MSDS 422 Assignment 7
Husein Adenwala

Introduction

The purpose of this assignment is to compare 3 convolutional neural network models for binary classification of dog vs cat images. The cats and dogs dataset contains 12501 images of cats and 12501 images of dogs.

Data preparation, exploration, visualization

The cats and dogs dataset are imported from the Microsoft website. The dataset contained 25,002 images that were evenly divided between two folders labeled cats and dogs, respectively.

12501 images Cat
12501 images Dog

I filtered the images to remove any corrupt or unusable files, which resulted in finding and removing two images from the dataset.

The original images are of different pixel sizes. I standardized the size by padding the images (with[0]). However, this did not improve any model performance and was excluded in the final training, validation and test data.

original images **Padded images**

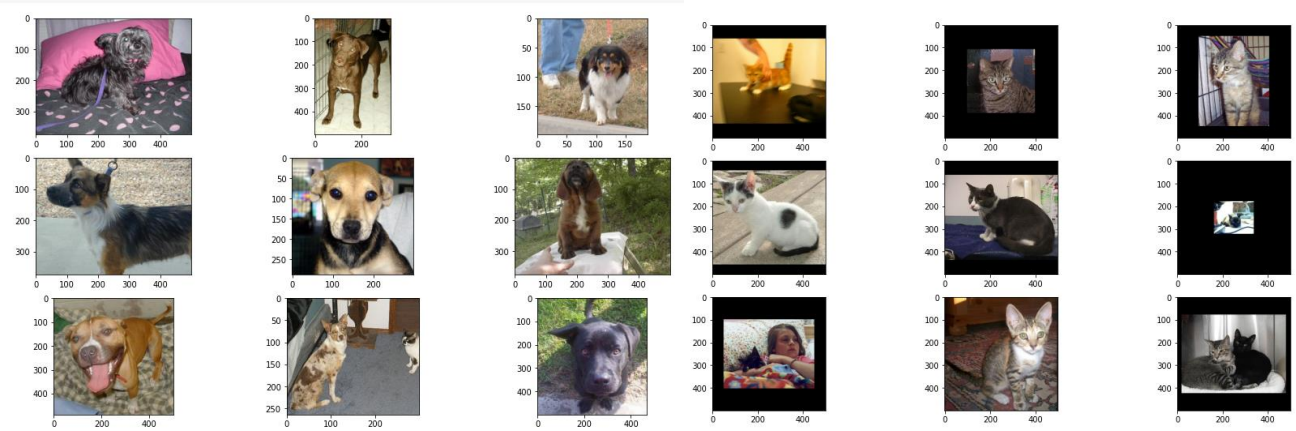


Image Transformation: Reshape, enhancement and Normalization

I used the following image augmentation process to normalize and enhance images that enables faster, better and stable training of deep neural networks. The data augmentation can also help in reducing overfitting.

- Pixel Normalization: scale pixel values to the range 0-1.
    - Rescale: The original images consist of RGB coefficients in the 0-255 range, but such values would be too high for our models to process, so I targeted values between 0 and 1 instead by scaling with a 1/255 factor.
- Pixel Centering: scale pixel values to have a zero mean.
    - This has the effect of centering the distribution of pixel values on zero: that is, the mean pixel value for centered images will be zero
- Pixel Standardization: scale pixel values to have a zero mean and unit variance.

    - Standardization is a data scaling technique that assumes that the distribution of the data is Gaussian and shifts the distribution of the data to have a mean of zero and a standard deviation of one.
- rotation_range is a value in degrees (0-180), a range within which to randomly rotate pictures. I used 20 degrees rotation for the images.
- width_shift and height_shift are ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally. I used 0.2 values as a lot of the images are not centered and this helped.
- shear_range is for randomly applying shearing transformations. I used 0.2 value to shear the image
- zoom_range is for randomly zooming inside pictures. This helps to focus on the pet image
- horizontal_flip is for randomly flipping half of the images horizontally --relevant when there are no assumptions of horizontal asymmetry; creates a diverse dataset.

After preprocessing, images were more focused on the dogs and cats; however, there is reduction in image quality, but this did not have much impact on model performance as image augmentation reduced the noise in the image. On the other hand, increasing the pixels improved image quality which increased the model complexity and took more processing time, but did not have any significant impact on model performance.

