**Problem statement:** **Pet Classification Model Using CNN.**

Description

**Project Objective:**

Build a CNN model that classifies the given pet images correctly into dog and cat images.   
The project scope document specifies the requirements for the project “Pet Classification Model Using CNN.” Apart from specifying the functional and non-functional requirements for the project, it also serves as an input for project scoping.

**Project Description and Scope:**

You are provided with a collection of images of pets, that is, cats and dogs. These images are of different sizes with varied lighting conditions and they should be used as inputs for your model.

You are expected to write the code for CNN image classification model using TensorFlow that trains on the data and calculates the accuracy score on the test data.

**Project Guidelines:**

Begin by creating the ipynb file in the same parent folder where the downloaded data set is kept. The CNN model should have the following layers:   
● Input layer   
● Convolutional layer 1 with 32 filters of kernel size[5,5]   
● Pooling layer 1 with pool size[2,2] and stride 2   
● Convolutional layer 2 with 64 filters of kernel size[5,5]   
● Pooling layer 2 with pool size[2,2] and stride 2   
● Dense layer whose output size is fixed in the hyper parameter: fc\_size=32   
● Dropout layer with dropout probability 0.4   
Predict the class by doing a softmax on the output of the dropout layers.   
This should be followed by training and evaluation:   
● For the training step, define the loss function and minimize it   
● For the evaluation step, calculate the accuracy   
Run the program for 100, 200, and 300 iterations, respectively. Follow this by a report on the final accuracy and loss on the evaluation data.

Created By – Selpa Singh

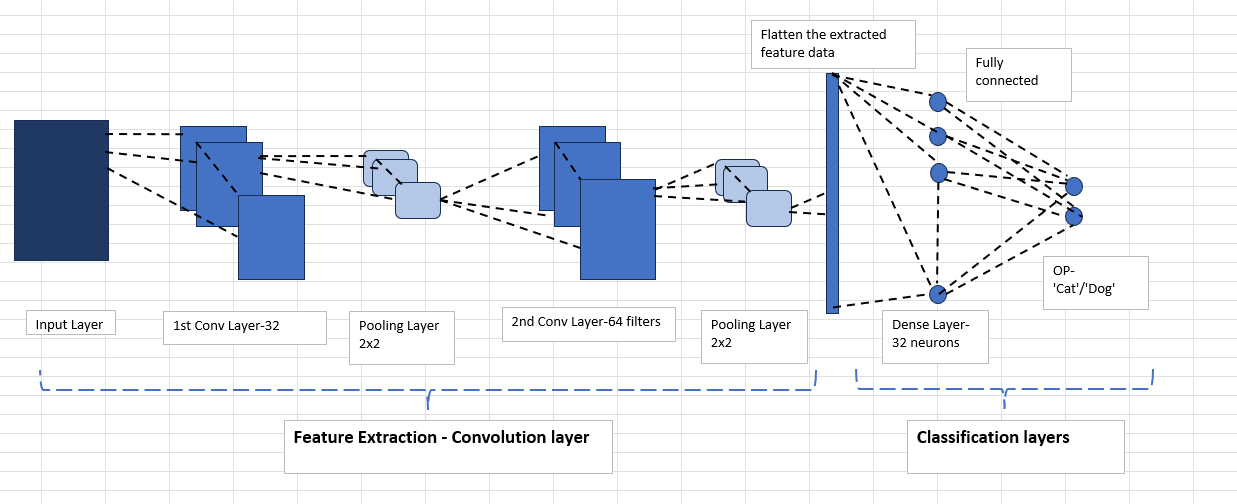
Initial understanding of the project and approach:

We need to create a CNN model which can read the images from given dataset and classify them to “Cat” and “Dog” category.

Our problem statement requires us to create a CNN model, which would be the suitable choice, considering its architecture and efficiency on Image dataset. So we will have 2 Convolution layers for feature extraction and Dense layer for Classification purpose. Based on the given information we need to create an architecture which will have below mentioned layers

1. Conv2D with 32 filter
2. Pooling Layer with size 2x2
3. Conv2D with 64 filter
4. Pooling Layer with size 2x2
5. Flatten Layer to be fed into Dense layer
6. Dense Layer with 32 neurons
7. Dropout layer with 40% (applied on Dense layer)
8. Final Output/Classification

