Deep Model for Improving Image Quality & Image classification model.

[Author: Anustup Das]

DataSet:

The provided data set is images of Cats and Dogs.
Training Images: 300 [150 cats, 150 Dogs]
Testing Images: 150 [75 cats, 75 Dogs]

The testing images provided are noisy.
For explanation we will call **Training Images** as **Train_clean** and **Testing Images** as **Test_Noisy.**

Approach:

The task is to design a deep model that will improve the quality of the noisy image and then evaluate the model by comparing the performance of a classifier that uses **Test_Noisy as the validation data** with the classifier that uses **Test_Noisy_Cleaned** as the validation data while training.
This problem is divided into 3 models.

Module 1:
Add noise to **Train_clean** and make a separate training set with the added noisy images of **Train_clean** namely **Train_noise** using the provided script that added **salt and pepper noise.**

Module 2:
Design a **Denoising Autoencoder** model that will learn to improve image quality.
Train_noise (X_train) will be the training data which will have **Train_clean(Y_train)** as the respective labels.
After training the model we will pass **Test_Noisy** to improve the quality of the images**.** The cleaned output of **Test_Noisy** will be stored as **Test_Noisy_Cleaned.**

Module 3:
Design a **2D CNN** model that will be able to classify the images into Cats and Dogs. Train the model once with **Test_Noisy** and once with **Test_Noisy_Cleaned** as the validation set and **Train_clean** as the training set.Then compare the result from both.

Data - Preparation:

The image data are loaded from the respective paths and converted into Grayscale Images. Then the data is resized into an array of [128 X 128]. Then the array values are divided by 255 in order to make the data easily interpretable by the Deep Learning Model.

Initially the shape of the X was *['Sample_number','Img_Size',''Img_Size'']* and Y was *['Sample_number',1]* as the label comprised of only one column.
The labels were transformed into "**onehot encoding**" form for easier interpretation for the CNN models. We only deal with onehot encoding form of labels in **Module 3.** In **Module 2** we just stored the titles of the images as labels for maintaining the integrity of the images.

X was reshaped into a 3D tensor in order to fit in to the CNN model. So the reshaped data has the from of: *['Sample_number','Channel_number','Img_Size',''Img_Size'']* e.g [300,1,128,128]. And the shape of the label data is now *[300,2].*

```
Shape of Train_Data before reshaping:  (300, 128, 128)
Shape of X_train after reshaping:   (300, 1, 128, 128)
Shape of X_test after reshaping:  (150, 1, 128, 128)
```

Note: *Channel_number* is 1 for grayscale images and 3 [R,G,B] for colored images.

Module 2 [Denoising Autoencoder]:

Denoising Autoencoder works great for improving image quality. The idea is to encode all the important information from the picture into much smaller dimension and then decode it. We define the Denoising Autoencoder model with the following specifications along with different values tried for tuning the hyperparameters:

Input Shape : 1 Channel, Image_Size. [1, 128, 128]

1st Hidden Layer: Conv2D [10/32 neurons, kernel_size:5, Activation funct: relu/tanh/sigmoid]

2nd Hidden Layer: MaxPooling1D [pool_size : 2/3]

3rd Hidden Layer:Conv2D [20/40 neurons, kernel_size:5, Activation funct:relu/tanh/sigmoid]

4th Hidden Layer: MaxPooling1D [pool_size : 2/3]

5th Hidden Layer: conv2d_transpose[20/32 neurons, kernel_size:2/5, Activation funct: relu/tanh/sigmoid]

6th Hidden Layer: UpSampling2D[pool_size : 2]

7th Hidden Layer: conv2d_transpose[20/10 neurons, kernel_size:2/5, Activation funct:

relu/tanh/sigmoid]

8th Hidden Layer: Conv2D [10/32 neurons, kernel_size:5, Activation funct: relu/tanh/sigmoid]

9th Hidden Layer: UpSampling2D[pool_size : 2]

10th Hidden Layer: conv2d_transpose[20/32 neurons, kernel_size:2/5, Activation funct: relu/tanh/sigmoid]

Output Layer: [Output Shape = 1,128,128] *//As the model will output an image of the same dimension as the input.*

Loss Function: "binary_crossentropy"

Optimizer: SDG/adadelta

Number of Epochs: 100

Batch Size: 15/25/40

The model trains for 100 epoch with a callback function which keeps track of both accuracy and loss and when we get a global minimum loss from the model the weights of that particular epoch is stored in a file named ***""DenoiseImg-Sigmoid-Best-weights-my_model-{epoch:03d}-{loss:.4f}-{acc:.4f}.hdf5"*** along with a stopping criteria which will monitor the training for n number of iterations and if the global minimum loss is not improved within that n iteration the training will be stopped.

