PROJECT REPORT

CMPE-257 - MACHINE LEARNING

Submitted By: Group 7
Team Members:
Qiao Liu - 013802893
Ching-Min Hu - 013726154
Dandan Zhao - 013795392
Fernanda Bordin - 013800638
Megha Rajam Rao - 013709488
Rajasree Rajendran - 013774358

Selected ML Algorithm: K-Nearest Neighbors
Google Colab link: https://drive.google.com/open?id=1OAd2U6eC-QisZo9bwG8GG2FUzSd9Q_Sy

Task Assignment

Task	Description	Names
Data Preparation	Load, pre-process and visualize data	Megha Rajam Rao
ML methods	KNN classifier, Bagging classifier, SVM - rbf, Logistic Regression, Random Forests, Extra Decision Tree, Gradient Boosting	Dandan Zhao CHING-MIN HU Fernanda Bordin
Neural networks	Dense(feedforward) and Densenet121 - failed experiment.	Fernanda Bordin
Neural Networks	Convolutional Neural Network - failed experiment.	Qiao Liu Rajasree Rajendran
Powerpoint presentation	Input on Neural network Input on Data preparation Input on ML algorithms Input on CNN	Fernanda Bordin Megha Rajam Rao Dandan Zhao, CHING-MIN HU Qiao Liu, Rajasree Rajendran
Report (contributors)	Input on Data Preparation Input on Neural network, overall analysis, KNN Input on ML algorithms Input on CNN	Megha Rajam Rao Fernanda Bordin Dandan Zhao, CHING-MIN HU Rajasree Rajendran, Qiao Liu

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)

Procedure

Models results overview:

MODEL	MILESTONE1_SCORE	MILESTONE2_SCORE
K-neighbors Classifier	61.3%	50.92% with 'n_neighbors': 8, 'weights': 'distance'
Bagging Classifier	62.9%	50.83% with 'bootstrap': False, 'bootstrap_features': False, 'max_samples': 0.01, 'n_estimators': 3
ExtraTree Classifier	67% with 'max_depth': 30, 'n_estimators': 500	48.5% with 'max_depth': 15, 'n_estimators': 180
RandomForest Classifier	65.60%with 'max_depth': 30, 'n_estimators': 500	48.17% with'max_depth': 15, 'n_estimators': 80
GradientBoosting Classifier	65.8%	47% with ‘learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 20
SVM kernel = “rbf”	63.5% with C=10	42.5% with 'C' : 1, 'gamma' : 0.01
Logistic Regression Classifier	59.6%	44.75% with 'C': 100, 'penalty': 'l2'
Neural Network (Feed Forward)	61.51%	50 %
Convolutional Neural Network (experiment 1)	74.10%	50%
CNN (experiment 2)	76.18%	57%-50%

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.

From the accuracy, most of the Machine Learning methods are not so good and showed little consistency between milestones.

KNN was the most consistent algorithm and had the best performance both in accuracy and running time, with **Bagging classifier** being a close second in performance. We ran a full analysis for KNN method, while exploring the most relevant aspects for Bagging.

Data Preparation:

- 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. This was helpful as the function can be separately used, to call any single combination that is of special interest. We utilized it to separately run the algorithms and fine-tune hyperparameters for the combinations with the highest accuracy, as part of an extended experimentation.

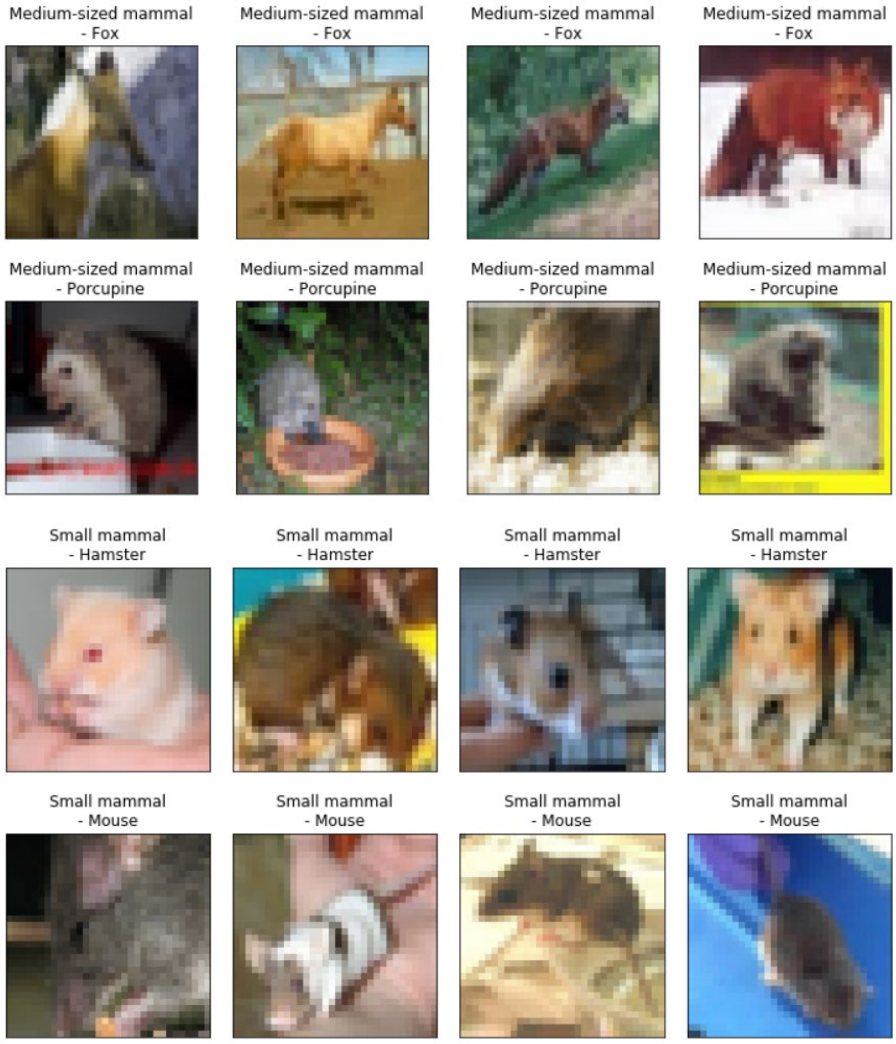
```
def train_test(small,medium): # test class for small and med
  small_mammals = ['hamster', 'mouse', 'rabbit', 'shrew', 'squirrel']
  small_mammals.remove(small)
  medium_sized_mammals = ['fox', 'porcupine', 'possum', 'raccoon', 'skunk']
  medium_sized_mammals.remove(medium)
```

```
# 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])
```