So, our architecture is going to look like below image

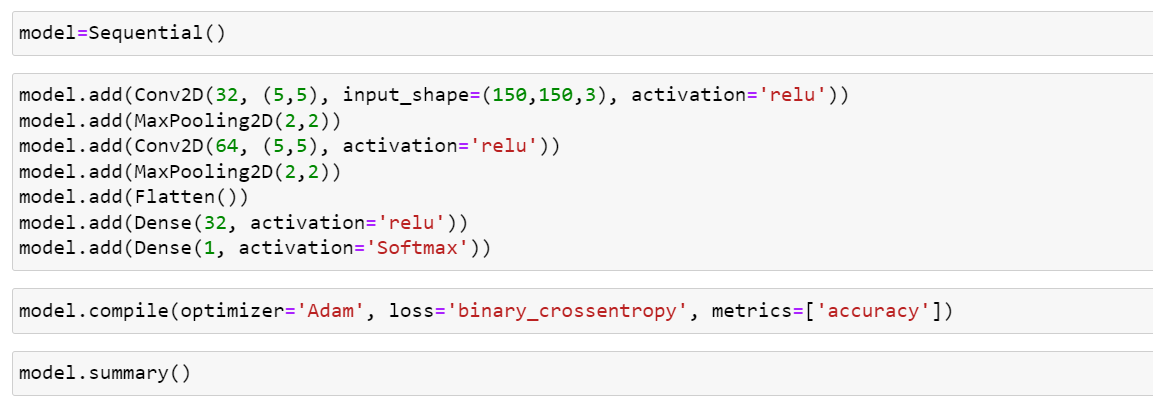


Dataset Analysis- Inferential Analysis:

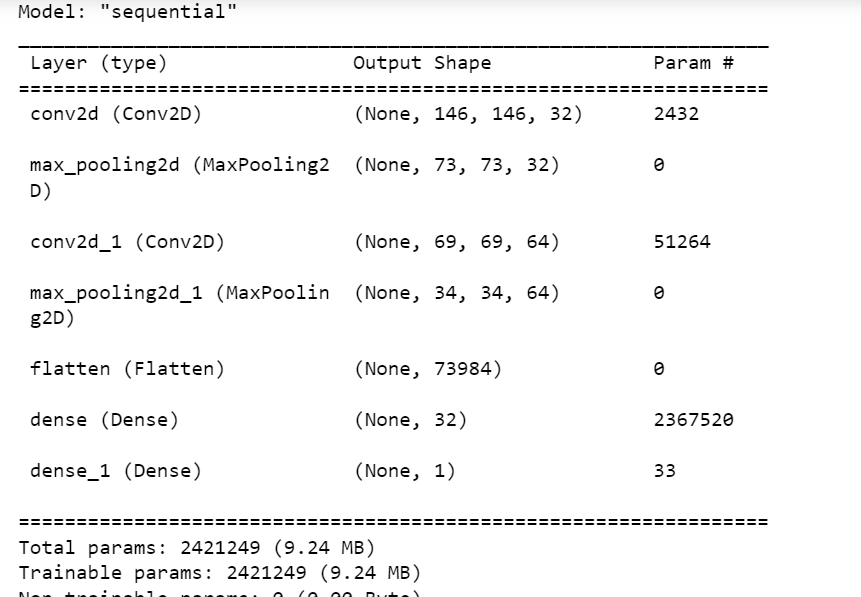
Looking at the Dataset we have 2 folders named “train” and “test”, each of these folders have sub folder named “cats” and “dogs”. In the train set we have 20 images of Cats and 20 images of Dogs. And for validation purpose we have 10 images of cats and 10 images of dogs. All the images have “\*.jpg” extension, so while extracting the image dataset we should look for the files with extension “\*.jpg”.

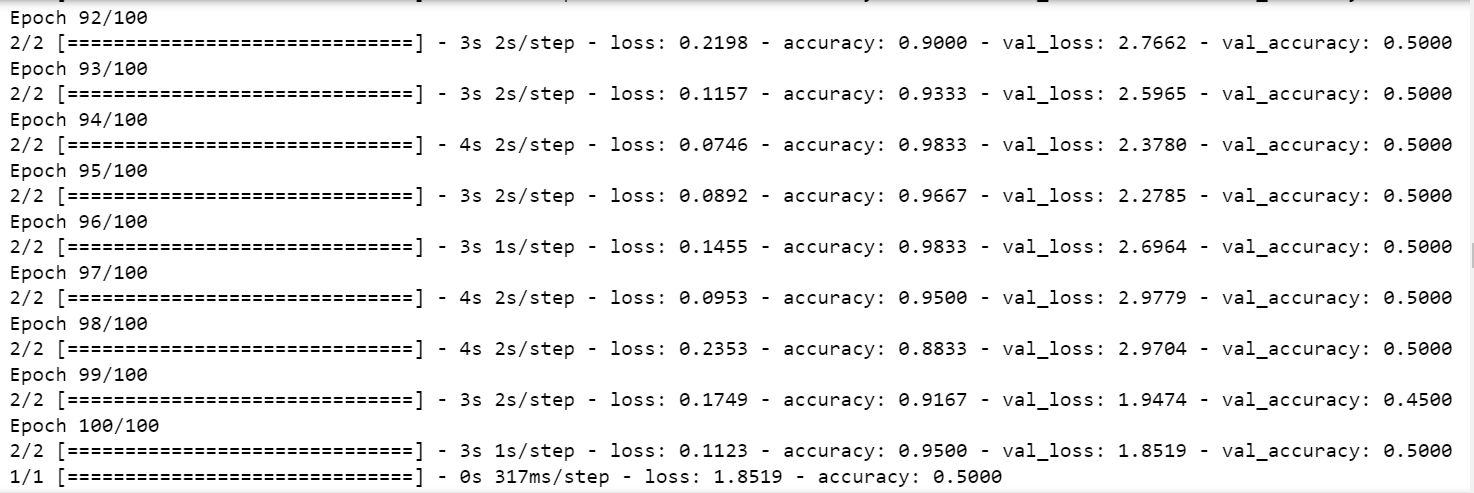
Looking at the size of dataset, the concerning part is – in general for Deep learning models we require huge dataset to train our model, otherwise there is a possibility of overfitting. Model may not be able to generalize the learning from the train data to be able to apply on validation data.

