Assignment 3 Report

Course Code: CS 512
Course Name: Computer Vision
Course Instructor: Dr. Gady Agam

AS3
Prepared by: Khalid Saifullah
Student ID: A20423546
Date: 20th March, 2023

1. Problem Statement

The objective is to write 4 programs that performs the following 4 tasks:

- A basic line fitting algorithm
- Estimate line parameters from points by line fitting that is robust to outliers
- Image classification on Kaggle's Cats and Dogs dataset. Perform data augmentation and also use VGG16 pre trained convolution base
- Image classification on CIFAR10 dataset using convolution blocks, inception blocks and residual blocks

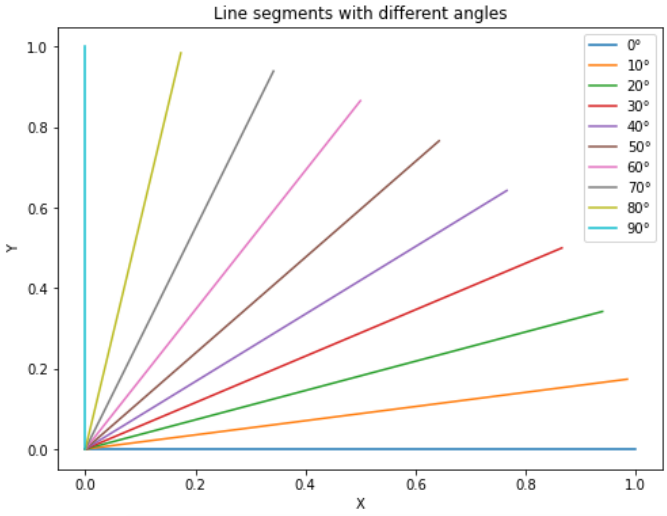
2. Proposed Solution

Keras and tensorflow packages were used along with python in google colab to preprocess the data, train the model and test it on the test dataset.

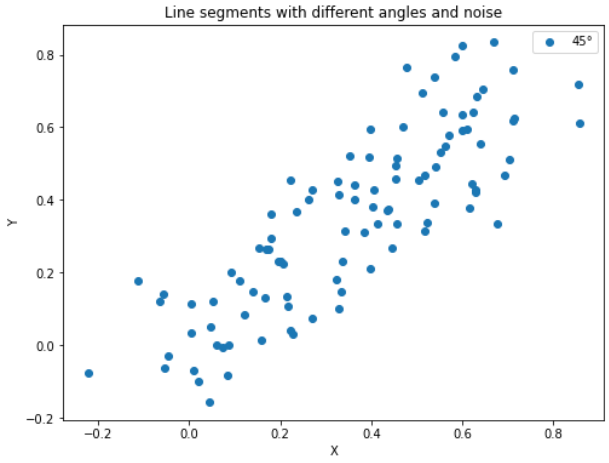
3. Implementation Details

Programming Problem 1 & 2:

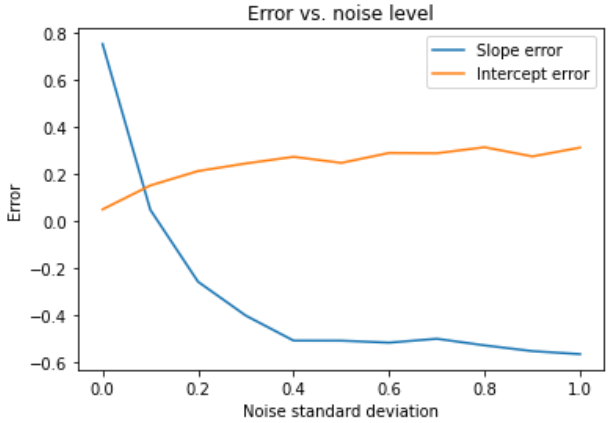
At first, some points were generated belonging to a line segment with given parameters of angle and distance from the origin.



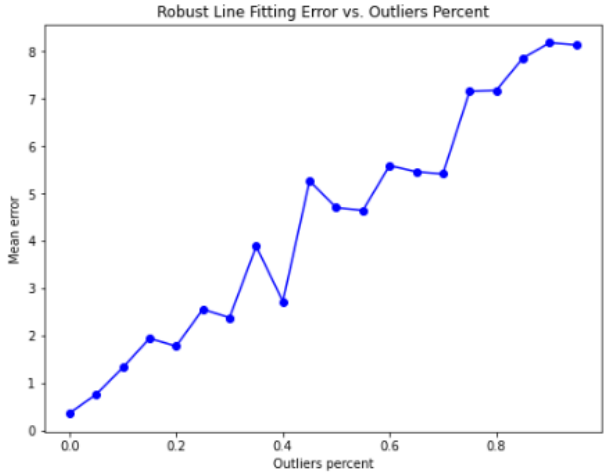
Then gaussian noise with specified mean and standard deviation to the points was generated for a 45-degree angled line segment.



