Project Title: Pet Classification Model Using CNN

Problem Statement:

The objective of this project is to build a CNN model that accurately classifies pet images into dogs and cats. The dataset consists of images with varying sizes and lighting conditions. The challenge is to develop a model that can effectively extract features from these images and correctly classify them.

Approach:

➔ Dataset Preparation:

1. Gather the dataset containing labeled images of cats and dogs.
2. Perform data preprocessing, including resizing the images to a consistent size and normalizing the pixel values.

➔ Model Architecture:

Design the CNN model with 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 with a fixed output size defined by the hyperparameter: fc_size = 32
Dropout layer with a dropout probability of 0.4
SoftMax layer for predicting the class probabilities.

➔ Model Training:

1. Split the dataset into training and validation sets.
2. Define the loss function, such as binary cross-entropy, and the optimization algorithm, such as Adam optimizer.
3. Train the model on the training set and validate it on the validation set.
4. Experiment with different hyperparameter settings such as image resizing, data augmentation, batchnormalization, regularization, and optimizers to improve the model's performance.

➔ Evaluation:

- Evaluate the trained model on a separate test set to measure its accuracy and loss.
- Monitor the model's performance metrics and analyze the results.

➔ Model Improvement:

If the initial accuracy is not satisfactory, these techniques are tried for better performance:

1. Adjusting hyperparameters such as learning rate, batch size, regularization, and optimizers.

2. Apply data augmentation techniques to increase the diversity of the training data.
3. Acquire more labeled data or improve the quality of the existing dataset.
4. Apply image resizing, rotation, horizontal/vertical flips for improving model's veracity.

Findings:

The first model is trained with the specified architecture for 100 epochs, but the model's performance wasn't that good as there was so much overfitting in the model. So, tried adding l2 regularization to the model which improved the validation accuracy and reduced overfitting. But adding batchnormalization worsened the model's performance.

```
model=Sequential()
model.add(Conv2D( 32, (5, 5 ), activation = 'relu', padding = 'same', input_shape = (128, 128 , 3 ),kernel_regularizer =tf.keras.regularizers.l2( l=0.03)))
# model.add(BatchNormalization())
model.add(MaxPooling2D(2,2))
model.add(Conv2D(64, (5, 5 ), activation = 'relu',kernel_regularizer =tf.keras.regularizers.l2( l=0.03)))
# model.add(BatchNormalization())
model.add(MaxPooling2D((2,2)))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(32, activation = 'relu',kernel_regularizer =tf.keras.regularizers.l2( l=0.001)))
model.add(Dense(2, activation = 'softmax'))

```

Figure 1. Model1 model architecture

The second model is trained for 200 epochs with reduced regularization constant as running it with the previous model's parameters has reduced accuracy and performance.

```
model2=Sequential()
model2.add(Conv2D( 32, (5, 5 ), activation = 'relu', padding = 'same', input_shape = (256,  256 , 3 ),kernel_regularizer =tf.keras.regularizers.l2( l=0.02)))
model2.add(MaxPooling2D(2,2))
model2.add(Conv2D(64, (5, 5 ), activation = 'relu',kernel_regularizer =tf.keras.regularizers.l2( l=0.02)))
# model2.add(BatchNormalization())
model2.add(MaxPooling2D((2,2)))
model2.add(Dropout(0.4))
model2.add(Flatten())
model2.add(Dense(32, activation = 'relu',kernel_regularizer =tf.keras.regularizers.l2( l=0.001)))
# model2.add(BatchNormalization())
model2.add(Dense(2, activation = 'softmax'))

```

```
model2.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
pets=model2.fit(training_set,epochs=200,validation_data = test_set,callbacks=[PlotLossesKerasTF()])
```

Figure 2. Model 2 model architecture

And the third model is run for 300 epochs with the same hyperparameters resulted in the same performance as the previous model.

```
model3=Sequential()
model3.add(Conv2D( 32, (5, 5 ), activation = 'relu', padding = 'same', input_shape = (256,  256 , 3 ),kernel_regularizer =tf.keras.regularizers.l2( l=0.02)))
model3.add(MaxPooling2D(2,2))
model3.add(Conv2D(64, (5, 5 ), activation = 'relu',kernel_regularizer =tf.keras.regularizers.l2( l=0.02)))
# model2.add(BatchNormalization())
model3.add(MaxPooling2D((2,2)))
model3.add(Dropout(0.4))
model3.add(Flatten())
model3.add(Dense(32, activation = 'relu',kernel_regularizer =tf.keras.regularizers.l2( l=0.0007)))
# model2.add(BatchNormalization())
model3.add(Dense(2, activation = 'softmax'))