Note: The final parameters which yielded best results the present in the submitted code.

Evaluating Model:

In this experiment we allowed the model to train for the total number of epochs without any stopping criteria. After training we plot the training and validation accuracy along with the loss over the total number of epochs and analysis where the model performed best. We then model the model with the saved weight of the best epochs.

The log for every epoch is stored in the file :
"*DenoiseImg_'ActivationFunctionName'_model_train.csv*".

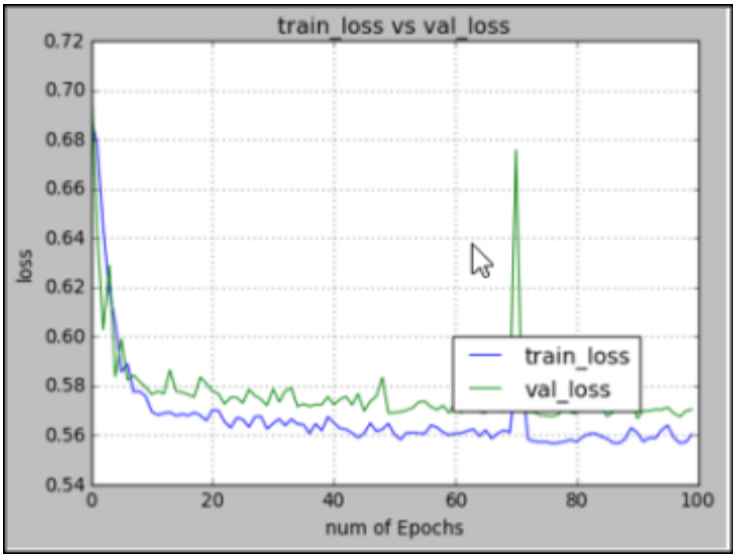


Fig: Graph showing the loss over train and validation data for 100 epochs.

The model in this experiment achieved its lowest loss value as **0.56684**.
Using different Activation function produced different results. In the below figure one can observe that "**tanh**" activation function did a better job in this model setting that other activation functions.

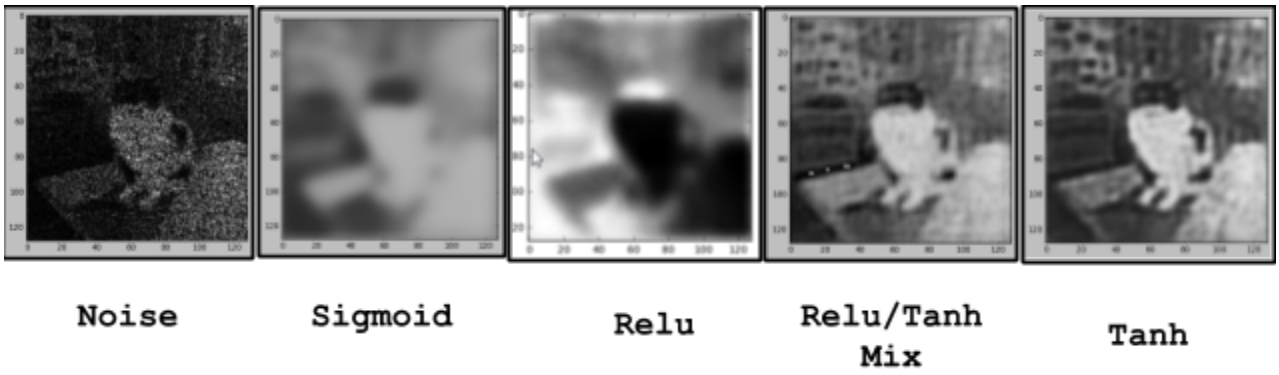


Fig: Image quality with different Activation functions.

