clf.best_estimator_.score(x_test2, y_test_bin)*100))
```

```
prediction score on testing data with the best estimator: 44.75%
```

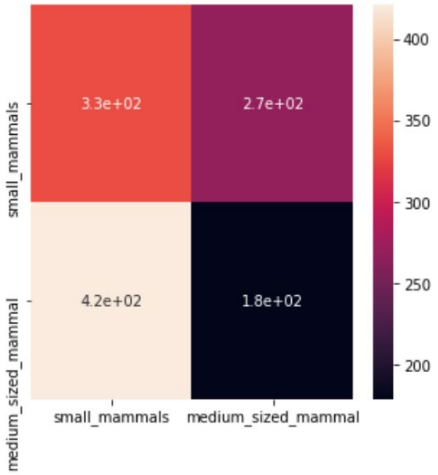
The predicted data can be seen here:

```
test_label = ['squirrel','fox']
lr_clf_pred=rbf_clf.predict(x_test2)
indices = [np.random.choice(range(len(x_test2))) for j in range(6)]
cifar_grid1(x_test2, y_test_bin,indices,3,lr_clf_pred,test_label)
```





Confusion Matrix :



Classification Report :

	precision	recall	f1-score	support
0	0.44	0.55	0.49	600
1	0.40	0.30	0.34	600
micro avg	0.42	0.42	0.42	1200
macro avg	0.42	0.42	0.42	1200
weighted avg	0.42	0.42	0.42	1200

K Nearest Neighbors classifier :

We tried K-nearest neighbors (KNN) hoping to find a better accuracy score. K-nearest neighbors classifier is used for classification and regression in pattern recognition. K is the number of nearest neighbors in KNN, which is the main deciding factor. KNN shows better performance with a lower number of features. KNN has a risk of overfitting when the number of dimensions are increased.

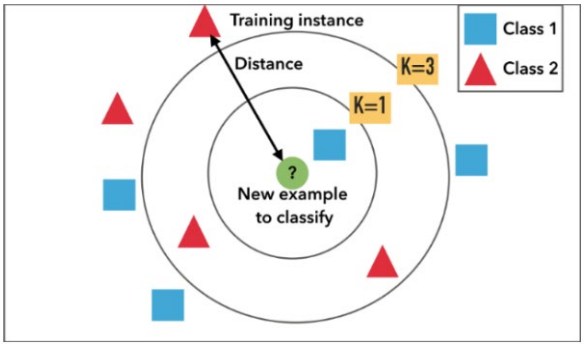


Figure 12 : KNN model diagram.

- The model was defined and built, then fit using training data. The prediction was generated using ***.predict()*** and performance was evaluated by finding the score.

```
start = time.time()
knn = KNeighborsClassifier()
knn.fit(x_train2, y_train_bin)
knn_pred=knn.predict(x_test2)
print ("KNN Accuracy: {}%".format(knn.score(x_test2, y_test_bin)*100))
end = time.time()
print('{} seconds'.format(end - start))
```

```
KNN Accuracy: 50.416666666666664%
193.1702675819397 seconds
```

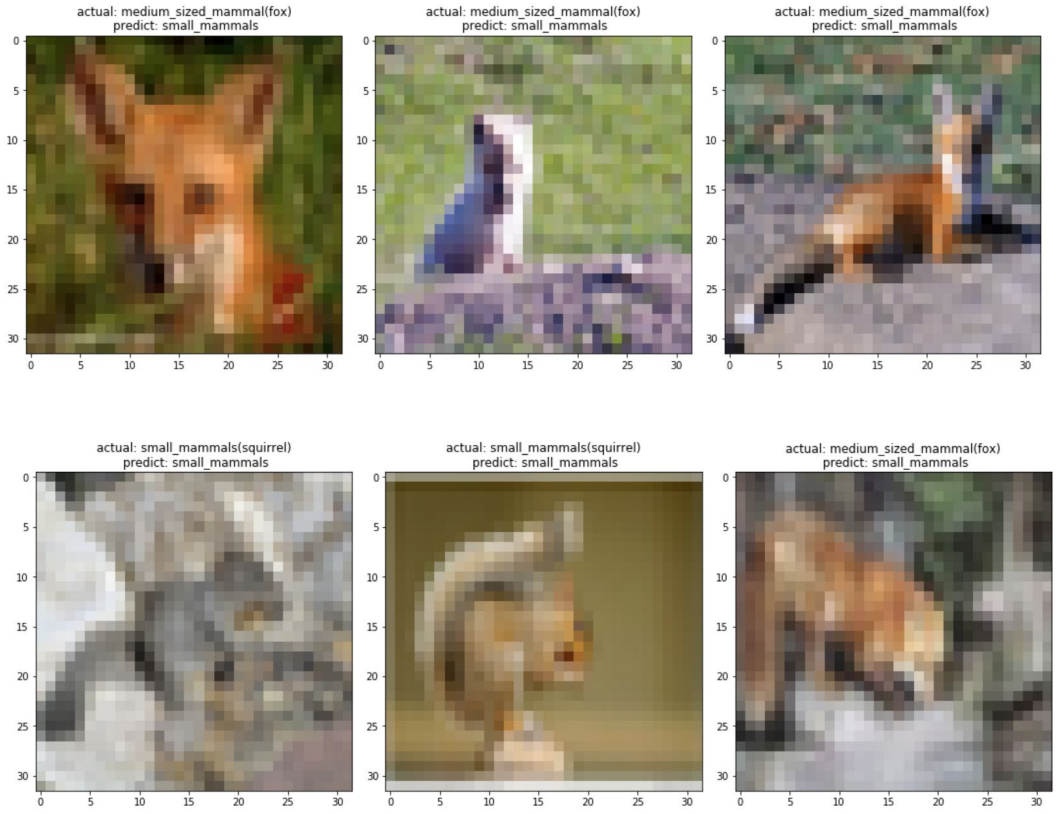
The prediction score on testing data with best estimator was found to be 50.92%.

```
print ('prediction score on testing data with the best estimator: %.2f%%' % (knn_clf.best_estimator_.score(x_test2, y_test_bin)*100))
```

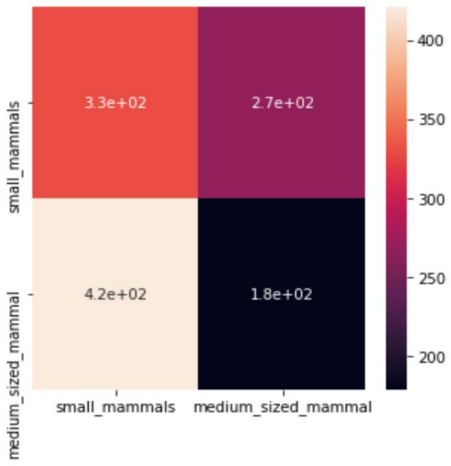
```
prediction score on testing data with the best estimator: 50.92%
```

The predicted data can be seen here:

```
test_label = ['squirrel','fox']
knn_clf_pred=rbf_clf.predict(x_test2)
indices = [np.random.choice(range(len(x_test2))) for j in range(6)]
cifar_grid1(x_test2, y_test_bin,indices,3,knn_clf_pred,test_label)
```



Confusion Matrix :



Classification Report :

	precision	recall	f1-score	support
0	0.44	0.55	0.49	600
1	0.40	0.30	0.34	600
micro avg	0.42	0.42	0.42	1200
macro avg	0.42	0.42	0.42	1200
weighted avg	0.42	0.42	0.42	1200

The model was run to find the accuracy score of different combinations using a function.

```
knn_accuracy=list()
knn_time=list()
print("KNN ")
for i in range(0,25):
  start = time.time()
  knn1 = KNeighborsClassifier(n_neighbors=8,weights='distance')
  knn1.fit(x_train3_[i], y_train_bin3[i])
  knn1_pred=knn1.predict(x_test3_[i])
  print ("Accuracy ({}, {}): {}%".format(test_list[i][0],test_list[i][1] ,knn1.score(x_test3_[i], y_test_bin3[i])*100))
  end = time.time()
  print('{} seconds'.format(end - start))
  knn_accuracy.append(knn1.score(x_test3_[i], y_test_bin3[i]))
  knn_time.append(end-start)
  # choose 36 img randomly
  #indices = [np.random.choice(range(len(x_test3_[i]))) for j in range(36)]
  #cifar_grid1(x_test3_[i], y_test_bin3[i], indices,4,lr_pred)
print("average accuracy: {}%".format(np.mean(knn_accuracy)*100))
print("average time: {} seconds".format(np.mean(knn_time)))
```

The resulting scores for each combination can be seen from the following table.

