1.Description of Cats and Dogs dataset

The database used to train and test the system described in this project is the Cat and dogs dataset which contains 3370 of cats and dog images and labeled with 0 and 1 respectively used for training the image recognition system and another 1124 images used as test dataset.Each image is gray-level image with size 28×28, or with 784 pixel in total as the features. Some samples are shown in Fig 1.

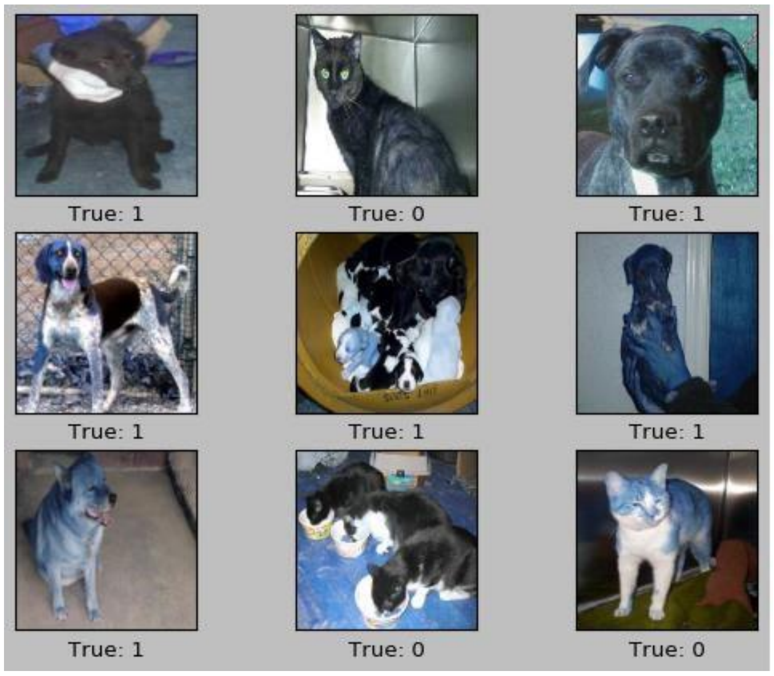


Fig 1: Sample images from the data set

2. Tools used

2.1. Tensor flow

Tensor Flow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models and it's an open source library for fast numerical computing. Tensor Flow was designed for use both in research and development production systems. It can run on single CPU systems, GPUs as well as mobile devices and large scale distributed systems of hundreds of machines.

2.2. Keras

Tensor Flow is very powerful libraries, but can be difficult to use directly for creating deep learning models.to overcome this problem I'm using Keras Python library that provides a clean and convenient way to create a range of deep learning models on top of Tensor Flow. Keras developed to make implementing deep learning models as fast and easy as possible for research and development. It runs on Python 2.7 or 3.5 and can seamlessly execute on GPUs and CPUs given the underlying frameworks. It is released under the permissive MIT license.in addition to that it allows for easy and fast prototyping through user friendliness, modularity, and extensibility. It supports both convolutional networks and recurrent networks, as well as combinations of the two.

3. Evaluation

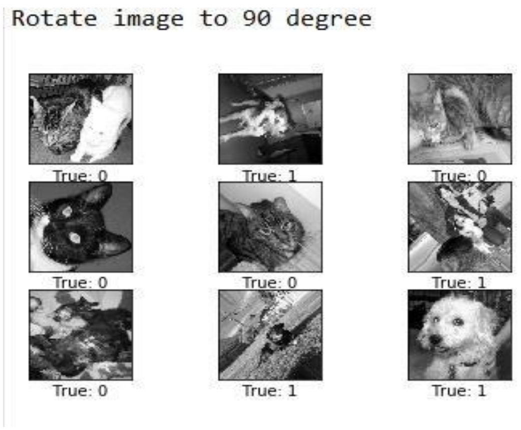
The main goal of the research project is to able to implement Convoltional neural network and get better performance at classifying the cats and dog dataset in order to do we can approach this problem in different ways one of this way is to improve performance with data set which include data augmentation and improving performance with algorithm tuning which include tuning learning rate, batch size and epochs.

3.1. Improving performance with Data augmentation

Data preparation is required when working with neural network and deep learning models. Increasingly data augmentation is also required on more complex object recognition tasks. Since we are trying to increase the performance of our model it recommended to have more data for this reason I'm using Keras image augmentation API .Keras provides the ImageDataGenerator class that defines the configuration for image data preparation and augmentation this include the capabilities such as ZCA whitening, Random rotation, Random shifts and more

3.1.1. Random Rotations

Deep learning algorithms often perform better with more data. To get more data I'm using random rotation technics by doing this we can get big gains by randomly shifting and rotating existing images and it improves the generalization of the model. As we can see below in Fig: 2 the images have been rotated left and right up to a limit of 90 .


```
Rotate image to 90 degree
```

.

Fig 2: 90 degree random rotation

3.1.2 Random Shift

Objects in our images may not be centered in the frame. They may be off-center in a variety of different ways. We can train our deep learning network to expect and currently handle off-center objects by artificially creating shifted versions of your training data. Fig 4 display shifted versions of the images.