Module 3 [CNN Classification Model]:

We define the CNN model for image classification with the following specifications along with different values tried for tuning the hyperparameters:

Input Shape : 1 Channel, Image_Size. [`1, 128, 128`]

1st Hidden Layer: Conv2D [64/32 neurons, kernel_size:5/3, Activation funct: relu/tanh]

2nd Hidden Layer: Conv2D [64/32 neurons, kernel_size:5/3, Activation funct: relu/tanh]

3rd Hidden Layer: MaxPooling1D [pool_size : 2/3]

4th Hidden Layer: Dropout [0.5/0.2 fraction of the input units to drop] *//To reduce overfitting.*

5th Hidden Layer: Conv2D [64/32 neurons, kernel_size:5/3, Activation funct: relu/tanh]

6th Hidden Layer: MaxPooling1D [pool_size : 2/3]

7th Hidden Layer: Dropout [0.5/0.2 fraction of the input units to drop]

8th Hidden Layer: Flatten

9th Hidden Layer: Dense [64/100 neurons, Activation function : relu/tanh]

10th Hidden Layer: Dropout [0.5/0.2 fraction of the input units to drop]

Output Layer: [2 neurons, Activation function : softmax] *//two values for target label*

Loss Function: 'categorical_crossentropy'

Optimizer: Adam/SDG

Number of Epochs: 10**,** 15, 20 [limited to small number of epochs for hardware limitations]

Batch Size: 16/25/50

The model trains for 15 epoch with a callback function which keeps track of both accuracy and loss and when we get a global minimum loss from the model the weights of that particular epoch is stored in a file named *"Noisyval_Best-weights-my_model-{epoch:03d}-{loss:.4f}-{acc:.4f}.hdf5"* for training with **Test_Noisy** and *"Cleanval_Best-weights-my_model-{epoch:03d}-{loss:.4f}-{acc:.4f}.hdf5"* for training with **Test_Noisy_Cleaned** as validation data along with a stopping criteria which will monitor the training for n number of iterations and if the global minimum loss is not improved within that n iteration the training will be stopped.

Note: The final parameters which yielded best results the present in the submitted code.

Evaluating Model:

In this experiment we allowed the model to train for the total number of epochs without any stopping criteria. After training we plot the training and validation accuracy along with the loss over the total number of epochs and analysis where the model performed best. We then model the model with the saved weight of the best epochs.

The log for every epoch is stored in the file : "*PetClassification_NoisyvalImg_model_train.csv*"
The log for every epoch is stored in the file : "*PetClassification_Cleanval_model_clean_train.csv*" for training with **Test_Noisy** and **Test_Noisy_Cleaned** respectively**.**

The model in this experiment achieved its lowest loss value as **0.67858** and highest accuracy value as **0.5733** for **Test_Noisy .**

The model in this experiment achieved its lowest loss value as **0.67902** and accuracy value as **0.6000** for **Test_Noisy_Cleaned .**

Result Comparison:

The Validation loss and Validation accuracy of both the models trained with **Test_Noisy** and **Test_Noisy_Cleaned** were compared and the below figure displays the results of the experiment.

