NAME: KANG CHIN SHEN (G1) MATRIC. NO: 01412921

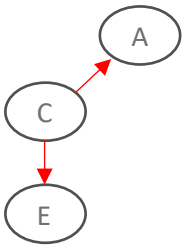
CS420 ASSIGNMENT 1 SOLUTIONS

QUESTION 1

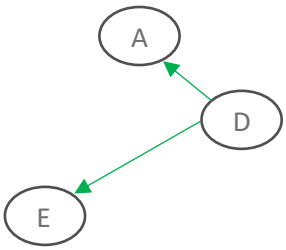
1. $A \perp E \mid \{C\}$ is false.

Justification:

- $A \perp E \mid \{C\}$ is true if given C, all trails between A and E are blocked in the Bayesian Network.
- Listing out the possible trails between A and E:



(Fig 1.1.1)



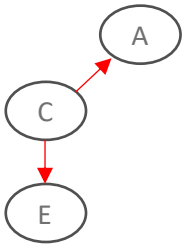
(Fig 1.1.2)

- In Fig 1.1.1, according to Common Cause structure, C decouples A and E. Hence, this trail is blocked.
- In Fig 1.1.2, according to Common Cause structure, D does not decouple A and E as it is not observed. Hence, this trail is not blocked.
- Since we have at least one trail between A and E is not blocked, the statement $A \perp E \mid \{C\}$ is false.

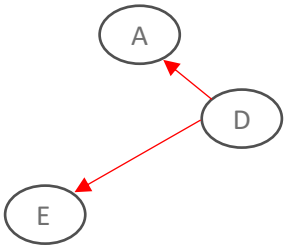
2. $A \perp E \mid \{C, D\}$ is true.

Justification:

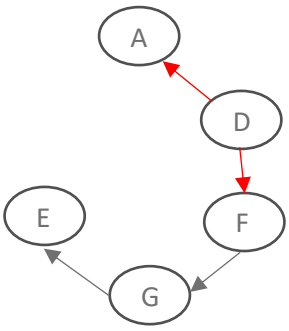
- $A \perp E \mid \{C, D\}$ is true if given C and D, all trails between A and E are blocked in the Bayesian Network.
- Listing out all possible trails between A and E:



(Fig 1.2.1)



(Fig 1.2.2)



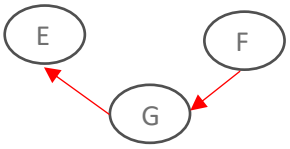
(Fig 1.2.3)

- In Fig 1.2.1, according to Common Cause structure, C decouples A and E. Hence, this trail is blocked.
- In Fig 1.2.2, according to Common Cause structure, D decouple A and E. Hence, this trail is blocked.
- In Fig 1.2.2, according to Common Cause structure and since D is observed, this trail blocked too.
- Since we have all possible trails between A and E being blocked, the statement $A \perp E \mid \{C, D\}$ is true.

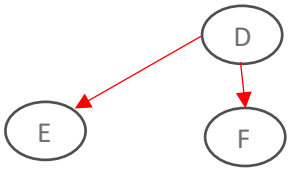
3. $F \perp E \mid \{G, D, A\}$ is true.

Justification:

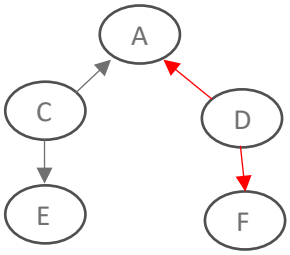
- $F \perp E \mid \{G, D, A\}$ is true if given G, D and A, all trails between F and E are blocked in the Bayesian Network.
- Listing out all possible trails between F and E:



(Fig 1.3.1)



(Fig 1.3.2)



(Fig 1.3.3)

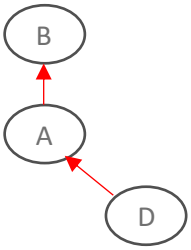
- In Fig 1.3.1, according to Cascade structure, G decouples F and E. Hence, this trail is blocked.
- In Fig 1.3.2, according to Common Cause structure, D decouples F and E. Hence, this trail is blocked.
- In Fig 1.3.3, according to Common Cause structure and since D is observed, this trail is blocked.
- Since we have all possible trails between F and E being blocked, the statement $F \perp E \mid \{G, D, A\}$ is true.

4. $B \perp D \mid \{A\}$ is true.

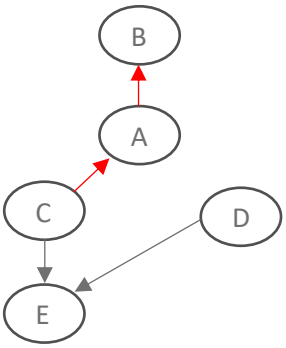
Justification:

- $B \perp D \mid \{A\}$ is true if given A, all trails between B and D are blocked in the Bayesian Network.

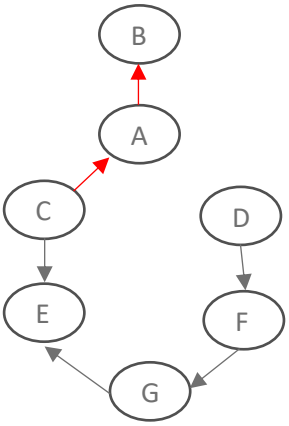
- Listing out all possible trails between B and D:



(Fig 1.4.1)



(Fig 1.4.2)



(Fig 1.4.4)

- In Fig 1.4.1, Fig 1.4.2, and Fig 1.4.3, according to Cascade structure and since A is given, all possible trails are blocked as A is the only node connected to B, i.e., all trails must pass through A to reach B.
- Since we have all possible trails between B and D being blocked, the statement $B \perp D \mid \{A\}$ is true.