The line parameters were estimated for the points above and the errors were computed for the known line parameters, which was -0.1924. Next, a graph is plotted showing error as a function of noise level.



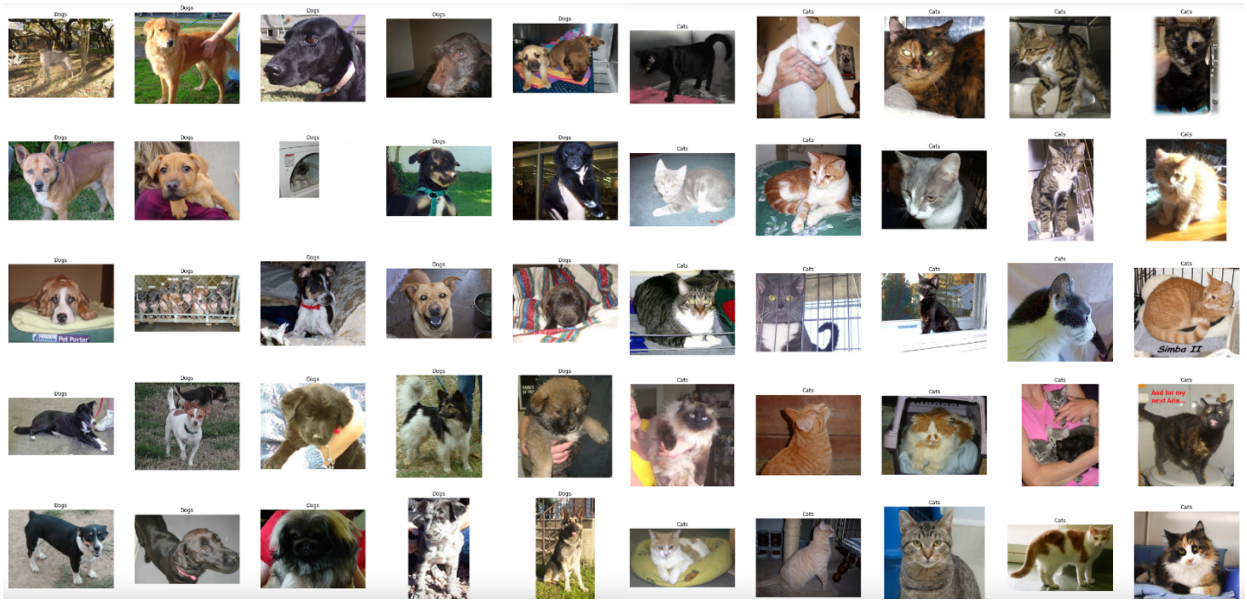
Then, a graph is plotted showing the error as a function of outliers percent.



Next, the cv2.fitLine line fitting function is used to perform robust estimation using CV_DIST_HUBER distance and a graph is plotted showing the error as a function of outliers percent.

Programming Problem 3:

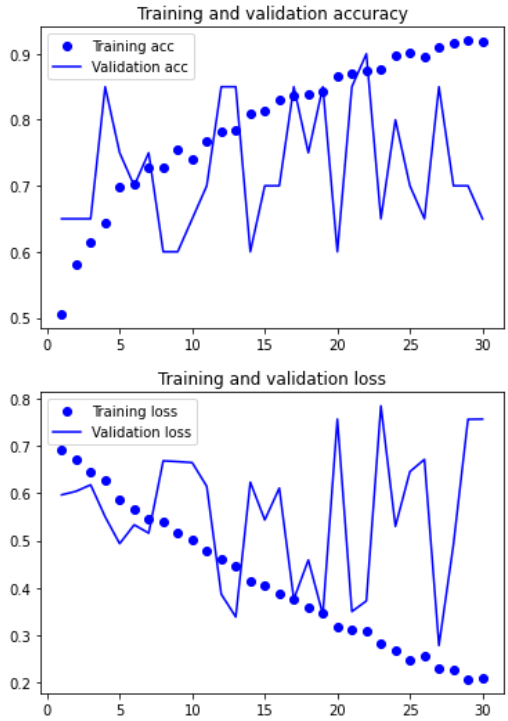
At first, the cats and dogs datasets were downloaded from the link. It had 12510 cat images and 12500 dog images. From there, a subset of 1000 cat images and 1000 dog images were put aside for training purpose, 500 cat images and 500 dog images were kept for validation and another 500 cat images and 500 dog images were kept for testing.



