FIT3181_Assignment1-2022

August 6, 2022

1 FIT3181: Deep Learning (2022)

CE/Lecturer: Dr **Trung Le** | trunglm@monash.edu *Head Tutor:* Mr **Thanh Nguyen** | thanh.nguyen4@monash.edu Department of Data Science and AI, Faculty of Information Technology, Monash University, Australia ***

2 Student Information

Surname: [**Enter your surname here**] Firstname: [**Enter your firstname here**] Student ID: [**Enter your ID here**] Email: [**Enter your email here**] Your tutorial time: [**Enter your tutorial time here**] ***

3 Deep Neural Networks

3.0.1 Due: 11:59pm Sunday, 18 September 2022 (Sunday)

Important note: This is an individual assignment. It contributes 20% to your final mark. Read the assignment instruction carefully.

3.1 Instruction

This notebook has been prepared for your to complete Assignment 1. The theme of this assignment is about practical machine learning knowledge and skills in deep neural networks, including feedforward and convolutional neural networks. Some sections have been partially completed to help you get started. **The total marks for this notebook is 100**.

- Before you start, read the entire notebook carefully once to understand what you need to do.
- For each cell marked with **#YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL**, there will be places where you **must** supply your own codes when instructed.

This assignment contains **three** parts:

- Part 1: Questions on theory and knowledge on machine learning and deep learning **[30 points], 30%**
- Part 2: Coding assessment on TensorFlow for Deep Neural Networks (DNN) **[30 points], 30%**

- Part 3: Coding assessment on TensorFlow for Convolution Neural Networks (CNN) **[40 points], 40%**

Hint: This assignment was essentially designed based on the lectures and tutorials sessions covered from Week 1 to Week 6. You are strongly recommended to go through these contents thoroughly which might help you to complete this assignment.

3.2 What to submit

This assignment is to be completed individually and submitted to Moodle unit site. **By the due date, you are required to submit one single zip file, named xxx_assignment01_solution.zip where xxx is your student ID, to the corresponding Assignment (Dropbox) in Moodle**.

For example, if your student ID is 12356, then gather all of your assignment solution to folder, create a zip file named 123456_assignment01_solution.zip and submit this file.

Within this zip folder, you **must** submit the following files: 1. **Assignment01_solution.ipynb**: this is your Python notebook solution source file. 1. **Assignment01_output.html**: this is the output of your Python notebook solution *exported* in html format. 1. Any **extra files or folder** needed to complete your assignment (e.g., images used in your answers).

Since the notebook is quite big to load and work together, one recommended option is to split solution into three parts and work on them seperately. In that case, replace **Assignment01_solution.ipynb** by three notebooks: **Assignment01_Part1_solution.ipynb**, **Assignment01_Part2_solution.ipynb** and **Assignment01_Part3_solution.ipynb**

You can run your codes on Google Colab. In this case, you need to capture the screenshots of your Google Colab model training and put in corresponding places in your Jupyter notebook. You also need to store your trained models to folder *./models* with recognizable file names (e.g., Part3_Sec3_2_model.h5).

3.3 Part 1: Theory and Knowledge Questions

[Total marks for this part: 30 points]

The first part of this assignment is for you to demonstrate your knowledge in deep learning that you have acquired from the lectures and tutorials materials. Most of the contents in this assignment are drawn from **the lectures and tutorials from weeks 1 to 3**. Going through these materials before attempting this part is highly recommended.

Question 1.1 Activation function plays an important role in modern Deep NNs. For each of the activation function below, state its output range, find its derivative (show your steps), and plot the activation fuction and its derivative **(a)** Leaky ReLU: $\text{LeakyReLU}\left(x\right)=\begin{cases} 0.01x & \text{if}\,x<0\\ x & \text{otherwise} \end{cases}$

[1.5 points]

(b) Softplus: $\text{Softplus}\left(x\right)=\ln\left(1+e^{x}\right)$

[1.5 points]

Numpy is possibly being used in the following questions. You need to import numpy here.

[ ]:
```
import numpy as np
```

Question 1.2 Assume that we feed a data point x with a ground-truth label $y = 2$ to the feed-forward neural network with the ReLU activation function as shown in the following figure **(a)** What is the numerical value of the latent presentation $h^1(x)$?

[1 point]

(b) What is the numerical value of the latent presentation $h^2(x)$?

[1 point]

(c) What is the numerical value of the logit $h^3(x)$?

[1 point]

(d) What is the corresonding prediction probabilities $p(x)$?

[1 point]

(e) What is the cross-entropy loss caused by the feed-forward neural network at (x, y)? Remind that $y = 2$.

