CS420 Assignment 1

Question 1

1. A ⊥ E | {D}
    - A → B → D → E
        - Blocked due to cascade on D, it is observed.
    - A → B ← C → F ← E
        - Blocked due to v-structure on F, neither F nor its its descendants are observed.
    - A → B → D → G → H ← F ← E
        - Blocked due to cascade on D, it is observed.
    - A → D → E
        - Blocked due to cascade on D, it is observed.
    - A → D → G → H → F → E
        - Blocked due to cascade on D, it is observed.

    Since all the trails are blocked, the statement is true.

2. A ⊥ H | {B, G}
    - A → D → G → H
        - Blocked due to cascade on D, it is observed.
    - A → D → E → F → H
        - Blocked due to cascade on D, it is observed.
    - A → D ← B ← C → F → H
        - Active trail
    - A → B ← C → F → H
        - Active trail

- A → B → D → G → H
  - Blocked due to cascade on B, it is observed.
- A → B → D → E → F → H
  - Blocked due to cascade on B, it is observed.
- A → B ← C → F ← E ← D → G → H
  - Blocked due to cascade on G, it is observed.

Since there are active trails, the statement is false.

3. A ⊥ C
   - A → B ← C
     - Blocked due to v-structure on B, neither B nor its descendants are observed.
   - A → B → D → E → F ← C
     - Blocked due to v-structure on F, neither F nor its descendants are observed.
   - A → B → D → G → H ← F ← C
     - Blocked due to v-structure on H, neither H nor its descendants are observed.
   - A → D → E → F ← C
     - Blocked due to v-structure on F, neither F nor its descendants are observed.
   - A → D ← B ← C
     - Blocked due to v-structure on D, neither D nor its descendants are observed.
   - A → D → G → H ← F ← C
     - Blocked due to v-structure on H, neither H nor its descendants are observed.

   Since all the trails are blocked, the statement is true.

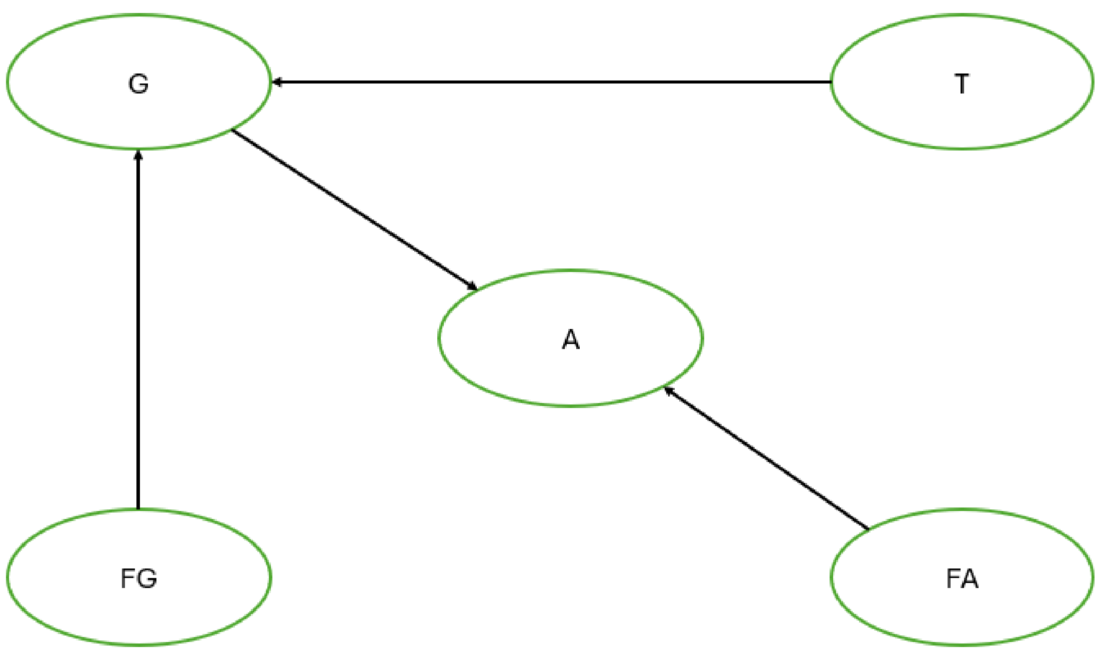
4. {D} ⊥ {C, F} | {A, B, E, G}

- D → E → F ← C
    - Blocked due to cascade on E, it is observed.
- D ← B ← C ← F
    - Blocked due to cascade on B, it is observed.
- D ← A → B → C → F
    - Blocked due to common cause on A, it is observed.
- D → G → H → F → C
    - Blocked due to cascade on G, it is observed.

