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 : Convolutional neural network (experiment 1)
Google Colab link : https://colab.research.google.com/drive/1xjUeQOhZvdT6f-h7MbwGkuQbAGDx8K-3

Task Assignment

Task	Description	Names
Data Preparation	Load, pre-process and visualize data	Megha Rajam Rao, Dandan Zhao
ML methods	Decision Tree, Random Forest, Extra Decision Tree, SVM with 4 different kernels, Adaboost classifier, Gradient boosting classifier, KNN classifier, Logistic classification	Dandan Zhao CHING-MIN HU
Neural networks	Dense(feedforward)	Fernanda Bordin
Neural Networks	Convolutional Neural Network	Qiao Liu Rajasree Rajendran
Powerpoint presentation	Input on Neural network Input on Data preparation Input on ML algorithms Input on CNN	Fernanda Bordin Dandan Zhao, Megha Rajam Rao Dandan Zhao, CHING-MIN HU Qiao Liu, Rajasree Rajendran
Report (contributors)	Input on Data Preparation Input on Neural network 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 superclasses (coarse labels) and corresponding classes. In this report we filtered the CIFAR-100 dataset to select images from the superclasses 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 mammals 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	SCORE	TIME TAKEN
DecisionTree Classifier	57.5%	15.55 s
Bagging Classifier	62.9%	115.31 s
ExtraTree Classifier	61.8% with default 67% with 'max_depth': 30, 'n_estimators': 500	0.84 s (default)
RandomForest Classifier	62.6% with default 65.60%with 'max_depth': 30, 'n_estimators': 500	2.03 s (default)
AdaBoost Classifier	58.3%	63.46 s
GradientBoosting Classifier	65.8%	141.87 s
SVM kernel = “rbf”	63.2% with default 63.5% with C=10	157.41 s (default)
SVM kernel = “poly”	58.5%	336.58 s
SVM kernel = “linear”	58.5%	337.63 s
SVM kernel = “sigmoid”	55.1%	125.41 s
Logistic Regression Classifier	59.6%	33.35 s
K-neighbors Classifier	61.3%	75.46 s
Neural Network (Feed Forward)	61.51%	256 s
Convolutional Neural Network (experiment 1)	74.10%	80.53 s

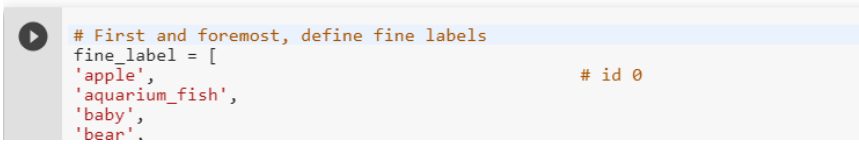
CNN (experiment 2)	76.18%	3200 s

For all these ML algorithms, we used the default values to get the result above. Then we tried to tune the different parameters in some selected algorithms, which show better accuracy, including extra decision tree, random forest, gradient boosting, SVM with 'rbf' kernel and logistic regression. From the accuracy, the following analysis we will emphasize on the ExtraDecisonTree Classifier and the SVM with the kernel "rbf", which shows a better performance in case of both accuracy and running time.
To select the best ML algorithm we decided to focus on accuracy and take into account processing time as well, comparing our results the most promising is **CNN (experiment 1**).

Data Preparation:

- In the initial step, we installed Keras API and Dill package. Keras is a neural network API written in Python and Dill can serialize and deserialize Python objects using Pickle module. Further, we imported all the necessary packages and modules and set the Tensorflow backend with 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. Thereafter, the class vectors were converted to binary class matrices using ***to_categorical()*** method.
- ***np.concatenate()*** function was used to swiftly combine the training and testing data.
- The assigned superclasses (i.e; small mammals and medium-sized mammals) were filtered after much experimentation. Initially, we tried to manually find label names but it was not a feasible solution. 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