- Further, we printed out 4 random images from each subclass.



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

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 ran the Random Forest Classifier.

Random Forest:

- 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
```

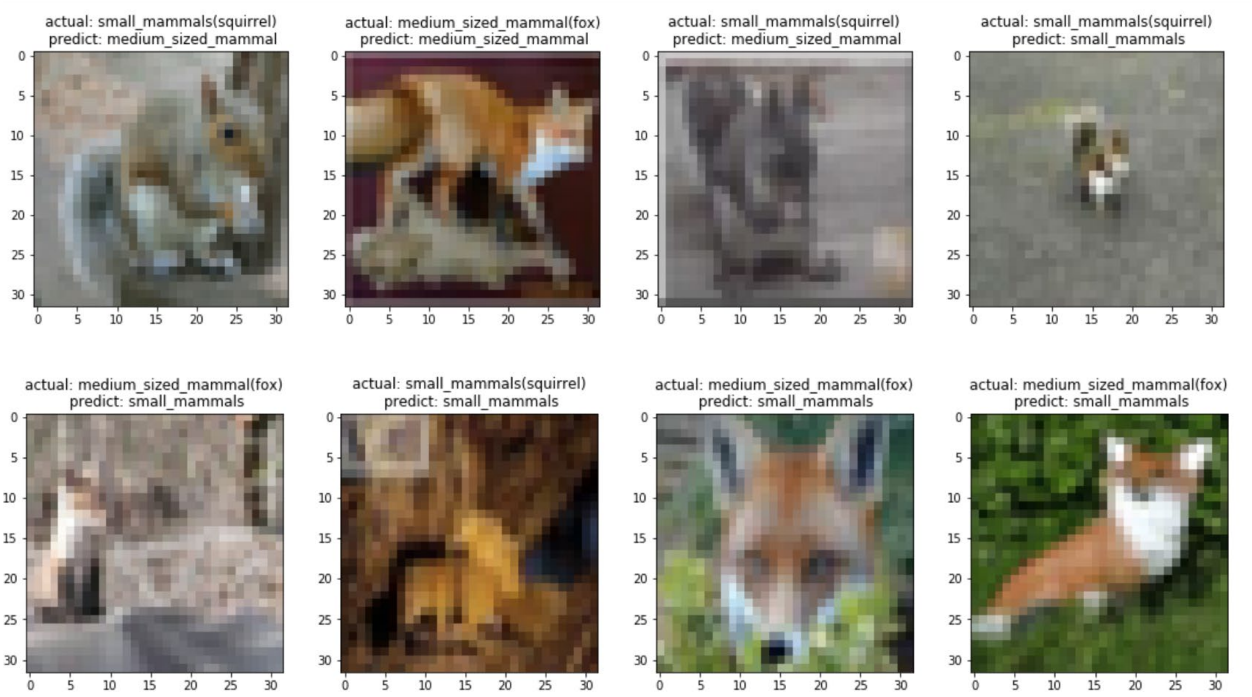
The prediction score on testing data with best estimator was found to be 48.17%.

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

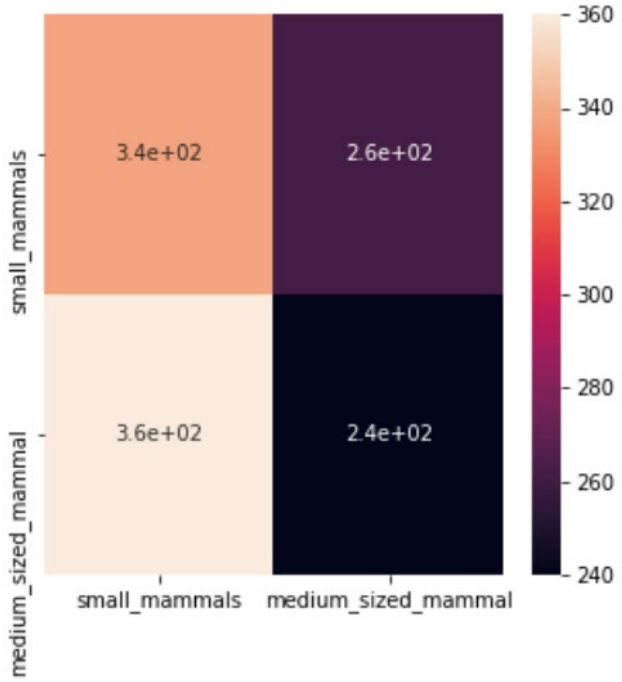
```
prediction score on testing data with the best estimator: 48.17%
```

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
```