Since all the trails are blocked, the statement is true.

Question 2

A)



B)

Variable Name	Domain (list all the entries of the domain)
T	{Low actual temperature = 0, High actual temperature = 1}
G	{Low measured temperature = 0, High measured temperature = 1}
FG	{Gauge not faulty = 0, Gauge is faulty = 1}
FA	{Alarm not faulty = 0, Alarm is faulty = 1}
A	{Alarm does not sound = 0, Alarm sounds = 1}

C)

$P(G = 1 \mid FG = 1, T) = 0.2$

$P(G = 0 \mid FG = 1, T) = 1 - 0.2 = 0.8$

$P(G = 1 \mid FG = 0, T) = 0.9$

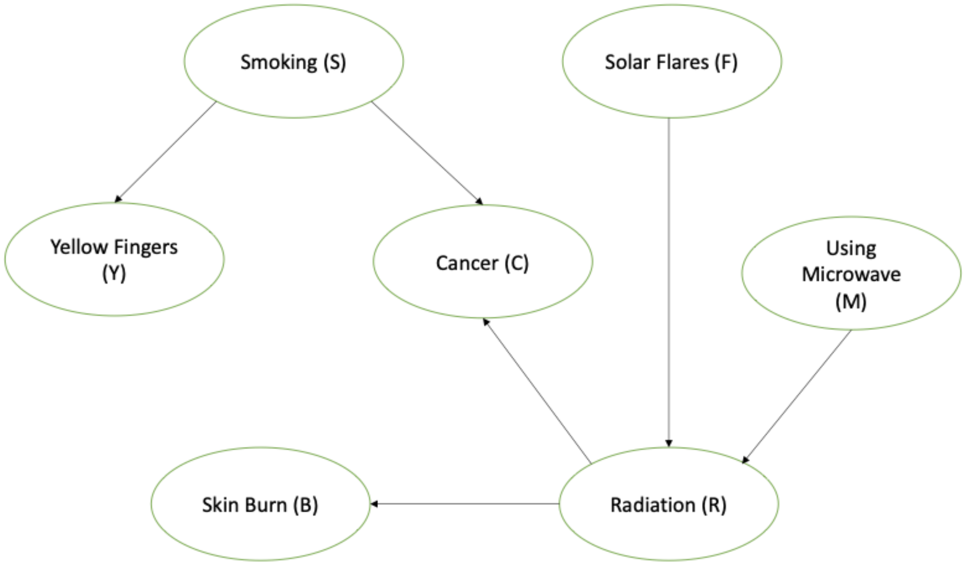
$P(G = 0 \mid FG = 0, T) = 1 - 0.9 = 0.1$

FG	T	$P(G = 1 \mid FG, T)$	$P(G = 0 \mid FG, T)$
1	1	0.2	0.8
1	0	0.8	0.2
0	1	0.9	0.1
0	0	0.1	0.9

D)

FA	G	$P(A = 1 \mid FA, G)$	$P(A = 0 \mid FA, G)$
0	0	1	0
0	1	1	0
1	0	0	1
1	1	0	1

Question 3



1.

