MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

GRAPHICAL PROCESSING UNITS FOR DEEP LEARNING

A undergraduate Seminar Report submitted to MAHE, Manipal in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

Instrumentation and Control Engineering

Submitted by

Saransh Agrawal

Reg. No. 170921060

DEPARTMENT OF INSTRUMENTATION AND CONTROL ENGINEERING

MANIPAL INSTITUTE OF TECHNOLOGY

NOVEMBER 2020

ABSTRACT

GPU (Graphical Processing Unit) enables faster matrix calculations than a CPU (Central Processing Unit). This is an important piece of hardware when it comes to digital image processing for various applications. The research in the development of the GPUs gave rise to research in Artificial Intelligence. Training a model for supervised learning can be an extremely time consuming task. Soon enough, the solution was found in the parallel processing power that the GPU offers. In my study, I have attempted to show how the training time compares for the CPU and the GPU.

Contents

Chapter 1

Introduction

Deep Learning is a part of Machine Learning that has revolutionised the way rules are defined for a complex problem. While it's intellectually difficult for a human to find patterns and mathematically describe it with a set of rules for an unstructured data, for a machine it's relatively easy. In the Deep Learning approach of Artificial Intelligence, a human writes a set of simple mathematical equations and stacks them, forming layers. A computer then iterates over these equations repeatedly, to adjust the weights and biases of these individual equations, until a suitable probability of the output being correct is reached. With this basic framework, many different applications have been developed that sometimes even surpass the probability for human correctness. Some of the applications include facial recognition, object classification and segmentation. The success of Deep Learning can be roughly credited to three major developments: [1]

1. New and improved algorithms
2. Availability of big data for training
3. Increasing computational power of GPUs

In this study we mainly look at how the availability of computational power, on specifically designed computers help carry out the training of various Deep

Learning models affected the growth of research in the field.

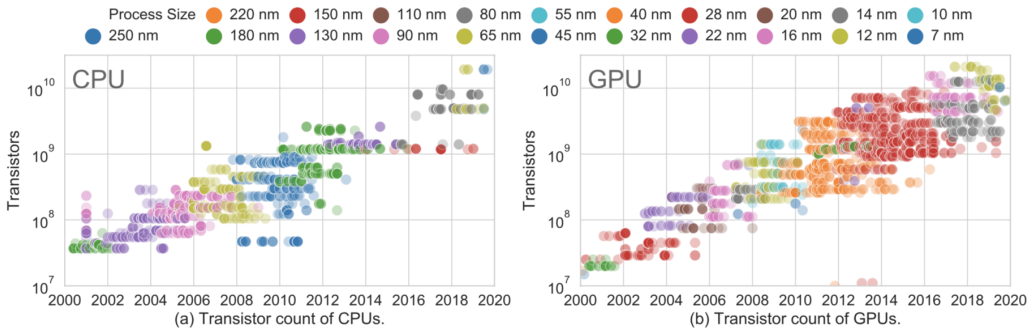

Figure 1.1: Moore's Law is still valid for both CPUs and GPUs.

The trend in Figure 1.1 clearly shows that the Moore's Law is still followed. The data [2] depicts that both CPUs and GPUs may have an equivalent number of transistors it is in fact the architecture of the GPU that makes it much more useful for training deep learning models rather than a CPU.

Chapter 2

Machine Learning

Machine Learning is a way of programming where we try to find the rules connecting the outputs to inputs. This is what is called as learning, since our algorithm starts with a random set of equations and tries to fit the output in these sets of equations by changing the coefficients of these equations to give the output.

Steps involved are:

1. **Pre-Processing:** Involves subjecting a dataset to various enhancements through known pre-processing methods, such as filling missing values, removing incorrect data.

2. **Division of Dataset:** The dataset is divided into three groups, Train set, Development set, Test set. Generally the ratio of 60:20:20 is used to divide the dataset.

3. **Model selection:** The most appropriate model is selected, which consists of either linear, polynomial or a set of interconnecting equations to reach the desired output.

4. **Initialization:** Randomly selecting the coefficients for these equations.

5. **Forward Propagation:** Involves taking a set of inputs and propagating it through the equations to reach the predicted output.