Keeping this in mind let’s create a very basic model to test this theory and plot its accuracy to be able to understand how the model behaves with such dataset and explore our next plan of actions.

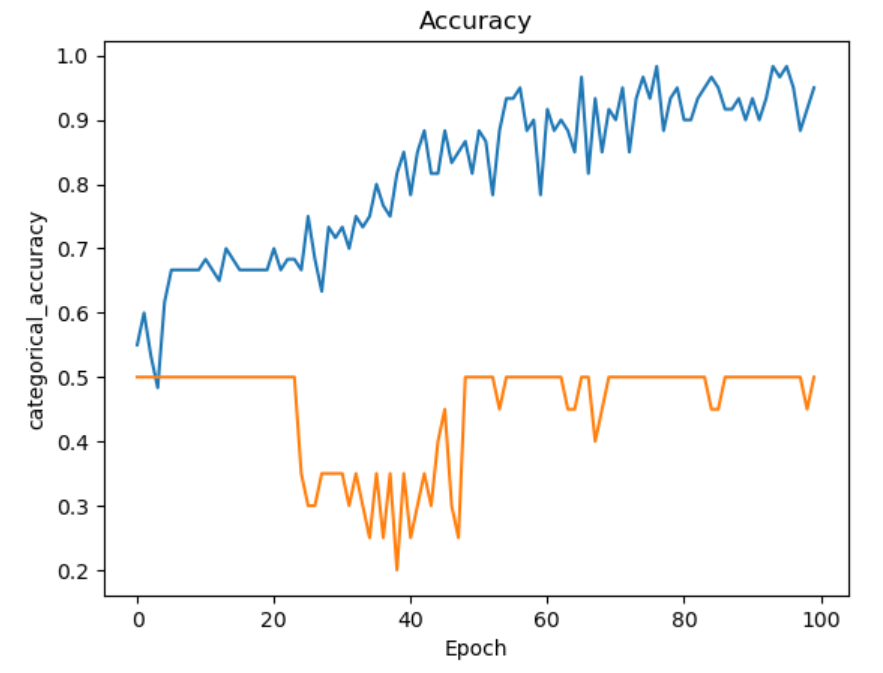
**1st Attempt for -Programming the Model architecture:**

**Model Architecture:**



**Model’s Accuracy for 100 Epochs:**

**Accuracy Plot:**



Looking at the Accuracy plot it is evident that Our model is not performing well.

From this model evaluation graph, we can see that out train data accuracy fluctuating but improving (blue lines), it is nearly 95%, however, the validation data accuracy is not improving. Instead, it keeps fluctuating. That is an indication that model might be overfitting. Model is not able to learn from the

#### Loss

Training loss is decreasing over the 100 epochs, This tells us that model is learning from the training data effectively

#### Accuracy

Training accuracy is increasing throughout the 100 epochs, and also reached 1 and then fell down, So, model is classifying training data correctly

#### Validation loss

Validation loss is also decreasing but there are fluctuations, this may be an indication that Model is not able to generalize the information learnt from training dataset

#### Validation Accuracy

Validation accuracy is improving and best accuracy is upto 0.5 but it decreases and it has a lot of fluctuation - This is clearly telling us that model is struggling to perform on validation data, which is basically overfitting scenario

Observation: The fact that accuracy for training data is improving – suggests that thr model is capable of learning from the input.

Another important thing to note is – even though the model is learning from input/train dataset, it is not able to generalize the learning and apply on unseen data, this observation is supported by the behavior seen for Validation accuracy.

