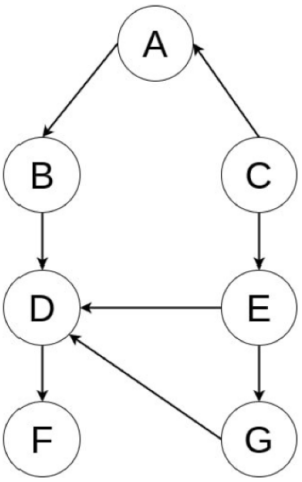
Question 1 [10 points]:

Consider the following Bayesian network:



Write True/False for the following conditional independence statements. Justify clearly your answer by showing active/blocked trails as necessary and appropriate rules for them to be active/blocked. [No coding required for this question. Each sub-question has **2 points**]

1. $A \perp G \mid \{F\}$ = False

Trail 1: A -> C -> E -> G = active

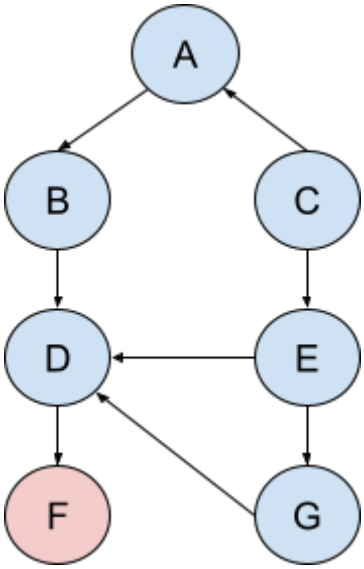
A <- C -> E = active common cause

C -> E -> G = active cascade

Trail 2: A -> B -> D -> G = active

A -> B -> D = active cascade

B -> D <- G = active cascade as F is a descendent of D



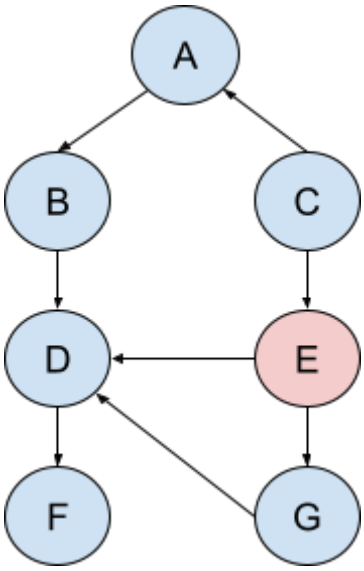
2. $A \perp G \mid \{E\}$ = True

Trail 1: A -> B -> D -> G = inactive

A -> B -> D = active cascade
B -> D <- G = inactive V structure

Trail 2: A -> C -> E -> G = inactive
A <- C -> E = active common cause
C -> E -> G = inactive cascade

Trail 3: A -> C -> E -> D -> G = inactive
C -> E -> D = inactive cascade

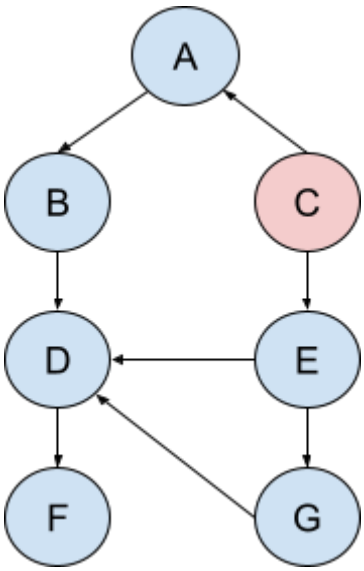


3. $A \perp E \mid \{C\}$ = True

Trail 1: A -> C -> E = inactive
A <- C -> E = inactive common cause

Trail 2: A -> B -> D -> E = inactive
A -> B -> D = active cascade
B -> D <- E = inactive V structure