QUESTION 2

Probabilities of each outcome for each machine:

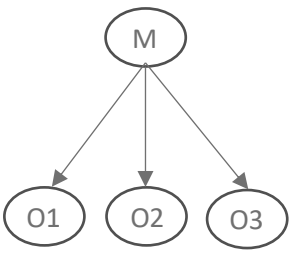
Outcome / Machine	0 (No Win)	1 (First Prize)	2 (Second Prize)	3 (Third Prize)
M1	0.8	0.05	0.075	0.075
M2	0.9	0.01	0.02	0.07
M3	0.7	0.1	0.1	0.1

*Note: Applied normalization property for computation of each probability in the table.

Probabilities of choosing a machine from the box:

- P(M1) = 0.2
- P(M2) = 0.7
- P(M3) = 0.1

(i) Bayesian Network



Explaination of each random variable of the Bayes Network, domain of each random variable and corresponding interpretation:

**R.V. represents 'Random Variable'*

Variable Name	Domain (Set of Values)	Interpretation
M	{1,2,3}	- R.V. 'M' denotes the machine that is chosen. - Three possible machines, i.e., M1, M2 and M3 are encoded into 1,2,3 respectively.
O1	{0,1,2,3}	- R.V. 'O1' denotes the first outcome value drawn. - Four possible values, i.e., 'No-Win', 'First Prize', 'Second Prize' and 'Third Prize' are encoded into 0,1,2,3 respectively.
O2	{0,1,2,3}	- R.V. 'O2' denotes the second outcome value drawn. - Four possible values, i.e., 'No-Win', 'First Prize', 'Second Prize' and 'Third Prize' are encoded into 0,1,2,3 respectively.
O3	{0,1,2,3}	- R.V. 'O3' denotes the third outcome value drawn. - Four possible values, i.e., 'No-Win', 'First Prize', 'Second Prize' and 'Third Prize' are encoded into 0,1,2,3 respectively.

(ii) Conditional Probability Table (CPT) associated with each variable:

CPT for R.V. M (Prior of M) or P(M):

P(M = 1)	P(M = 2)	P(M = 3)
0.2	0.7	0.1

CPT for R.V. O1 or P(O1 | M):

M	P(O1 = 0 | M)	P(O1 = 1 | M)	P(O1 = 2 | M)	P(O1 = 3 | M)
1	0.8	0.05	0.075	0.075
2	0.9	0.01	0.02	0.07
3	0.7	0.1	0.1	0.1

CPT for R.V. O2 or P(O2 | M):

M	P(O2 = 0 | M)	P(O2 = 1 | M)	P(O2 = 2 | M)	P(O2 = 3 | M)
1	0.8	0.05	0.075	0.075
2	0.9	0.01	0.02	0.07
3	0.7	0.1	0.1	0.1

CPT for R.V. O3 or P(O3 | M):

M	P(O3 = 0 | M)	P(O3 = 1 | M)	P(O3 = 2 | M)	P(O3 = 3 | M)
1	0.8	0.05	0.075	0.075
2	0.9	0.01	0.02	0.07
3	0.7	0.1	0.1	0.1

(iii)

- Want to find P(M | O1 = 0, O2 = 1, O3 = 3)
- Apply Bayes Rules:

$$P(M \mid O1 = 0, O2 = 1, O3 = 3) = \frac{P(M)\ P(O1 = 0, O2 = 1, O3 = 3 \mid M)}{P(O1 = 0, O2 = 1, O3 = 3)}$$

- Prior of M:

P(M = 1)	P(M = 2)	P(M = 3)
0.2	0.7	0.1

- Likelihood $P(O1 = 0, O2 = 1, O3 = 3 \mid M)$:
  - $P(O1 = 0, O2 = 1, O3 = 3 \mid M = 1) = 0.8 \bullet 0.05 \bullet 0.075 = 3 \times 10^{-3}$
  - $P(O1 = 0, O2 = 1, O3 = 3 \mid M = 2) = 0.9 \bullet 0.01 \bullet 0.07 = 6.3 \times 10^{-4}$
  - $P(O1 = 0, O2 = 1, O3 = 3 \mid M = 3) = 0.7 \bullet 0.1 \bullet 0.1 = 7 \times 10^{-3}$

- Numerator, $P(M)\ P(O1 = 0, O2 = 1, O3 = 3 \mid M)$:
  - $P(M = 1)\ P(O1 = 0, O2 = 1, O3 = 3 \mid M = 1)$
    $= 0.2 \bullet 3 \times 10^{-3}$
    $= 6 \times 10^{-4}$
  - $P(M = 2)\ P(O1 = 0, O2 = 1, O3 = 3 \mid M = 2)$
    $= 0.7 \bullet 6.3 \times 10^{-4}$
    $= 4.41 \times 10^{-4}$
  - $P(M = 3)\ P(O1 = 0, O2 = 1, O3 = 3 \mid M = 3)$
    $= 0.1 \bullet 7 \times 10^{-3}$
    $= 7 \times 10^{-4}$

- Denominator, $P(O1 = 0, O2 = 1, O3 = 3)$ is same across the probabilities, so:
  $P(M \mid O1 = 0, O2 = 1, O3 = 3) \propto P(M)\ P(O1 = 0, O2 = 1, O3 = 3 \mid M)$

- In conclusion, M3 was most likely to have been picked from the box.

QUESTION 3

Implementation of Bayes Network with the specified conditional probabilities into pgmpy (screenshots):

```
[48] # Install pgmpy on Google Collab
     !pip install pgmpy
```

```
[49] # Import relevant packages
     from pgmpy.models import BayesianModel
     from pgmpy.factors.discrete import TabularCPD
     import sys
```