Still, there were some poor quality images in the original dataset that individually were not useful but with the majority of images being correct, the neural net would not be affected by any poor images.

Training: Found 20250 images belonging to 2 classes.
Validation: Found 2500 images belonging to 2 classes.
Testing: Found 2248 images belonging to 2 classes.

.

Review research design and modeling methods

For this analysis, I used the Convolution Neural Network (CNN) model to classify Dogs and Cats Images. CNN can solve classification problems and its strength is its ability to dynamically create complex prediction functions and emulate human thinking.

CNN architecture design has the most impact on accuracy. I compared 3 models to evaluate the impact of regularization and the architecture design of layers and number of nodes.

I started with the base model without regularization (Model 1), then a base model with regularization (Model 2) and finally, a best model which is inspired by the vgg-16 image classification model (Model 3).

I used regularization techniques of batch normalization, early stopping and the dropout method to reduce any overfitting and improve model performance. I used 30 Epoch, Relu activation function and Adam optimizer function in all the models. Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. Adam default configuration parameters do well on most problems.

I found that for this problem and dataset, the CNN model with increasing filters per additional CNN layers performed better as compared to decreasing filters per additional CNN layer.

Review results, evaluate models

All models included the following parameters:

Optimizer: Adam

Epoch: 30

Activation function: Relu (hidden layers), Sigmoid (output layer),

Regularization: Early stopping(patience =5), dropout, and Batch normalization (model 2 and model 3 only).

Model 1- No Regularization: 3 Convolution layers model and 1 Dense layer

I used this as the base model where the processed images data that had 150,150,3 pixels are fed into 3 convolution layers with 16, 32 and 64 filters and each layer was connected to max pooling and used Relu activation function respectively, before it was connected to a flatten layer, which was then connected to a dense layer with 448 nodes, which was then connected to 1 output layer that used sigmoid activation function.

Time to process: 1h 16min 8s

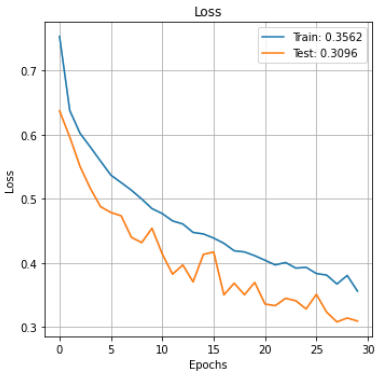
Train: `loss: 0.3434 - acc: 0.8502`

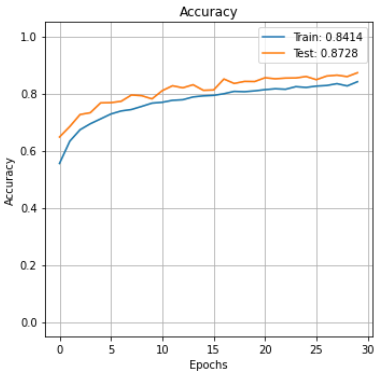
Test: `loss: 0.3650 - acc: 0.8327`

Validate: `loss: 0.3096 - acc: 0.8728`

(Test = on validation data in the graphn below)

The graph shows some underfitting on the validation data as the validation (test in the graph) data images are not augmented/preprocessed.

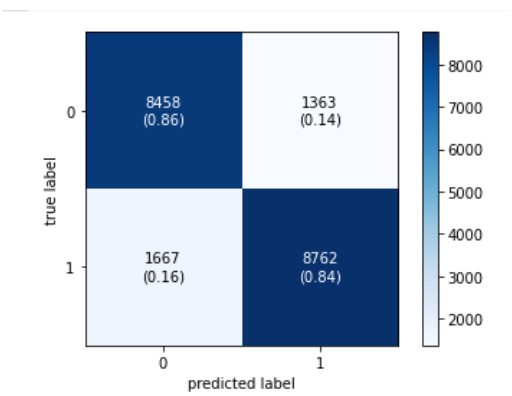





```
Model: "base"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_31 (Conv2D)           (None, 148, 148, 16)      448
_________________________________________________________________
max_pooling2d_31 (MaxPooling (None, 74, 74, 16)        0
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 72, 72, 32)        4640
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 34, 34, 64)        18496
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 17, 17, 64)        0
_________________________________________________________________
flatten_9 (Flatten)          (None, 18496)             0
_________________________________________________________________
dense_20 (Dense)             (None, 448)               8286656
_________________________________________________________________
dense_21 (Dense)             (None, 1)                 449
=================================================================
Total params: 8,310,689
Trainable params: 8,310,689
Non-trainable params: 0
```