K-neighbors	Fox	Porcupine	Possum	Raccoon	Skunk	AVERAGE
Hamster	53.08%	56.41%	56.92%	53.75%	53.34%	54.70%
Mouse	49.92%	54.92%	54.83%	52.83%	52.00%	52.90%
Rabbit	48%	55.00%	55.75%	52.00%	52.17%	53%
Shrew	43.25%	50.16%	50.58%	49.00%	49.58%	48.51%
Squirrel	50.67%	56.33%	56.16%	54.83%	54.08%	54.41%
AVERAGE	48.92%	54.56%	54.85%	52.48%	52.23%	52.61%

K-neighbors had the best overall score (52.61%), with its best individual score being 56.92% (in this iteration). Results had the following characteristics:

- Best pairing: **Hamster & Possum (56.92%)**
- Worst pairing: **Fox & Shrew (43.25%)**
- Easiest class to predict: **Possum (54.85%)**

- Hardest class to predict: **Shrew (48.51%)**

It becomes clear that there is a relation between the pairing being used as test, the easiness to predict that class and the model's performance.

Bagging Classifier:

As the next step, we tried Bagging classifier or Bootstrap Aggregation, which is used for classification and regression problems. Bagging classifier accepts the designation of a base classifier as input, and that is replicated n times by bagging ensemble. Primary deciding hyper-parameters are number of base classifiers.

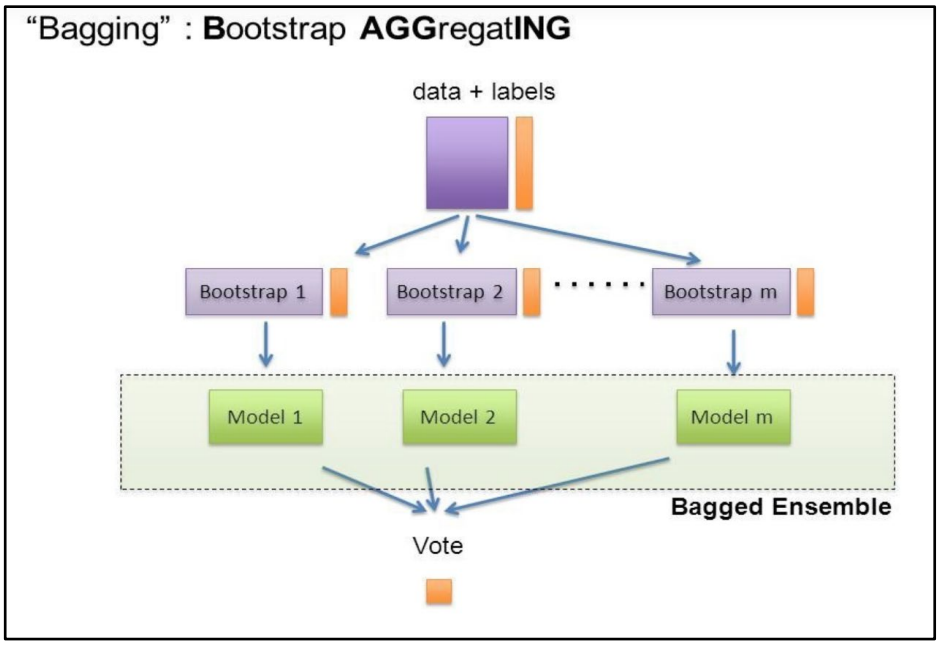


Figure 13 : Bagging classifier architecture

Bagging classifier implementation was done by defining and building, then fitting using training data. The prediction was generated using ***.predict()*** and performance was evaluated by finding the score.

```
start = time.time()
bag = BaggingClassifier()
bag.fit(x_train2, y_train_bin)
#bag_pred=bag.predict(x_test2)
print ("Bagging Accuracy: {}%".format(bag.score(x_test2, y_test_bin)*100))
end = time.time()
print('{} seconds'.format(end - start))
```

```
Bagging Accuracy: 46.416666666666664%
190.04374408721924 seconds
```

The prediction score on testing data with best estimator was found to be 50.83%.

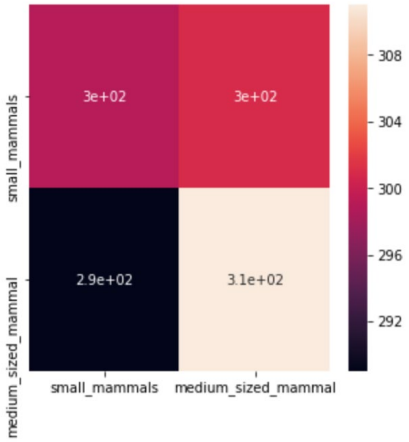
```
print ('prediction score on testing data with the best estimator: %.2f%%' % (bag_clf.best_estimator_.score(x_test2, y_test_bin)*100))
```

```
prediction score on testing data with the best estimator: 50.83%
```

The predicted data can be seen here:

```
test_label = ['squirrel','fox']
bag_clf_pred=bag_clf.predict(x_test2)
indices = [np.random.choice(range(len(x_test2))) for j in range(6)]
cifar_grid1(x_test2, y_test_bin,indices,3,bag_clf_pred,test_label)
```

Confusion Matrix :



Classification Report :

	precision	recall	f1-score	support
0	0.51	0.50	0.50	600
1	0.51	0.52	0.51	600
micro avg	0.51	0.51	0.51	1200
macro avg	0.51	0.51	0.51	1200
weighted avg	0.51	0.51	0.51	1200

The model was run to find the accuracy score of different combinations using a function.

```
bag_accuracy=list()
bag_time=list()
print("Bagging Classifier ")
for i in range(0,25):
  start = time.time()
  bag1 = BaggingClassifier(bootstrap = False, bootstrap_features = False, max_samples = 0.01, n_estimators = 3)
  bag1.fit(x_train3_[i], y_train_bin3[i])
  bag1_pred=bag1.predict(x_test3_[i])
  print ("Accuracy ({}, {}): {}%".format(test_list[i][0],test_list[i][1] ,bag1.score(x_test3_[i], y_test_bin3[i])*100))
  end = time.time()
  print('{} seconds'.format(end - start))
  bag_accuracy.append(bag1.score(x_test3_[i], y_test_bin3[i]))
  bag_time.append(end-start)
  # choose 36 img randomly
  #indices = [np.random.choice(range(len(x_test3_[i]))) for j in range(36)]
  #cifar_grid1(x_test3_[i], y_test_bin3[i], indices,4,lr_pred)
print("average accuracy: {}%".format(np.mean(bag_accuracy)*100))
print("average time: {} seconds".format(np.mean(bag_time)))
```

The resulting scores for each combination can be seen from the following table.

Bagging	Fox	Porcupine	Possum	Raccoon	Skunk	AVERAGE

Hamster	58.09%	48.09%	62.33%	50.84%	47.50%	53.37%
Mouse	47.50%	56.92%	56.17%	52.75%	51.50%	52.97%
Rabbit	45%	40.50%	51.17%	53.75%	52.50%	49%
Shrew	43.84%	46.75%	49.50%	47.67%	51.17%	47.79%
Squirrel	47.50%	50.84%	50.58%	49%	49.58%	49.50%
AVERAGE	48.39%	48.62%	53.95%	50.80%	50.45%	50.44%

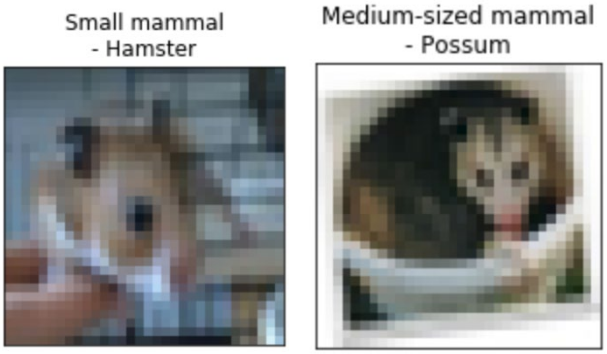
This classifier had a score of 59.05% for our standard pairing (Fox & Hamster). The average score was 50.44%.