[1 point]

(e) Assume that we are applying the label smoothing technique (i.e., link for main paper from Goeff Hinton) with $\alpha = 0.1$. What is the relevant loss caused by the feed-forward neural network at (x, y)?

[1 point]

You need to show both formulas and numerical results for earning full mark. Although it is optional, it is great if you show your numpy code for your computation.

Question 1.3 Assume that we are constructing a multilayered feed-forward neural network for a classification problem with three classes where the model parameters will be generated randomly using your student ID. The architecture of this network is $(3(Input) \rightarrow 4(LeakyReLU) \rightarrow 3(Output))$ as shown in the following figure. Note that the LeakyReLU has the same formula as the one in Q1.1. We feed a feature vector $x = \begin{bmatrix} 1 & -1 & 1.5 \end{bmatrix}^T$ with ground-truth label $y = 3$ to the above network.

You need to show both formulas, numerical results, and your numpy code for your computation for earning full marks.

[ ]:
```
#Code to generate random matrices and biases for W1, b1, W2, b2
import numpy as np
student_id = 1234          #insert your student id here for example 1234
np.random.seed(student_id)
W1 = np.random.rand(4,3)
```

```
b1 = np.random.rand(4,1)
W2 = np.random.rand(3,4)
b2 = np.random.rand(3,1)
```

Forward propagation

(a) What is the value of $\bar{h}^1(x)$?

[1 point]

Show your fomular

[ ]:
```
# Show your code
```

(b) What is the value of $h^1(x)$?

[1 point]

Show your fomular

[ ]:
```
#Show your code
```

(c) What is the predicted value $\hat{y}$?

[1 point]

Show your fomular

[ ]:
```
#Show your code
```

(d) Suppose that we use the cross-entropy (CE) loss. What is the value of the CE loss l?

[1 point]

Show your fomular

[ ]:
```
#Show your code
```

Backward propagation

(e) What are the derivatives $\frac{\partial l}{\partial h^2}$, $\frac{\partial l}{\partial W^2}$, and $\frac{\partial l}{\partial b^2}$?

[6 points]

Show your fomular

[ ]:
```
#Show your code
```

(f) What are the derivatives $\frac{\partial l}{\partial h^1}$, $\frac{\partial l}{\partial \bar{h}^1}$, $\frac{\partial l}{\partial W^1}$, and $\frac{\partial l}{\partial b^1}$?

[6 points]

Show your fomular

[ ]:
```
#Show your code
```

SGD update

(g) Assume that we use SGD with learning rate $\eta = 0.01$ to update the model parameters. What are the values of W^2, b^2 and W^1, b^1 after updating?

[5 points]

Show your fomular

```
[ ]: #Show your code
```

3.4 Part 2: Deep Neural Networks (DNN)

[Total marks for this part: 30 points]

The first part of this assignment is for you to demonstrate your basis knowledge in deep learning that you have acquired from the lectures and tutorials materials. Most of the contents in this assignment are drawn from **the tutorials covered from weeks 1 to 4**. Going through these materials before attempting this assignment is highly recommended.

In the first part of this assignment, you are going to work with the **FashionMNIST** dataset for *image recognition task*. It has the exact same format as MNIST (70,000 grayscale images of 28×28 pixels each with 10 classes), but the images represent fashion items rather than handwritten digits, so each class is more diverse, and the problem is significantly more challenging than MNIST.

Question 2.1. Load the Fashion MNIST using Keras datasets [5 points]

We first use keras incoporated in TensorFlow 2.x for loading the training and testing sets.

```
[ ]: import tensorflow as tf
     from tensorflow import keras
```

```
[ ]: tf.random.set_seed(1234)
```

We first use keras datasets in TF 2.x to load Fashion MNIST dataset.

```
[ ]: fashion_mnist = keras.datasets.fashion_mnist
     (X_train_full_img, y_train_full), (X_test_img, y_test) = #Insert your code here
```

The shape of X_train_full_img is $(60000, 28, 28)$ and that of X_test_img is $(10000, 28, 28)$. We next convert them to matrices of vectors and store in X_train_full and X_test.

```
[ ]: num_train = X_train_full_img.shape[0]
     num_test = X_test_img.shape[0]
     X_train_full =  #Insert your code here
     X_test =     #Insert your code here
     print(X_train_full.shape, y_train_full.shape)
     print(X_test.shape, y_test.shape)
```

Question 2.2. Preprocess the dataset and split into training, validation, and testing datasets [5 points]

You need to write the code to address the following requirements: - Print out the dimensions of X_train_full and X_test - Use 10% of X_train_full for validation and the rest of X_train_full for training. This splits X_train_full and y_train_full into X_train, y_train (90%) and X_valid, y_valid (10%). - Finally, scale the pixels of X_train, X_valid, and X_test to $[0, 1]$) (i.e., $X = X/255.0$).