```

```
model3.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
pets=model3.fit(training_set,epochs=300,validation_data = test_set,callbacks=[PlotLossesKerasTF()])
```

Figure 3. Model 3 model architecture

Screenshots:

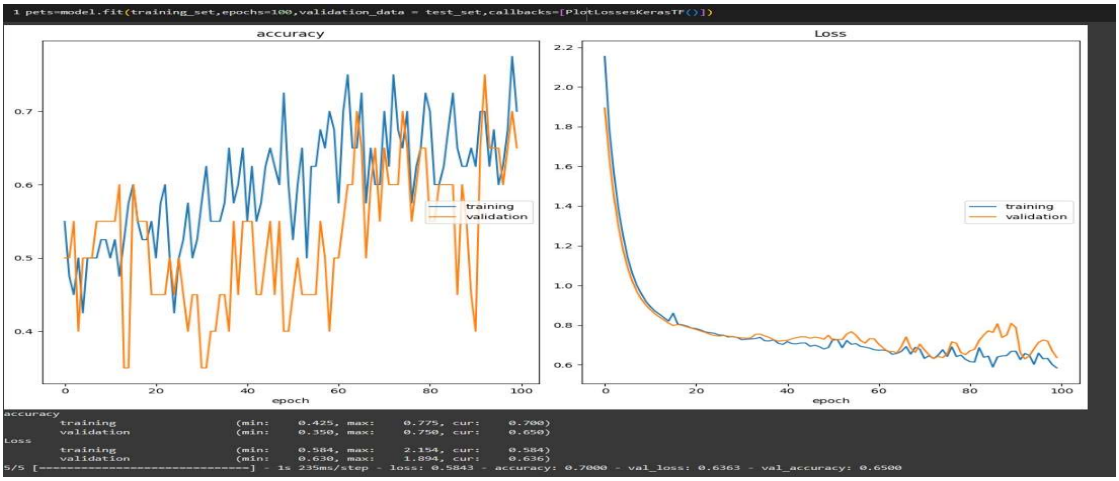

Figure4. Model1 performance

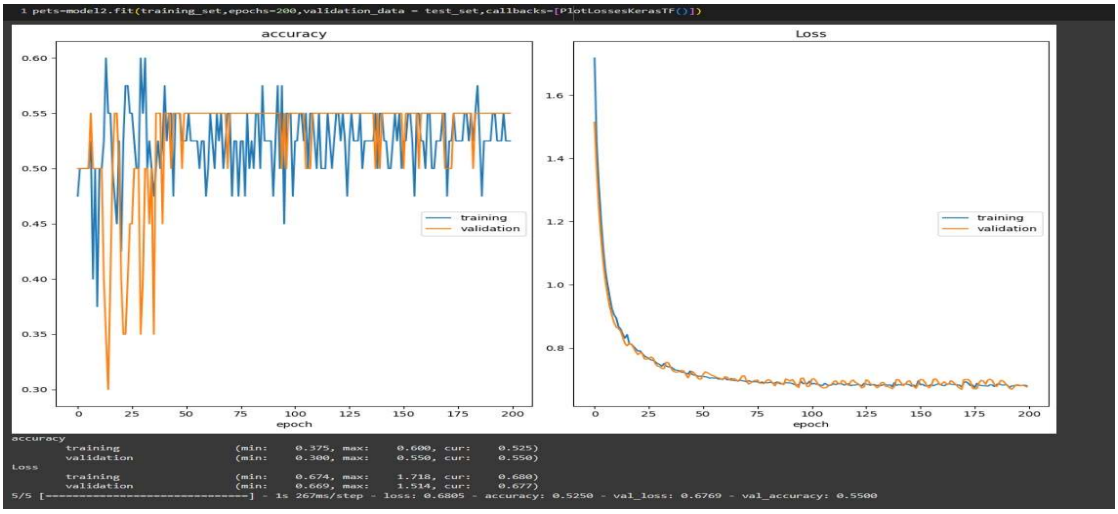

Figure 5. Model2 performance

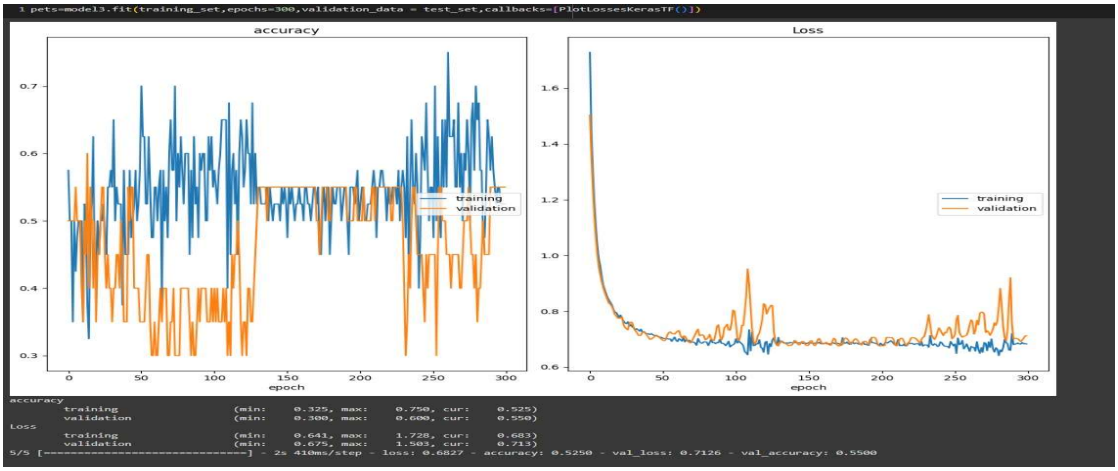

Figure 6. Model3 performance