Results had the following characteristics:

- Best pairing: **Hamster & Possum (62.33%)** → same as K-neighbors
- Worst pairing: **Rabbit & Porcupine (40.50%)**
- Easiest class to predict: **Possum (53.95%**) → same as K-neighbors
- Hardest class to predict: **Shrew (47.79%)** → same as K-neighbors

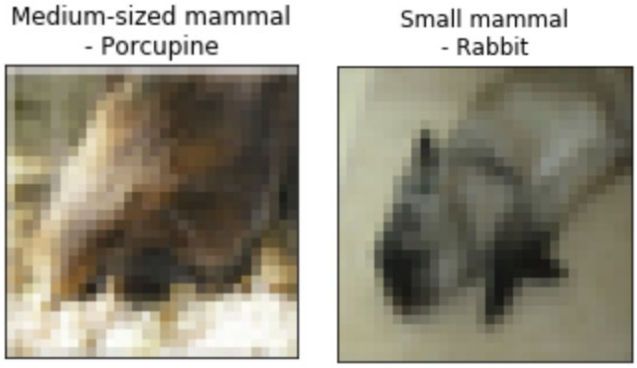
Overall Analysis :

Comparing the results from K-neighbors and Bagging, some similarities come to light.
In both cases the best result was obtained with **Hamster & Possum** as the testing sample, which was not the obvious answer once you look at their pictures. Personally, we expected the Skunk to have the best result given its very distinctive coloring. Nonetheless this result is coherent with the rest of the results, since both Hamster and Possum ranked high their individual scores.

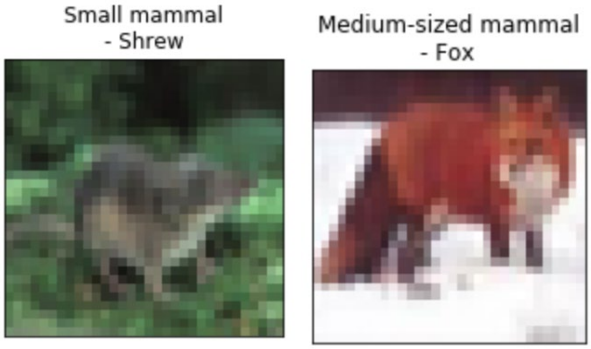


Different results were obtained as the worst pairing to predict **Fox & Shrew (K-neighbors)** and **Rabbit & Porcupine (Bagging)**.
Rabbit & Porcupine are indeed hard to distinguish, since there are many similarities once

there is no size difference and the detail level is reduced to a point where one can no longer see the needles of the Porcupine.



Once we look at Fox & Shrew, it is harder to understand, since both classes are very different from each other. Here we attribute the bad performance to the difficulty of predicting the species itself, Shrew was in both models the worst class to predict with Fox coming as a close second. Basically, we believe that Shrew and Fox are not being confused with each other, but with other species of the mega-classes.



In this milestone, we used multiple algorithms to train models to recognize small sized mammals and medium sized mammals. During the project, we understood the importance of evaluating our model. By comparing training loss and testing loss, we can adjust the model to avoid overfitting. By looking at the confusion matrix and confusion report, we can infer the cause of our errors, and fine tune our model accordingly. The most important thing for a machine learning/deep learning model is not just a high score, but also robustness so it can predict for different cases.
For the milestone two (image classification for super classes: small and medium mammals) we selected **K- Nearest Neighbors** as the most promising, since it delivered a good score (50.92%) spending a reasonable amount of time processing (193.17s).

Milestone 3 - Prediction on two testing subclasses from each of the two superclasses

After a quick performance assessment of the machine learning models for the new data separation (4 classes as test set) that is later shown on the performance section, we focused our analysis on Bagging Classifier. For this model we ran both the milestone 3 analysis and the bonus.

Bagging Classifier

As before the model was defined and built, then fit using training data. We used the grid search to tune the parameters n_estimators, max_sample, bootstrap, and bootstrap_features. The prediction was generated using ***.predict()*** and generated a score of 52% (for 2 pairs missing data).

```
start = time.time()
bag = BaggingClassifier(n_estimators = 3, max_samples = 0.01, bootstrap = True, bootstrap_features = False)
bag.fit(x_remaining_mammals, y_remaining_mammals_bin)
bag_pred=bag.predict(x_rest_mammals)
scores = cross_val_score(bag, x_remaining_mammals,y_remaining_mammals, cv=5)
print(scores)
scores = cross_val_score(bag, x_rest_mammals,y_rest_mammals_bin, cv=5)
print(scores)
print ("Bagging Accuracy: {}%".format(bag.score(x_remaining_mammals, y_remaining_mammals_bin)*100))
print ("Bagging Accuracy: {}%".format(bag.score(x_rest_mammals, y_rest_mammals_bin)*100))
end = time.time()
print(end - start)
```

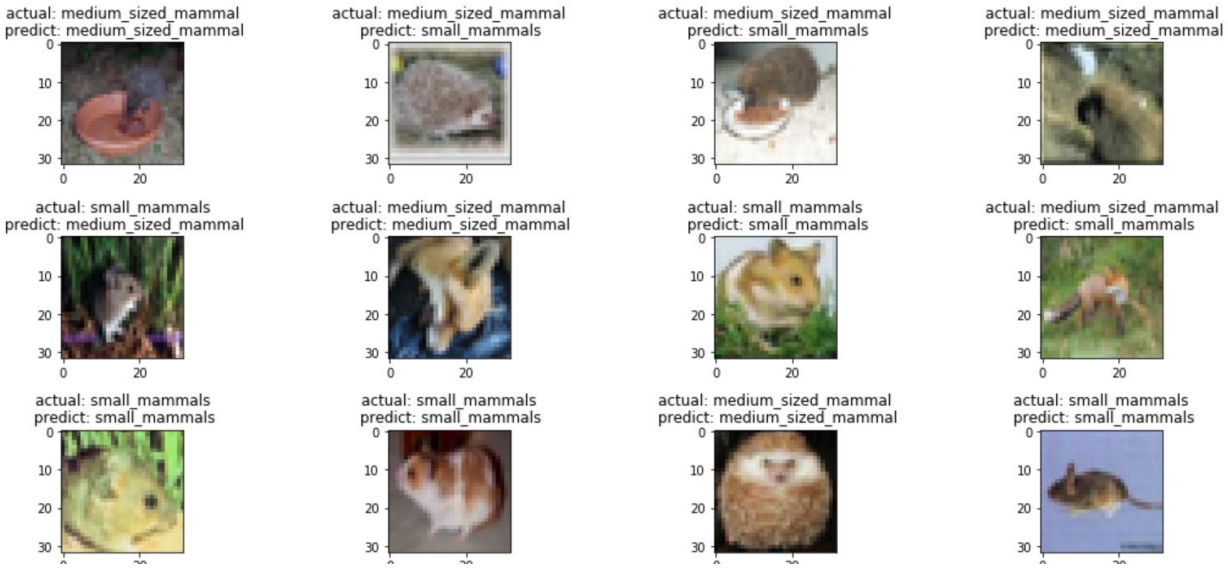
```
[0.33541667 0.35416667 0.35625    0.35416667 0.26666667]
[0.53194444 0.53055556 0.52222222 0.525      0.54583333]
Bagging Accuracy: 62.0%
Bagging Accuracy: 52.0%
3.4020745754241943
```

The best parameters were:

- For 2 pairs missing data, n_estimators: 3, max_sample: 0.01, bootstrap: False, bootstrap_features: False
- For 3 pairs missing data, n_estimators: 3, max_sample: 0.01, bootstrap: True, bootstrap_features: False

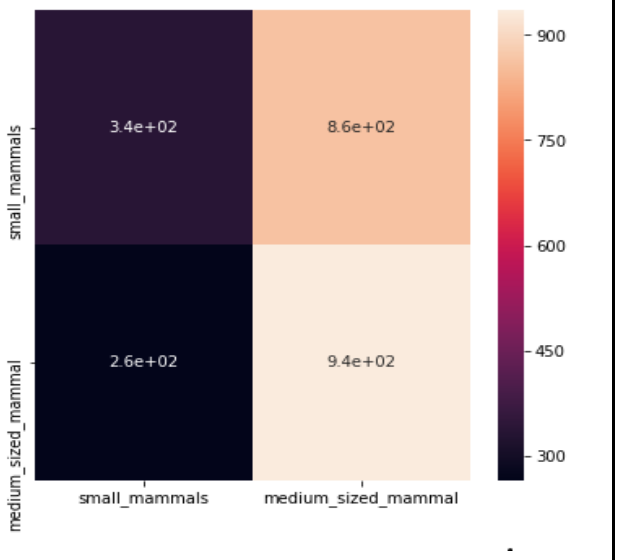
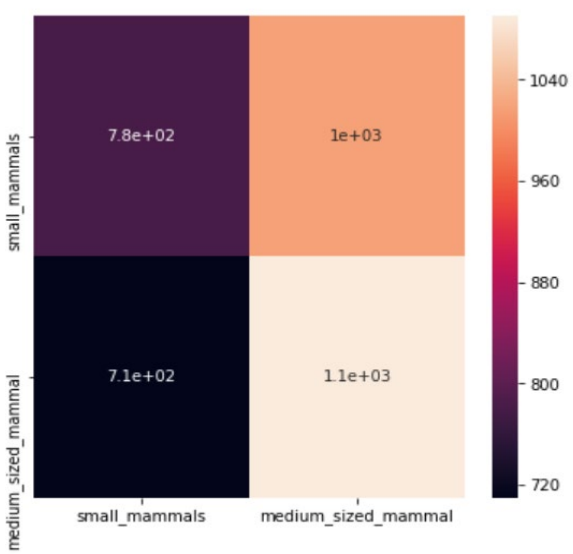
For 2 pairs missing, we took a 36 image sample as follows.:


actual: small_mammals
predict: medium_sized_mammal
actual: small_mammals
predict: small_mammals
actual: small_mammals
predict: small_mammals
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: small_mammals