Trail 3: A -> B -> D -> G -> E = inactive
B -> D <- G = inactive V structure



4. $A \perp E \mid \{C,D\}$ = False

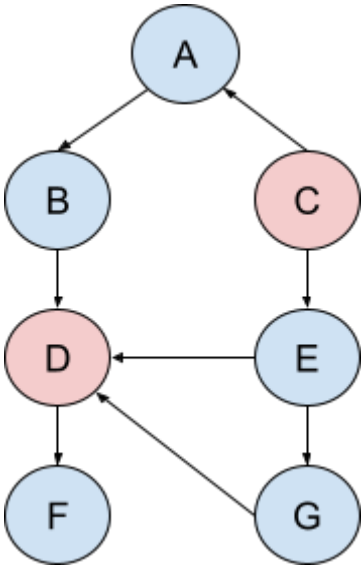
Trail 1: A -> C -> E = inactive

A <- C -> E = inactive common cause

Trail 2: A -> B -> D -> E = active

A -> B -> D = active cascade

B -> D <- E = active V structure



5. $A \perp D \mid \{B,E\}$ = True

Trail 1: A -> B -> D = inactive

A -> B -> D = inactive cascade

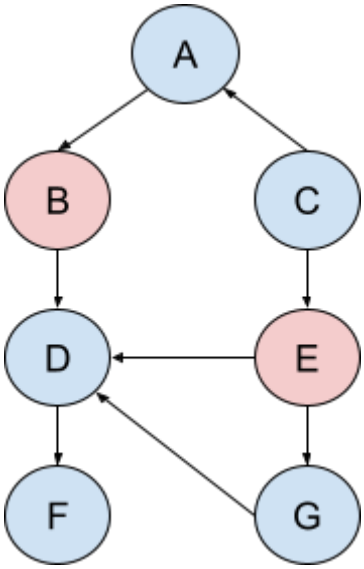
Trail 2: A -> C -> E -> D = inactive

A <- C -> E = active common cause

C -> E -> D = inactive cascade

Trail 3: A -> C -> E -> G -> D = inactive

C -> E -> G = inactive cascade



Question 2 [10 points]

The Ministry of Health purchased COVID-19 tests for distribution to their population. They are produced by three different manufacturers: A, B and C. Each citizen will receive two tests at home and the manufacturer of each test is chosen independently and identically distributed. Therefore, a citizen may receive two tests from the same manufacturer or from two different ones. Each manufacturer is equally likely to be chosen (P=1/3).

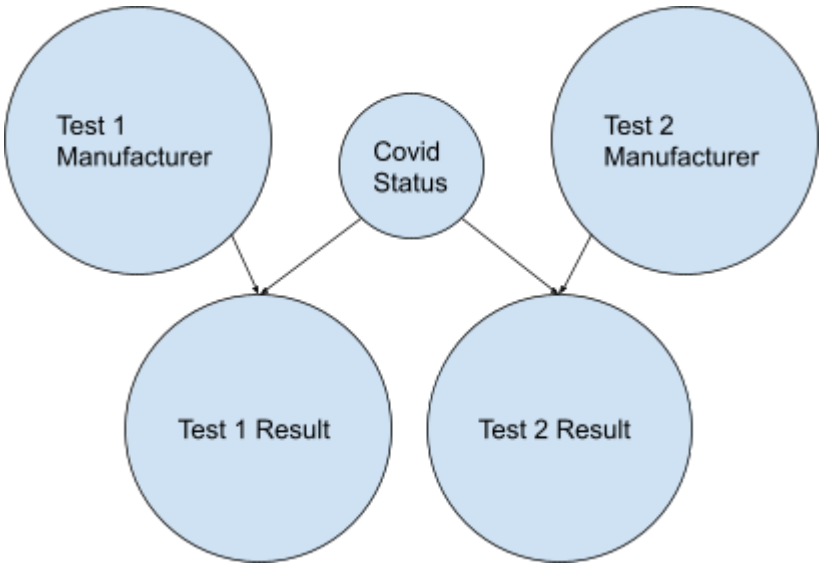
The tests' ability to provide accurate results fluctuates. In other words, they have different chances of producing *true positive* (TP) and *true negative* (TN) results. A True positive occurs when a test is positive and the patient has COVID19. True negatives, on the other hand, are tests in which the result is negative and the patient does not have the illness. The manufacturers describe their tests in the following way:

*A: P(TP) = 0.7 and P(TN) = 0.99
*B: P(TP) = 0.8 and P(TN) = 0.95
*C: P(TP) = 0.9 and P(TN) = 0.91

Assume that before taking the tests, the prior probability of having covid is 0.3.
A person can self-test either using one test kit or both the test kits.

(i) Draw the Bayes net corresponding to this setup. Explain what each random variable of this Bayes represents, and show the domain of each random variable **[3 points]**
Hint: There should be 5 random variables. You can use a table like below to describe random variables and their interpretation:

Variable Name	**Domain (Set of Values)**	**Interpretation** *(intuitive explanation of what the variable represents)*



Variable Name	**Domain (Set of Values)**	**Interpretation**
Covid Status	{Covid, No Covid}	This variable represents whether the individual has Covid or not.
Test 1 Manufacturer	{A, B, C}	This variable represents the manufacturer of the first test.
Test 2 Manufacturer	{A, B, C}	This variable represents the manufacturer of the second test.
Test 1 Result	{Positive, Negative}	This variable represents the results of the first test.
Test 2 Result	{Positive, Negative}	This variable represents the results of the second test.

