RAMAIAH
Institute of Technology

Department of Information Science and Engineering

Report on

Dogs vs Cats Image Classification

Submitted in partial fulfilment of the CIE for the subject :
Deep Learning (ISE741)

By

Ishan Sharma – 1MS19IS050

Under the guidance of
Dr. Vijaya Kumar B P
Dept. of ISE, MSRIT
Bengaluru-560054

Dogs vs Cats Image Classification

- **Importing packages**
  - os — to read files and directory structure
  - numpy — for some matrix math outside of TensorFlow
  - matplotlib.pyplot — to plot the graph and display images in our training and validation data

- **Data Loading**

  The dataset we are using is a filtered version of Dogs vs. Cats dataset from Kaggle.

- **Setting Model Parameters**

  For convenience, let us set up variables that will be used later while pre-processing our dataset and training our network.

  BATCH_SIZE = 100
  IMG_SHAPE = 150 Our training data consists of images with width of 150 pixels and height of 150 pixels

  After defining our generators for training and validation images, flow_from_directory method will load images from the disk and will apply rescaling and will resize them into required dimensions using single line of code.

- **Data Augmentation**

  Overfitting often occurs when we have a small number of training examples. One way to fix this problem is to augment our dataset so that it has sufficient number and variety of training examples. Data augmentation takes the approach of generating more training data from existing training samples, by augmenting the samples through random transformations that yield believable-looking images. The goal is that at training time, your model will never see the exact same picture twice. This exposes the model to more aspects of the data, allowing it to generalize better.

In tf.keras we can implement this using the same ImageDataGenerator class we used before. We can simply pass different transformations we would want to our dataset as a form of arguments and it will take care of applying it to the dataset during our training process.

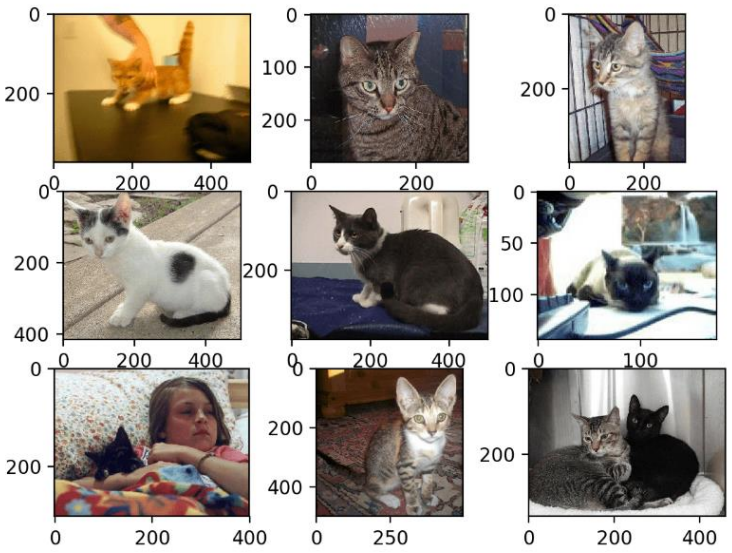
To start off, let's define a function that can display an image, so we can see the type of augmentation that has been performed. Then, we'll look at specific augmentations that we'll use during training.

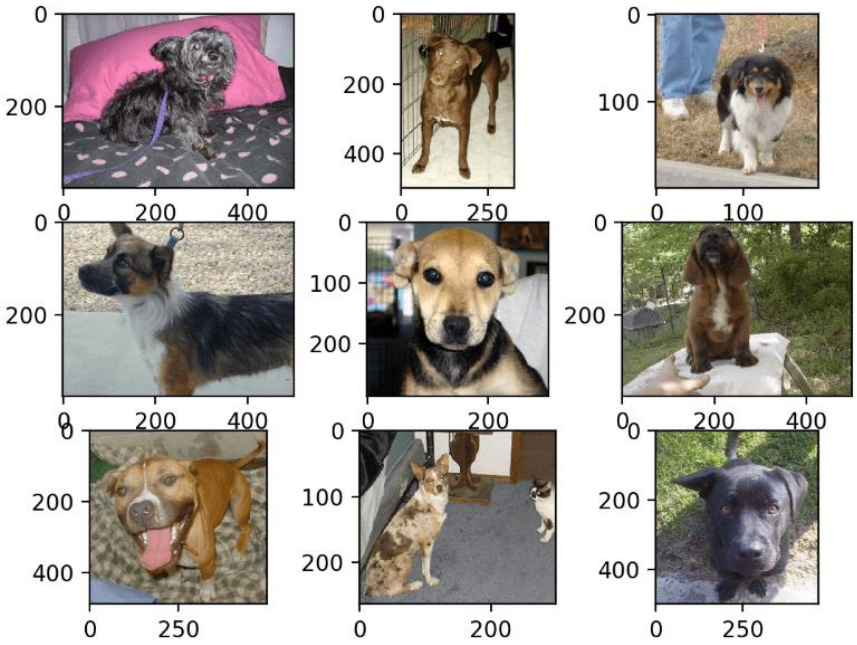
- **Flipping the image horizontally**

We can begin by randomly applying horizontal flip augmentation to our dataset and seeing how individual images will look after the transformation. This is achieved by passing horizontal_flip=True as an argument to the ImageDataGenerator class.
To see the transformation in action, let's take one sample image from our training set and repeat it five times. The augmentation will be randomly applied (or not) to each repetition.

- **Rotating the image**

The rotation augmentation will randomly rotate the image up to a specfied number of degrees. Here, we'll set it to 45.
To see the transformation in action, let's once again take a sample image from our training set and repeat it. The augmentation will be randomly applied (or not) to each repetition.