predict: medium_sized_mammal
actual: small_mammals
predict: medium_sized_mammal
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: small_mammals
predict: small_mammals
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: small_mammals
predict: medium_sized_mammal
actual: medium_sized_mammal
predict: medium_sized_mammal
actual: small_mammals
predict: medium_sized_mammal
actual: small_mammals
predict: small_mammals
actual: small_mammals
predict: small_mammals
actual: small_mammals
predict: medium_sized_mammal
actual: small_mammals
predict: medium_sized_mammal
actual: small_mammals
predict: medium_sized_mammal
actual: small_mammals
predict: small_mammals
actual: small_mammals
predict: medium_sized_mammal
actual: medium_sized_mammal
predict: small_mammals
actual: medium_sized_mammal
predict: medium_sized_mammal



Confusion Matrix & Classification Report:

Since our data set is balanced, that is we have equal amounts of medium and small mammals, if a model is predicting 100% in one class (or just flipping a coin), it would still reach a 50% accuracy. That makes this analysis even more relevant. The table below contains the confusion matrix for milestone 3 and the bonus.

	
Confusion Matrix: Bagging (2 pairs)	Confusion Matrix: Bagging (3 Pairs)

Both matrix show that the model has a tendency to predict medium sized mammal more frequently than small mammal, this effect was also observed in milestone 1 and 2. Interestingly, reducing the number of classes available to train the model, reduces this effect, even as its precision is also reduced. This observations are confirmed by the classification report.

	precision	recall	f1-score	support
small_mammals	0.56	0.28	0.37	1200
medium_sized_mammal	0.52	0.78	0.62	1200
micro avg	0.53	0.53	0.53	2400
macro avg	0.54	0.53	0.50	2400
weighted avg	0.54	0.53	0.50	2400

Classification report: Bagging (2 pairs)

	precision	recall	f1-score	support
small_mammals	0.52	0.43	0.48	1800
medium_sized_mammal	0.52	0.61	0.56	1800
micro avg	0.52	0.52	0.52	3600
macro avg	0.52	0.52	0.52	3600
weighted avg	0.52	0.52	0.52	3600

Classification report: Bagging (3 Pairs)

As commented before, the f1-scores as well as the recall score, clearly points to a tendency of predicting medium sized mammals more often. This is further supported by the fact that, in 2 pair missing data, the small mammals have a bigger precision than medium sized mammals. For the classification report of 3 pairs missing data, we can see that 'Medium sized Mammals' have a same precision with 'Small Mammals', indicating a more balance if less effective model.

Overall predictions for loop:

2 Pairs missing scenario

The resulting scores for each combination can be seen on the table below. Notice that differently from milestone 2, here there are not clear clusters of "good" and "bad" scores.

	fox, porcupine	fox, possum	fox, raccoon	fox, skunk	porcupine, skunk	porcupine, raccoon	porcupine, possum	skunk, raccoon	skunk, raccoon	possum, raccoon	
hamster,mouse	50.04%	54.67%	58.71%	50.58%	51.96%	55.71%	52.92%	55.42%	43.00%	59.25%	53.23%
hamster,rabbit	56.08%	48.33%	52.67%	48.33%	48.25%	54.75%	52.08%	53.25%	47.38%	48.92%	51.00%
hamster, shrew	52.67%	51.46%	55.67%	49.29%	58.25%	53.29%	53.21%	53.71%	52.25%	56.38%	53.62%
hamster, squirrel	45.75%	56.50%	52.67%	51.17%	49.83%	58.58%	46.63%	47.54%	53.00%	55.04%	51.67%
mouse, rabbit	53.37%	54.40%	54.42%	50.88%	52.08%	53.13%	54.75%	51.00%	56.63%	52.33%	53.30%
mouse, shrew	50.00%	46.00%	47.58%	54.16%	59.71%	46.38%	56.79%	51.13%	48.75%	51.00%	51.15%
mouse, squirrel	55.63%	54.83%	54.00%	50.42%	53.63%	51.83%	53.67%	54.75%	48.54%	50.58%	52.79%
rabbit, shrew	50.54%	48.96%	52.38%	46.46%	54.04%	57.21%	54.25%	53.92%	55.13%	49.92%	52.28%
rabbit, squirrel	53.25%	52.46%	52.63%	47.63%	54.00%	53.92%	56.42%	52.29%	52.83%	53.63%	52.91%
shrew, squirrel	51.25%	46.88%	49.29%	45.42%	54.58%	53.67%	50.46%	51.67%	49.54%	50.38%	50.31%
	51.86%	51.45%	53.00%	49.43%	53.63%	53.85%	53.12%	52.47%	50.71%	52.74%	52.23%

In general the following characteristics were found:

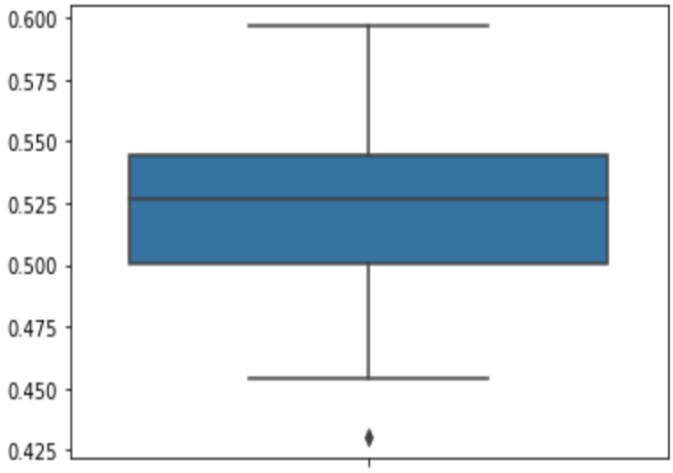
- Best testing pair: porcupine & skunk vs mouse & shrew (59.71%)
- Worst testing pair: skunk & raccoon vs hamster & mouse (43%)

- Easiest group to predict: Porcupine, Raccoon (53.58%)
- Hardest group to predict: Fox, Skunk (49.43%)

Since we are looking from the perspective of the testing data set, it's important to remember that saying the porcupine & skunk vs mouse & shrew are the **best pair** to use as testing set actually means that they are the classes **adding the least value** to train the model.