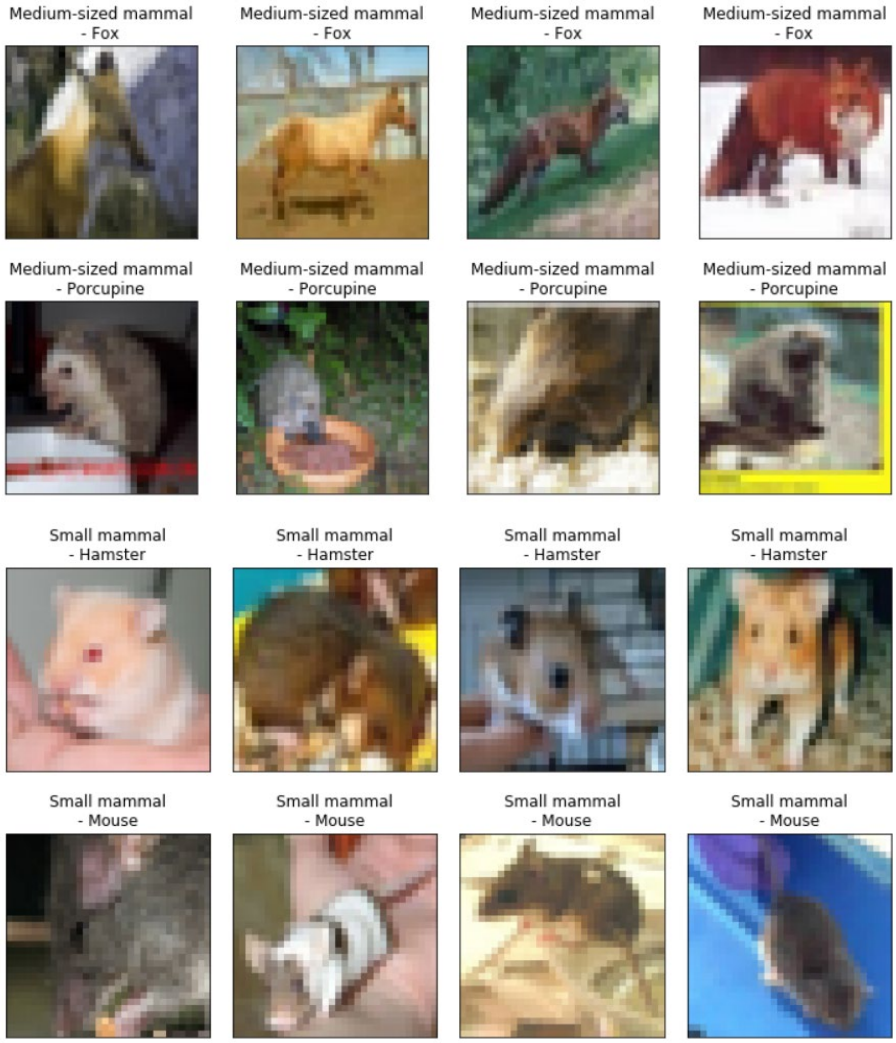


```
# First and foremost, define fine labels
fine_label = [
'apple',                                    # id 0
'aquarium_fish',
'baby',
'bear',
```

```
[ ]  # 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]
```

- 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).



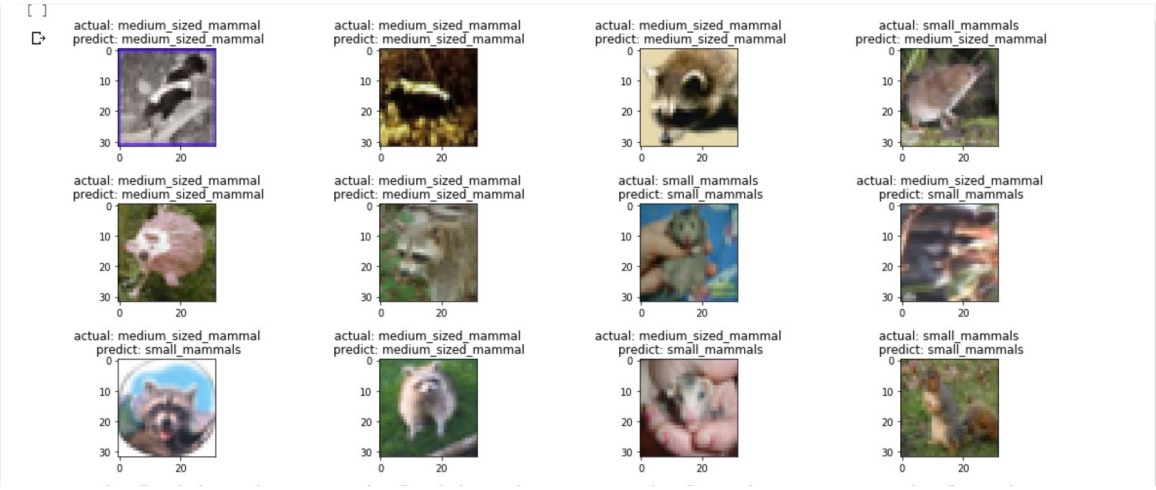
- Once the dataset was validated and ready, we proceeded ahead with the algorithms.