Create the Bayesian network

```
[50] # Create a model which containts edges zof the graph
     model = BayesianModel([('Smoking', 'Yellow Fingers'),
                            ('Smoking', 'Cancer'),
                            ('Solar Flare', 'Radiation'),
                            ('Microwave', 'Radiation'),
                            ('Radiation', 'Cancer'),
                            ('Radiation', 'Skin Burn'),
                           ])

     # Enter conditional probability distribution for each variable:

     # Prior probability for Smoking
     cpd_smoking = TabularCPD(variable='Smoking', variable_card=2, values=[[0.9], [0.1]])

     # Prior probability for Solar Flare
     cpd_solar_flare = TabularCPD(variable='Solar Flare', variable_card=2, values=[[0.999], [0.001]])

     # Prior probability for Microwave
     cpd_microwave = TabularCPD(variable='Microwave', variable_card=2, values=[[0.001], [0.999]])

     # Conditional probability for P(Yellow Fingers | Smoking)
     cpd_yellow_fingers = TabularCPD(variable='Yellow Fingers',
                                     variable_card=2,
                                     values = [[0.9, 0.1],
                                               [0.1, 0.9]],
                                     evidence = ['Smoking'],
                                     evidence_card=[2])
```

```
[50] # Conditional probability for P(Radiation | Solar Flare, Microwave)
     cpd_radiation = TabularCPD(variable='Radiation',
                                variable_card=2,
                                values = [[0.99, 0.8, 0.7, 0.5],
                                          [0.01, 0.2, 0.3, 0.5]],
                                evidence = ['Solar Flare', 'Microwave'],
                                evidence_card=[2, 2])

     # Conditional probability for P(Cancer | Smoking, Radiation)
     cpd_cancer = TabularCPD(variable='Cancer',
                             variable_card=2,
                             values = [[0.9, 0.4, 0.8, 0.1],
                                       [0.1, 0.6, 0.2, 0.9]],
                             evidence = ['Smoking', 'Radiation'],
                             evidence_card=[2, 2])

     # Conditional probability for P(Skin Burn | Radiation)
     cpd_skin_burn = TabularCPD(variable='Skin Burn',
                            variable_card=2,
                            values = [[0.99, 0.9],
                                      [0.01, 0.1]],
                            evidence = ['Radiation'],
                            evidence_card=[2])

     # Insert the nodes and conditional probability into the defined model
     model.add_cpds(cpd_smoking,
                    cpd_solar_flare,
                    cpd_microwave,
                    cpd_yellow_fingers,
                    cpd_radiation,
                    cpd_cancer,
                    cpd_skin_burn)
```

Validate Network Parameters

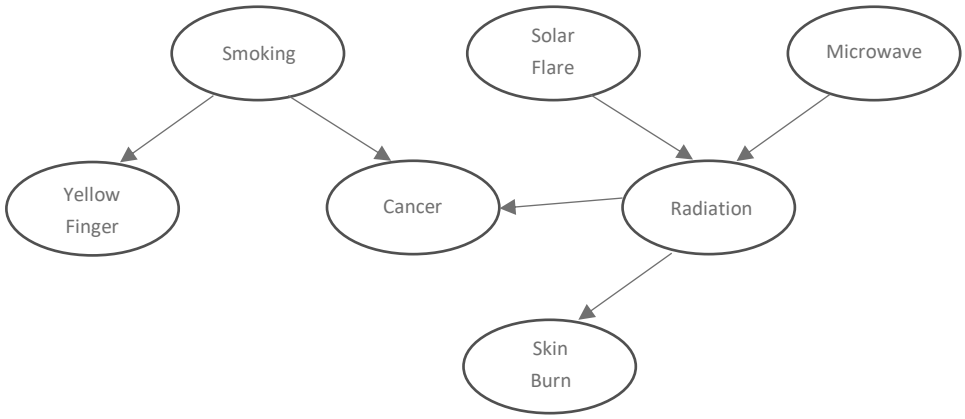
```
[51] # Validate the Bayesian Model
     print('Result of validation of constructed Bayesian Network: ' + str(model.check_model()))
```

```
Result of validation of constructed Bayesian Network: True
```

```
[52] # Check the conditional probability associated with each node to ensure correctness

     print('Prior probability for Smoking')
     print(model.get_cpds('Smoking'))

     print('\nPrior probability for Solar Flare')
     print(model.get_cpds('Solar Flare'))

     print('\nPrior probability for Microwave')
     print(model.get_cpds('Microwave'))

     print('\nConditional probability for P(Yellow Fingers | Smoking)')
     print(model.get_cpds('Yellow Fingers'))

     print('\nConditional probability for P(Radiation | Solar Flare, Microwave)')
     print(model.get_cpds('Radiation'))

     print('\nConditional probability for P(Cancer | Smoking, Radiation)')
     print(model.get_cpds('Cancer'))

     print('\nConditional probability for P(Skin Burn | Radiation)')
     print(model.get_cpds('Skin Burn'))
```

```
Prior probability for Smoking
+------------+-----+
| Smoking(0) | 0.9 |
+------------+-----+
| Smoking(1) | 0.1 |
+------------+-----+

Prior probability for Solar Flare
+----------------+-------+
| Solar Flare(0) | 0.999 |
+----------------+-------+
| Solar Flare(1) | 0.001 |
+----------------+-------+

Prior probability for Microwave
+--------------+-------+
| Microwave(0) | 0.001 |
+--------------+-------+
| Microwave(1) | 0.999 |
+--------------+-------+

Conditional probability for P(Yellow Fingers | Smoking)
+-------------------+------------+------------+
| Smoking           | Smoking(0) | Smoking(1) |
+-------------------+------------+------------+
| Yellow Fingers(0) | 0.9        | 0.1        |
+-------------------+------------+------------+
| Yellow Fingers(1) | 0.1        | 0.9        |
+-------------------+------------+------------+

Conditional probability for P(Radiation | Solar Flare, Microwave)
+--------------+----------------+-----+----------------+----------------+
| Solar Flare  | Solar Flare(0) | ... | Solar Flare(1) | Solar Flare(1) |
+--------------+----------------+-----+----------------+----------------+
| Microwave    | Microwave(0)   | ... | Microwave(0)   | Microwave(1)   |
+--------------+----------------+-----+----------------+----------------+
| Radiation(0) | 0.99           | ... | 0.7            | 0.5            |
+--------------+----------------+-----+----------------+----------------+
| Radiation(1) | 0.01           | ... | 0.3            | 0.5            |
+--------------+----------------+-----+----------------+----------------+

Conditional probability for P(Cancer | Smoking, Radiation)
+-----------+-------------+-------------+-------------+-------------+
| Smoking   | Smoking(0)  | Smoking(0)  | Smoking(1)  | Smoking(1)  |
+-----------+-------------+-------------+-------------+-------------+
| Radiation | Radiation(0)| Radiation(1)| Radiation(0)| Radiation(1)|
+-----------+-------------+-------------+-------------+-------------+
| Cancer(0) | 0.9         | 0.4         | 0.8         | 0.1         |
+-----------+-------------+-------------+-------------+-------------+
| Cancer(1) | 0.1         | 0.6         | 0.2         | 0.9         |
+-----------+-------------+-------------+-------------+-------------+

Conditional probability for P(Skin Burn | Radiation)
+--------------+--------------+--------------+
| Radiation    | Radiation(0) | Radiation(1) |
+--------------+--------------+--------------+
| Skin Burn(0) | 0.99         | 0.9          |
+--------------+--------------+--------------+
| Skin Burn(1) | 0.01         | 0.1          |
+--------------+--------------+--------------+
```