Thus we can conclude that having **Fox & Skunk** in the **training set** makes for the **best predictive model.**

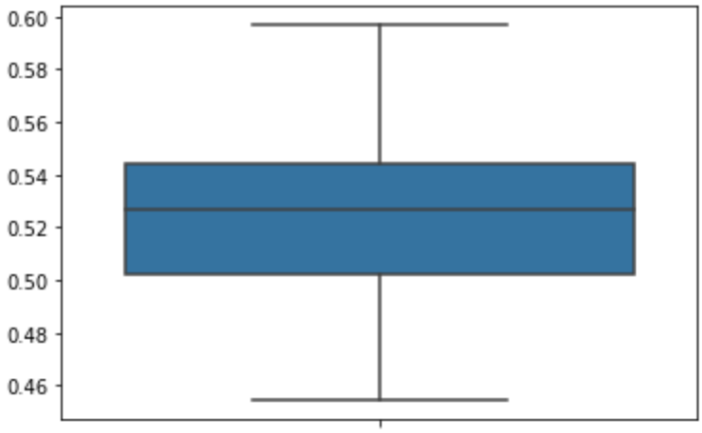
Another interesting thing to notice is that when we look at the statistical distribution of the scores and plot it, we can see that our worst score (43%) is an outlier (see figure below).



	2_pair_missing
count	100.000000
mean	0.522257
std	0.033361
min	0.430000
25%	0.500300
50%	0.526500
75%	0.544050
max	0.597100

Bagging's scores: Statistical distribution (2 pairs missing)

If we exclude it from the analysis, that is, if we commit to always have "skunk & raccoon vs hamster & mouse" in the training data set, the performance the model is improved slightly (from 52.2% to 52.3%).



	2_pair_missing
count	99.000000
mean	0.523189
std	0.032196
min	0.454200
25%	0.502100
50%	0.526700
75%	0.544100
max	0.597100

Bagging's scores: Statistical distribution (2 pairs missing) - without outlier

Looking at the statistical data, once the outlier is removed, we can conclude that the model is relatively consistent through all parrings. A standard deviation of 3.2% is an acceptable value and even if we have a 14% delta from the worst prediction to the best prediction, 50% of the scores remain within 4 percentual points.

3 Pairs missing scenario

The resulting scores for each combination can be seen on the table below. Once again, looking at the top 5 (best and worse) combinations, there are no clear clusters, although the group "porcupine, raccoon, possum" generates consistently bad results. As indicated on the previous session, this means that this is a group important for the training of the model.

	fox, porcupine, skunk	fox, porcupine, raccoon	fox, porcupine, possum	fox, skunk, raccoon	fox, skunk, possum	porcupine, skunk, raccoon	porcupine, skunk, possum	skunk, raccoon, possum	porcupine, raccoon, possum	fox, possum, raccoon	
hamster,mouse,rabbit	50.33%	46.58%	56.11%	46.11%	49.08%	47.02%	52.44%	49.52%	42.41%	54.11%	49.37%
hamster,mouse,shrew	50.00%	50.27%	52.00%	49.22%	48.36%	52.44%	56.11%	51.91%	52.41%	44.16%	50.69%
hamster,mouse,squirrel	50.11%	53.52%	54.75%	49.91%	54.30%	59.77%	49.16%	50.83%	44.61%	54.22%	52.12%
hamster,rabbit,shrew	49.17%	58.13%	53.16%	43.91%	54.61%	51.19%	50.94%	53.38%	52.97%	51.58%	51.90%
hamster,rabbit,squirrel	54.25%	56.75%	45.38%	51.58%	52.94%	51.41%	48.36%	47.27%	42.33%	53.72%	50.00%
hamster,shrew,squirrel	49.61%	53.08%	55.13%	54.00%	49.00%	53.33%	51.61%	50.88%	50.66%	53.27%	52.13%
mouse,rabbit,shrew	53.69%	53.69%	54.63%	53.86%	52.61%	50.33%	54.36%	47.02%	51.86%	46.19%	51.49%
mouse,rabbit,squirrel	59.91%	52.69%	51.36%	55.22%	56.47%	53.25%	47.72%	53.36%	49.97%	54.38%	52.48%
rabbit,shrew,squirrel	49.62%	50.17%	49.41%	50.44%	56.22%	54.80%	52.14%	55.39%	49.75%	48.03%	51.60%
mouse,shrew,squirrel	51.27%	55.47%	54.72%	52.05%	50.50%	49.75%	52.52%	49.53%	54.92%	48.69%	51.85%
	51.80%	53%	52.67%	50.63%	52.40%	52.33%	51.54%	50.90%	49.19%	50.84%	51.50%

In general the following characteristics were found:

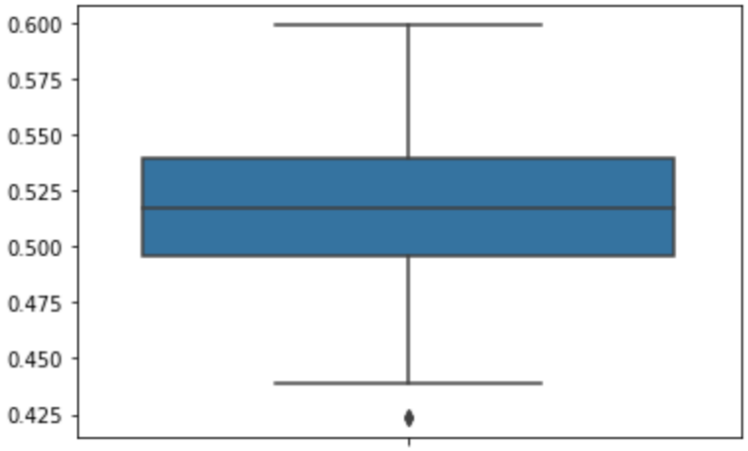
- Best testing pair: fox, porcupine, shunk vs mouse, rabbit, squirrel (59.91%)
- Worst testing pair: porcupine, raccoon, possum vs hamster, rabbit, squirrel (42.33%)

- Easiest group to predict: Porcupine, Raccoon, **Fox*** (53%)
- Hardest group to predict: Porcupine, Raccoon, **Possum** (49.19%)

A very intriguing effect happened in this analysis, where changing one class of the testing set, made a group go from being the easiest to predict to the hardest to predict. Even more, Fox, a class that in both M2 and M3 analysis was very relevant for model training, in this scenario loses its relevance. In M2, Fox was second to last worst class to predict and in M3, Fox was part of the hardest group to predict. Possum, on the other hand, in M2 was the easiest class to predict and middle of the pack in M3.

This reverse in behavior is quite intriguing and indicates that there is a more complex dynamic between classes than initially thought. Most probably, there are several optimal combinations and the interaction between classes varies depending on who is on the training set.

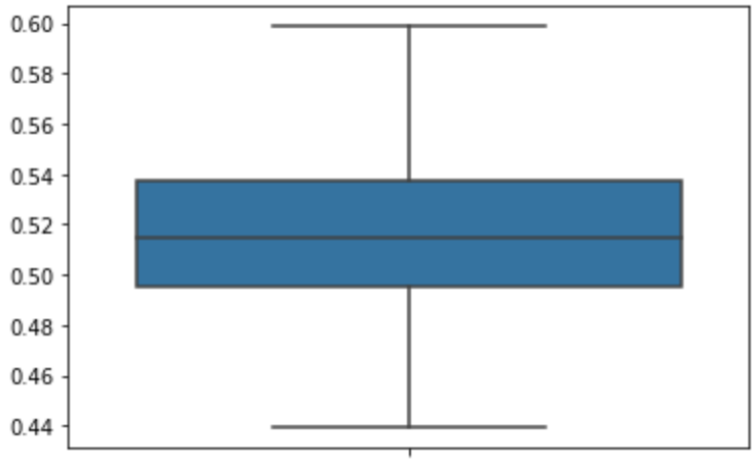
Looking at the statistical distribution of the scores, once again we see that our worst score is an outlier at (42.3%).



	3 Pairs (%)
count	100.000000
mean	0.515333
std	0.034236
min	0.423300
25%	0.495900
50%	0.517350
75%	0.538950
max	0.599100

Bagging's scores: Statistical distribution (3 pairs missing)

Excluding all outliers, the performance the model is improved slightly (from 51.1% to 51.4%). As commented before, there seems to be a more formalized cluster of low performance, since 20 scores were considered outliers.