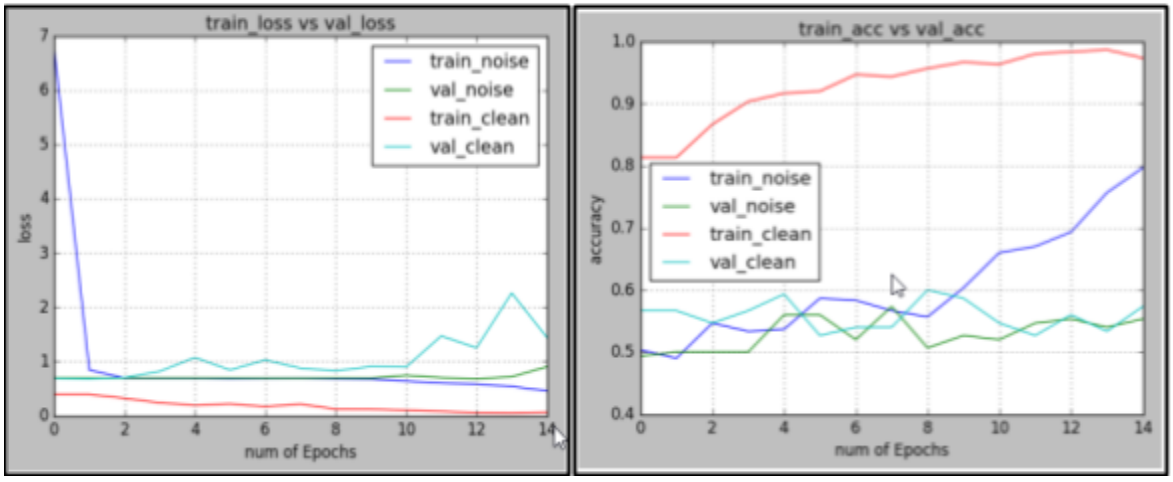
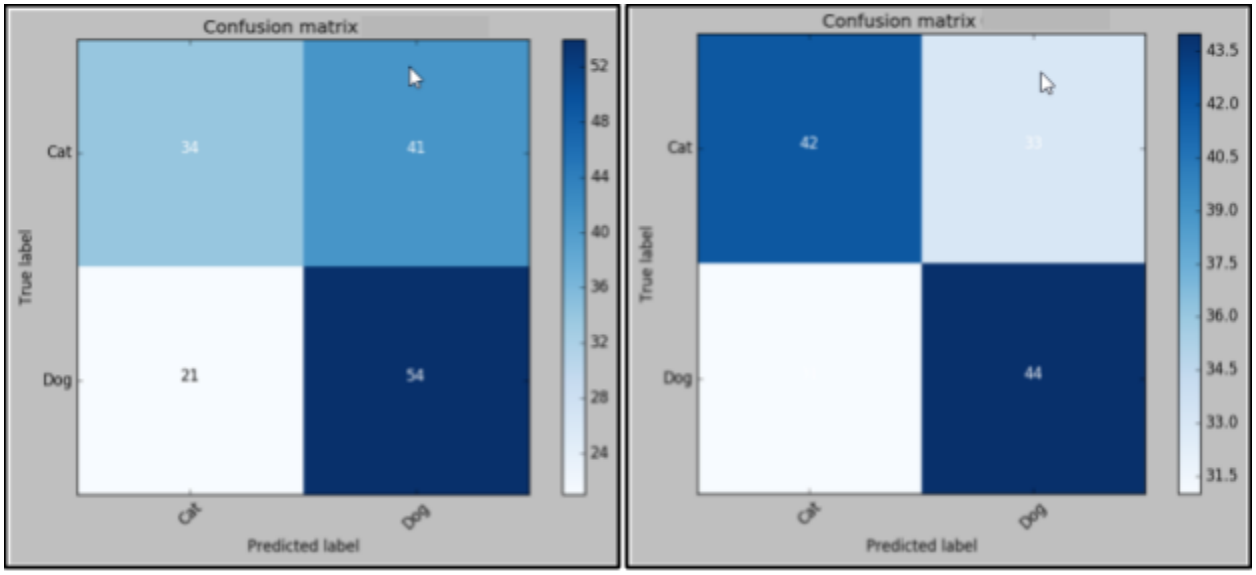


Fig: Graph showing comparison between the two training over loss and accuracy on train and validation data for 15 epochs.

As seeing from the above graph the model appears to perform slightly better with cleaned test data than noisy test data in terms of validation accuracy. But the validation loss is higher than the validation loss of noisy test data. The model also seems to overfits the training data in both

the cases. One of the reason maybe because of low epochs and also less amount of available training data. While comparing the confusion matrix of the model for both the validation data it seems to perform poorly. Thus it can be concluded that the image quality of the cleaned data is not very good and further modification is needed to the **Denoising Autoencoder**.



Noisy Test Data **Cleaned Test Data**

Fig: Confusion matrix showing comparison between CNN and LSTM.

Future Scope:

There is a lot of scope of improvement of the models with furthermore tuning of the hyper parameters and observing the effects on the model until we can achieve lowest possible loss for the **Denoising Autoencoder**.

Different data augmentation techniques can be used to deal with the small number of training data problem for the **CNN classification model also**.