Now we need to figure out how to solve the overfitting scenario we observed

Next Plan of action:

Based on the performance of the model we can see that there might be overfitting happening, and also the model is not learning and generalizing the information from training set. To improve this, we can take following action

For overfitting we have below options

* adjust the learning rate,
* Batch normalization
* increase the dropout layer to 50%,
* kernel initializers (RandomNormal/Uniform, glorot\_normal/Uniform, he\_normal/Uniform,zeros,ones),
* kernel regularization (L1-Lasso, L2-ridge, L1\_l2 -elasticnet)
* Data augmentation (feed the same training data multiple times, with rotation/scaling/flipping)
* Increase the dataset – after investigation I found that tfds has a huge data of 26000 images of Dogs and Cats. If other approach does not work, we can try using this dataset to make sure our model performs better if it is trained on sufficient pool of data
* Utilize transfer learning concept, i.e. use a model which is already trained with larger dataset, and train it with our dataset for prediction. This should be our last resort

Action1 – Adjust learning rate – Learning rate changed to 0.0001. We are already Using Adam optimizer which has the momentum as well as adaptive learning

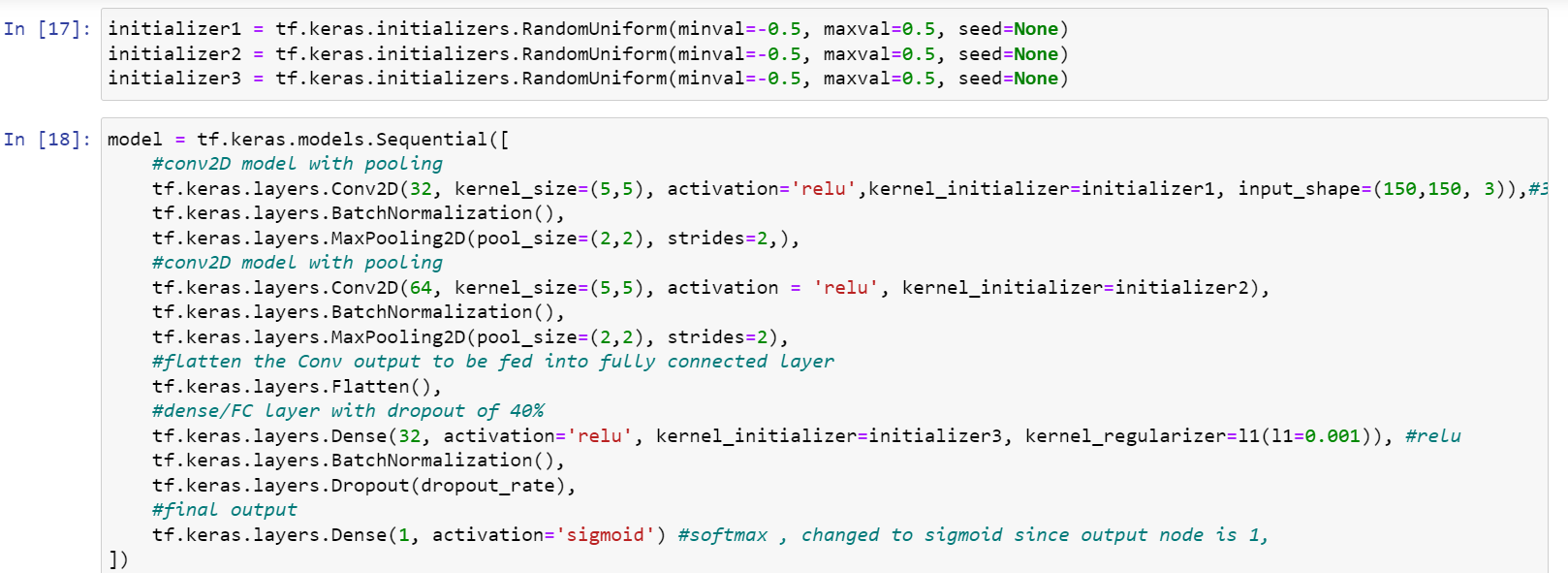
Action2 – Add BatchNormalization() from tensor flow – added batch normalization for each layer