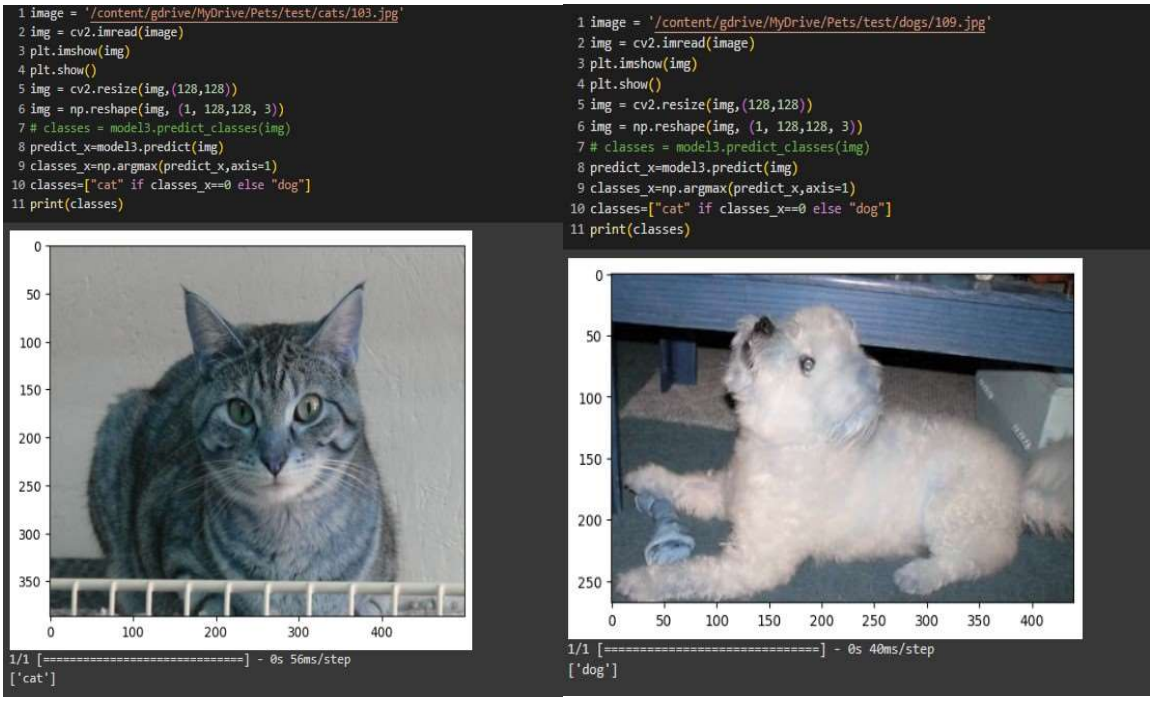

Fig 7. Testing the cat and dog images on the model 1 predictions

Conclusion:

Overall, the model1 has given good performance when compared to the other two models with high epochs (200, 300). So, using model1 as a base model for this project is a better option as it has given 65% validation accuracy with no overfitting and the model converged almost well with low loss. L2 regularization has played a significant role in model convergence.