6. **Comparison with Actual Output:** The predicted output ($\hat{y}$) is compared with the actual output(γ). For this, an appropriate loss function is used, such as mean absolute error, mean squared error, cross entropy loss.

7. **Back Propagation:** With a non-zero loss, gradient descent is performed, with appropriate derivative of the equation used.

8. **Update Coefficients:** The derivative is multiplied with a learning rate (α) and subtracted from the current coefficient to find the new coefficient.

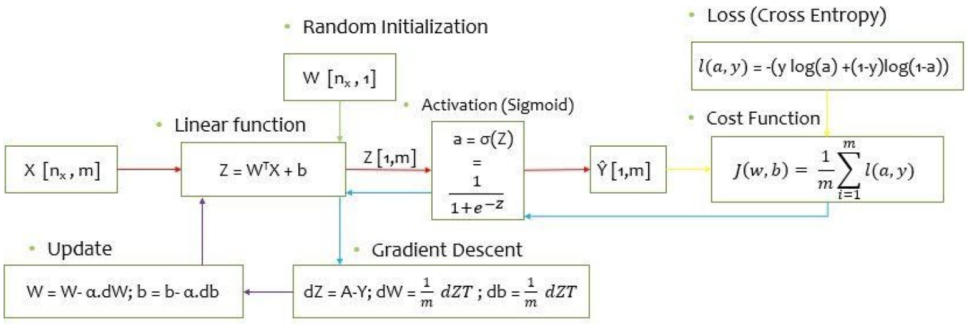

Figure 2.1: Block Diagram of Logistic Regression.

Random initialization: The weight matrix is initialized with random numbers that are close to zero, the matrix is fixed to the shape $[\eta, 1]$ for a simple logistic regression. However, the general matrix form is [number of neurons in previous layer, number of neurons in current layer].

Linear function: The neuron contains of two parts - a linear function and an activation function. In the linear function, the computer learns the weights and bias, and gives a specific output.

Activation function: The task of the activation function is to make sure the output of linear function would map to a desired range. There are

many activation functions, such as Sigmoid, Tanh and Relu. For a binary classification task, we would want our output to be mapped to [0,1] and here, we commonly use a Sigmoid or a Tanh function.

Loss and Cost: Loss determines the distance between our predicted output ($\hat{y}$) and actual output (y). It is a key component that guides our gradient descent to move in a particular direction. Our aim is to reduce cost, and we move in a direction which reduces the cost. Cost is calculated for the entire training batch, and is the mean, or average, calculated by adding the losses of individual instances and dividing by the total number of instances.

Gradient Descent: In gradient descent, we attempt to back propagate the loss by obtaining a derivative of loss with respect to z. With the obtained dz/dl, the partial derivative of l with respect to weight w and bias b can be obtained, for updation.

Update: The partial derivatives of loss with respect to weight obtained in the previous step is multiplied to a learning rate (α) and then subtracted from the current weight, and this new updated weight is expected to decrease cost of the model in the next iteration.

Learning Rate (α): This hyperparameter determines how big of a step our gradient descent will take. A larger learning rate may make the model over step and skip the global minima, while a smaller learning rate may increase training time of the model. A lower learning rate can also lead a model to get stuck at a local minima, and stop training early. Various regularization techniques are used to overcome this problem.

Chapter 3

Deep Neural Network

Deep Neural Network is an architecture, where several simple neurons are connected to form a layer, and several layers are interconnected to form a network. The intention is that the neurons in initial layers will learn to recognize the basic features of an input while the farther layers will build on that information and learn more complicated features. By doing so, the potential of every single neuron lead to solve a complicated problem by having each neuron learn a basic feature.

