**Name**:

**Student Number**:

**Graduate students must answer the bonus question.**

1. (15 points) We have a densely connected neural network with the following specifications:

   - 1200 nodes in the input layer (I)
   - 25 nodes in the output layer (O)
   - 2 hidden layers
   - 500 nodes in hidden layer 1 (H-1)
   - 750 nodes in hidden layer 2 (H-2)

   Answer the following questions:

   (a) (2 points) What is the size of the feature space? Explain why.

   (b) (2 points) What is the number of parameters in the 1st hidden layer H-1 assuming no bias terms? (why).

   (c) (2 points) What is the number of parameters in the 2nd hidden layer H-2 assuming no bias terms? (why).

   (d) (2 points) What is the number of parameters in the output layer O assuming no bias terms? (why).

   (e) (2 points) What is the overall size of the network in number of parameters? (why).

   (f) (5 points) Write the equation of the network assuming that each layer (H-1, H-2, and O) utilizes the tanh activation functions.

2. (10 points) We have a perceptron, with one linear layer with the input size of $3 \times 1$ and one bias vector and output size of $2 \times 1$. Suppose the input is $\mathbf{x}$, the weights matrix is $\mathbf{w}$, the bias term is $\mathbf{b}$, and the activation map is $\tanh(\cdot)$.

   (a) (2 points) Write down the equation of this perceptron in terms of one matrix multiplication.

   (b) (4 points) Draw the computational graph (with only one matrix multiplication node), and for each edge write the size of the matrix representing the computation.

   (c) (4 points) Write down the full backprop equation for the derivative of the activation output with respect to weights+biases and inputs.

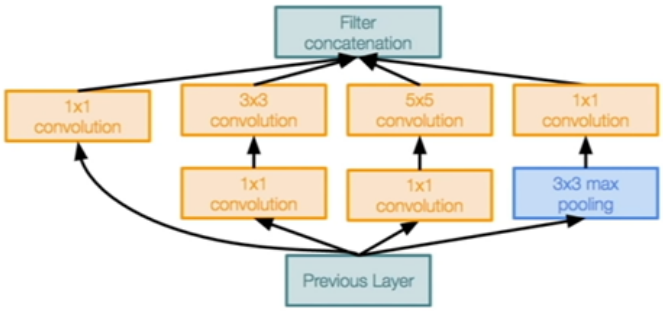
3. (15 points) We have designed a Convolutional Neural Network (CNN) to classify 10 classes, with the following specifications:

   - Input Layer $256 \times 256 \times 3$
   - Conv-1 Layer 128 filters of size $3 \times 3$, stride 1, padding 1.
   - Conv-2 Layer 64 filters of size $5 \times 5$, stride 2, padding 2.
   - Conv-3 Layer 128 filters of size $7 \times 7$, stride 2, no padding.
   - Pool-1 Layer max-pooling of size $2 \times 2$, stride 2, no padding.
   - Conv-4 Layer 64 filters of size $7 \times 7$, stride 1, no padding.
   - Conv-5 Layer 128 filters of size $7 \times 7$, stride 1, no padding.
   - FC-1 (fully connected) Layer of size $N \times 1$ same size as unwrapped previous layer.
   - FC-2 (fully connected) Layer of size $M \times 1$.

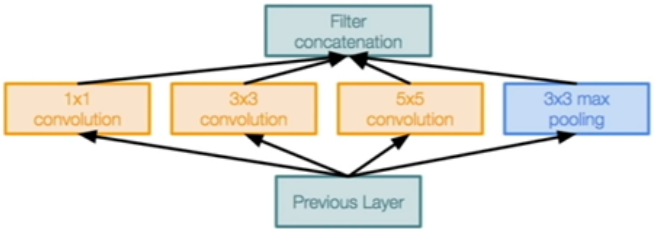
   Fill in the following table:

| Layer | Input Size | Kernel Size | Output Size | No. of Params |
|---|---|---|---|---|
| Conv-1 | × × | × × | × × | |
| Conv-2 | × × | × × | × × | |
| Conv-3 | × × | × × | × × | |
| Pool-1 | × × | × × | × × | |
| Conv-4 | × × | × × | × × | |
| Conv-5 | × × | × × | × × | |
| FC-1 | | NA | | |
| FC-2 | | NA | | |

4. (25 points) Consider the architecture of GoogLeNet and answer the following questions:
   (a) (5 points) Describe the Inception module of the GoogLeNet shown in Figure 1(a).
   (b) (10 points) Assume the input from the previous layer of the inception module in Figure 1(a) is $128 \times 128 \times 512$, the depth of each of the bottom $1 \times 1$ conv-modules is 64, the depth of the top $1 \times 1$ module is 512, the depth of the $3 \times 3$ convolution is 128, and the depth of $5 \times 5$ convolution is 64.
      1. Calculate the padding used for the $3 \times 3$ and $5 \times 5$ convolutions to keep outputs the same as the inputs.
      2. Calculate the output size of each module and show on graph.
      3. Calculate the number of operations in each module.
      4. Calculate and number of parameters in each module.
      5. Calculate the total number of operations of the whole module.
      6. Calculate the total number of parameters of the whole module.
   (c) (5 points) Calculate the number of parameters and number of operations in the naive inception module shown in Figure 1(b). Discuss why the GoogLeNet module is more efficient than the naive module.
   (d) (5 points) Discuss how the Inception module differs in terms of the number of parameters from the naive Inception module.