Applying Zoom

We can also apply Zoom augmentation to our dataset, zooming images up to 50% randomly.

One more time, take a sample image from our training set and repeat it. The augmentation will be randomly applied (or not) to each repetition.

- **Putting it all together**

We can apply all these augmentations, and even others, with just one line of code, by passing the augmentations as arguments with proper values.

Here, we have applied rescale, rotation of 45 degrees, width shift, height shift, horizontal flip, and zoom augmentation to our training images.

Let's visualize how a single image would look like five different times, when we pass these augmentations randomly to our dataset.

- **Creating Validation Data generator**

Generally, we only apply data augmentation to our training examples, since the original images should be representative of what our model needs to manage. So, in this case we are only rescaling our validation images and converting them into batchesusing ImageDataGenerator.

The model consists of four convolution blocks with a max pool layer in each of them.

Before the final Dense layers, we're also applying a Dropout probability of 0.5. This mean that 50% of the values coming into the Dropout layer will be set to zero. This helps to prevent overfitting.

Then we have a fully connected layer with 512 units, with a relu activation function. The model will output class probabilities for two classes — dogs and cats — using softmax.

```
def build_model():
  model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),

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

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

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

    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(2, activation='softmax')
  ])
```

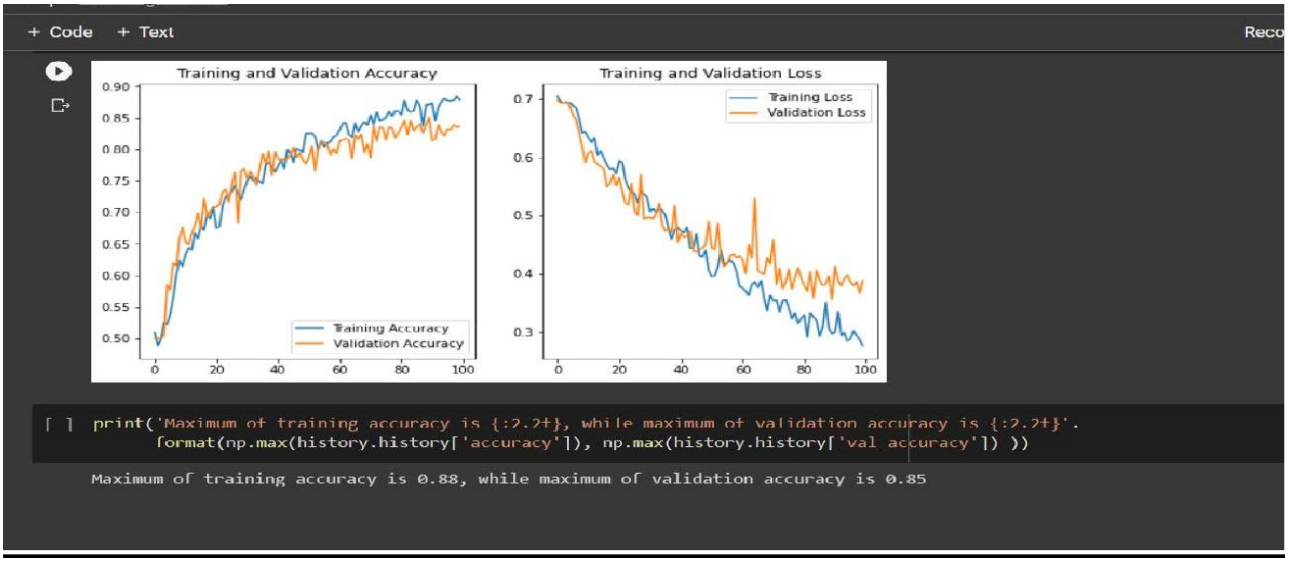
```
# The patience parameter is the amount of epochs to check for improvement
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_acc', patience=10)


EPOCHS = 100
history = model.fit_generator(
    train_data_gen,
    steps_per_epoch=int(np.ceil(total_train / float(BATCH_SIZE))),
    epochs=EPOCHS,
    validation_data=val_data_gen,
    validation_steps=int(np.ceil(total_val / float(BATCH_SIZE))),
    callbacks=[early_stop])
```

```
20/20 [==============================] - 17s 869ms/step - loss: 0.5362 - accuracy: 0.7195 - val_loss: 0.4941 - val_accuracy: 0.7660
Epoch 30/100
20/20 [==============================] - ETA: 0s - loss: 0.5319 - accuracy: 0.7365WARNING:tensorflow:Early stopping conditioned on metric `val_
20/20 [==============================] - 17s 858ms/step - loss: 0.5319 - accuracy: 0.7365 - val_loss: 0.4959 - val_accuracy: 0.7690
Epoch 31/100
20/20 [==============================] - ETA: 0s - loss: 0.5069 - accuracy: 0.7495WARNING:tensorflow:Early stopping conditioned on metric `val_
20/20 [==============================] - 17s 838ms/step - loss: 0.5069 - accuracy: 0.7495 - val_loss: 0.4963 - val_accuracy: 0.7530
Epoch 32/100
20/20 [==============================]  ETA: 0s  loss: 0.5115  accuracy: 0.7580WARNING:tensorflow:Early stopping conditioned on metric `val_
20/20 [==============================] - 17s 877ms/step - loss: 0.5115 - accuracy: 0.7580 - val_loss: 0.4942 - val_accuracy: 0.7650
Epoch 33/100
```

✓ 1s completed at 12:59

- **Visualizing results of the training**




```
print('Maximum of training accuracy is {:2.2f}, while maximum of validation accuracy is {:2.2f}'.
      format(np.max(history.history['accuracy']), np.max(history.history['val_accuracy'])))
```

```
Maximum of training accuracy is 0.88, while maximum of validation accuracy is 0.85
```