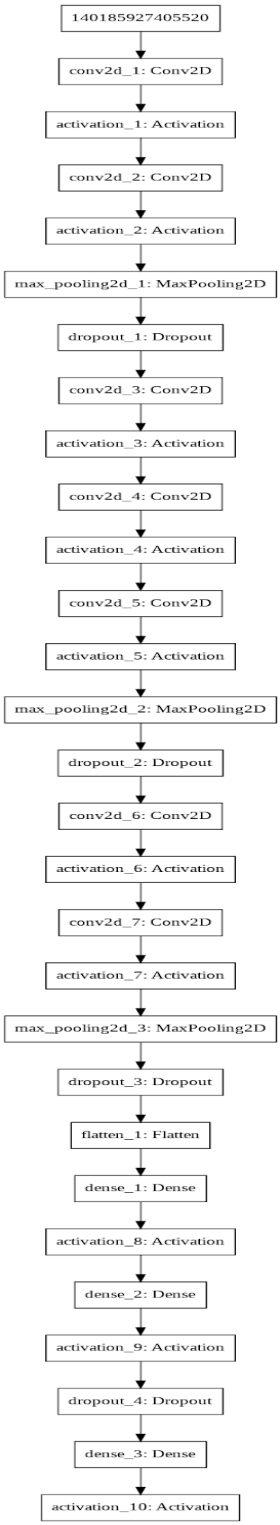
	3 Pairs (%)
count	80.000000
mean	0.514800
std	0.033532
min	0.439100
25%	0.495275
50%	0.514950
75%	0.537550
max	0.599100

Bagging's scores: Statistical distribution (3 pairs missing) - without outliers

Looking at the statistical data, this models behaves a less stable manner. The shear number of outliers, as well as the change in behavior for some classes seem to indicate a complex interaction between classes. Whereas 50% of the scores remain within a 4 percentual range, the model if drastically affected if certain classes, such as possum or shrew, are not used in the training.

CNN model

We used a long and narrow convolutional neural network to recognize our images. The structure of our model is listed below:



Overall predictions for loop:

2 Pairs missing scenario

We ran the CNN on all 100 pairs, accuracy score are as follows. The top 5 scores were colored blue, and the lowest 5 scores were colored orange.The pairs in the index were the ones taken out of the test set.

CNN	fox, porcupine	fox, possum	fox, raccoon	fox, skunk	porcupine, possum	porcupine, raccoon	porcupine, skunk	possum,racoon	possum, skunk	skunk, raccoon	
hamster,mouse	67.50%	72.38%	74.67%	74.96%	62.96%	72.33%	71.33%	68.71%	72.50%	72.62%	71.00%
hamster,rabbit	64.04%	68.12%	74.00%	75.62%	59.75%	70.17%	73.33%	73.67%	77.42%	79.83%	71.60%
hamster, shrew	67.00%	69.12%	72.62%	77.17%	61.83%	76.46%	73.38%	76.21%	70.08%	74.96%	71.88%
hamster, squirrel	76.12%	67.42%	69.96%	74.92%	76.75%	61.25%	69.21%	70.25%	71.96%	70.21%	70.81%
mouse, rabbit	66.17%	60.75%	65.46%	68.67%	59.96%	70.96%	72.92%	77.04%	77.17%	76.29%	69.54%
mouse, shrew	60.42%	59.00%	67.17%	64.04%	62.54%	71.88%	72.33%	71.75%	69.17%	72.25%	67.06%
mouse, squirrel	66.75%	65.75%	71.92%	68.50%	65.42%	72.62%	69.62%	73.83%	72.04%	74.00%	70.05%
rabbit, shrew	50.54%	60.04%	59.96%	69.00%	70.08%	73.21%	76.46%	73.42%	74.58%	79.12%	68.64%
rabbit, squirrel	61.38%	58.96%	66.46%	65.75%	60.42%	66.12%	68.33%	67.96%	70.38%	71.92%	6 5.77%
shrew, squirrel	65.54%	61.46%	68.71%	68.12%	60.54%	72.96%	74.92%	70.88%	69.88%	68.67%	68.17%
	64.55%	64.30%	69.09%	70.68%	64.03%	70.80%	72.18%	72.37%	72.52%	73.99%	69.45%

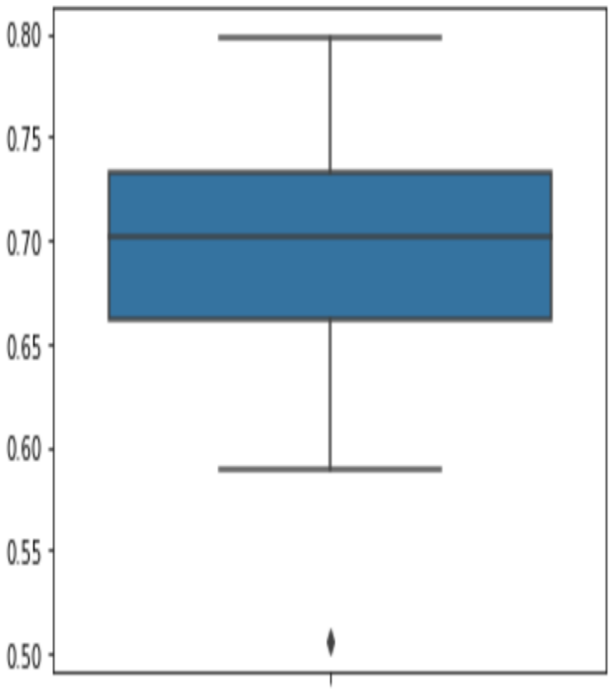
In general the following characteristics were found:

- Excluding this pair caused the best result: hamster & rabbit vs skunk & raccoon (79.83%)
- Excluding this pair caused the worst result: rabbit & shrew vs fox & porcupine (50.54%)
- Excluding this pair makes it the easiest group to predict: skunk, raccoon (73.99%)
- Excluding this pair makes it the hardest group to predict: porcupine, possum (64.03%)

Thus we can conclude that, for this CNN model, skunk and raccoon were the most difficult to separate, and porcupine is the easiest to identify.

The statistics for the 100 pairs were:

count	100.000000
mean	0.694495
std	0.055315
min	0.505400
25%	0.661575
50%	0.701250
75%	0.733425
max	0.798300



The outlier was the rabbit & shrew vs fox & porcupine pair, which indicates that it plays an important role in helping our model distinguish the 2 subclasses.