(ii) Write conditional probabilities (numerical values) associated with each node of this Bayes net. As there are 5 variables, please specify one conditional probability table (CPT) for each variable **[2 points]**

P(Covid Status)	
Covid	No Covid
0.3	0.7

P(Test 1 Manufacturer)		
A	B	C

⅓	⅓	⅓

P(Test 2 Manufacturer)		
A	B	C
⅓	⅓	⅓

P(Test 1 Result | Test 1 Manufacturer, Covid Status)				
	Covid Status			
	Covid		No Covid	
	Test 1 Result		**Test 1 Result**	
Test 1 Manufacturer	Positive	Negative	Positive	Negative
A	0.7	0.3	0.01	0.99
B	0.8	0.2	0.05	0.95
C	0.9	0.1	0.09	0.91

P(Test 2 Result | Test 2 Manufacturer, Covid Status)				
	Covid Status			
	Covid		No Covid	
	Test 2 Result		**Test 2 Result**	
Test 2 Manufacturer	Positive	Negative	Positive	Negative
A	0.7	0.3	0.01	0.99
B	0.8	0.2	0.05	0.95
C	0.9	0.1	0.09	0.91

(iii) Are the results of the two tests dependent or independent given the evidence that the Covid Status is known? Justify your answer. **[1 point]**

The results of the two tests are independent given the evidence that the Covid Status is known. Due to the common cause structure, knowing the Covid Status decouples Test 1 Result and Test 2 Result.

(iv) Assume you took both tests at home. After being tested twice in a matter of minutes, the first test was positive and the second negative. What is the probability that you actually have COVID19? Show your analytical computations (without using pgmpy toolbox) **[4 points]**

P(C, M1, M2, T1, T2) = P(C) P(M1) P(M2) P(T1 | C, M1) P(T2 | C, M2)

Covid, Positive, Negative			
M1 / M2	A	B	C
A	0.3 x ⅓ x ⅓ x 0.7 x 0.3 = 0.007	0.3 x ⅓ x ⅓ x 0.7 x 0.2 = 0.00467	0.3 x ⅓ x ⅓ x 0.7 x 0.1 = 0.00233
B	0.3 x ⅓ x ⅓ x 0.8 x 0.3 = 0.008	0.3 x ⅓ x ⅓ x 0.8 x 0.2 = 0.00533	0.3 x ⅓ x ⅓ x 0.8 x 0.1 = 0.00267
C	0.3 x ⅓ x ⅓ x 0.9 x 0.3 = 0.009	0.3 x ⅓ x ⅓ x 0.9 x 0.2 = 0.006	0.3 x ⅓ x ⅓ x 0.9 x 0.1 = 0.003

No Covid, Positive, Negative			
M1 / M2	A	B	C
A	0.7 x ⅓ x ⅓ x 0.01 x 0.99 = 0.00077	0.7 x ⅓ x ⅓ x 0.01 x 0.95 = 0.00074	0.7 x ⅓ x ⅓ x 0.01 x 0.91 = 0.00071
B	0.7 x ⅓ x ⅓ x 0.05 x 0.99 = 0.00385	0.7 x ⅓ x ⅓ x 0.05 x 0.95 = 0.00369	0.7 x ⅓ x ⅓ x 0.05 x 0.91 = 0.00354
C	0.7 x ⅓ x ⅓ x 0.09 x 0.99 = 0.00693	0.7 x ⅓ x ⅓ x 0.09 x 0.95 = 0.00665	0.7 x ⅓ x ⅓ x 0.09 x 0.91 = 0.00637

P(C = Covid | T1 = Positive, T2 = Negative)
= P(C = Covid, M1, M2, T1 = Positive, T2 = Negative) / P(T1 = Positive, T2 = Negative)
= 0.048 / (0.048 + 0.03325)
= 0.591 (3 s.f.)

Question 3 [10 points]

In this question below, we will construct a small Bayesian network in the pgmpy toolbox. This network models the relationship between *yellow fingers (Y)*, *smoking (S)*, *cancer (C)*, *skin burn (B)*, *radiation (R)*, *solar flares (F)*, *Weakened Immune System (I)*, abusing *alcohol (A),* and *using a cellphone (P)*. In this model, smoking can cause yellow fingers and cancer. Solar flares or using a cellphone can cause radiation. Radiation can cause cancer and skin burn. Alcohol abuse can both cause cancer or weaken the immune system, and a weakened immune system can cause cancer.

Consider "0" represents a variable being a false, "1" represents a variable being true. Each variable in this problem is binary, i.e. can only take two values (0 or 1).

The prior probability of smoking P(S=1) is 0.1. The prior probability of solar flares P(F=1) is 0.001. The prior probability of using a cellphone P(P=1) is 0.999. The prior probability of alcohol abuse P(A) is 0.1.

Conditional probabilities for skin burn (B) are given below:

R	P(B=1|R)
0	0.02
1	0.2

Conditional probabilities for Weakened Immune System (I) are given below:

A	P(I=1|A)
0	0.05
1	0.3

Conditional probabilities for radiation (R) are given below:

F	P	P(R=1|F,P)
0	0	0.01
0	1	0.05
1	0	0.2
1	1	0.3

Conditional probabilities table for cancer C are given as:

S	A	I	R	P(C=1|A,I,R,S)
0	0	0	0	0.1
0	0	0	1	0.5
0	0	1	0	0.2
0	0	1	1	0.6
0	1	0	0	0.3
0	1	0	1	0.6

0	1	1	0	0.4
0	1	1	1	0.8
1	0	0	0	0.2
1	0	0	1	0.6
1	0	1	0	0.3
1	0	1	1	0.7
1	1	0	0	0.4
1	1	0	1	0.7
1	1	1	0	0.5
1	1	1	1	0.9