1. Bayesian Network showing the nodes and arrows, i.e., relationship among all the variables:



2. P(Radiation | Cancer = 1):

```
[53] ############## Inference #################

     # Import VariableElimination library
     from pgmpy.inference import VariableElimination

     # Set up for Variable Elimination
     infer = VariableElimination(model)
```

```
[57] # Question 3 (Part 2) Compute P(Radiation | Cancer = 1)
     phi_query = infer.query(['Radiation'], evidence={'Cancer':1}, joint = False)
     factor = phi_query['Radiation']
     print('Probability of Radiation given Cancer = 1:')
     print(factor)

Finding Elimination Order: : 100% 3/3 [00:00<00:00, 35.77it/s]
Eliminating: Microwave: 100% 3/3 [00:00<00:00, 48.73it/s]
Probability of Radiation given Cancer = 1:
+--------------+------------------+
| Radiation    |   phi(Radiation) |
+==============+==================+
| Radiation(0) |           0.4110 |
+--------------+------------------+
| Radiation(1) |           0.5890 |
+--------------+------------------+
```

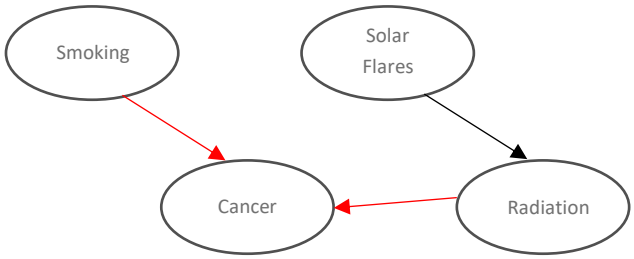
- Based on the computation,
    - P(Radiation = 0 | Cancer = 1) = 0.4110
    - P(Radiation = 1 | Cancer = 1) = 0.5890

3. P(Cancer | Skin Burn = 1):

```
[55] # Question 3 (Part 3) Compute P(Cancer | Skin Burn = 1)
     phi_query = infer.query(['Cancer'], evidence={'Skin Burn':1}, joint = False)
     factor = phi_query['Cancer']
     print('Probability of Cancer given Skin Burn = 1:')
     print(factor)

Finding Elimination Order: : 0% 0/4 [00:00<?, ?it/s]
Eliminating: Radiation: 100% 4/4 [00:00<00:00, 58.12it/s]
Probability of Cancer given Skin Burn = 1:
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.5185 |
+-----------+---------------+
| Cancer(1) |        0.4815 |
+-----------+---------------+
```

- Based on the computation,
    - P(Cancer = 0| Skin Burn = 1) = 0.5185
    - P(Cancer = 1 | Skin Burn = 1) = 0.4815

4. Smoking and Solar Flares are independent. The only trail (as shown below) between 'Smoking' and 'Solar Flares' is blocked. This is because the 'Cancer' node (or any of its descendants) are not observed. Hence, 'Smoking' and 'Solar Flares' are D-separated by 'Cancer'.



5. P(Cancer = 1 | Microwave = 0):

```
[56] # Question 3 (Part 5) Compute P(Cancer = 1 | Microwave = 0)
     phi_query = infer.query(['Cancer'], evidence={'Microwave':0}, joint = False)
     factor = phi_query['Cancer']
     print('Probability of Cancer given Microwave = 0:')
     print(factor)

     Finding Elimination Order: : 0%|          | 0/3 [00:00<?, ?it/s]
     Eliminating: Radiation: 100%|██████████| 3/3 [00:00<00:00, 48.11it/s]
     Probability of Cancer given Microwave = 0:
     +-----------+---------------+
     | Cancer    |   phi(Cancer) |
     +===========+===============+
     | Cancer(0) |        0.8846 |
     +-----------+---------------+
     | Cancer(1) |        0.1154 |
     +-----------+---------------+
```

- Based on the computation, P(Cancer = 1 | Microwave = 0) = 0.1154

QUESTION 4

(a) Download and load the k_mnist dataset:

```
[1] # Import the required libraries

    %matplotlib inline
    %tensorflow_version 2.x

    import numpy as np
    import pandas as pd
    import tensorflow as tf
    import matplotlib.pyplot as plt
    import tensorflow_datasets as tfds

    from keras import layers
```

```
[2] # (a) Download and load the k_mnist dataset.
    (train_images, train_labels), (test_images, test_labels) = tfds.as_numpy(tfds.load(
        'kmnist',
        split=['train', 'test'],
        batch_size=-1,
        as_supervised=True,
    ))
```

```
[3] # Reshape the images to required dimension
    train_images = train_images.reshape((60000, 28, 28))
    test_images = test_images.reshape((10000, 28, 28))
```

```
[4] # Check shapes of train_images, train_labels etc
    print(train_images.shape)
    print(train_labels.shape)
    print(test_images.shape)
    print(test_labels.shape)

    (60000, 28, 28)
    (60000,)
    (10000, 28, 28)
    (10000,)
```

```
[5] # Assign name to corresponding classes
    class_names = ['o', 'ki', 'su', 'tsu', 'na', 'ha', 'ma', 'ya', 're', 'wo']

    # Show first 25 training images below
    plt.figure(figsize=(10,10))

    for i in range(25):
        plt.subplot(5,5,i+1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        plt.imshow(train_images[i], cmap=plt.cm.binary)
        plt.xlabel(class_names[train_labels[i]])
```