You have now the separate training, validation, and testing sets for training your model.

```
[ ]: import math
N = X_train_full.shape[0]
i = math.floor(0.9*N)
X_train, y_train = #Insert your code here
X_valid, y_valid = #Insert your code here
X_train, X_valid, X_test = #Insert your code here
```

Question 2.3. Visualize some images in the training set with labels [5 points]

You are required to write the code to show **random** 36 images in X_train_full_img (which is an array of images) with labels as in the following figure. Note that the class names of Fashion MNIST are as follows - "1:T-shirt/top", "2:Trouser", "3:Pullover", "4:Dress", "5:Coat", "6:Sandal", "7:Shirt", "8:Sneaker", "9:Bag", "10:Ankle boot"

```
[ ]: import matplotlib.pyplot as plt
%matplotlib inline
```

```
[ ]: # YOU ARE REQUIRED TO INSERT YOUR CODES IN THIS CELL
```

Question 2.4. Write code for the feed-forward neural net using TF 2.x [5 points]

We now develop a feed-forward neural network with the architecture $784 \to 20(ReLU) \to 40(ReLU) \to 10(softmax)$. You can choose your own way to implement your network and an optimizer of interest. You should train model in 20 epochs and evaluate the trained model on the test set.

```
[ ]: #Insert your code here and you can add more cells if necessary
```

Question 2.5. Tuning hyper-parameters with grid search [5 points]

Assume that you need to tune the number of neurons on the first and second hidden layers $n_1 \in \{20, 40\}$, $n_2 \in \{20, 40\}$ and the used activation function $act \in \{sigmoid, tanh, relu\}$. The network has the architecture pattern $784 \to n_1(act) \to n_2(act) \to 10(softmax)$ where n_1, n_2, and act are in their grides. Write the code to tune the hyper-parameters n_1, n_2, and act. Note that you can freely choose the optimizer and learning rate of interest for this task.

```
[ ]: #Insert your code here. You can add more cells if necessary
```

Question 2.6. Experimenting with the label smoothing technique [5 points]

Implement the label smoothing technique (i.e., link for main paper from Goeff Hinton) by yourself. Note that you cannot use the built-in label-smoothing loss function in TF2.x. Try the label smoothing technique with $\alpha = 0.1, 0.15, 0.2$ and report the performances. You need to examine the label smoothing technique with the best architecture obtained in **Question 2.5**.

[ ]:
```
#Insert your code here. You can add more cells if necessary
```

3.5 Part 3: Convolutional Neural Networks and Image Classification

**

[Total marks for this part: 40 points]

**

This part of the asssignment is designed to assess your knowledge and coding skill with Tensorflow as well as hands-on experience with training Convolutional Neural Network (CNN).

The dataset we use for this part is a small animal dataset consisting of $5,000$ images of cats, dogs, fishes, lions, chickens, elephants, butterflies, cows, spiders, and horses, each of which has 500 images. You can download the dataset at download here and then decompress to the folder `datasets\Animals` in your assignment folder.

Your task is to build a CNN model using *TF 2.x* to classify these animals. You're provided with the module models.py, which you can find in the assignment folder, with some of the following classes:

1. `AnimalsDatasetManager`: Support with loading and spliting the dataset into the train-val-test sets. It also supports generating next batches for training. `AnimalsDatasetManager` will be passed to CNN model for training and testing.
2. `DefaultModel`: A base class for the CNN model.
3. `YourModel`: The class you'll need to implement for building your CNN model. It inherits some useful attributes and functions from the base class `DefaultModel`

Firstly, we need to run the following cells to load and preprocess the Animal dataset.

[1]:
```
%load_ext autoreload
%autoreload 2
```

Install the package `imutils` if you have not installed yet

[2]:
```
! pip install imutils
```

```
Requirement already satisfied: imutils in
c:\users\trung\anaconda3\envs\tf2x_cpu\lib\site-packages (0.5.4)
```

[3]:
```
import os
import matplotlib.pyplot as plt
plt.style.use('ggplot')
%matplotlib inline
import models
from models import SimplePreprocessor, AnimalsDatasetManager, DefaultModel
```

[4]:
```
def create_label_folder_dict(adir):
    sub_folders= [folder for folder in os.listdir(adir)
                  if os.path.isdir(os.path.join(adir, folder))]
```

```
    label_folder_dict= dict()
    for folder in sub_folders:
        item= {folder: os.path.abspath(os.path.join(adir, folder))}
        label_folder_dict.update(item)
    return label_folder_dict
```

[5]:
```
label_folder_dict= create_label_folder_dict("./datasets/Animals")
```

The below code helps to create a data manager that contains all relevant methods used to manage and process the experimental data.

