**Dogs and Cats Classifier using Deep Learning**

**INTRODUCTION**

Our project revolves about using deep learning techniques to classify the images of Cats and Dogs. Our classification system uses a Convolutional Neural Network (CNN), which takes an image as input, assigns weights and biases to each aspect of the image, and then uses that information to classify the image. On a very high level, for each image, the prediction is compared with its existing label, and the error between the prediction and the truth is computed. By modifying the parameters of the network, the error is minimized via backpropagation, thus increasing the prediction ability of the network. The main objective here is to teach the model the various distinguishing features of cats and dogs. Upon completion of the training of the model, it will be able to differentiate between images of cats and dogs. We have used the Keras library for training the classifier.

We are also exploring the state-of-the-art CNN models to examine the challenges involved in assembling a series of CNN layers to perform a specific function using a data pipeline. The individual layers each perform distinct tasks, so in order to develop a real-world CNN application, it is often necessary to experiment before finding the ideal layer combination.

**METHODS**

***Data***

Data for training the classifier was taken from [Kaggle](https://www.kaggle.com/competitions/dogs-vs-cats/data), which has 25,000 images of dogs and cats in the training archive. This is a binary classification problem where the response variable is 0 and 1, i.e, 0 for cats and 1 for dogs.

***Architectural Design***

Majorly four distinct kinds of layers are used in our project, i.e, Convolutional, Batch Normalization, Max Pooling, and the Dropout layers.

**Convolutional Layers:** The major component of convolutional neural networks are their convolutional layers. Convolutions are simple operations that result in the activation of an input when a filter is applied. Filters are a set of learnable weights which are learned using the backpropagation algorithm. Each pixel produces its own weights which act as image filters. Weights are small matrices produced with each pixel. As each filter in a layer creates a new image, the new image is combined and then passed through each neuron in the next layer and so on until the end of the network is reached. Filters are used to detect patterns such as edges and repeating the same filter on an input produces a feature map that indicates the location and strength of features from an input, such as edges and corners.

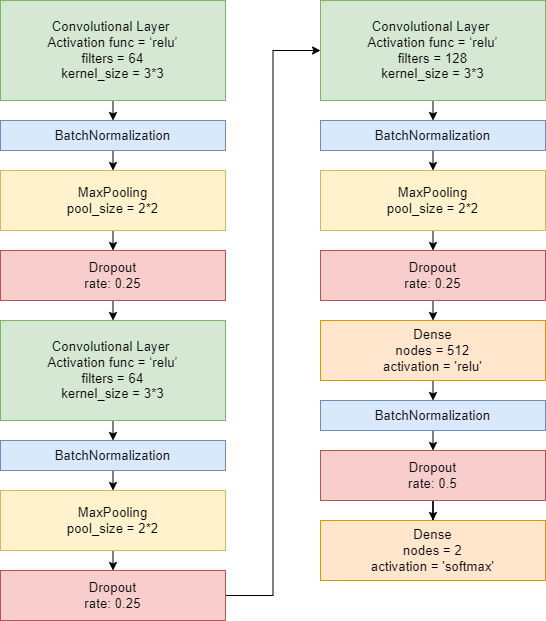
**Batch Normalization Layers:** The usage of Batch normalization is based on the concept of Normalization used in machine learning.Normalization is a preprocessing technique that standardizes data. In other words, different features of data that have different ranges are converted onto the same range scale. By not normalizing the data before we train, we affect the learning speed of the network and make it more difficult to train. A batch normalization layer allows the learning process to happen on every layer of the network independently. The output of the previous layers are normalized by this layer. When batch normalization is used, learning becomes efficient. It can also be used as a regularization method to prevent overfitting. Adding the layer standardizes the inputs or outputs of the sequential model. This can be done at many points in the model. It is often placed just after the convolution and pooling layers and just before the sequential model.

**Max Pooling Layers:** Pooling layers perform similar functions to convolutional layers, but they take specific values, such as the maximum value in a certain region or the average value in a region. These are used to reduce the dimensionality of the output of a specific layer. This helps the algorithm to actually look for relevant information such as objects in the images rather than just whitespace/backspace or the picture background. Thus, a feature map containing the most prominent features of the previous feature map would be the output after max-pooling.