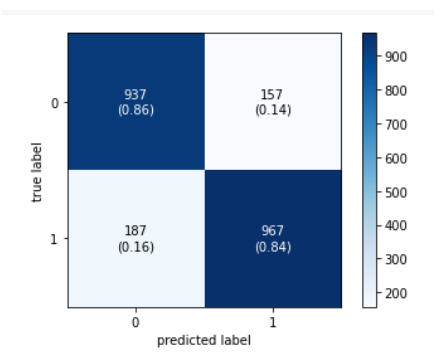
The confusion matrix shows that the model performed well on train and test data; the accuracy of true positive and true negative is balanced. The model performed similarly on test and train data and there was no overfitting. Although I did not use any regularization technique in this model, it seems that image augmentation/preprocessing had reduced any over fitting.

Train



	precision	recall	f1-score	support
0	0.84	0.86	0.85	9821
1	0.87	0.84	0.85	10429
accuracy			0.85	20250
macro avg	0.85	0.85	0.85	20250
weighted avg	0.85	0.85	0.85	20250

Test



	precision	recall	f1-score	support
0	0.83	0.86	0.84	1094
1	0.86	0.84	0.85	1154
accuracy			0.85	2248
macro avg	0.85	0.85	0.85	2248
weighted avg	0.85	0.85	0.85	2248

Model 2 Regularization: 3 Convolution layers model and 1 Dense layer

In this model, I added the regularization techniques of drop out and batch normalization. Similar to the first model the processed images data that had 150,150,3 pixels was fed into 3 convolution layers with 16, 32 and 64 filters and each layer was connected to max pooling and used Relu activation function and implemented dropout (0.2) respectively. This was connected to a flatten layer which was connected to a dense layer with 448 nodes that applied batch normalization before it was connected to 1 output layer that used sigmoid activation function. I uses early stopping based on validation accuracy (I am not sure if this affected the model accuracy and performance)

Time to process: 1h 1min 26s (Early stopping on epoch 27)

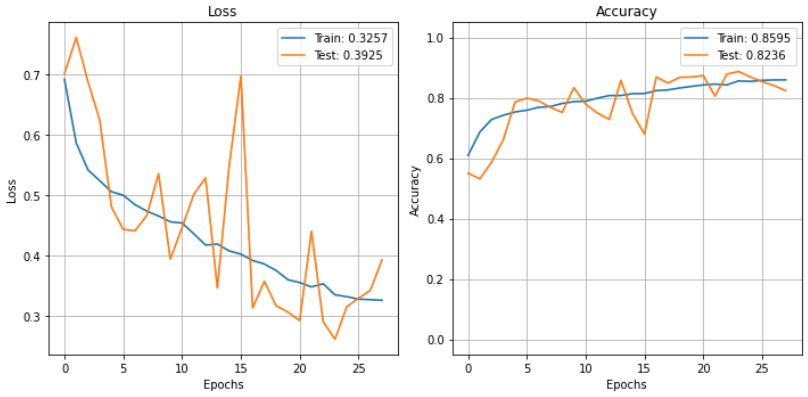
Train: `loss: 0.3498 - acc: 0.8474`

Test: `loss: 0.3094 - acc: 0.8680`

Validate: `loss: 0.3925 - acc: 0.8236`

(Test = validation data in the graph below)

The graph shows some underfitting on the validation data as the validation data images are not augmented:



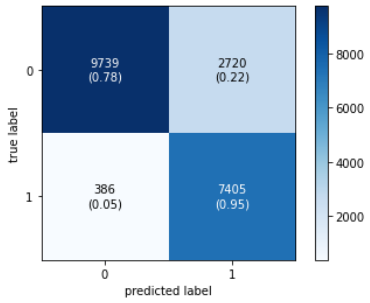
Model: "base + Regularization"