3.1.3. ZCA whitening

Whitening transform of an image is a linear algebra operation that reduces the redundancy in the matrix of pixel images. Less redundancy in the image is intended to better highlight the structures and features in the image to the learning algorithm. Typically, image whitening is performed using the Principal Component Analysis (PCA) technique. More recently, an alternative called ZCA shows better results. Fig 5 shows sample ZCA Whitening images

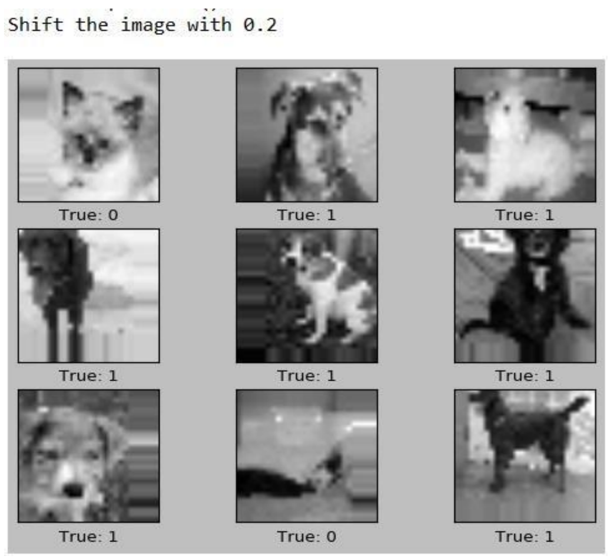


Fig 3: Sample Random shifted Images

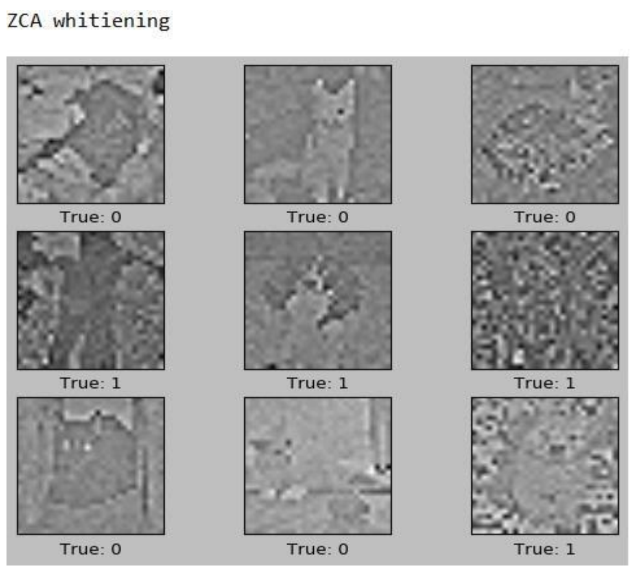


Fig 4: Sample ZCA whitiening Images

3.2. Improving performance with algorithm tuning

3.2.1. Convolutional neural network

My own implementation of Convolutional neural network architecture is composed of a convolutional layer, max poling layer and fully connected layer having a rectifier activation function used for the neurons in the hidden layer. A SoftMax activation function is used on the output layer to turn the outputs into probability-like values and allow one class of the 2 to be selected as the model's output prediction. The logarithmic loss is used as the loss function (called categorical cross entropy in Kera's) and the efficient ADAM gradient descent algorithm is used to learn the weights . And the model is fit over different epoch's values and with updates for different Bach size and drop out . The test data is used as the validation dataset, allowing us to see the skill of the model as it trains, and the images used to test the model are grayscale images with size of 28x28. Below summarizes the network architecture.

- ❖ The first hidden layer is a convolution layer called a Conv2D. The layer has 32 feature maps, which with the size of 5×5 and a rectifier activation function. This is the input layer, expecting images shape with (pixels, width, height).
- ❖ Next layer contains a pooling layer that takes the max called MaxPooling2D. It is configured with a pool size of 2×2.
- ❖ The next layer is a regularization layer using drop out called Drop out. It is configured to randomly exclude 50% ,60% and 40% of neurons in the layer in order to reduce over fitting.
- ❖ Next is a layer that converts the 2D matrix data to a vector called Flatten. It allows the output to be processed by standard fully connected layers.
- ❖ Finally, a fully connected layer with 128 neurons and rectifier activation function for fifth layers. and Below tables summarizes the result I have found during my exptrement .

Learning rate	Epochs	Batch size	Drop out	Accuracy
0.0001	**10**	**100**	**0.5**	**62.46%**
0.001	**10**	**100**	**0.6**	**62.90%**
0.01	**20**	**64**	**0.6**	**51.33%**
0.000	**100**	**200**	**0.6**	**67.35%**
0.001	**150**	**200**	**0.5**	**69.84%**
0.001	**100**	**32**	**0.4**	**67.53%**
0.0001	**100**	**32**	**----**	**69.40%**

3.3 How to reproduce the reslut

In order to the code to be work the folow ing depenceies shoud be fine

- ➢ Envaroment used for makeing th eproject visible
  - ✓ Tensorflow
  - ✓ Keras
  - ✓ python 3.5.
- ➢ Data set Can be found in here
  - ✓ https://drive.google.com/open?id=1679Bld2d8Wxii-44V9X1EgRyGzlR_8zd
- ➢ dependency used to reproduce the result
  - ✓ scikit-learn
  - ✓ matplotlib
  - ✓ opencv
  - ✓ numpy
  - ✓ tqdm (progress bar)
- ➢ detail result of my experment Result can be found .html format
  - ✓ https://drive.google.com/open?id=1y70DUnYaJwbANFLAkaH-Igz19A1_y2Qb