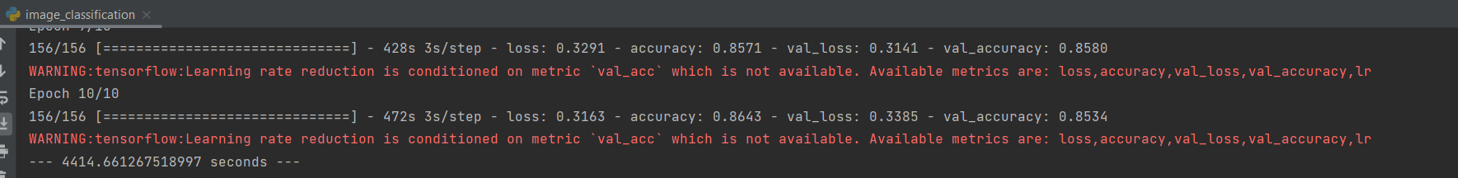
**Dropout Layers:** These layers are another common characteristic of CNNs. Dropout is a mask that nullifies the contributions of some neurons towards the next layer and leaves the others unmodified. Dropout layers can be applied to an input vector, in which case they nullify some of its features; however, they can also be applied to hidden layers, in which case some of their neurons are nullified. In CNN training, dropout layers prevent overfitting of the training data. The first set of training samples will disproportionately affect the learning if they are not present. As a result, later samples and batches will no longer be able to learn features that appear only later.

***Project Architectural Diagram***

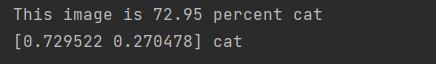
This shows the CNN architecture and hyperparameters for each layer used in the network.

****RESULTS**

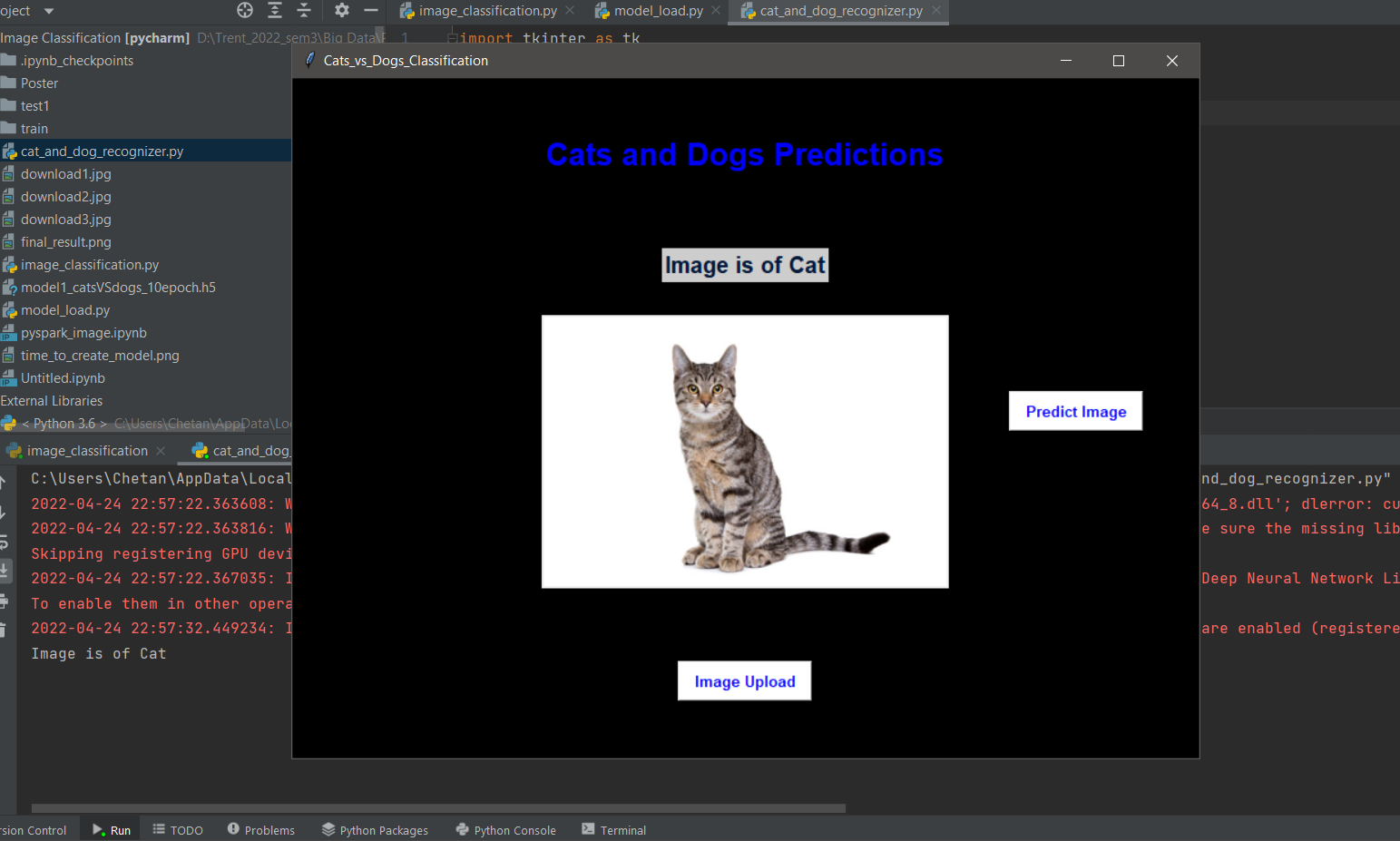
It took 4414 seconds to train the model on our system using Keras with python. Our system configuration is Intel i7-8750H CPU @ 2.20GHz 2.21 GHz, 24 GB RAM. Operating System used is Windows 10. We tried to predict an image of a cat. Our model is able to predict this image accuracy with 72.95 percentage of accuracy. Using our training model we also created a frontend using the ‘tkinter’ library. After saving our training model we load it and use it to predict the image after loading it.