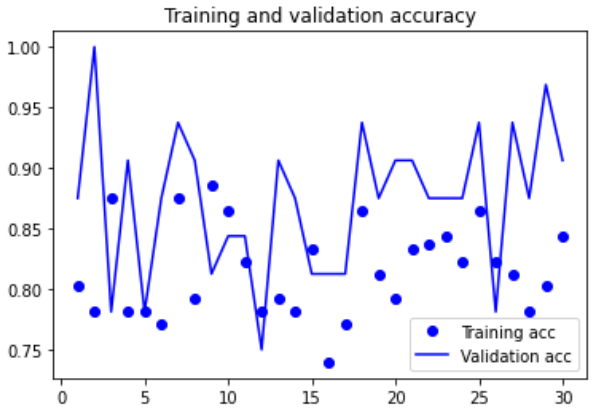
The final model architecture is defined as the following.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dropout (Dropout)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

There are 2 classes in our case. Binary cross entropy was used as the loss function, RMSprop was used as the optimizer and the metric was selected to be ‘accuracy’. The learning rate was set at 1e-4. The training and validation datasets were standardized. The model was then fitted using 100 steps per epoch and a total of 30 epochs. The performance on training and validation set is given below.

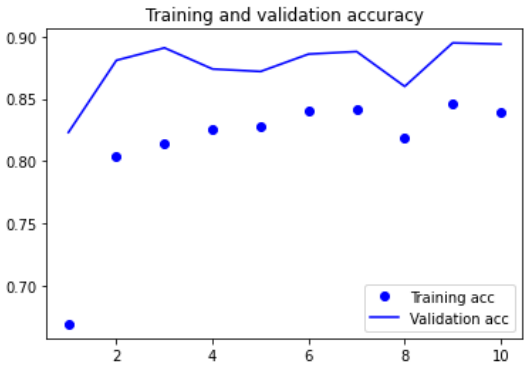


Next, the data generator was used to perform data augmentation. The performance reported is given below.



Finally, the convolution layers were replaced with pre-trained convolution base of VGG16. The VGG16 architecture consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. The convolutional layers use 3x3 filters and have a stride of 1, and are followed by rectified linear unit (ReLU) activations and 2x2 max pooling layers with a stride of 2. The fully connected layers have 4096 neurons each, followed by a final softmax layer for classification. In our case, we use the weights of imagenet problem i.e. we do not update the weights of the model with training. The number of epochs was set at 10 as it was computationally expensive to set it any higher. The performance reported is given below.

```
Epoch 1/10
63/63 [==============================] - 870s 14s/step - loss: 0.7882 - accuracy: 0.6690 - val_loss: 0.3698 - val_accuracy: 0.8230
Epoch 2/10
63/63 [==============================] - 841s 13s/step - loss: 0.4282 - accuracy: 0.8035 - val_loss: 0.2904 - val_accuracy: 0.8810
Epoch 3/10
63/63 [==============================] - 887s 14s/step - loss: 0.3975 - accuracy: 0.8140 - val_loss: 0.2642 - val_accuracy: 0.8910
Epoch 4/10
63/63 [==============================] - 889s 14s/step - loss: 0.3773 - accuracy: 0.8255 - val_loss: 0.2872 - val_accuracy: 0.8740
Epoch 5/10
63/63 [==============================] - 891s 14s/step - loss: 0.3767 - accuracy: 0.8275 - val_loss: 0.2983 - val_accuracy: 0.8720
Epoch 6/10
63/63 [==============================] - 886s 14s/step - loss: 0.3637 - accuracy: 0.8400 - val_loss: 0.2590 - val_accuracy: 0.8860
Epoch 7/10
63/63 [==============================] - 886s 14s/step - loss: 0.3508 - accuracy: 0.8420 - val_loss: 0.2528 - val_accuracy: 0.8880
Epoch 8/10
63/63 [==============================] - 850s 14s/step - loss: 0.3723 - accuracy: 0.8190 - val_loss: 0.3078 - val_accuracy: 0.8600
Epoch 9/10
63/63 [==============================] - 883s 14s/step - loss: 0.3384 - accuracy: 0.8465 - val_loss: 0.2409 - val_accuracy: 0.8950
Epoch 10/10
63/63 [==============================] - 838s 13s/step - loss: 0.3456 - accuracy: 0.8395 - val_loss: 0.2474 - val_accuracy: 0.8940
```