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

```
[26] # We first create a model which containts edges of the graph
     model = BayesianModel([('Smoking', 'Yellow Fingers'), ('Smoking', 'Cancer'), ('Solar Flares', 'Radiation'), ('Using Microwave', 'Radiation'), ('Radiation',

     # Enter conditional probability distribution for each variable

     # Prior probability for Smoking P(S)
     cpd_S = TabularCPD(variable='Smoking', variable_card=2, values=[[0.8], [0.2]])

     # Prior probability for Solar Flares P(F)
     cpd_F = TabularCPD(variable='Solar Flares', variable_card=2, values=[[0.999], [0.001]])

     # Prior probability for Using Microwave P(M)
     cpd_M = TabularCPD(variable='Using Microwave', variable_card=2, values=[[0.1], [0.9]])

     WARNING:pgmpy:BayesianModel has been renamed to BayesianNetwork. Please use BayesianNetwork class, BayesianModel will be removed in future.
```

```
[27] # Conditional probability for Skin Burn or P(B|R)
     cpd_B = TabularCPD(variable='Skin Burn',variable_card=2, values = [[0.99, 0.9], [0.01, 0.1]],
                                      evidence = ['Radiation'],
                                      evidence_card=[2])

     # Conditional probability for Radiation or P(R|F, M)
     cpd_R = TabularCPD(variable='Radiation', variable_card=2, values =  [[0.99, 0.9, 0.7, 0.5], [0.01, 0.1, 0.3, 0.5]],
                                      evidence = ['Solar Flares', 'Using Microwave'],
                                      evidence_card=[2, 2])
     # Conditional probability for Cancer or P(C|S, R)
     cpd_C = 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 Yellow Fingers or P(Y|S)
     cpd_Y = TabularCPD(variable='Yellow Fingers',variable_card=2, values = [[0.9, 0.1], [0.1, 0.9]],
                                      evidence = ['Smoking'],
                                      evidence_card=[2])

     model.add_cpds(cpd_B, cpd_R, cpd_C, cpd_S, cpd_F, cpd_M, cpd_Y)
```

```
[28] print(model.check_model())

     True
```

```python
from pgmpy.inference import VariableElimination
# Going to do variable elimination
infer = VariableElimination(model)

# Compute probability of Radiation given 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)

# Compute probability of Cancer given 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)

# Compute probability of Cancer given Using Microwave = 0
phi_query = infer.query(['Cancer'], evidence={'Using Microwave':0}, joint = False)
factor = phi_query['Cancer']
print('Probability of Cancer given Using Microwave = 0')
print(factor)
```

```
Probability of Radiation given Cancer = 1
+--------------+------------------+
| Radiation    |   phi(Radiation) |
+==============+==================+
| Radiation(0) |           0.6438 |
+--------------+------------------+
| Radiation(1) |           0.3562 |
+--------------+------------------+
Probability of Cancer given Skin Burn = 1
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.6092 |
+-----------+---------------+
| Cancer(1) |        0.3908 |
+-----------+---------------+
Probability of Cancer given Using Microwave = 0
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.8744 |
+-----------+---------------+
| Cancer(1) |        0.1256 |
+-----------+---------------+
```

2. $P(R = 1 \mid C = 1) = 0.6438$
3. $P(C = 1 \mid B = 1) = 0.3908$

4. Trails between Smoking and Using Microwave:
   a. S → C ← R ← M

   Since Cancer is observed, the trail is active. Therefore Smoking and Using Microwave are independent.
5. P(C = 1 | M = 0) = 0.1256

Question 4

- Code and answers is submitted in zip file.

Question 5

a and b)

Download MobileNetV2 model

```
#<Write code for downloading MobileNetV2>

# Load the MobileNetV2 model pre-trained on ImageNet
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(minSize, minSize, 3))

# Freeze the pre-trained layers
for layer in base_model.layers:
    layer.trainable = False
```

c)

Add custom layers at the end of downloaded model

```
[ ] #<Write code for adding custom layers>
    from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
    from tensorflow.keras.models import Sequential
    # Create a new Sequential model
    model = Sequential()

    # Add the pre-trained base model
    model.add(base_model)


    # Add the rest of the layers
    model.add(GlobalAveragePooling2D())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(10, activation='softmax'))  # 10 classes for CIFAR-10
```

d)