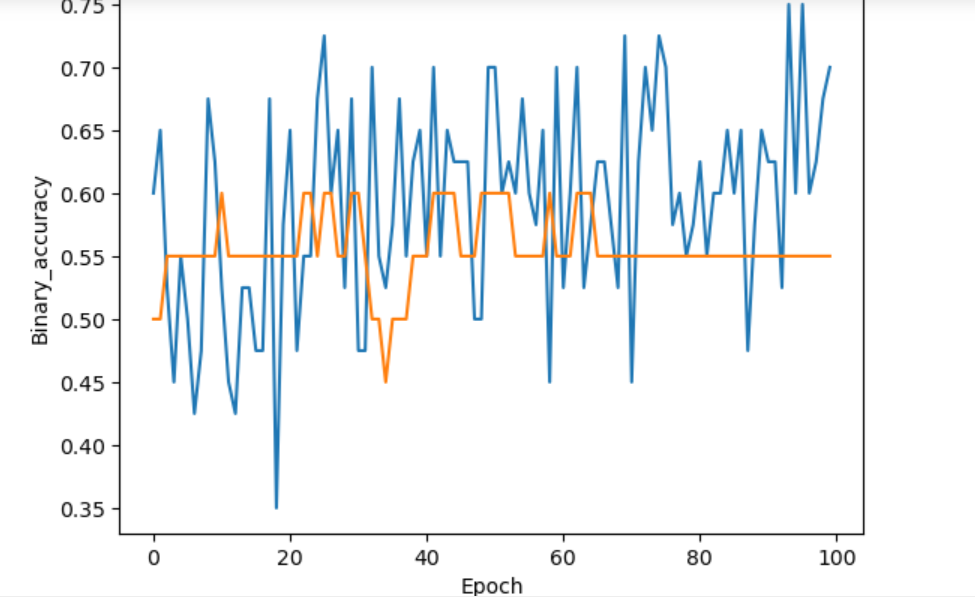
Action3 – Kernel initialization – I am using “RandomUniform” from keras with min -0.5 and max 0.5,

Action4 – Kernel regularization – I am using L1(Lasso) regularization with penalty 0.001, we will monitor the impact of these actions and then decide whether to keep it or remove it

Action5 – Data Augmentation – I believe this will help us improve our model’s performance since we have a small dataset for training, it is good to try to increase it by imageGenerator by adjusting its shear,zoom range, slit, flip or rotation

**2nd attempt of Programming Model architecture:**



**Model Evaluation:**

Analysis:

The changes we did to our model did not help get desired Accuracy, in-fact now we see that train accuracy is also not going beyond 0.75, which tells us that model is not overfitting anymore but it is underfitting.

The fluctuation in the accuracy is not a good indication for the model.

We may need to simplify the model. We have added a regularization for each of the layers, although it is a good practice but excess of regularization can also prevent the model from learning and drawing valuable insights from the data.

So’ let’s try experimenting with various values for regularization and initialization.

**Outcome: -** After experimenting with Lasso/Ridge and Elasticnet regularization applied on layers for nearly a 100 times , my observation is – the regularization is preventing the model to learn proper, so taking a judgement call here, I have decided to remove the regularization.

**Augmentation observation: -** I observed that augmentation is increasing the dataset size and helping the model learn better, so I have decided to keep it in the final model

**Activation function observation:** - when I used activation function as ‘Tanh’ in the first layer and ‘sigmoid’ in the output layer, my accuracy was best of all, so, I am modifying the architecture with these activation function for final model

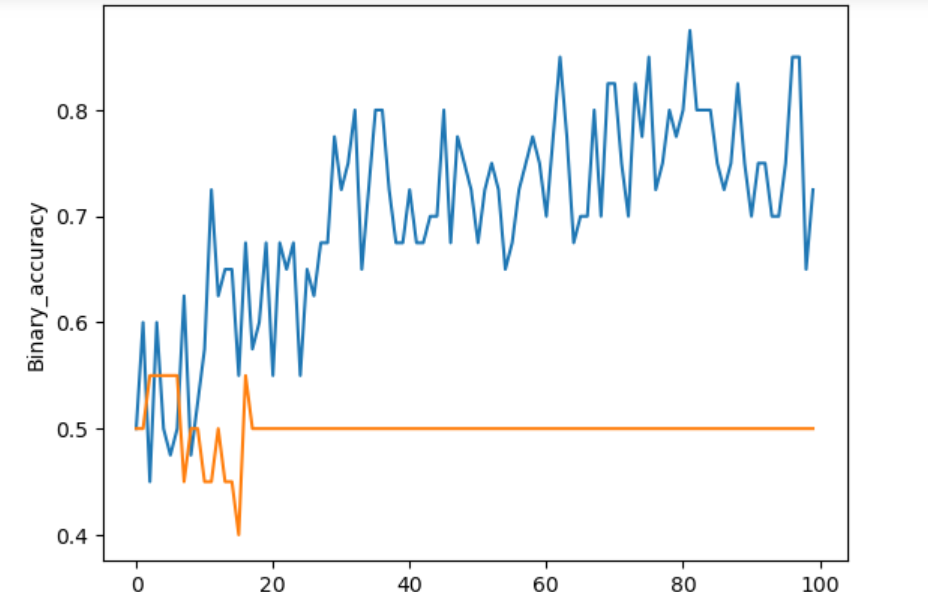
**BatchNormalization Observation:-** I did not observe any good or bad impact on the model with batch normalization, but I noticed that model does not have very drastic fluctuation in the loss and accuracy after including it , so I have decided to keep it for each layer