[6]:
```
sp = SimplePreprocessor(width=32, height=32)
data_manager = AnimalsDatasetManager([sp])
data_manager.load(label_folder_dict, verbose=100)
data_manager.process_data_label()
data_manager.train_valid_test_split()
```

```
butterfiles 500
Processed 100/500
Processed 200/500
Processed 300/500
Processed 400/500
Processed 500/500
cats 501
Processed 100/500
Processed 200/500
Processed 300/500
Processed 400/500
Processed 500/500
chickens 500
Processed 100/500
Processed 200/500
Processed 300/500
Processed 400/500
Processed 500/500
cows 500
Processed 100/500
Processed 200/500
Processed 300/500
Processed 400/500
Processed 500/500
dogs 501
Processed 100/500
Processed 200/500
Processed 300/500
Processed 400/500
Processed 500/500
elephants 500
```

```
Processed 100/500
Processed 200/500
Processed 300/500
Processed 400/500
Processed 500/500
fishes 500
Processed 100/500
Processed 200/500
Processed 300/500
Processed 400/500
Processed 500/500
horses 500
Processed 100/500
Processed 200/500
Processed 300/500
Processed 400/500
Processed 500/500
lions 500
Processed 100/500
Processed 200/500
Processed 300/500
Processed 400/500
Processed 500/500
spiders 500
Processed 100/500
Processed 200/500
Processed 300/500
Processed 400/500
Processed 500/500
```

Note that the object `data_manager` has the attributes relating to *the training, validation, and testing sets* as shown belows. You can use them in training your developped models in the sequel.

[7]:
```
print(data_manager.X_train.shape, data_manager.y_train.shape)
print(data_manager.X_valid.shape, data_manager.y_valid.shape)
print(data_manager.X_test.shape, data_manager.y_test.shape)
print(data_manager.classes)
```

```
(4000, 32, 32, 3) (4000,)
(500, 32, 32, 3) (500,)
(500, 32, 32, 3) (500,)
['butterfiles' 'cats' 'chickens' 'cows' 'dogs' 'elephants' 'fishes'
 'horses' 'lions' 'spiders']
```

We now run the **default model** built in the **models.py** file which serves as a basic baseline to start the investigation. Follow the following steps to realize how to run a model and know the built-in methods associated to a model developped in the DefaultModel class.

We first initialize a default model from the DefaultModel class. Basically, we can define the relevant parameters of training a model including `num_classes`, `optimizer`, `learning_rate`, `batch_size`, and num_epochs.

[8]:
```
network1 = DefaultModel(name='network1',
                        num_classes=len(data_manager.classes),
                        optimizer='sgd',
                        batch_size= 128,
                        num_epochs = 20,
                        learning_rate=0.5)
```

The method build_cnn() assists us in building your convolutional neural network. You can view the code (in the **models.py** file) of the model behind a default model to realize how simple it is. Additionally, the method summary() shows the architecture of a model.

[9]:
```
network1.build_cnn()
network1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
average_pooling2d (AveragePo (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
average_pooling2d_1 (Average (None, 8, 8, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dense (Dense)                (None, 10)                40970
=================================================================
Total params: 106,538
Trainable params: 106,538
Non-trainable params: 0
_________________________________________________________________
None
```

To train a model regarding to the datasets stored in data_manager, you can invoke the method fit() for which you can specify the batch size and number of epochs for your training.