(b) Create an ANN with following configuration:

```
[7] # (b) Create ANN with :
    # 1 input layer
    # 3 hidden layers, with 128, 128, and 64 nodes each, and all using relu activation function
    # This configuration achieves overall accuracy of >80% on the testing dataset after training

    model = tf.keras.Sequential()
    model.add(layers.Flatten(input_shape=(28, 28)))
    model.add(layers.Dense(128, activation=tf.nn.relu))
    model.add(layers.Dense(128, activation=tf.nn.relu))
    model.add(layers.Dense(64, activation=tf.nn.relu))
```

(c) Create 1 output layer with following configuration:

```
[8] # (c) Create 1 output layer with:
    # size of 10
    # softmax activation function
    model.add(layers.Dense(10, activation=tf.nn.softmax))
```

- The size of output layer is 10.
- The activation function for the output layer is softmax.

(d) Compile and train the neural network:

```
[10] # (d) Compile the neural network with:
     # cross entropy loss function
     # Adamax optimizer with learning rate of 1×10^-3
     model.compile(optimizer=tf.keras.optimizers.Adamax(learning_rate=1e-3),
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])
```

```
[11] # Train the neural network
     # Run the stochastic gradient descent for batch size : 16 and epochs : 6
     model.fit(train_images, train_labels, batch_size=16, epochs=6)
```

```
Epoch 1/6
3750/3750 [==============================] - 14s 4ms/step - loss: 1.8914 - accuracy: 0.7003
Epoch 2/6
3750/3750 [==============================] - 15s 4ms/step - loss: 0.5382 - accuracy: 0.8504
Epoch 3/6
3750/3750 [==============================] - 11s 3ms/step - loss: 0.3898 - accuracy: 0.8921
Epoch 4/6
3750/3750 [==============================] - 11s 3ms/step - loss: 0.2994 - accuracy: 0.9144
Epoch 5/6
3750/3750 [==============================] - 12s 3ms/step - loss: 0.2364 - accuracy: 0.9317
Epoch 6/6
3750/3750 [==============================] - 11s 3ms/step - loss: 0.1947 - accuracy: 0.9427
<keras.callbacks.History at 0x7f49a1b23890>
```

- The appropriate loss function is cross-entropy loss.

```
[12] # Evaluate the result of the model applied on the test images
     test_loss, test_acc = model.evaluate(test_images, test_labels)
     print('Test accuracy:', test_acc)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.7236 - accuracy: 0.8309
Test accuracy: 0.8309000134468079
```

```
[13] # Get all predictions for test data
     predictions = model.predict(test_images)
```

```
[14] # Prediction visualization
     # Correct predictions are highlighted in green
     # Incorrect predictions are highlighted in red

     plt.figure(figsize=(10,10))

     for i in range(25):
         plt.subplot(5,5,i+1)
         plt.xticks([])
         plt.yticks([])
         plt.grid(False)
         plt.imshow(test_images[i], cmap=plt.cm.binary)
         predicted_label = np.argmax(predictions[i])
         true_label = test_labels[i]
         if predicted_label == true_label:
           color = 'green'
         else:
           color = 'red'
         plt.xlabel("{} ({})".format(class_names[predicted_label],
                                     class_names[true_label]),
                                     color=color)
```

(e) Plot the average training error on the y-axis vs the epoch number:

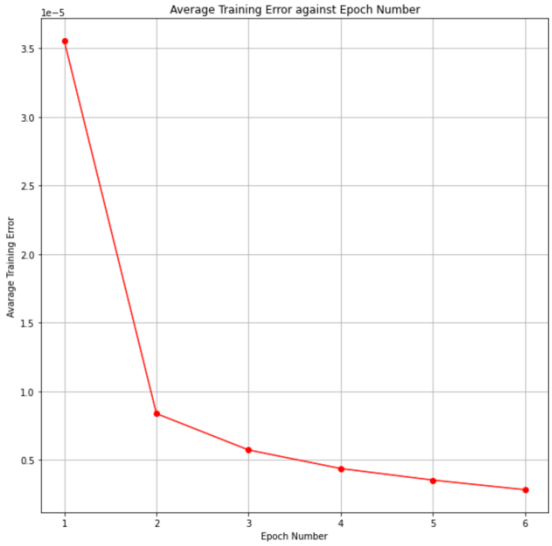
```
[15] # (e) Plot the average training error on the y-axis vs the epoch number

     # Simulate a similar training environment
     eval_model = tf.keras.Sequential()
     eval_model.add(layers.Flatten(input_shape=(28, 28)))
     eval_model.add(layers.Dense(128, activation=tf.nn.relu))
     eval_model.add(layers.Dense(128, activation=tf.nn.relu))
     eval_model.add(layers.Dense(64, activation=tf.nn.relu))
     eval_model.add(layers.Dense(10, activation=tf.nn.softmax))

     eval_model.compile(optimizer=tf.keras.optimizers.Adamax(learning_rate=1e-3),
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])

     hist = eval_model.fit(train_images, train_labels, batch_size=16, epochs=6)
```

```
Epoch 1/6
3750/3750 [==============================] - 12s 3ms/step - loss: 2.1340 - accuracy: 0.7109
Epoch 2/6
3750/3750 [==============================] - 11s 3ms/step - loss: 0.5022 - accuracy: 0.8559
Epoch 3/6
3750/3750 [==============================] - 11s 3ms/step - loss: 0.3430 - accuracy: 0.9003
Epoch 4/6
3750/3750 [==============================] - 11s 3ms/step - loss: 0.2617 - accuracy: 0.9219
Epoch 5/6
3750/3750 [==============================] - 12s 3ms/step - loss: 0.2112 - accuracy: 0.9355
Epoch 6/6
3750/3750 [==============================] - 12s 3ms/step - loss: 0.1691 - accuracy: 0.9474
```

```
[16] # Extract the training metrics for evaluation
     hist_df = pd.DataFrame(hist.history)

     epoch = [1,2,3,4,5,6]
     hist_df['epoch no'] = epoch
     hist_df = hist_df.set_index('epoch no')

     hist_df
```