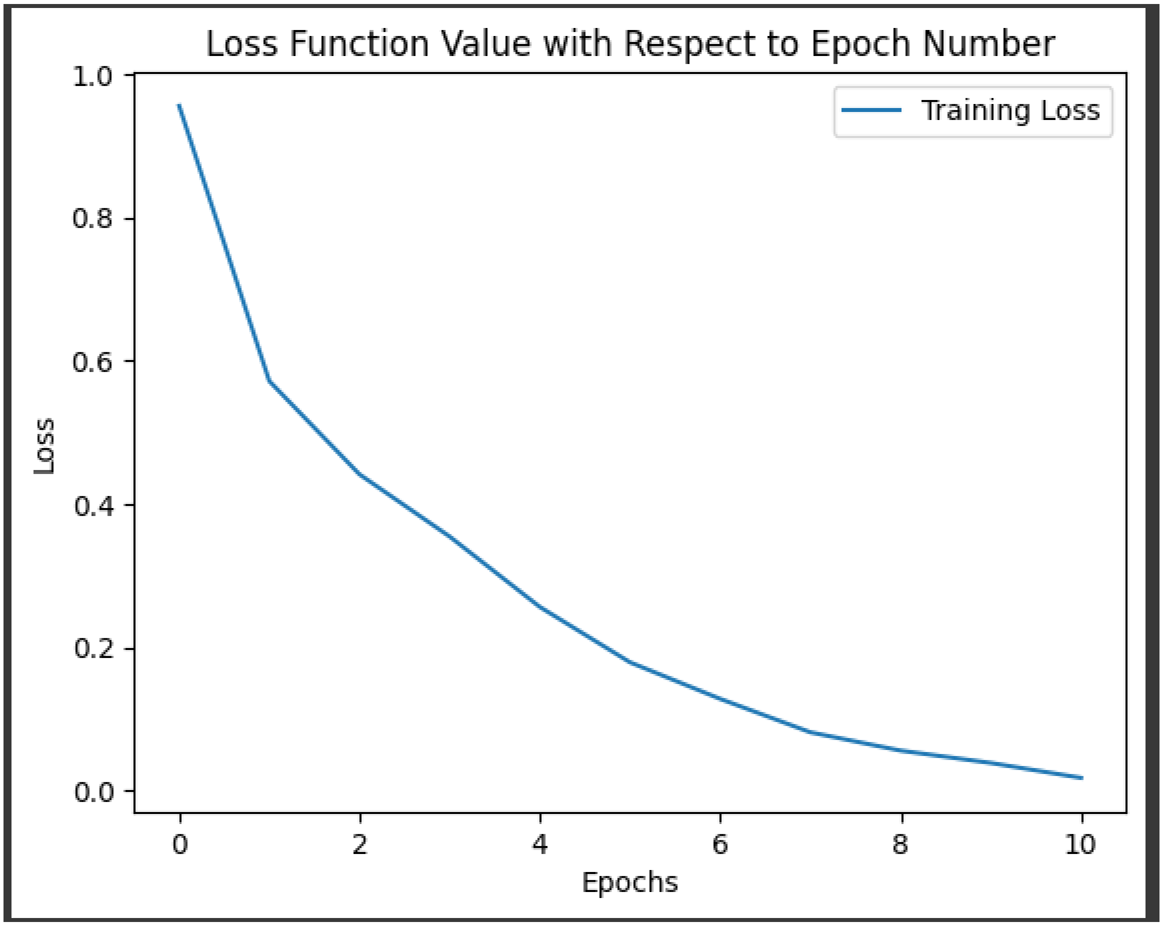
Add loss function, compile and train the model, and check accuracy on test data

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

# Limit the training data and labels to 10,000 samples
limited_train_images = resized_train_images
limited_train_labels = train_labels[:10000]

# Compile the model with sparse categorical crossentropy
model.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Define early stopping criteria
early_stopping = EarlyStopping(monitor='val_loss', patience=8, restore_best_weights=True)

# Train the model with early stopping
history = model.fit(limited_train_images, limited_train_labels, batch_size=64, epochs=50,
                    validation_data=(resized_test_images, test_labels), callbacks=[early_stopping])

# Evaluate the model
test_loss, test_accuracy = model.evaluate(resized_test_images, test_labels, verbose=1)

print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

```
Epoch 1/50
157/157 [==============================] - 20s 78ms/step - loss: 0.9554 - accuracy: 0.6790 - val_loss: 0.7351 - val_accuracy: 0.7427
Epoch 2/50
157/157 [==============================] - 6s 38ms/step - loss: 0.5715 - accuracy: 0.7978 - val_loss: 0.7501 - val_accuracy: 0.7443
Epoch 3/50
157/157 [==============================] - 6s 40ms/step - loss: 0.4413 - accuracy: 0.8450 - val_loss: 0.6973 - val_accuracy: 0.7705
Epoch 4/50
157/157 [==============================] - 8s 53ms/step - loss: 0.3546 - accuracy: 0.8785 - val_loss: 0.7296 - val_accuracy: 0.7607
Epoch 5/50
157/157 [==============================] - 9s 55ms/step - loss: 0.2564 - accuracy: 0.9164 - val_loss: 0.7614 - val_accuracy: 0.7579
Epoch 6/50
157/157 [==============================] - 6s 40ms/step - loss: 0.1791 - accuracy: 0.9472 - val_loss: 0.7866 - val_accuracy: 0.7666
Epoch 7/50
157/157 [==============================] - 6s 38ms/step - loss: 0.1281 - accuracy: 0.9646 - val_loss: 0.8054 - val_accuracy: 0.7637
Epoch 8/50
157/157 [==============================] - 8s 54ms/step - loss: 0.0815 - accuracy: 0.9820 - val_loss: 0.8337 - val_accuracy: 0.7691
Epoch 9/50
157/157 [==============================] - 8s 53ms/step - loss: 0.0560 - accuracy: 0.9904 - val_loss: 0.8981 - val_accuracy: 0.7639
Epoch 10/50
157/157 [==============================] - 9s 56ms/step - loss: 0.0388 - accuracy: 0.9949 - val_loss: 0.9108 - val_accuracy: 0.7711
Epoch 11/50
157/157 [==============================] - 9s 56ms/step - loss: 0.0180 - accuracy: 0.9993 - val_loss: 0.9346 - val_accuracy: 0.7729
313/313 [==============================] - 4s 12ms/step - loss: 0.6973 - accuracy: 0.7705
Test accuracy: 77.05%
```

