MACHINE LEARNING III ASSIGNMENT 1

1. CATS & DOGS

The first exercise consisted of building a neural net that is able to classify images of both cats and dogs. The first exercise consisted of three questions whereby the notebook was split into existing code and the requirements of the first assignment:

Existing code:

As seen below, the neural network consist of provided code being the input layer and the first convolution and max-pooling layer. The input layer takes in images that are 150x150 and is tuned to accept the three-color channels; red, green, blue:

```
img_input = layers.Input(shape=(150, 150, 3)).
```

The next two layers exist of a convolution layer with 16 filters that are 3x3, followed by a max-pooling layer with a 2x2 window:

```
x = layers.Conv2D(16, 3, activation='relu')(img_input)
x = layers.MaxPooling2D(2)(x)
```

[1 point] TODO 1. Add a convolutional layer with the specifications defined in the code.

Following the existing code, we created two layers with a convolution layer containing 32 filters that are 3x3, followed by a max-pooling layer with a 2x2 window:

```
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
```

Lastly, we create an additional two layers as before, a convolution layer with 54 filters that are 3x3 and another max-pooling of a 2x2 window:

```
x = layers.Conv2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
```

[1 point] TODO 2. Flatten the output of the last convolutional layer to add a dense layer.

```
x = layers.Flatten()(x)
```

[1 point] TODO 3. Add the dense layer with the specifications defined in the code.

```
x = layers.Dense(512, activation='relu')(x)
output = layers.Dense(1, activation='sigmoid')(x)
```

2. REGULARIZATION

[1 point] TODO 1. Data Augmentation. Explain how ImageDataGenerator is working. Specifically, explain what all the parameters, already set, and their respective values mean. One by one, from rotation_range to fill_mode.

As our dataset is small, data augmentation is used to increase the size of our dataset. Data augmentation through the "imageDataGenerator" creates new images five times, where the model never sees the same augmented image twice. This allows us to train our model better by avoiding overfitting. The below table details the different changes that can be done:

Function	Function Process[1]	Output[2]
Rotation range	Rotates the image, randomly, by degrees. The function allows for a range between 0-180.	Each iteration of each image is rotated, randomly, by up to a maximum of 40 degrees either left or right (images can be rotated to 1, 2, 3,…, 10, …, 15, up to 40)
Width shift / Height shift	Width Shift: Used to randomly widen or make the image thinner by a factor that is between 0-1 Height shift: Used to randomly elongate or shorten the image by a factor that is between 0-1	Both the Width shift & Height shift are set to .2, whereby for: - Width Shift: each image is up to 20 % wider or thinner than the original image size - Height shift: each image is up to 20 % longer or shorter than the original image
Shear range	This function randomly shifts all points in an image in any direction in parallel, whereby the input is between 0 & 1	The image is shifted randomly in any direction by a maximum of 20 % each time. Whereby it can be shifted up, down left, or right.
Zoom range	Randomly zooms, whereby the input is between 0-1	The image is randomly zoomed in by maximum of 20 %
Horizontal flip	Flips half the images, randomly, horizontally. The input here is True or False.	The code allows for horizontal flips (true), whereby half the images will be transposed with a horizontal flip
Fill mode	Fills in any newly created pixels created by any of the above aforementioned functions. Can be set to constant, nearest, reflect, or wrap.	Here the images are filled with Nearest, whereby each image newly created pixels. This is the default option where the closest pixel value is chosen and repeated for all the empty values. (E.g. aaaaaa|abcd|dddd)

1 Here the input puts a max limit on whatever number is inserted into the function
2 It should be noted that a .flow() function is used whereby augmenting each picture 5 times.

[1,5 points] TODO 2. Dropout.

Explain Dropout as a regularization technique.

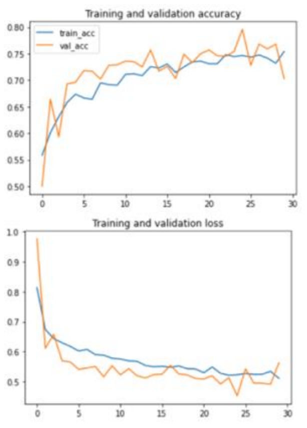
Dropout is a regularization technique, created by Google, with the aim of reducing overfitting. The thought behind this technique is that each node has to rely on itself and not the output of other nodes. Each individual node must create a productive output on its own. Thus, usually set between 0.2-0.5. If set to 0.3, 30 % of the values in a vector produced by an individual node in a vector will be replaced by 0's.

Add a Dropout layer with a dropout rate of 0.5.

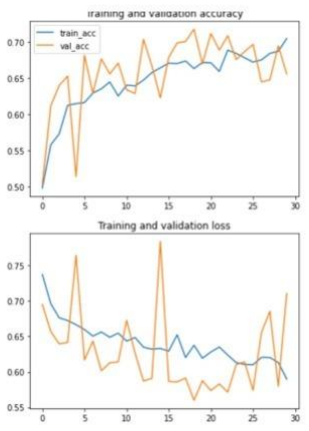
```
x = layers.Dropout(0.5)(x)
```

Try dropout of 0.9 and 0.1 rate. Explain the different behavior.