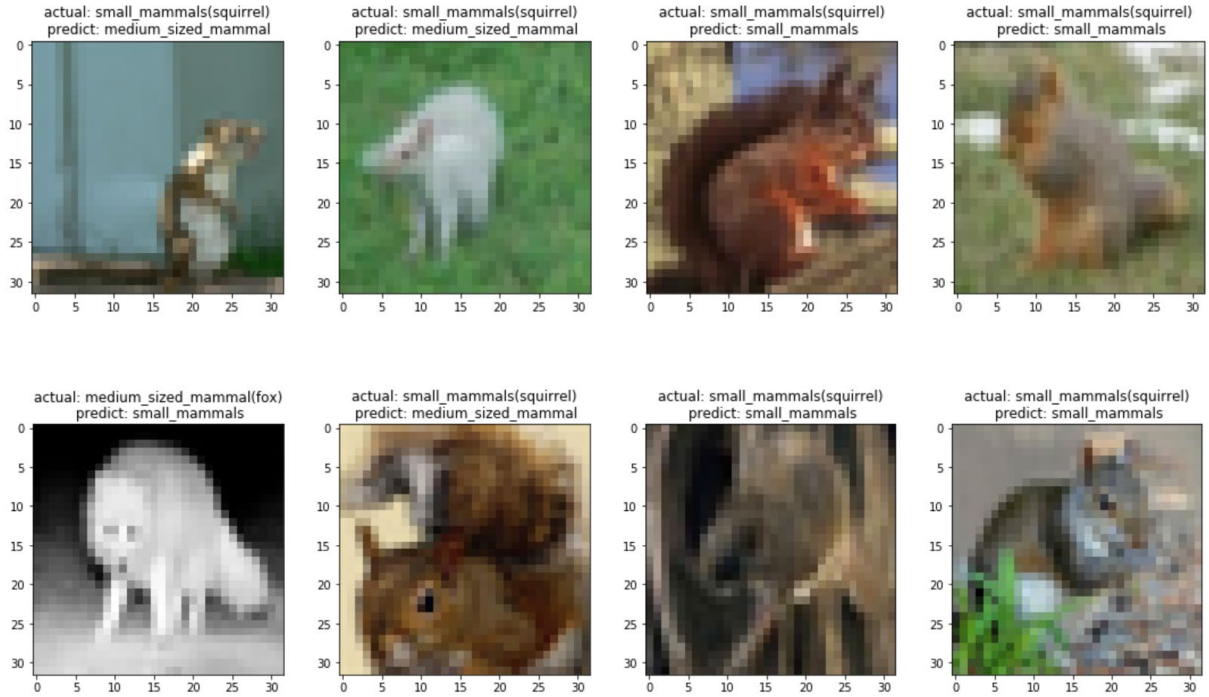
The prediction score on testing data with best estimator was found to be 48.50%.

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

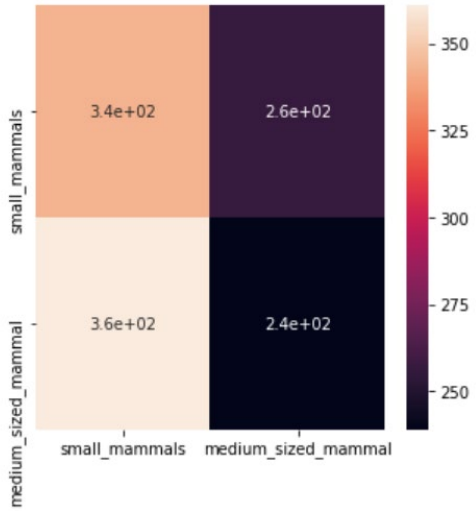
```
prediction score on testing data with the best estimator: 48.50%
```

The predicted data can be seen here:

```
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)
```



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:

- 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
```

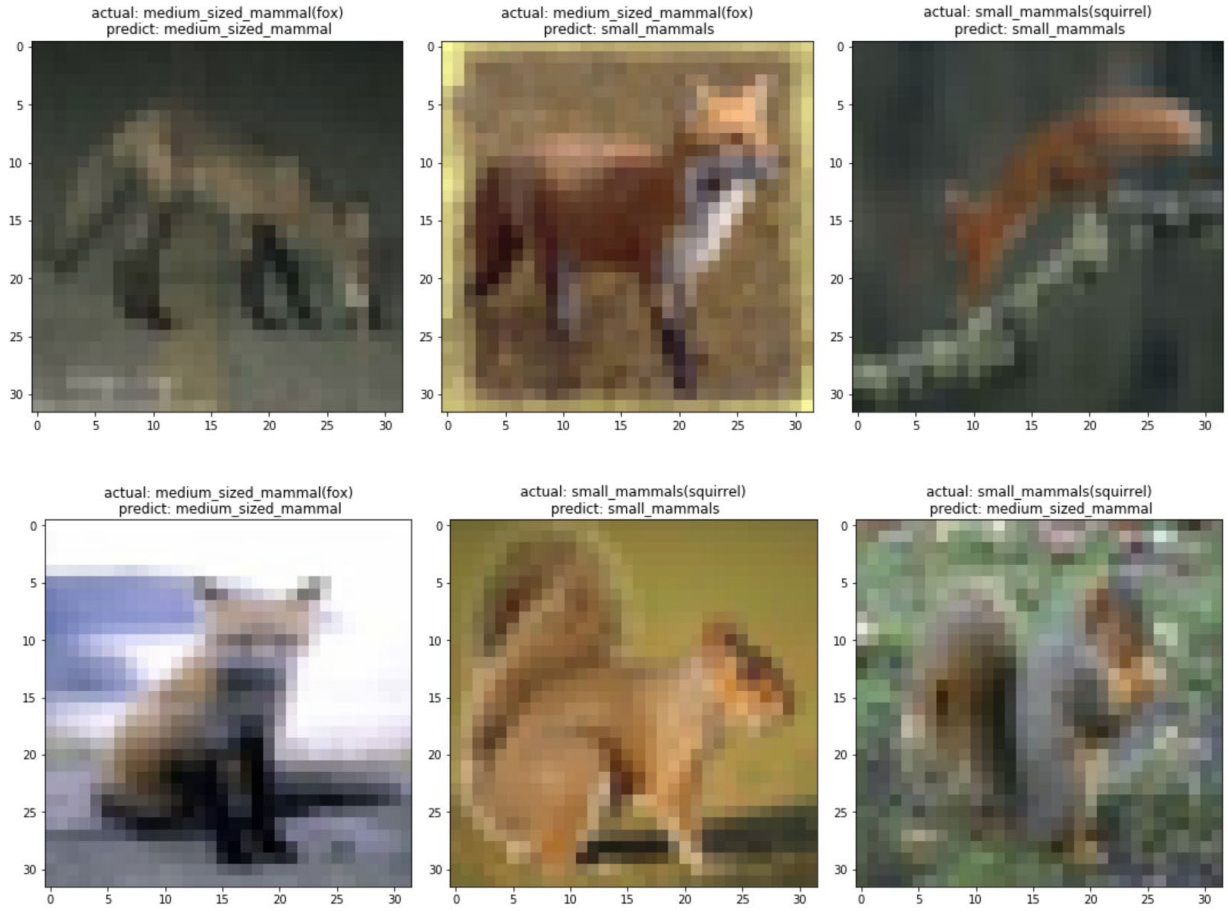
The prediction score on testing data with best estimator was found to be 47.00%.

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

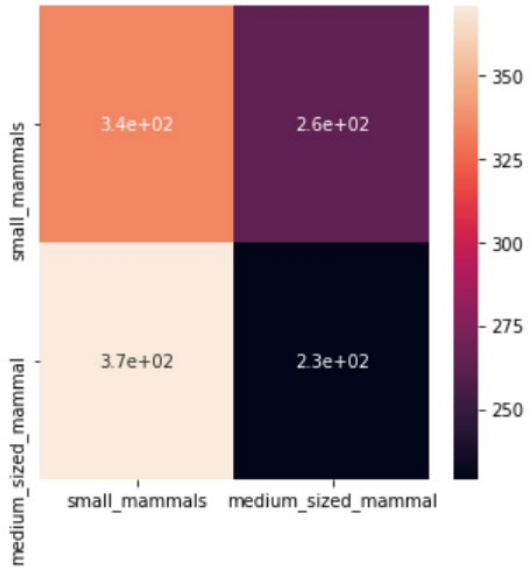
```
prediction score on testing data with the best estimator: 47.00%
```

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:

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
```