The conditional probability table for yellow fingers (Y) is given as:

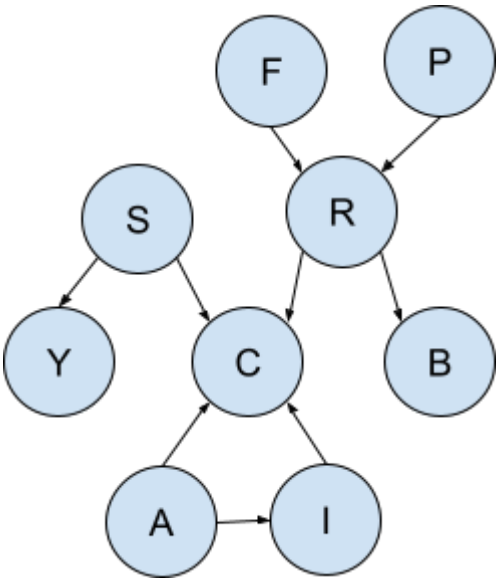
S	P(Y=1|S)
0	0.04
1	0.95

Implement the above Bayes net with the specified conditional probabilities into pgmpy (Install it from http://pgmpy.org/).

- **Show screenshot of your code denoting the implementation in pgmpy and attach as part of your solution pdf.** [**2.5 points**]
  Should show screenshot of the code. Otherwise -1 point.

Answer the following questions. You can use functions implemented in pgmpy to compute the numerical answers as required for some of the below questions as appropriate. Also show snippets of your code used in solution pdf. [Each sub-part has **1.5 points**]

1. Draw the Bayesian network clearly showing the nodes and arrows showing relationship among all the variables (you can use drawing tools in PowerPoint or Keynote)



Create the Bayesian Network

```
# graph edges
model = BayesianModel([('F', 'R'), ('P', 'R'), ('R', 'C'), ('R', 'B'), ('S', 'Y'), ('S', 'C'), ('A', 'C'), ('A', 'I'), ('I', 'C')])

# prior probability [0, 1]
cpd_S = TabularCPD(variable='S', variable_card=2, values=[[0.9], [0.1]])
cpd_F = TabularCPD(variable='F', variable_card=2, values=[[0.999], [0.001]])
cpd_P = TabularCPD(variable='P', variable_card=2, values=[[0.001], [0.999]])
cpd_A = TabularCPD(variable='A', variable_card=2, values=[[0.9], [0.1]])

# conditional probability
cpd_I = TabularCPD(variable='I',variable_card=2, values = [[0.95, 0.7], [0.05, 0.3]],
                                evidence = ['A'],
                                evidence_card=[2])
cpd_R = TabularCPD(variable='R',variable_card=2, values = [[0.99, 0.95, 0.8, 0.7], [0.01, 0.05, 0.2, 0.3]],
                                evidence = ['F', 'P'],
                                evidence_card=[2, 2])
cpd_B = TabularCPD(variable='B',variable_card=2, values = [[0.98, 0.8], [0.02, 0.2]],
                                evidence = ['R'],
                                evidence_card=[2])
cpd_C = TabularCPD(variable='C',variable_card=2, values = [[0.9, 0.5, 0.8, 0.4, 0.7, 0.4, 0.6, 0.2, 0.8, 0.4, 0.7, 0.3, 0.6, 0.3, 0.5, 0.1],
[0.1, 0.5, 0.2, 0.6, 0.3, 0.6, 0.4, 0.8, 0.2, 0.6, 0.3, 0.7, 0.4, 0.7, 0.5, 0.9]],
                                evidence = ['S', 'A', 'I', 'R'],
                                evidence_card=[2, 2, 2, 2])
cpd_Y = TabularCPD(variable='Y',variable_card=2, values = [[0.96, 0.05], [0.04, 0.95]],
                                evidence = ['S'],
                                evidence_card=[2])
model.add_cpds(cpd_S, cpd_F, cpd_P, cpd_A, cpd_I, cpd_R, cpd_B, cpd_C, cpd_Y)
```

Inference

```
from pgmpy.inference import VariableElimination

infer = VariableElimination(model)

phi_query = infer.query(['R'], evidence={'C':1}, joint = False)
factor = phi_query['R']
print('Probability of radiation given cancer')
print(factor)

phi_query = infer.query(['C'], evidence={'B':1, 'Y':1, 'A':1}, joint = False)
factor = phi_query['C']
print('Probability of cancer given skin burn, yellow fingers, abuses alcohol')
print(factor)

phi_query = infer.query(['C'], evidence={'A':0, 'P':0}, joint = False)
factor = phi_query['C']
print('Probability of cancer given no alcohol abuse, no cellphone')
print(factor)
```