(a)




(b)

5. (30 points) **The VGG-Lite**: For this problem you will implement a version of VGGnet, we call VGG-Lite. You can start from the code in the programming assignment Intro-PA-01, and implement the VGG-lite.

   1. **Imports:** In GoogleColab, import the needed libraries.

```
[74] import tensorflow.keras
     import tensorflow
     from tensorflow.keras.preprocessing.image import ImageDataGenerator
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Conv2D, MaxPooling2D
     from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense
     from tensorflow.keras import backend as K
     from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
     from tensorflow.keras.applications.resnet50 import ResNet50
     import numpy as np
     from tensorflow.keras.preprocessing import image
     import matplotlib.pyplot as plt
     %matplotlib inline
     from tensorflow.keras.utils import plot_model
```

   2. **Mount Drive and Data Preparation:** You will need to mount your drive and unzip the dataset file. If you have done this part from Intro-PA-01, you should have the cats-and-dogs dataset google drive.

   3. **Hyper-parameter Setup:** You will need to set up the hyper parameters of your network. These include No. of epochs, batch_sizes, training_samples, validation_samples, etc. See samples below:

```
[76] epochs = 5
     batch_size = 8
     training_samples = 200
     validation_samples = 400
     img_width = 200
     img_height = 200
     channels = 3
     input_shape = (img_width,img_height,3)
```

   4. **Data Preparation:** You will then need to prepare your training and validation data. See below example:

```
train_data_dir = 'Small_set_cats_vs_dogs/train'
validation_data_dir = 'Small_set_cats_vs_dogs/val'
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.4,
    zoom_range=0.4,
    rotation_range=20,
    width_shift_range=0.4,
    height_shift_range=0.4,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode="nearest")

val_datagen = ImageDataGenerator(rescale=1. / 255)
print("Training Images ... ")
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary')

print("Validation Images ... ")
validation_generator = val_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary')
```

5. **Network Setup:** This step is the most important step of your code. For this task, you will need to study the VGGNet (see slides and the seminal paper by Simonyan & Zimmerman [1]). To complete this step you will need to perform the following tasks:
   - Study the overall architecture of VGGNet from the course slides and the paper by Simonyan & Zimmerman. Once you understand the idea, the architecture, and the model, design a network based on VGGNet.
   - Draw a diagram of your network.
   - Calculate the computational cost, in terms of each layer sizes, number of parameters, and potential memory requirements. Include in results section of your report.
   - Implement your network in GoogleColab.
   - Compile your model (network).
   - Use Keras commands to print out your model summary and optional callbacks, etc. (Hint: you can do the same as you did for Intro-PA-01)
6. **Training and Validation:** Train and validate your model. Make sure to save some checkpoints for your model. Include these information, e.g. a few snapshots of your model while it is training (loss, acc., etc.) in your report.
7. **Visualization:** Plot the graphs for your training and validation for both loss and accuracy. Include in results section of your report.
8. **Evaluation and Testing:** Test and evaluate your model on the test data. Include in results section of your report.

**Deliverables**

(a) (2 points) **Mounting and Data Preparation:** Perform Items 1 and 2.

(b) (2 points) **Hyper-parameter Setup and Data Preparation:** Perform Items 3 and 4.

(c) (12 points) **Model Setup:** Establish a suitable model (network) based on VGG.

(d) (2 points) **Model Training:** Determine a loss function, optimizer, etc. and train your network. Include the training process of your network in the report.

(e) (2 points) **Model Evaluation:** Show the loss and accuracy values of the network on this test set. **Your network should have a test accuracy of more than 90%.**

(f) (5 points) **Code**: Submit your code as a python 3 file.

(g) (10 points) **Report:** Write a report on the results of your network. Conduct some experimentation on various parameters and show how the results change. Include the following in your report:

**Introduction Section:** In introduction section give an overview the task at hand, dataset, and your approach.

**Network:** In this section you will show the diagrams of the network that you designed. Make sure to include the different iteration of your network, how you setup and worked with a suitable set of hyper-parameters, the final choice of your hyper-parameters, and any relevant information about your design, the architecture, and model.

**Results:** In this section you will present the results of your network. Include your network summary, final diagram, test and validation graphs for loss and accuracy, and evaluation results (accuracy, and other relevant measures).

(h) (10 points (bonus)) **Grad Only [EC for Undergrad]** Implement and AlexNet-Lite architecture and compare your results with the VGGNet-Lite in terms of accuracy, loss, speed, and memory.

# References

[1] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.