**Dropout Observation:-** There was a very slight improvement in the loss function for train and test with dropout layer included for 2nd convolution layer with 50% and for final outcome fully connected layer with 50%, so I have decided to keep this

**Final attempt of Programming Model architecture:**



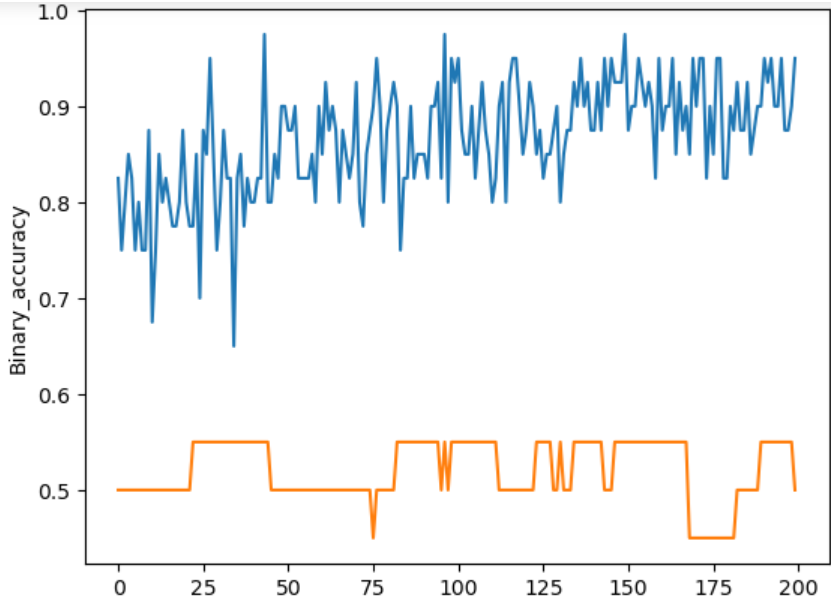
**Model Evaluation first 100 Epoch:**



Model converged at 0.5000 accuracy, all the cat’s images are classified correct but even the dog images are classified as cat. 10 out of 20 prediction is correct