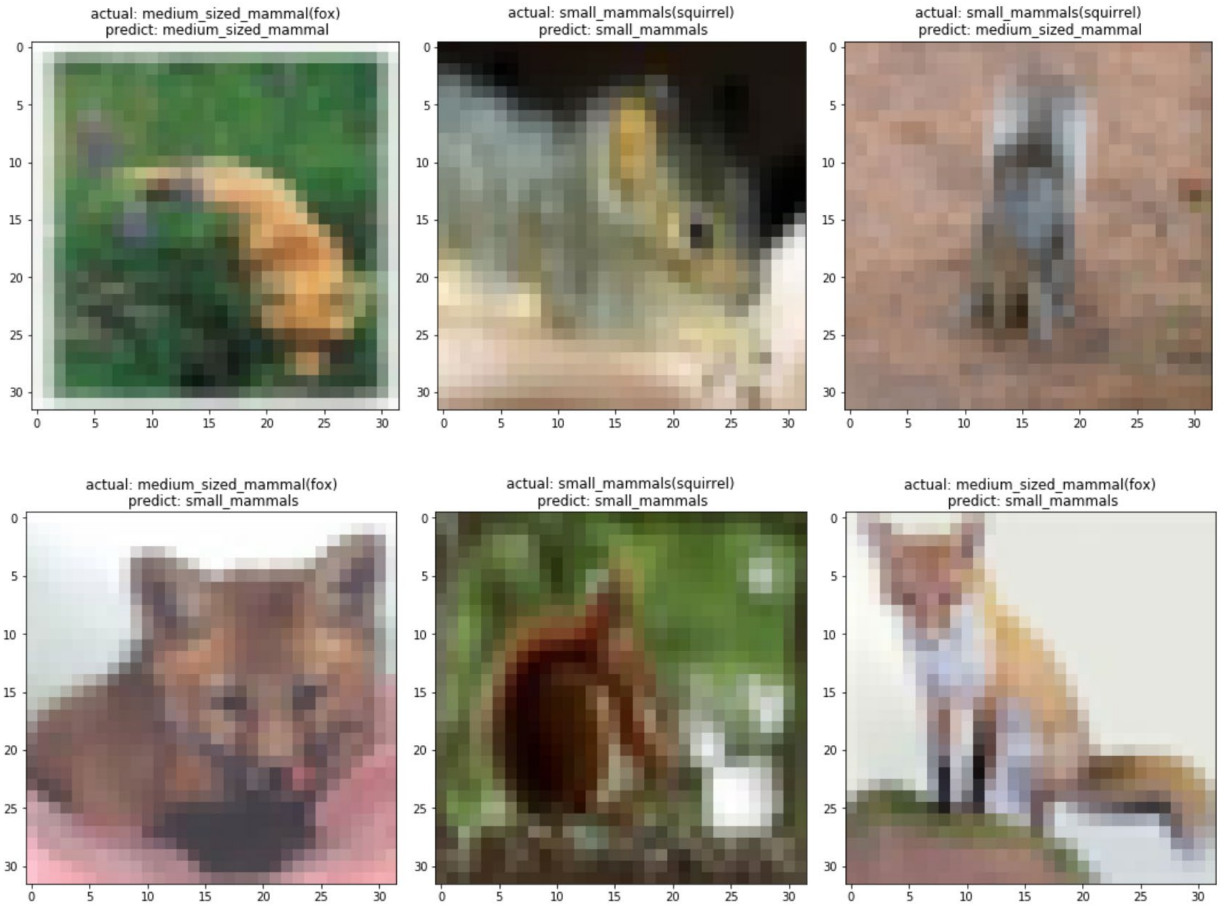
The prediction score on testing data with best estimator was found to be 42.50%.

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

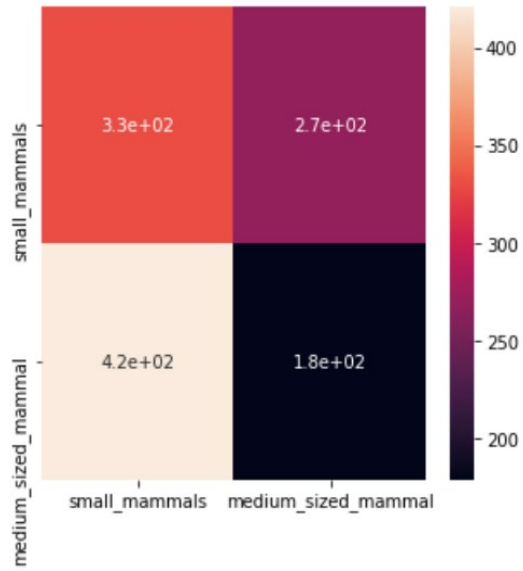
```
prediction score on testing data with the best estimator: 42.50%
```

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:

- 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%.

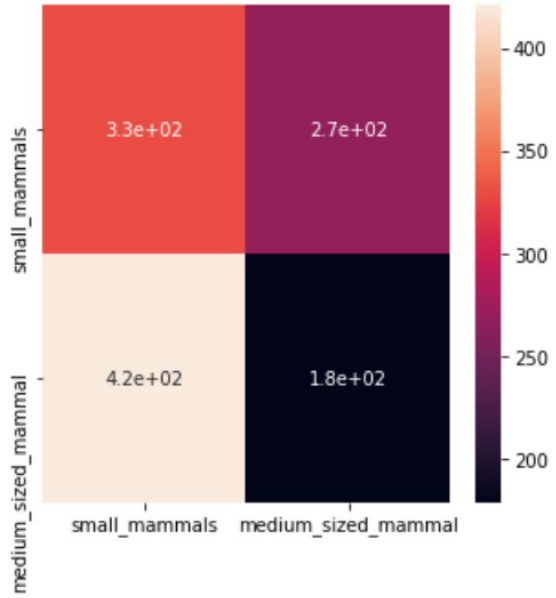
```
print ('prediction score on testing data with the best estimator: %.2f%%' % (lr_clf.best_estimator_.score(x_test2, y_test_bin)*100))
```

```
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:

```
print(classification_report(y_test_bin, lr_clf_pred))
```

```
              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-neighbors Classifier:

- 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
```

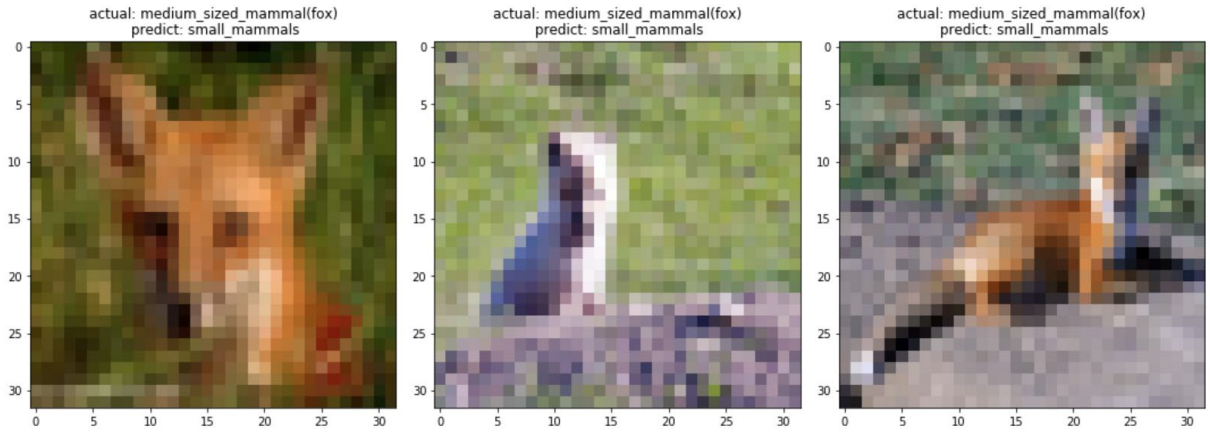
The prediction score on testing data with best estimator was found to be 50.92%.

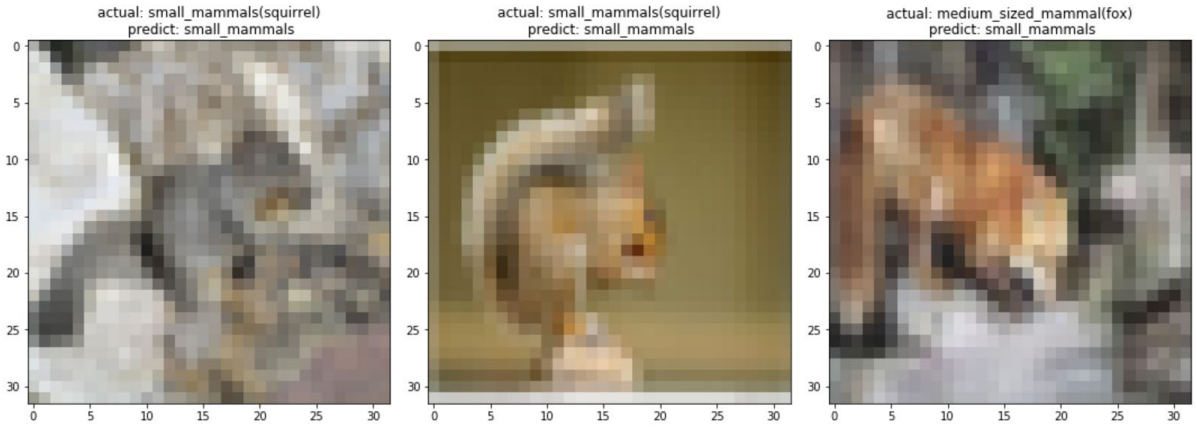
```
print ('prediction score on testing data with the best estimator: %.2f%%' % (knn_clf.best_estimator_.score(x_test2, y_test_bin)*100))
```

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

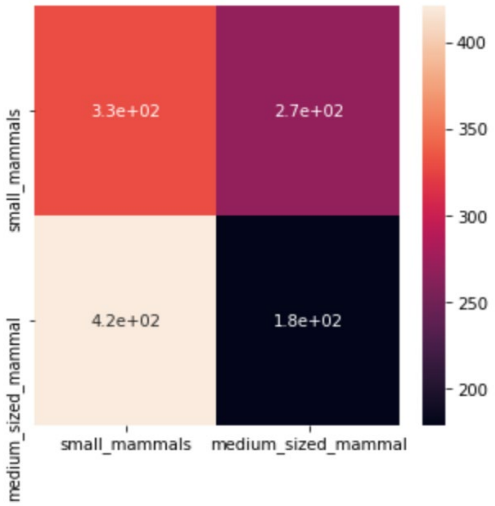
The predicted data can be seen here:

```
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. This 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
```

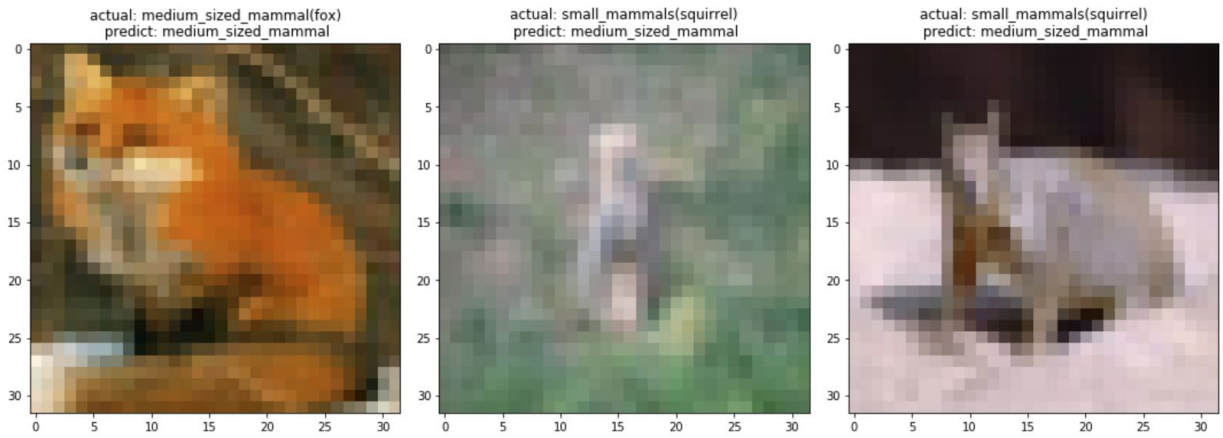
The prediction score on testing data with best estimator was found to be 50.83%.

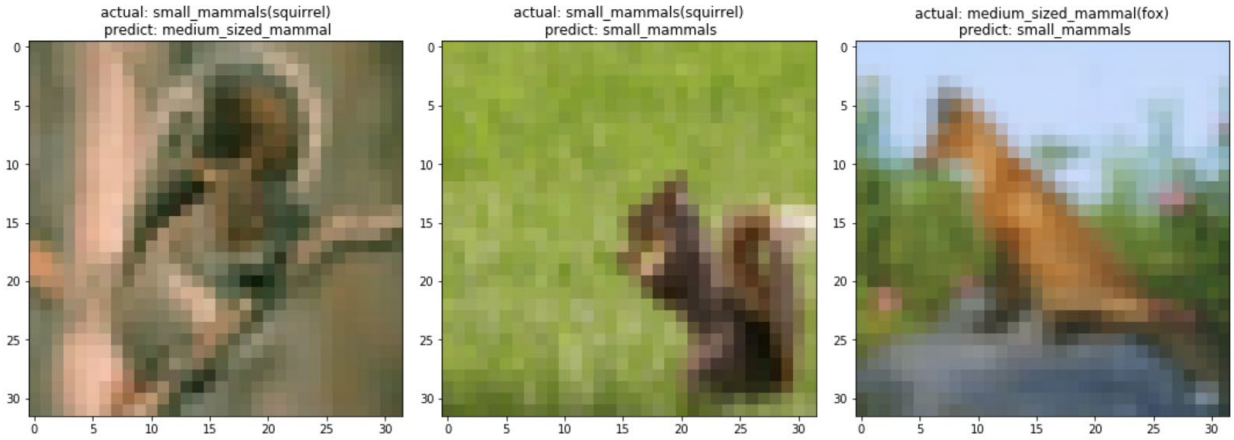
```
print ('prediction score on testing data with the best estimator: %.2f%%' % (bag_clf.best_estimator_.score(x_test2, y_test_bin)*100))
```

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

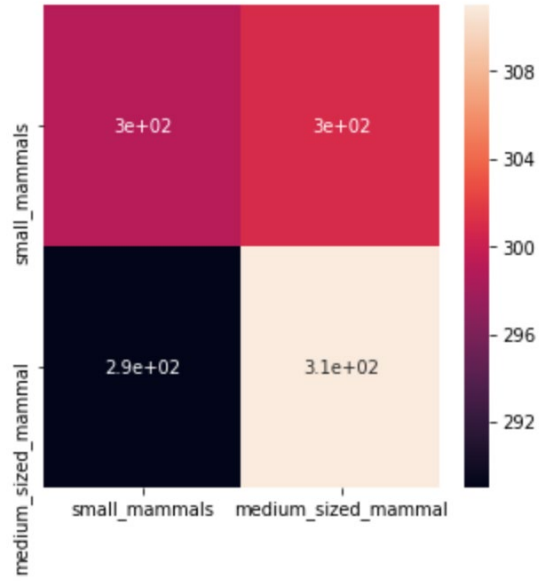
The predicted data can be seen here:

```
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

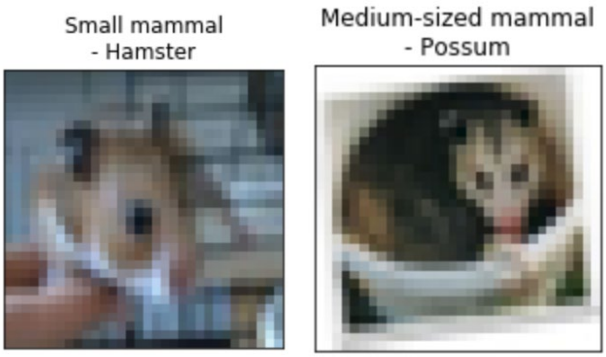
Overall analysis:

Comparing the results from K-neighbors and Bagging, some similarities come to light.

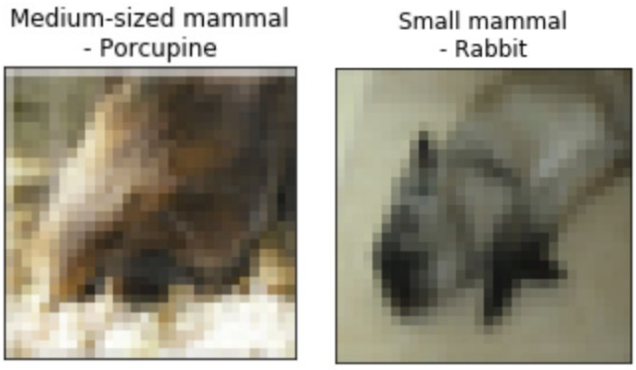
In both cases the best result was obtained with **Hamster & Possum** as the testing sample, which was not the obvious answer once you look at their pictures. Personally, we expected the

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.

Challenges faced:

The data set proved to be one of the big challenges we had to face, with some of our classes being very alike and, in some cases, hard to tell apart even for humans.
One major aspect was that pictures were not to scale, with small or medium animals both occupying the same area of the image. Another was the pixilation of picture, which drastically reduced the level of details and in some cases made it impossible to know what it was.