Programming Problem 4:

The CIFAR10 dataset is first downloaded from the link and loaded into the program. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The 10 classes are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. Some sample images are:

We built a basic convolutional neural networks with several convolution blocks, where each convolution block contains convolution, pooling, and normalization layers. Then the output of the convolution layers was flattened and passed to a single dense layer that produced the output using softmax activation. The model architecture is given below.

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_19 (Sequential)   (None, 16, 16, 32)        1024
_________________________________________________________________
sequential_20 (Sequential)   (None, 8, 8, 64)          18752
_________________________________________________________________
sequential_21 (Sequential)   (None, 4, 4, 128)         74368
_________________________________________________________________
sequential_22 (Sequential)   (None, 2, 2, 256)         296192
_________________________________________________________________
flatten_2 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                10250
=================================================================
Total params: 400,586
Trainable params: 399,626
Non-trainable params: 960
_________________________________________________________________
```

The performance evaluation on the validation set is also given below. The accuracy is around 71% for the basic convolutional neural network.

```
model.fit(X_train, Y_train, batch_size=128, epochs=10, validation_data=(X_val, Y_val)) #epochs=10 original

Epoch 1/10
313/313 [==============================] - 142s 451ms/step - loss: 1.3013 - accuracy: 0.5469 - val_loss: 2.5546 - val_accuracy: 0.2128
Epoch 2/10
313/313 [==============================] - 136s 433ms/step - loss: 0.8316 - accuracy: 0.7107 - val_loss: 1.1211 - val_accuracy: 0.6065
Epoch 3/10
313/313 [==============================] - 135s 430ms/step - loss: 0.6161 - accuracy: 0.7871 - val_loss: 0.9011 - val_accuracy: 0.6991
Epoch 4/10
313/313 [==============================] - 136s 435ms/step - loss: 0.4556 - accuracy: 0.8449 - val_loss: 0.9441 - val_accuracy: 0.7034
Epoch 5/10
313/313 [==============================] - 135s 432ms/step - loss: 0.3271 - accuracy: 0.8898 - val_loss: 0.9871 - val_accuracy: 0.7011
Epoch 6/10
313/313 [==============================] - 139s 444ms/step - loss: 0.2112 - accuracy: 0.9320 - val_loss: 0.9485 - val_accuracy: 0.7166
Epoch 7/10
313/313 [==============================] - 137s 437ms/step - loss: 0.1302 - accuracy: 0.9607 - val_loss: 0.9605 - val_accuracy: 0.7228
Epoch 8/10
313/313 [==============================] - 136s 435ms/step - loss: 0.0877 - accuracy: 0.9758 - val_loss: 0.9501 - val_accuracy: 0.7501
Epoch 9/10
313/313 [==============================] - 136s 433ms/step - loss: 0.0699 - accuracy: 0.9799 - val_loss: 1.2128 - val_accuracy: 0.7225
Epoch 10/10
313/313 [==============================] - 136s 433ms/step - loss: 0.0709 - accuracy: 0.9773 - val_loss: 1.3206 - val_accuracy: 0.7090
<keras.callbacks.History at 0x7f6b66789ee0>

[24] model.evaluate(X_test,Y_test)

313/313 [==============================] - 9s 29ms/step - loss: 1.3585 - accuracy: 0.7129
[1.358510971069336, 0.7128999829292297]
```

The convolutional blocks are then replaced with the inception blocks.