[16] ✓ 0.3s Python

2. What is the probability of radiation given Cancer = 1 (show values for R=0, 1)?

```
Probability of radiation given cancer
+------+----------+
| R    |   phi(R) |
+======+==========+
| R(0) |   0.8306 |
+------+----------+
| R(1) |   0.1694 |
+------+----------+
```

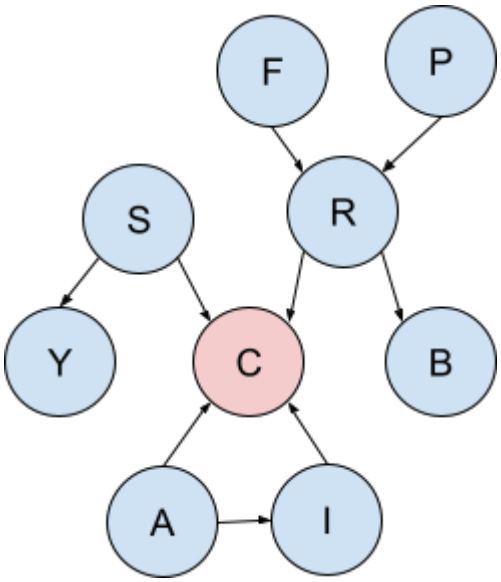
3. What is the probability of cancer given the patient has skin burn, yellow fingers and abuses alcohol (show values for C=0, 1)

```
Probability of cancer given skin burn, yellow fingers, abuses alcohol
+------+----------+
| C    |   phi(C) |
+======+==========+
| C(0) |   0.4834 |
+------+----------+
| C(1) |   0.5166 |
+------+----------+
```

4. Are Smoking and skin burn independent given that cancer is present? Justify your answer.

Trail 1: S -> C -> R -> B = active
S -> C -> R = active V structure
C -> R -> B = active common cause

Hence, smoking and skin burn are independent given that cancer is present.



5. What is the probability of cancer (=1) if you never abused alcohol or used a cellphone?

```
Probability of cancer given no alcohol abuse, no cellphone
+------+----------+
| C    |   phi(C) |
+======+==========+
| C(0) |   0.8809 |
+------+----------+
| C(1) |   0.1191 |
+------+----------+
```

Question 4 [10 points]

In this question, we will use an ANN classifier to classify the handwritten digits (from the MNIST dataset). In a slight twist, to check the robustness of ANN-based methods, we corrupt MNIST images via adding some noise. We will use the "**mnist_corrupted/zigzag**" dataset available via tfds (https://www.tensorflow.org/datasets/catalog/mnist_corrupted).

You can build your solution on top of the python notebook covered in class to classify the standard MNIST dataset. The solution should perform the following tasks:

a) Load the **mnist_corrupted/zigzag** dataset via tfds. Load both training and testing datasets separately. **[2 points]** (refer to this page to learn how to load data from tfds: https://www.tensorflow.org/datasets/overview#as_numpy_tfdsas_numpy)

▾ (a) Load the dataset `mnist_corrupted/zigzag` from `tensorflow_datasets`. [**2 points**]

```
[5] import tensorflow_datasets as tfds

    ## write your code here
    dataset_name = "mnist_corrupted/zigzag"
    train_images, train_labels = tfds.as_numpy(tfds.load(
        dataset_name,
        split='train',
        batch_size=-1,
        as_supervised=True,
    ))

    test_images, test_labels   = tfds.as_numpy(tfds.load(
        dataset_name,
        split='test',
        batch_size=-1,
        as_supervised=True,
    ))
```

```
# Test size of different loaded numpy arrays
print('Image size:', train_images[0].shape)
print('Training data size:',train_images.shape)
print('Testing data size:', test_images.shape)
```

```
Image size: (28, 28, 1)
Training data size: (60000, 28, 28, 1)
Testing data size: (10000, 28, 28, 1)
```

b) Create an ANN with 1 input layer and **at least** one hidden layer, with at least 2 nodes per layer. You can use relu or any other activation function for hidden layers. You are free to create additional hidden layers or increase the number of nodes to maximize the final accuracy. You are encouraged to systematically explore the different hyper-parameter configurations. Your objective is to get an overall accuracy of 90% or more on the testing dataset. **[2 points]**

(b) Build a dense ANN with at least one hidden layer with at least two nodes. You should try different hyper parameter configurations to get the best performing ANN (atleast 90% accuracy). [**2 points**]

```
[84] model = tf.keras.Sequential()
     outputs = 10
     ## write your code here to build your dense ANN. Input layer is created below
     model.add(layers.Flatten(input_shape=(train_images[0].shape)))
     model.add(layers.Dense(128, activation=tf.nn.relu))      # hidden layer
     model.add(layers.Dense(128, activation=tf.nn.relu))      # hidden layer
     model.add(layers.Dense(128, activation=tf.nn.relu))      # hidden layer
     model.add(layers.Dense(outputs, activation=tf.nn.softmax))   # output layer
```

c) Create 1 output layer. What is the size of output layer? What should be the activation function for the output layer? **[1 points]**