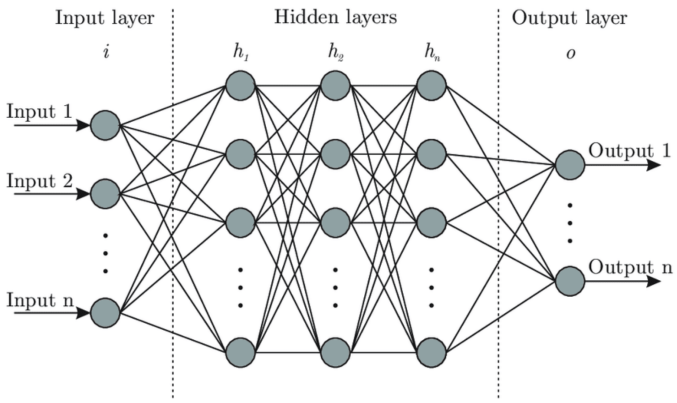

Figure 3.1: Deep Neural Network

Input Layer: This layer accepts the features of an input, the dimension of this layer is affected by the number of features an input possesses. The intention behind this layer is to learn the very basics of an instance, for example

in an image classification task, this model will learn to recognize certain colours or edges in an image.

Hidden Layers: These layers start combining the features learnt from the previous layers to make a new feature, for example, it can combine the edges detected with colour detected outputs from the previous layer and make a red edge detected output. This output will be carried forward in the layers to form more complex layers, and more complex features.

Output Layer: The number of neurons in this layer is determined by the number of predictions the model is expected to print. For binary classification one neuron is enough, with sigmoid activation function, however for multi class classification, with softmax activation function, one neuron will be dedicated to output the probability of one individual class.

Chapter 4

Convolutional Neural Network

A Convolutional Neural Network, or CNN, is a special type of neural network where at least one of the layers is performing Convolutions on the inputs.

$$s(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{4.1}$$

The general equation of convolution is given in Equation 4.1 [1]. In Machine Learning, the input is usually multidimensional and the kernel(w), also called filter, is a multidimensional array of parameters that are learnt by the computer.

One of the key steps of convolution is flipping the kernel to fulfil the commutative property of convolution. While it is necessary for proving the applications of neural networks, it's not a necessity, because our aim is to learn the weights of the kernel and flipping will only add an unnecessary step that is computationally expensive and ultimately our algorithm will just learn the flipped version of kernel. Thus for practical purposes, cross-correlation is performed, as seen in Equation 4.2.

$$s(i,j) = (I \circledast K)(i,j) = \sum_m \sum_n I(i+mc, j+n)K(m,n) \tag{4.2}$$

A convolutional neural network (Figure 4.1) has three advantages over a normal neural network: [1]

1. **Sparse interactions:** In normal neural networks, all neurons interact with all the neurons of neighbouring layers, while a CNN has sparse interactions, achieved by making the kernel smaller than input size. So the weight matrix need not be as big as the number of inputs, it is comparatively very small.

2. **Parameter sharing:** In CNN, a small kernel is placed and used on the entirety of the input, thus every parameter sees the entire input unlike to neural network.

3. **Equivariant representations:** When the input to a convolution changes, the output changes in the same way, so if any transformation is done to the input then the convolution kernel will reflect the same change in the output.

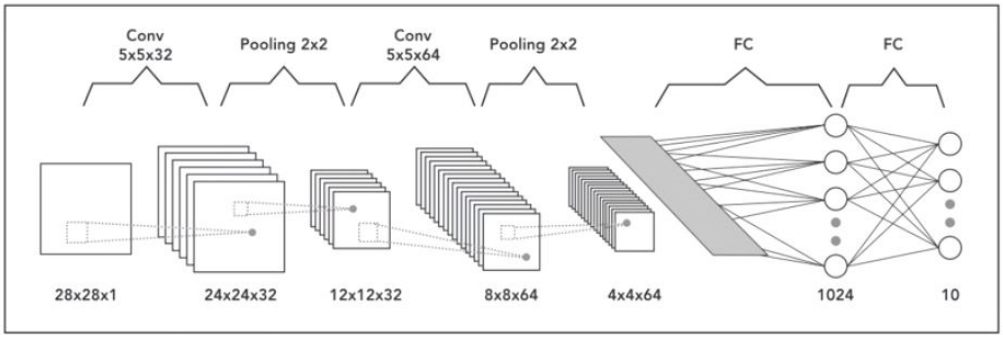

Figure 4.1: Architecture of Convolutional Neural Network [3]

Chapter 5

Graphical Processing Unit (GPU)

A Graphical Processing Unit, or GPU, was designed to display images on the monitor. It was designed to work in conjunction with the Central Processing Unit, and take the graphical workload off the CPU [4]. GPU has a specially designed architecture that makes it powerful, allowing it to render images faster than a CPU. For many years, GPUs found applications mainly in the graphical industry, for instance in computer gaming and architectural design, however, due to the unique architecture of the GPU, which allows them to process simple matrix calculations faster than any other CPU, the GPU has found its use in the Artificial Intelligence industry.