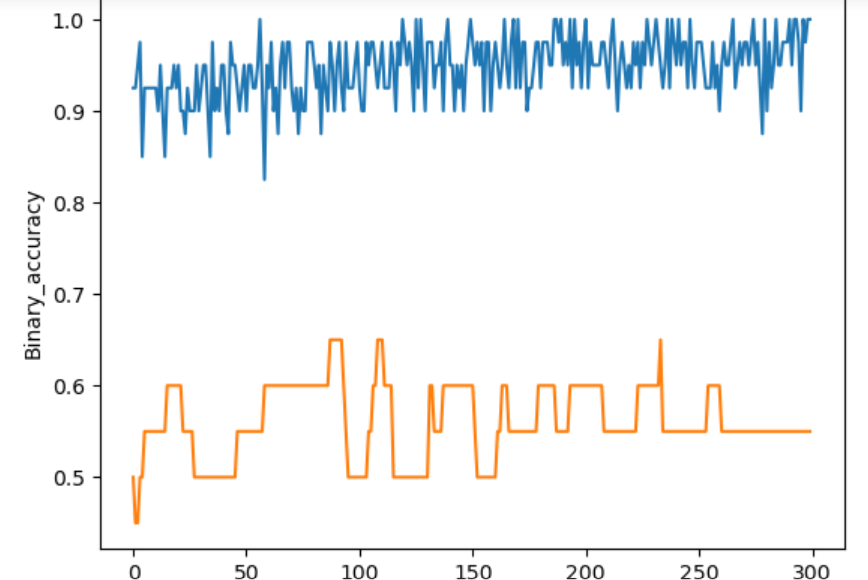
**Model Evaluation next 200 Epoch:**

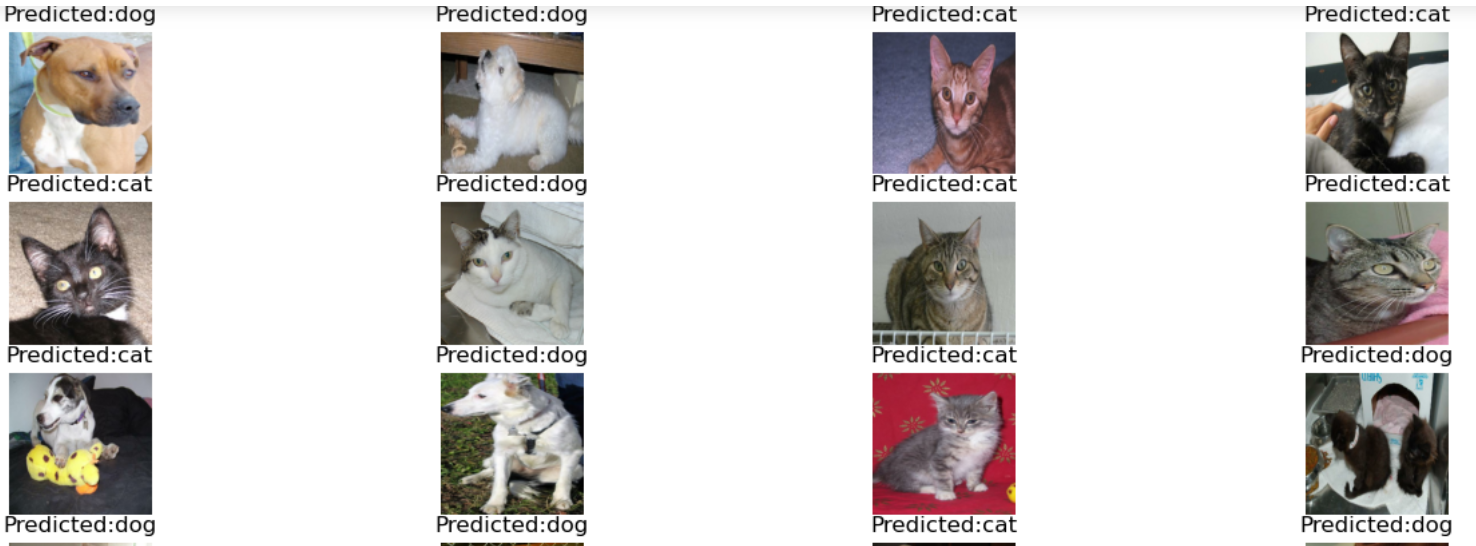


Model converged at 0.5000 accuracy for validation data. Still so much of fluctuation. Actual prediction has improved, but accuracy score is still 0.5000. Now model has identified some images of Dogs correctly and some of cat images correctly, so generalization has improved. 11 out of 20 prediction is correct



**Model Evaluation next 300 Epoch:**





Model’s maximum accuracy achieved was 0.6500, but it did not stay at those level, instead it converged at 0.5500 accuracy level. But for image prediction it has actually improved. 13 out of 20 images are classified correctly

Now this raises the question – what is going wrong in the model? Why did it not reach expected levels ? Is it only behaving irregular due to small dataset and refusing to generalize the learning or this is in-fact an inefficient model.

To find the answer of this question I did some research and found out that “TFDS” has a dataset named “cats\_vs\_dogs “which contains 26000 images for cats and dogs, So I have decided to run the same model with this dataset from TFDS

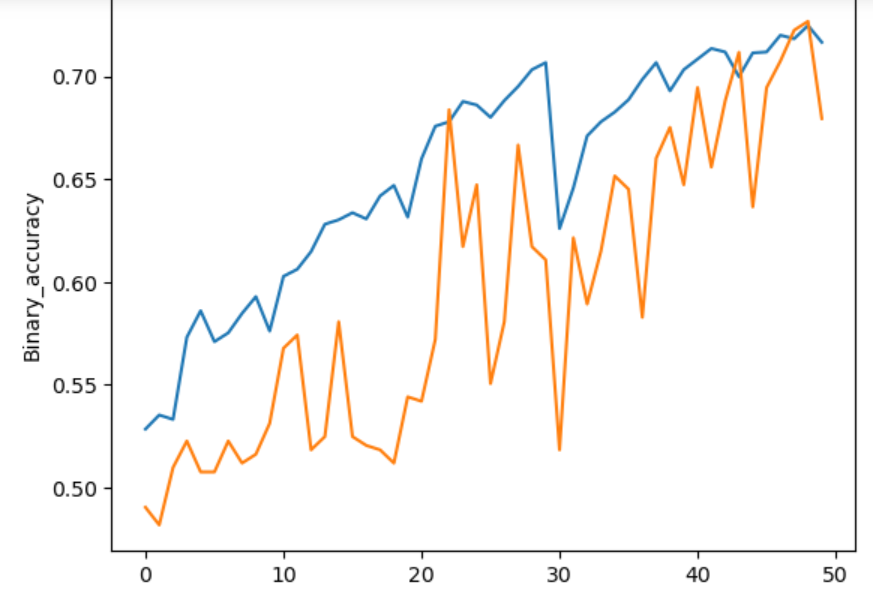
Observation1: since the dataset is so huge, my CPU started crashing as soon as I run my model. So I decided to use a small set of actual data for training and testing.

For Training I used 10% of actual data, which will be 2600 images, and for test I used 2% of actual data – which will be 520 images. As expected, – my CPU is not crashing anymore with smaller subset of actual data.