Layer (type)	Output Shape	Param #
conv2d_34 (Conv2D)	(None, 148, 148, 16)	448
max_pooling2d_34 (MaxPooling	(None, 74, 74, 16)	0
dropout_16 (Dropout)	(None, 74, 74, 16)	0
conv2d_35 (Conv2D)	(None, 72, 72, 32)	4640
max_pooling2d_35 (MaxPooling	(None, 36, 36, 32)	0
dropout_17 (Dropout)	(None, 36, 36, 32)	0
conv2d_36 (Conv2D)	(None, 34, 34, 64)	18496
max_pooling2d_36 (MaxPooling	(None, 17, 17, 64)	0
dropout_18 (Dropout)	(None, 17, 17, 64)	0
flatten_10 (Flatten)	(None, 18496)	0
dense_22 (Dense)	(None, 448)	8286656
batch_normalization_4 (Batch	(None, 448)	1792
dense_23 (Dense)	(None, 1)	449

Total params: 8,312,481
Trainable params: 8,311,585
Non-trainable params: 896

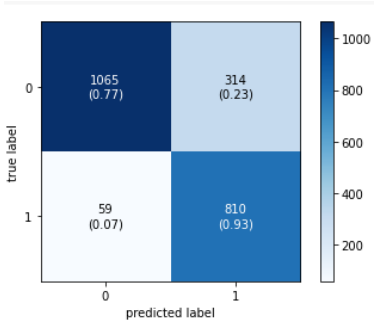
The confusion matrix shows that the model performed similarly on train and test data. The accuracy of true positive and true negative is imbalanced. The model has high false negatives.

Train



	precision	recall	f1-score	support
0	0.96	0.78	0.86	12459
1	0.73	0.95	0.83	7791
accuracy			0.85	20250
macro avg	0.85	0.87	0.84	20250
weighted avg	0.87	0.85	0.85	20250

Test



	precision	recall	f1-score	support
0	0.95	0.77	0.85	1379
1	0.72	0.93	0.81	869
accuracy			0.83	2248
macro avg	0.83	0.85	0.83	2248
weighted avg	0.86	0.83	0.84	2248

Model 3: 5 Convolution layers model and 2 Dense layers (with regularization)

This is Best model where the processed images data that had 150,150,3 pixels were fed into 5 convolution layers with 32 (included padding), 64, 128, 256 and 512 filters and each layer was connected to max pooling, used Relu activation function and implemented dropout(0.2) respectively. This was connected to a flatten layer which was connected to two dense layers with 256 and 128 nodes and batch normalization respectively and connected to 1 output layer which used sigmoid activation function. I also used Early stopping, but the model took almost 30 epochs to get the best accuracy.

Time to process: 1h 18min 7s

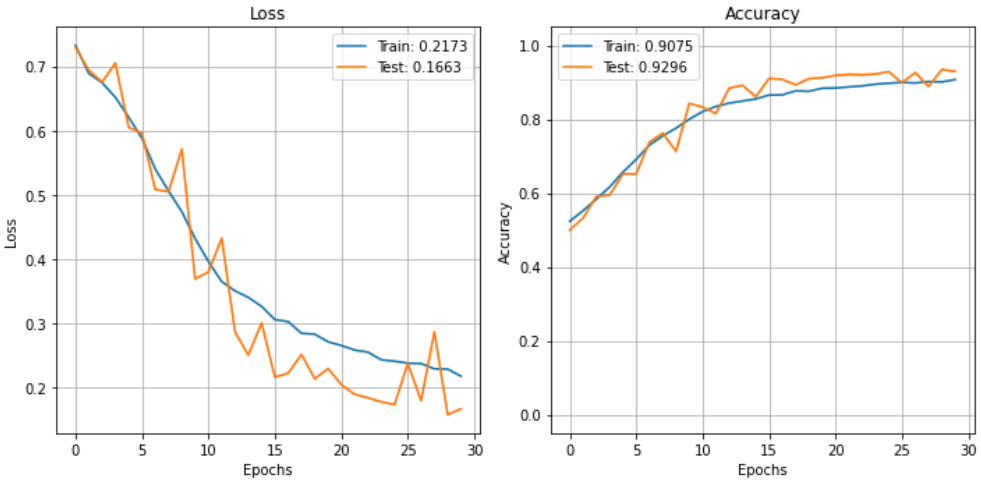
Train: `loss: 0.1727 - acc: 0.9285`

Test: `loss: 0.1999 - acc: 0.9168`

Validate: `loss: 0.1663 - acc: 0.9296`

(Test = on validation data in the graphn below)

The graph shows the model is slightly underfitting. validation data images are not augmented.