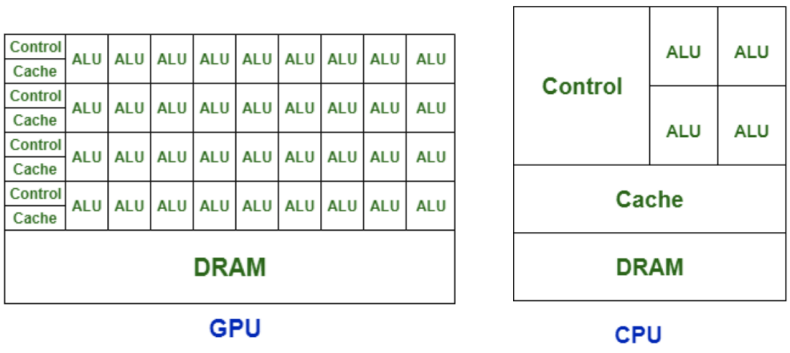

Figure 5.1: Architecture of GPU vs CPU

The architectural differences are elaborated below: [5]

1. **Cores:** A core is composed of an Arithmetic Logic Unit, Cache Memory and Controller. A core describes an individual part of the processor that is capable of processing computations on its own. While a typical CPU has 4-8 cores, the GPU can have anywhere between 300 and up to a few thousand cores in more advanced GPUs. This drastic change allows the GPU to have more parallel processing capabilities than a CPU.

2. **Arithmetic Logic Unit:** As it is evident from the Figure 5.1, the CPUs do not lack when it comes to the number of transistors present, as compared to a GPU. However, in a CPU, the ALU is very long, having large number of logic gates. The presence of these logic gates accounts for the large number of transistors used. The ALU of a CPU is thus robust and can perform complex calculations in fewer number of clock cycles. The ALU of a GPU is very different from a CPU, containing less number of logic gates which can solve easy equations but would require a large number of clock cycles to solve complex equations.

3. **Clock Frequency:** The Clock Frequency of a CPU is very high compared to that of a GPU, because the clock has to run a relatively low number of cores, while for the GPU, the clock is utilized by many cores, all working at the same time.

5.1 What makes GPU better for Deep Learning?

As it is evident from Chapters 2, 3 and 4 a deep learning model is a combination of linear equations connected to form a network, the network can learn complicated tasks but the core of it is simple linear equations. Due the higher

feature size that propagates through the network, every neuron is performing multiple linear equations to learn a weight associated with a feature received from the previous layer, and many such neurons are present in a layer. A CPU can solve a single linear equation faster than a GPU, but the GPU can solve multiple linear equations faster than a CPU because of its parallel processing power.

Chapter 6

Experimental Demonstration of Performance

CPU – Intel core i5-8250U	**Nvidia GeForce GT-1030**	**Nvidia T4 Tensor core**
Cores: 4, Threads: 8	CUDA cores: 384	CUDA cores: 2560, Turing Tensor cores: 320
Base frequency: 1.60 GHz, MAX Turbo Freq: 3.4 GHz	Base clock frequency: 1228 MHz, Boost: 1468 MHz	Base clock frequency: 585 MHz, Boost: 1590 MHz
RAM: 8GB, MAX Bandwidth: 37.5 Gb/s	RAM: 2GB DDR5, MAX Bandwidth: 48GB/s	RAM: 16GB GDDR6, MAX Bandwidth: 300Gb/s
	CUDA compute compatibility: 6.1	CUDA compute compatibility: 7.5

Table 6.1: Comparing Three Systems

[6] [7] [8]

Cores: It is clearly visible the scale of difference between the number of

cores. The GeForce GT-1030 has 96 times more cores than the CPU, while T4 has 720 times more cores than the CPU.

Clock Frequency: The CPU base cock is 1.3 times faster than the GT-1030 and 2.7 times faster than the T4.