ExtraDecisonTree 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.
- We could generate the accuracy as **67%** with 'max_depth': 30, 'n_estimators': 500.

```
[ ]  # train the model by tuning the parameters using gridsearch, cross_validation by split the trainning data into 5
     tuned_parameters = {'n_estimators': [10,50,100,500,1000], 'max_depth': [5, 10,20,30]}

[ ]  %%time
     Extratree = GridSearchCV(ExtraTreesClassifier(), tuned_parameters, cv=5, verbose=0)
     Extratree.fit(x_train_1, y_train_bin)

     CPU times: user 29min 29s, sys: 995 ms, total: 29min 30s
     Wall time: 29min 31s
```

Results:



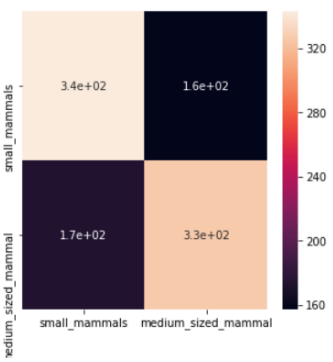
x

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 t han medium_sized_mammals.

confusion matrix:

```
[ ]  cm=confusion_matrix(y_test_bin,  Extratree_pred, labels=[0,1], sample_weight=None)
     plt.figure(figsize = (5,5))
     df_cm = pd.DataFrame(cm, index = [i for i in ['small_mammals','medium_sized_mammal']],
                          columns = [i for i in ['small_mammals','medium_sized_mammal']])

     ax=sns.heatmap(df_cm, annot=True)
```



Classification report:

```
[ ]  print(classification_report(y_test_bin, Extratree_pred))
```

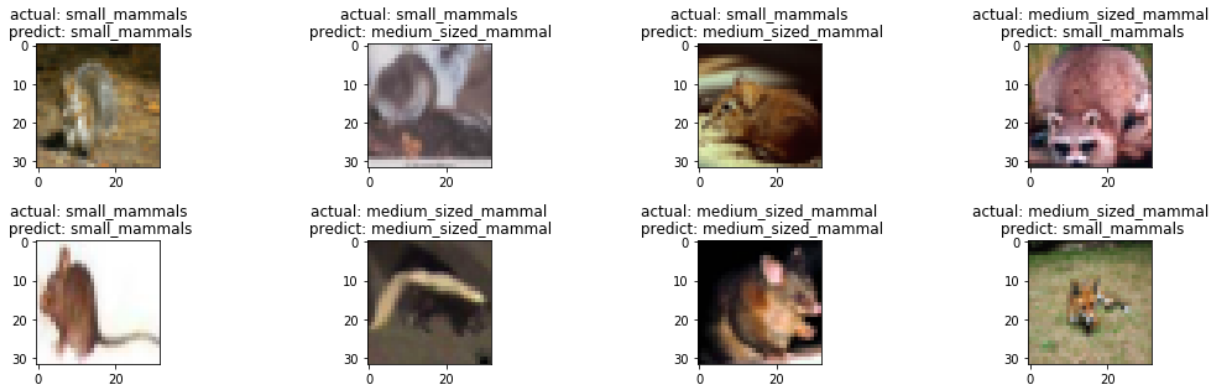
	precision	recall	f1-score	support
0	0.66	0.69	0.68	500
1	0.68	0.65	0.66	500
micro avg	0.67	0.67	0.67	1000
macro avg	0.67	0.67	0.67	1000
weighted avg	0.67	0.67	0.67	1000

SVM with RBF classifier:

- 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.

```
tuned_parameters = {'kernel': ['rbf'],
                    'C': [1, 10]}
```