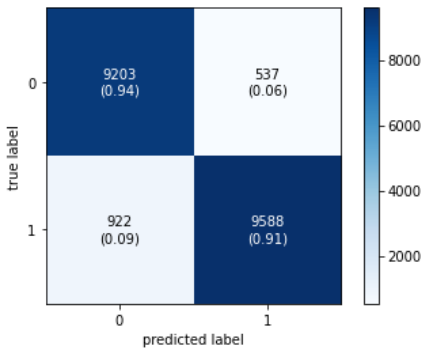
Model: "best_CNN"

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 150, 150, 32)	896
max_pooling2d_5 (MaxPooling2	(None, 75, 75, 32)	0
dropout_5 (Dropout)	(None, 75, 75, 32)	0
conv2d_6 (Conv2D)	(None, 73, 73, 64)	18496
max_pooling2d_6 (MaxPooling2	(None, 36, 36, 64)	0
dropout_6 (Dropout)	(None, 36, 36, 64)	0
conv2d_7 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_7 (MaxPooling2	(None, 17, 17, 128)	0
dropout_7 (Dropout)	(None, 17, 17, 128)	0
conv2d_8 (Conv2D)	(None, 15, 15, 256)	295168
max_pooling2d_8 (MaxPooling2	(None, 7, 7, 256)	0
dropout_8 (Dropout)	(None, 7, 7, 256)	0
conv2d_9 (Conv2D)	(None, 5, 5, 512)	1180160
max_pooling2d_9 (MaxPooling2	(None, 2, 2, 512)	0
dropout_9 (Dropout)	(None, 2, 2, 512)	0
flatten_1 (Flatten)	(None, 2048)	0
dense_3 (Dense)	(None, 256)	524544
batch_normalization_2 (Batch	(None, 256)	1024
dense_4 (Dense)	(None, 128)	32896
batch_normalization_3 (Batch	(None, 128)	512
dense_5 (Dense)	(None, 1)	129

Total params: 2,127,681
Trainable params: 2,126,913
Non-trainable params: 768

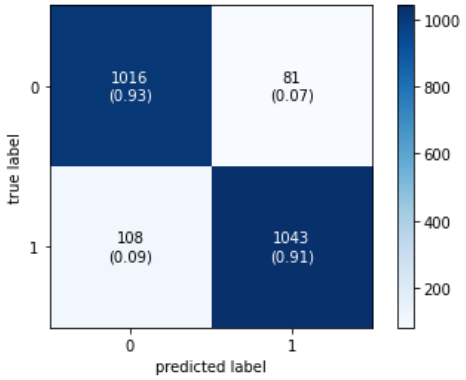
The confusion matrix shows that the model performed well on train and test data and the accuracy of true positive and true negative was balanced. The model performed similarly on test and train data and there was no overfitting. The regularization technique and image preprocessing for this model seems to have minimized any overfitting.

Train



	precision	recall	f1-score	support
0	0.91	0.94	0.93	9740
1	0.95	0.91	0.93	10510
accuracy			0.93	20250
macro avg	0.93	0.93	0.93	20250
weighted avg	0.93	0.93	0.93	20250

Test



	precision	recall	f1-score	support
0	0.90	0.93	0.91	1097
1	0.93	0.91	0.92	1151
accuracy			0.92	2248
macro avg	0.92	0.92	0.92	2248
weighted avg	0.92	0.92	0.92	2248

Summary of the 3 models' data

Model 2 with the imbalance accuracy performed below expectation and model 1 without regularization performed above expectation, but model 3 had the best train and test accuracy. Also, the accuracy and loss graph for model 3 showed that it did not appear to overfit and that suggests that it was not memorizing the training data.

	Number of Conv2D	Number of Dense layer	Processing Time	Training Accuracy	Test Accuracy
Model 1	3	1	1h 16min 8s	85.02	87.28
Model 2	3	1	1h 1min 26s	84.74	86.80
Model 3	5	2	1h 19min 7s	92.85	91.68

Implementation and programming

This assignment was done using google colab utilizing Tensoflow Keras packages, NumPy, scikit-learn and pandas.

I loaded the Pet Images data (dogs and cats) from Microsoft website into tmp Colab drive. The pet images folder contained a folder for Dog and Cat images. I created training and testing folders for the images.

I used matplotlib and OS packages for exploratory data analysis and to remove corrupt images.