RAM: The Intel CPU supports up to 32GB of RAM while the GPUs come with fixed RAM. For the GT-1030, it is 2GB, while for T4 it is 16GB. The T4 also has a higher bandwidth than the CPU and GT-1030 GPU.

CUDA compute compatibility: This is a rating given by Nvidia to its GPUs, based on several factors which amount to a general idea about the GPU's performance.

6.1 Model

A model is created, that accepts an RGB image and classify them based on whether it's a cat or a dog. The task is binary classification and appropriate activation functions are used as described below. B.1 The Model [9] is built to accept an RGB input image of size [256,256,3], while the batch size can be given by the user. There are a total of 15 CONV2D layers containing 32,64,128,256,512 number of filters progressively. 6 MAX POOLING layers are interconnected in between the model to allow feature reduction. After all the convolutional layers a FLATTEN layer is attached to make the data ready to be accepted by the Dense Layers. The output of Flatten layer is 4608 features. A.1 The output of Flatten is followed by 3 Dense layers with appropriate use of Dropout and Batch Normalization to regularize the model while training. The last dense layer uses a sigmoid activation function that maps the final output between 0 and 1, signifying the two classes. Loss function used is Binary Cross Entropy which provides a good penalty for binary classification tasks and make the model train faster. The total parameters in the model are: 21,463,713 out of which 21,455,521 are trainable.

6.2 Performance on Different Systems

6.2.1 Performance on Intel core i5-8250U CPU

The images are normalized and made to the appropriate size of [256,256,3] by sampling. The batch size was chosen to be 16, while the dataset contains 20,000 classified images to be trained. The result was that training was very slow and even with complete CPU utilization the expected time for epoch 1 to be completed was 2 hours and hence the model would take too much time to train.

6.2.2 Performance on Nvidia GeForce GT-1030

The training conditions are kept same as was for the CPU. However, this GPU surpassed the performance of the CPU drastically. With an average time of 11 minutes per epoch with complete GPU utilization, the training was boosted drastically. The disadvantage, however, came when the batch size was increased. Due to small RAM size of 2GB, out of which only 1.6 GB was available to use, there was not enough memory to allocate the high matrix sizes. An out of memory (OOM) error occurred and the model stopped training.

6.2.3 Performance on Nvidia T4 Tensor core

With the availability of more RAM, the batch size was selected to be 128, to allow complete utilization of the GPU. While keeping all other parameters the same as that in previous cases, the T4 surpassed the above two processors and got an average of 6.3 minutes per epoch.

6.2.4 Final Training

The final model was hence trained on the Nvidia T4 and reached a 96.4% accuracy in under 30 epochs, which amount to a total training time of 189 minutes. The training graphs are shown in Figures 6.1 and 6.2. [10]