epoch no	loss	accuracy
1	2.134021	0.710883
2	0.502182	0.855867
3	0.342971	0.900267
4	0.261690	0.921867
5	0.211160	0.935450
6	0.169056	0.947350

```
[17] # Plot the result
     SAMPLE_NO = 60000

     hist_arr = hist_df.to_numpy()

     x = np.array([1,2,3,4,5,6])


     y = np.array([
          hist_arr[0][0],
          hist_arr[1][0],
          hist_arr[2][0],
          hist_arr[3][0],
          hist_arr[4][0],
          hist_arr[5][0],
         ])


     y = y / SAMPLE_NO

     plt.figure(figsize=(10,10))
     plt.plot(x,y,color='red',marker='o')
     plt.xlabel('Epoch Number')
     plt.ylabel('Avarage Training Error')
     plt.title("Average Training Error against Epoch Number")
     plt.grid(True)
     plt.show()
```



(f) Shows the final accuracy for different classes and overall accuracy on the testing data:

```
[18] # (f) final accuracy of different classes and overall accuracy on the testing data

     predicted_classes = [0,0,0,0,0,0,0,0,0,0]
     expected_classes = [0,0,0,0,0,0,0,0,0,0]

     for i in range(10000):
       exp_label = int(test_labels[i])
       pre_label = int(np.argmax(predictions[i]))

       if exp_label == pre_label:
         predicted_classes[pre_label] += 1

       expected_classes[exp_label] += 1

     print('-----Final accuracy for different classes-----')

     for i in range(10):
       perc_val = "{:.2f}".format(predicted_classes[i]/expected_classes[i] * 100)
       print("For label " + str(i) + " ('" + str(class_names[i]) +
             "') : Accuracy is " + str(perc_val) + "%\n")

     print('\n\n-----Final accuracy for test dataset-----')
     print('overall accuracy: ' + str("{:.2f}".format(test_acc * 100)) + '\n\n')
```

```
-----Final accuracy for different classes-----
For label 0 ('o') : Accuracy is 88.10%

For label 1 ('ki') : Accuracy is 83.70%

For label 2 ('su') : Accuracy is 84.10%

For label 3 ('tsu') : Accuracy is 87.90%

For label 4 ('na') : Accuracy is 78.00%

For label 5 ('ha') : Accuracy is 76.40%

For label 6 ('ma') : Accuracy is 89.10%

For label 7 ('ya') : Accuracy is 80.00%

For label 8 ('re') : Accuracy is 80.30%

For label 9 ('wo') : Accuracy is 83.30%


-----Final accuracy for test dataset-----
overall accuracy: 83.09
```

QUESTION 5

```
[9] # Import the required libraries
    import cv2
    import tensorflow as tf
    import keras.applications.mobilenet_v2

    from keras.datasets import cifar10
```

Import Dataset

```
[10] # Load training data, labels; and testing data and their true labels
     (train_images, train_labels), (test_images, test_labels) = cifar10.load_data()
     print ('Training data seize:', train_images.shape)
     print ('Test data size', test_images.shape)

     # Normalize pixel values between -1 and 1
     train_images = train_images / 127.5 - 1
     test_images = test_images / 127.5 - 1

     # Class names for different classes
     class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer','dog', 'frog', 'horse', 'ship', 'truck']
```

```
Training data seize: (50000, 32, 32, 3)
Test data size (10000, 32, 32, 3)
```

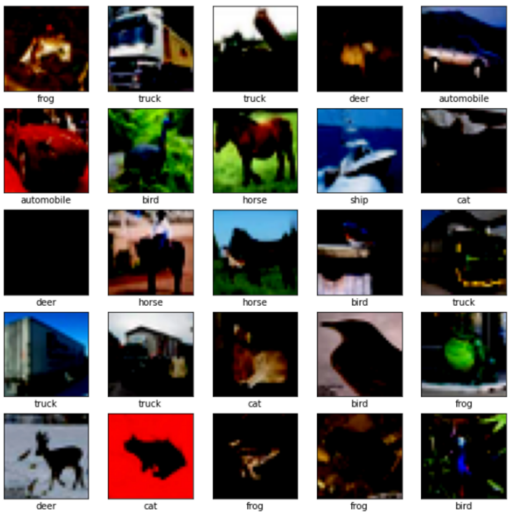
```
[8] # Testing the GPU
    device_name = tf.test.gpu_device_name()
    if device_name != '/device:GPU:0':
      raise SystemError('GPU device not found')
    print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

Visualize dataset

```
[12] # Show first 25 training images below
     plt.figure(figsize=(10,10))
     for i in range(25):
         plt.subplot(5,5,i+1)
         plt.xticks([])
         plt.yticks([])
         plt.grid(False)
         plt.imshow(train_images[i], cmap=plt.cm.binary)
         plt.xlabel(class_names[train_labels[i][0]])
```

[12]



```
[13] # Check the shapes of train_images, train_labels, etc
     print(train_images.shape)
     print(train_labels.shape)
     print(test_images.shape)
     print(test_labels.shape)
```

```
(50000, 32, 32, 3)
(50000, 1)
(10000, 32, 32, 3)
(10000, 1)
```

Resize images for use with MobileNetV2

```
[14] # Upsize all training and testing images to 96x96 for use with mobile net

     # Minimum size requried for mobileNetV2
     minSize = 96

     # Initialize a numpy array resized_train_images to store all the resized training images
     resized_train_images = np.zeros((50000, minSize, minSize, 3), dtype=np.float32)

     # Resize the training images
     for i in range(50000):
       resized_train_images[i] = cv2.resize(train_images[i], dsize=(minSize, minSize), interpolation=cv2.INTER_AREA)