(c) Answer the following questions: [**1 point**]

What is the size of your output layer? [10]

What is the activation function of your output layer? [softmax]

d) Compile and train the neural network with the appropriate loss function (what should be the loss function type?) **[2 points]**

(d) Compile and train your model [**2 points**]

What is the loss function you use? [sparse categorical crossentropy]

```
[85] ### write your code here to compile model
     # Compile the model with appropriate Loss function. metrics is something you can monitor (but model does not optimize metric)
     model.compile(optimizer="Adam",
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])
```

```
[86] ### write your code here to train your model
     # Run the stochastic gradient descent for specified epochs
     epochs = 10
     history = model.fit(train_images, train_labels, epochs=epochs)
```

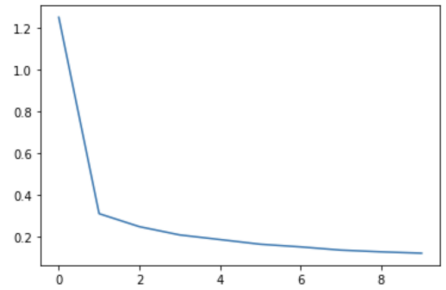
```
Epoch 1/10
1875/1875 [==============================] - 9s 4ms/step - loss: 1.2513 - accuracy: 0.8187
Epoch 2/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3087 - accuracy: 0.9112
Epoch 3/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2462 - accuracy: 0.9272
Epoch 4/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2066 - accuracy: 0.9391
Epoch 5/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1843 - accuracy: 0.9456
Epoch 6/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1619 - accuracy: 0.9520
Epoch 7/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1491 - accuracy: 0.9560
Epoch 8/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1337 - accuracy: 0.9609
Epoch 9/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1254 - accuracy: 0.9634
Epoch 10/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1189 - accuracy: 0.9657
```

e) Plot the training loss (as given by the Keras API) on the y-axis against the epoch number **[1 point]** (refer to this page to see how to extract the history: https://www.tensorflow.org/guide/keras/train_and_evaluate)

▾ (e) Plot the training loss across the different epochs [**1 point**]

```
[87] plt.plot(history.history['loss'])
```

```
[<matplotlib.lines.Line2D at 0x7f0f19061bb0>]
```



f) What is the final accuracy for different classes (the proportion of correctly classified instances of each class) and overall accuracy on the testing data? **[1 point]**

g) For total accuracy (on testing data):
   1. >=90% **[0.5 point],**
   2. >=95% **[additional 0.5 points]**

▾ (f) What is the overall accuracy and per-class accuracy on test dataset? [**1 point**]

```
[88] #### write your code to report overall accuracy on test set
     test_loss, test_acc = model.evaluate(test_images, test_labels)
     print('Test accuracy:', test_acc)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.2172 - accuracy: 0.9526
Test accuracy: 0.9526000022888184
```

What is the overall accuracy? [0.95260]

```
[106] ### write your code to report per-class accuracy
      ### Use confusion matrix from sklearn.

      from sklearn.metrics import confusion_matrix
      import numpy as np

      test_predictions = model.predict(test_images)
      cm = confusion_matrix(test_labels, np.argmax(test_predictions,axis=1))
      per_class_accuracy = np.diag(cm) / cm.sum(axis=1)
      print(per_class_accuracy)
      print(np.mean(per_class_accuracy))
```

```
313/313 [==============================] - 1s 2ms/step
[0.97959184 0.99207048 0.92926357 0.95049505 0.95315682 0.95852018
 0.96033403 0.942607   0.95379877 0.90386521]
0.9523702953063417
```

Question 5 [10 points]

In this question, we will learn how to use transfer learning in the context of CNNs. More hints to solve this question are available at: https://www.tensorflow.org/tutorials/images/transfer_learning . In particular,

- You will first create and train a CNN using the MNIST dataset. The CNN we use is the LeNet-5 which was one of the first successful CNN proposed by Yann LeCun (https://en.wikipedia.org/wiki/LeNet)
- You will then use this trained CNN, and adapt it to the **binary_alpha_digits** dataset (https://www.tensorflow.org/datasets/catalog/binary_alpha_digits).

We have provided you a code skeleton (Q5_vision_lenet.ipynb) to help you finish this problem. You are free to change this file as required.