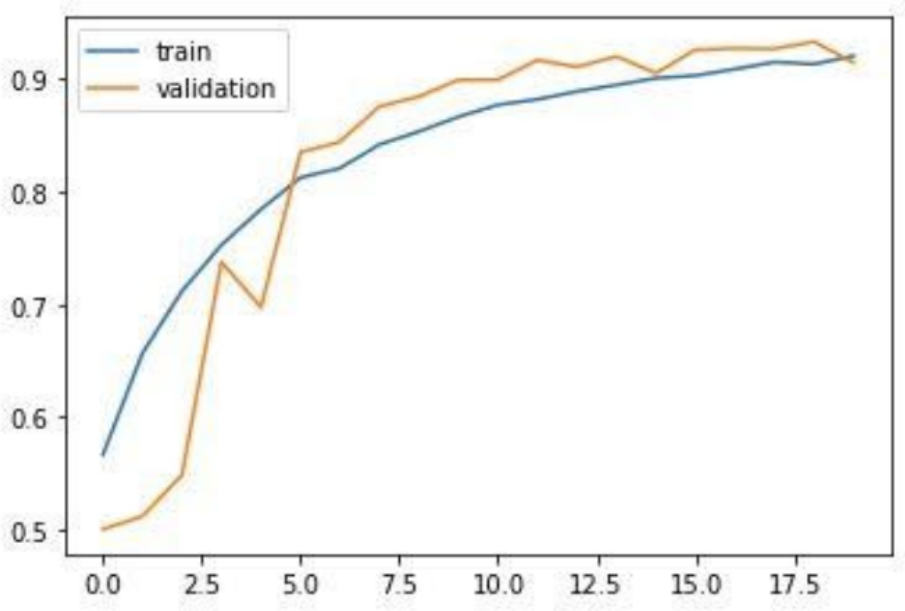

Figure 6.1: Accuracy versus Epoch

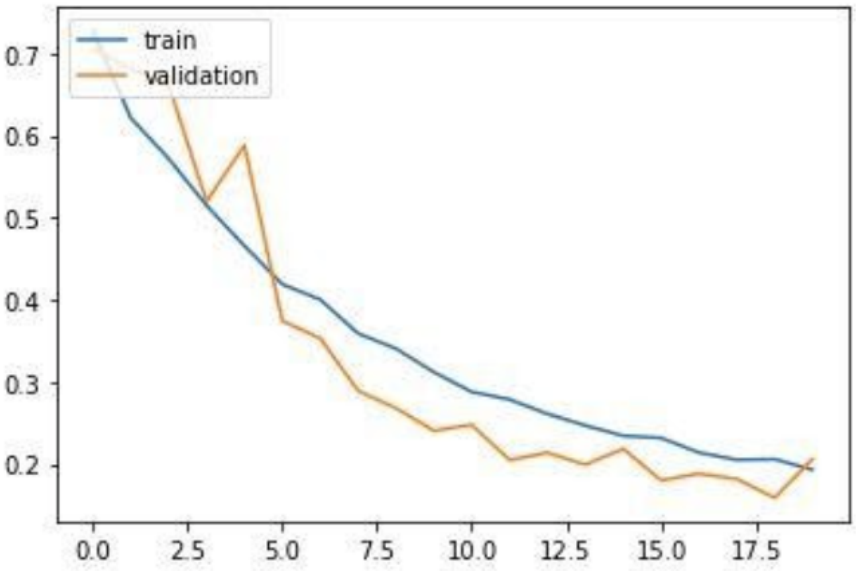

Figure 6.2: Loss versus Epoch

Appendices

Appendix A

Code

A.1 Cats Versus Dogs

```
from google.colab import drive
drive.mount('/content/drive')



import tensorflow as tf  \cite{tensorflow2015-whitepaper}
import matplotlib.pyplot as plt \cite{hunter2007matplotlib}
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator

image = plt.imread("D:\python_practice_deeplearning\data\\train\cats\cat.60.jpg")
plt.imshow(image)

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation = 'relu', padding = 'same', input_shape =(256,256,3)),
    tf.keras.layers.Conv2D(32, (3,3), activation = 'relu', padding = 'same'),
    tf.keras.layers.MaxPooling2D(pool_size = (2,2), strides = (2,2)),
    tf.keras.layers.Conv2D(64, (3,3), activation = 'relu', padding = 'same'),
    tf.keras.layers.Conv2D(64, (3,3), activation = 'relu', padding = 'same'),
    tf.keras.layers.MaxPooling2D(pool_size = (2,2), strides = (2,2)),
    tf.keras.layers.Conv2D(128, (3,3), activation = 'relu', padding = 'same'),
    tf.keras.layers.Conv2D(128, (3,3), activation = 'relu', padding = 'same'),
    tf.keras.layers.Conv2D(128, (3,3), activation = 'relu', padding = 'same'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2), strides = (2,2)),
    tf.keras.layers.Conv2D(256, (3,3), activation = 'relu', padding = 'same'),
    tf.keras.layers.Conv2D(256, (3,3), activation = 'relu', padding = 'same'),
    tf.keras.layers.Conv2D(256, (3,3), activation = 'relu', padding = 'same'),
    tf.keras.layers.MaxPooling2D(pool_size=(3,3), strides = (3,3)),
```

```
    tf.keras.layers.Conv2D(512, (3,3), activation = 'relu', padding = 'same'),
    tf.keras.layers.Conv2D(512, (3,3), activation = 'relu', padding = 'same'),
    tf.keras.layers.Conv2D(512, (3,3), activation = 'relu', padding = 'same'),
    tf.keras.layers.MaxPooling2D(pool_size=(3,3), strides = (3,3)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(2048, activation = 'relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(2048, activation = 'relu'),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(1, activation = 'sigmoid')
])
adam = tf.keras.optimizers.Adam(learning_rate=0.0001)
model.compile(optimizer=adam, loss='binary_crossentropy', metrics=['acc'])

model.summary()

train_dir = "/content/drive/My Drive/Dataset/CAT_V_DOG/data/train"
test_dir = "/content/drive/My Drive/Dataset/CAT_V_DOG/data/test"

train_datagen = ImageDataGenerator(rescale = 1./255, rotation_range = 40,
                                   width_shift_range = 0.2,
                                   height_shift_range = 0.2,
                                   zoom_range = 0.2,
                                   shear_range = 0.2,
                                   horizontal_flip = True,
                                   fill_mode = 'nearest'
                                  )
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size = (256, 256),
                                                    batch_size = 128,
                                                    class_mode = 'binary')
test_datagen = ImageDataGenerator(rescale = 1./255,
                                  )
test_generator = test_datagen.flow_from_directory(test_dir,
                                                    target_size = (256, 256),
                                                    batch_size = 128,
                                                    class_mode = 'binary')


checkpoint_cb = tf.keras.callbacks.ModelCheckpoint("/content/drive/My Drive/Dataset/CAT_V_DOG/cat_V_dog_dropout_16-07
save_best_only = True)
early_stopping_cb = tf.keras.callbacks.EarlyStopping(patience = 10, restore_best_weights = True)
```

```
history = model.fit(train_generator,
                    epochs=20,
                    verbose=1,
                    callbacks = [checkpoint_cb, early_stopping_cb],
                    validation_data=test_generator, workers = 32)

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.legend(['train', 'validation'], loc = 'upper left')
plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['train', 'validation'], loc = 'upper left')
plt.show()

tf.keras.models.save_model(model,
filepath = '/content/drive/My Drive/Dataset/CAT_V_DOG/cat_V_dog_dropout_16-07_test_save_as_tf',
save_format = 'tf')
```