```
Epoch 1/10
313/313 [==============================] - 312s 992ms/step - loss: 1.5489 - accuracy: 0.4512 - val_loss: 1.2756 - val_accuracy: 0.5405
Epoch 2/10
313/313 [==============================] - 306s 977ms/step - loss: 1.1470 - accuracy: 0.5967 - val_loss: 1.1130 - val_accuracy: 0.6036
Epoch 3/10
313/313 [==============================] - 308s 984ms/step - loss: 0.9797 - accuracy: 0.6532 - val_loss: 1.1073 - val_accuracy: 0.6142
Epoch 4/10
313/313 [==============================] - 311s 995ms/step - loss: 0.8646 - accuracy: 0.6990 - val_loss: 0.9815 - val_accuracy: 0.6616
Epoch 5/10
313/313 [==============================] - 307s 980ms/step - loss: 0.7748 - accuracy: 0.7290 - val_loss: 1.0233 - val_accuracy: 0.6449
Epoch 6/10
313/313 [==============================] - 307s 981ms/step - loss: 0.6866 - accuracy: 0.7588 - val_loss: 0.9670 - val_accuracy: 0.6709
Epoch 7/10
313/313 [==============================] - 308s 984ms/step - loss: 0.6060 - accuracy: 0.7874 - val_loss: 1.0310 - val_accuracy: 0.6569
Epoch 8/10
313/313 [==============================] - 304s 973ms/step - loss: 0.5239 - accuracy: 0.8169 - val_loss: 1.0047 - val_accuracy: 0.6757
Epoch 9/10
313/313 [==============================] - 309s 987ms/step - loss: 0.4438 - accuracy: 0.8468 - val_loss: 1.1049 - val_accuracy: 0.6620
Epoch 10/10
313/313 [==============================] - 307s 980ms/step - loss: 0.3662 - accuracy: 0.8755 - val_loss: 1.1115 - val_accuracy: 0.6713
313/313 [==============================] - 14s 45ms/step - loss: 1.1115 - accuracy: 0.6713
Test loss: 1.1114801168441772
Test accuracy: 0.6712999939918518
```

The inception blocks are then replaced with the residual blocks.

```
Epoch 1/10
313/313 [==============================] - 627s 2s/step - loss: 5.1503 - accuracy: 0.3871 - val_loss: 2.9296 - val_accuracy: 0.1716
Epoch 2/10
313/313 [==============================] - 642s 2s/step - loss: 1.2543 - accuracy: 0.5528 - val_loss: 1.6202 - val_accuracy: 0.4571
Epoch 3/10
313/313 [==============================] - 635s 2s/step - loss: 1.0518 - accuracy: 0.6266 - val_loss: 1.2074 - val_accuracy: 0.5784
Epoch 4/10
313/313 [==============================] - 631s 2s/step - loss: 0.9041 - accuracy: 0.6819 - val_loss: 2.1502 - val_accuracy: 0.4685
Epoch 5/10
313/313 [==============================] - 625s 2s/step - loss: 0.7859 - accuracy: 0.7236 - val_loss: 1.4872 - val_accuracy: 0.5391
Epoch 6/10
313/313 [==============================] - 641s 2s/step - loss: 0.6822 - accuracy: 0.7593 - val_loss: 1.3554 - val_accuracy: 0.5675
Epoch 7/10
313/313 [==============================] - 627s 2s/step - loss: 0.5999 - accuracy: 0.7886 - val_loss: 1.1356 - val_accuracy: 0.6356
Epoch 8/10
313/313 [==============================] - 662s 2s/step - loss: 0.4971 - accuracy: 0.8272 - val_loss: 1.6183 - val_accuracy: 0.5575
Epoch 9/10
313/313 [==============================] - 680s 2s/step - loss: 0.4344 - accuracy: 0.8472 - val_loss: 1.3075 - val_accuracy: 0.6195
Epoch 10/10
313/313 [==============================] - 662s 2s/step - loss: 0.3724 - accuracy: 0.8687 - val_loss: 1.4879 - val_accuracy: 0.6128
313/313 [==============================] - 34s 107ms/step - loss: 1.4879 - accuracy: 0.6128
Test loss: 1.487900972366333
Test accuracy: 0.6128000020980835
```

4. Results and Discussion

It should be mentioned that a few issues were encountered while solving the programming questions. Firstly, within the Kaggle's cats and dogs dataset, there were 1 cat image and 1 dog image which couldn't be opened or were empty. Because of these images, the training and testing dataset were showing errors. They were eventually filtered out. Another issue faced was that tensorflow was showing an error which says that the input ran out of data. This was solved by setting the following 2 lines of code:

steps_per_epoch = len(X_train)//batch_size
validation_steps = len(X_test)//batch_size # if you have validation data

5. References

1. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf
2. https://arxiv.org/abs/1409.1556
3. https://arxiv.org/abs/1409.4842
4. https://arxiv.org/abs/1512.03385
5. https://arxiv.org/abs/1603.05027
6. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53