1) I extended the MobileNetV2 model by adding 1 GlobalAveragePooling2D layer, 1 dense layer with 512 nodes and ReLU activation and 1 dense output layer with 10 nodes and Softmax activation. Initially I added a MaxPool2D layer instead, but it was giving me very low training accuracy.

2)



a. As seen from the screenshot for part (d), I added an EarlyStopping callback to the training process. This avoids overfitting and stops training as soon as val_loss stops improving. I set the patience parameter to 8 after experimenting with a few values and since the model reaches the required accuracy (≥ 70%) in a few epochs, I set it to a relatively low number. I also set restore_best_weights = True so that when training is stopped, the model's weights will be rolled back to the state when it achieved the best performance on the validation loss.

b. For the mini batch size, I experimented with a few values such as 32, 64 and 128 and decided on 64 as it gave me the highest training accuracy and a relatively smooth loss function with respect to epoch number graph.

c. Similarly for learning rate, I experimented with different values ranging from 0.1 to 0.00001. I eventually settled on 0.001 as it gave me the necessary training accuracy when combined with the other set parameters.

3)

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

```python
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

# Limit the training data and labels to 10,000 samples
limited_train_images = resized_train_images
limited_train_labels = train_labels[:10000]

# Compile the model with sparse categorical crossentropy
model.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Define early stopping criteria
early_stopping = EarlyStopping(monitor='val_loss', patience=8, restore_best_weights=True)

# Train the model with early stopping
history = model.fit(limited_train_images, limited_train_labels, batch_size=64, epochs=50,
                    validation_data=(resized_test_images, test_labels), callbacks=[early_stopping])

# Evaluate the model
test_loss, test_accuracy = model.evaluate(resized_test_images, test_labels, verbose=1)

print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

```
Epoch 1/50
157/157 [==============================] - 20s 78ms/step - loss: 0.9554 - accuracy: 0.6790 - val_loss: 0.7351 - val_accuracy: 0.7427
Epoch 2/50
157/157 [==============================] - 6s 38ms/step - loss: 0.5715 - accuracy: 0.7978 - val_loss: 0.7501 - val_accuracy: 0.7443
Epoch 3/50
157/157 [==============================] - 6s 40ms/step - loss: 0.4413 - accuracy: 0.8450 - val_loss: 0.6973 - val_accuracy: 0.7705
Epoch 4/50
157/157 [==============================] - 8s 53ms/step - loss: 0.3546 - accuracy: 0.8785 - val_loss: 0.7296 - val_accuracy: 0.7607
Epoch 5/50
157/157 [==============================] - 9s 55ms/step - loss: 0.2564 - accuracy: 0.9164 - val_loss: 0.7614 - val_accuracy: 0.7579
Epoch 6/50
157/157 [==============================] - 6s 40ms/step - loss: 0.1791 - accuracy: 0.9472 - val_loss: 0.7866 - val_accuracy: 0.7666
Epoch 7/50
157/157 [==============================] - 6s 38ms/step - loss: 0.1281 - accuracy: 0.9646 - val_loss: 0.8054 - val_accuracy: 0.7637
Epoch 8/50
157/157 [==============================] - 8s 54ms/step - loss: 0.0815 - accuracy: 0.9820 - val_loss: 0.8337 - val_accuracy: 0.7691
Epoch 9/50
157/157 [==============================] - 8s 53ms/step - loss: 0.0560 - accuracy: 0.9904 - val_loss: 0.8981 - val_accuracy: 0.7639
Epoch 10/50
157/157 [==============================] - 9s 56ms/step - loss: 0.0388 - accuracy: 0.9949 - val_loss: 0.9108 - val_accuracy: 0.7711
Epoch 11/50
157/157 [==============================] - 9s 56ms/step - loss: 0.0180 - accuracy: 0.9993 - val_loss: 0.9346 - val_accuracy: 0.7729
313/313 [==============================] - 4s 12ms/step - loss: 0.6973 - accuracy: 0.7705
Test accuracy: 77.05%
```

Note: I have added the Q3 code file in the zip folder as well in case the screenshot is not clear.