     # Initialize a numpy array resized_test_images to store all the resized test images
     resized_test_images = np.zeros((10000, minSize, minSize, 3), dtype=np.float32)

     # Resize the test images
     for i in range(10000):
       resized_test_images[i] = cv2.resize(test_images[i], dsize=(minSize, minSize), interpolation=cv2.INTER_AREA)
```

```
[15] print(resized_train_images.shape)
     print(train_images.shape)
```

```
(50000, 96, 96, 3)
(50000, 32, 32, 3)
```

(a) Download the pre-trained MobileNetV2 network from Keras Application:

(b) Remove the final output layer of the network:

Download MobileNetV2 model

```
# (a) Download the pre-trained MobileNetV2 network from Keras Applications

IMG_SHAPE = (96, 96, 3)

# (b) Remove the final output layer of the network by setting include_top to False
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE, include_top=False, weights='imagenet')

# Freeze the layers in pre-trained model, except last 8 layers
for layer in base_model.layers[:-8]:
  layer.trainable = False

# Show the summary and read the number of trainable parameters
base_model.summary()
```

(c) Extend the MobileNetV2 model

Add custom layers at the end of downloaded model

```
# (c) Extend the MobileNetV2 model

# Create a global average layer before connecting MobileNetV2 with new model
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()

model = tf.keras.Sequential()
model.add(base_model)

# Extend the MobileNetV2 model with:
# 1 global average layer
# 6 hidden dense layers, with 128, 64, 128, 64, 128, 64 nodes each, and all using relu activation function
# 1 output layer with size of 10, and softmax activation function
# This configuration achieves overall accuracy of >85% on the testing dataset after training

model.add(global_average_layer)

model.add(layers.Dense(128, activation=tf.nn.relu))
model.add(layers.Dense(64, activation=tf.nn.relu))
model.add(layers.Dense(128, activation=tf.nn.relu))
model.add(layers.Dense(64, activation=tf.nn.relu))
model.add(layers.Dense(128, activation=tf.nn.relu))
model.add(layers.Dense(64, activation=tf.nn.relu))

model.add(layers.Dense(10, activation=tf.nn.softmax))
```

(d) Compile the model using appropriate loss function, and train the modified network from part (c):

Add loss function, compile and train the model, and check accuracy on test data

```
[74] # (d) Compile the neural network with:
     # cross-entropy loss function
     # Adamax optimizer with learning rate of 1x10^-3

     model.compile(optimizer=tf.keras.optimizers.Adamax(learning_rate=1e-3),
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])
```

```
[75] # Show the hierarchy of the model
     model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_96 (Functi  (None, 3, 3, 1280)       2257984
 onal)

 global_average_pooling2d_10  (None, 1280)             0
  (GlobalAveragePooling2D)

 dense_55 (Dense)            (None, 128)               163968

 dense_56 (Dense)            (None, 64)                8256

 dense_57 (Dense)            (None, 128)               8320

 dense_58 (Dense)            (None, 64)                8256

 dense_59 (Dense)            (None, 128)               8320

 dense_60 (Dense)            (None, 64)                8256

 dense_61 (Dense)            (None, 10)                650

=================================================================
Total params: 2,464,010
Trainable params: 936,586
Non-trainable params: 1,527,424
_________________________________________________________________
```

- Freeze all the layers except the last 8 layers of MobileNetV2 model by setting those layers to layer.trainable = False
- Train the model with parameters from those 8 layers together with the newly added layers

```
[ ] # Train the neural network
    # Run the stochastic gradient descent for batch size : 32 and epochs : 6
    epochs = 6
    batch_size = 32

    hist2 = model.fit(resized_train_images, train_labels, batch_size=batch_size, epochs=epochs)
```

```
Epoch 1/6
1563/1563 [==============================] - 435s 274ms/step - loss: 0.7156 - accuracy: 0.7589
Epoch 2/6
1563/1563 [==============================] - 414s 265ms/step - loss: 0.5022 - accuracy: 0.8297
Epoch 3/6
1563/1563 [==============================] - 414s 265ms/step - loss: 0.4090 - accuracy: 0.8592
Epoch 4/6
1563/1563 [==============================] - 415s 266ms/step - loss: 0.3353 - accuracy: 0.8833
Epoch 5/6
1563/1563 [==============================] - 419s 268ms/step - loss: 0.2723 - accuracy: 0.9047
Epoch 6/6
1563/1563 [==============================] - 418s 267ms/step - loss: 0.2222 - accuracy: 0.9222
```

```
[78] test_loss, test_acc = model.evaluate(resized_test_images, test_labels, batch_size=32)
```

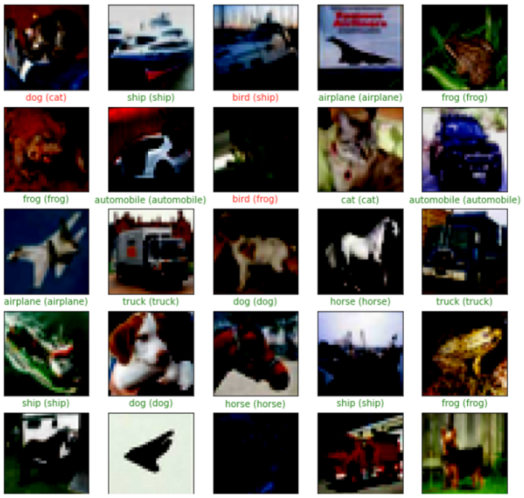
```
313/313 [==============================] - 61s 192ms/step - loss: 0.5373 - accuracy: 0.8524
```

```
[85] print('Test accuracy:', test_acc)
```

```
Test accuracy: 0.8524000144004822
```

```
[80] # Get all predictions for test data
     predictions = model.predict(resized_test_images)
```