The failed experiments:

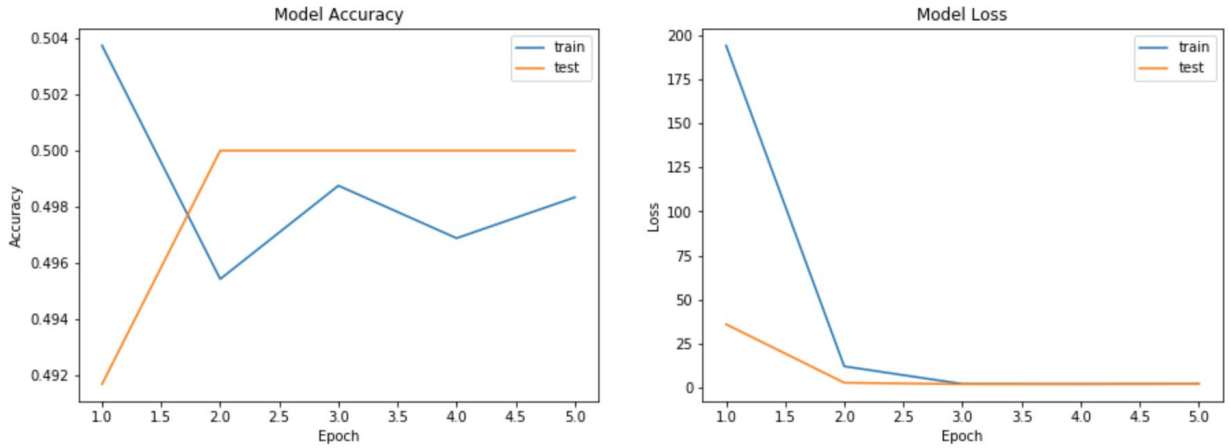
During milestone 2, we trained some other models with more than a hundred times trials totally, however, they did not reach our result of expectation. The models and the results will be shown below, some algorithms we compare their performance with the baseline (default parameters) to the best parameters after we tuned, some algorithms the best parameters performance even worse than baseline, which probably means they don't fit into this problem.

Model	**Baseline Performace**	**Score with best parameters**
Random Forest	**47.92%**	**48.17%**
ExtraDecisionTree	**49.67%**	**48.5%**
Gradient Boosting	**41.08%**	**47.00%**
SVM with ‘rbf’	**44.50%**	**42.50%**

Logistic Regression	**39.75%**	**44.75%**

Feedforward neural network:

Feedforward proved to not be an effective way to analyze this dataset. After several attempts at making the model work, it would always predict the same class delivering a misleading result of 50% accuracy.



DenseNet 121:

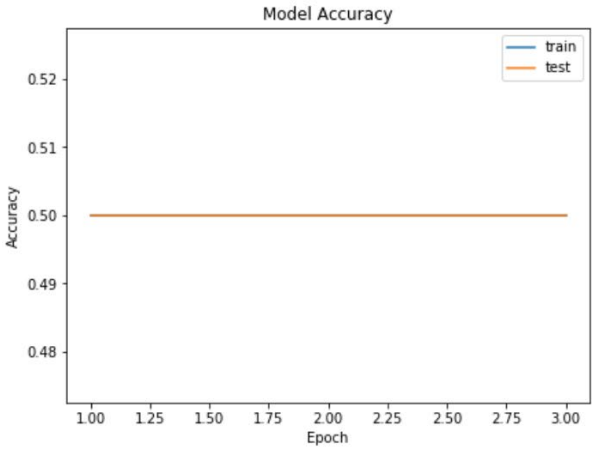
Given its successful application to other parts of this dataset (other teams), we decided to explore its effectiveness in classifying mammals. Using as a basis the code from https://www.kaggle.com/jutrera/training-a-densenet-for-the-stanford-car-dataset, we applied it to our dataset. Running time increased a lot (over 10 minutes per epoch), which was expected since it runs consecutives models and has over 47 layers.
Unfortunately, we encountered a similar result as in feed forward network, with the model predicting only one type of class and getting a 50% accuracy score.

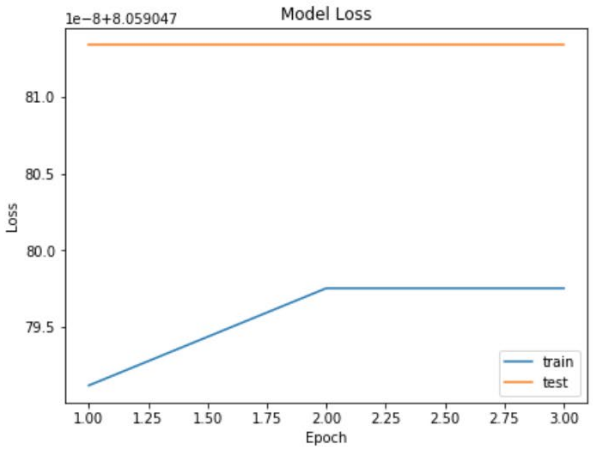
CNN:

Since we got excellent results with Convoluted neural Networks for our Milestone 1, we decided to try the same with this milestone as well. But unfortunately, we encountered the same fate as the Feedforward neural network and Dense121 network, we obtained only 50% accuracy score with the model predicting only one type of class.

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

```
CNN score: 50.00%
```





Conclusion:

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).

References

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://www.kaggle.com/jutrera/training-a-densenet-for-the-stanford-car-dataset