[10]:
```
network1.fit(data_manager, batch_size = 64, num_epochs = 20)
```

```
Train on 4000 samples, validate on 500 samples
Epoch 1/20
4000/4000 [==============================] - 6s 1ms/sample - loss: 2.3015 -
```

```
accuracy: 0.1150 - val_loss: 2.3054 - val_accuracy: 0.1120
Epoch 2/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3031 -
accuracy: 0.1067 - val_loss: 2.3009 - val_accuracy: 0.1200
Epoch 3/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.2984 -
accuracy: 0.1075 - val_loss: 2.3056 - val_accuracy: 0.1040
Epoch 4/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3049 -
accuracy: 0.0997 - val_loss: 2.3053 - val_accuracy: 0.0840
Epoch 5/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3052 -
accuracy: 0.0925 - val_loss: 2.3045 - val_accuracy: 0.1040
Epoch 6/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3055 -
accuracy: 0.0943 - val_loss: 2.3046 - val_accuracy: 0.0980
Epoch 7/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3043 -
accuracy: 0.0985 - val_loss: 2.3035 - val_accuracy: 0.0980
Epoch 8/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3049 -
accuracy: 0.0915 - val_loss: 2.3084 - val_accuracy: 0.0840
Epoch 9/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3054 -
accuracy: 0.0900 - val_loss: 2.3065 - val_accuracy: 0.0860
Epoch 10/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3049 -
accuracy: 0.0945 - val_loss: 2.3071 - val_accuracy: 0.0860
Epoch 11/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3053 -
accuracy: 0.0988 - val_loss: 2.3046 - val_accuracy: 0.0980
Epoch 12/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3047 -
accuracy: 0.1000 - val_loss: 2.3058 - val_accuracy: 0.1000
Epoch 13/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3025 -
accuracy: 0.1047 - val_loss: 2.3054 - val_accuracy: 0.0840
Epoch 14/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3052 -
accuracy: 0.0882 - val_loss: 2.3052 - val_accuracy: 0.0960
Epoch 15/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3052 -
accuracy: 0.0975 - val_loss: 2.3066 - val_accuracy: 0.0840
Epoch 16/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3041 -
accuracy: 0.1015 - val_loss: 2.3019 - val_accuracy: 0.1040
Epoch 17/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3052 -
```

```
accuracy: 0.0930 - val_loss: 2.3049 - val_accuracy: 0.1040
Epoch 18/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3053 -
accuracy: 0.0962 - val_loss: 2.3070 - val_accuracy: 0.1040
Epoch 19/20
4000/4000 [==============================] - 6s 1ms/sample - loss: 2.3050 -
accuracy: 0.0957 - val_loss: 2.3059 - val_accuracy: 0.0860
Epoch 20/20
4000/4000 [==============================] - 5s 1ms/sample - loss: 2.3051 -
accuracy: 0.0955 - val_loss: 2.3057 - val_accuracy: 0.0960
```

Here you can compute the accuracy of your trained model with respect to a separate testing set.

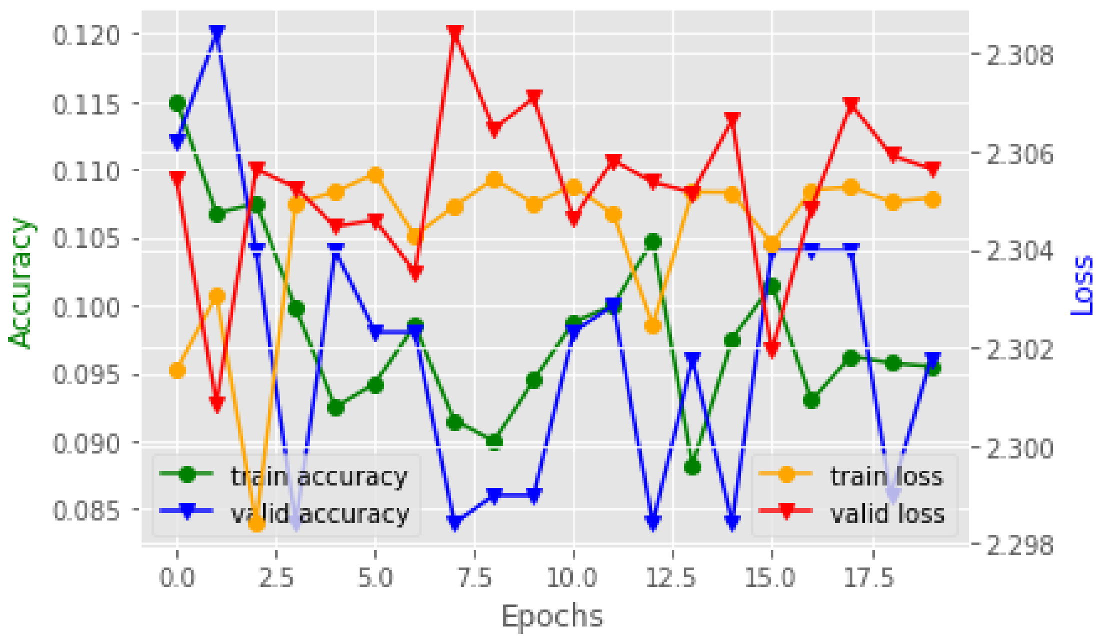
[11]:
```
network1.compute_accuracy(data_manager.X_test, data_manager.y_test)
```

```
500/500 [==============================] - 0s 250us/sample - loss: 2.3036 -
accuracy: 0.0980
```

[11]: 0.098

Below shows how you can inspect the training progress.