Then I created a train, test and validation split using TensorFlow Data Image Generator function which generated batches of tensor image data with real-time data augmentation to normalize, standardize and transform the data.

Using the model method from Kera's, I created all 3 model objects, which included the model architecture. In that architecture, I used batch normalization (default) and dropout (0.2) regularization method on the hidden layer/layers from Kera's package.

Then I used the compile method to select the algorithm and optimizer and metric for the model.

I then fitted the model using 10 epochs and validation test of 0.2 and early stopping (for model 2 and 3 only). The model fit was saved in object (history).

I used python magic function %%time to calculate the processing time.

I used the model evaluate method to get the accuracy and loss on training and test data.

I used Kera's plot history function on the history of the model to plot the loss and accuracy plots for each epoch on the train and test/validation data.

I used confusion matrix function from scikit-learn and created a confusion matrix for test and train data for all 3 models.

I used the scikit-learn classification report to get the accuracy, recall and precision statistics for test and train data for all 3 models

Exposition, problem description, and management recommendations

Based on my analysis above Management should utilize a convolutional neural network (CNN) for their image classification problem, I would recommend Model 3, which has 5 convolution layers and 2 hidden dense layers This model performed best based on test and train accuracy and has the smallest difference between test and train accuracy, which along with the loss and accuracy graph confirm that the model does not overfit data. It is the least time efficient but the processing time difference between the model is not significant.

Additionally, Image augmentation does improve model performance in training and this should be further explored along with some changes in CNN architecture to improve the model performance.

Appendix

Code for 3 models

```
[139] model = tf.keras.models.Sequential([

        tf.keras.layers.Conv2D(16, (3, 3),  activation='relu', input_shape=(150, 150, 3)),
        tf.keras.layers.MaxPooling2D(2, 2),

        tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
        tf.keras.layers.MaxPooling2D(2, 2),

        tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
        tf.keras.layers.MaxPooling2D(2, 2),


        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(448, activation='relu'),
        tf.keras.layers.Dense(1, activation='sigmoid')
    ], name= 'base')
```

```
model = tf.keras.models.Sequential([

    tf.keras.layers.Conv2D(16, (3, 3),  activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Dropout(0.2)
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Dropout(0.2)
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),



    tf.keras.layers.Dropout(0.2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(448, activation='relu'),
    tf.keras.layers.BatchNormalization()
    tf.keras.layers.Dense(1, activation='sigmoid')
], name= 'base + Regularization')
```

```
model = tf.keras.models.Sequential([


    tf.keras.layers.Conv2D(32, (3, 3),padding="same", activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),


    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(128, (3, 3),  activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(256, (3, 3),  activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(512, (3, 3),  activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(1, activation='sigmoid')
],name= 'best_CNN')
```

Image Data generator for train , test and validated

Validation_generator_valid = test data

```
[134] TRAINING_DIR = "/tmp/cats_v_dogs/training/"
      train_datagen = ImageDataGenerator(rescale=1.0/255.,validation_split=0.1,featurewise_center=True,
          featurewise_std_normalization=True,
          rotation_range=20,
          width_shift_range=0.2,
          height_shift_range=0.2,
          horizontal_flip=True #, preprocessing_function = img_transf
                          )
      train_generator = train_datagen.flow_from_directory(TRAINING_DIR, classes=["cats","dogs"],
                                                          batch_size=250,
                                                          class_mode='binary',
                                                          target_size=(150, 150), subset='training'
                                                          #,shuffle=False
                                                          )

      VALIDATION_DIR = "/tmp/cats_v_dogs/testing/"
      validation_datagen = ImageDataGenerator(rescale=1.0/255.)
      validation_generator = validation_datagen.flow_from_directory(VALIDATION_DIR,classes=["cats","dogs"],
                                                                    batch_size=250,
                                                                    class_mode='binary',
                                                                    target_size=(150, 150 ))

      validation_generator_valid = train_datagen.flow_from_directory(
          TRAINING_DIR, # same directory as training data
        #  target_size=(img_height, img_width),
          batch_size=250,classes=["cats","dogs"],
          class_mode='binary',
          subset='validation',target_size=(150, 150 ), shuffle=False)
```

```
Found 20250 images belonging to 2 classes.
Found 2500 images belonging to 2 classes.
Found 2248 images belonging to 2 classes.
```