Confusion matrix comparison and classification report of the best and worst performing pairs:

Rabbit, shrew vs. fox, porcupine

	Predicted Class Small Mammals	Predicted Class Medium sized Mammals
Class Small Mammals	906	294
Class Medium sized Mammals	190	1010

	precision	recall	f1-score	support
Small Mammals	0.83	0.76	0.79	1200
Medium sized Mammals	0.77	0.84	0.81	1200
micro avg	0.80	0.80	0.80	2400
macro avg	0.80	0.80	0.80	2400
weighted avg	0.80	0.80	0.80	2400

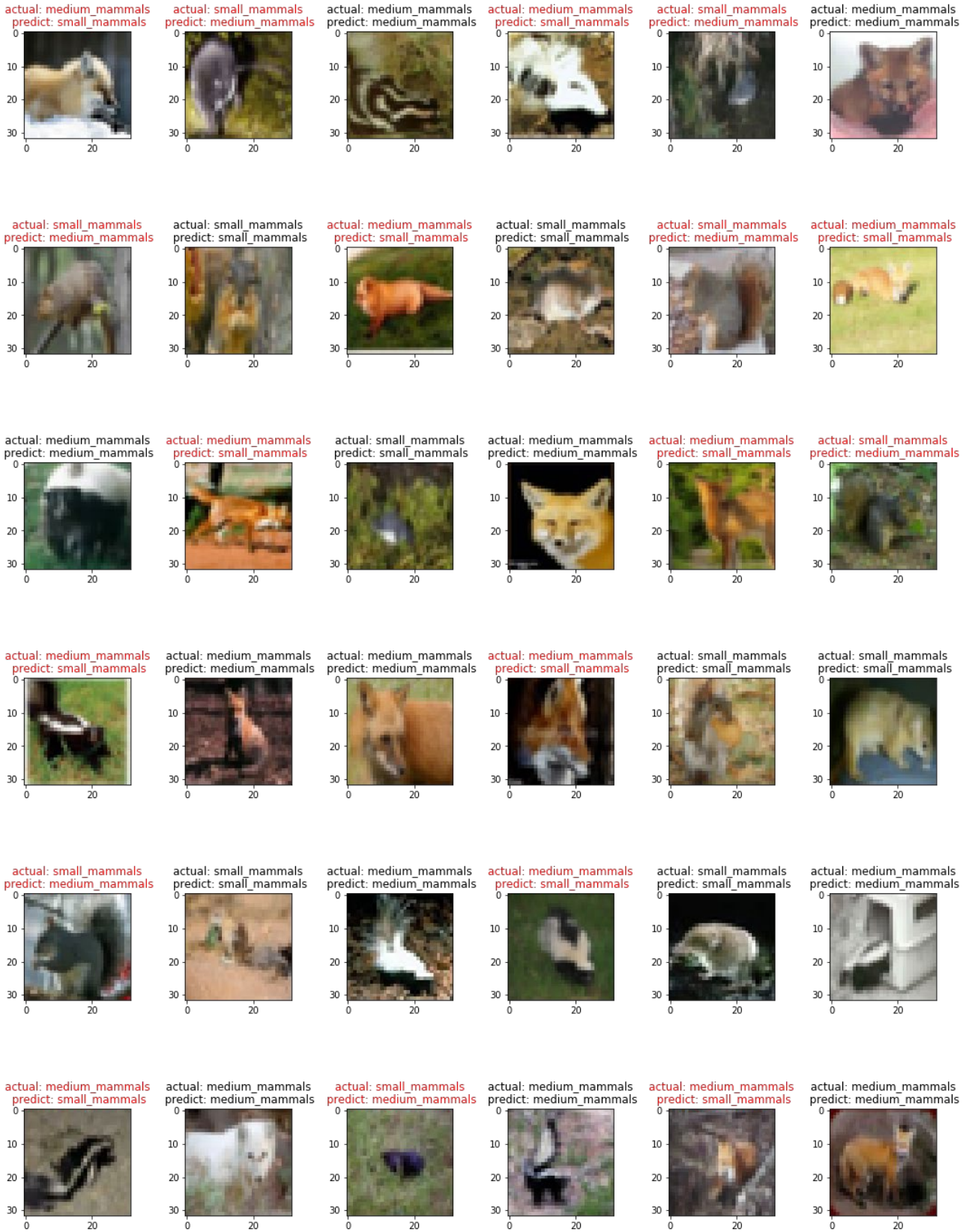
Hamster,rabbit vs. skunk, raccoon

	Predicted Class Small Mammals	Predicted Class Medium sized Mammals
Class Small Mammals	917	283
Class Medium sized Mammals	648	552

	precision	recall	f1-score	support
Small Mammals	0.59	0.76	0.66	1200
Medium sized Mammals	0.66	0.46	0.54	1200
micro avg	0.61	0.61	0.61	2400
macro avg	0.62	0.61	0.60	2400
weighted avg	0.62	0.61	0.60	2400

Generating random 36 images and see the prediction results. Wrong predictions were

colored red:


actual: medium_mammals
predict: small_mammals
actual: small_mammals
predict: medium_mammals
actual: medium_mammals
predict: medium_mammals
actual: medium_mammals
predict: small_mammals
actual: small_mammals
predict: medium_mammals
actual: medium_mammals
predict: medium_mammals
actual: small_mammals
predict: medium_mammals
actual: small_mammals
predict: small_mammals
actual: medium_mammals
predict: small_mammals
actual: small_mammals
predict: small_mammals
actual: small_mammals
predict: medium_mammals
actual: medium_mammals
predict: small_mammals
actual: medium_mammals
predict: medium_mammals
actual: medium_mammals
predict: small_mammals
actual: small_mammals
predict: small_mammals
actual: medium_mammals
predict: medium_mammals
actual: small_mammals
predict: medium_mammals
actual: medium_mammals
predict: small_mammals
actual: medium_mammals
predict: medium_mammals
actual: medium_mammals
predict: small_mammals
actual: small_mammals
predict: small_mammals
actual: small_mammals
predict: medium_mammals
actual: small_mammals
predict: small_mammals
actual: medium_mammals
predict: medium_mammals
actual: medium_mammals
predict: small_mammals
actual: small_mammals
predict: small_mammals
actual: medium_mammals
predict: small_mammals
actual: medium_mammals
predict: medium_mammals
actual: small_mammals
predict: medium_mammals
actual: medium_mammals
predict: medium_mammals
actual: medium_mammals
predict: small_mammals
actual: medium_mammals
predict: medium_mammals

BONUS: 3 Pairs missing scenario

We ran the exact same model on 2 pairs of test data and 3 pairs or missing data. The top 5 scores were colored blue, and the lowest 5 scores were colored orange.The pairs in the index were the ones taken out in the test set.

	fox, porcupine,skurk	fox, porcupine,raccoon	fox, porcupine,possum	fox, skurk, raccoon	fox, skurk,possum	porcupine, skurk, raccoon	porcupine, skurk,possum	skurk, raccoon,possum	porcupine, raccoon,possum	fox, possum,raccoon	
hamster,mouse,rabbit	66.06%	73.39%	69.94%	66.72%	64.97%	64.25%	63.39%	70.92%	68.31%	61.19%	66.91%
hamster,mouse,shrew	62.08%	71.19%	66.92%	65.75%	64.92%	63.61%	64.64%	67.53%	69.11%	61.47%	65.72%
hamster,mouse,squirrel	68.94%	68.44%	68.25%	68.36%	66.39%	68.00%	68.72%	71.92%	69.69%	65.58%	68.43%
hamster,rabbit,shrew	70.81%	75.17%	69.11%	67.31%	71.69%	64.33%	65.58%	69.47%%	65.69%	59.44%	67.68%
hamster,rabbit,squirrel	63.92%	74.56%	70.83%	65.69%	76.39%	71.47%	66.92%	77.14%	75.19%	64.58%	70.67%
hamster,shrew,squirrel	68.50%	65.58%	66.42%	59.39%	72.17%	73.17%	68.89%	73.44%	67.64%	61.39%	67.66%
mouse,rabbit,shrew	70.08%	72.31%	69.86%	66.33%	75.22%	73.86%	72.25%	74.06%	67.72%	63.36%	70.51%
mouse,rabbit,squirrel	55.92%	66.22%	66.67%	63.03%	68.97%	73.22%	72.33%	76.17%	74.61%	64.25%	68.14%
rabbit,shrew,squirrel	65.11%	73.22%	69.25%	64.67%	73.92%	68.00%	65.39%	71.67%	70.75%	61.44%	68.34%
mouse,shrew,squirrel	70.94%	61.25%	65.72%	63.17%	74.14%	71.72%	72.78%	71.78%	68.28%	63.36%	68.31%
	66.24%	70.13%	68.30%	65.04%	70.88%	69.16%	68.09%	72.74%	69.70%	62.61%	

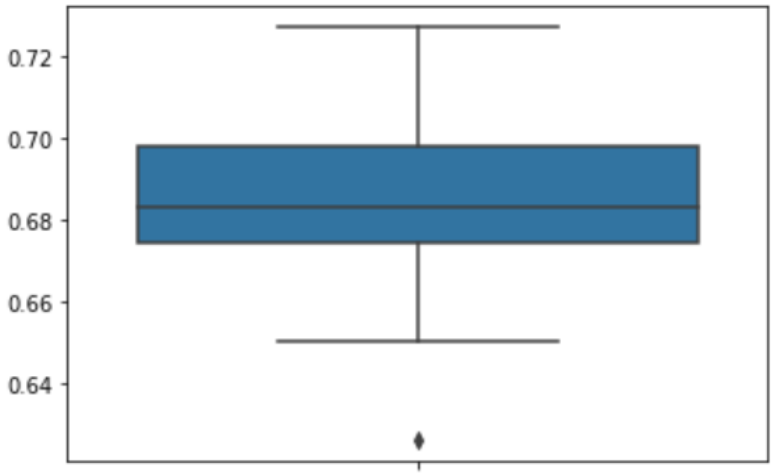
In general the following characteristics were found:

- Excluding this pair caused the best result: hamster & rabbit & squirrel vs skunk & raccoon & possum (77.14%)
- Excluding this pair caused the worst result: mouse & rabbit & squirrel vs fox & porcupine & skunk (50.54%)
- Excluding this pair makes it the easiest group to predict: skunk, raccoon, possum (72.74%)
- Excluding this pair makes it the hardest group to predict: fox & possum & raccoon (62.61%)

This results were consistent with the previous milestone 3 results, showing a consistent pattern that skunk, raccoon and possum are the most difficult to separate for this model and including fox in the training data will give us better results.