Please perform the following tasks in your solution:

a) Create the LeNet-5 CNN architecture using Keras API (see code skeleton for the number and types of layers to create). Train the model on the MNIST dataset. **[3 points]**

(a) Create and train Lenet-5 Using Keras API on MNIST dataset **[3 points]**

```
'''
Create a NN with 1 input layer
1 conv2D layer, 6 filters, 5x5 filter size, stride = (1, 1), activation tanh, use padding='same' argument (check it on https://keras.io/api/layers/convolu
1 AveragePooling2D layer (use default arguments in https://www.tensorflow.org/api_docs/python/tf/keras/layers/AveragePooling2D )
1 conv2D layer, 16 Filters, 5x5 filter size, stride = (1, 1), activation tanh, padding='valid'
1 AveragePooling2D layer (Default arguements)
1 conv2D layer, 120 filters, 5x5 filter size, stride = (1, 1), activation tanh, padding='valid'
Flatten layer
1 Dense layer, 84 units, tanh activation
1 output layer
'''
input_shape = train_images[0].shape
model = tf.keras.Sequential()
model.add(layers.Conv2D(6, kernel_size=(5, 5), strides=(1, 1),
                activation='tanh',
                padding='same',
                input_shape=input_shape))
model.add(layers.AveragePooling2D())
model.add(layers.Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='tanh', padding='valid'))
model.add(layers.AveragePooling2D())
model.add(layers.Conv2D(120, kernel_size=(5, 5), strides=(1, 1), activation='tanh', padding='valid'))
model.add(layers.Flatten())
model.add(layers.Dense(84, activation='tanh'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 28, 28, 6)         156

 average_pooling2d_12 (Avera  (None, 14, 14, 6)        0
 gePooling2D)

 conv2d_19 (Conv2D)          (None, 10, 10, 16)        2416

 average_pooling2d_13 (Avera  (None, 5, 5, 16)         0
 gePooling2D)

 conv2d_20 (Conv2D)          (None, 1, 1, 120)         48120

 flatten_6 (Flatten)         (None, 120)               0

 dense_35 (Dense)            (None, 84)                10164

 dense_36 (Dense)            (None, 10)                850

=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

```
[157] # Compile the model with appropriate Loss function
      model.compile(optimizer=tf.optimizers.Adam(learning_rate=0.001),
                    loss='sparse_categorical_crossentropy',
                    metrics=['accuracy'])
```

```
      # Train the model on MNIST dataset
      epochs = 5
      batch_size = 32
      model.fit(train_images, train_labels, batch_size=batch_size, epochs=epochs)
```

```
Epoch 1/5
1875/1875 [==============================] - 9s 4ms/step - loss: 0.2299 - accuracy: 0.9309
Epoch 2/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0893 - accuracy: 0.9722
Epoch 3/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0594 - accuracy: 0.9815
Epoch 4/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0463 - accuracy: 0.9854
Epoch 5/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0371 - accuracy: 0.9883
<keras.callbacks.History at 0x7faacf330a00>
```

b) What is the accuracy of your trained LeNet-5 model on the MNIST training dataset? Try to get an accuracy above 90%. **[0.5 points]**

▾ (b) Check Accuracy on Test Data **[0.5 point]**

```
[159] test_loss, test_acc = model.evaluate(test_images, test_labels)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.0518 - accuracy: 0.9838
```

```
[160] # Try to get 90% or more accuracy
      print('Test accuracy:', test_acc)
```

```
Test accuracy: 0.9837999939918518
```

c) Download the **binary_alpha_digits** dataset using tfds, and split the dataset into 20% testing data and 80% training data. **[1 point]**

(c) Download binary_alpha_digits dataset using tfds, split dataset **[1 point]**

```
[163] ## write your code here
      dataset_name = "binary_alpha_digits"

      ds_images, ds_labels = tfds.as_numpy(tfds.load(
          dataset_name,
          split='train',
          batch_size=-1,
          as_supervised=True,
      ))
```

```
[164] ## Split dataset into 20% testing and 80% training

      test_size = 0.2   # fraction of test data

      from sklearn.model_selection import train_test_split

      train_images, test_images, train_labels, test_labels = train_test_split(ds_images, ds_labels, test_size=test_size)

      # Check training, testing data size
      print(train_images.shape)
      print(test_images.shape)
```

Notebook stored in Google Drive

```
(1123, 20, 16, 1)
(281, 20, 16, 1)
```

d) As the dimension of images in the binary_alpha_digits are different from the image size in MNIST dataset, upscale images in binary_alpha_digits to match the image size in MNIST dataset using OpenCV. This is required as we would like to use the LeNet trained using the MNIST dataset for binary_alpha_digits. **[2 points]**

(d) Upscale training, testing data to MNIST image size (28, 28, 1) **[2 points]**

```
[166] #'Upscale Data'
      newSize = 28

      # create a numpy array for storing upscaled training images
      train_upscale = np.zeros((train_images.shape[0], newSize, newSize, 1))
      #<<Write code for upscaling training data>> Look at function cv2.resize in opencv https://pythonexamples.org/python-opencv-cv2-resize-image/
      import cv2
      for i in range(train_images.shape[0]):
          train_upscale[i,:,:,0] = cv2.resize(train_images[i,:,:,0], (newSize, newSize), interpolation=cv2.INTER_CUBIC)

      print(train_upscale.shape)

      # create a numpy array for storing upscaled testing images
      test_upscale = np.zeros((test_images.shape[0], newSize, newSize, 1))
      #<<Write code for upscaling testing data>>
      for i in range(test_images.shape[0]):
          test_upscale[i,:,:,0] = cv2.resize(test_images[i,:,:,0], (newSize, newSize), interpolation=cv2.INTER_CUBIC)
      print(test_upscale.shape)
```