[12]:
```
network1.plot_progress()
```



You can use the method `predict()` to predict labels for data examples in a test set.

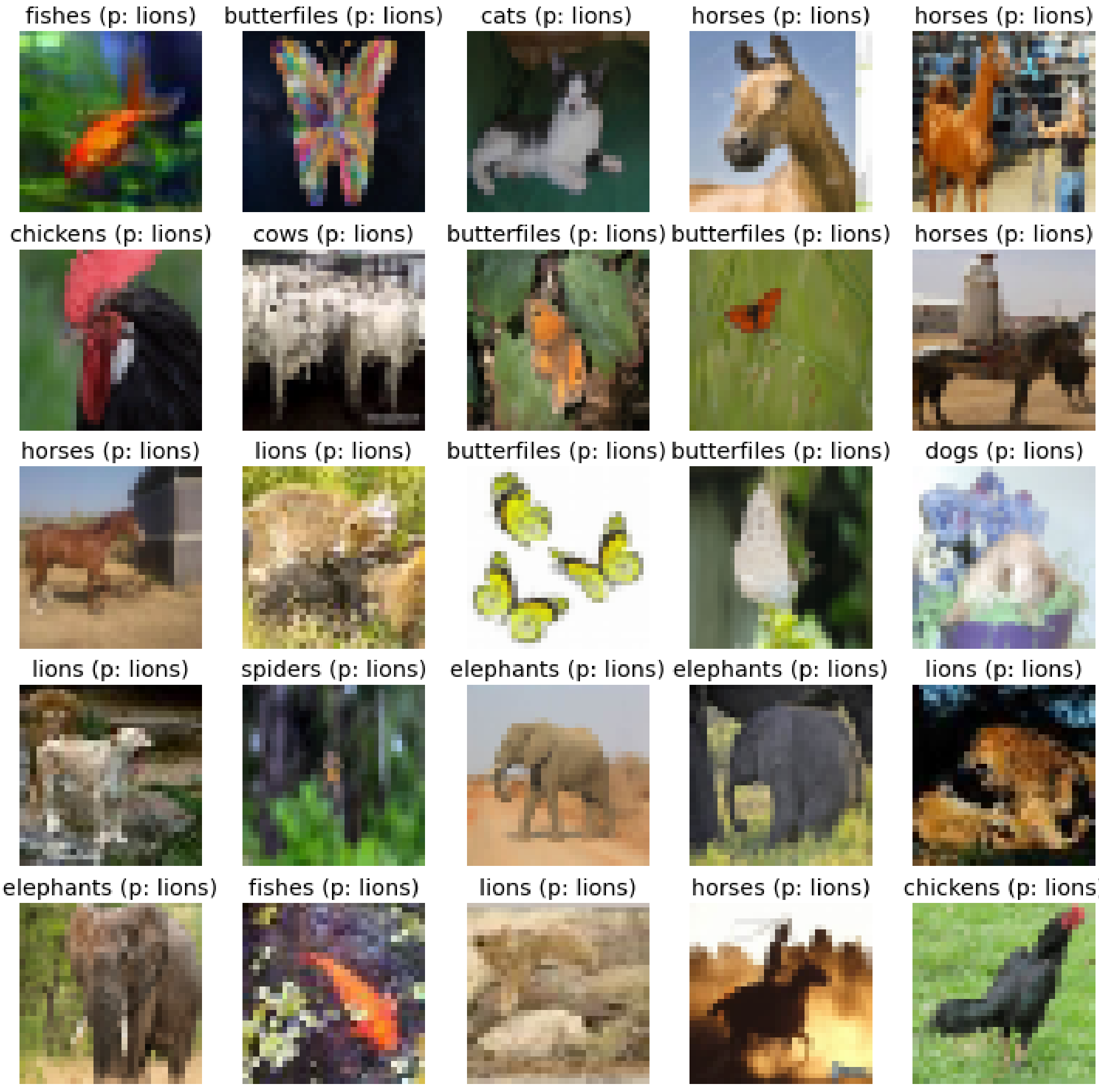
[13]:
```
network1.predict(data_manager.X_test[0:10])
```

[13]: array([8, 8, 8, 8, 8, 8, 8, 8, 8, 8], dtype=int64)

Finally, the method `plot_prediction()` visualizes the predictions for a test set in which several images are chosen to show the predictions.