Observation2: Since the dataset is huge, it is taking a lot of time for training, so instead of running the model for 100, 200, 300 epochs, I decided to run it for 50, 20 and 30 epochs

Observation3:- Model’s accuracy has improved with large dataset. Model is able to learn from dataset and also generalize it better. That explains the scenario above, model is learning efficiently but due to smaller dataset, its generalization capabilities were limited

**Model Evaluation first 50 Epoch:**

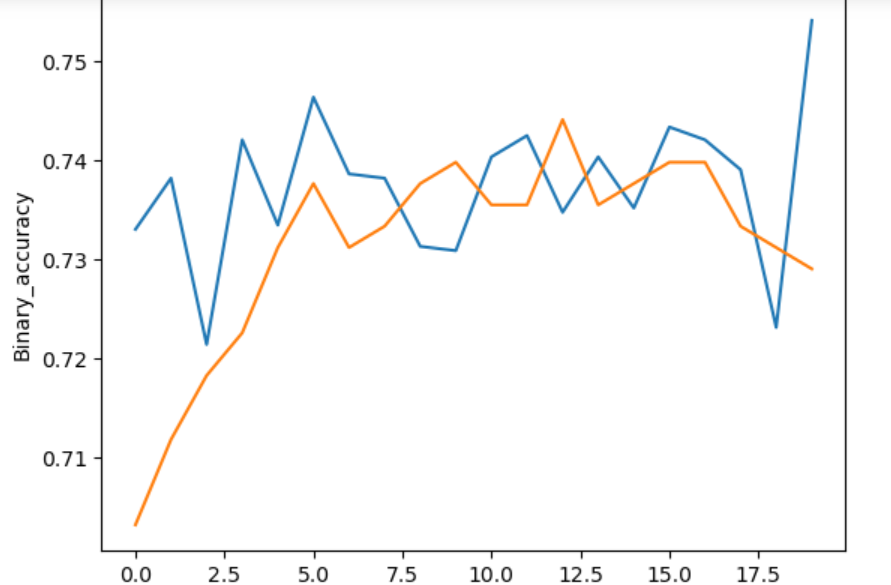


Model’s accuracy for 50 epoch is 0.6750, it went upto 0.7250 and then converged at 0.6250.

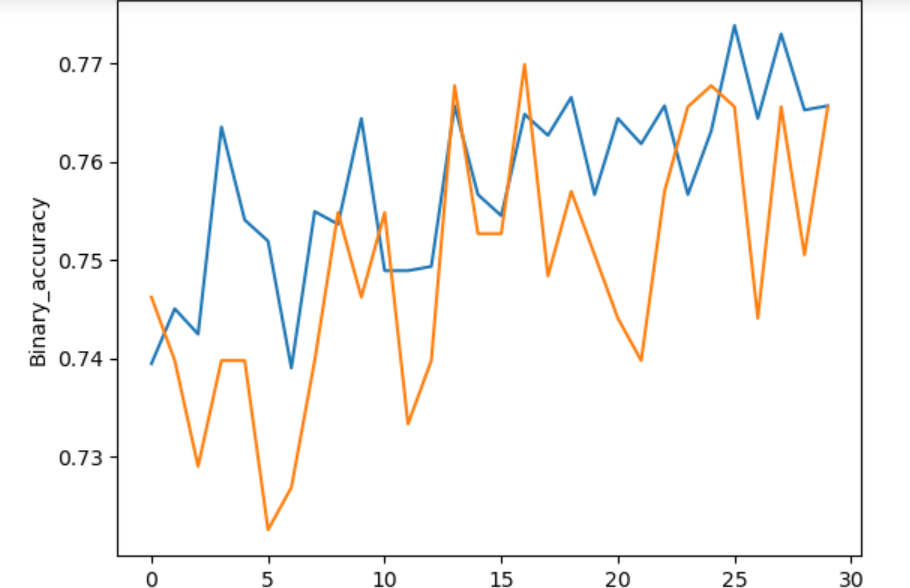
14 out of 20 images were classified correctly.



**Model Evaluation next 20 Epoch:**



**Model Evaluation last 30 Epoch:**



Learnings from project –

1. Model is capable of classifying the images.
2. Due Small dataset model’s ability to generalize the learning is affected badly
3. When dataset is small, we should use data augmentation technique, which can produce same images with tilt, rotation, blur, flip. Which increases the overall data from which model can learn
4. First target should be to make the model learn from dataset without overfitting.
5. Validation loss and accuracy represents model’s ability to generalize the learning, and apply on new unseen data
6. Too much of regularization and normalization can limit the learning ability of model
7. Reducing the learning rate, improves the model’s ability to learn over several epochs but it also reduces the speed of learning
8. It is always a good idea to plot the accuracy/loss for the model to be able to understand the progress quickly
9. It is a good practice to plot images for prediction and analyze it manually – how good the model is