```
(1123, 28, 28, 1)
(281, 28, 28, 1)
```

e) Remove the final output layer of LeNet you have trained on MNIST (to do this, please check the flag “include_top” in Keras and the tensorflow link for transfer learning noted earlier) **[0.5 point]**

(e) Transfer learning-- Remove Last layer from your trained LeNet **[0.5 points]**

```
[168] ## You can decide whether to train the whole network again or fix layer weights from the MNIST-trained network
      ## Check link: https://keras.io/getting_started/faq/#how-can-i-freeze-keras-layers
      #<<Write code for either freezing or training all layers from scratch>>
      # model.trainable = False

      ## Code for removing last layer
      model.layers.pop()
```

```
<keras.layers.core.dense.Dense at 0x7faae0ef6100>
```

f) After removing the final output layer, extend your trained LeNet model by adding at least one hidden layer (dense, convolution, max pooling or any other type of layer). Also attach one final output layer. In this part, you are free to explore and decide how many hidden layers to add, their type, the number of nodes in each layer and the activation function yourself. Keep in mind, the *output layer* must have the appropriate number of nodes and activation function that matches the given task. **[1.5 points]**

(f) Transfer learning-- Add new layers to LeNet **[1.5 points]**

```
[169] ## Add one or more hidden layer
      ## Add output layer
      # 128 relu tanh relu tanh - 19%
      model.add(layers.Dense(128, activation='relu'))
      model.add(layers.Dense(128, activation='tanh'))
      model.add(layers.Dense(128, activation='relu'))
      model.add(layers.Dense(128, activation='tanh'))
      model.add(layers.Dense(36, activation='softmax'))
```

```
[170] # Compile the model with appropriate Loss function
      model.compile(optimizer=tf.optimizers.Adam(),
                    loss='sparse_categorical_crossentropy',
                    metrics=['accuracy'])
```

g) Train the model and show accuracy on the testing dataset (of **binary_alpha_digits)**. You can either fix all the weights of your MNIST-trained LeNet model and train only the layers you have added, or train the whole network again. Choose the setting that gives you higher accuracy given the computational resources. Check link https://keras.io/getting-started/faq/#how-can-i-freeze-keras-layers.
Try to **achieve** a testing data accuracy of **50% or more** (you can report the **best** over multiple **run**s). Please make sure that in your submitted jupypter notebook, **logs** show your best run. **Note:** some variation between runs is expected, with *the true accuracy*

being somewhere in between. You are **not** required to *reliably* get 50 percent accuracy over *all runs*, but try to demonstrate from the log files that one run achieved 50 percent. **[1.5 points]**

(g) Train the model and show accuracy on the testing dataset (test_upscale) **[1.5 point]**

```
## Your code here
epochs = 50
batch_size = 32
model.fit(train_upscale, train_labels, batch_size=batch_size, epochs=epochs)

test_loss, test_acc = model.evaluate(test_upscale, test_labels)
print('Test accuracy:', test_acc)
```

```
Epoch 37/50
36/36 [==============================] - 0s 5ms/step - loss: 0.2232 - accuracy: 0.9190
Epoch 38/50
36/36 [==============================] - 0s 5ms/step - loss: 0.2133 - accuracy: 0.9190
Epoch 39/50
36/36 [==============================] - 0s 5ms/step - loss: 0.1955 - accuracy: 0.9323
Epoch 40/50
36/36 [==============================] - 0s 5ms/step - loss: 0.1651 - accuracy: 0.9421
Epoch 41/50
36/36 [==============================] - 0s 5ms/step - loss: 0.1370 - accuracy: 0.9466
Epoch 42/50
36/36 [==============================] - 0s 5ms/step - loss: 0.1135 - accuracy: 0.9617
Epoch 43/50
36/36 [==============================] - 0s 5ms/step - loss: 0.1061 - accuracy: 0.9617
Epoch 44/50
36/36 [==============================] - 0s 5ms/step - loss: 0.0981 - accuracy: 0.9671
Epoch 45/50
36/36 [==============================] - 0s 5ms/step - loss: 0.1005 - accuracy: 0.9519
Epoch 46/50
36/36 [==============================] - 0s 5ms/step - loss: 0.1467 - accuracy: 0.9466
Epoch 47/50
36/36 [==============================] - 0s 5ms/step - loss: 0.1284 - accuracy: 0.9457
Epoch 48/50
36/36 [==============================] - 0s 6ms/step - loss: 0.1136 - accuracy: 0.9439
Epoch 49/50
36/36 [==============================] - 0s 5ms/step - loss: 0.4401 - accuracy: 0.8629
Epoch 50/50
36/36 [==============================] - 0s 6ms/step - loss: 0.3486 - accuracy: 0.8744
9/9 [==============================] - 0s 4ms/step - loss: 2.3659 - accuracy: 0.5053
Test accuracy: 0.5053380727767944
```