[14]:
```
network1.plot_prediction(data_manager.X_test, data_manager.y_test, data_manager.
↪classes)
```

<Figure size 432x288 with 0 Axes>



Question 3.1 After running the above cells to train the default model and observe the learning curve. Report your observation (i.e. did the model learn well? if not, what is the problem? What would you do to improve it?). Write your answer below.

[4 points]

#Your answer and observation here

.....

For questions 3.2 to 3.9, you'll need to write your own model in a way that makes it easy for you to experiment with different architectures and parameters. The goal is to be able to pass the parameters to initialize a new instance of `YourModel` to build different network architectures with different parameters. Below are descriptions of some parameters for `YourModel`, which you can find in function `__init__()` for the class `DefaultModel`:

1. `num_blocks`: an integer specifying the number of blocks in our network. Each block has the pattern `[conv, batch norm, activation, conv, batch norm, activation, mean pool, dropout]`. All convolutional layers have filter size $(3, 3)$, strides $(1, 1)$ and 'SAME' padding, and all mean pool layers have strides $(2, 2)$ and 'SAME' padding. The network will consists of a few blocks before applying a linear layer to output the logits for the softmax layer.

2. `feature_maps`: the number of feature maps in the first block of the network. The number of feature_maps will double in each of the following block. To make it convenient for you, we already calculated the number of feature maps for each block for you in line 106

3. `drop_rate`: the keep probability for dropout. Setting `drop_rate` to 0.0 means not using dropout.

4. `batch_norm`: the batch normalization function is used or not. Setting `batch_norm` to `None` means not using batch normalization.

5. The `skip connection` is added to the output of the second `batch norm`. Additionally, your class has a boolean property (i.e., instance variable) named `use_skip`. If `use_skip=True`, the skip connectnion is enable. Otherwise, if `use_skip=False`, the skip connectnion is disable.

Below is the architecture of one block:

Below is the architecture of the entire deep net with `two blocks`:

Here we assume that the first block has `feature_maps = feature_maps[0] = 32`. Note that the initial number of feature maps of the first block is declared in the instance variable `feature_maps` and is multiplied by 2 in each follpwing block.

[ ]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
```

[ ]:
```
tf.random.set_seed(1234)
```

Question 3.2 Write the code of the `YourModel` class here. Note that this class will inherit from the `DefaultModel` class. You'll only need to re-write the code for the `build_cnn` method in the `YourModel` class from the cell below. Note that the `YourModel` class is inherited from the `DefaultModel` class.

[4 points]

[ ]:
```
class YourModel(DefaultModel):
    def __init__(self,
                 name='network1',
                 width=32, height=32, depth=3,
                 num_blocks=2,
                 feature_maps=32,
                 num_classes=4,
                 drop_rate=0.2,
                 batch_norm = None,
                 is_augmentation = False,
                 activation_func='relu',
                 use_skip = True,
                 optimizer='adam',
                 batch_size=10,
                 num_epochs= 20,
                 learning_rate=0.0001,
                 verbose= True):
        super(YourModel, self).__init__(name, width, height, depth, num_blocks,␣
 ↪feature_maps, num_classes, drop_rate, batch_norm, is_augmentation,
                                        activation_func, use_skip, optimizer,␣
 ↪batch_size, num_epochs, learning_rate, verbose)

    def build_cnn(self):
        #Insert your code here
```

Question 3.3 Once writing your own model, you need to compare two cases: (i) *using the skip connection* and (ii) *not using the skip connection*. You should set the instance variable `use_skip` to either `True` or `False`. For your runs, report which case is better and if you confront overfitting in training.

[6 points]

#Write your report and observation here

.....

[ ]:
```
our_network_skip = YourModel(name='network1',
                      feature_maps=32,
                      num_classes=len(data_manager.classes),
                      num_blocks=3,
                      drop_rate= 0.0,
                      batch_norm=True,
                      use_skip = True,
                      optimizer='adam',
                      learning_rate= 0.001)
drop_out_network.build_cnn()
drop_out_network.summary()
```

[ ]:
```
our_network_skip.fit(data_manager, batch_size=32, num_epochs=20)
```

[ ]:
```python
our_network_no_skip = YourModel(name='network1',
                               feature_maps=32,
                               num_classes=len(data_manager.classes),
                               num_blocks=3,
                               drop_rate= 0.0,
                               batch_norm=True,
                               use_skip = False,
                               optimizer='adam',
                               learning_rate= 0.001)
drop_out_network.build_cnn()
drop_out_network.summary()
```

[ ]:
```python
our_network_no_skip.fit(data_manager, batch_size=32, num_epochs=20)
```

Question 3.4 Now, let us tune the $num\_blocks \in \{2, 3, 4\}$, $use\_skip \in \{True, False\}$, and $learning\_rate \in \{0.001, 0.0001\}$. Write your code for this tuning and report the result of the best model on the testing set. Note that you need to show your code for tuning and evaluating on the test set to earn the full marks. During tuning, you can set the instance variable `verbose` of your model to `False` for not showing the training details of each epoch.

[4 points]

#Report the best parameters and the testing accuracy here

.....

[ ]:
```python
#Insert your code here. You can add more cells if necessary
```

Question 3.5 We now try to apply data augmentation to improve the performance. Extend the code of the class `YourModel` so that if the attribute `is_augmentation` is set to `True`, we apply the data augmentation. Also you need to incorporate early stopping to your training process. Specifically, you early stop the training if the valid accuracy cannot increase in three consecutive epochs.

[4 points]

[ ]:
```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Wtire your code in the cell below. Hint that you can rewrite the code of the `fit` method to apply the data augmentation. In addition, you can copy the code of `build_cnn` method above to reuse here.