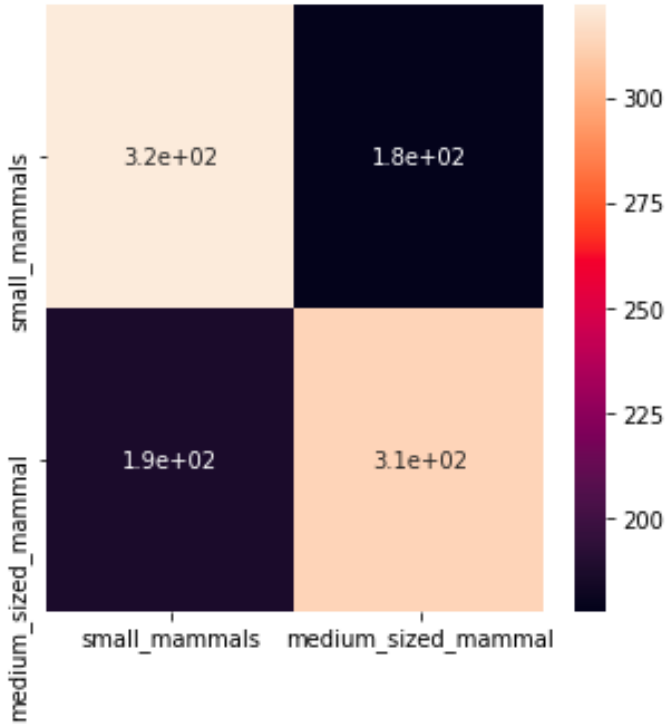
Results:



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

Lessons Learned:
The most important lesson that we learnt from this part is that data is the key, and data should be validated first and then worked on. For Milestone 2, we will improve our work and tune the value of parameters in order to obtain best performance results for each algorithms.

Neural Network for Image Recognition:

For the next step, we decided to move on to Neural Network for image classification. Neural networks mimic the functioning of human brain - by taking in raw images and processing the output directly.

- 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:

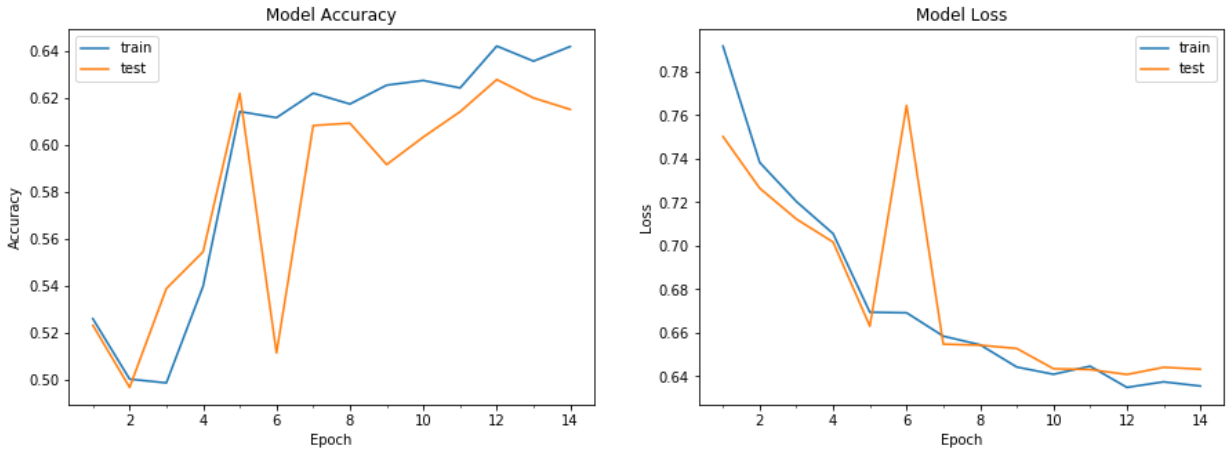
On comparing the predictions with the actual data, the score was found to be **61.51%**.