Appendix B

Model Summary

B.1 Cats Versus Dogs

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 256, 256, 32)      896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 256, 256, 32)      9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 128, 128, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 128, 128, 64)      18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 128, 128, 64)      36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 64, 64, 128)       73856
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 64, 64, 128)       147584
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 64, 64, 128)       147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 256)       295168
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 32, 256)       590080
```

```
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 32, 32, 256)       590080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 10, 10, 256)       0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 10, 10, 512)       1180160
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 10, 10, 512)       2359808
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 10, 10, 512)       2359808
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 3, 3, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 2048)              9439232
_________________________________________________________________
dropout (Dropout)            (None, 2048)              0
_________________________________________________________________
batch_normalization (BatchNo (None, 2048)              8192
_________________________________________________________________
dense_1 (Dense)              (None, 2048)              4196352
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0
_________________________________________________________________
batch_normalization_1 (Batch (None, 2048)              8192
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 2049
=================================================================
Total params: 21,463,713
Trainable params: 21,455,521
Non-trainable params: 8,192
_________________________________________________________________
```

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[2] Y. Sun, N. B. Agostini, S. Dong, and D. Kaeli, "Summarizing CPU and GPU Design Trends with Product Data," 2020.

[3] M. Rizwan, "Convolutional Neural Network – In a Nut Shell." [Online]. Available: https://engmrk.com/convolutional-neural-network-3/

[4] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and Scalability of GPU-Based Convolutional Neural Networks," 02 2010, pp. 317–324.

[5] A. Kayid, Y. Khaled, and M. Elmahdy, "Performance of CPUs/GPUs for Deep Learning workloads," 05 2018.

[6] Intel, "Intel® Core™ i5-8250U Processor." [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/124967/intel-core-i5-8250u-processor-6m-cache-up-to-3-40-ghz.html

[7] NVIDIA, "NVIDIA T4 70W LOW PROFILE PCIe GPU ACCELERATOR." [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-product-brief.pdf

[8] Nvidia, "NVIDIA GT 1030." [Online]. Available: https://www.nvidia.com/en-us/geforce/graphics-cards/gt-1030/specifications/

[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow,

A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[10] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.