[ ]:
```python
class YourModel(DefaultModel):
    def __init__(self,
                 name='network1',
                 width=32, height=32, depth=3,
                 num_blocks=2,
                 feature_maps=32,
                 num_classes=4,
                 drop_rate=0.2,
```

```
                 batch_norm = None,
                 is_augmentation = False,
                 activation_func='relu',
                 use_skip = True,
                 optimizer='adam',
                 batch_size=10,
                 num_epochs= 20,
                 learning_rate=0.0001):
        super(YourModel, self).__init__(name, width, height, depth, num_blocks,␣
↪feature_maps, num_classes, drop_rate, batch_norm, is_augmentation,
                                        activation_func, use_skip, optimizer,␣
↪batch_size, num_epochs, learning_rate)

    def build_cnn(self):
        #reuse code of previous section here

    def fit(self, data_manager, batch_size=None, num_epochs=None):
        #insert your code here
```

Question 3.6 Leverage your best model with the data augmentation and try to observe the difference in performance between using data augmentation and non-using it.

[4 points]

#Write your answer and observation here

…..

[ ]:
```
#Insert your code here. You can add more cells if necessary
```

Question 3.7 Exploring Data Mixup Technique for Improving Generalization Ability.

[4 points]

Data mixup is another super-simple technique used to boost the generalization ability of deep learning models. You need to incoroporate data mixup technique to the above deep learning model and experiment its performance. There are some papers and documents for data mixup as follows: - Main paper for data mixup link for main paper and a good article article link.

You need to extend your model developed above, train a model using data mixup, and write your observations and comments about the result.

#Write your answer and observation here

…..

[ ]:
```
#Insert your code here. You can add more cells if necessary
```

Question 3.8 Attack your best obtained model with PGD, MIM, and FGSM attacks with $\epsilon = 0.0313, k = 20, \eta = 0.002$ on the testing set. Write the code for the attacks and report the robust accuracies. Also choose a random set of 20 clean images in the testing set and visualize the original and attacked images.

[5 points]

```
[ ]: #Insert your code here. You can add more cells if necessary
```

Question 3.9 Train a robust model using adversarial training with PGD $\epsilon = 0.0313, k = 10, \eta = 0.002$. Write the code for the adversarial training and report the robust accuracies. After finishing the training, you need to store your best robust model in the folder `./models` and load the model to evaluate the robust accuracies for PGD, MIM, and FGSM attacks with $\epsilon = 0.0313, k = 20, \eta = 0.002$ on the testing set.

[5 points]

```
[ ]: #Insert your code here. You can add more cells if necessary
```

The following is an exploring question with bonus points. It is great if you try to do this question, but it is **totally optional**. In this question, we will investigate a recent SOTA technique to improve the generalization ability of deep nets named *Sharpness-Aware Minimization (SAM)* (link to the main paper). Furthermore, SAM is simple and efficient technique, but roughly doubles the training time due to its required computation. If you have an idea to improve SAM, it would be a great paper to top-tier venues in machine learning and computer vision. Highly recommend to give it a try.

Question 3.10 (**additionally exploring question**) Read the SAM paper (link to the main paper). Try to apply this techique to the best obtained model and report the results. For the purpose of implementating SAM, we can flexibly add more cells and extensions to the `model.py` file.

[5 points]

```
[ ]: #Insert your code here. You can add more cells if necessary
```

**

END OF ASSIGNMENT

GOOD LUCK WITH YOUR ASSIGNMENT 1!

**