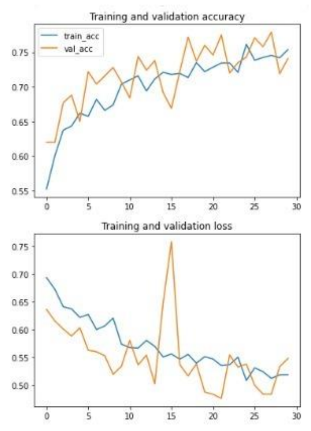
When compared to using a dropout of 0.5, the model accuracy decreases when using a dropout of 0.9 and 0.1. As mentioned previously, the dropout adds 0's, whereby we can see that the model accuracy decreases significantly when training with a dropout of 0.9, replacing 90% of the vector of an individual node to 0's to an accuracy average of approximately 0.65 whilst with a dropout of 0.1 it decreases to 0.73. Thus, it can be extrapolated that each model needs to be fine-tuned to find the best accuracy with the least loss for each model.



Dropout rate 0.5



Dropout rate 0.9



Dropout rate 0.1

[0,5 points] TODO 3. Fit the final model following the specifications in the code.

```
history = model.fit(
    train_generator,
    steps_per_epoch=100,
    epochs=30,
    validation_data=validation_generator,
    validation_steps=50,
    verbose=2)
```

3. STAND ON THE SHOULDERS OF GIANTS

[1 point] TODO 1. Do some research and explain the inception V3 topology. Which database do they use for training it? How was it trained?

Created by Google, Inception V3 is the third iteration of the model. The model uses databases such as Imagenet, although consisting of over 10 million pre-labeled images, inception uses over 1.3 million images. Here it is split into train and test, whereby only images from the train are used during the training process and only images from the test are used to evaluate model accuracy.

InceptionV3 uses the concept of transfer learning. As the model is pre-trained on over 1 million images, transfer learning allows a user to retrain the final layers of an existing model. This allows the model to retain what it has learned with the original training and reapply it to a smaller dataset. Thus, resulting in a higher accuracy with less training time and CPU required. This can be seen with the code function "unfreeze mixed 7" and "Mixed 6" in our notebooks. It has allowed us to unfreeze those layers and reteach them with new and improved weights in order to make our model more accurate.

Furthermore, this version regularizes data whereby introducing 4 new concepts, all techniques to improve overall model accuracy:

1. RMSProp Optimizer.
2. Factorized 7x7 convolutions.
3. BatchNorm in the Auxillary Classifiers.
4. Label Smoothing

[1 point] TODO 2. Inception V3 is set by default to admit input images of (299, 299, 3) dimensions; but we want (and we will use) inputs of (150, 150, 3). If the net is already trained, how is that even possible?

The model does allow for a change in input size, whereby one has to include the code input_shape=((150, 150, 3), include_top=False, weights=None). This allows one to change the input size where the minimum is 75x75. The last layer output changes as you decrease the image input shape whereby in **CNN Cats_vs_Dogs Exercise 3.ipynb** the last layer output shape decreases to (None, 7, 7, 768), as the images are 150x150 whilst in the **CNN Flowers Exercise 3.ipynb** it reduces to (None, 4, 4, 768). Furthermore, the model retrains itself to produce less total and training parameters whilst the non-trainable parameters are the same.

In **CNN Cats_vs_Dogs Exercise 3.ipynb** the output is:
Total params: 47,512,481, Trainable params: 38,537,217, Non-trainable params: 8,975,264

In **CNN Flowers Exercise 3.ipynb** the output is**:**
Total params: 21,564,325, Trainable params: 12,589,061, Non-trainable params: 8,975,264

One must then resize all the images to 100x100 using the function before running the model:

```
IMG_SIZE = 100 # All images will be resized to 100x100
def format_example(image, label):
  image = tf.cast(image, tf.float32)
  image = image = image / 255.0
  image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
  return image, label
```

[2 points] TODO 3. Use Inception V3 to outperform the results we obtained in the Fashion MNIST database with our first neural net. This is, run Inception V3 with the other database.

See notebook **CNN Flowers Exercise 3.ipynb.**

4. YOLO_V3 [Advanced & Optional] [3 extra points]

What Is YOLOv3?

You Only Look Once (YOLO) is a neural network made for detecting objects and their location in the image created by Joseph Redmon. Yolov3 just means that it is the third version of YOLO, and it is exciting news that YOLOv4 has already been launched in April 2020. Object detection is a task in computer vision to detect objects, their location, their magnitude, and what they are. In other words, it is algorithms you use to classify images, is it a cat or a dog? YOLOv3 is in the category of Region Proposal Classification Network (RCNN), but compared to its' siblings, YOLO looks more like a Fully Convolutional Neural Network (FCNN). It is an algorithm using DarkNet and it is written in C.

Each image is associated with a label file saved as .txt file, and to be able to get this we have three options; we can build them from scratch with our own pictures and labels using a tool, we can use already labeled images or we can use pre-trained models. YOLOv3 is basically just another state-of-the-art (SOA), real-time object detection system for images and videos.

For the optional exercise we have made a tutorial uploaded to GitHub. This can be accesses through this link:
https://github.com/Jonashellevang/IE_MBD_2020/tree/master/YOLOv3%20Tutorial

The file contains a description of YOLOv3, DarkNet and a step by step procedure of how to run the code. We were also able to modify some code to download the validation images where it put labels on photos with guns. All the photos have been uploaded to Google Drive, and if you can't access the link, please let us know, although it should not be an issue:
https://drive.google.com/drive/folders/1--BNMdyp5gpVoK15XO76I4h_6AHgAB3P?usp=sharing