The statistics for the 100 pairs were:

count	100.000000
mean	0.682720
std	0.022876
min	0.626100
25%	0.674725
50%	0.683050
75%	0.698075
max	0.727400



Performance

Through the three milestones we tested several classifier models and neural networks. As the process progressed we excluded certain models or didn't explore them to their fullest extent. The chart below shows the evolution of the score of certain models:

- In milestone 1, we show the score where we randomly selected 83% of the data to train and 17% to test;
- In milestone 2, we show the score for the testing pair Raccoon vs Hamster
- In milestone 3 (both parts), we show the score for the testing pair Raccoon vs Hamster

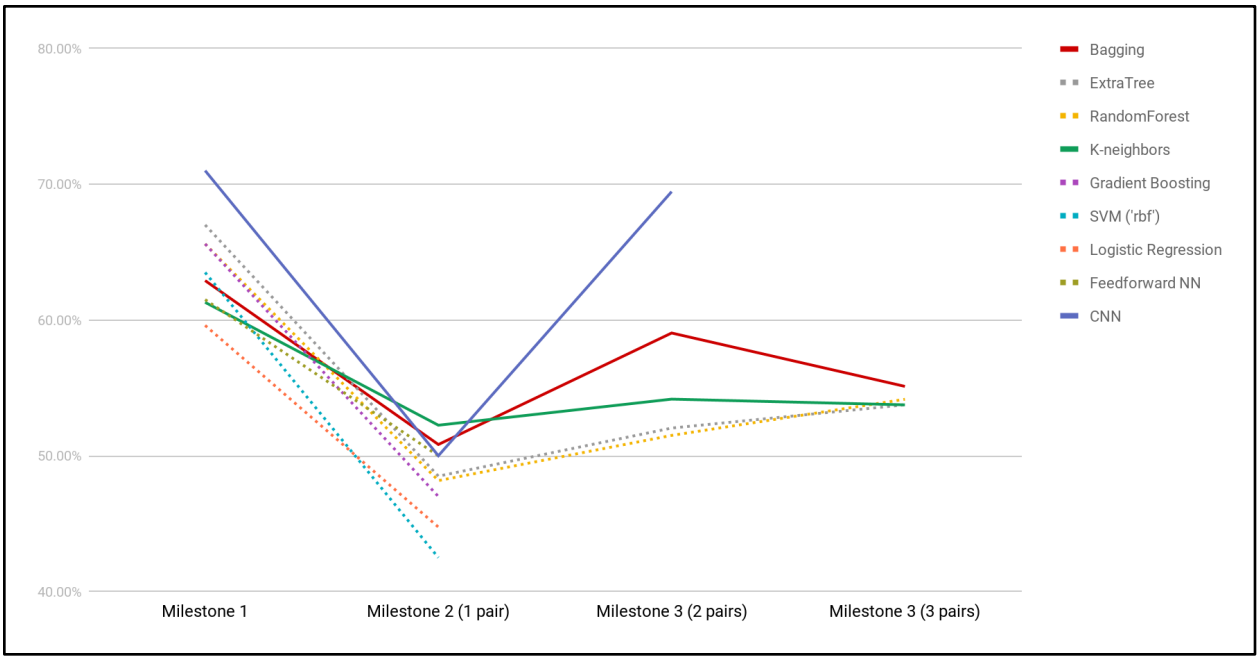


Figure : Comparison of algorithm performance

As the chart shows, in milestone 1, most algorithms delivered scores varying from 60 to 70%, it was once we singled out the pair Raccoon & Hamster as testing data (milestone 2) that we were able to start visualizing the consistency of each model.

Models such as Gradient Boosting, SVM, Logistic Regression presented such a decrease in performance that they were no longer considered for future experiments. The neural networks, both convolutional (CNN) and feedforward started to overfit very easily and their confusion matrix showed that their score of 50% was because they were only classifying as one category (medium mammals). For this reason, only the CNN model was tested on milestone 3.

For the top two models in milestone 2 (KNN and Bagging) we ran a loop to check their scores once we varied the 2 classes used exclusively for testing. We found a score difference of approximately 13% from the “best” paring to the “worst” paring, showing that the test paring had a very big impact on the score.

On milestone 3, in order to select which models we wanted to run the deeper analysis (running the loop again), we first ran a simple analysis using our standar pair (Raccoon & Hamster), this allowed us to drop random forest and extra tree models from the analysis. Both bagging and K-neighbors performed well, so we moved forward with bagging. CNN ended up making a surprise come back, that we’ll be discussing further on.
The table below has the models results overview, including the averages, max and min for the models where the loop was run.

MODEL	MILESTONE 1 SCORE	MILESTONE 2 SCORE	MILESTONE 3 (2 PAIRS) SCORE	MILESTONE 3 (3 PAIRS) SCORE
K-neighbors Classifier	61.3%	52.25%	54.17%	53.75%
Bagging Classifier	62.9%	50.83% Mean: 50.47% Std: 4.7% Min: 40.5% 25%: 47.75% 50%: 50.58% 75%: 52.50% Max: 62.33%	59.04% Mean: 52.23% Std: 3.34% Min: 43.00% 25%: 50.03% 50%: 52.65% 75%:54.40% Max: 59.71	55.11% Mean: 51.53% Std: 3.42% Min: 42.33% 25%: 49.59% 50%: 51.73% 75%: 53.89% Max: 59.91%
ExtraTree Classifier	67%	48.5%	52.04%	53.75%
RandomForest Classifier	65.60%	48.17%	51.50%	54.16%
Gradient Boosting Classifier	65.8%	47%	NA	NA
SVM kernel = “rbf”	63.5%	42.5%	NA	NA
Logistic Regression Classifier	59.6%	44.75%	NA	NA
Neural Network (Feed Forward)	61.51%	50%	NA	NA
Convolutional Neural Network (experiment 1)	74.10%	50%	NA	NA
CNN (experiment 2)	76.18%	50%	69.45% Mean:69.45% Min:50.54%	68.27% Mean: 68.27% Min: 62.61% Max: 72.74%

			Max: 79.83%	

Conclusion

After analysing the data, we decided to focus on bagging for our analysis of traditional machine learning models, since it presented the most consistently positive result.
With convolutional neural network, the results were highly influenced by the training data. We could also infer that deep learning models do not perform consistently. CNN showed good performance for milestone 1, whereas it was a bad choice for milestone 2. We could improve the performance of our models by using GPU for processing. By using GPU a huge amount of time was saved. We could infer that usage of GPU is highly advantageous in case of machine learning models. Overall, the project gave us good insights on image classification and various machine learning algorithms. Machine learning is a cumulative process, where constant learning leads to better outcomes.

References

https://adventuresinmachinelearning.com/keras-tutorial-cnn-11-lines/

https://www.researchgate.net/figure/Architecture-of-the-random-forest-model_fig1_301638643

https://www.semanticscholar.org/paper/Evaluation-of-RBF-and-MLP-in-SVM-kernel-tuned-for-Kamarulzaini-Ismail/025d65bf7f049d6f86be731b84c234ad64414d13/figure/2

https://missinglink.ai/guides/neural-network-concepts/neural-networks-image-recognition-methods-best-practices-applications/

https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529

https://www.mabl.com/blog/image-classification-with-tensorflow
http://uc-r.github.io/gbm_regression

Berhane, F. (2016). *Deep Neural Network for Image Classification: Application*. From Data Scientist : https://datascience-enthusiast.com/DL/Deep-Neural-Network-for-Image-Classification.html
Kinli, F. (2018, September). *[Deep Learning Lab] Episode-5: CIFAR-100*. From Medium.com: https://medium.com/@birdortyedi_23820/deep-learning-lab-episode-5-cifar-

100-a557e19219ba; https://nextjournal.com/mpd/image-classification-with-keras
corochann (2017): *CIFAR-10, CIFAR-100 dataset introduction*
https://corochann.com/cifar-10-cifar-100-dataset-introduction-1258.html

Mohtadi Ben Fraj(Dec 21, 2017): *In Depth: Parameter tuning for Random Forest*

https://medium.com/all-things-ai/in-depth-parameter-tuning-for-random-forest-d67bb7e920d
Aarshat Jain (Feb 21, 2016): *Complete Guide to Parameter Tuning in Gradient Boosting (GBM) in Python*

https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/

https://towardsdatascience.com/machine-learning-part-3-logistics-regression-9d890928680f

http://blog.xnextcon.com/?p=213

https://medium.com/@rrfd/boosting-bagging-and-stacking-ensemble-methods-with-sklearn-and-mlens-a455c0c982de