Training Model Time



Prediction percentage of testing data.



Frontend using ‘tkinter’ using training data to predict the image of the uploaded file.

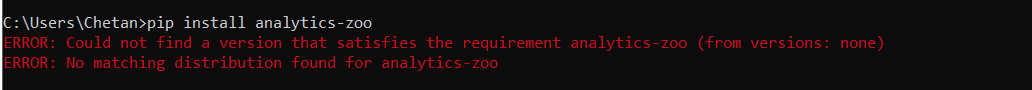
**CONCLUSION**

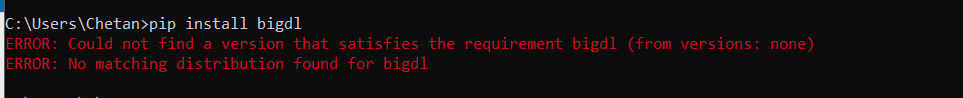
We were able to train our model in 4414 seconds. Our model is able to classify cats and dogs successfully. We tried to predict an image of a cat. Our model is able to predict this image as a cat with an accuracy of 72.95 percentage of accuracy.

**PYSPARK INSTALLATION AND SETUP OBSTACLES**

We wanted to create the same model as created above using pyspark as well but due to system configuration problems and library issues we were not able to create models in pyspark.

We were able to install pyspark and its related dependencies but unfortunately, we were not able to install libraries required to create machine learning models like analytics-zoo and bigdl. Analytics Zoo is an open-source Big Data AI platform and includes the following features for scaling end-to-end AI to distributed Big Data. We were not able to find the version that satisfies the requirements. Therefore, we could not able to compare the time to train the model and its performance with pyspark from python.





**REFERENCES**

1. <https://www.simplilearn.com/tutorials/deep-learning-tutorial/guide-to-building-powerful-keras-image-classification-models>
2. <https://keras.io/examples/vision/image_classification_from_scratch/>
3. <https://www.section.io/engineering-education/image-classifier-keras/>
4. <https://stackabuse.com/image-classification-with-transfer-learning-in-keras-create-cutting-edge-cnn-models/>