```
[81] # Code to visualize predictions
     # Incorrect predictions are highlighted in red
     plt.figure(figsize=(10,10))
     for i in range(25):
         plt.subplot(5,5,i+1)
         plt.xticks([])
         plt.yticks([])
         plt.grid(False)
         plt.imshow(resized_test_images[i], cmap=plt.cm.binary)
         predicted_label = np.argmax(predictions[i])
         true_label = test_labels[i][0]
         if predicted_label == true_label:
           color = 'green'
         else:
           color = 'red'
         plt.xlabel("{} ({})".format(class_names[predicted_label],
                                     class_names[true_label]),
                                     color=color)
```



(1) A total of 8 extra layers added to the pre-trained MobileNetV2 model (after removing the final output layer of the network):

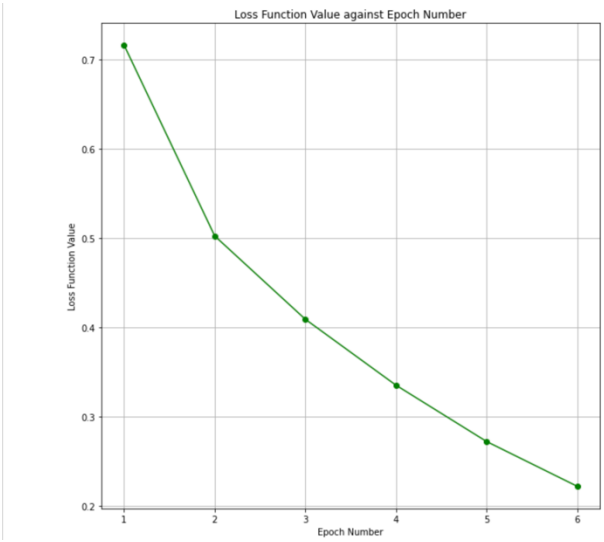
- 1 global average layer created using GlobalAveragePooling2D() method
- 6 hidden dense layers:
  - 1st dense layer: 128 nodes, with relu activation function
  - 2nd dense layer: 64 nodes, with relu activation function
  - 3rd dense layer: 128 nodes, with relu activation function
  - 4th dense layer: 64 nodes, with relu activation function
  - 5th dense layer: 128 nodes, with relu activation function
  - 6th dense layer: 64 nodes, with relu activation function
- 1 output layer of size 10, with softmax activation function

(2) Plot the loss function value with respect to epoch number of training data:

Extra code for producing different plots

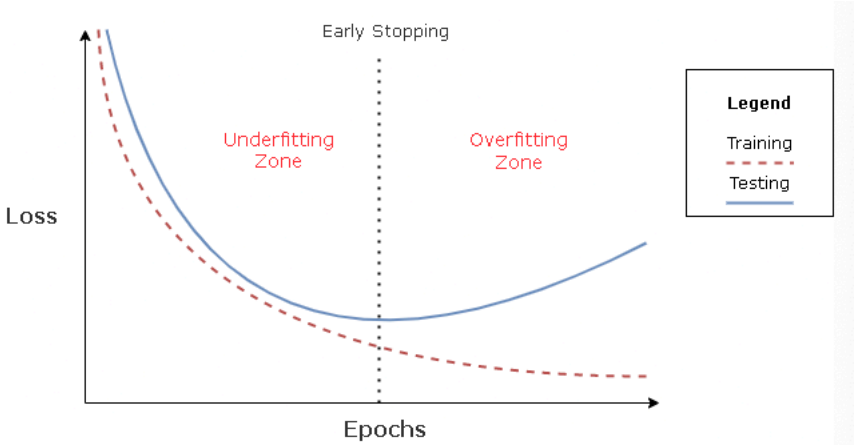
```
[46] #Extract the training metrics for evaluation
     hist_df = pd.DataFrame(hist2.history)

     epoch = [1,2,3,4,5,6]
     hist_df['epoch no'] = epoch
     hist_df = hist_df.set_index('epoch no')

     # Plot the result
     SAMPLE_NO = 60000

     hist_arr = hist_df.to_numpy()

     x = np.array([1,2,3,4,5,6])
```

```
[88] y = np.array([
          hist_arr[0][0],
          hist_arr[1][0],
          hist_arr[2][0],
          hist_arr[3][0],
          hist_arr[4][0],
          hist_arr[5][0],
         ])

     plt.figure(figsize=(10,10))
     plt.plot(x,y,color='green',marker='o')
     plt.xlabel('Epoch Number')
     plt.ylabel('Loss Function Value')
     plt.title("Loss Function Value against Epoch Number")
     plt.grid(True)
     plt.show()
```



How did you decide when to terminate training?

'Early stopping should be used almost universally' – Ian Goodfellow et al., Deep Learning. In fact, this is my main approach for solving both question 4 and question 5 of CS420 Assignment 1. This is done by analyzing the loss graph in both the training and validation(testing) until they split as the epoch number increases. Initially, both training and validation decreases at a significant degree. Eventually, we reach a point where they split, i.e., the training loss flatten out/continue to decrease at a very low rate, while validation loss starts to increase as shown below:



Reference: https://towardsdatascience.com/the-million-dollar-question-when-to-stop-training-deep-learning-models-fa9b488ac04d

However, due to limitation of GPU computation power and other resources, I decided to stop the training before reaching the ideal 'split' when training loss ≈ 0.2 and epoch no. = 6. This may not be the optimal solution but is decent to provide a good prediction on the testing dataset.

How did you decide the mini-batch size for the training?

According to the published article, 'The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset" by Ibrahem Kandel, Mauro Castelli, a higher batch size does not usually achieve high accuracy, and the learning rate and the optimizer. Hence, for a training dataset of size 50,000 images, which is a medium if not small dataset, my preference is to lower the learning rate and decrease the batch size to 32. This method achieved a better outcome after several experiments.

(3) Show the accuracy of the trained classifier over the entire testing dataset:

```
[89] # Show the overall accuracy

     print('-----Final accuracy for test dataset-----')
     print('overall accuracy: ' + str("{:.2f}".format(test_acc * 100)) + '\n\n')

     -----Final accuracy for test dataset-----
     overall accuracy: 85.24
```

~END~