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

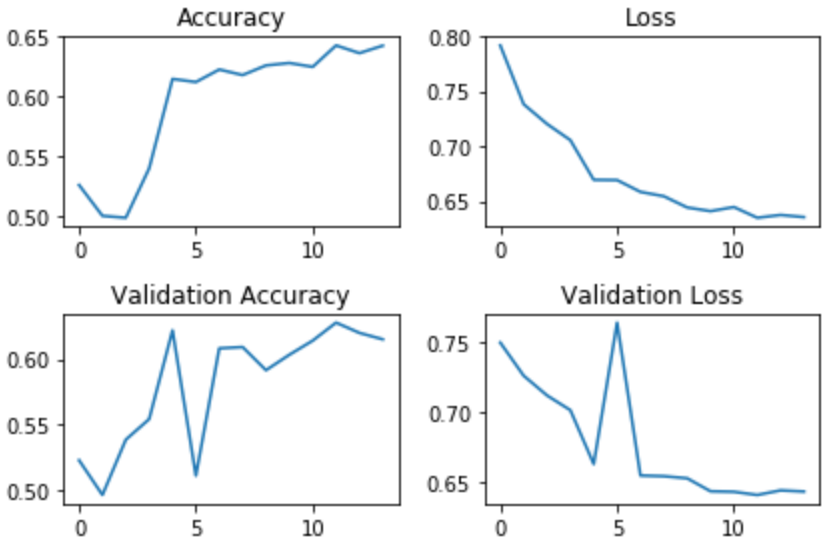
```
NN score: 61.51%
```

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. Also, CNN gives better accuracy for image classification problems.

- 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])
```

Model Evaluation:

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

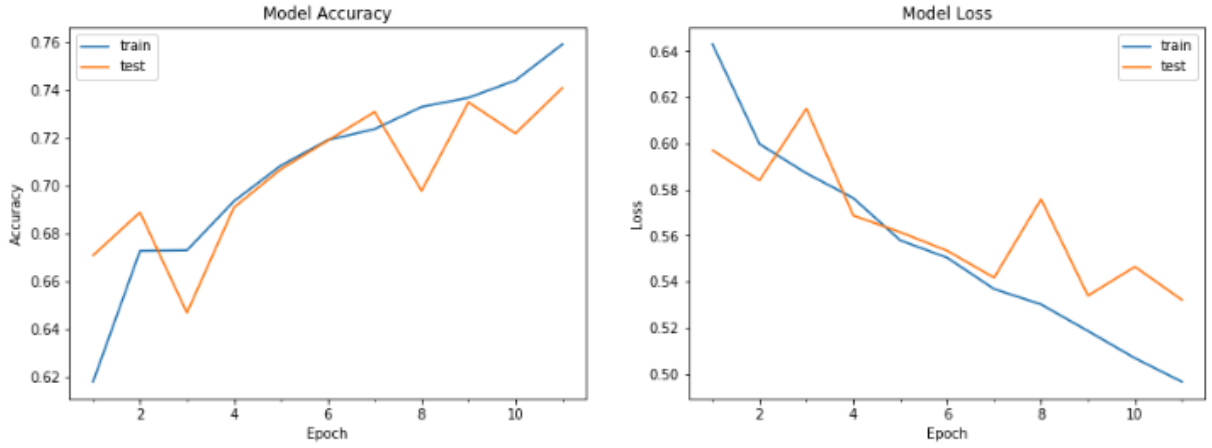
```
scores = model_cnn1.evaluate(x_test, y_test_c, verbose=0)
print("CNN score: %.2f%%" % (scores[1]*100))
```

```
CNN score: 74.10%
```

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

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

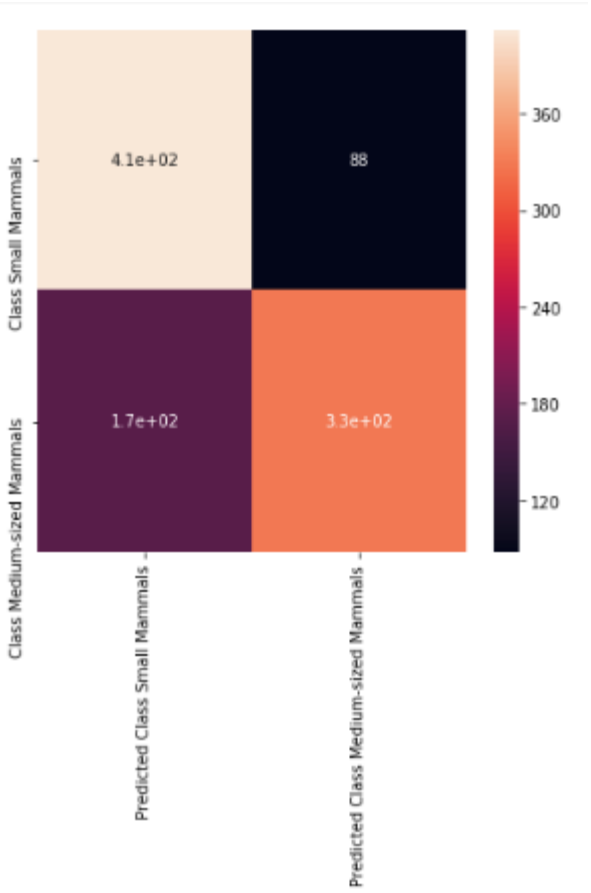
The model accuracy and loss were plotted using ***matplotlib.***



We also created the confusion matrix and classification report of the model.

Confusion Matrix:

```
                          Predicted Class Small Mammals  \
Class Small Mammals                                 412
Class Medium-sized Mammals                          171

                          Predicted Class Medium-sized Mammals
Class Small Mammals                                         88
Class Medium-sized Mammals                                 329
```



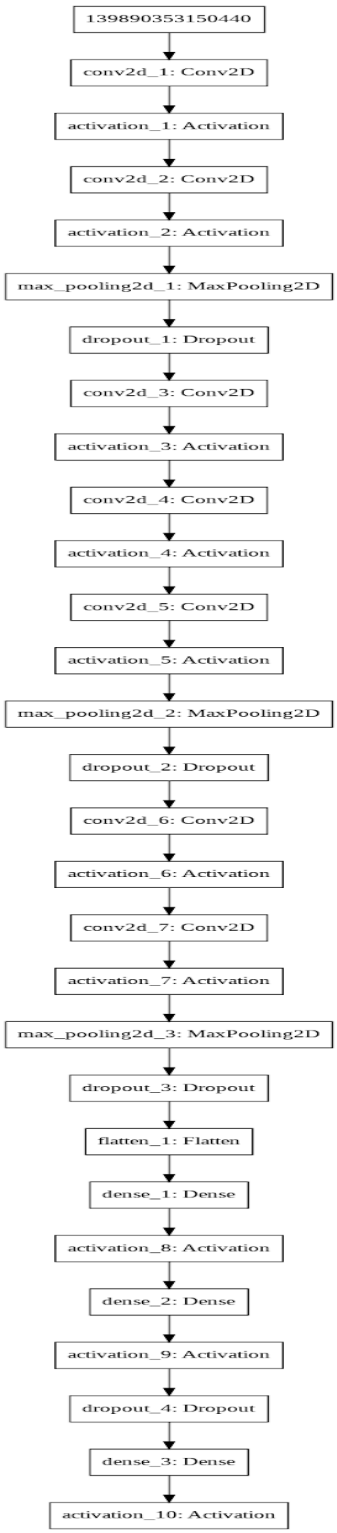
Classification Report :

	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.

Convolutional Neural Network Model 2 :

Model Architecture:



Because we were trying to do feature recognition for small pictures, we chose to use a long and narrow convolutional neural network to detect the different features for these 2 types of mammals. Training time for each epoch is around 56s. We ran a total of 76 epoches to reach the current model.

Model Evaluation:

Score:

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

```
NN score: 76.18%
```

Confusion Matrix:

	Predicted Class Small Mammals	Predicted Class Medium sized Mammals
Class Small Mammals	416	84
Class Medium sized Mammals	140	360

Classification Report:

	precision	recall	f1-score	support
Small Mammals	0.75	0.83	0.79	500
Medium sized Mammals	0.81	0.72	0.76	500
micro avg	0.78	0.78	0.78	1000
macro avg	0.78	0.78	0.78	1000
weighted avg	0.78	0.78	0.78	1000

From the classification report, we can see that ‘Medium sized Mammals’ have a higher precision than ‘Small Mammals’. This means that this model has a smaller chance of misclassifying small mammals as medium sized mammals than the other way around.
The recall scores, combining with the confusion matrix, indicates that the model is more inclined to predict an image as a small mammal.
This indicates that this model performs better at recognizing medium sized mammals.

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 one (image classification for super classes: small and medium mammals) we selected **convolutional neural network (experiment 1**) as the most promising, since it delivered a good score (74.10%) spending a reasonable amount of time processing (80.53s).

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

Brownlee, J. (2017). *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. From Machine learning mastery: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

codename, D. (2018). *Stackexchange*. From Stackexchange: https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw

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

Tutorial, T. (2018). *Explore overfitting and underfitting*. From Tensorflow Core: https://www.tensorflow.org/tutorials/keras/overfit_and_underfit